AMIGA-Cluster

Version 2.0

Integriertes Software-Entwicklungs-System für alle
Amiga-Modelle ab Kickstart 2.0

Software:	Ulrich Sigmund, Thomas Pfrengle Wolfgang Baron, Thomas Rocks Stefan Herr, Norbert Scherer Peter Vohmann
Dokumentation:	Thomas Pfrengle, Jürgen Braun Stefan Schade, Kai Bolay Tom Richter
Layout:	Jürgen Braun, Thomas Pfrengle Stefan Schade, Peter Vohmann
Support:	Norbert Scherer, Holger Hoffstätte, Alexander Golde, Kahwe Kazemi
Generalvertrieb:	DTM-Computersysteme Dreiherrenstein 6a 6200 Wiesbaden-Auringen 06127/4064

Teil I

Bedienung der Entwicklungsumgebung

Einführung in Cluster

Cluster für Fortgeschrittene

Systemprogrammierung

Hardwarenahe Programmierung

Lizenzbedingungen **Cluster** ist kein urheberrechtsfreies Softwarepacket. Die Benutzung von **Cluster** ohne eine gültige Lizenz ist hiermit untersagt. Kein Teil dieses Handbuches sowie des dazugehörigen Programms darf in irgendeiner Form (Druck, Fotokopie oder einem sonstigen Verfahren) ohne schriftliche Genehmigung reproduziert oder vervielfältigt werden. Sämtliche Teile des Pakets sind nicht kopiergeschützt, um die Erstellung von Sicherheitskopien der Programmdisketten oder eine Installation auf Festplatte zu ermöglichen. Kopien dürfen lediglich zu Archivierungszwecken oder zur Installation auf einem einzigen Amiga gemacht werden, jegliches weitere kopieren, sowie die Weitergabe von Kopien ist verboten. Es wird hiermit darauf hingewiesen, daß alle Pakete eine laufende Registriernummer enthalten, und daß StoneWare, falls StoneWare eine nicht rechtmäßige Kopie von **Cluster** erhält, gegen den unter dieser Nummer registrierten Käufer rechtliche Schritte anstreben wird.

Garantie StoneWare garantiert Ihnen den einwandfreien Zustand von Disketten und Handbuch. StoneWare kann leider keine Haftung für die Fehlerfreiheit von Programmen und Handbuch geben. Weiterhin garantiert StoneWare in keiner Weise die Tauglichkeit des Programms für einen bestimmten Zweck. StoneWare haftet nicht für Schäden, die direkt oder indirekt durch die Beutzung von **Cluster** entstehen. StoneWare behält sich vor, Teile des Handbuches oder des Programms zu verändern, ohne jeden Anwender persönlich davon in Kenntnis zu setzen. Sollten Fehler entdeckt werden, so ist StoneWare bestrebt, diese möglichst schnell zu korrigieren.

Warenzeichen Amiga, AmigaDOS, Kickstart, Workbench und Intuition sind eingetragene Warenzeichen von Commodore-Amiga Inc.

Dieses Handbuch wurde komplett auf einem Amiga mit dem HiTex-Editor 3.0 geschrieben und mit Amiga-TEX gesetzt.

Vorwort

Nach nun fast zwei Jahren, in denen der Cluster-Compiler auf dem Markt ist, war es an der Zeit, ein komplett neues Handbuch, sowie eine stark überarbeitete Version der Software herauszugeben. Außerdem sind wir nun in der erfreulichen Lage über einen kompetenten Vertrieb an unserer Seite zu verfügen, so daß sich **Cluster** in der nächsten Zeit noch stärker verbreiten wird.

Was bietet nun Version 2.0? Es hat sich einiges getan, seit Version 1.0 im Herbst 1990 das Licht der Kölner Messe erblikkte. Dabei konnten wir vor allem auf die Erfahrungen und Anregungen unserer bisherigen Kunden zurückgreifen. Ein Ergebnis dieser Erfahrungen halten Sie im Moment in den Händen: Ein komplett überarbeitetes Handbuch, wobei besonders auf leichte Verständlichkeit für Anfänger großen Wert gelegt wurde. Da wir immer wieder den Hinweis erhielten, daß unser bisheriges Handbuch für Neulinge recht problematisch sei, war dies auch nötig.

Aber auch die Software hat grundlegende Veränderungen erfahren, so wurde der Compiler überarbeitet, so daß er Code für alle 680XX Prozessoren generieren (bis hin zum 68040), sowie Libraries und Devices erzeugen kann. Die Betriebsystemmodule wurden an OS 2.0 angepaßt. Als letzte Neuerung kam nun auch ein neuer Editor hinzu, da der normale, nicht User-Interface-Styleguide konform programmiert, immer wieder Kritikpunkt von Clustergegnern war. Selbstverständlich wird der normale Editor, in einer überarbeiteten Version, weiterhin ausgeliefert, da es viele Clusterprogrammierer gibt, die diesen ebenso wie ich schätzen. Im Zuge dieser Neuerung werden nun auch Cli-Versionen von Compiler, Linker, Loader und Make ausgeliefert, die über AREXX von jedem Editor aus ansprechbar sind.

Jedoch nicht nur die Bedienung der einzelnen Programme hat sich verbessert, nein auch die Sprache selbst wurde nochmals überarbeitet und erweitert. So verfügt sie nun über objektorientierte Elemente (Objekte, Methoden einschließlich Mehrfachvererbung und dynamisches Binden), benutzerdefinierte Exceptions (Ausnahmebehandlung, welche selbst unter C++ noch im Versuchsstadium ist), ein mächtiges Resourcehandlingsystem und vieles mehr. Lassen Sie sich also überraschen.

Aufgrund dieser Neuerungen empfehle ich jedem auch schon erfahrenen Clusterprogrammierer, die Einführung in **Cluster** komplett zu lesen, auch wenn einiges trivial erscheinen sollte.

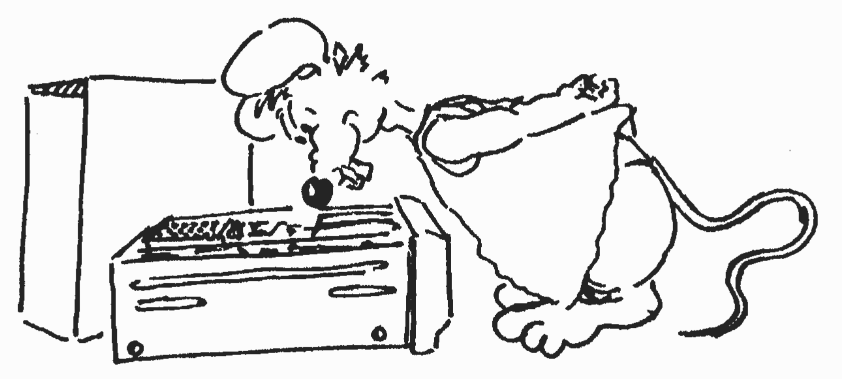
An dieser Stelle möchte ich mich nun bei all jenen bedanken, die uns bei der Fertigstellung dieser neuen Version unterstützt haben, sei es durch tatkräftige Hilfe bei Software und Handbuch oder auch nur durch Vorschläge und konstruktiver Kritik.

Thomas Pfrengle

Inhaltsverzeichnis

Kapitel 1

Einführung



Heute spricht alles von EDV, Computersteuerung, Computerüberwachung, vom Segen der Computer und mindestens ebenso oft von den Gefahren, die von Computern ausgehen, vom Datenschutz, vom gläsernen Menschen und dem Überhandnehmen der Informationsflut. In immer neue Bereiche dringt der Computer heute vor, doch den wenigsten Menschen ist eigentlich bewußt, was ein Computer genau ist, und wie er arbeitet. Wir sind uns sicher, daß auch einige von Ihnen bisher ihren Computer benutzt, vielleicht sogar in BASIC programmiert haben, ohne sich über die inneren Abläufe Ihres Computers zu kümmern. Daher möchten wir Ihnen hier einen kleinen Überblick über die Arbeitsweise Ihres Computers geben, um Ihnen möglicherweise das Verstehen mancher Vorgehensweisen beim Programmieren zu erleichtern.

Früher in der Anfangsphase der Entwicklung des Computers hätte man einfach sagen können, ein Computer sei eine Rechenmaschine. Heute jedoch scheint dies nur noch in den wenigsten Fällen zuzutreffen. Computer haben scheinbar nur noch selten etwas mit Zahlen zu tun. Vielmehr schreibt man auf ihnen Texte oder läßt den Computer Texte korrigieren. Sie steuern Maschinen in allen Bereichen der Technik und überwachen Produktionsabläufe. Ja sogar ganze Filmsequencen wie z. B. in „Terminator 2“ werden heute mit Computern erzeugt. Wenn man jedoch genauer hinsieht, stellt man fest, daß der Computer auch hier nur mit Zahlen jongliert, allerdings auf besondere Weise.

Eine Maschine zu konstruieren, die dem Menschen die Rechenarbeit abnimmt und welche auch noch schneller als er ist, war der Traum der Menschen, seit sie Zahlen verwenden. Blaise Pascal entwarf im 17. Jahrhundert eine erste mechanische Rechenmaschine, die alle Grundrechenarten beherrschte. Leider wurde sie nie gebaut, und erst in heutiger Zeit weiß man, daß sie tatsächlich funktioniert hätte. Allerdings hätte man sie aufgrund der großen

Reibung der vielen hundert Rädchen, Achsen und Steuerteile nicht betreiben können.

Erst am Anfang unseres Jahrhunderts wurde die erste Rechenmaschine auf elektrischer Basis entwickelt, allerdings nahm sie noch den Platz einer kleinen Turnhalle ein. Interessant ist allerdings, daß aus dieser Zeit der Röhren und Relais sich der Begriff „Bug“ gehalten hat, der daher kommt, daß immer wieder Käfer (englisch Bugs) sich in den Computer verirrten und Kurzschlüsse verursachten. So spricht man auch heute noch bei Programmfehlern von Bugs oder von Debugging bei der Fehlersuche.

Erst in den 60er Jahren wurden Computer so klein, daß man sie problemlos in ein Zimmer stellen und transportieren konnte. Seither hat sich allerdings bis auf die Geschwindigkeit, der Speichergröße und der Rechnergröße nicht mehr viel geändert.

Wie aber abeitet nun ein solcher Computer?

Das Herz des Computers ist der Prozessor. Er ist das ausführende Organ, nur er kann rechnen und weiß, was er wann zu tun hat. Im Prinzip ist die Aussage, daß er weiß, was er zu tun hat, nicht ganz richtig, da der Computer immer nur einem Programm folgt. Ein solches Programm ist nun eine Folge von Befehlen und wie könnte es anders sein, da ein Prozessor nur Zahlen kennt, sind auch die Befehle Zahlen, wobei jede eine bestimmte Bedeutung hat. Diese Befehle stehen nun nacheinander im Speicher. Dieser besteht aus Millionen kleiner Einheiten, die die Zustände 0 oder 1 darstellen können. Diese Einheiten nennt man auch Bits. Der Übersicht wegen werden jeweils acht solcher Bits zu einem Byte zusammengefaßt.

Die einzelnen Befehle können nun verschiedenes bewirken, z. B. zwei Zahlen addieren oder das Programm an einer anderen Stelle im Speicher fortsetzen. Es ist jemandem kaum zuzumuten, sich all diese Zahlen zu merken und mit ihnen zu programmieren. Anfangs wurde dies mit Lochkarten so gemacht: Man ordnete jeder Zahl einen kurzen Namen zu, welcher dann von einem Programm in den Maschinen-Code übersetzt wurde. Diese sehr primitive Programmiersprache nennt man Assembler und wird auch heute noch verwendet. Jedoch haben alle Assembler-Befehle gemeinsam, daß ein einzelner Befehle nicht allzuviel bewirkt. Man benötigt also sehr viel mehr Befehle in Assembler, als in einer Hochsprache wie **Cluster**. Außerdem bietet Assembler neben dem Vorteil der hohen Geschwindigkeit, da der Prozessor voll ausgenutzt werden kann, noch einige Nachteile: So muß sich der Programmierer selbst darum kümmern, wo im Speicher seine Daten liegen. Es wird nicht überprüft, ob man z. B. versucht einen Buchstaben zu quadrieren. Daraus folgt natürlich eine viel höhere Fehleranfälligkeit bei der Assemblerprogrammierung, als dies bei einer Hochsprache der Fall ist.

Eine Hochsprache nun benutzt normalerweise kurze Anweisungen im Klartext, wie beispielsweise `IF` (Falls), `REPEAT` (wiederhole), so daß ein solches Programm relativ leicht lesbar ist. Außerdem bewirkt manch ein Hochsprachenbefehl meist mehr als ein vergleichbarer Assemblerbefehl.

Nun gibt es generell zwei Verfahren wie Hochsprachen arbeiten, entweder als Interpreter wie z. B. BASIC oder als Compiler wie z. B. C, Modula oder **Cluster**.

Ein Interpreter nimmt sich jeden Befehl einzeln vor, übersetzt ihn in die entsprechende Maschinencodefolge und führt ihn aus. Dies geschieht bei jedem Programmstart von neuem ebenso bei Programmteilen, die mehrfach aufgerufen werden — jedesmal wird neu übersetzt. Dies hat natürlich zwei große Nachteile: Einmal kostet der Übersetzungsvorgang viel Zeit, zum anderen muß bei jedem Programmstart auch der Interpreter im Speicher sein, so daß 1 MB rasch knapp werden kann.

Ein Compiler hingegen übersetzt einmal das Programm und speichert es dann in übersetzter Form ab, so daß das Programm danach alleine lauffähig ist. Ein Nachteil ist offensichtlich, vor dem Start muß erst übersetzt werden. Allerdings glauben wir, daß dieser kleine Nachteil durch unseren schnellen Compiler und den sehr viel schnelleren Programmen mehr als ausgeglichen wird.

Bei Sprachen wie z. B. **Cluster** oder Modula, die eine Programmteilung in einzelne Module zulassen, müssen diese einzelnen Teile zu einem fertigen Programm zusammengefaßt werden. Diese Aufgabe übernimmt der Linker, er verbindet die bereits compilierten Module zum ausführbaren Programm.

An dieser Stelle soll nun die Theorie enden. Einen Computer mit all seinen Einzelheiten zu beschreiben, würde den Rahmen dieses Handbuches um ein vielfaches sprengen, im Kapitel 6 stehen noch einige Informationen über die interne Arbeitsweise eines Computers. Stattdessen wollen wir gemeinsam unser erstes Programm in **Cluster** schreiben.

Dazu installieren Sie bitte **Cluster** auf ihren System. Wie dies geschieht erfahren Sie unter 2.1 ab Seite 3, danach kommen Sie bitte hierher zurück.

Wenn alles gutgegangen ist, müßte **Cluster** nun auf Ihrem System einsatzbereit sein. Falls Sie **Cluster** auf Disketten installiert haben, booten Sie mit der Bootdiskette, starten Sie den Editor, und legen Sie Ihre Cluster-Diskette in das Laufwerk. Falls Sie über eine Festplatte verfügen, booten Sie wie gewohnt und starten den Editor.

Nun müßten Sie vor sich das Fenster des Editors sehen können und darin ein kleines farbiges Rechteck, den Cursor. Er gibt die Schreibposition an. Nun geben Sie bitte folgenden Programmtext ein, bitte achten Sie darauf, daß ihnen dabei kein Fehler unterläuft, Computer sind irrsinnig pingelig.

```
MODULE ErstesProgramm;
FROM InOut IMPORT WriteGrp;

VAR
  i  : INTEGER;


BEGIN
  FOR i:= 1 TO 10 DO
    WriteString("Mein erstes Clusterprogramm");WriteLn;
  END
END ErstesProgramm.
```

Nachdem Sie dies getan haben, speichern Sie den Text in das Verzeichnis „`Cluster:Work/txt`“ unter dem Namen „`ErstesProgramm.mod`“. Falls Sie mit dem normalen Cluster-Editor arbeiten, erreichen Sie den Filerequester über **SHIFT** + **F6** ⇔ (SAVE), falls Sie mit dem neuen HiTex-Editor arbeiten, wählen Sie aus dem Menue „Project“ „Save as...“ aus.

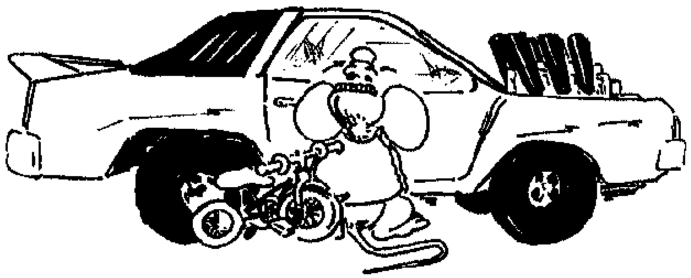
Danach drücken Sie auf **F8**, um den Text zu compilieren. Hat der Compiler den Text ohne Fehlermeldung übersetzt, können Sie nun **Alt** + **F8** drücken, um das Programm zu starten. Nun sollte in einem Ausgabefenster der Text „Mein erstes Clusterprogramm“ erscheinen.

Hat der Compiler jedoch einen Fehler gemeldet, prüfen Sie bitte, ob Sie alles so eingegeben haben, wie es hier steht.

Prima, Sie haben eben Ihr erstes Clusterprogramm geschrieben und gestartet. Nun wollen wir Sie aber nicht länger aufhalten, sondern wünsche Ihnen viel Spaß mit Ihrer neuen Programmiersprache. Sollten Sie einmal fast über einem Fehler verzweifeln, denken Sie immer an folgende zwei Regeln:

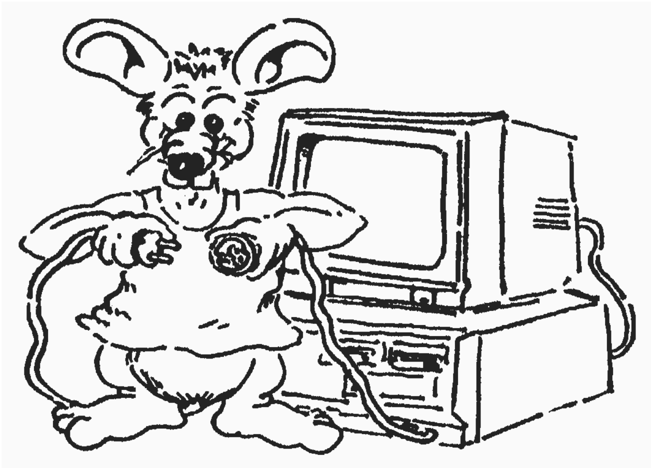
- Der Computer macht immer nur das, was Sie ihm sagen.

- Der Fehler steckt meist in dem Programmstück, in dem man ihn am wenigsten vermutet.



Cluster la vista !

Kapitel 2
Installation und Bedienung der Entwicklungsumgebung



Die meisten Compiler, die derzeit auf dem Markt sind, bestehen aus getrennten Komponenten: Editor, Compiler, Linker, und gelegentlich ist noch ein Debugger vorhanden. Dies hat jedoch zur Folge, daß man vor jedem Übersetzen erst den Editor verlassen muß, sofern man nicht genug Speicher hat um alle Programme gleichzeitig im Speicher zu halten, um das Programm vom CLI oder Workbench aus zu compilieren. Dasselbe ist dann mit dem Linker zu machen und so weiter.

Da dies nun doch ein ziemlicher Aufwand ist, und gerade deshalb Anfänger von einer Compilersprache abhält, greifen diese lieber zu Basic, bei dem man direkt aus dem Editor sein Programm starten kann, ohne dabei einen riesigen Aufwand zu haben.

Doch auch erfahreneren Programierern geht die oben beschriebene Prozedur mit der Zeit auf die Nerven. Vor allem, da gerade aufgrund des erneuten Einlesens des Quelltextes durch den Compiler viel Zeit verloren geht. (Computerfreaks sind ja bekanntlich von Natur aus ungeduldig, und jeder an sich überflüssige Tastendruck ist schließlich nach der hundertsten Wiederholung zuviel).

Aus diesem Grund beschlossen wir einen anderen Weg zu gehen und den Komfort eines Interpreters mit den Vorzügen eines Compilers zu verbinden. So entstand eine Entwicklungsumgebung, bei der Editor, Compiler, und Linker (in naher Zukunft auch ein Debugger) in einem einzigen Programm verbunden sind. Die Folge ist, daß man nun direkt aus dem Editor (= Speicher $\rightarrow$ schnell) compilieren, linken und das Programm auch direkt starten kann.

Selbstverständlich befindet sich auch noch eine über ARexx ansteuerbare CLI-Version von Compiler und Linker auf der Diskette, damit Sie auch weiterhin ihren Lieblingseditor verwenden können, ohne dabei auf Komfort verzichten zu müssen.

2.1 Installation

Achtung: Bevor Sie irgendeinen der nächsten Schritte ausführen, legen Sie die erste Diskette ein, und schauen Sie nach, ob sich darauf ein File „ReadMe“ befindet. Falls dem so ist, führen Sie einen Doppelklick darauf aus, und lesen den Text genau durch. Dort finden Sie nämlich eventuelle letzte Änderungen, die nicht mehr in das Handbuch aufgenommen werden konnten.

Bevor Sie die Entwicklungsumgebung (im folgenden „**EU**“ genannt) benutzen können, müssen Sie **Cluster** auf ihrem Rechner installieren. Hierzu dient das Programm „`Cluster Install`“, das Sie auf der ersten Diskette finden.

Starten Sie es einfach von der Workbench, und folgen Sie den Anweisungen, die das Programm ihnen gibt.

Falls Sie das Programm auf Festplatte installiert haben, ist die **EU** sofort einsatzbereit.

Falls Sie Disketten benutzen, booten Sie jetzt von „Cluster-Boot“. Legen Sie die Arbeitsdiskette „Cluster“ in Ihr zweites Laufwerk. Falls Sie nur eines besitzen, legen Sie sie ein, sobald Sie dazu aufgefordert werden.

2.2 Aufruf und Parameter

2.2.1 Start von der Workbench

Von der Workbench starten Sie die **EU** einfach, indem Sie auf das Editor-Icon mit der Maus einen Doppelklick ausführen. Sie haben dabei die Möglichkeit, durch eine erweiterte Auswahl[2] einen oder

[2]Dies geschieht, indem Sie die Shift-Taste gedrückt halten und dabei mit der Maus die gewünschten Texte durch einen einmaligen Klick anwählen; dann

mehrere Texte beim Start mit einzuladen, wobei bei mehreren Texten diese in getrennten Fenstern untergebracht werden.

Eine weitere Möglichkeit die **EU** zu starten besteht darin, auf ein gewünschtes Text-Icon einen Doppelklick auszuführen, sofern sich der Editor im Verzeichnis „`Cluster:`“ befindet (wenn nicht müssen Sie erst den Default Tool Eintrag Ihrer Texte mit dem Info-Befehl der Workbench ändern). Dadurch wird die **EU** gestartet und der angewählte Text geladen. Führt man einen Doppelklick auf ein Icon eines gespeicherten „`OldStates`“ aus, wird die **EU** gestartet, und dieser State geladen.
(OldStates: siehe 2.4.7)

2.2.2 Start von der SHELL/CLI

Von Shell/CLI rufen Sie die **EU** unter folgendem Format auf:

`Editor` $\{TextNamen\}$ |`OldState`

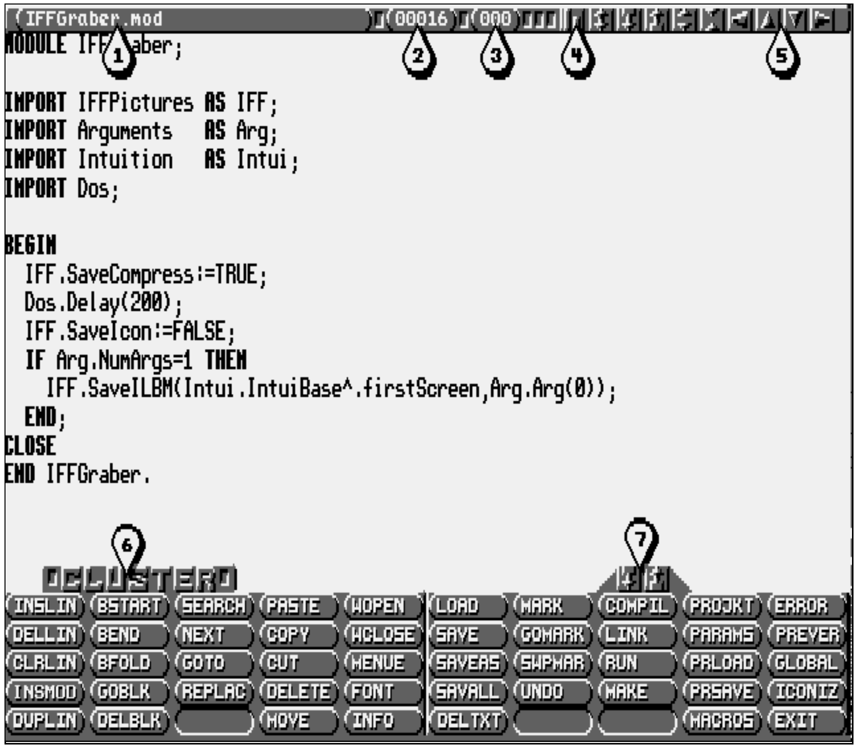
Werden dabei mehrere Textnamen hintereinander angegeben, wird für jeden Text ein Textfenster geöffnet. Gibt man einen „`OldState`“ als Parameter an, wird dieser geladen.

Der Aufruf der ARexx-Version von Compiler, Linker, Loader und Make werden in einem eigenem Kapitel weiter hinten beschrieben.

wählen Sie den Editor mit immer noch gedrückter Shift-Taste und einem Doppelklick an.

2.3 Allgemeine Bedienung des Editors

2.3.1 Bildschirmaufbau



Im oberen Teil des Bildschirms befindet sich die Titelzeile; jedes Fenster hat seine eigene (wie Sie neue Fenster öffnen, kommt später). In ihr steht der Name des aktuellen Textes **(1)**, daneben befinden sich zwei Felder mit Nummern, wobei das erste **(2)** die aktuelle Zeile, das andere **(3)** die aktuelle Spalte angibt, in der sich der Cursor[3] gerade befindet.

Ein kleines Stück weiter rechts befinden sich eine Reihe von Symbolen, wovon uns zunächst mal nur das erste und die letzten vier interessieren. Das erste **(4)**, mit einem großen Rechteck neben einem kleinen, schaltet auf Mausklick zwischen dem großen und dem kleinen Font (Zeichensatz) um. Die vier Pfeile **(5)** dienen dazu, den Text in die entsprechende Richtung zu scrollen[4].

Am unteren Bildschirmrand befindet sich das Menü, seine Bedienung wird in 2.3.2 erklärt. An der Lasche mit der Aufschrift Cluster **(6)** kann man das Menü fassen und nach unten oder oben schieben. Die beiden Pfeilen rechts **(7)** dienen dazu, das Menü entweder ganz ein- oder ganz auszufahren.

2.3.2 Benutzung des Menüs

Die einzelnen Punkte des Menüs lassen sich durch die Maus anwählen, indem man sie mit der linken Taste anklickt. Da sich der gesamte Editor sowohl über die Maus, als auch über die Tastatur bedienen läßt, können Sie auch alle Menüfunktionen über Tasten, bzw. über Tastenkombinationen erreichen (so daß man nach ein bißchen Übung das Menü ruhig nach unten schieben kann, um statt dessen mehr Textzeilen zur Verfügung zu haben). Jedoch nicht über so komplizierte Kombinationen wie z. B. Ctrl+x-Esc+j, die Sie sicher aus anderen Programmen her kennen, sondern ganz einfach auf folgende Weise:

Das Menüfeld besteht aus zweimal fünf (also 10) Spalten, jede dieser Spalten entspricht einer Funktionstaste – also **F1** der ersten, **F2** der zweiten, und so weiter bis **F10** für die zehnte Spalte. Des weiteren besteht es aus fünf Zeilen, wovon sich die Felder in

[3] Ein kleines blinkendes Rechteck, das die Schreibposition angibt

[4] Der Textausschnitt wird entsprechend weiterbewegt

der ersten durch die entsprechende Funktionstaste erreichen läßt (z. B. **F4** für **PASTE**), die in der zweiten durch die entsprechende Funktionstaste zusammen mit den **Shift** -Tasten (z. B. **SHIFT**+**F3** für **NEXT**), die dritte Zeile entsprechend zusammen mit den **Alt** -Tasten, die vierte mit **Ctrl** und die fünfte mit den *A* -Tasten.

Im übrigen ist das Menü so aufgebaut, daß die wichtigsten Funktionen in den oberen Zeilen, und die unwichtigeren oder gar gefährlichen in den unteren liegen. Dadurch haben Sie die Möglichkeit, das Menü nur ein paar Zeilen auszuziehen und dennoch problemlos zu arbeiten. Andererseits besteht nicht die Gefahr, durch vertippen z. B. etwas zu löschen, denn **Ctrl**+**F4** drückt man nicht so leicht wie nur z. B. **F4**.

Die leeren, vertieft liegenden Felder sind noch nicht mit Funktionen belegt und für Updateversionen vorgesehen.

2.3.3 Bedienung der Requester

Auch wenn oben schon einmal erwähnt, sei noch einmal betont, daß sich alle Requester komplett sowohl durch die Maus als auch durch die Tastatur bedienen lassen. Ausgenommen sind natürlich Namenseingaben.

Zur Bedienung mit der Maus, einfach den gewünschten Button[5] anklicken.

Bei Benutzung des Keyboards dient ein blinkendes Rechteck (der Einfachheit halber im Folgenden auch Cursor genannt) dazu, das gewünschte Feld auszuwählen. Zu diesem Zweck kann man ihn mit den Cursortasten[6] von Gadget zu Gadget bewegen. Um die Funtion auszuführen, muß man dann nur noch **Return** (manchmal auch zweimal) drücken.

Die Anfangsposition wurde extra immer so gewählt, daß auch durch ein übereiltes Return nichts zerstört werden kann.

Zur Benutzung der Stringgadgets (Felder zur Eingabe von Zeichenketten) sei nur gesagt, daß man durch drücken von **SHIFT**+**Del** einen bestehenden Namen löschen, mit der Maus den Cursor bewegen, und sogar Macros (dazu später) in ihnen verwenden kann.

Bei Stringrequestern, die zur Eingabe eines Pfades dienen, erhält man nach drücken von **Help** einen Filerequester, mit dem sich der entsprechende Pfad zusammenbauen läßt.

Eine besondere Art von Stringgadgets sind Felder, die zur Eingabe von Zahlen dienen.

Hier haben Sie Möglichkeit, direkt im Gadget zu rechnen. Wollen Sie z. B. den Inhalt verdoppeln, so müssen Sie nur $\boxed{*}$ und „2“ und Return drücken, und die Rechnung wird ausgeführt. Das selbe funktioniert natürlich auch mit allen anderen Rechenarten.

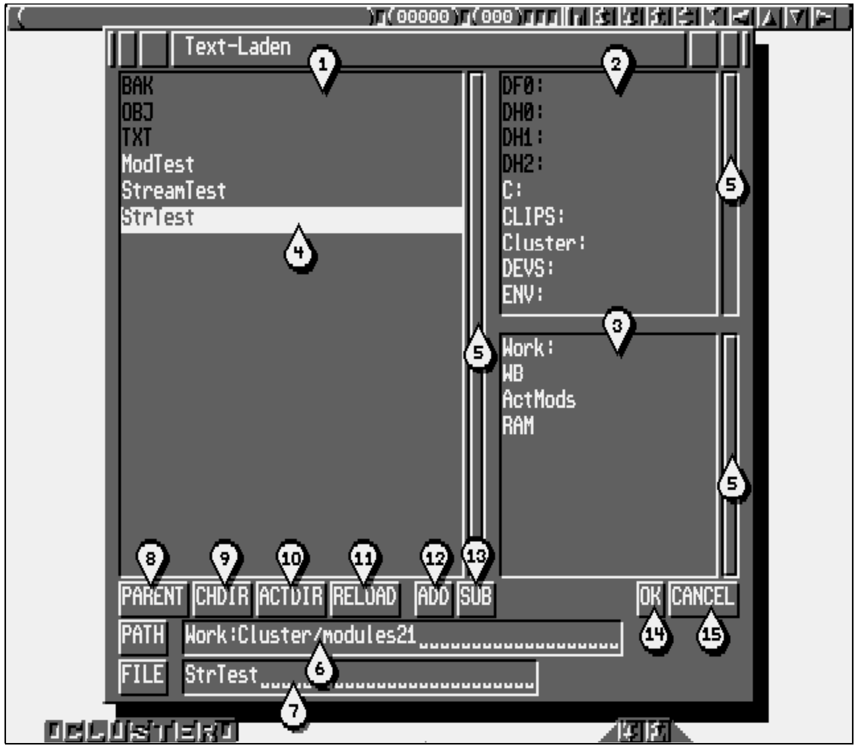
[5]Knopf, Gadget
[6]Pfeiltasten neben der Returntaste

In einigen Requestern finden sich Tasten, die bei Betätigung den Eindruck erzeugen, nun nicht mehr erhaben (= aus) sondern vertieft (= an) zu liegen, sonst aber nichts bewirken; sie dienen dazu, Einstellungen vorzunehmen, die zur Ausführung anderer Funktionen nötig sind, etwa ob ein Text im ASCII- oder im Clusterformat gespeichert werden soll.

In wieder anderen befinden sich Gadgets, die wie Schieberegler aussehen (Proportionalgadgets), wie sie z. B. zur Farbeinstellung verwendet werden. Man bedient sie, indem man entweder mit der Maus den Knopf faßt (anklicken und Mausknopf gedrückt halten) und ihn bewegt, ober- oder unterhalb in das Feld klickt, wodurch der Knopf schrittweise auf den Mauszeiger zuwandert, oder den Cursor auf den gewüschten Schieber bewegt und ihn durch die entsprechenden Cursortasten (auch mit Alt oder Shift) verstellt.

Des weiteren haben alle Requester gemeinsam, daß man sie über die **Esc**-Taste verlassen kann (die entspricht dem Cancel-Button).

2.3.4 Laden eines Textes



Um einen Text zu laden drücken Sie **F6** ⇔ (LOAD)[7]; sollte sich ein noch nicht gesicherter Text im Speicher befinden, wird selbstverständlich zuerst noch einmal nachgefragt, ob er nicht erst noch gesichert werden soll. Danach wird ein Requester geöffnet.

Er besteht aus einer Anzeige des aktuellen Verzeichnisinhalts **(1)**. Das Verzeichnis wird jedoch erst geladen, wenn man in das

[7]Der Text in einer solchen Box hinter der Tastenkombination gibt die Abkürzung des entsprechenden Menüfeldes bei Mausbenutzung an

Anzeigefeld mit der Maus klickt oder mit dem Cursor hinein fährt[8] (Dies spart Zeit, besonders bei der Arbeit mit Diskettenlaufwerken). Darin werden Verzeichnisse und Dateien, die sich im eingestellten Verzeichnis befinden, alphabetisch aufgelistet, Verzeichnisse oberhalb der Files in dunkler Schrift.

Rechts daneben befindet sich eine Scrollbox **(2)**, die die vorhandenen Laufwerke beinhaltet. Real vorhandene (physikalische) Laufwerke werden in dunkler Schrift oberhalt von denen angezeigt, welche mit dem Assign-Befehl erstellt wurden (logische Laufwerke). Näheres zu diesem AmigaDOS-Befehl lesen Sie bitte in Ihrem AmigaDOS-Handbuch nach.

Unterhalb dieses Feldes liegt eine Auswahlbox **(3)** mit voreingestellten Pfaden. Dazu später mehr.

Um ein File, Laufwerk oder voreingestellten Pfad auszuwählen, kann man ihn mit der Maus anklicken oder mit dem Cursor, der innerhalb der Auswahlfelder zu einem Markierungsbalken **(4)** wird, anfahren und auf Return drücken. Gibt man in der Auswahlbox für Files ein Zeichen ein, springt der Markierungsbalken auf den ersten Eintrag, der mit diesem Zeichen beginnt. Drückt man beim Bewegen des Markierungsbalkens mit den Cursor-Tasten auf **SHIFT**, kann man an das obere oder untere Ende des Auswahlfeldes springen.

Ist ein Verzeichnis so groß, daß der Platz in der Auswahlbox nicht ausreicht, kann deren Inhalt mit den Proportional-Gadgets **(5)** gescrollt werden. Befindet sich der Cursor auf einem solchen Gadget, kann man dieses auch mit den Cursortasten bewegen. Drückt man dabei auf **Alt**, springt man seitenweise, mit **SHIFT** springt man an den Anfang oder das Ende der Liste.

[8]Dies geschieht nur dann, wenn das Verzeichnis gewechselt wurde. Wurde schon zuvor ein Text aus diesem Verzeichnis geladen, wird es sofort angezeigt, da sich der Verzeichnisinhalt noch im Speicher befindet.

Um das Verzeichnis, das gerade angezeigt wird, zu wechseln, kann man im Stringrequester **PATH (6)** einen neuen Pfad eingegeben oder mit dem Markierungsbalken oder der Maus ein Unterverzeichnis, ein Laufwerk oder einen vordefinierten Pfad auswählen. In das übergeordnete Verzeichnis gelangt man durch Anwählen von **PARENT (8)**.

Mit **CHDIR (9)** kann man das momentane Verzeichnis zum Standardverzeichnis[9] erklären, welches alle neuen Texte gemeinsam haben werden. Um von einem anderen Verzeichnis in das Standardverzeichnis zurückzukehren, muß man **ACTDIR (10)** anwählen.

Da der Verzeichnisinhalt im Speicher gehalten wird, um unötige Wartezeiten zu vermeiden, muß man, wenn das Verzeichnis von einem anderen Programm oder der Workbench bzw. SHELL geändert worden ist, die Anzeige auf den neuesten Stand bringen. Dies geschieht mit **RELOAD (11)**.

Um Pfade, die man oft benötigt, leichter zu erreichen, haben Sie die Möglichkeit, diese unter einem Kürzel abzuspeichern. Dies geschieht, indem Sie über eine der oben genannten Möglichkeiten einen Pfad auswählt und danach auf **ADD (12)** klickt. Daraufhin wird man aufgefordert, eine Abkürzung für diesen Pfad einzugeben. Ist dies geschehen, findet man den neuen Pfad in der Auswahlbox. Um einen solchen voreingestellten Pfad zu löschen, klickt man auf den zu löschenden Pfad, und danach auf **SUB (13)**.

Um ein File zu laden, kann man dessen Namen im Stringrequester **FILE (7)** eingeben und auf **OK (14)** klicken oder ein File mit der Maus und einem Doppelklick auswählen. Außerdem können Sie eine Datei mit dem Markierungsbalken und einem Doppelreturn auswählen.

[9]Dieses Verzeichnis wird von nun an immer angezeigt, wenn ein Dateirequester dargestellt wird.

Durch Drücken des Gadgets **CANCEL (15)** können Sie den Filerequester verlassen, ohne ein Aktion auszuführen.

2.3.5 Bewegung im Text

2.3.5.1 Bewegung in der Zeile

Bei jeder Zeicheneingabe bewegt sich der Cursor eine Spalte weiter nach rechts. Durch Drücken von $\boxed{\leftarrow}$ / $\boxed{\rightarrow}$ bewegen Sie ihn eine Spalte links/rechts, zusammen mit der **Alt**-Taste springt er wortweise, zusammen mit **SHIFT** springt er an den Anfang oder das Ende einer Zeile.

Bewegen Sie den Cursor weiter nach rechts, scrollt der Editor automatisch den Text nach links, bis Spalte 253. Das heißt, daß Sie pro Zeile für 254 Zeichen Platz haben.

Weiterhin besitzt der Editor eine Tabfunktion. Durch Drükken der Tabtaste springt der Cursor nach rechts, durch **SHIFT** + **Tab** nach links, der Betrag des Sprunges ergibt sich aus dem eingestellten Tabmodus:

- **Tabindent**
  Der Cursor springt immer unter den nächsten Wortanfang der vorhergehenden Zeile. Dadurch sind auch Spalten ohne konstanten Abstand möglich, was vor allem z. B. bei Variablendefinitionen zu größerer Übersichtlichkeit beiträgt.

- **Normaltab**
  Bei diesem Tabulator, haben alle Stops einen konstanten Abstand, den Sie über **Ctrl** + **Tab** ändern können.

Um zwischen diesen beiden Modi zu wechseln, drücken Sie **Alt** + **Tab**.

2.3.5.2 Bewegung im Text

Durch ↑ / ↓ bewegt sich der Cursor eine Zeile nach oben oder unten, gerät er dabei an den oberen oder unteren Fensterrand, scrollt der Text automatisch nach; mit **Alt** bewegt er sich seitenweise, mit **SHIFT** an den Anfang/Ende der Seite bzw. des Textes.

Mit der Maus kann man den Cursor zum einen, wie unter 2 erwähnt, über die vier Pfeilsymbole in der Fenstertitelleiste durch den Text bewegen, zum anderen durch Klicken mit der linken Taste an eine beliebigen Position, in ein beliebiges Fenster setzen.

Um eine bestimmte Zeile anzuspringen, brauchen Sie nur **Alt** + **F3** ⇔ (`GOTO`) drücken, oder direkt mit der rechten Maustaste in die Zeilenanzeige in der Fenstertitelzeile klicken. Daraufhin erscheint dort ein Cursor, und Sie können die gewünschte Zeilennummer eingeben, welche dann nach Drücken von Return angesprungen wird.

Für die Returntaste existieren zwei verschiedene Modi: In dem einen trennt Return die aktuelle Zeile an der momentanen Cursorposition, fügt eine neue Zeile ein, und springt in diese unter den Anfang der vorhergehenden Zeile, wie man es aus Textverarbeitungen kennt.

Im anderen Modus (Werkseinstellung) bewegt Return den Cursor nur an den Anfang, ohne die Zeile zu trennen; ist die nächste Zeile leer, wird der Cursor unter das erste Zeichen der vorhergehenden Zeile gesetzt.

Wird in diesem Modus eine ehemals leere Zeile nach der Bearbeitung mit Return verlassen, wird automatisch eine neue Leerzeile eingefügt.

2.3.5.3 Suchen im Text

Sie können mit **F3** ⇔ (`SEARCH`) ein bestimmtes Wort anspringen. Die Suche beginnt von der aktuellen Position des Cursors ab und der Cursor wird dann auf die Position des ersten Vorkommens gesetzt.

Mit **SHIFT** + **F3** ⇔ (`NEXT`) springen Sie jeweils bis zum nächsten Auftreten des Suchbegriffes.

2.3.5.4 Die Marke

Sie dient ebenfalls der Cursorsteuerung. In jedem Textfenster kann man eine eigene Marke mit Hilfe von **F7** ⇔ (`MARK`) setzen.

Diese kann danach von jeder beliebigen Textstelle durch **SHIFT** + **F7** ⇔ (`GOMARK`) wieder angesprungen werden.

Durch Drücken von **Alt** + **F7** ⇔ (`SWPMAR`) wird die Marke an die augenblickliche Cursorposition gesetzt und der Cursor auf die alte Marke. So kann man durch mehrmaliges Drücken von **ALT** + **F7** zwischen zwei Textstellen hin und her springen.

Wenn Sie **Alt** + **Help** drücken, wird ebenfalls eine Marke an die Cursorposition gesetzt, gleichzeitig wird vom Textanfang an nach dem Wort gesucht, auf dem sich der Cursor beim Aufruf der Funktion befand. Diese Funktion ist besonders praktisch, wenn Sie in einem Programm nach der Deklaration einer Prozedur, eines Typens, etc. suchen wollen, welche ja stets vor dem ersten Auftreten des Bezeichners stehen müssen.

2.3.6 Editieren

2.3.6.1 Editieren in einer Zeile

Zum Editieren in der Zeile stehen Ihnen zwei verschiedene Modi zur Verfügung: Einfügen oder Überschreiben.

In welchem Modus man sich befindet, können Sie am Cursor erkennen. Im Insertmode (Einfügen) arbeiten Sie mit einem Strichcursor, im Overwritemode (Überschreiben) mit einem Blockcursor.

Zwischen diesen Modi kann man mit **Alt** + **Del** umschalten.

Der Insertmode wirkt sich auch auf den Tab aus. Stehen rechts vom Cursor schon Wörter, werden diese durch Tab bis an die nächste Tabposition geschoben. Ideal also, um ganze Tabellen einzurücken.

Mit **BACKSPACE**[10] ← löscht man das Zeichen links vom Cursor, welches normalerweise das zuletzt eingebene ist, alle Zeichen rechts vom Cursor rücken dabei um ein Zeichen nach links.

Mit **Del** löscht man das Zeichen auf dem der Cursor gerade steht und schiebt alle Zeichen rechts davon um eine Spalte nach links.

Mit **SHIFT** zusammen fügt **Del** ein Leerzeichen ein, dies ist besonders für den Overwritemode wichtig.

2.3.6.2 Editieren im Text

Wie schon in 2.3.5.2 erwähnt verfügt der Editor über zwei Möglichkeiten, auf ein Return zu reagieren.

Bei der einen, bei der die Zeile nicht getrennt wird, kann man durch **SHIFT** + **Return** eine Zeile trennen, bei der anderen bewegt **SHIFT** + **Return** den Cursor einfach an den Anfang der neuen Zeile, ohne die Zeile zu trennen. Durch **Alt** + **Return** kann man unabhängig vom Modus zwei Zeilen vereinigen.

F1 ⇔ (`INSLIN`) fügt eine Leerzeile ein.

Mit **SHIFT** + **F1** ⇔ (`DELLIN`) kann man die aktuelle Zeile entfernen oder mit **Alt** + **F1** ⇔ (`CLRLIN`) auch einfach nur löschen. Bei erster Methode wird die Zeile vom Bildschirm entfernt und der Text rückt entsprechend zusammen. Bei zweiter Methode wird die Zeile gelöscht, sie bleibt allerdings für neue Eintragungen frei. Ideal also, um eine Programmzeile für eine neue Anweisung freizumachen. *A* + **F1** ⇔ (`DUPLIN`) verdoppelt die aktuelle Zeile.

[10]Die Taste mit dem Pfeil nach links, über der Returntaste

Durch **Ctrl** + **F3** ⇔ (`REPLAC`) können Sie ein bestimmtes Wort durch ein anderes ersetzen. Schalten Sie dabei den Schalter **BLOCK** an, wird nur innerhalb des markierten Blocks (siehe Blockfunktionen) ersetzt. Mit **INVISIBLE** bewirken Sie, daß das Ersetzen geschieht, ohne daß man es direkt sieht; die Funktion ist dann schneller. Besonders wichtig im Zusammenhang mit **ALL**.

Wählen Sie **ALL**[11], so werden alle entsprechenden Wörter ersetzt, ohne daß vorher noch mal nachgefragt wird.

Wenn nicht, werden Sie bei jedem Auftreten des Wortes gefragt, ob es ersetzt werden soll oder nicht. Außerdem haben Sie dabei jedesmal die Möglichkeit, doch noch auf **ALL** umzusteigen oder die Aktion mit **CANCEL** abzubrechen.

2.3.7 Blockfunktionen

Mit Hilfe des Blocks, einem Zwischenspeicher, kann man ganze Textstücke innerhalb des Textes, aber auch von einem Text in einen anderen kopieren, verschieben oder löschen.

Man definiert einen Block, indem man entweder den Blockanfang mit **F2** ⇔ (`BSTART`) und das Blockende mit **SHIFT** + **F2** ⇔ (`BEND`) setzt[12]. Oder, indem man mit dem rechten Mausknopf den Blockanfang setzt, und dann ohne den Knopf loszulassen

[11]Schalter rechts unten im sich öffnenden Requester
[12]Der so markierte Block wird dann invertiert dargestellt

den Mauszeiger nach unten bewegt, und damit den Block aufzieht.

Erreicht man dabei den unteren bzw. rechten Fensterrand, scrollt der Text automatisch nach. Durch Loslassen der Maustaste setzt man das Blockende. Wichtig ist, daß Sie auch an einem bestehenden Block durch **F2** / **SHIFT** + **F2** den Blockanfang oder das Blockende versetzen können.

Mit **Ctrl** + **F2** ⇔ (GOBLK) springt man an den Blockanfang, befindet man sich schon dort, an das Blockende.

A + **F2** ⇔ (DELBLK) löscht einen bestehenden Block, nicht den Text im Block, hebt also nur die Blockmarkierung auf.

Einen definierten Block kann man nun mit **SHIFT**+**F4** ⇔ (COPY) in den Puffer kopieren. Diesen kann man dann wieder mit **F4** ⇔ (PASTE) an jeder beliebigen Stelle, auch in einem anderen Window, wieder einfügen. Dabei wird automatisch das eingefügte Textstück zum neuen Block erklärt, den man nun bequem durch **Ctrl** + ← / → um den Standardtab nach links oder rechts verschieben kann, um ihn in die Einrückstruktur des umgebenden Texts einzupassen.

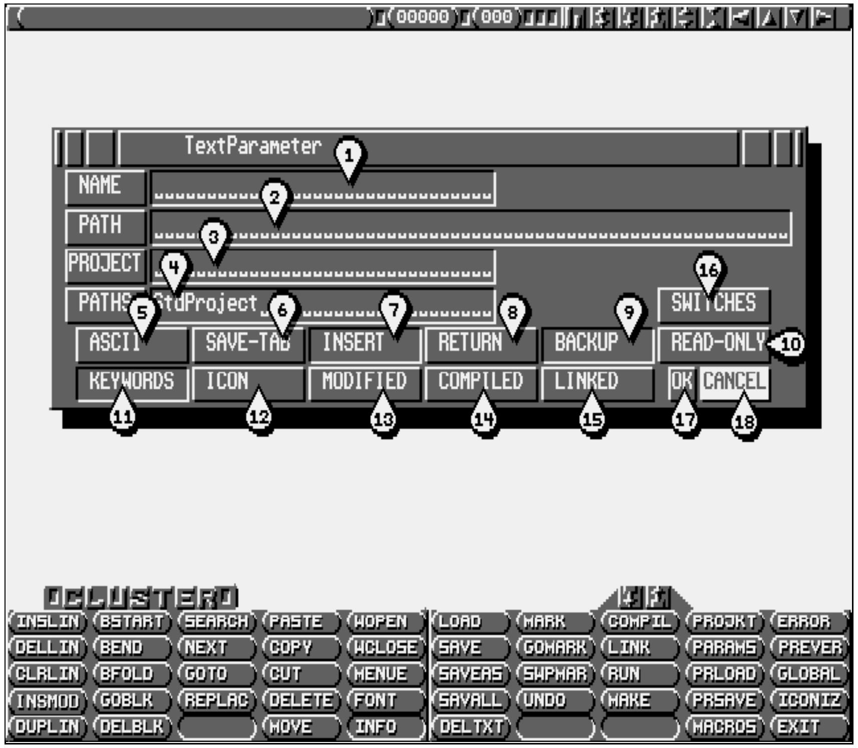
Auch **Alt** + **F4** ⇔ (CUT) kopiert den Block in den Puffer, löscht jedoch gleichzeitig den markierten Text.

Ctrl + **F4** ⇔ (DELETE) löscht den markierten Text, ohne ihn vorher in den Puffer zu kopieren, fragt aber vorher noch einmal zur Vorsicht nach.

A + **F4** ⇔ (MOVE) vereinigt die Funktion von CUT und PASTE. Diese Funktion dient dazu, einen Block innerhalb eines Fensters zu verschieben. Einfach einen Block markieren und an der Stelle, an die er hin soll, *A* + **F4** drücken.

2.3.8 Sichern eines Textes

2.3.8.1 Textinfo



Bevor man einen Text sichert bzw. speichert, kann man noch diverse Einstellungen vornehmen, die beim Speichern dann berücksichtigt werden. Hierzu dient das Feld **INFO** im Menü (auch *A* + **F5**). In dem erscheinenden Requester kann man den Namen **(1)**, den Pfad **(2)** und das Projekt **(3)** (mehr dazu in 2.6), sowie diverse CompilerSwitches **(16)** verändern bzw. neu eingeben.

Wenn Sie die Pfadangaben des Textes nicht eintippen wollen, dann drücken Sie doch einfach in einem der Pfadrequester **Help**, und schon öffnet sich der gewohnte Filerequester, mit dem Sie den gewünschten Pfad auswählen können.

Im Feld **PATHS (4)** kann man außerdem eine Pfadliste angeben, die beim Importieren von Modulen berücksichtigt werden soll (siehe 2.6).
Außerdem kann man hier festlegen, ob:

(5) - Der Text im Clusterformat oder im ASCII-Format (Schalter ASCII eingeschaltet) gespeichert wird. Im Clusterformat werden noch einige Informationen mehr abgespeichert, so zum Beispiel die Cursor- und Blockposition, ob Insert oder Overwrite, ob der Text schon compiliert wurde und einiges mehr. Außerdem spart dieses Format Platz.

(6) - Tabulatoren als entsprechend viele Leerzeichen oder als Tabs (Schalter **SAVE-TAB** auf an) gespeichert werden, Tabspeicherung spart Platz auf der Diskette, doch gibt es einige Programme, die Tabs nicht vertragen.

(7) - Zeigt an, ob der Text sich im Insert- oder im Overwritemodus befindet.

(8) - Hat keine Bedeutung mehr, da diese Einstellung jetzt global vorgenommen wird.

(9) - Vor der Speicherung eines veränderten Textes ein Backup von der alten Version erstellt werden soll (**BACKUP** auf an).

(10) - Änderungen im Text untersagt sind, man ihn also nur lesen kann (**READ-ONLY** an), um zu verhindern, daß ein bereits fertiggestellter Text aus Versehen beschädigt wird.

(11) - Schlüsselwörter fett dargestellt werden sollen. Ist dieser Schalter bei ASCII-Texten angeschaltet, wird beim Abspeichern des Textes ein spezieller Header an den Anfang des Textes gesetzt, der die Informationen, die normalerweise im im Clusterformat enthalten sind, enthält. Beim erneuten Einladen wird dieser Header automatisch entfernt, nachdem er ausgewertet wurde.

(12) - Dieser Text und das gelinkte Programm mit Icons versehen werden sollen.

Des weiteren gibt es noch drei Schalter, die etwas über den Zustand des Textes aussagen:

(9) - MODIFIED gibt an, ob der Text seit dem letzten Speichern verändert worden ist.

(10) - COMPILIED gibt an, ob der Text schon übersetzt worden ist.

(11) - LINKED gibt an, ob der Text schon gelinkt worden ist.

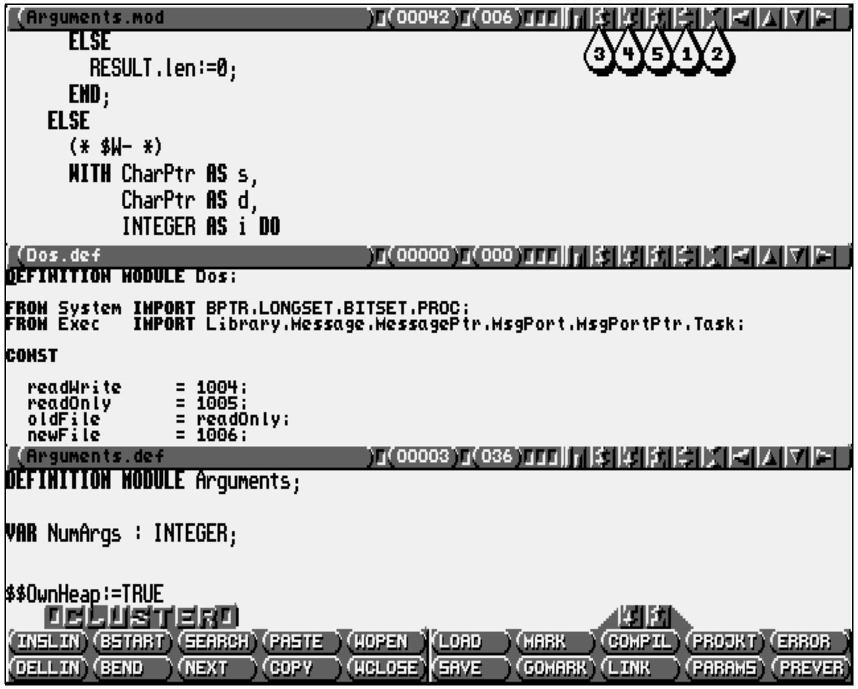
Diese Schalter kann man auch bewußt verändern, allerdings sollte man damit sehr vorsichtig sein und es nur tun, wenn man genau Bescheid weiß.

Durch Drücken von **OK (17)** übernimmt man die gemachten Einstellungen in das Programm.

Durch Drücken von **Esc** oder **CANCEL** verläßt man den Requester wieder, ohne Änderungen zu übernehmen.

2.4 Sonderfunktionen des Editors

2.4.1 Window-Handhabung



Hier wird endlich das Geheimnis um die schon oft angesprochenen Windows (Fenster) gelüftet. Nehmen wir an, Sie möchten von einem Text in einen anderen etwas kopieren, oder aus einem anderen Grund mit mehreren Texten gleichzeitig arbeiten, so benötigen Sie ein zweites oder drittes Fenster.

Dabei handelt es sich jedoch nicht um Workbenchfenster, sondern durch eine neue Titelzeile wird lediglich ein Teil der gesamten Schreibfläche abgetrennt.

Um nun ein neues Window zu erhalten, muß man nur mit der linken Maustaste die oberste Titelleiste packen und Sie nach unten ziehen. Beim Loslassen der Taste entsteht am oberen Bildschirmrand eine neue Titelzeile, noch ohne Namen.

Achtung, packen Sie die Zeile nicht ganz oben, da Sie sonst die ganze Schreibfläche bewegen.

Indem Sie ein neues Fenster geöffnet haben, haben Sie bereits gelernt wie man die Fenstergröße verändert. Sie müssen nur die Titelzeile des gewünschten Fensters packen und können diese nach oben oder unten verschieben.

Sollten dabei andere Fenster im Wege sein, werden diese mitgeschoben.

Dasselbe können Sie erreichen, wenn Sie in dem aktivierten Fenster (Fenstertitel ist hell dargestellt, blinkender Cursor) **Ctrl** + **SHIFT** + ↑ / ↓ drücken. Dadurch bewegen Sie die Titelleiste des aktivierten Fensters nach oben oder unten. Bewegen Sie die oberste Leiste nach unten, wird ebenfalls ein neues Fenster erzeugt.

Sie aktivieren ein Fenster, indem Sie es anklicken oder mit dem Cursor durch **Ctrl** + ↑ / ↓ von Fenster zu Fenster springen.

Wenn in Zukunft vom aktuellen Text oder Fenster die Rede ist, wird damit dann immer das momentan aktive Fenster und der in ihm befindliche Text bezeichnet. Dieses ist daran zu erkennen, daß der Namen in der Titelzeile hell ist.

Ein Fenster löschen Sie, indem Sie es aktivieren, und dann *A* + **F6** ⇔ (DELTXT) drücken. Selbstverständlich wird erst noch nachgefragt, ob ein noch nicht gesicherter Text abgespeichert werden soll. Wird diese Funktion bei nur einem einzigen Fenster aufgerufen, wird nur der Text gelöscht, das Fenster bleibt bestehen.

Wollen Sie einem Fenster die gesamte Schreibfläche gönnen, so drücken Sie **F5** ⇔ (WOPEN) oder klicken Sie auf das Symbol mit den zwei auseinander weisenden Pfeilen **(1)** in der Titelleiste.

Um ein Fenster ganz zu schließen drücken Sie **SHIFT** + **F5** ⇔ (`WCLOSE`) oder klicken die zueinander weisenden Pfeile **(2)** an.

Wollen Sie die Fenster anders gruppieren, haben Sie dazu verschiedene Möglichkeiten:

- Mit dem Doppelpfeilsymbol **(3)** vertauschen Sie das Fenster mit dem darüber liegenden.
- Mit dem Pfeil nach unten **(4)** wird Ihr Fenster zum untersten.
- Mit dem Pfeil nach oben **(5)** zum obersten.

Bei diesen Vertauschungsaktionen bleibt die Größe der Fenster unverändert. Übrigens kann man in den verschiedenen Fenstern unabhängig den großen oder den kleinen Font (Zeichensatz) verwenden. Dies ermöglicht es Ihnen, in einem Fenster mehr Zeilen darzustellen und damit die sichtbare Informationsmenge zu erhöhen.

2.4.2 Folding

Unter Folding versteht man das Einklappen eines bestimmten Bereiches des Textes. Es dient dazu, für größere Übersicht zu sorgen. Unwichtige Programmteile können weggeblendet werden und stören nicht beim Durchsehen eines Programmes. Natürlich sind diese so verdeckten Programmteile auf Tastendruck wieder aufdeckbar.

Anstelle des Textes steht nach dem Einklappen dort ein kleines Rechteck ▭, in dessen Zeile nicht editiert werden kann. Eingeklappt werden können Prozeduren, indem man mit dem Cursor in die Zeile des Prozedurkopfes geht und auf **Help** drückt. Um die

Prozedur wieder aufzuklappen, wieder auf den Prozedurkopf gehen und erneut auf **Help** drücken.

Einen markierten Block kann man ebenfalls mit **Help** einklappen. Um ihn wieder aufzuklappen, gehen Sie mit dem Cursor in die Zeile mit dem Rechteck und drücken **Help**.

Eingeklappte Textstücke lassen sich beliebig schachteln, d.h. man kann eine Prozedur auch einklappen, wenn eine darin befindliche Prozedur oder ein Block schon eingeklappt wurde.

2.4.3 Save All

Haben Sie die **EU** mit RUN gestartet und wollen Sie sicher sein, daß alle Texte gesichert sind, bevor Sie ein anderes Programm starten, müssen Sie nicht erst jedes Fenster aktivieren und **SHIFT** + **F6** drücken.

Durch Drücken von **Ctrl** + **F6** ⇔ (`SAVALL`) wird für alle noch nicht gesicherten Texte nachgefragt, ob Sie diese abspeichern wollen.

2.4.4 Undo

Um ungewollte Änderungen in einer Zeile rückgängig zu machen, verfügt der Editor über einen Korrekturpuffer. Durch Drücken von **Ctrl** + **F7** ⇔ (`UNDO`) können Sie, soweit Sie die bearbeitete Zeile noch nicht verlassen haben, diese wieder in ihren ursprünglichen Zustand versetzen.

Nebenbei läßt sich diese Funktion hervorragend dazu ge(miß)-brauchen, um eine Zeile zu verschieben. Dazu löschen Sie die zu verschiebende Zeile mit **SHIFT** + **F1** ⇔ (`DELLINE`), sie steht nun im Korrekturpuffer.

Nun können Sie, solange Sie nichts in einer anderen Zeile verändern, den Cursor bewegen und an der gewünschten Position

Ctrl + F7 drücken, wodurch der Pufferinhalt an dieser Stelle eingefügt wird.

2.4.5 Macros

Um Sie bei monotonen Formatierarbeiten oder anderen sich wiederholenden Arbeiten zu entlasten, können Sie auf allen Tasten, außer den Funktionstasten, eine beliebige Tastenfolge (Macro), einschließlich Menüfunktionen speichern. Diese wird bei Aufruf dann genauso wiederholt.

Um ein Macro aufzuzeichnen drücken Sie Ctrl und die Taste, auf der Sie das Macro speichern wollen. Der Bildschirm wird invertiert und alle Eingaben von jetzt an, auch andere Macroaufrufe, werden aufgenommen. Die Aufnahme wird durch erneutes Drücken von Ctrl und der Macro-Taste beendet.

Der Aufruf des Macros erfolgt über die rechte *A*-Taste (die linke bis auf wenige Tastenausnahmen z.B. n, m) und der Taste, auf der das Macro liegt.

Um häufig benutzte Marcos nicht nach jedem Neustart neu definieren zu müssen, können Sie die Macrobelegung aller Tasten durch *A* + F9 ⇔ (`MACROS`) unter einem von Ihnen gewählten Name abspeichern oder eine bereits gespeicherte bei einem Neustart wieder laden[13]. So haben Sie die Möglichkeit, sich mehrere Macrotabellen anzulegen und zwischen diesen zu wechseln.

[13]Da wir im Moment an einem komplett neuen Editor arbeiten, lohnte es sich nicht mehr, für das Laden der Macro-Tabelle einen eigen Menüpunkt zu belegen (aus Zeitgründen). Sie erreichen diese Funktion, indem Sie im Macro-Save-Requester auf Esc drücken.

2.4.6 Iconiz

Wenn Sie mit der Shell arbeiten, empfiehlt es sich, den Editor mit „`run`“ zu starten, damit man nebenbei auf der SHELL weiterarbeiten kann.

Will man nun auf der SHELL oder Workbench etwas machen, so sollte man nicht nur die Schreibfläche nach hinten schalten, sondern Ctrl + F10 ⇔ (`ICONIZ`) drücken, besonders wenn man wenig Speicher hat. Dadurch schließt sich der Editor (Speichergewinn) und auf der Workbench erscheint ein kleines Icon.

Führt man auf dieses einen Doppelklick aus, so öffnet sich der Editor wieder, alle Texte bleiben dabei unverändert.

2.4.7 OldStates

Arbeitet man längere Zeit an einem oder an mehreren großen Projekten, so ist es mühsam, nach jedem Neustart alle entsprechenden Texte zu laden. Daher läßt sich der aktuelle Zustand des Editors, d. h. welche Texte geladen sind, wieviele Fenster an welchen Positionen geöffnet sind, sowie welches das aktuelle Directory ist, als sogenannter „OldState“ abspeichern. Hierzu drückt man Ctrl + F9 ⇔ (`PRSAVE`), worauf man den vertrauten Filerequester erhält und damit den aktuellen Status abspeichern kann. Standardmäßig liegen sie in „`Cluster:`“.

Führt man nun einen Doppelklick auf das Icon des States aus, wird der Editor geladen, sowie alle Texte, die im Statefile vermerkt sind. Man befindet sich also im gleichen Zustand wie zu dem Zeitpunkt, da man diesen State abgespeichert hat.

Die gleiche Wirkung erzielt man, wenn man in der Shell einen gespeicherten „State“ als Parameter beim Programmstart angibt oder nach dem Start des Editors auf Alt + F9 ⇔ (`PRLOAD`) geht, wodurch man einen Filerequester erhält, mit dem man den gewünschten „State“ aussuchen kann.

2.4.8 TimeSave

Besonders nach längerem Arbeiten an einem Text ist es sehr ärgerlich, wenn durch irgendein anderes Programm der Computer abstürzt und alle Arbeit umsonst war. Die letzte Abspeicherung lag nach Murphy garantiert schon einige Zeit zurück. Um dies zu vermeiden, bietet der Editor eine Autosave-Funktion an, die nach einer festgelegten Zeit alle veränderten Texte in das Verzeichnis „`Cluster:TimeSave`“ abspeichert. Dadurch ist auch sichergestellt, daß das Orginal auf Platte oder Diskette solange erhalten bleibt, bis Sie den Text von Hand abspeichern. Den Zeitabstand, nach dem gespeichert werden soll, läßt sich im Global-Requester (siehe unter 2.4.9) einstellen.

2.4.9 Voreinstellungen

An der **EU** kann man, ähnlich wie auf der Workbench, mit einem Preference-Requester verschiedene Voreinstellungen vornehmen. Man gelangt in ihn durch Drücken von **Alt** + **F8** ⇔ (`GLOBAL`).

Folgende Einstellungen können Sie damit vornehmen:

(1) - Bewirkt, daß beim Verlassen des Requesters in den Interlacemodus umgeschaltet wird, um mehr Bildschirmzeilen dargestellt zu bekommen (nur mit Flikkerfixer zu empfehlen).

(2) - Besitzer eines Rechners mit ECS und Amiga OS 2.0 können mit diesem Schalter zwischen PAL und NTSC umschalten. Im NTSC-Mode hat man zwar nur 200, bzw. im Interlace 400 Bildschirmzeilen, dafür aber eine Bildwiederholfrequenz von 60 Hz und das Flimmern wird reduziert.

(3) - Die glücklichen Besitzer eines Rechners mit ECS können hiermit den Productivity-Mode einschalten.

(4) - Ist dieser Schalter an, werden beim Verlassen der **EU** alle Einstellungen automatisch gespeichert, auch wenn nicht SAVE angewählt worden ist.

(5) - Schaltet das Vergeben von Icons an Texte standardmäßig an.

(6) - Schaltet die Erzeugung von Backups an.

(7) - Ist dieser Schalter an, ist das Menü nach dem Start der **EU** ausgefahren.

(8) - Schaltet den kleinen Font (Zeichensatz) ein.

(9) - Schaltet auf ASCII-Save. Der Text wird im ISO-ASCII-Standard gespeichert[14].

(10) - Schaltet Tab-Save ein (nur in Verbindung mit ASCII-Save).

(11) - Schaltet den Insert-Modus (Einfügen von Zeichen) ein.

[14] Im Gegensatz zu einem recht weit verbreiteten Computersystem sind Files in diesem Standard auch zwischen anderen Computersystemen übertragbar.

(12) - Bewirkt, daß Return eine Zeile teilt.

(13) - Schaltet Fettdruck von Schlüsselwörtern ein. Empfehlenswert.

(14) - Sorgt dafür, daß beim Iconisieren eine Saveall-Abfrage durchgeführt wird. Wenn man ein gelinktes, absturzgefährdetes Programm testen will und Iconize aufruft, verliert man keine Texte, falls der Versuch zu einem Systemabsturz führt. Sehr zu empfehlen.

(15) - Diese Option arbeitet zusammen mit den OldStates. Ist sie eingeschaltet, wird beim Verlassen des Editors gefragt, ob der aktuelle Zustand (Texte, Fensterposition etc.) wieder unter dem selben „OldState“ gespeichert werden soll, der zuletzt geladen wurde. Wurde kein „State“ geladen, wird der Zustand unter

„`Cluster:State`“

gespeichert.

(16) - Schaltet den Symbolcache ein, ist er schon eingeschaltet, löscht dieser Schalter den Cache. Näheres siehe unter 2.5 ab Seite 34.

(17) - Schaltet den Versionsvergleich zwischen Symboldateien im Cache und denen auf der Diskette ein oder aus.

(18) - Ist dieser Schalter eingeschaltet, benutzt der Editor zwei Bitplanes für die Textdarstellung. Kommentare, die mit einem „|“ eingeleitet sind und am Zeilenanfang stehen, werden in einer anderen Farbe als der übrige Text dargestellt. Dies erhöht die Übersichtlichkeit enorm, führt aber, außer auf einem Amiga 3000/4000, im Interlace zu starkem Flackern beim Scrollen. Außerdem ist eine Verlangsamung zu erkennen, da der Blitter nicht schnell genug ist, um die zusätzliche Bitplane schnell genug mitzukopieren. Ist zu empfehlen, wenn auch anfangs gewöhnungsbedürftig.

(19) - Farbeinsteller für die Vordergrundfarbe.

(20) - Farbeinsteller für die Hintergrundfarbe.

(21) - Textspeichergröße beim Programmstart.

(22) - Anzeige der aktuellen Textspeichergröße. Wenn Sie also nur für einmal die Speichergröße ändern wollen, dann sollten Sie dies in diesem Feld und nicht im Feld STANDARD (21) tun.

(23) - Hier trägt man den Zeitraum (in Minuten) ein, nach dem alle veränderten Texte automatisch gespeichert werden sollen. Trägt man hier Null ein, findet kein Autosave statt.

(24) - Standardprojekt, das beim Start eingestellt ist.

(25) - Gibt das aktuelle Verzeichnis für neue Texte beim Neustart an.

(26) - Macrobelegung, die beim Start geladen wird.

(27) - Lädt die zuletzt gespeicherte Einstellung.

(28) - Macht Änderungen seit dem Betreten des Requesters rückgängig.

(29) - Speichert eine Einstellung. Falls Sie AUTOSAVE nicht angeschaltet haben, müssen Sie das tun, wenn Sie nicht nach jedem Neustart die Einstellungen von neuem machen wollen.

(30) - Übernimmt die Einstellungen, ohne Sie abzuspeichern, und verläßt den Requester.

(31) - Verläßt den Requester, ohne die Einstellungen zu übernehmen.

Wichtig: Die Felder **„ICON“(5)**, **„BACKUP“(6)**, **„ASCII“(9)**, **„SAVE-TAB“(10)**, **„INSERT“(11)** und **„KEYWORDS“(13)** sind eigentlich Einstellungen, welche zu einem bestimmten Text gehören und gelten nur als Starteinstellung für neuerstellte Texte. Im ASCII-Format vorliegende Texte übernehmen diese Einstellungen ebenfalls, mit Ausnahme von **KEYWORDS**, welches standardmäßig ausgeschaltet wird, soweit der Text keinen Header hat. Hat ein geladener Text andere Einstellungen, werden selbstverständlich diese genommen.

2.5 Der Compiler

Eine Bemerkung vorneweg: Sollten Sie das eine oder andere in den nächsten Kapiteln über Compiler, Linker, Loader oder Make nicht sofort verstehen, dann machen Sie sich keine Sorgen; alle Fragen werden mit Sicherheit im Einführungskapitel in die Programmiersprache **Cluster** geklärt.

2.5.1 Compiler-Switches

Compilerswitches[15] veranlassen den Compiler, einen bestimmten Bereich eines Moduls oder auch ein ganzes Modul in einer anderen Weise zu übersetzen, als standardmäßig vorgesehen.

Switches lassen sich, wenn Sie für ein ganzes Modul gelten sollen, als Parameter dem Compiler übergeben oder direkt im Quelltext setzen, was vor allem bei bereichsbezogenen Switches sinnvoll ist.
Eine solcher Switch wird folgendermaßen definiert:

`$$xxx:=yyy`

d. h. der Switch `xxx` wird gleich dem Ausdruck `yyy` gesetzt.
Beispiel

`$$RangeChk:=FALSE`

Handelt es sich um einen stapelbaren Schalter, d. h. ein Schalter der sich nur auf einen bestimmten Bereich bezieht, so können diese durch

`$$xxx:=OLD`

wieder auf ihren ursprünglichen Zustand zurückgesetzt werden. Die maximale Schachtelungstiefe beträgt dabei 16.

Zu jedem Projekt können im Projekt-Requester bis zu 24 Schalter selbstdefiniert werden. Wie diese definiert werden können Sie

[15] Switch = Schalter

Abschnitt 2.6 ab Seite 49 entnehmen.

Diese selbstdefinierten Schalter, sowie die vordefinierten, können zur bedingten Compilierung eingesetzt werden. Das heißt, je nach Schalterstellung kann man den Compiler veranlassen, ein anderes Stück Text zu übersetzten. Dies geschieht z. B. durch die folgende IF-Konstuktion:

```
$$IF xxx THEN
  Text1
$$OR_IF yyy THEN
  Text2
$$ELSE
  Text3
$$END
```

Dabei wird für diese IF-Abfrage kein Code erzeugt, sondern in Abhängigkeit der Schalter '`xxx`' und '`yyy`' entweder `Text1` oder `Text2` oder `Text3` übersetzt. Dabei darf eine Bedingung aus mehreren Schaltern bestehen, die über boolsche[16] Operatoren verknüpft sind:

```
$$IF xxx AND yyy THEN

$$END
```

[16] Näheres über boolsche Operatoren finden Sie im Einführungskapitel

Beispiel: ein benutzerdefinierter Schalter 'Turbo' für eine 020/881 Version:

```
MODULE ...;

$$IF Turbo THEN
  $$MC68020:=TRUE
  $$MC68881:=TRUE
$$END

...

$$IF Turbo THEN
  TYPE  Float = REAL;
  IMPORT VectorReal AS Vectors;
$$ELSE
  TYPE  Float = FFP
  IMPORT VectorFFP AS Vectors;
$$END
...
```

Je nachdem, ob Turbo gesetzt ist, wird ein anderes Programm compiliert. Ein anderes Beispiel wäre eine deutsche und eine anderssprachige Version des selben Programms, oder ein Modul, das sowohl in einer Library, als auch in einem normalen Programm benutzt werden soll. In einer Library darf z. B. kein `ALLOC_HEAP` aufgerufen werden.
Neben den selbst definierten Schaltern sind folgende Switches vordefiniert:

Name	**Kurz**	**Geltung**	**Init**	**Bedeutung**
OverflowChk	V	Bereich	Info[17]	Überlaufcheck bei Rechenoperationen.
RangeChk	R	Bereich	Info	Bereichsprüfung bei Zuweisungen und Indizierungen.
StackChk	S	Bereich	Info	Stacküberprüfung am Prozedur/Modulanfang.
StrZeroChk	Z	Bereich	Info	Test bei Stringzuweisungen, ob `Str.data[Str.len]=&0`.
NilChk	N	Bereich	Info	Test bei Dereferenzierungen, ob der Zeiger NIL enthält.
ReturnChk		Bereich	Info	Test, ob eine Funktion einen Wert mit RETURN zurückliefert.

[17]Info bedeutet hier, daß der Initialisierungswert von der Schalterstellung im Feld SWITCHES im Textinfo-Requester abhängt

Name	**Kurz**	**Geltung**	**Init**	**Bedeutung**
LocalChk	L	Bereich	Info	Test, ob eine Prozedur, die an eine Variable zugewiesen wird, eine globale Prozedur ist.
ConstChk		Bereich	TRUE	Der Compiler verhindert, daß REF-Parameter oder Konstanten verändert werden. Dieser Switch hat nur Auswirkungen wärend der compilierung, er hat keine Auswirkung auf den erzeugten Code.
RelaxPtr		Bereich	FALSE	Erlaubt bei Konstantendefinitionen die 'PTR wegzulassen, wenn ein Zeiger statt der Struktur erwartet wird.
ChipMem	C	Modul	Info	Das Modul wird ins ChipMem gelegt.
LongAllign	A	Modul	Info	Variablen und Elemente von Strukturen, die länger als 3 Bytes sind werden auf Langwortgrenzen gelegt.
Library		Modul	Info	Das Modul soll in einer Library verwendet werden.
Device		Modul	Info	Das Modul soll in einem Device verwendet werden.

Name	**Kurz**	**Geltung**	**Init**	**Bedeutung**
Debug		Modul	Info	Das Modul wird mit Debuginformation compiliert.
WithArea	W	Struktur	TRUE	Bei lokalen Variablen, bei Parametern einer Prozedur oder bei einer WITH-Anweisung wird ein Sicherungsbereich eingerichtet. Dieser ist nötig, falls innerhalb der Struktur eine Prozedur aufgerufen wird. Wird innerhalb keine Prozedur benutzt, sollte `(* $W- *)` davor gesetzt werden. Der Compiler meldet einen Fehler, wenn doch eine Prozedur aufgerufen wird.
WithModify	M	Struktur	TRUE	Nach einer WITH Anweisung wird der Wert des Registers (falls verändert) wieder in die Variable gesichert. `(* $M- *)` unterbindet dies, falls das Ergebnis nicht mehr benötigt wird.

Name	Kurz	Geltung	Init	Bedeutung
OwnHeap	O	Prozedur	FALSE	Funktionen, die einen offenen Rückgabetyp haben, müssen sich in einigen Situationen selbst um die Allozierung von Rückgabestackspeicher kümmern (z.B. wenn man die Ergebnisse direkt an einen VAR-Parameter übergeben will; siehe „Strings“). Dieser Schalter zeigt an, daß die Prozedur sich darum kümmert, und auch in diesen Situationen benutzt werden kann.
PushRegs	P	Prozedur	FALSE	Sichert alle Register am Prozeduranfang. Diese Prozedur kann dann auch innerhalb einer WITH Struktur benutzt werden, ohne daß die Registervariablen gesichert werden müßten.

Name	Kurz	Geltung	Init	Bedeutung
EntryCode	E	Prozedur	TRUE	Die Prozedur hat keinen Eingangscode. Darf nur verwendet werden, wenn keine lokalen Parameter auf dem Stack benötigt werden, z. B. wenn eine Prozedur rein in Assembler kodiert ist.
ModulaII		Bereich	Info	Es gelten die Syntaxregeln von Modula 2.
MC68020		Bereich	Info	Es wird Code für den 68020++ erzeugt.
MC68881		Bereich	Info	Es wird Code für den 68881++ erzeugt.
MC68040		Bereich	FALSE	Es wird Code für den 68040 erzeugt.
LargeCode		Modul	FALSE	Falls ein Modul im Objectfile 32KB überschreitet, meldet sich der Compiler mit einer Fehlermeldung. Nun kann man entweder das Modul in mehrere kleine zerlegen, oder man setzt diesen Schalter, was allerdings zu einer geringfügigen Geschwindikeitseinbuße führt.

Name	Kurz	Geltung	Init	Bedeutung
AssTypeChk		Bereich	FALSE	DerAssembler führt einen Typecheck durch.
ComplexAss		Bereich	TRUE	Der Assembler erlaubt komplexere Adressierungsarten
Symbols		Modul	FALSE	Sorgt dafür, daß im gelinktenProgramm die symbolischen Bezeichnernoch enthalten sind, so daß man bei Verwendung eines Assemblerdebuggers, der Symbole unterstützt, direktauf die Prozedur namen zugreifen kann.

Die Schalter können global, im Info Requester oder im Quelltext gesetzt oder gelöscht werden. Laufzeitchecks sollten erst nach längerer Erprobung ausgeschalten werden. Dann aber sollte man sie abschalten, da sie das Programm verlangsamen und länger machen. Auch diese vordefinierten Schalter lassen sich zur bedingten Compilierung einsetzen. Die Kurzform der Schalter sollte man eigentlich nicht mehr benutzen, da sie zu wenig Ausagekraft haben. Will man Sie dennoch verwenden, gilt:

```
(* $R- *)       <=> $$RangeChk:=FALSE
(* $R+ *)       <=> $$RangeChk:=TRUE
(* $R= *)       <=> $$RangeChk:=OLD
```

Wie Sie Schalter im Text verändern können, haben Sie im letzten Abschnitt gesehen. Im Textinforequester funktioniert dies folgendermaßen: Begeben Sie sich in den Textinforequester *A* +

F5 ⇔ INFO, und klicken Sie auf Switches. Daraufhin öffnet sich folgender Requester:

In den dargestellten Schaltern finden Sie die meisten der oben aufgeführten Switches. Die strukturbezogenen sind nicht enthalten, da von hier aus nur globale Einstellungen gemacht werden können. Ist ein Schalter eingedrückt, gilt er als eingeschaltet. Wird ein hier gesetzter Schalter lokal neu definiert, gilt diese Einstellung. Folglich gilt Lokal vor Global.

Folgende Schalter entsprechen aus Platzgründen nicht den Bezeichnungen der Compilerswitches, die im Quelltext angegeben werden können:

- Der Schalter **EXECUTBL (1)** gibt an, ob ein Objektfile für ein normales Modul erzeugt werden soll.

  Standardeinstellung ist TRUE. Im Falle, daß man ein Modul nur in einer Library verwenden will, kann man durch Ausschalten verhindern, daß ein zweites unnötiges Objektfile erzeugt wird.

- **STR.ZERO (2)** entspricht dem zuvor beschriebenen Schalter „StrZeroChk".

- **Allign 4 (3)** entspricht dem zuvor beschriebenen Schalter „LongAllign".

- **RetChk. (4)** entspricht dem zuvor beschriebenen Schalter „ReturnChk".

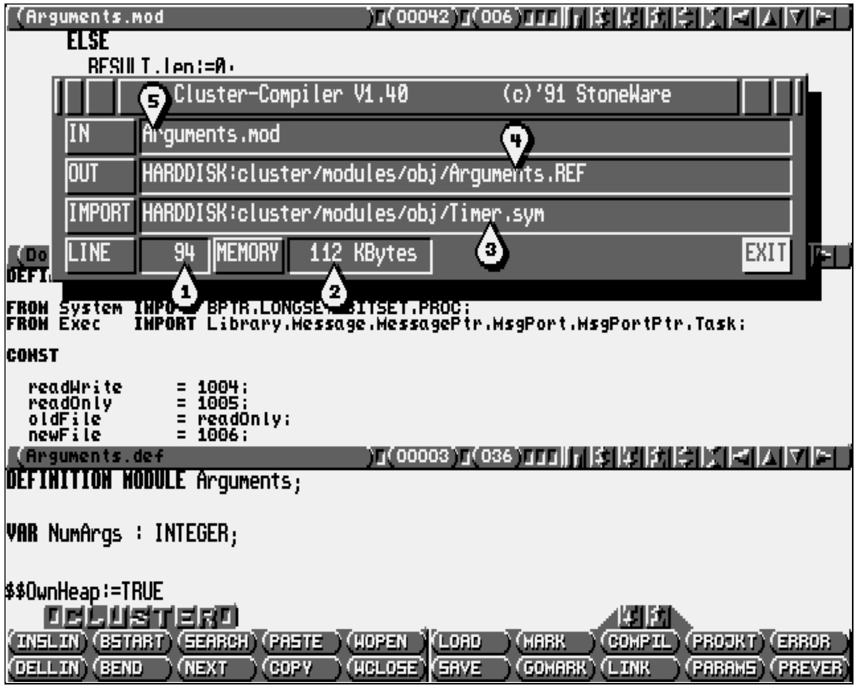
Der Switch **(5)** ist ein Beispiel für einen selbstdefinierten Schalter. Selbstdefinierte Schalter können außer Modulglobal auch noch Projektglobal gesetzt werden (siehe Abschnitt 2.6). Der kleine Schalter daneben **(6)** gibt dabei an, ob die Projektglobale Einstellung übernommen werden soll (Schalter eingedrückt), oder ob die in diesem Requester gemachten Einstellung beachtet werden soll (Schalter ausgeschaltet).

Der Schalter Switches läßt sich übigens nur anwählen, wenn in Textinfo - PATHS ein bereits definiertes Projekt eingetragen ist. Siehe Kapitel 2.3.8.1 Seite 19

2.5.2 Compilieren eines Programms

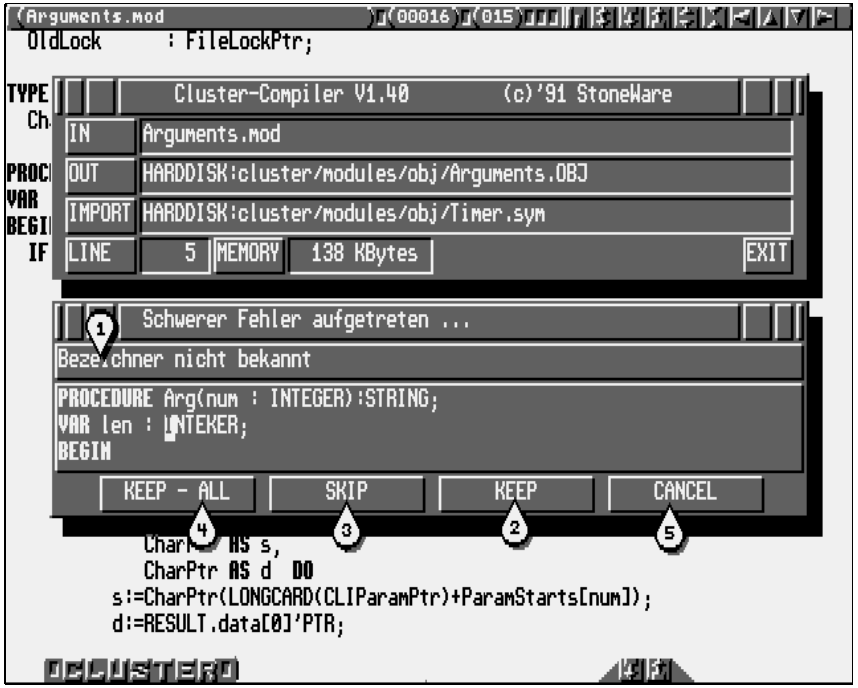
2.5.2.1 Compilieren von der EU aus

Um einen Text zu compilieren, müssen Sie sein Fenster aktivieren und **F8** ⇔ (COMPIL) drücken. Daraufhin erscheint ein Requester, und der Compiler beginnt den Text zu übersetzen.



Beim Compilieren wird die ungefähre aktuelle Position im Text **(1)**, der noch freie Speicher **(2)**, die momentan eingelesenen Module **(3)**, die erzeugte Datei **(4)** und den Namen des gerade übersetzten Textes **(5)** anzeigt.

2.5.3 Fehlerbehandlung und Fehlermeldungen



Wird bei der Übersetzung ein Fehler entdeckt, erscheint ein Requester mit einer Meldung, die den Fehler genauer spezifiziert **(1)**.

Nun kann man den Fehler in eine Fehlerliste aufnehmen **KEEP (2)**, um ihn später wiederzufinden oder man kann ihn einfach überspringen **SKIP (3)**, was besonders bei Folgefehlern praktisch ist. Bei beiden Aktionen wird die Compilierung danach fortgesetzt. Will man, daß alle auftretenden Fehler in die Liste aufgenommen werden, geht man auf **KEEP-ALL (4)**. Man kann den Übersetzungsvorgang aber auch mit **CANCEL (5)** abbre-

chen (Dies ist zu empfehlen, wenn sich der Compiler nach einem Fehler verheddert).

Ist die Übersetzung abgeschlossen, können Sie nun, vorausgesetzt Sie haben sich Fehler mit Keep gemerkt, diese mit **F10** ⇔ (ERROR) anspringen. Mit **SHIFT** + **F10** ⇔ (PREVER) können Sie wieder zum vorhergehenden Fehler zurückkehren.

Dabei wird jeweils die entsprechende Stelle im Text angesprungen und die Fehlermeldung noch einmal ausgegeben.

Nach einem Durchlauf können Sie die Liste beliebig oft zurückgehen und danach wieder von vorne beginnen.

Alle Fehlermelungen sind im Einzelnen im Anhang aufgeführt und erläutert. Ein Hinweis: Alle Meldungen, die einen „(Compiler-Fehler)“ oder ähnliches beinhalten, sind nicht durch Sie verursacht, sondern es ist ein Fehler des Compilers. Dies sollte eigentlich nicht mehr vorkommen, kann allerdings die Folge eines anderen Fehlers sein.

Sollte dies doch einmal grundlos passieren, dann melden Sie uns bitte dies und schicken uns den Quelltext, bei dem der Fehler auftrat.

2.5.4 Der Symbol-Cache

Die **EU** ist mit einem Symbol-Cache ausgestattet. D. h. Symboldateien, die durch Import beim Compilieren geladen wurden, werden im Speicher gehalten, so daß sie bei der Übersetzung des nächsten Textes nicht mehr geladen werden müssen. Der Vorteil gegenüber in die RAM-Disk kopierten Modulen ist, daß weniger Speicher verbraucht wird und eine höhere Compiliergeschwindigkeit erreicht wird.

Compiliert man allerdings ein Definitionsmodul, dessen Symboldatei sich bereits im Cache befindet, wird dieses automatisch dort entfernt, damit keine Versionskonflikte entstehen. Beim nächsten Importieren wird die Symboldatei wieder neu geladen.

Das Cache kommt vor allem Computerbesitzern mit viel Speicher, aber ohne Festplatte zugute. Hat man allerdings wenig Speicher, sollte man den Cache im Global-Requester ausschalten.

Normalerweise wird vor jedem Zugriff auf den Cache kontrolliert, ob die Version des Moduls im Cache, mit der auf Disk/Platte übereinstimmt. Man kann diesen Check aber auch abschalten, um die Zahl der Diskettenzugriffe zu verringern. Voraussetzung ist, daß während der Arbeit mit der **EU**, von der Shell oder Workbench aus, keine andere Version eines 'gecachten' Moduls in das entsprechende Verzeichnis kopiert wird, oder daß durch eine zweite **EU** dasselbe Modul compiliert wird.

2.6 Das Projekt-System

Da der Compiler während der Übersetzung diverse Dateien laden und das Compilat ja auch wieder abspeichern muß, es aber sehr aufwendig wäre, jedesmal für jede Datei einen kompletten Pfad anzugeben, sind wir dazu übergegangen für jedes Programmprojekt ein Projektverzeichnis anzulegen.

Dieses enthält die Verzeichnisse „`TXT`" für die Quelltexte und „`OBJ`" für die Compilate, sowie bei Bedarf „`BAK`" für Sicherungskopien.

Damit ein Programm übersetzt werden kann, muß es zuvor in das `TXT`-Verzeichnis eines solchen Projektverzeichnisses abgespeichert werden.

Um Ihnen die Erzeugung des Projektverzeichnises zu erleichtern, existiert ein kleines Programm namens „`Project`", das Sie am besten in das `c:`-Verzeichnis ihrer Bootdiskette kopieren. Sie finden es nach der Installation in „Cluster:ClusterTools". Sie benutzen es, indem Sie es folgendermaßen aufrufen:

`Project ,,Pfad'' [`*-i*`]`

z. B. `Project df1:Work`

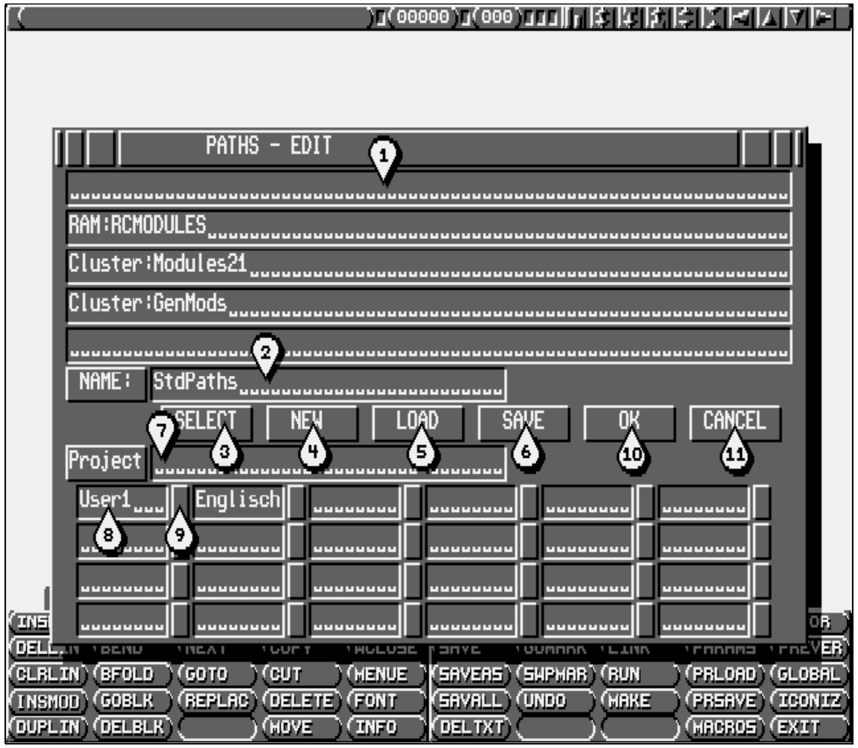
richtet ein Projektverzeichnis names Work im Verzeichnis `df1:` ein. Gibt man noch den zweiten Parameter '`-i`' an, werden für die Verzeichnisse keine Icons erzeugt.

Das Verzeichnis haben Sie nun, doch woher weiß der Compiler, welches das richtige ist?

Zuerst versucht er das Verzeichnis, aus dem der Text stammt. Sonst benutzt er eine Liste von Pfaden, wobei er die Liste von oben nach unten durchsucht. So können Sie ruhig ein Modul in zwei Verzeichnissen haben und können sicher sein, daß immer zuerst im ersten aufgeführten Modulverzeichnis gesucht wird. Außerdem haben Sie damit die Möglichkeit, Module zu importieren, die nicht in ihrem augenblicklichen Projektverzeichnis liegen.

2.6.1 Definition von Pfadliste und Userswitches in der EU

Um solch eine Pfadliste zu definieren, dient der PATHS-EDIT-Requester, den Sie über **F9** ⇔ `PROJKT` erreichen.



In den fünf Stringgadgets **(1)** kann man die Pfade der Projektverzeichnisse eingeben, in denen er von oben nach unten suchen soll.

Zur Auswahl der Pfade besteht auch hier die Möglichkeit, über **Help** einen Filerequester zu erhalten.

Möchte man eine neue Pfadliste erzeugen, drücke man **NEW (4)**, gebe einen neuen Namen für die Liste in **NAME: (2)**, sowie die gewünschten Pfade ein, danach wähle man **SAVE (6)** an.

Möchte man eine bestimmte Liste ändern, kann man sie über **SELECT (3)** auswählen, verändern und wieder abspeichern.

Um die Liste unter Select auf den neuesten Stand zu bringen, kann man auf **LOAD (5)** gehen, worauf das Verzeichnis „`Cluster:Projects`" neu geladen wird. Damit werden auch Projekte, die erzeugt, aber nicht abgespeichert wurden, wieder gelöscht.

Zu einer Pfadliste gehört auch ein Eintrag, der das Programm angibt, das gelinkt oder „gemaked" werden soll, falls von einem Modul, das kein Hauptmodul ist und keinen Projekteintrag im Textinforequester besitzt, eine dieser Funktionen aufgerufen wird. Dieser läßt sich im Feld **Project (7)** eintragen. Ist bei einem Text, der diese Pfadliste benutzt, kein Projekt eingetragen, wird das hier eingetragene benutzt.

Außerdem lassen sich zu jeder Pfadliste eigene Compilerswitches definieren. Erst nachdem man einen Switch hier definiert hat, kann man ihn benutzen. Dazu dienen die Stringgadgets **(8)**. Der kleine Schalter daneben **(9)** gibt die projektglobale Einstellung dieses Schalters an. Ist in TextInfo-Switches bei dem entsprechenden Schalter der kleine Schalter daneben eingedrückt, wird die im PATHS-EDIT-Requester gemachte Einstellung für diesen Text übernommen.

Neue Texte erhalten automatisch die Pfadliste „`StdProject`".

Die Compilate werden stets in das Projektverzeichnis gelegt, aus dem der Quelltext stammt, hat dieser noch kein Verzeichnis, wird er in das Projekt „`Cluster:Work`" abgelegt.

2.6.2 Definition von Pfadliste und Userswitches bei der Verwendung von Fremdeditoren

Bei der Verwendung eines anderen Editors als dem, der in die **EU** integriert ist, entsteht natürlich das Problem, wie definiert man hier benutzerdefinierte Compilerswitches oder eine Pfadliste, nach der die Module importiert werden sollen ?

Ganz einfach, ein solches Projektfile läßt sich mit jedem Editor leicht erzeugen, da es sich dabei um eine Textdatei handelt. Diese hat folgenden Aufbau: Jede Zeile beginnt mit einem Schlüsselwort, nach dem der Rest der Zeile entsprechend interpretiert wird.
Ein Beispiel für ein solches File:

```
NAME <ProjectNamen>
PATH <Pfad1>
PATH <Pfad2>
SWITCH T <Name>
SWITCH F <Name>
```

- Nach dem Schlüsselwort `NAME` folgt, durch ein Space[18] getrennt, der Name dieses Projectes. Dieser Eintrag kann auch wegfallen.

- Nach `PATH` folgt, durch ein Space getrennt, der Pfad zu dem gewünschten Modulverzeichnis, z. B.: `Cluster:Modules`. Es können bis zu fünf Einträge dieser Sorte gemacht werden.

- Nach `SWITCH` folgt, durch ein Space getrennt, ein „`T`“ falls dieser Switch global eingeschaltet sein soll, sonst ein „`F`“. Darauf folgt, wieder durch ein Space getrennt, der Name des neuen Switches. Von dieser Art Eintrag können insgesammt 30 definiert werden.

[18] Leerzeichen

Bei allen Schlüsselwörtern ist es notwendig, daß sie komplett groß geschrieben werden. Eine so erstellte Datei muß dann in das Verzeichnis „`Cluster:Projects`“ gespeichert werden. Will man einem Text nun diese Pfadliste zuordnen, kann dies mit Hilfe des ARexx-Befehls „`SetProject`“ geschehen. Mehr dazu unter 2.10, auf Seite 62.

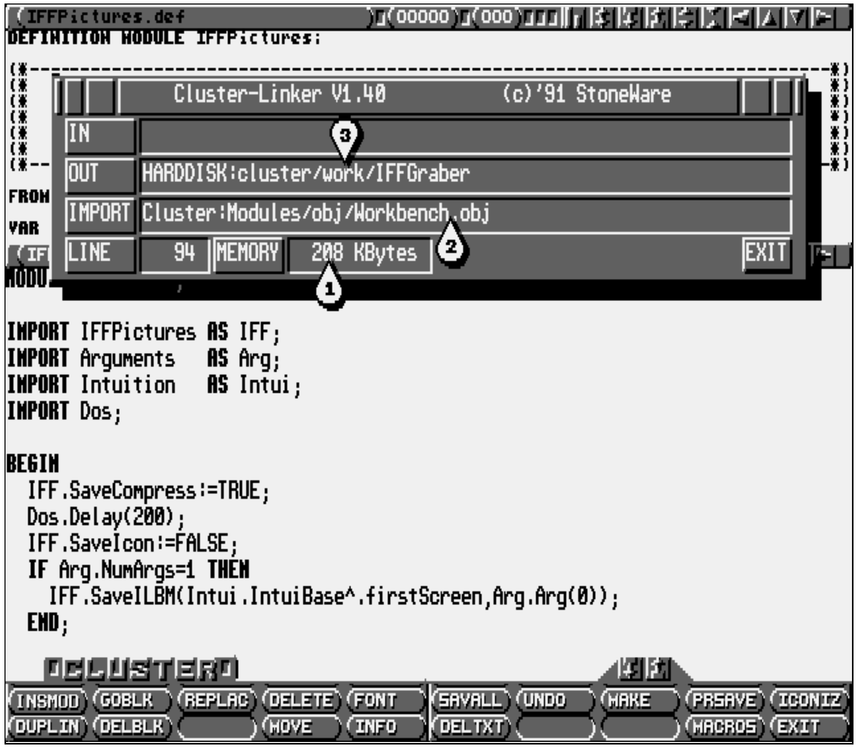
2.7 Der Linker

Der Linker verbindet die einzeln übersetzten Module zum fertigen Programm, das Sie direkt von der SHELL oder von der Workbench aus aufrufen können.

Das besondere an diesem Linker ist, daß er selektiv linkt, d. h. er klebt nicht einfach alle Module aneinander wie viele andere Linker, sondern es werden nur die Teile eines Moduls eingebunden, die mindestens einmal benutzt werden. Dadurch ist es möglich, lange Biblioteksmodule zu schreiben und dennoch kurze Programme zu erhalten.

2.7.1 Linken eines Programms

2.7.1.1 Linken von der EU aus



Zum Linken eines Programmes drücken Sie einfach **SHIFT** + **F8** ⇔ (LINK). Daraufhin erscheint ein Requester, er zeigt den noch freien Speicher **(1)**, das Modul, das gerade eingebunden wird **(2)** und den Namen des fertigen Programm **(3)**.

Befanden Sie sich beim Aufruf des Linkers gerade im Quelltext des bereits compilierten Hauptmoduls, so wird dieses nun gelinkt.

Waren Sie gerade in einem Definitions- oder einem Implementationsmodul, öffnet sich ein Stringrequester, in den Sie den Namen des Programmes, das gelinkt werden soll, eintragen müssen.

Wollen Sie diesen nicht bei jedem Linken neu eingeben, können Sie ihr Hauptmodul im Path-Select-Requester im Feld Projekt eintragen, und es so zum Hauptprojekt machen, oder als Projekt dieses Textes (INFO) eintragen.

Rufen Sie nun den Linker auf, linkt er automatisch das eingetragene Programm, sofern Sie nicht gerade ein anderes Hauptmodul bearbeiten.

2.7.2 Fehlermeldungen des Linkers

Tritt beim Linken ein Fehler auf, so bricht der Linker den Linkvorgang ab und gibt eine Meldung aus.

Eine genaue Beschreibung der Fehlermeldungen des Linkers finden Sie im Anhang.

2.8 Der Loader

Mit dem Loader hat man die Möglichkeit, ein Programm zu starten, ohne es vorher zu linken. Man könnte auch sagen, daß der Loader das Programm direkt in den Speicher linkt.

Möglicherweise werden Sie fragen, wozu das ganze gut sein soll, da man ja ein Programm linken kann und es dann nur noch aufrufen muß?

Der wohl größte Vorteil ist, daß man ein Programm direkt aus der **EU** starten kann. Zusätzlich werden bei Verwendung des Loaders Laufzeitfehler mit der Programmzeile, in der sie aufgetreten sind, ausgegeben.

Übrigens haben Sie meist die Möglichkeit, auch wenn der Editor bei einen Programmfehler abstürzen sollte, eventuell noch nicht gesicherte Texte zu speichern.

2.8.1 In der EU

In der **EU** ist der Loader mit einer Run-Funktion gekoppelt. Das heißt, wenn Sie **Alt** + **F8** ⇔ (`RUN`) drücken, wird der aktuelle Text compiliert, sofern er das noch nicht war, dann in den Speicher „gelinkt" und schließlich gestartet.

Zuvor findet jedoch noch eine Sicherheitsabfrage statt, ob noch nicht abgespeicherte Texte zuerst gesichert werden sollen.

Wichtig:
Erwartet Ihr Programm beim Start Parameter, müssen Sie diese angeben. Dies geschieht, indem Sie **SHIFT** + **F9** ⇔ (`PARAMS`) drücken, worauf sich ein Requester öffnet, in den Sie die Parameter nacheinander eintragen können.

Nach der Ausführung wird nachgefragt, ob Sie das Programm noch einmal starten wollen; wenn nicht, wird es wieder aus dem

Speicher entfernt.

Tritt bei der Ausführung ein Laufzeitfehler auf, so wird das Programm abgebrochen und der Laufzeitfehler nebst Programmzeile im Sourcecode, sowie das Modul, in dem er auftrat, angezeigt.

2.8.2 Laufzeitfehler

Nun ist es doch an der Zeit, einmal zu erklären, was eigentlich Laufzeitfehler, auch Runtime Errors genannt, sind.

Laufzeitfehler sind Fehler, die während der Übersetzung des Programmtextes nicht entdeckt werden können, da Sie nicht aus Schreib- oder Konstruktionsfehlern herrühren, sondern i. d. R. immer Denkfehlern des Programmieres zuzuschreiben sind.
So wird z.B.

```
i:=j DIV 0
```

bereits bei der Übersetzung als Division durch Null erkannt und gemeldet. Dagegen wird

```
i:=j DIV k
```

anstandslos übersetzt, da in diesem Fall `k` eine Variable ist. Der Compiler nimmt im Normalfall an, daß die Variable nicht Null ist, da der Programmierer sich darum schon kümmern muß. Wird `k` jedoch im Laufe des Programms Null und man teilt durch `k`, führt das augenblicklich zu einem Laufzeitfehler.

Dies war nur ein Beispiel für einen Laufzeitfehler, meist werden Sie dadurch verursacht, daß Grenzen oder Zahlenbereiche z. B. in einer Schleife überschritten werden.

Aber auch höchst ungebetene rot lächelnde Inder führen zu einem Laufzeitfehler, sofern sie abgefangen werden konnten; ansonsten müssen Sie wahrscheinlich neu booten.

Eine komplette Aufführung aller Laufzeitfehler finden Sie im Anhang mit Beschreibung und Hilfen zur Fehlersuche.

2.9 Das Make

Nehmen wir einmal an, Sie haben in einem großen Projekt ein weit unten gelegenes Definitionsmodul, das womöglich noch von anderen Definitionsmodulen importiert wird. Wenn Sie dieses ändern, so müßten Sie jetzt die große Compilierorgie starten und alle Module, die das veränderte Modul importieren, bei Definitionsmodulen auch noch deren Implementationsteile, neu compilieren, um dann beim Linken zu erfahren, daß Sie dabei ein Modul vergessen haben und alles noch einmal machen dürfen.

Wenn Sie das einmal gemacht haben, werden Sie bestimmt nichts sehnlicher wünschen, als daß so etwas Ihnen nicht noch einmal passiert.

Damit Sie diese Erfahrung nicht auch nur einmal machen müssen, besitzt **Cluster** ein sogenanntes Make.

Voraussetzung ist jedoch, daß alle beteiligten Texte im Cluster-Format vorliegen oder im ASCII-Format mit eingeschalteten Keywords, so daß das File mit einem kleinen Header versehen ist, in dem die wichtigsten Informationen enthalten sind.

Das Make untersucht erst die Abhängigkeiten der Module untereinander, um Sie danach, wenn nötig, in der richtigen Reihenfolge zu compilieren.

Da diese Abhängigkeitsfeststellung relativ viel Zeit in Anspruch nimmt, wird dies nur einmal durchgeführt und die ermittelten Daten als Datei, mit angehängtem „`.MKE`“, im „`OBJ`“ des Projektverzeichnis des Hauptmoduls abgelegt.

Die Prüfung und die Erzeugung dieser Makedatei erfolgt durch das Programm „`CreateMake`“. Es wurde extra vom eigentlichen Make getrennt, da man diese Prozedur nur beim ersten Mal durchführen muß, bei jedem weiteren Make nicht mehr.

Sie müssen diese aber jedesmal, wenn Sie in einem Modul ein neues Modul importieren, wiederholen, da ja im Bedarfsfall auch dieses neue Modul gefunden und compiliert werden muß.

Eine Ausnahme bilden Module, die in einem anderen Projekt-Verzeichnis liegen, da das Make in einem solchen Fall davon ausgeht, daß die importierten fremden Module mit Sicherheit vor dem importierenden compiliert wurden.

Ein Tip: Die Makedatei ist ein ganz gewöhnliches Textfile, in dem aufgeführt ist, welches Modul welche anderen importiert. Wenn Sie also nach einem Createmake doch noch ein neues Modul in einem anderen importieren, müssen Sie nicht unbedingt ein neues „`CreateMake`“ ausführen.

Laden Sie einfach die Makedatei in den Editor, und tragen sie hinter der Importliste des bearbeiteten Moduls den Namen des neuen Moduls ein.

Sie sehen also, man kann eine Makedatei auch von Hand erzeugen oder verändern.

Nun zur Bedienung, durch Drücken von **Ctrl** + **F8** ⇔ (`MAKE`) erscheint ein Requester mit mehreren Gadgets.

Wenn Sie nur 1MB oder gar noch weniger freien Speicher haben, ist es ratsam darauf zu achten, daß Sie keinen zu großen Textspeicher eingestellt haben.

Führen Sie in diesem Projekt zum ersten mal ein Make aus, oder haben Sie eine neue Modulabhängigkeit erzeugt (s. o.), so wählen Sie zuerst **CREATE** an, um eine neue Makedatei zu erzeugen. Dabei wird als Hauptmodul – sofern Sie diese Funktion von einem anderen Modul, als dem Hauptmodul aus anwählen – das unter Projekt eingetragene angenommen[19]. Ist keines gesetzt, so erscheint ein Requester, in den man den Projektnamen (gleichbedeutend mit dem Namen des Hauptmoduls) eingeben muß. Als

[19]Zuerst wird dabei der Projekteintrag im Textinforequester dieses Textes, dann der im Path-Edit-Requester gemachte berücksichtigt.

Pfad zum Projektverzeichnis wird dabei der Pfad des Textes angenommen, von dem aus Make gestartet wurde.

Danach brauchen Sie nur noch Make anzuwählen und alle Texte werden geprüft. Wenn einer neu compiliert werden muß, wird er in das aktuelle Textfenster geladen (keine Angst, der darin befindliche Text wird gespeichert und am Ende wieder eingeladen) und übersetzt.

Schließlich müssen Sie nur noch das Hauptprogramm linken.

Wählt man statt **MAKE REMAKE**, dann werden alle Module, die vom Hauptmodul direkt oder indirekt importiert werden, in der richtigen Reihenfolge neu übersetzt, ohne Rücksicht, ob sie schon compiliert waren. Dies ist vor allem praktisch, wenn man in PATHS-EDIT einen projektglobalen Schalter umgeschaltet hat und um dadurch alle beteiligten Module mit diesem Schalter neu zu übersetzen. Ist man sich sicher, daß solch ein Schalter nur Implementationsmodule betrifft, kann man auch **REMAKE IMPL.** anwählen, daraufhin werden alle abhängigen Implementationsmodule compiliert.

2.10 ARexx-Ansteuerung von Compiler, Linker, Loader und Make

Um Ihnen auch die Möglichkeit zu geben, mit Ihrem Lieblingseditor zu arbeiten, liegt dem Paket eine Version von Compiler, Linker, Loader und Make bei, die über ARexx ansteuerbar ist. Dieses Programm wird einmal gestartet, bleibt dann im Hintergrund im Speicher und wartet auf ARexx-Kommandos. Aufruf:

```
run RexxCluster
```

Danach können folgende Kommandos an den ARexx-Port `CLUSTER.REXX` übergeben werden:

Compile {`Filenamen Fehlerdatei [Fileposition]`}
Übersetzt die Datei `FileNamen`[20]. Eventuell möchte man, daß das File, das der Compiler übersetzen soll, an einer anderen Stelle liegt als der, die bei Filenamen angegeben ist, beispielsweise, der Editor speichert sie ins RAM oder in die PIPE, damit der Übersetzungsvorgang beschleunigt wird, und speichert den Text nur am Arbeitsende wieder an dem Platz, an dem er eigentlich liegen sollte. In einem solchen Fall kann man die wirkliche Position des Files mit `Fileposition` angeben. Daß man nicht gleich diesen Pfad als `Filenamen` angibt, hat den Zweck, daß der Compiler am `Filenamen` immmer noch erkennen kann, welche Module im selben Verzeichnis wie der zu übersetzende Text liegen. Eventuelle Fehler schreibt der Compiler in die Datei `FehlerDatei`. Diese Fehlerdatei besteht aus einzelnen Strings, die sich folgendermaßen aufbauen:

[20] Dabei muß der komplette Pfad angegeben werden.


```
FehlerMeldung Zeile Spalte
```

Diese Datei kann man danach leicht mittels eines ARexx-Skripts parsen.

Link {`Filenamen Fehlerdatei`}
Linkt die Datei `FileNamen` und schreibt eventuelle Fehler in die Datei `Fehlerdatei`.

Run {`Filenamen Fehlerdatei`}
Startet das Programm `FileNamen` und schreibt eventuelle Laufzeitfehler mit Positionsangabe in die Datei `Fehlerdatei`.

SetArgString {`Argumentstring`}
Hiermit kann eine Argumentzeile eingetragen werden, die bei Run dem Programm übergeben wird.

Create {`FileNamen Fehlerdatei`}
Erzeugt eine Makedatei für das Modul `FileNamen`, eventuelle Fehler werden in die Datei `Fehlerdatei` geschrieben.

Make {`FileNamen Fehlerdatei`}
Führt ein Make auf das Modul `FileNamen` aus, eventuelle Fehler werden in die Datei `Fehlerdatei` geschrieben.

ReMake {`FileNamen Fehlerdatei`}
Führt ein ReMake auf das Modul `FileNamen` aus, eventuelle Fehler werden in die Datei `Fehlerdatei` geschrieben.

ReMakeImpl { `FileNamen Fehlerdatei`}
Compiliert alle Implementationsmodule, die von dem Modul `FileNamen` abhängig sind. Eventuelle Fehler werden in die Datei `Fehlerdatei` geschrieben.

SetProject {`Projectnamen`}
Setzt das Project `Projectnamen` als Pfadliste, nach der die Modulverzeichnisse ausgewählt werden.

Quit
Beendet das Programm.

Wenn Sie das Make verwenden wollen, muß jeder Text noch einen kleinen Header erhalten, der als Platzhalter dient, damit das Make darin für den Makevorgang wichtige Daten ablegen kann. Der Header sieht folgendermaßen aus, und muß noch vor der ersten Programmzeile stehen:

```
|##########|
|#MAGIC   #|********
|#PROJECT #|"Editor"
|#PATHS   #|"StdProject"
|##########|
```

Für das Make notwendig sind nur die erste, die letzte und die Zeile mit dem Eintrag `MAGIC`. Die 8 Sterne hinter `MAGIC` sind absolut notwendig, da an diese Stelle das Make seine Daten ablegt. Die Zeilen `PROJECT` und `PATHS` können auch wegfallen, sie sind nur ein Vorschlag, sie könnten dazu dienen, von dem den Compiler aufrufenden ARexx-Script geparst zu werden. So gibt `PROJECT` an, welches das Hauptprogramm ist, das gelinkt oder gemaked werden soll, falls man den Linker oder das Make von einem anderen Modul als vom Hauptmodul aufruft. `PATHS` gibt den Namen der PfadListe an, die man dann per ARexx mit `SetProject` an den Compiler übergeben kann, bevor man den Compiler startet. Die mitgelieferten ARexx-Scripte berücksichtigen diese Schlüsselwörter.

Falls Ihr Editor keine ARexx-Macros ausführen kann, besteht auch die Möglichkeit, die mitgelierferten Skripte gleichen Namens

aufzurufen, die sich nach der Installation in `Cluster:` befinden. Beispiele für solche ARexx-Skripte finden Sie im Anhang.

2.11 Der Debugger

An dieser Stelle soll einmal die endgültige Anleitung des in Vorbereitung befindlichen Debuggers stehen. Da wir es aber noch nicht geschafft haben, ihn vollständig fertigzustellen, wir Ihnen die Vorversion aber dennoch nicht vorenthalten wollten, steht hier eine kurze Anleitung der vorläufigen Cli-Version des Debuggers.

Bevor man ein Modul mit dem Debugger bearbeiten kann, muß man es, sowie das Hauptmodul des dazugehörigen Programms, mit dem Switch `Debug:=TRUE` compilieren. Danach ruft man den Debugger mit dem Programmnamen als Parameter auf.

Hat er alle Module importiert, kann man mit dem Durchsteppen beginnen. Trifft man dabei auf eine Prozedur, wird in deren BEGIN-Teil das Tracen fortgesetzt. Dies funktioniert auch in Prozeduren aus anderen Modulen, soweit diese mit `Debug:=TRUE` compiliert wurden.

```
Debugger [!Projectnamen] Programmnamen
```

Dabei ist `!Projectnamen` der Name des Projektes, nach dem der Debugger die Module, die sich nicht im aktuellen Verzeichnis befinden, auswählen soll. Das „!“ vor dem Namen ist notwendig, damit der Debugger den Projektnamen nicht für das Programm hält.

Folgende Kommandos stehen dazu zur Verfügung:

`s, Return`	:	Führe die nächste Zeile aus.
`o`	:	Führe die nächste Zeile aus, überspringe dabei Prozeduren.
`g`	:	Führe das Programm bis zum Ende aus oder bis zu einem Breakpoint.
`g line`	:	Führe das Programm bis zu Zeile `line` aus.
`b+ line`	:	Setzte Breakpoint in Zeile `line`.
`b- line`	:	Lösche Breakpoint in Zeile `line`.
`c+`	:	Schaltet automatisches Löschen des Bildschirms vor jeder Ausgabe ein.
`c-`	:	Schaltet automatisches Löschen des Bildschirms vor jeder Ausgabe aus.
`ct`	:	Spezieller Clearmodus vor Listingausgabe, dabei wird bei jedem Schritt der Bildschirm gelöscht und die Programmzeilen um die augenblickliche Position herum neu ausgegeben.
`t`	:	Zeigt die Zeilen um die aktuelle Position an.
`t n m`	:	Zeigt die Zeilen von `n` bis `m` an.
`t+`	:	Schaltet automatische Listinganzeige an.

`t+ n`	:	Schaltet automatische Listingsanzeige an, wobei `n` Zeilen benutzt werden. Beim Autolist wird von da an diese Anzahl verwendet.
`t-`	:	Schaltet Autolist wieder aus.
`? xxx`	:	Zeigt den Inhalt der im nächstgelegenen Sichtbarkeitsbereich liegenden Variablen `xxx` an.
`? n xxx`	:	Zeigt den Inhalt der Variablen `xxx` in der Aktionsstufe `n` an.
`w xxx`	:	Bewirkt, daß die Variable `xxx` bei jedem Schritt neu ausgelesen und angezeigt wird. Ausschalten wieder mit `w xxx`.
`x`	:	Bricht das Programm ab, indem in den Close-Teil gesprungen wird.
`*`	:	Bricht das Programm ab, ohne daß der Close-Teil angesprungen wird. Empfehlenswert, wenn ein Programm sich im Close-Teil aufhängt.
`e`	:	Nach dem Auftreten einer Exception kann man hiermit den Exceptiontext erhalten.

Am empfehlenswertesten ist folgende Einstellung: Debugger mit einem Programm starten, dann „`+10`“, und darauf „`ct`“ eingeben.

Spielen Sie ein bißchen damit herum, aber speichern Sie vorher alle Texte. Wir sind sicher, daß der Debugger auch in dieser Version schon sehr nützlich sein kann.

Wir sind selbstverständlich für Anregungen sowie Fehlermeldung, den Debugger betreffend, dankbar.

2.12 Übersicht aller Editorfunktionen

2.12.1 Cursortasten

$\boxed{\leftarrow}$	:	ein Zeichen nach links
$\boxed{\leftarrow}$ + **SHIFT**	:	auf Zeilenanfang springen
$\boxed{\leftarrow}$ + **Alt**	:	ein Wort nach links
$\boxed{\rightarrow}$	:	ein Zeichen nach rechts
$\boxed{\rightarrow}$ + **SHIFT**	:	ans Ende der Zeile springen
$\boxed{\rightarrow}$ + **Alt**	:	ein Wort nach rechts
$\boxed{\uparrow}$	:	eine Zeile hoch
$\boxed{\uparrow}$ + **SHIFT**	:	an den Anfang der Seite springen; befindet sich der Cursor schon dort, springt er an den Anfang des Textes
$\boxed{\uparrow}$ + **Alt**	:	eine Seite nach oben
$\boxed{\uparrow}$ + **Ctrl**	:	ein Fenster nach oben
$\boxed{\downarrow}$	:	eine Zeile runter
$\boxed{\downarrow}$ + **SHIFT**	:	ans Ende der Seite, befindet man sich schon dort, an das Ende des Textes
$\boxed{\downarrow}$ + **Alt**	:	eine Seite nach unten
$\boxed{\downarrow}$ + **Ctrl**	:	ein Fenster nach unten

2.12.2 Steuertasten

BACKSPACE	:	löscht das Zeichen links vom Cursor
Del	:	löscht das Zeichen unter dem Cursor
Del + **SHIFT**	:	ein Zeichen einfügen, in Stringgadgets ganze Zeile löschen
Del + **Alt**	:	zwischen Insert/Overwrite-Modus wechseln
RETURN	:	an nächsten Zeilenanfang springen
RETURN + **SHIFT**	:	zerlegt eine Zeile oder springt an den Anfang der nächsten Zeile, je nach dem, welcher Returnmodus eingestellt worden ist
RETURN + **Alt**	:	zwei Zeilen aneinanderhängen
TAB	:	ein Tabulator nach rechts
TAB + **SHIFT**	:	ein Tabulator nach links
TAB + **Alt**	:	Tabulatormodus wechseln
TAB + **Ctrl**	:	Tababstand ändern
Esc	:	Requester verlassen

Help	:	Öffnet in manchen Stringgadgets einen File-Requerster, um einen Pfad auszusuchen
Help + **Alt**	:	Setzt eine Marke an der Cursorposition und sucht das Wort, auf dem man sich gerade befand, vom Textanfang aus
Ctrl + Taste	:	Macro aufnehmen
A + Taste	:	Macro aufrufen

2.12.3 Funktionstasten und Menüfunktionen

F1	⇔	`INSLIN`	: Zeile einfügen
F1 + **SHIFT**	⇔	`DELLIN`	: Zeile entfernen
F1 + **Alt**	⇔	`CLRLIN`	: Zeile löschen
F1 + **Ctrl**	⇔	`INSMOD`	: Wechselt zwischen Insert- und Overwrite-Modus
F1 + *A*	⇔	`DUPLIN`	: Verdoppelt eine Zeile

F2	⇔	BSTART	: Blockanfang setzen
F2 + **SHIFT**	⇔	BEND	: Blockende setzen
F2 + **Alt**	⇔	BFOLD	: Klappt einen markierten Textbereich ein
F2 + **Ctrl**	⇔	GOBLK	: Springe an den Blockanfang, ist man schon dort, dann springt man an das Blockende
F2 + *A*	⇔	DELBLK	: lösche Blockdefinition
F3	⇔	SEARCH	: auf ein bestimmtes Wort springen
F3 + **SHIFT**	⇔	NEXT	: nach dem nächsten Auftreten des mit F3 gesuchten Wortes suchen
F3 + **Alt**	⇔	GOTO	: auf eine Zeile durch Angabe der Zeilennummer springen
F3 + **Ctrl**	⇔	REPLAC	: Suchen und Ersetzen
F3 + *A*	⇔		: für Updates reserviert

F4	⇔	PASTE	: Pufferinhalt einfügen
F4 + **SHIFT**	⇔	COPY	: kopiert den Block in den Puffer
F4 + **Alt**	⇔	CUT	: schneidet den Block aus und kopiert ihn in den Puffer
F4 + **Ctrl**	⇔	DELETE	: löscht den Block
F4 + *A*	⇔	MOVE	: schneidet den Block aus und fügt ihn an der Cursorposition ein
F5	⇔	WOPEN	: gibt dem aktuellen Fenster alle verfügbaren Zeilen
F5 + **SHIFT**	⇔	WCLOSE	: fährt ein Fenster in sich zusammen
F5 + **Alt**	⇔	MENUE	: fährt das Menü ein/aus
F5 + **Ctrl**	⇔	FONT	: schaltet zwischen dem großen und dem kleinen Zeichensatz um
F5 + *A*	⇔	INFO	: öffnet den Textinforequester

F6	⇔	LOAD	: öffnet den Filerequester
F6 + **SHIFT**	⇔	SAVE	: sichert den aktuellen Text
F6 + **Alt**	⇔	SAVEAS	: sichert den Text unter einem anderen Namen
F6 + **Ctrl**	⇔	SAVALL	: sichert alle Texte nach Abfrage
F6 + *A*	⇔	DELTXT	: entfernt ein Fenster
F7	⇔	MARK	: setzt eine Marke
F7 + **SHIFT**	⇔	GOMARK	: springt auf die Marke
F7 + **Alt**	⇔	SWPMAR	: vertauscht Marke mit dem Cursor
F7 + **Ctrl**	⇔	UNDO	: macht Veränderungen in einer Zeile rückgängig
F7 + *A*	⇔		: für Updates reserviert
F8	⇔	COMPIL	: startet den Compiler
F8 + **SHIFT**	⇔	LINK	: startet den Linker
F8 + **Alt**	⇔	RUN	: startet den Loader
F8 + **Ctrl**	⇔	MAKE	: öffnet den Makerequester
F8 + *A*	⇔		: für Updates reserviert

F9	⇔	PROJKT	: öffnet den Paths-Edit-Requester
F9 + **SHIFT**	⇔	PARAMS	: Parameter für den Loaderstart
F9 + **Alt**	⇔	PRLOAD	: Lädt einen gespeicherten OldState
F9 + **Ctrl**	⇔	PRSAVE	: speichert den momentanen Zustand des Editors als OldState.
F9 + *A*	⇔	MACROS	: Lädt/Speichert eine Macrobelegung
F10	⇔	ERROR	: springt auf den nächsten Fehler
F10 + **SHIFT**	⇔	PREVER	: springt zum vorhergehenden Fehler
F10 + **Alt**	⇔	GLOBAL	: öffnet den Voreinstellungsrequester
F10 + **Ctrl**	⇔	ICONIZ	: verwandelt den Editor zu einem Icon
F10 + *A*	⇔	EXIT	: beendet die **EU**

2.13 Mitgelieferte Hilfsprogramme

2.13.1 Cprint

Gibt einen Clustertext auf dem Drucker aus, Schlüsselwörter werden dabei fett gedruckt.
Aufruf:
`Cprint` *textnamen*

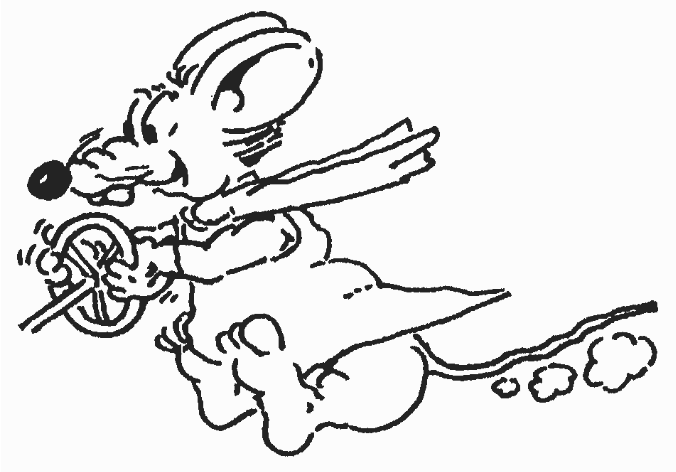
2.13.2 CreateImport

Erzeugt ein Modul, das alle Module eines Projekt-Verzeichnisses importiert, praktisch wenn man ein ganzes Projekt 'maken' will.
Aufruf:
`CreateImport` *name*
name : gibt dabei den Namen des ImportModuls an, das erzeugt werden soll.

Zum Aufruf in das Projekt-Verzeichnis gehen, das man später 'maken' will. Den Text des Importmoduls finden Sie dann im `TXT`-Verzeichnis.

Kapitel 3

Einführung in die Programmiersprache Cluster



3.1 Wer dieses Kapitel lesen sollte...

Anfänger! – Sie sind ein Anfänger? Sie verstehen vom Programmieren so viel wie Arnold Schwarzenegger vom Flötespielen? Sie sind guten Willens, doch dauernd wirft Ihnen jemand Knüppel in Form von unverständlichen Worten zwischen die Beine? Sie finden Informatiker doof und können nicht über Computerwitze lachen?

Dann sollten Sie dieses Kapitel entspannt und in Ruhe durchlesen, anstatt in eine Buchhandlung zu rennen, sich dort ehrfürchtig vor einer riesigen Bücherwand aufzubauen, sich schließlich für einen unaussprechlichen Titel zu entscheiden, das Ding nach Hause zu schleppen und bei der Lektüre desselben immer tiefer in den Sessel zu versinken, weil Sie das Gefühl nicht loswerden, daß der Autor jedes zweite Wort selbst nicht verstanden hat.

Also lassen Sie uns lieber zusammen ganz langsam an die Arbeit gehen und vorsichtig versuchen, den Kampf mit den vielen schrecklichen Fachwörtern souverän anzugehen und hinter uns zu bringen. Sie werden sehen – das geht!

Aber auch Personen, die schon etwas Programmiererfahrung gesammelt haben, sei es in Pascal, Modula 2 oder C sind herzlich in diesem Kapitel willkommen. Besonders Modula 2-Programmierer werden sich recht schnell heimisch fühlen, da **Cluster** eine von Modula 2 abgeleitete Programmiersprache ist. **Cluster** enthält einige nützliche Erweiterungen und Ergänzungen zu Modula 2, also sollte auch der erfahrene Programmierer dieses Kapitel zumindest überfliegen.

3.2 Was bietet dieses Kapitel?

Ziel dieses Kapitels soll es also sein, Ihnen einigermaßen leicht verdaulich elementare Grundlagen für die Programmierung Ihres

Amiga zu vermitteln.

Wenn Sie dieses Kapitel halbwegs konzentriert hinter sich gebracht haben, sollten Sie wissen:

- was ein Programm ist,
- was der Begriff „Algorithmus" bedeutet,
- wie ein **Cluster**-Programm aufgebaut ist,
- wie man kleine Problemstellungen zerlegt und in die Programmiersprache **Cluster** überträgt,
- wie man die Informationen, die dieses Handbuch enthält, für seine eigenen Programme nutzt.

Am besten und schnellsten lernt man bekanntlich etwas, wenn man es selber tut. Darum sollten Sie ab und zu Ihre Maschine anwerfen und die hier vorgeführten Beispiele eintippen, ausprobieren und verändern. Denken Sie immer daran, daß man aus Fehlern am meisten lernt. Ärgern Sie sich also nicht, wenn nicht gleich alles so klappt, wie Sie sich das vorstellen.

3.3 Was ist ein Computerprogramm?

So wie ein CD-Player die auf den CDs gespeicherten Informationen benutzt, um Töne zu erzeugen, so benötigt ein Computer ein Programm, um zu wissen, was zu tun ist[2]. Ein solches Programm besteht aus Anweisungen für den Computer: Das Programm enthält Befehle, der Computer führt sie aus. Er folgt dabei Schritt für

[2]Natürlich hinkt dieses Beispiel ziemlich, aber es viel uns gerade kein besseres ein.

Schritt genau dem Weg, den das Programm vorgibt. Ein Programm ist also eine Befehlsliste. Zum Verständnis stellen Sie sich einfach mal folgende Situation vor: Sie wachen nachts auf und haben einen tierischen Hunger auf eine Schneckensuppe[3]. Was ist zu tun?

1. Sie stehen auf.
2. Sie gehen in die Küche und schalten das Licht an.
3. Sie greifen mit einer Hand nach der Dose.
4. Ihre andere Hand schnappt sich den Dosenöffner.
5. Sie öffnen die Dose.
6. Den Inhalt geben Sie in einen Topf.
7. Sie erwärmen nun den Topf.
8. Sie essen die Suppe.

Diese acht Schritte stellen also ein Programm für das Problem „Heißhunger auf Schneckensuppe" dar.

Für die Lösung ein und desselben Problems gibt es meistens viele Möglichkeiten, die sich durch Eleganz und Länge oft erheblich unterscheiden. Die Art und Weise, wie ein Problem auf dem Computer zur Lösung umgesetzt wird, macht die Qualität eines Programmierers aus.

Natürlich kann der Computer nur schlecht eine Dose knacken, aber im Prinzip müssen Sie mit jedem Problem, welches Sie auf Ihrem Computer lösen wollen, so verfahren: Es in kleinstmögliche Teilprobleme zerlegen und somit den Weg zum Ziel beschreiben.

[3] Alle die nichtkulinarisch veranlagt sind, können nachfolgende Referenzen zur Schneckensuppe fast vollständig durch Gulaschsuppe ersetzen

3.3.1 Wofür sind Algorithmen gut?

Algorithmen: Nun doch diese fürchterlichen Fremdwörter, denken Sie jetzt vielleicht. Aber alles halb so wild, Algorithmen sind gar nicht so übel.

Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift[4] für ein Problem, die so präzise ist, daß sie von einem Computer durchgeführt werden kann[5]. Ist also ein Algorithmus praktisch ein Programm und umgekehrt, sind nicht die Begriffe Programm und Algorithmus demzufolge identisch?

Beinahe – ein Programm ist die Formulierung eines Algorithmus in einer bestimmten Programmiersprache, bei uns in **Cluster**. Außerdem kann ein Programm mehrere Algorithmen enthalten. Während Algorithmen relativ allgemein beschrieben werden können und an keine formellen Vorschriften gebunden sind, d. h. unabhängig von der Syntax[6] von **Cluster** sind, sind Programme wesentlich konkreter. Sie sind nämlich auf einem Computer direkt ausführbar, und das ist es ja, was wir wollen: richtige Programme schreiben, die auf dem Computer auch laufen.

Beispiel: Berechnung des Mittelwertes zweier Zahlen:

Algorithmus: Den Mittelwert m zweier Zahlen a und b bildet man, indem man beide Zahlen addiert und anschließend durch 2 dividiert.

Programm:

[4] Eine Befehlsliste

[5] Computer legen sehr viel Wert auf Präzision

[6] Die Syntax einer Programmiersprache ist vergleichbar mit der Rechtschreibung einer natürlichen Sprache

```
...
ReadReal(a);             |erste Zahl einlesen
ReadReal(b);             |zweite Zahl einlesen
mittelwert:=(a+b)/2;     |Mittelwert berechnen
WriteReal(mittelwert);   |Ergebnis ausgeben
...
```

Dieses Teilprogramm stellt die für Ihren Rechner verständliche Umsetzung des oben aufgeführten Algorithmus in der Sprache **Cluster** dar. Sie müssen nicht unbedingt alles verstehen. Es dient lediglich der anschaulichen Unterscheidung zwischen Algorithmus und Programm.

3.3.2 Was ist eine Sprache?

Man unterscheidet natürliche und künstliche Sprachen: Natürliche Sprachen haben sich im Laufe der Evolution in Jahrmillionen entwickelt und werden vom Menschen zur Verständigung untereinander verwendet. Sie existieren in Form von gesprochener Sprache und Schriftsprache und sind ständiger Veränderung unterworfen. Künstliche Sprachen sind erst seit gut hundert Jahren auf dem Markt und nicht natürlich gewachsen, sondern von Menschenhand konstruiert worden, um Schlußfolgerungen und Denkvorgänge darstellen und durchdringen zu können. Seit den 40er Jahren entstanden dann daraus die ersten Programmiersprachen.

Künstliche Sprachen wie die Programmiersprachen sind im Gegensatz zu den natürlichen Sprachen nach strengen Regeln aufgebaut und haben ein festes, endliches Grundvokabular sowie eine feste Syntax. Während bei den natürlichen Sprachen die Bedeutung eines Wortes oftmals Auslegungssache ist, besitzen Wörter

und Sätze in Programmiersprachen eine genau definierte Bedeutung.

3.3.3 Was ist eine Programmiersprache?

Um dem Computer ein Programm bzw. die in ihm enthaltenen Befehle verständlich zu machen, muß man mit ihm in einer bestimmten Sprache sprechen, was meistens schriftlich über die Computertastatur geschieht. Diese Sprache heißt nun aber nicht Deutsch oder Französisch (natürliche Sprachen), sondern z. B. **Cluster** (künstliche Sprache). Sie fragen vielleicht, warum man mit dem Gerät nicht einfach deutsch reden kann, aber das funktioniert leider aus technischen Gründen z. Zt. noch nicht.

Der Computer ist nun einmal ein blödes Ding, nicht intelligenter als eine Kaffeemaschine und obendrein noch schrecklich sensibel. Man mußte erst eigene Sprachen für ihn entwickeln, bis er mit dem Menschen kommunizieren wollte[7], und diese Sprachen mag er, wenn er sie denn wirklich versteht, nur in ihrer reinsten Form.

Pustekuchen also mit Unsauberkeiten und solchen Sachen wie Dialekt o. ä. .

Damit der Mensch aber auch etwas davon hat, ist man ihm bei dieser Entwicklung ein ganzes Stück entgegengekommen und hat ihm das Leben soweit wie möglich erleichtert, indem man z. B. Vokabeln aus menschlichen Sprachen zum Einsatz kommen ließ. Meistens hat man sich dabei am Englischen orientiert, so bei Pascal, Modula 2 und damit auch bei **Cluster**. Wer die englische Sprache nicht mag, was man gut nachvollziehen kann, muß sich trotzdem davor nicht fürchten. Es gibt nämlich doch noch einige

[7]Korrekter ist natürlich: Bis man sich mit dem Computer verständigen konnte

wesentliche Unterschiede zur gesprochenen Sprache.

Wenn man sich erst einmal an diese kleinen Unterschiede zwischen natürlicher und künstlicher Sprache gewöhnt hat und obendrein noch in der glücklichen Lage ist, mit einem so eleganten Werkzeug wie **Cluster** mit dem Computer reden zu dürfen, macht das Ganze einen Mordsspaß.

3.3.4 Syntax – die Grammatik einer Programmiersprache

Eine Sprache wird durch eine Folge von Zeichen definiert, welche bestimmten Regeln gehorchend zusammengesetzt werden dürfen. Den hierdurch beschriebenen formalen Aufbau der Sätze oder Wörter, die zur Sprache gehören, bezeichnet man als ihre Syntax.

Die Syntax einer Programmiersprache legt fest, welche Zeichenreihen korrekt formulierte Programme der Sprache sind und welche nicht. Um präzise feststellen zu können, ob ein Programm syntaktisch korrekt ist, muß man zuvor die Syntax der Sprache formal beschreiben.

3.4 Was ist Programmierung?

Programmierung ist einfach der Vorgang der Programmerstellung. Dazu gehört eine vernünftige Beschreibung, vielleicht auch eine graphische Darstellung des zu lösenden Problems bzw. des dafür geeigneten Lösungsweges und natürlich das „Füttern” des Computers mit Programmtext (Quellcode oder auf Neudeutsch „Sourcecode“).

3.4.1 Was versteht man unter „strukturierter Programmierung”?

Strukturierte Programmierung ist ein Programmierverfahren, das auf der Modularisierung eines Problems beruht. Dabei werden aus dem vorliegenden Problem verschiedene Teilprobleme (im Programm: Prozeduren) konstruiert, die unabhängig voneinander entwickelt und auch in anderen Programmen wieder verwendet werden können.

Zum Erreichen dieses Ziels werden zwei Methoden angewendet: die Top-down-Methode[8] und die Bottom-up-Methode[9].

Für uns soll hier zunächst einfach gelten: Zerlege das gegebene Problem in möglichst viele Teilprobleme und in die Beziehungen zwischen diesen Teilprobleme, die Schnittstellen genannt werden. Um Ihnen dieses Verfahren, was dem Top-down-Entwurf sehr nahe steht, etwas deutlicher zu machen, soll noch einmal das Beispiel mit der Schneckensuppe angeführt werden:

> Es soll gezeigt werden, wie man die einzelnen Teilschritte des Schneckensuppenbeispiels noch weiter aufteilen kann. Dazu nehmen wir uns einfach die Schritte (1) und (7) vor. Man könnte sie folgendermaßen weiter zerlegen:

1.Schritt: Sie stehen auf:
- a) Augen aufschlagen
- b) Nachttischlampe anknipsen

[8] Durch schrittweise Verfeinerung wird das Ausgangsproblem in einfacher zu lösende Teilprobleme zerlegt. Hierbei wird bei jedem Entwurfsschritt festgelegt, was die Untermodule (Teilprogramme) leisten sollen. Die Funktionen der Module einer Ebene sollen nur durch die Funktionen jener Module der direkt darunterliegenden Ebene verwirklicht werden

[9] geht nicht vom gegebenen Problem, sondern von den Funktionen und Möglichkeiten der zur Verfügung stehenden Basismaschine aus. Dies dreht die Vorgehensweise von Top-down also praktisch gerade um

c) Bettdecke zurückschlagen
e) Füße aus dem Bett wuchten
f) Pantoffeln anziehen
g) Bett verlassen

7.Schritt: Sie erwärmen nun den Topf:

a) Herdplatte anstellen
b) Schneckensuppe gelegentlich umrühren
c) wenn die Suppe warm ist, Herd abschalten und Topf herunterziehen

Als Beispiel für eine Schnittstellenbeziehung könnte man hier 1.e als Voraussetzung für 1.f nennen: Bevor man sich die Pantoffeln anziehen kann, müssen die Füße erst einmal auf dem Boden stehen. Oder folgende Bedingung: Um die Dose überhaupt sehen und greifen zu können, muß unbedingt das Licht angeschaltet werden, sonst kann es passieren, daß man sich mit dem Dosenöffner den Magen öffnet, was eher zu den unangenehmen Dingen des Lebens gehört und nichts mehr mit Programmierung zu tun hat...

7.c wäre übrigens eine Beispiel für eine Verzweigung, die weiter unten behandelt wird: Die Suppe wird erst dann vom Herd genommen, wenn sie warm ist.

Diese Verfeinerungsschritte führt man solange durch, bis das behandelte Problem ohne weitere Verfeinerung lösbar bzw. programmierbar ist.

3.5 Variablen und Konstanten

Variablen sind Speicherplätze im Rechner, in denen Zahlen, Buchstaben, Wörter oder andere Objekte (dazu später) gespeichert werden. Eine Variable hat einen Namen (Variablenname), einen sogenannten Datentyp und einen Dateninhalt. In anderen Worten:

Wenn man sich den Computerspeicher als eine Anzahl kleiner Kästchen oder Schubladen vorstellt und in jedes dieser Kästchen etwas abgespeichert bzw. abgelegt werden kann, stellen die Variablennamen die Beschriftung dieser Kästchen dar. Mit Variablen kann genauso gearbeitet werden wie mit den Daten selbst. Mit einer Zahlenvariablen kann z. B. gerechnet werden wie mit anderen Zahlen.

Der Name der Variablen und ihr Typ werden im Deklarationsteil eines Programmes festgelegt. Der Dateninhalt der Variablen kann bei der Deklaration (siehe unter 3.5.3 ab Seite 26) und im Programm zugewiesen und jederzeit geändert werden.

Meistens benutzt man Buchstaben, um Variablen darzustellen, so auch in der Programmiersprachen **Cluster**, wo allerdings auch bestimmte Kombinationen aus Zahlen und Buchstaben zulässig sind. Mit Variablen kann man in erster Linie rechnen[10].

Bei der Bildung von Variablennamen sind der Phantasie kaum Grenzen gesetzt (sie sollten lediglich – so weit möglich – selbsterklärend sein), doch gibt es einige einschränkende Regeln hierfür. Anhand der folgenden Beispiele werden Sie schnell dahinter kommen.
Zulässige Variablennamen sind zum Beispiel:

```
x, x1, x11
text, Text, TEXT, TeXt
a1Zeichen, Aktenzeichen2XOrdner3b
```

Variablennamen dürfen also aus einer beliebigen Kombination von kleinen und großen Buchstaben und Zahlen bestehen, wobei am Anfang eines Namens keine Zahl stehen darf.
Unzulässig sind daher z. B.

[10] Abhängig vom Typ der Variablen

```
1Text, 10xyz
```

Auch Leerzeichen und Bindestriche darf Ihr Variablenname nicht enthalten. Unzulässig sind deshalb auch folgende Gebilde:

```
Anfang Text, Min-Max, a3-b2
```

Erlaubt ist jedoch der Unterstrich. Somit darf z. B.

```
Min_Max
```

verwendet werden.

Beachten Sie außerdem – und damit kommen wir auch schon ans Ende dieser vielleicht etwas langweiligen Angelegenheit – daß die Variablen `Text`, `text`, `TEXT` und `TeXt` nicht identisch sind. Wir kommen hier auf eine besondere Eigenschaft von **Cluster** zu sprechen, die man Case-Sensitivität nennt und weiter unten auch noch einmal kurz angesprochen wird. „Case-sensitiv" heißt nichts anderes, als daß Groß- und Kleinschreibung – anders als bei einigen anderen Programmiersprachen – unterschieden werden.

Es gibt, wie schon angedeutet, verschiedenartige Variablen, die sich durch ihre Grundmenge unterscheiden. Man spricht von verschiedenen „Typen" von Variablen. Das ist für die Programmierung in **Cluster** sehr wichtig. Z. B. gibt es Variablen, in die nur ganze Zahlen eingesetzt werden dürfen, andere nehmen nur Buchstaben auf usw. Daraus folgt, daß man eine Variablen `a`, die nur ganze Zahlen aufnimmt, nicht mit einer Variablen `b`, die beispielsweise nur Buchstaben und Sonderzeichen aufnehmen darf, einfach multiplizieren kann.

Im Gegensatz zu Variablen bekommen *Konstanten* einmal am Anfang eines Programms einen festen Wert zugewiesen und behalten diesen bis zum Ende bei. Dieser Wert darf innerhalb des Programmes nicht mehr geändert werden.

Es hat sich eingebürgert, Variablen klein zu schreiben[11]. Das ist eine Konvention, an die man sich halten sollte. Wird ein Variablenname aus mehrern Worten gebildet, so sollte jedes weitere Wort mit einem Großbuchstaben beginnen (z. B. zeilenZaehler, hilfsZeiger, ...).

Wie Variablen bzw. Konstanten in einem Programm festgelegt (deklariert) werden, wird unter 3.5.3 ab Seite 26 besprochen.

3.5.1 Operationen und Operatoren

Der Unterschied zwischen Operation und Operator sei hier nur der Form wegen kurz erklärt: Während die Operation einen konkreten Arbeitsvorgang im Computer darstellt, ist ein Operator ein Funktionszeichen für arithmetische Operationen[12].

```
Flaeche = Laenge * Breite
```

Hier stellt der Stern (*) den Operator bzw. das Operatorzeichen für die auszuführende Operation – nämlich eine Multiplikation – dar.

Das ganze Gebilde wird als Gleichung, die Teile links und rechts vom Gleichheitszeichen als Ausdrücke (Terme) bezeichnet.

In **Cluster** gibt es außerdem einen Zuweisungsoperator „`:=`". Will man der Variablen „`Test`" zum Beispiel den Wert `5` zuweisen, so schreibt man: `Test:=5`.

In der Mathematik würde man $test = 5$ schreiben. Allerdings ist bei der Programmierung „`test := test + 10.5`" möglich, welches im mathematischen Sinne falsch wäre: $test = test + 10.5$ ist falsch. Aus diesem Grunde wurde auch nicht das Gleichheitszeichen verwendet, wie in anderen Programmiersprachen (BASIC, C, ...),

[11] Mit Ausnahme von globalen Variablen
[12] Es existieren außerdem auch Mengen-Operatoren

sondern die Kombination von Doppelpunkt und Gleichheitszeichen „`:=`".

Obiger Ausdruck bedeutet also in Wirklichkeit: Nimm den Wert (Inhalt) der Variablen `test` (hier: `5`), addiere dazu den Wert `10.5` und weise das Ergebnis der Variablen `test` zu. Der vorherige Wert (`5`) wird folglich überschrieben!

3.5.2 Variablentypen

Jetzt soll Ihnen wertvolles Wissen nicht länger vorenthalten bleiben und nähere Informationen zu Variablen gegeben werden. Die verschiedenen Typen finden Sie jetzt gleich hier zusammengestellt, so daß Sie eventuelle Fragen auf einen, höchstens aber auf zwei Blicke beantwortet bekommen.

Zu jedem Typ ist als erstes die Grundmenge aufgeführt. Sie werden sich ja sicherlich noch daran erinnern, daß verschiedene Variablentypen verschiedene Grundmengen haben. Falls das nicht der Fall sein sollte, dann schielen Sie noch einmal nach oben unter 3.5 auf Seite 10.

Auf die Grundmenge folgen die Länge des Typen in Bytes[13] und die für den entsprechenden Typ zulässigen Operatoren in Anführungszeichen, wie sie eben besprochen worden sind. Zum Schluß folgt eine kurze Beschreibung mit Hinweisen, und ab und zu ein Beispiel.

3.5.2.1 CARDINAL / LONGCARD / SHORTCARD

Grundmenge: Positive ganze Zahlen im Bereich von `0` bis `65535` (`CARDINAL`) bzw. von `0` bis `4294967295` (`LONGCARD`). Es existiert auch noch der Typ `SHORTCARD`, der für den Bereich `0` bis `255` definiert ist.

Länge: `CARDINAL` 2 Bytes, `LONGCARD` 4 Bytes, `SHORTCARD` 1 Byte.

[13] Entspricht dem Platz, den eine Variable diesen Typs im Speicher belegt.

Operatoren: „+", „-", „*", „DIV", „MOD" (Zu DIV und MOD s. a. Tabelle 3.1 auf Seite 17) und die Vergleichsoperatoren: „<" (kleiner), „<=" (kleiner gleich), „=" (gleich), „>=" (größer gleich), „>" (größer), „#" (ungleich).

Beschreibung: CARDINAL und LONGCARD sind die einfachsten Variablentypen in **Cluster** und haben, wie Sie unschwer an der Grundmenge erkennen können, wenig mit kirchlichen Würdenträgern zu tun, obwohl sich dieser Gedanke natürlich aufdrängt.

Wann immer es möglich ist, sollte CARDINAL dem Typ LONGCARD vorgezogen werden, da Rechnungen mit LONGCARD-Zahlen immer mehr Rechenzeit und Speicher vom Computer verlangen.

Darum überlegen Sie sich vorher, welche Grundmenge für Ihr Problem wirklich notwendig ist.

3.5.2.2 INTEGER / LONGINT / SHORTINT

Grundmenge: Positive und negative ganze Zahlen im Bereich von -32768 bis 32767 (INTEGER) bzw. von -2147483648 bis 2147483647 (LONGINT). Auch hier gibt es noch einen Typ SHORTINT, dessen Variablen Werte zwischen -128 bis 127 aufnehmen können.

Länge: INTEGER 2 Bytes, LONGINT 4 Bytes, SHORTINT 1 Byte.

Operatoren: „+", „-", „*", „DIV", „MOD" und die Vergleichsoperatoren: „<", „<=", „=", „>=", „>", „#".

Beschreibung: Auch hier gilt, was schon bei CARDINAL / LONGCARD / SHORTCARD gesagt wurde: Rechnen mit LONGINT-Zahlen ist zeitaufwendiger als mit der INTEGER-Variante.

Mit DIV wird eine ganzzahlige Division durchgeführt, die nur die Vorkommastelle ausgibt. Berechnet man also z. B. 16 DIV 5, dann wird der Wert 3 ausgegeben, denn die 5 „paßt" dreimal in die 16. Mit MOD wird der ganzzahlige Rest einer DIV-Operation ausgegeben. Bei unserem Beispiel würde diese Operation den Wert 1 liefern. Wie in der Grundschule gelehrt wird: 3 mal 5 sind 15, Rest 1.

Zur weiteren Verdeutlichung der Operatoren DIV und MOD dient die kleine Beispieltabelle, die Ihnen diese Funktionen nochmals eingängig machen soll (Tabelle 3.1)

16	DIV	4	=	4	16	MOD	4	=	0
5	DIV	15	=	0	5	MOD	15	=	5
15	DIV	4	=	3	15	MOD	4	=	3
10	DIV	4	=	2	10	MOD	4	=	2
-14	DIV	3	=	-4	-14	MOD	3	=	-2
-10	DIV	-4	=	2	-10	MOD	-4	=	-2
10	DIV	-4	=	-2	10	MOD	-4	=	2

Tabelle 3.1: Beispiele für DIV und MOD

Der eine oder andere Leser dürfte jetzt etwas verwirrt sein. Die Vielzahl der ganzzahligen Datentypen ist in **Cluster** nicht gerade gering. Nun, wofür braucht man so viele Datentypen:

Bevor man eine Variable im Programm verwendet, sollte man drüber nachdenken, wie die zu speichernden Zahlen beschaffen sind. Sollen negative Zahlen verarbeitet werden, so muß man den Datentyp `INTEGER` verwenden. Je nachdem wie groß nun die Zahlen werden können, verwendet man `SHORTINT`, `INTEGER` oder `LONGINT`. Nun könnte man erwiedern, nehme ich halt den größen Datentyp. Hier spricht dagegen, daß damit mitunter wertvoller Speicher verschwendet wird. Der größte Datentyp nimmt viermal so viel Speicher ein, wie der kleinste Datentyp. Außerdem benötigt eine Rechnung mit einem größeren Datentyp mehr Zeit, als mit einem kleineren Datentyp.

Sind Sie jedoch sicher, daß Sie nur positive Zahlen verwenden werden, so legen Sie besser eine Variable vom Typ `CARDINAL` an. Hier hat man den Vorteil über einen größeren Zahlenbereich zu verfügen, denn der Speicherplatz kann nun vollständig für die positiven Zahlen verwendet werden. Außerdem führt der Compiler einen Check durch, ob nicht aus Versehen einer solchen Variable ein negativer Wert zugewiesen wird.

3.5.2.3 REAL / LONGREAL

Grundmenge: Praktisch jede rationale Zahl, deren Mantisse (die Erklärung hierzu folgt gleich) eine Genauigkeit von 8 Stellen (`REAL`[14]) bzw. 15 Stellen (`LONGREAL`) und einen Exponenten hat, der Werte zwischen `-23` und `23` (`REAL`) bzw. zwischen `-308` und `308` annehmen kann. Zur rechnerinternen Darstellung von `REAL`- / `LONGREAL`-Zahlen mittels Mantisse und Exponent s. a. die hierzu gehörige Beschreibung.

Länge: `REAL` 4 Bytes, `LONGREAL` 8 Byte.

[14] REAL kann erst ab Kickstart 2.0 benutzt werden. Siehe `FFP`

Operatoren: „`+`", „`-`", „`*`", „`/`", „ ^ " (Potenz) und die Vergleichsoperatoren: „`<`", „`<=`", „`=`", „`>=`", „`>`", „`#`".

Beschreibung: Um die Frage zu beantworten, warum auch `REAL` bzw. `LONGREAL` eine eingeschränkte Grundmenge besitzen, muß man etwas ausholen und ein wenig in die Gedärme des Computers eindringen. Wen das ekelt oder wer sich für die mathematischen Feinheiten des Lebens wenig interessiert, der kann getrost ein paar Sätze überspringen und sich an den Beispielen gütlich tun.

Nun denn: Weil sich der Computer sehr große Zahlen mit möglichst wenig Speicheraufwand merken möchte – er ist schon sparsam, unser Computer – und sie dadurch auch schneller verarbeiten kann, trennt er jede `REAL`- und `LONGREAL`-Zahl in Mantisse und Exponent auf. Das ist die mathematische Darstellung. Die Mantisse ist eine Dezimalzahl, deren Betrag größer oder gleich 1 und kleiner 10 mit einer beschränkten Anzahl von Stellen hinter dem Komma ist (bei `REAL` sind das, wie oben erwähnt, 8 Stellen). Der Exponent ist eine positive oder negative ganze Zahl, die die Anzahl der Stellen angibt, um die die Mantisse gegenüber der tatsächlichen Zahl verschoben ist.

Damit ergibt sich folgende Darstellung:

```
Zahl = Mantisse * 10 ^ Exponent.
```

Zu kompliziert? Hier ein paar Beispiele:

Zahl	=	Mantisse	* 10 ^	Exponent
1737524.0000	=	1.737524	* 10 ^	6
-101.3537	=	-1.013537	* 10 ^	2
5.13	=	5.13	* 10 ^	0

```
  0.05243    =  5.243    * 10 ^  -2
-0.00004254 = -4.254     * 10 ^  -5
```

Eine Zahl, die einer `REAL`- oder `LONGREAL`-Variablen zugewiesen werden soll, kann wahlweise eine normale Darstellung (z. B. `33651`, `65.873` oder `0.761`) haben, oder aber in Exponentendarstellung erscheinen. Bei der Exponentendarstellung wird zuerst die Mantisse geschrieben, gefolgt von einem großen oder kleinen „`E`" für Exponent (hier wird nicht zwischen Groß- und Kleinschreibung unterschieden) und dem Exponenten (z. B. `3.3651e4, 107.25E-3` oder `-0.15e-7`).

Noch einmal zur Verdeutlichung: Folgende Zuweisungen in **Cluster** sind äquivalent (⇔), d. h. gleichwertig:

real1:=15142.221 ⇔ real1:=1.514221e4 ⇔ real1:=151.4221E2

real2:=−0.006252⇔real2:=−6.252E−3⇔real2:=−0.6252e−2

3.5.2.4 FFP

Grundmenge: Die gleiche wie Real.

Länge: 4 Bytes.

Operatoren: „`+`", „`-`", „`*`", „`/`", „ `^` " (Potenz) und die Vergleichsoperatoren: „`<`", „`<=`", „`=`", „`>=`", „`>`", „`#`".

Beschreibung: Falls Sie nur Kickstart 1.3 besitzen, können Sie keine `REAL`-Zahlen benutzen, da erst ab 2.0 die entsprechende Library dazugekommen ist. Dafür existiert jedoch ein anderer Zahlentyp, der von einem MC68000 auch ohne Library verarbeitet werden kann. Dies ist `FFP`. Auf einem normalen 68000 wird dieser Typ auch schneller verarbeitet, als `REAL`. Besitzt man jedoch eine Turbokarte mit FPU[15], dann sollte man `REAL` verwenden, da dieser Typ dann schneller verarbeitet wird.

Besitzt man nur Kickstart 1.3, und möchte ein Programm, das `REAL` verwendet, muß man nicht jedes `REAL` durch `FFP` ersetzen. Folgende Typdefinition am Programmanfang reicht vollkommen aus:

```
TYPE
  REAL = FFP
```

Also noch mal, `FFP` immer dann, wenn das Programm unter 1.3 laufen soll, sonst immer `REAL`.

3.5.2.5 BOOLEAN

Grundmenge: Die Wahrheitswerte „wahr" (`TRUE`) und „falsch" (`FALSE`).

Länge: 1 Byte.

Operatoren: „`AND`", „`OR`", „`NOT`"

Beschreibung: Hier dringen wir ein bißchen in die Boolesche Algebra vor, die grundlegend für die Funktionsweise eines Computers ist. Das ist jetzt einmal eine ganz andere Welt der Mathematik, die mit Aussagelogik zu tun hat. Hier können Ausdrücke nur wahr oder falsch sein. Boolesche Variablen haben in Programmen oft Schalterfunktion, d. h. sie

[15] Mathematischer Coprozessor

verkörpern den Zustand eines angeschalteten oder abgeschalteten Knopfes.

Wie sich die Operatoren AND, OR und NOT auf die Variablen auswirken, machen die nun folgenden Tabellen, die man ihres Inhalts wegen auch Wahrheitstabellen nennt, deutlich:

Ausdruck1	**Ausdruck2**	**Ausdruck1** `AND` **Ausdruck2**
`TRUE`	`TRUE`	`TRUE`
`FALSE`	`TRUE`	`FALSE`
`TRUE`	`FALSE`	`FALSE`
`FALSE`	`FALSE`	`FALSE`

Ausdruck1	**Ausdruck2**	**Ausdruck1** `OR` **Ausdruck2**
`TRUE`	`TRUE`	`TRUE`
`FALSE`	`TRUE`	`TRUE`
`TRUE`	`FALSE`	`TRUE`
`FALSE`	`FALSE`	`FALSE`

Ausdruck	`NOT` **Ausdruck**
`TRUE`	`FALSE`
`FALSE`	`TRUE`

BOOLEAN-Werte können miteinander auf Gleichheit und Ungleichheit überprüft werden. Dies geschieht mit „`=`“ für Gleichheit und

„`#`“ für Ungleichheit. Beachten Sie auch, daß Ergebnisse von Relationen (z. B. `a = 1`) immer vom Typ BOOLEAN sind. Beispiele hierzu werden weiter unten folgen.

3.5.2.6 CHAR

Grundmenge: Alle auf Ihrem Computer darstellbaren Zeichen (siehe ASCII(ISO)-Tabelle im Anhang), also Klein- und Großbuchstaben, die Ziffern 0 bis 9 sowie Sonderzeichen wie z. B. ?, *, # usw. Eine Variable des Typs `CHAR` kann immer nur ein Zeichen aufnehmen.

Länge: 1 Byte.

Operatoren: Nur bedingt vorhanden: z. B. die Vergleichsoperatoren „`=`" (gleich) und „`#`" (ungleich), die in Booleschen Ausdrücken verwendet werden können. Außerdem `<`, `<=`, `>=` und `>`.

Beschreibung: „`CHAR`" ist eine Abkürzung für „character" (Zeichen). Dieser Variablentyp wird immer dann benutzt, wenn man Eingaben, die z. B. nur aus „j" (Ja) oder „n" (Nein) bestehen, abprüfen oder eine Datei zeichenweise parsen möchte. Besonders wichtig wird dieser Typ im Zusammenhang mit dem Typ `STRING`, der sich aus einzelnen Zeichen zusammensetzt. Er wird unter 3.14.5 ab Seite 100 beschrieben.

3.5.2.7 Typenkonvertierung

In einem Ausdruck darf bei **Cluster** immer nur ein Variablentyp vorkommen, was in bestimmten Fällen die Notwendigkeit einer Typenkonvertierung hervorruft. Schließlich will man ja mal eine `REAL`-Zahl mit einer `INTEGER`-Zahl multiplizieren o. ä.

Ein Beispiel: Sie wollen die `INTEGER`-Variable `a` mit der REAL-Variablen `b` multiplizieren und das Ergebnis einer REAL-Variablen `c` zuweisen. Es muß also a in eine `REAL`-Zahl verwandelt werden. Das geschieht einfach so:


```
...
VAR
  a   : INTEGER; | Variablendeklaration
  b,c : REAL;
...
  c:=REAL(a)*b; | INTEGER-"a" wird nach
                | REAL konvertiert
...
```

Sie haben es gesehen? Nur die Variable (oder den Ausdruck), die (der) konvertiert werden soll, in Klammern setzen und den neuen Typ davorschreiben.

Bei der Konvertierung ist darauf zu achten, daß bei der Verwandlung von `REAL` nach `INTEGER` die Nachkommastellen einfach abgeschnitten werden. Wundern Sie sich also nicht über Rundungsfehler, die Sie jedoch mit Hilfe von geeigneten Standardfunktionen umgehen können.

Im Prinzip ist jede Typenkonvertierung einfach möglich, sollte dies mal nicht der Fall sein, macht sich der Compiler in seiner ihm eigenen Art bemerkbar. Keine Angst also davor, daß Sie was verpassen könnten.

Neben dieser Art der Typkonvertierung existiert noch eine zweite, nämlich durch `CAST`. Der unterschied zu der oben beschriebenen Konvertierung ist der, das `CAST` nur auf gleichlange Typen anwendbar ist, und wirklich nur eine Änderung des Typs vornimmt, ohne eine eigenliche Konvertierung des Wertes im Speicher vorzunehmen. Wandelt man also mit `CAST` eine `REAL`- in eine `LONGINT`-Zahl, wird man kaum etwas vernünftiges erhalten, da der Aufbau der Zahlen komplett verschieden sind. In manchen Fällen, kann allerdings auch ein solches Ergebnis erwünscht sein.

Im Prinzip kann man alle gleichlangen Typen, egal wie sie sich zusammensetzen mit `CAST` in einander umwandeln, mehr Beispiele finden sich auch im Kapitel über harwarenahe Programmierung. Die Verwendung von `CAST` sieht folgendermaßen aus:

```
CAST(<Typ>,<Variable>)
```

3.5.3 Variablendeklaration

Die Variablendeklaration ist nicht zwingend notwendig, doch gibt es nur sehr wenige Module, die ohne sie auskommen. Beispielsweise wäre ein Modul denkbar, welches nur eine bestimmte Warnmeldung ausgibt, für die keine Variablen benötigt werden.[16]

Die Deklaration beginnt mit dem Schlüsselwort „**VAR**" (s. u.). Ihm folgen alle globalen Variablen (was es damit auf sich hat, wird weiter unten erklärt) zusammen mit dem gewünschten Typ. Zwischen Variable und Variablentyp steht ein „:", der Deklarationsoperator. Sollen mehrere Variablen vom selben Typ sein, so können sie durch Kommata getrennt aufgelistet und dann gemeinsam durch „:" ihrem Typ zugeordnet werden. Die einzelnen Deklaration werden durch Semikola getrennt. Ein Beispiel macht das alles klarer:

[16] Außerdem gibt es noch die Definitions- und Implementationsmodule, die ohne Variablendeklaration auskommen und auf die gegen Ende dieses Kapitels noch ausführlich eingegangen wird.


```
MODULE Beispiel_fuer_Variablendeklaration;
...
VAR
  x                    : INTEGER;
  y,y0,y1              : REAL;
  zeichen              : CHAR;
  wohnort              : STRING(30);
  jaOderNein,absage    : BOOLEAN;
  MeinString           : String4711; |selbstdefinier-
                                     |ter Typ
...
```

Die Einrückungen dienen nur der Übersichtlichkeit und Lesbarkeit; für den Programmablauf sind sie unerheblich. Man nennt diese Eigenschaft von **Cluster**, die in „Besonderheiten von **Cluster**" noch näher erläutert werden wird, Formatfreiheit.

Als Besonderheit von **Cluster** kann bei der Variablendeklaration ein Wert direkt mitgegeben werden, welches wie folgt aussehen kann:

```
VAR
 heute  :  INTEGER := 10;
 morgen :  REAL := 123.56;
```

Dies erspart Ihnen eine spätere Initialisierung[17] der Variablen.

[17] Erstmalige Belegung von Variablen mit Werten

Übungen 1:

Unter dieser Rubrik sind Aufgaben zur Selbstkontrolle zusammengefaßt. Nicht, daß wir Sie mit Hausaufgaben überlasten wollen, denn Sie können ja schließlich selbst entscheiden, ob Sie diese lösen wollen. Bedenken Sie jedoch, daß es ein gutes Zeichen ist, wenn Sie die Aufgaben lösen konnten. Ansonsten sollten Sie lieber den einen oder anderen Abschnitt nochmals durcharbeiten.

Im übrigen, die Lösungen zu den Fragen finden Sie im Anhang dieses Handbuchs.

1. Welche der folgenden Variablennamen sind in **Cluster** gültig?
   (a) `Erster-Name`
   (b) `zaehler`
   (c) `ErsatzWert`
   (d) `Y991`
   (e) `Vorname$Nachname`
   (f) `Durch_Schlag`
2. Welche zehn elementaren Datentypen existieren in **Cluster**?
3. Welcher Unterschied besteht zwischen `DIV` und `/`?
4. Betrachten Sie folgende Variablenerklärungen:
   ```
   VAR
     ch : CHAR;
     I  : INTEGER;
   ```
   Ist folgender Befehl gültig?
   ```
   ch := I;
   ```


3.6 Das erste Programm

Damit Sie auch gleich schon sozusagen live am Computer mitmachen und die hier aufgeführten Beispiele eingeben können, kurz ein paar Worte zur Bedienung des Editors und seiner für Sie im Moment relevanten Funktionen, die Ihnen teilweise aus diesem Handbuch bekannt sein sollten.

```
MODULE Addition;

FROM InOut IMPORT WriteInt;

VAR a,b,c : INTEGER;

BEGIN

    a:=10;
    b:=23;
    c:=a+b;
    WriteInt(c);  |mit "Write" und Konsorten werden
                  |Bildschirmausgaben veranlaßt.
END Addition.
```

Verwenden Sie die untenstehende Gebrauchsanleitung für das ganze Kapitel.

Fangen wir am besten gleich an. Den Editor sollten Sie nach Anleitung bereits gestartet haben und nun den fast leeren Bildschirm vor sich haben. Geben Sie, möglichst in derselben Form wie angegeben, den aufgeführten Programmtext ein.
Achten Sie bei der Eingabe auf Groß- und Kleinschreibung! Sind Sie mit der Eingabe fertig, öffnen Sie das **Cluster**-Menu am unteren Bildschirmrand, indem Sie es mit dem Mauszeiger an der

linken Lasche „packen", die Maustaste gedrückt lassen und nach oben ziehen.

Speichern[18] Sie das Programm durch Klicken auf (`SAVE`) in das Verzeichnis „`Cluster:Work/TXT`" unter dem Namen „Addition.mod"[19] ab.

Klicken Sie das Feld (`COMPIL`) an. Der **Cluster**-Compiler nimmt nun seine Arbeit auf. Warten Sie's ab! Entweder gibt er eine Fehlermeldung aus oder Sie bekommen ein blinkendes (`EXIT`) präsentiert. Eventuelle Fehler korrigieren Sie bitte nach Anleitung des Compilers – da muß man durch und lernt eine ganze Menge dabei, wie Ihnen jeder Programmierer bestätigen kann.

Ist alles o.k., klicken Sie auf (`RUN`). Der Rechner legt nun wieder los und importiert[20] die benötigten Module (siehe weiter unten). Dann startet das Programm (in unserem Fall werden zwei Zahlen addiert – nicht so besonders aufregend, schon klar.). Ist das Programm fertig, können Sie mit der RETURN-Taste in den Editor zurückkehren.

Wollen Sie sich später das Programm einmal zurückholen, so geschieht dies mittels Mausklick auf den Knopf (`LOAD`). Den Editor verläßt man mit (`EXIT`). Wurde das zuletzt bearbeitete Programm noch nicht gesichert, so wird nachgefragt – das ist ein Service, nicht wahr?

Das obenstehende Programm ist – wie bereits gesagt – sehr, sehr einfach. Es addiert zwei Zahlen und gibt das Ergebnis aus. Was Ihnen – hoffentlich – als erstes auffällt[21], sind die fettgedruckten Wörter. Die erscheinen nicht nur hier im Druckbild so, sondern werden vom Editor erkannt und automatisch fett dargestellt – probieren Sie das am besten gleich einmal aus. Das soll ein Zeichen dafür sein, daß er sie verstanden hat. Diese Wörter nennt man Schlüsselwörter. Sie müssen ausschließlich GROSS geschrieben werden!

[18] Das regelmäßige Speichern des Textes gehört zu den wichtigsten Regeln im Umgang mit einem Computer, egal mit welchem Programm Sie arbeiten.

[19] Vorerst achten Sie sich bitte darauf, daß jedes ihrer Programme beim Abspeichern ein „`.mod`" angehängt bekommt.

[20] lädt sie von Diskette oder Festplatte in den Speicher des Rechners

[21] wenn nicht, sollten Sie erst mal 'ne Tasse Kaffee trinken und einen Moment die Augen schließen

Jedes Programm fängt für uns zuerst einmal mit dem Schlüsselwort „**`MODULE`**" an, gefolgt vom Namen des Programms, denn wir wollen ja auch ein bißchen Ordnung haben und zudem erfahren, welchen Zweck ein vorliegendes Programm erfüllt, was nicht selten aus seinem Namen ersichtlich wird. Der Name „**`MODULE`**" deutet schon auf den modularen Aufbau eines **Cluster**-Programmes und somit auf die gesamte Programmierungsphilosophie dieser Sprache hin.

Enden tut das Ganze – wie sollte es anders sein – mit einem fetten „**`END`**", dem nochmals der Programmname und dann ein Punkt folgt. Das wäre der äußere Rahmen. Ganz einfach eigentlich, denken Sie jetzt? Nun ja, es soll Ihnen wirklich nicht der Wind aus den Segeln genommen werden, schließlich haben auch andere diese Sprache verstanden, aber es nutzt alles nichts: Es muß immer wieder auf die Pingeligkeit des Rechners hinwiesen werden. Schreiben Sie „**`MODULE`**" nicht klein! Der Compiler erschreckt Sie dann mit Worten wie `"Schwerer Fehler aufgetreten!"`, wenn Sie Ihr Programm compilieren.

3.6.1 Konstantendeklaration

Die Konstantendeklaration kann innerhalb eines Programms wegfallen und tut dies auch öfter als die Variablendeklaration. Begonnen wird sie mit dem Schlüsselwort „**`CONST`**", gefolgt vom Konstantennamen, dem Zuweisungsoperator „`=`"[22] und dem zuzuweisenden Wert. Die einzelnen Deklarationen werden wiederum durch Semikola getrennt. Das Ende der Konstantendeklaration ist dann erreicht, wenn die Variablendeklaration (s. o.), eine Prozedur (s. „Prozeduren" unter 3.13.1 ab Seite 75) oder das Hauptprogramm beginnt. Auch hier ist die Hoffnung groß, daß das nachfolgende Beispiel alle Unklarheiten aus der Welt schafft und offene Fragen fest verschließt.

```
MODULE Beispiel_fuer_Konstantendeklaration;
...
CONST
  Pi             = 3.14;
  Mehrwertsteuer = 15;
  Chef           = "Arnold Schwarzenegger";
  Wahr           = TRUE;

VAR
  Gehalt : LONGREAL; |Mit der Variablendeklaration
                     |endet die Konstanten-
                     |deklaration
...
```

[22] Hierbei handelt es sich eigenlich um keine Zuweisung wie bei Variablen, vielmehr soll durch das Gleichheitszeichen ausgedrückt werden, daß die Konstante und ihr Wert identisch sind. An jeder Stelle, an der die Konstante verwendet wird, würde sonst dieser Wert stehen.

3.7 Typdeklaration

Sie wissen ja nun von oben, was Typen sind. Die Deklaration von Typen unter Verwendung von Standardtypen kann in einem Programm ebenfalls wahlweise erfolgen. Sie wird begonnen mit dem Schlüsselwort „**`TYPE`**", gefolgt von der Deklaration der Typen.

```
MODULE Beispiel_fuer_Typendeklaration;
...
TYPE
   String32 = STRING(32);
   NewInt   = INTEGER;
   Tester   = BOOLEAN;

CONST
  Pi = 3.14;
...
```

Will man andere als die Standardtypen verwenden, so muß man die gewünschten Typen selber definieren. Dies kommt im folgenden Abschnitt zur Sprache.

3.7.1 Selbstdefinierte Typen

Für einfache Programme reichen die eingebauten elementaren Datentypen meist aus. Bei komplexeren Projekten sind aber Daten erforderlich, die mit keinem dieser Standardtypen übereinstimmen. Für solche Fälle ist es mit **Cluster** möglich, eigene *benutzerdefinierte* Datentype zu erstellen.

Dies kann auf zwei verschiedene Arten geschehen: Entweder durch eine Typdeklaration mittels „**`TYPE`**", oder durch Angabe der

Typdeklaration hinter dem Doppelpunkt in der Variablendeklaration.
Mit

```
TYPE
  INDEX   = CARDINAL;
  SALDO   = REAL;
  ZAEHLER = INDEX;
| Beachten Sie, daß ZAEHLER vom Typ INDEX ist,
| das selbst wieder vom Typ CARDINAL ist
```

werden drei neue Typen erzeugt: `INDEX`, `SALDO` und `ZAEHLER`.

Wurde ein neuer Typ definiert, so kann sein Name überall dort benutzt werden, wo ein Typbezeichner erforderlich ist. Beispielsweise ist die folgende Variablenerklärung in Verbindung mit einer zuvor erfolgten Typerklärung richtig:

```
VAR
  i:  INDEX;
  j:  INDEX;
  k:  INDEX;
  Ueberfaellig, Angemahnt:  SALDO;
  zaehl :  ZAEHLER;
```

Hierbei sind `i, j, k` und `zaehl` vom Typ `CARDINAL` und `Ueberfaellig` und `Angemahnt` vom Typ `REAL`.

Nun werden Sie sich vielleicht fragen, worin der Vorteil besteht, `i, j, k` und `zaehl` vom Typ `INDEX` statt `CARDINAL` und `Ueberfaellig` und `Angemahnt` vom Typ `SALDO` statt `REAL` zu erklären – warum erstellt man scheinbar unnötige Typen, wenn die eingebauten doch ausreichend sind? Die Antwort lautet, daß es jetzt viel einfacher ist, die Typerklärungen anzupassen, sobald sie geändert werden sollen. Falls beispielsweise für `i, j, k` negative Werte benötigt werden, muß nur die Definition von `INDEX` auf

`INTEGER` geändert werden. Es sind keine weiteren Eingriffe notwendig. Bei großen Programmen kann das eine beträchtliche Einsparung an Editierzeit bedeuten. Außerdem kann eine sinnvolle eigene Bezeichnung zur Dokumentation eines Programms beitragen.

3.7.1.1 Der Unterbereichstyp

Am einfachsten erzeugt man einen neuen Typ, indem man einen vorhandenen einschränkt. Dies ist bei allen zählbaren Typen möglich. Zählbar sind Typen wie `CHAR` oder `INTEGER`, sowie Aufzählungstypen (dazu später mehr), nicht jedoch z. B. `REAL`, da die in dieser Menge enthaltenen Elemente unzählbar sind. Einigen müßte noch der Begriff reelle Zahlen geläufig sein. Die Unzählbarkeit ergibt sich daraus, daß zwischen jeweils zwei reellen Zahlen z. B. `4.2` und `4.3` beliebig viele Zahlen liegen, nämlich `4.21`, `4.211`, `4.2111` usw. Da dies immer so ist, egal welche zwei reelle Zahlen genommen werden, sind sie nicht abzählbar. Mathematisch sagt man auch, reelle Zahlen sind nicht diskret, oder auch überabzählbar. Aber nun wieder auf die Erde zurück.

Der gewünschte (eingeschränkte) Bereich wird einfach in eckigen Klammern angegeben.

Unterbereichstypen repräsentieren eine Untermenge der Elemente eines einfachen Typs. Die Grenzen werden in eckigen Klammern getrennt durch „`..`“ gegeben. Wird nur ein Wert angegeben, bedeutet dies einen Unterbereich von null an mit sovielen Elementen.

Beispiel:

```
[1..10], [-200..200], ["A".."Z"], [10], [gruen..blau]
```

Beispiel: Definition eines Unterbereichstyps

```
...
TYPE
  GrossBuchstaben = ["A".."Z"];
  Monate          = [1..12];
...
```

Die auf diese Weise neu erzeugten Typen bleiben zuweisungskompatibel[23] zu ihren Basistypen. Wird einer Variablen des neuen Typs ein Wert zugewiesen, der nicht im angegebenen Bereich enthalten ist, wird von der Laufzeitumgebung ein Laufzeitfehler[24] ausgegeben.
Sie sehen, Unterbereichstypen sind besonders beim Testen von Programmen hilfreich, denn es wird ja eine Laufzeitfehlermeldung ausgegeben, wenn der eingeschränkte Zahlenbereich unter- oder überschritten wurde. Somit sollte man an allen Stellen, an denen man von vornherein weiß, daß nur bestimmte Werte zugewiesen werden dürfen, Unterbereichstypen verwenden, damit eine falsche Zuweisung sofort erkannt wird.

3.7.1.2 Der Aufzählungstyp

Eine weitere Möglichkeit, Typen selber zu erzeugen, besteht darin, zusätzliche abzählbare Typen zu erzeugen, indem man eine be-

[23]kompatibel: Die Variablen dieser Typen können ohne große Probleme einander zugewiesen werden
[24]Laufzeitfehler: Dies ist ein Fehler, welcher bei der Laufzeit des Programmes auftritt und gemeldet wird

schränkte Anzahl von Bezeichnern als Wertebereich angibt. Diese Bezeichner werden intern der Reihe nach durchnumeriert.
Allgemein sieht das so aus:

```
TYPE <Bezeichner> = (<Bezeichner> {,<Bezeichner>});
```

Beispiel: Definition eines Aufzählungstyps:

```
...
TYPE
  Monate = (Januar, Februar, Maerz, April, Mai,
            Juni,Juli, August, September,
            Oktober, November, Dezember);
...
```

Diese Deklaration erzeugt einen Typen, dessen Wertebereich die Werte Januar, Februar, ... enthält. Dies sind jedoch keine Textstücke (`STRINGS`), sondern Werte, die diesen Namen tragen. Es ist möglich, Variablen dieses Typs miteinander zu vergleichen, einen Wert vor- oder zurückzugehen oder als Laufvariable in einer FOR-Schleife zu verwenden (s. u.).

Die Numerierung der Elemente eines Aufzählungstyps kann unterbrochen und mit einem neuen Wert fortgesetzt werden:

```
FileAccess=(readWrite=1004,readOnly=1005,newFile=1006);
```

oder auch:

```
FileAccess=(readWrite=1004,readOnly,newFile);
```

Von derartigen Typen lassen sich auch Unterbereiche bilden:

```
TYPE {<Bezeichner> = [ <KonstAusdruck>..<KonstAusdruck> ]}
```

Beispiel:

```
TYPE
  Monate = (Januar, Februar, Maerz, April, Mai,
            Juni,Juli, August, September,
            Oktober, November,Dezember);
  Sommer = [Juli..September];
  Winter = [Januar..Maerz];
VAR
  m : Monate;
  s : Sommer;
  w : Winter;
...
```

Diese Typen erleichtern dem Programmierer das Leben dadurch, daß er selbst weniger Überprüfungen vornehmen muß, da diese zum großen Teil das Laufzeitsystem[25] des Compilers übernimmt.

Ist also bei einer Variablen klar, daß sie während des Programms nur bestimmte Werte annehmen kann, sollte man einen Unterbereichstyp verwenden, da das Laufzeitsystem eine Bereichsverletzung (Stichwort RangeCheck) meldet, falls versehentlich ein falscher Wert zugewiesen wird.

Aufzählungstypen erhöhen vor allem die Lesbarkeit eines Programms. Außerdem kann durch sie sichergestellt werden, daß nur bestimmte Konstanten verwendet werden, z. B. Parameter für Prozeduren (Stichwörter OpenFile, Accessmode).

[25] Das Laufzeitsystem ist ein spezielles Programm, welches vom Compiler zum fertigen Programm hinzugefügt wird. Es hat u. a. die Aufgabe, Fehler, welche während der Ausführung des Programmes auftreten, abzufangen (z. B. „Division durch Null")

3.8 Kommentare und Anmerkungen

In jeden Quelltext gehören Kommentare, damit man später noch weiß, was man sich an dieser oder jener Stelle einmal gedacht hat. Kommentare lassen sich in **Cluster** an jeder Stelle in beliebiger Länge einfügen und können auf zwei verschiedene Arten in den Programmtext eingefügt werden. In der einen Art läutet man einen Kommentar mit der Zeichenfolge „`(*`" ein, wobei man anschließend durch „`*)`" dem Compiler mitteilen muß, wo die Bemerkung wieder beendet ist. So ein Kommentar kann zwischen zwei Anweisungen geschoben werden, sich aber auch über mehrere Zeilen erstrecken. Außerdem können Kommentare, anders als in C, geschachtelt werden, um damit Programmteile zu Testzwecken auszuklammern. In anderen Programmiersprachen steht man dann oft vor einem großen Stück Arbeit.

Bei der anderen Art setzt man ein „|" und zeigt damit an, daß man von hier bis zum Ende der Zeile einen Kommentar setzen möchte. Kommentare werden vom Compiler ignoriert, haben also keinerlei Einfluß auf den Verlauf des Programms.

Diejenigen, welche schon Erfahrung in BASIC gesammelt haben, dürfen zur Erleichterung feststellen, daß eventuell vorhandene Kommentare im Gegensatz zu interpretiertem BASIC die Geschwindigkeit des Programmes nicht herabsetzen. Also sollte man die Möglichkeit der Kommentargebung reichlich in Anspruch nehmen.

Ein kleines Programm soll Ihnen die richtige Anwendung der Kommentierungszeichen sichtbar machen (siehe Abbildung 3.1).

Natürlich braucht man nicht allem und jedem einen Kommentar an die Seite zu stellen. Finden Sie selber das richtige Maß und schrecken Sie nicht davor zurück, lieber zu viele als zu wenige Bemerkungen in Ihrem Text unterzubringen. Denken Sie aber daran, daß Sie ein Programm und keinen Roman schreiben.

```
MODULE Addition;
FROM InOut IMPORT WriteInt, WriteLn;

VAR a,b,c :  INTEGER;

| hier werden drei Variablen deklariert
BEGIN

   a:=10; |hier wird 'a' der Wert 10 zugewiesen
   b:=23; |der Variablen b wird der Wert 23 zugewiesen
   c:=a+b; (* a und b werden addiert *)
   (*
     Dies ist ein mehrzeiliger
     Kommentar
   *)
   WriteInt(c); (* das Ergebnis wird ausgegeben *)
   (* Geschachtelte Kommentare (* Hallo *) sind
     auch möglich *)
   WriteLn;
END Addition.
```

Abbildung 3.1: Beipiele für Kommentierung

Kommentare sind Lesehilfen und haben dokumentarischen Charakter.

3.9 Was sind Schlüsselwörter?

In fast allen Programmiersprachen sind Zeichenfolgen definiert, die eine in der Sprache genau festgelegte Bedeutung haben, wie z. B. die Zeichenfolgen **MODULE**, **VAR**, **BEGIN** usw., die Sie bereits in den Beispielprogrammen gesehen haben.

Solche Zeichenfolgen bezeichnet man als *Schlüsselwörter* der Programmiersprache. Sie dürfen i. d. R., so auch bei **Cluster**, nicht als Bezeichner[26] in Programmen verwendet werden. Man spricht in diesem Fall auch von „reservierten Wörtern" der Programmiersprache.

[26]Bezeichner (engl. identifier): sind z. B. Namen für Variablen, Konstanten und Prozeduren, wobei wir auf letztere noch zu sprechen kommen

3.10 Aufbau eines Cluster-Programms

Sie haben es sicher schon vermutet: Strukturiert natürlich! Die Schemata in Abbildung 3.2 sollen Ihnen eine kleine Übersicht über die Grobstruktur eines **Cluster**-Programms geben.

```
MODULE ...

FROM ...  IMPORT

TYPE

CONST

VAR

BEGIN
...
END ...  .
```

Abbildung 3.2: Programm-Schemata

Die Reihenfolge der einzelnen Deklarationsteile kann beliebig variiert werden. Ein mehrfaches Auftauchen einzelner Teile ist ebenfalls möglich. In anderen Worten, **CONST**-Deklarationen könnten beispielsweise auch nach den **VAR**-Deklarationen erfolgen.

Etwas ausführlicher sieht dieses Schema wie folgt aus:


```
MODULE <Programmname>;

{FROM <Bibliotheksmodul> IMPORT <Bezeichner>
{,<Bezeichner>};}

[VAR <Variablenname>:<Variablentyp>;
    {<Variablenname>:<Variablentyp>;}]

{PROCEDURE <Prozedurname>;
[VAR <Variablenname>:<Variablentyp>;
   {<Variablenname>:<Variablentyp>;}]

BEGIN
  <Anweisungsfolge>
END <Prozedurname>;}

BEGIN
  <Anweisungsfolge>
[CLOSE
  <Anweisungsfolge>]
END <Programmname>.
```

Hier soll Ihnen kurz die hier verwendete Schreibweise mit den verschiedenen Klammerarten erläutert werden, welche im Programmtext dann natürlich nicht (!) erscheinen: Die spitzen Klammern (`< >`) umschließen Modul- oder andere Namen, die benutzerdefiniert bzw. in der Bibliothek der Standardmodule festgelegt sind. Geschweifte Klammern (`{ }`) stehen für keine oder beliebig häufige Wiederholung des geklammerten Inhalts. Eckige Klammern (`[ ]`) bedeuten, daß der entsprechende Text weggelassen werden kann.

Dieses Schema braucht Sie nicht in Verwirrung zu stürzen. Es ist einfacher, als Sie vielleicht denken mögen.

3.10.1 Der BEGIN-Teil

Mit dem Schlüsselwort „**BEGIN**" beginnt nun im wahrsten Sinne des Wortes das eigentliche Programm – der Hauptteil. In ihm sind die Anweisungen, die zur Lösung des betroffenen Problems notwendig sind, nacheinander aufgelistet. Dieser Teil wird beim Programmstart ausgeführt. Er endet entweder mit dem Schlüsselwort „**CLOSE**" (s. „Der CLOSE-Teil" unter 3.10.2) oder mit dem Schlüsselwort „**END**", welches das Programm abschließt.

3.10.2 Der CLOSE-Teil

Dieser Programmteil gelangt dann zur Ausführung, wenn das eigentliche Programm beendet wird. Sei es wegen der abgeschlossenen Abarbeitung des BEGIN-Teils, sei es wegen eines Programmabruchs durch den Benutzer oder aufgrund eines Fehlers. Üblicherweise befinden sich hier spezielle Anweisungen, die reservierten Speicher wieder freigeben o. ä., die uns im Moment nicht interessieren sollen.

Eingeleitet wird dieser Teil mit dem Schlüsselwort „**CLOSE**", beendet wird er durch „**END**".

3.10.3 Die IMPORT-Zeilen

Die Programmiersprache **Cluster** unterstützt – wie auch Modula 2, **Cluster** stammt ja davon ab – das Zerlegen von Programmen in Module (s. Was versteht man unter „strukturierter Programmierung"? unter 3.4.1).

Der Vorteil daran ist, wie bereits oben angedeutet, daß nicht nur ein, sondern viele verschiedene Programme auf solche ausgelagerten Programmteile zugreifen und sie nutzen können. Stellen Sie sich z. B. vor, Sie müssen in Ihren Programmen immer wieder

Kreise zeichnen. Dann wäre es sinnvoll, diesen Programmteil in ein Modul auszulagern und bei Bedarf zu IMPORTieren. Jetzt können alle Programme, die einen Kreis zeichnen sollen oder wollen, auf dieses Modul zugreifen, und Sie sparen sich eine Menge Arbeit, indem Sie nicht immer wieder dasselbe neu programmieren müssen. Ist doch gut, oder?

Sie sollten bei der Gestaltung der Schnittstelle, d. h. der Definition jener Variablen, die jeweils übergeben werden müssen, auf eine sinnvolle Lösung achten. Sie sollte allgemein gültig und somit für viele Fälle verwendbar sein. Ferner macht es Sinn, Programmteile, die z. B. jeweils graphische Funktionen übernehmen, in einem Modul zusammenzufassen.

Für den hier genannten Fall des Kreisezeichnens ist es allerdings nicht mehr notwendig, eine solche Prozedur (Programmteil) selber zu basteln. **Cluster** kommt mit einer ganzen Reihe von Standardmodulen ins Haus, auf die direkt zugegriffen werden kann. Die Beschreibung dieser Programmteile finden Sie weiter hinten in diesem Handbuch unter Kapitel 7. Also: einfach nur zugreifen!

Wie macht man das nun? Der Compiler muß ja wissen, welche Typen, Variablen und Prozeduren ein Programm benutzen will und in welchen Modulen sie zu finden sind. Dazu gibt es die IMPORT-Zeilen, deren Syntax so aussieht:

```
FROM <Modulname> IMPORT <Objektname> {,<Objektname>};
```

Sie können durch Wiederholung dieser Anweisung von beliebig vielen weiteren Modulen Prozeduren, Konstanten, Variablen und Typen importieren – ohne Ende sozusagen.

Sollen aus einem Modul alle Prozeduren oder Konstanten, Variablen usw. importiert werden, so schreibt man einfach:

```
IMPORT <Modulname> {,<Modulname>};
```

Da dies zu Namenskonflikten[27] führen kann, spricht man hier von einem unqualifizierten Import. Das bedeutet, daß auf die importierten Module nicht direkt über ihren Namen zugegriffen werden kann, sondern über den Modulnamen in Verbindung mit dem Objektnamen, womit Namenskonflikte vermieden werden können.

Gibt es im Modul „`Graph`" z. B. ein Objekt „`DrawPixel`", so kann mit „`Graph.DrawPixel`" darauf zugegriffen werden.

Sie können sich denken, daß das bei der Länge mancher Modul- und Objektnamen schnell zur Hölle werden kann. Deshalb kann der Name beim Import wie folgt geändert werden:

```
IMPORT <Modulname> [AS <NeuerName>];
```

Man sollte noch wissen, daß bei einem qualifizierenden Import alle nicht genannten Objekte automatisch unqualifiziert importiert werden. Auch dabei ist es möglich, den Modulnamen nach eigenen Wünschen zu ändern:

```
FROM <Modulname> [AS <NeuerName>] IMPORT ...  ;
```

Beispiel:

Aus dem Modul „`InOut`" werden die Objekte „`WriteString`" (Textausgabe) und „`WriteLn`" (Zeilenvorschub) qualifizierend, alle anderen Prozeduren (z. B. „`ReadInt`" (Eingabe einer Integerzahl), „`WriteInt`" (Ausgabe einer Integerzahl) usw.) unqualifiziert importiert:

[27] Wenn zwei oder mehrere Bezeichner, Prozedurnamen, Variablen usw. die gleiche Bezeichnung haben, kommt es zu s. g. Namenskonflikten


```
...
FROM InOut AS io IMPORT WriteString,
WriteLn;
...
```

Auf diese Objekte kann nun auf verschiedene Weisen zugegriffen werden:

```
WriteString
WriteLn
io.WriteString(Test") |(io.  ist hier überflüssig,
                      | aber möglich)
io.ReadInt(i)
io.WriteInt(5)
...
```

Nicht erlaubt ist

```
Read(c)
Write("a")
...
```

weil der Compiler hier nicht weiß, wo er `Read` oder `Write` herholen soll!

Übungen 2:

1. Was erreicht folgende Codezeile?

   `WriteString("Ich liebe \Cluster{}!");`

2. Schreiben Sie ein kurzes Programm, welches zwei INTEGER-Zahlen einliest und die Summe von beiden ausgibt.

3. Wofür ist der CLOSE-Teil eines Programmes oder Moduls gut?

4. Welchen Unterschied gibt es zwischen einem CARDINAL und einem INTEGER-Wert?

5. Welchen Typ haben die folgenden Konstanten?

   (a) `FALSE`

   (b) `-50000`

   (c) `45.78`

   (d) `200`

   (e) `'L'`

   (f) `"Hallihallo"`

6. Wie deklariert man eine Konstante `TestWert` von 70000?

7. Welcher der folgenden Ausdrücke ist wahr (TRUE)?

   (a) `25 < 25`

   (b) `NOT (7 < 9)`

   (c) `(5 = 4) OR (5 = 5)`

   (d) `40 = 40`

8. Schreiben Sie ein Programm, welches eine REAL-Zahl nach INTEGER wandelt und dann wieder zurück. Eine Bildschirmausgabe muß nicht unbedingt erfolgen.

3.11 Besonderheiten von Cluster

Neben der Case-Sensitivität , die eine strikte Unterscheidung zwischen Groß- und Kleinschreibung in der Interpretation durch den Compiler bedeutet, ist eine weitere Besonderheit von **Cluster** die Formatfreiheit . Diese besagt, daß die Form, in welcher ein Programm geschrieben ist, seinen Inhalt nicht beeinflußt. An den Beispielen, die noch folgen werden, ist diese Eigenschaft gut zu erkennen.

3.11.1 Das Semikolon

An dieser Stelle noch eine Bemerkung zum Semikolon: Ein Semikolon trennt Anweisungen voneinander. Daher ist vor einem abschließenden **`END`** ein Semikolon nicht erforderlich, denn **`END`** ist ja keine Anweisung, sondern ein Schlüsselwort. Sollten Sie dort trotzdem ein Semikolon setzen, ist das nicht weiter tragisch, da es in **Cluster** die sogenannte Leeranweisung gibt, die – sofern überhaupt – aus keinem Befehl besteht. Klingt komisch und direkt unheimlich, darum hier einige Beispiele:
Folgende (korrekte) Anweisungsfolgen enthalten Leeranweisungen:

```
BEGIN END (enthält 1 Leeranweisung)

BEGIN
  a:=b; (beinhaltet ebenfalls 1 Leeranweisung)
END

BEGIN ; ; END (3 (!)  Leeranweisungen)
```

3.11.2 Eingabe von Steuerzeichen

Zeichen wie `"A"`, `"9"`, ...sind allesamt Zeichen, welche über Tastatur eingegeben werden können. Sie gehören zu den sogenannten

druckbaren Zeichen. Es gibt aber eine Zahl von Zeichen, die nicht einfach von Tastatur eingegeben werden können. Hierfür kann eine besondere Eingabeform von **Cluster** verwendet werden:

```
&xxx
```

Jedes Zeichen kann durch seinen ASCII-Code mit vorgestelltem „`&`" erreicht werden, beispielsweise „`&27`" für Escape[28].

3.12 Die ersten Schlüsselwörter

3.12.1 Einfache Strukturen in Cluster

Die grundlegende Struktur in **Cluster** ist die Anweisungssequenz, d. h. eine Folge von Anweisungen, die durch Semikola voneinander getrennt werden.

Anweisungen können z. B. einfache Zuweisungen, Schleifen oder Prozeduraufrufe sein. Wenden wir uns zunächst den Schleifen zu:

3.12.2 Schleifenstrukturen: Was eine Schleife ist

Eine Schleife erlaubt es, eine Folge von Anweisungen mehrmals aufzurufen bzw. zu wiederholen. Am Anfang oder am Ende der Schleife ist die Kontrollinformation für die Anzahl der Schleifendurchläufe enthalten. Die folgenden Erläuterungen werden dies verdeutlichen:

[28] Escape oder ESC wird häufig für Druckeransteuerungen benötigt

3.12.2.1 Die REPEAT..UNTIL-Struktur

Dies ist die einfachste Schleifenstruktur in **Cluster**. Die Syntax dieser Struktur sieht wie folgt aus:

Syntax:

```
REPEAT
  <Anweisungsfolge>
UNTIL <Boolausdruck>; | (Wahrheitswert)
```

Beschreibung:

Der von **REPEAT** und **UNTIL** eingeschlossene Programmteil (Schleifenkörper), wird solange wiederholt, bis der Boolesche Ausdruck wahr (TRUE) wird. In jedem Fall wird der Programmteil einmal durchlaufen, da die Abbruchbedingung erst am Ende der Schleife steht. Beachten Sie bitte, daß der boolesche Ausdruck erst am Ende der Schleife berechnet wird.

Dieser Umstand kann bei manchen Schleifen u. U. sehr wichtig sein. Besteht der Boolausdruck aus der Überprüfung einer Variablen auf einen bestimmten Wert, so sollte man darauf achten, daß diese Variable ihren Wert innerhalb der Schleife auch ändert, da es ansonsten vorkommen kann, daß die Abbruchbedingung nie erfüllt und die Schleife somit unendlich oft ausgeführt wird (Endlosschleife). Zum besseren Verständnis wieder ein Beispiel:

Beispiel:

```
...
Anzahl:=5;
Wert:=12;
Potenz:=1;
REPEAT | korrekte Schleife!
```



```
    Potenz:=Potenz*Wert;
    Anzahl:=Anzahl-1
UNTIL Anzahl=0;|Abbruchbedingung
...

...
REPEAT | korrekte Schleife!
    a:=a+2;
    b:=b*0.5;
    s:=a+b
UNTIL s>Grenze;
...

...
REPEAT
| Diese Schleife kann zur Endlosschleife werden!
    s:=s*2;
| (je nach Wert von s und gesamtwert bei Beginn
| der Schleife)
    c:=gesamtwert/s
UNTIL c>10;

...
REPEAT
    c:=c-4; | wird zur Endlosschleife, wenn b nicht
    a:=a+2 | von Anfang an 5 ist!
UNTIL b=5;
...
```

Anmerkung: Es ist üblich, die Anweisungsfolge innerhalb der Schleife einzurücken, um sie deutlicher hervorzuheben. Das unterstreicht die Struktur Ihres Programms und erhöht die Übersichtlichkeit.

Wie sie an den Beispielen sehen, ist es nicht unbedingt notwendig, vor dem **UNTIL** ein Semikolon zu setzen, da das **REPEAT UNTIL** der Rahmen ist, der die Anweisungssequenz umgibt; es ist aber natürlich nicht verboten, da es ja noch die leere Anweisung gibt.

3.12.2.2 Die WHILE..DO..END-Struktur

Syntax:

```
WHILE <Boolausdruck> DO
  <Anweisungsfolge>
END
```

Beschreibung:
Im Gegensatz zur `REPEAT`-Struktur steht bei der `WHILE`-Konstruktion die Abbruchbedingung am Anfang der Schleife. Darum kann die Schleife einfach übersprungen werden, wenn die Abbruchbedingung erfüllt ist, und braucht nicht ein einziges Mal zur Ausführung zu kommen. Die `WHILE`-Schleife wird solange ausgeführt wie die Bedingung erfüllt ist. Die Schleife wird dann nicht mehr erneut ausgeführt, wenn die Auswertung der Bedingung (Boolausdruck) ein `FALSE` ergibt.

Beispiel:

```
...
| an dieser Stelle erfolgt die Eingabe von 'Anzahl'
WHILE Anzahl # 0 DO
    c:=a*b+2;
    s:=s+c;
    DEC(Anzahl)
END;
...
...
WHILE c # 1 DO | kann zur Endlosschleife werden,
    c:=c/2; | wenn c zu Beginn negativ ist.
    s:=s+c
END;
...
```


3.12.2.3 Die FOR..DO..END-Struktur

Syntax:

```
FOR <Variable>:=<Ausdruck> TO <Ausdruck>
[BY <Konstante>] DO
  <Anweisungsfolge>
END
```

Beschreibung:
Auch diese Struktur ist eine Schleifenstruktur. Im Unterschied zu den beiden vorherigen Konstrukten steht hier jedoch bereits vor Ausführung der Schleife fest, wie oft sie wiederholt werden soll. Falls Sie bisher in BASIC programmiert haben, so sollte Ihnen die `FOR`-Schleife vertraut sein, denn sie ist sehr ähnlich wie die `FOR/NEXT`-Schleife in BASIC aufgebaut.

Wenn Sie noch einmal die allgemeine Form oben betrachten, dann bedeutet `<Variable>` die Schleifensteuerungsvariable vom Typ `CARDINAL` oder `INTEGER` (natürlich auch die kürzeren und längeren Formen), sowie Aufzählungstypen und `CHAR`. `<Ausdruck>` ist ein `CARDINAL`- oder `INTEGER`-Ausdruck und `<Anweisungsfolge>` sind ein oder mehrere zu wiederholdende Befehle. Die Befehle, die wiederholt ausgeführt werden, heißen Schleifenkörper. Der BY-Abschnitt ist optional. Im allgemeinen wiederholt die FOR-Schleife ihren Code, bis der Wert der Steuerungsvariablen größer dem Wert des Zielausdrucks ist. Falls `BY` fehlt, wächst die Steuerungsvariable bei jedem Schleifendurchlauf um eins. Ansonsten nimmt die Steuerungsvariable um den Wert des BY-Ausdrucks zu oder ab, falls die Konstante hinter `BY` negativ ist. Der erste Ausdruck gibt den Startwert wieder, der zweite Ausdruck den Endwert der Schleifenzählung. Sind Startwert und Endwert gleich, wird die Schleife einmal durchlaufen. Ist der Startwert größer als der Endwert, und keine negative Schrittweite gewählt, wird die Schleife nie durchlaufen. Genauso bei negativer Schrittweite und einem Startwert der kleiner als der Endwert ist. Wichtig: Der Wert hinter `BY` muß eine Konstante sein. Die Werte für den Start- und Endwert können Variablen sein.

Beispiel: Berechnung der Summe der ersten n natürlichen Zahlen:

```
ReadInt(n); | Einlesen von n
summe:=0;
FOR i:=1 TO n DO
    summe:=summe+i
END;
...
| Berechnung der Summe der natürlichen,
| ungeraden Zahlen bis n
ReadInt(n);
summe:=0;
FOR i:=1 TO n BY 2 (* Schrittweite +2 *) DO
    summe:=summe+i
END;
...
| Berechnung von n!  (Fakultät)
ReadInt(n);
produkt:=1;
FOR i:=n TO 2 BY -1 (* Schrittweite -1 *) DO
    produkt:=produkt*i
END;
```


3.12.3 Verzweigungsstrukturen: Die IF..THEN..ELSE - Struktur

Syntax:

```
IF <Boolausdruck> THEN
   <Anweisungsfolge1>
ELSE
   <Anweisungsfolge2>
END
```

Beschreibung:
Bei dieser Struktur wird über den Booleschen Ausdruck entschieden, welcher Programmteil im folgenden weiterbearbeitet werden soll. In der einfachsten Form dieser Struktur existiert nur eine einzige Anweisungsfolge, die dann ausgeführt wird, wenn der Boolausdruck `TRUE` wird. In diesem Fall fällt der **ELSE**-Zweig weg.
Beispiel:

```
IF Gehalt<3000 THEN
  Gehalt:=Gehalt+100
END;
```

Hier wird z. B. einem Angestellten ein Sonderzuschlag gewährt, wenn er monatlich weniger als DM 3000.– verdient.

Diesen Vorgang könnte man mit Hilfe eines **ELSE**-Zweiges noch weiter differenzieren:

```
IF Gehalt<3000 THEN
   Gehalt:=Gehalt+100
ELSE
   Gehalt:=Gehalt+50
END;
```

Wer nicht in den Genuß einer Gehaltserhöhung von DM 100,– kommt, weil er DM 3000,– oder mehr verdient, muß mit einem Zuschlag von nur DM 50,– vorliebnehmen.
Für die **IF**..**THEN**-Struktur existiert auch die folgende Variante:

```
 IF <Boolausdruck> THEN <Anweisungsfolge>
{OR_IF <Boolausdruck> THEN <Anweisungsfolge>}
[ELSE <Anweisungsfolge>]
 END
```

Zu kompliziert? – Überhaupt nicht. Zur Erinnerung, die geschweiften Klammern deuten an, daß ihr Inhalt beliebig oft wiederholt, aber auch ganz fortgelassen werden kann. Die eckigen Klammern stehen um einen Text, den man beliebig aussparen kann; in diesem Beispiel das **ELSE**.

Nun jedoch zum eigentlichen Thema: Die oben dargestellte Variante ist für den Fall gedacht, daß nicht nur ein oder zwei, sondern viele verschiedene Fälle abgeprüft werden sollen. Beim Ausführen der Struktur wird jeder `<Boolausdruck>` der Reihe nach geprüft, bis einer `TRUE` ergibt.
Wurde ein solcher Ausdruck gefunden, wird die nachfolgende `<Anweisungsfolge>` ausgeführt, sonst der **ELSE**-Teil.

Beispiel:
Angenommen, alle Mitarbeiter von oben sollen DM 100,– mehr Gehalt bekommen, wenn 24 sind, DM 200,–, wenn sie 25 Jahre alt sind, und DM 300,– mehr, wenn sie über 25 Jahre alt sind. Alle anderen müssen sich mit DM 50,– zufrieden geben (das ist nicht gerade ein Beispiel, welches aus dem Leben gegriffen ist, aber trotzdem... Schicken Sie uns ein schöneres, wenn Ihnen zufällig eines über den Weg läuft). Im Programmtext sähe das so aus:


```
IF    Alter = 24 THEN Gehalt:=Gehalt+100
OR_IF Alter = 25 THEN Gehalt:=Gehalt+200
OR_IF Alter > 25 THEN Gehalt:=Gehalt+300
ELSE
  Gehalt:=Gehalt+50
END;
```

Natürlich gibt es nicht nur **OR_IF**, sondern auch dazu passend ein **AND_IF**. Mit dessen Hilfe ist es möglich, einen komplizierten Boolean-Ausruck in mehrere Fälle aufzugliedern. Weiterhin ergibt sich hiermit oft auch eine Beschleunigung des Programms. Wird eine Konstruktion der Form

```
IF <Bool 1> AND_IF <Bool 1.1> ...
```

verwendet, so muß der zweite IF-Fall nur überprüft werden, wenn der erste `TRUE` ist. Dies führt wie bei **AND**, zu Geschwindigkeitssteigerungen, insbesondere dann, wenn diese Abfragen innerhalb einer Schleife ausgeführt werden müssen.

Bedenken Sie, daß bei `AND_IF`- und `OR_IF`-, wie auch bei der normalen IF-Schleife, ein **ELSE**-Zweig möglich ist. Hiermit sind diese Strukturen sehr flexibel einsetzbar und es lassen sich so sehr übersichtliche Eintscheidungsabläufe aufstellen. Im Gegensatz zu einem einfachen **AND** ist mit **AND_IF** folgende Konstruktion möglich:

```
IF a=1
  AND_IF b>3 THEN
    WriteString("Beide Bedingungen erfüllt");
    WriteLn;
  ELSE
    WriteString("b<=3");
  END
ELSE
  WriteString{"a#1");
END
```

Man sieht, abhängig davon, welche Bedingung nicht zutrifft, wird ein anderer **ELSE**-Teil ausgeführt.

Statt langer Worte werden Ihnen nun einige weitere Beispiele mit kurzen Erläuterungen gezeigt, die Ihnen die Funktionsweise und die Anwendungsmöglichkeiten dieser Struktur veranschaulichen sollen.

Beipiele:
In folgendem Beispiel wird der Fall dargestellt, daß ein Wert, der ein Eingabe- oder auch ein berechneter Wert sein kann, als eine Art von Schlüssel für bestimmte auszuführende Operationen dient. Dies kann auf verschiedene Art und Weisen formuliert werden, was je nach dem zu besserem oder schlechterem Code führt, den der Compiler erzeugt, und zudem in Übersichtlichkeit und Geschwindigkeit verschieden ist.

```
IF (a=1) OR (a>=-10) AND (a<=-5) THEN ...
OR_IF (a>=2) AND (a<=10)THEN ...
OR_IF (a=11) OR(a=20)THEN ...
ELSE ...
END;
```

Diese Version ist sehr aufwendig und führt daneben auch zu schlechtem Objektcode, da der Compiler noch nicht erkennen kann, daß

hier sehr häufig nur mit dem `a` geprüft wird. Eine sinnvolle Vereinfachung stellt der **OF**-Vergleichsoperator dar. Mit diesem Operator kann eine ganze Werteliste angegeben werden, mit welchen verglichen werden soll.

```
<Ausdruck> OF <Ausdruck> [..<Ausdruck>]
              ,<Ausdruck> [..<Ausdruck>]
```

Hierzu zwei Beispiele:

```
IF c OF "a","b","c" THEN

IF i OF 1,3,4..6 THEN
```

Im ersten Beispiel ist die Bedingung erfüllt, wenn `c` den Wert „a“„b“ oder „c“ hat. Im zweiten wenn `i` den Wert `1` oder `3` hat oder zwischen `4` und `6` liegt. Will man eine Bedingung so formulieren, das sie dann wr sein soll, wenn eine Variable keiner der folgenden Werte entspricht, kann dies folgendermaßen geschehen:

```
IF c NOT OF "a","b","c" THEN
```

Außerdem können Variablen in einer **OF**-Liste angegeben werden.

```
... x OF a..b, c ...
```

Das Beispiel von oben sieht nun so aus:

```
IF a OF 1,-10..-5 THEN ...
OR_IF a OF 2..10 THEN ...
OR_IF a OF 11,20 THEN ...
ELSE ...
END;
```

Der Term (`a OF 1,-10..-5`) ist dabei ein boolscher Ausdruck. Er läst sich also bei Bedarf auch mittels boolscher Verknüpfungen mit anderen boolschen Ausdrücken kombinieren.

Aber auch diese Version ist dann noch nicht vollkommen befriedigend, wenn viele verschiedene Fälle mit einem Schlüssel zu prüfen sind. Deshalb bietet **Cluster** die Möglichkeit, den Ausdruck als Schlüssel zu erklären, mit welchem dann mehrere Listen geprüft werden können:

```
IF KEY <Ausdruck>
  OF <OfListe1> THEN ...  END
  OF <OfListe2> THEN ...  END
  OF <OfListe3> THEN ...  END
ELSE ...
END;
```

Es besteht weiterhin die Möglichkeit, einen solchen Schlüssel in einem OR_IF-Fall zu verwenden. Anstelle des **THEN** kann eine weitere Fallunterscheidung mit **AND_IF** folgen:


```
...
OR_IF KEY <Ausdruck>
      OF <OfListe> AND_IF <Ausdruck1> THEN ...
                   OR_IF <Ausdruck2> THEN ...
                   OR_IF KEY ...
                   ELSE ...
                   END
      OF ...
OR_IF ...
...
```

Das Beispiel von oben sieht dann so aus:

```
IF KEY a
  OF 1,-10..-5 THEN ...  END
  OF 2..10    THEN ...  END
  OF 11..20   THEN ...  END
ELSE ...
END;
```

Das sieht nun doch wesentlich übersichtlicher und schöner aus als oben.

Folgendes Programm prüft, ob ein Zeichen in `c` ein großer bzw. kleiner Buchstabe oder eine Zahl ist:

```
  IF KEY c
   OF "a".."z","ä","ö","ü","ß" THEN
     WriteString("Kleiner Buchstabe");
     (* --> gibt einen Text aus *)
     WriteLn
     (* --> Zeilenvorschub *)
   END
   OF "A".."Z","Ä","Ö","Ü" THEN
     WriteString("Großer Buchstabe");
     WriteLn
   END
   OF "0".."9" THEN
     WriteString("Zahl");
     WriteLn
   END
 ELSE
   WriteString("Unklares Zeichen");
   WriteLn
 END;
```

Sie sehen es: Mit **IF**..**THEN** hat man eine sehr mächtige und vielfältige Kontrollstruktur in der Hand, die einem sehr viele Möglichkeiten bietet. Probieren Sie es aus. Wie Sie Zeichen eingeben können, die nicht darstellbar sind, lesen Sie bitte unter 3.11.2 auf Seite 50 nach.

3.12.4 Weitere Schleifenarten

3.12.4.1 Zum zweiten Male: Die WHILE-Struktur

Syntax:

```
WHILE <Boolausdruck> DO ...
OR_WHILE <Boolausdruck> DO ...
OR_WHILE <Boolausdruck> DO ...
END;
```

Beschreibung:
Wie man sieht, ist die **WHILE**-Struktur mächtiger, als sie zuerst oben angegeben wurde. In Wirklichkeit verfügt sie über alle Fähigkeiten, die auch die **IF**-Struktur besitzt.

Nacheinander werden alle Boolausdrücke geprüft, bis einer TRUE wird. Die dazugehörige Anweisungsfolge wird ausgeführt; anschließend wird die Schleife beim obersten Vergleich wieder begonnen. Ist keiner der Boolausdrücke wahr, wird die Schleife verlassen.

Auch hier ist ein **ELSE**-Zweig vorhanden, welcher dann ausgeführt wird, wenn keiner der angegebenen Ausdrücke wahr ist. Zusätzlich wird die **WHILE**-Schleife anschließend verlassen. Dies ist sehr wichtig.

Überdies ist auch hier – entsprechend der **AND_IF**-Konstruktion – die Verwendung von **AND_WHILE** und von **KEY** möglich.

Syntax:

```
WHILE <BoolAusdruck1>
  AND_WHILE  <BoolAusdruck2> DO
    Anweisung1
  ELSE
    Anweisung2
  END
ELSE
  Anweisung3
END
```

Bei einer solchen Schleife geschieht folgendes: `Anweisung1` wird solange ausgeführt, wie `BoolAusdruck1` und `BoolAusdruck2` wahr sind. Wird die Schleife aufgrund von BoolAusdruck1 verlassen, wird der ELSE-Teil mit der `Anweisung3` ausgeführt. Wird sie über den `BoolAusdruck2` verlassen, wird `Anweisung2` ausgeführt. Sie sehen, im Gegensatz zu einer `WHILE`-Schleife mit durch `AND` verknüpften Boolausdrücken, muß man in diesem Fall nicht nach der Schleife mit einer `IF`-Abfrage herausfinden, wodurch die Schleife abgebrochen worden ist.

Selbstverständlich läßt sich ein `AND_WHILE`-Zweig auch wieder mit beliebig vielen `OR_WHILE`s oder mit einem `KEY` versehen.

Syntax für `WHILE KEY`:

```
WHILE KEY <Ausdruck>
  OF <Ausdruck1> DO
                   <Anweisung1>
                 END
  OF <Ausdruck2> DO
                   <Anweisung2>
                 END
ELSE
  <Anweisung3>
END
```

Diese Schleife wird solange durchlaufen, wie einer der `OF`-Fälle zutrifft. Trifft keiner zu, wird der `ELSE`-Teil durchlaufen, und die Schleife verlassen.
Auch hier läßt sich jeder `OF`-Fall mit einem `AND_WHILE` kombinieren:

```
WHILE KEY <Ausdruck>
  OF <Ausdruck1> AND_WHILE <BoolAusdruck> DO
                   <Anweisung1>
                 END
  OF <Ausdruck2> DO
                   <Anweisung2>
                 END
ELSE
  <Anweisung3>
END
```

Warscheinlich erkennen Sie nun die Mächtigkeit dieser Schleife, normalerweise sollte es damit nie notwendig sein, `LOOP` (s. u.) zu verwenden. Im gesammten Compiler wurde nicht ein einziges Mal `LOOP` verwendet.

3.12.4.2 Der Notnagel: LOOP

`LOOP` ist eines von den unfeinen Dingen in der Welt der Software, und ein guter Programmierer rümpft beim Anblick dieses Konstruktes die Nase. Die Verwendung dieser Schleife sollte sich auf absolute Notfälle beschränken. In der Programmiersprache C wird diese Art von Schleifenform eher häufig verwendet. Erst im Zusammenhang mit Exceptions (siehe unter 3.20.2 ab Seite 126) erhält **`LOOP`** wieder eine Berechtigung.

Syntax:

```
LOOP
  <Anweisungsfolge>
END;
```

Beschreibung:
Hier steht die Abbruchbedingung nicht am Anfang oder am Ende des Schleifenkörpers, sondern innerhalb desselben. Die Schleife kann also jederzeit (mittels **`EXIT`**) verlassen werden, was zwar kaum der Philosophie der strukturierten Programmierung entspricht, aber der Einfachheit wegen leider manchmal zur Anwendung verleitet.

Vermeiden Sie bitte diese Möglichkeit (das geht ohne Probleme), und nehmen Sie folgende Darstellung als abschreckendes Beispiel:


```
...
a := 0;
LOOP
  INC(a);
  IF a = b THEN EXIT END;
  DEC(b);
END;
...
```

3.12.5 Die WITH-Struktur

Neben den bisher aufgeführten Strukturen gibt es noch eine, die dem Programmierer die Arbeit erleichtern soll: die WITH-Struktur. Mit der WITH-Struktur kann einer tiefliegenden Struktur für einen Programmbereich ein eigener Name zugeordnet werden.

Vorsicht für alle, die „**`WITH`**" schon von Pascal oder Modula 2 her kennen. In **Cluster** ist die Funktion verändert und gleichzeitig verbessert worden: Durch die Umbenennung mit **`AS`** ist es nun nämlich möglich innerhalb eines **`WITH`** auch auf zwei Recordvariablen des gleichen Typs vereinfacht zugreifen zu können. Dies war in Modula 2 noch nicht möglich.

```
WITH <VarAusdruck> [AS <NeuerName>] ,
     <VarAusdruck> [AS <NeuerName>] DO
     <Anweisungsfolge>
END
```

Sie können mit „**`WITH`**" also längere Variablennamen abkürzen. Wird anstelle des Ausdrucks „`<VarAusdruck>`" ein Typ angegeben, so wird eine neue Variable erzeugt.

Weiterhin kann der Compiler Optimierungen durchführen, da er Variablen, soweit möglich, in den Prozessor-Registern läßt und Adreßausdrücke nur einmal berechnen muß.

Ein Hinweis für alle fortgeschrittenen Programmierer: Nicht nur Elemente von Record oder Arrays können umbenannt werden, sondern dank **AS** auch solche, die nur über mehrere Pointer dereferenziert erhältlich wären.

Beispiel:

```
WITH
   Dieser_wunderbar_lange_Variablenname AS wV1,
   Dieser_ebenso_schoene_Variablenname  AS wV2,
   rec.array[6]                         AS i,
   Ptr^.field[i]^.elem^                 AS c    DO
   ...
END;
```

Wie man aus dem Listing leicht sehen kann, wird die Schreibarbeit hiermit sehr vereinfacht. Für alle, die die den Teil ab `rec.array[6]` nicht verstanden haben, sei zur Beruhigung gesagt, dies war nur ein Beispiel für die Profis, und muß zu diesem Zeitpunkt noch nicht verstanden werden. Wenn Sie dieses Handbuch durchgearbeitet haben, werden Sie es warscheinlich auch verstehen.

Übungen 3:

1. Erstellen Sie ein Programm mittels FOR-Schleife, welches die Zahlen zwischen 1 und N zusammenzählt. N ist eine ganze Zahl, welche vom Benutzer eingegeben wird.

2. Erklären Sie den wesentlichen Unterschied zwischen WHILE...DO und REPEAT...UNTIL-Schleifen.

3. Welche Scheifenarten gibt es in **Cluster**? (Ein Tip, es sind vier)

4. Wie oft wird die nachfolgende Scheife ausgeführt?

```
x := 1;
REPEAT
  x := x + 2
UNTIL x > 100;
```

5. Bestimmen Sie bitte für folgendes Beispiel, welche Anweisungsfolge ausgeführt wird, wenn a=-10, b=TRUE, c="d" ist, bzw. für welche Variablenwerte die einzelnen Anweisungsteile ausgeführt werden.

```
IF (a>0) OR (NOT b AND (c OF "a".."z") THEN
  Anweisung1;
OR_IF KEY c
  OF "A" AND_IF b THEN
                    Anweisung2;
                  ELSE
                    Anweisung3
                  END
  OF "B".."Z"     THEN
                    Anweisung4;
                  END
ELSE
  Anweisung5;
END
```


3.13 Standardfunktionen

Wie bei einem Taschenrechner verschiedene mathematische Funktionen fest eingebaut sind, so kommt auch **Cluster** mit einer Reihe von vordefinierten Standardfunktionen ins Haus.

Sie folgen nun in tabellarischer Form, wobei für Sie als angehenden Programmierer vor allem von Bedeutung ist, auf welchen Typ von Zahl diese Funktionen wirken, d. h. aus welcher Grundmenge eine als Argument der Funktion verwendete Variable stammen darf, und welchen Zahlentyp sie als Ergebnis liefern. Ferner enthält die Tabelle eine kurze Beschreibung, was eigentlich berechnet wird.

Die Standardfunktionen `SUCC` und `PRED` liefern bei Integern etc. den Wert $+/-1$, bei Zeigern auf Strukturen einen Zeiger des selben Typs, dessen Zieladresse um die Größe eines Zielelements erhöht ist[29]. Über Zeiger und dynamische Strukturen lesen Sie bitte im Kapitel für fortgeschrittene Programmierer weiter.

Beispiel:

```
SUCC(3)   = 4
PRED("B") = "A"
```

Neben diesen Standardfunktionen gibt es auch noch sogenannte Standardprozeduren (was eine Prozedur ist, folgt anschließend), die nicht – wie eine Funktion – einen Wert zurückliefern, sondern das Argument verändern (also keinen anderen Typ als den Typ des Arguments erzeugen):

[29]Ein Hinweis für C-Programmierer: Mithilfe dieser Funktionen läßt sich dasselbe erreichen wie wenn unter C ein Zeiger um eins erhöht oder erniedrigt wird. Der Zeiger wird auf das nächste Element gesetzt. Zeigt der Zeiger

Funktion	**wirkt auf**	**erzeugt**	**berechnet**
`SIN(x)`	`REAL`	`REAL`	Sinus von `x`
`COS(x)`	`REAL`	`REAL`	Cosinus von `x`
`TAN(x)`	`REAL`	`REAL`	Tangens von `x`
`ASIN(x)`	`REAL`	`REAL`	Arcussinus von `x`
`ACOS(x)`	`REAL`	`REAL`	Arcuscosinus von `x`
`ATAN(x)`	`REAL`	`REAL`	Arcustangens von `x`
`SINH(x)`	`REAL`	`REAL`	Sinushyperbolicus von `x`
`COSH(x)`	`REAL`	`REAL`	Cosinushyperbolicus von `x`
`TANH(x)`	`REAL`	`REAL`	Tangenshyperbolicus von `x`
`SQRT(x)`	`REAL`	`REAL`	Quadratwurzel von `x`
`EXP(x)`	`REAL`	`REAL`	Eulersche Zahl hoch `x`
`LN(x)`	`REAL`	`REAL`	Natürlicher Logarithmus von `x`
`LOG(x)`	`REAL`	`REAL`	Dekadischer Logarithmus von `x`
`ABS(x)`	`REAL`	`REAL`	Betrag von `x`
`ODD(x)`	`CARDINAL`	`BOOLEAN`	liefert true/false, falls `x` ungerade/gerade
`PRED(x)`	zählbar	`zählbar`	Wert von `x`, um eins vermindert
`SUCC(x)`	zählbar	zählbar	Wert von `x`, um eins erhöht
`CEIL(x)`	`REAL`	`REAL`	Kleinste ganze Zahl `>= x`
`FLOOR(x)`	`REAL`	`REAL`	Größte ganze Zahl `<= x`

Tabelle 3.2: Standardfunktionen von **Cluster**

Prozedur	**wirkt auf**	**Ergebnis**
`INC(x)`	`INTEGER`	erhöht `x` um 1
`INC(x,y)`	`INTEGER`	erhöht `x` um `y`
`DEC(x)`	`INTEGER`	erniedrigt `x` um 1
`DEC(x,y)`	`INTEGER`	erniedrigt `x` um `y`
`EVEN(x)`	`INTEGER`	rundet `x` auf die nächste gerade Zahl ab

Tabelle 3.3: Standardprozeduren von **Cluster**

3.13.1 Prozeduren

Das Prozedurkonzept ist eines der mächtigsten Konzepte imperativer Programmiersprachen: Hierdurch kann jede als Programm formulierte Vorschrift zu einer elementaren Anweisung in einem anderen Programm werden. Darüberhinaus dienen Prozeduren der Zerlegung und Strukturierung umfangreicher Programme, und sie erlauben die Verwendung rekursiver Techniken (siehe unter 3.16 ab Seite 113). Nebenbei sparen sie, falls ein Programmteil mehrfach verwendet werden soll, Tipparbeit und Speicherplatz.

beispielsweise auf ein Element eines Arrays von `RECORD`s, so wird der Zeiger auf den nächsten Record des Arrays gesetzt

```
PROCEDURE <Bezeichner> [<Liste von formalen Parametern>];(1)
[<Variablendeklarationen>](2)
BEGIN
  <Anweisungen>
END;
```

Eine Prozedur besteht aus dem Schlüsselwort **PROCEDURE** gefolgt von einem Bezeichner (dem Namen der Prozedur), aus einer Liste von formalen Parametern (wahlfrei), aus einer Folge von Deklarationen (Variablen- und/oder Typdeklarationen, ebenfalls wahlfrei) und aus einer Folge von Anweisungen.

Der Bezeichner und die Liste der formalen Parameter bilden den *Prozedurkopf* (1), die Deklarationen und Anweisungen den *Prozedurrumpf* (2). Der Prozedurrumpf ist in der Regel ein Block[30].

Aus einem Programm heraus wird die Prozedur mit ihrem Namen aufgerufen, der Prozedurname wird also zu einer selbstdefinierten Anweisung.

Beispiel:

```
PROCEDURE GutenTagSagen;
BEGIN
  WriteString("Guten Tag!");
END GutenTagSagen;
...
BEGIN(* Hauptprogramm *)
...
  IF a=1 THEN GutenTagSagen END;
| Die Prozedur wird mit ihrem Namen aufgerufen
...
```

[30] Das heißt, in ihm können alle Elemente auftreten, die auch zwischen **MODULE** und **CLOSE** auftreten können

Variablen, die innerhalb einer Prozedur definiert wurden, gelten auch nur innerhalb dieser Prozedur. Solche Variablen werden als *lokale Variablen* bezeichnet. Globale Variablen dagegen sind solche, die im Hauptprogramm, das heißt außerhalb der Prozedur, deklariert worden sind. Sie sind überall im Programm und somit auch in Prozeduren verfügbar.

Zwar ist es denkbar, alle in einem Programm benötigten Variablen global zu definieren. Dabei jedoch geht ein beträchtlicher Teil an Übersichtlichkeit verloren. Zum anderen ist sichergestellt, daß auf eine lokale Variable keine andere Prozedur außerhalb zugreifen kann.

Wird in einer Prozedur eine Variable definiert, die denselben Namen hat wie eine Variable des Hauptprogramms, so gilt innerhalb einer Prozedur die dort definierte Variable. Nach Abarbeitung des Unterprogramms gelangt die globale Variable dann wieder „an die Macht". hierbei spricht man auch von Sichtbarkeitsbereichen. Ein Sichbarkeitsbereich ist die Definintionsebene eines Moduls oder einer Prozedur. In einem Sichbarkeitsbereich kann man immer auf die Elemente, die in ihm definiert worden sind sowie auf die Elemente der über ihm liegenden Sichbarkeitsbereiche zugreifen (über ihm soll heißen in Richtung Modulebene).

Im Gegensatz zu „C" kann man in **Cluster** auch innerhalb einer Prozedur weitere Prozeduren definieren, die *lokale Prozeduren* genannt werden. Diese lokalen Prozeduren können nur von den Prozeduren aus aufgerufen werden, innerhalb derer sie definiert wurden. Der Vorteil dieser Prozeduren, sie können auf die Variablen und Übergabeparameter (s. u.) der sie umgebenden Prozeduren zugreifen. Außerdem können Sie auf diese Weise mehreren Prozeduren den gleichen Namen geben, solange sie sich nicht im gleichen Sichtbarkeitsbereich befinden. Existiert eine globale

Prozedur mit gleichem Namen, kann innerhalb des Sichbarkeitsbereiches der Prozedur nur auf die lokale zugegriffen werden. Bei geschickter Verwendung *kann* das zur besseren Lesbarkeit des Programms stark beitragen. Bei wiederkehrenden Anweisungssequencen die nur innerhalb einer Funktion verwendet werden sollte man auf alle Fälle eine lokale Prozedur verwenden und keine modulglobale. Lokale Prozeduren können also immer auf die Elemente der über ihnen liegenden Porzeduren zugreifen, doch nie eine Prozedur auf die einer zu ihr lokalen Prozedur. Eine lokale Prozedur kann an jeder Stelle zwischen dem Prozedurkopf der sie umgebenden Prozedur und deren **BEGIN**-Teil definiert werden. Lokale Prozeduren können selbstverständlich auch selbst wieder lokale Prozeduren enthalten, und das nahezu beliebig tief.

Es ist auch möglich, innerhalb einer Prozedur weitere Module zu importieren oder von einem bereits importierten Modul weitere Bezeichner zu importieren. Dies geschieht wie im Modul durch eine IMPORT-Anweisung nach dem Prozedurkopf.

Oft ist es notwendig, einer Prozedur einige Werte mit auf den Weg zu geben. Man spricht in diesem Fall von Prozeduren mit Übergabewerten. Z. B. wäre es sinnvoll, einer Prozedur, die einen Punkt auf den Bildschirm zeichnen soll, die `x`- und `y`-Koordinaten des Punktes zu übermitteln. Dadurch kann die Prozedur durch die Übergabewerte zu unterschiedlichem Verhalten veranlaßt werden, der Punkt wird dadurch an die gewünschte Stelle gezeichnet.

Bei den Prozeduren mit Übergabewerten (Parametern) unterscheidet man Werteparameter (call by value) und Variablenparameter (call by reference). Vom Aufbau her gesehen unterscheiden sich die beiden nur dadurch, daß beim Variablenparameter vor der Liste der Übergabewerte ein **VAR** steht. Inhaltlich ergibt sich folgender Unterschied: Ein Werteparameter wird in der Prozedur

selbst nicht verändert, von ihm wird beim Prozeduraufruf eine Kopie erstellt, mit der dann innerhalb der Prozedur gearbeitet wird. Die übergebene Variable bleibt dabei unverändert. Erfährt dagegen ein Variablenparameter in der Prozedur eine Manipulation, so ist diese über die Prozedurgrenzen hinaus gültig, da in diesem Fall die übergebene Variable selbst verändert wird. Ein Beispiel soll dies verdeutlichen:

Die beiden Beispiele sind fast identisch und unterscheiden sich nur durch das **VAR** im zweiten Programm, was auf einen Variablenparameter hinweist.

```
MODULE Werteparameter_Test;

FROM InOut IMPORT ReadInt, WriteInt, WriteLn;

VAR global1, global2 :  INTEGER;

PROCEDURE Werteparameter_Demo(i,j:INTEGER);
BEGIN
  i:=i+10;
  j:=j+20;
END Werteparameter_Demo;

BEGIN
  ReadInt(global1);
  ReadInt(global2);
  Werteparameter_Demo(global1,global2);
  WriteInt(global1);
  WriteLn;
  WriteInt(global2);
END Werteparameter_Test.
```

Als Ausgabe ergeben sich hier die ursprünglich eingelesenen Werte. Anders jedoch beim zweiten Programm:

```
MODULE Variablenparameter_Test;
FROM InOut IMPORT ReadInt, WriteInt, WriteLn;
VAR global1, global2 :  INTEGER;
PROCEDURE Variablenparameter_Demo(VAR i,j:INTEGER);
BEGIN
  i:=i+10;
  j:=j+20;
END Variablenparameter_Demo;
BEGIN
  ReadInt(global1);
  ReadInt(global2);
  Variablenparameter_Demo(global1,global2);
  WriteInt(global1);
  WriteLn;
  WriteInt(global2);
END Variablenparameter_Test.
```

Die übergebenen Parameter werden in der Prozedur verändert und modifiziert an den aufrufenden Programmteil zurückgegeben.

Eine weitere Möglichkeit ergibt sich durch die Verwendung von „**REF**" . Hierbei wird sichergestellt, im Gegensatz zu „**VAR**", daß der übergebene Wert nicht verändert wird. Wird innerhalb der Prozedur versucht, den übergebenen Wert zu verändern, so meldet sich der Compiler in seiner eigenen Art mit einer Fehlermeldung.

Nun wird der eine oder andere fragen, wofür das gut sein soll. Diese Übergabeart lohnt sich für größere Typen (Arrays oder Records), denn bei der ersten Methode wird der übergebene Wert kopiert und mit der Kopie innerhalb der Prozedur weitergearbeitet.

Sie werden vielleicht sagen, man kann ja für größere Typen, die innerhalb der Prozedur nicht verändert werden sollen, die zweite Methode verwenden. Dies ist richtig, allerdings wird hierbei nicht

sichergestellt, daß der übergebene Wert nicht verändert wird. Solange Sie alleine an einem Projekt arbeiten, mag dies nicht entscheidend sein, bei Gruppenarbeit gibt aber erst Methode drei die nötige Sicherheit.

Einige zusätzliche Hinweise: Wollen Sie mehrere Variablen des gleichen Typs übergeben, so können Sie diese durch Komma trennen und den Typ nach einem Doppelpunkt anfügen:

```
PROCEDURE Name(i,j,k,l:INTEGER);
```

Wollen Sie hingegen mehrere verschiedene Variablentypen übergeben, so werden diese durch Semikolon getrennt. Ein eventuell vorhandener **REF**- oder **VAR**-Parameter gilt nur bis zum nächsten Semikolon:

```
PROCEDURE Name(i,j:INTEGER;oo,pp:REAL);
PROCEDURE NeuerName(VAR help:CHAR;zaehler:CARDINAL);
PROCEDURE Pro3(REF k:BOOLEAN;VAR kaelte:REAL);
```

Ebenso können sogenannte Defaultwerte definiert werden. Diese Werte werden automatisch eingesetzt, wenn die Übergabeparameter weggelassen wurden. Somit lassen sich sehr flexible Programme und Prozeduren erstellen.

```
PROCEDURE Defwerte(val :  LONGINT;breite :  INTEGER:=0);
```

Diese Prozedur kann aufgerufen werden mit

`Defwerte(10,4)` oder aber auch mit `Defwerte(41)`.

Hat eine Prozedur viele Parameter, kann nicht immer davon ausgegangen werden, daß die zu überspringenden Parameter am Ende liegen. In diesem Fall muß die Zuweisung durch Schlüsselwörter (die Namen der Parameter) vorgenommen werden[31]:

```
PROCEDURE DefNeu(wert         : INTEGER;
                 flag1,flag2 : BOOLEAN:=FALSE;
                 breite      : INTEGER:=640;
                 hoehe       : CARDINAL:=512);


DefNeu(10,TRUE,FALSE,320,256);
DefNeu(5,breite:=320);
DefNeu(8,breite:=320,hoehe:=250);
```

Sehr wichtig ist, daß die Reihenfolge der Parameter eingehalten werden muß.

[31] C++-Programmierer sollten spätestens jetzt begeistert sein oder zumindest erstaunt aufschauen

3.13.1.1 `STATIC`-Variablen

Das Schlüsselwort **STATIC** kann nur bei Variablendeklarationen innerhalb von Prozeduren verwendet werden. Diese Variablen überleben ihre Prozedur, und haben beim nächsten Aufruf noch den selben Wert wie beim Verlassen. Werden sie vorinitialisiert, so geschieht dies nur einmal am Programmanfang.
Beispiel:

```
PROCEDURE statictest();

VAR
  test1 STATIC: INTEGER;
  test2 STATIC:= 10;

BEGIN
  ...
END statictest;
```

Vielleicht werden Sie sich jetzt fragen, was dann der Vorteil gegenüber einer globalen Variablen sein soll? Ganz einfach, eine **STATIC**-Variable ist nur innerhalb der Prozedur sichtbar. Das heißt, sie kann ohne Probleme auch den gleichen Namen wie eine bereits definierte globale Variable haben, ohne daß es zu einem Namenskonflikt kommt.

3.13.1.2 `FORWARD`-Prozeduren

Wie Sie mittlerweile wissen, muß eine Prozedur vor ihrem ersten Aufruf deklariert sein.

Soll eine Prozedur verwendet werden, bevor ihr Prozedurrumpf implementiert werden kann, wenn sich also z. B. zwei Prozeduren gegenseitig aufrufen, muß sie mit dem Schlüsselwort **FORWARD** im voraus deklariert werden.

Dies ist erforderlich, da es ich bei **Cluster** um einen Single-Pass-Compiler handelt[32].
Um eine Prozedur FORWARD zu deklarieren, schreibt man vor den Prozedurkopf das Schlüsselwort **FORWARD**. Der Prozedurkopf wird dann an den Anfang des Moduls gestellt und zwar vor die Stelle, an der die Prozedur zum ersten Male aufgerufen wird. Zu beachten ist, daß der eigentliche Prozedurkopf nicht entfernt wird und auch kein Schlüsselwort vorangestellt bekommt:

```
FORWARD PROCEDURE fortest(VAR a: INTEGER);

PROCEDURE fortest(VAR a: INTEGER);
```

3.13.2 Funktionsprozeduren

Bisher können wir Prozeduren zwar Parameter übergeben, und mittels **VAR**-Parametern auch Ergebnisse zurückgeben, jedoch ist es nicht möglich ein solches Ergebnis direkt in mathematischen Ausdrücken zu verwenden oder sie direkt an eine andere Prozedur zu übergeben. Man mußte immer eine Variable übergeben, und danach dann diese Variable verwenden, auch wenn man den Wert nach der einmaligen Verwendung nicht mehr gebraucht wurde.

Dieses Problem lösen Funktionsprozeduren. Das sind Prozeduren, die direkt ein Ergebnis zurückgeben können, und daher direkt an Stelle einer Variablen oder Konstanten des Rückgabetyps verwendet werden können. Im Prinzip waren alle zuvor beschriebenen Standardfunktionen wie „SIN“ oder „COS“ Funktionsprozeduren.

Welchen Typ eine Funktion zurückgibt, muß bei deren Definition festgelegt werden, indem man hinter der Parameterliste statt

[32] Single-Pass= Der Compiler geht nur einmal über der Programmtext, im Gegensatz zu Multi-Pass-Compilern

des „;“ einen Doppelpunkt gefolgt von dem gewünschten Typ und einem abschließenden Semikolon schreibt.

Innerhalb der Prozedur wird der Wert, den man als Ergebnis zurückgeben will, an das aufrufende Programm mittels **RETURN** zurückgegeben.

Als einfaches Beispiel soll ein kleines Programm mit der Funktionsprozedur Quadratwurzel zur Wurzelberechnung dienen.

Beispiel:

```
MODULE Quadratwurzel_berechnen;

FROM InOut IMPORT ReadReal, WriteLn, WriteString,
                   WriteReal;

VAR i :  REAL;

PROCEDURE Quadratwurzel(a:REAL):REAL;
(* Der Ergebnistyp muß
   in einer Funktions-
   prozedur separat
   ausgewiesen werden *)
BEGIN
  RETURN SQRT(a)
| Der Funktionswert wird zurückgegeben
END Quadratwurzel;

BEGIN
  WriteString("Bitte geben Sie eine positive
   Zahl ein:  ");
  ReadReal(i);
  WriteLn;
  WriteReal(Quadratwurzel(i),10,3);
(* Der Funktionsname *)
END Quadratwurzel_berechnen.
(* beinhaltet den Funktionswert *)
```

Ein „**RETURN**” ohne Rückgabewert innerhalb von Funktionsprozeduren ist nicht möglich. Wird jedoch „**RETURN**“ innerhalb von normalen Prozeduren verwendet, so wird die Prozedur auf der Stelle verlassen.

Hat eine Funktion nur einen Rückgabetyp, jedoch keine Parameter, muß man sie folgender Maßen definieren:

```
PROCEDURE Test():BOOLEAN;
...
```

Beim Aufruf muß man dann entsprechend schreiben

```
IF Test() THEN
...
```

Ohne die folgenden Klammern würde der Compiler im letzten Beispiel den Term `Test()` nicht als BOOLEAN, sondern als Prozedurtyp (s. u.) interpretieren und einen Fehler melden, da eine IF-Abfrage eine boolsche Bedingung erwartet.

Funktionen sollte nach Möglichkeit keine Nebeneffekte haben, d. h. sie sollten nur einen Wert zurückliefern, und keine anderen Variablen verändern. Eine Ausnahme machen hier Prozeduren, deren Rückgabewert etwas darüber aussagt, ob die Prozedur erfolgreich ausgeführt werden konnte. Jedoch sollte man für diesen Zweck keine Funktionen verwenden, sondern Fehler durch Exceptions (s .u.) anzeigen.

Bei manchen Funktion kann es sein, daß der Rückgabewert nur ein Nebenprodukt ist, und man ihn gar nicht immer benötigt. Statt ihn nun einer Dummyvariablen[33] zuzuweisen, ist es sinnvoller, explizit anzugeben, daß man darauf verzichten will, dies kann mit **FORGET** geschehen. Besonders bei einigen Betriebssystemroutinen kann dies sinnvoll sein.

```
FORGET Test()
```

[33] Eine Variable, deren Inhalt niemals verwendet wird, praktisch ein „Placebo“ für den Compiler

3.14 Felder und Mengen

3.14.1 Die Menge

Eine Menge kann beliebige Elemente aus einer Grundmenge enthalten oder leer sein (leere Menge). Jedes Element darf höchstens einmal, nicht jedoch doppelt oder noch häufiger vorkommen. Enthält eine Menge, wie dies meistens der Fall ist, nicht alle Elemente, die in der Grundmenge vorhanden sind, so nennt man sie Teilmenge der Grundmenge.

In **Cluster** ist der Typ einer Menge die Grundmenge. Diese muß ein zählbarer Typ mit maximal 32 Elementen, also ein Unterbereichs- oder Aufzählungstyp sein.
Beispiel:

```
...
TYPE
  Monate = (Januar, Februar, Maerz, April, Mai,
            Juni, Juli, August, September, Oktober,
            November, Dezember);
  MonSet = SET OF Monate;
...
```

Eine Konstante dieses Typs wird durch Aufzählen der enthaltenen Elemente gebildet:
Beispiel:

```
CONST   SommerMonate = MonSet:{Juli,August,September};
```

Um zu prüfen, ob ein bestimmtes Element in einer Menge vorhanden ist, kann man den Operator **IN** benutzen:

```
VAR Mons :  MonSet;
...
```

```
IF Juni IN Mons THEN ...  END;
```

Ein einzelnes Element kann mit `INCL` bzw. `EXCL` in eine Menge eingefügt bzw. aus einer Menge entfernt werden.

```
INCL(Mons,Juni);   |fügt in die Menge "Mons" den Juni ein
EXCL(Mons,Januar); |entfernt den Januar aus der Menge "Mons"
```

Außerdem sind folgende Mengenoperationen möglich:

Vereinigungsmenge	„+”
Schnittmenge	„*”
Mengendifferenz	„−”
symmetrische Differenz	„/”

Möchte man die Elemente einer Menge einer anderen hinzufügen, oder genau diese aus einer anderen Menge entfernen, kann man die Standardprozeduren `UNI`/`SEC` verwenden.

```
UNI(Mons,{Juni,August});        | fügt in die Menge "Mons" die
                                | Menge {Juni,August} ein.
SEC(Mons,{Januar,Mai,Dezember});| entfernt den Januar, den Mai
                                | un den Dezember aus der
                                | Menge "Mons"
```

Ein etwas größeres Beispiel sollte den Sachverhalt klarer darstellen:


```
MODULE bl1;
FROM InOut IMPORT WriteString, WriteLn;
TYPE
  Oktal= [0..7];
  OktSet = SET OF Oktal;
VAR
  tief, hoch:  OktSet;
BEGIN
  tief := OktSet:{1,2};
  IF 1 IN tief THEN
    WriteString("1 ist enthalten")
  END;
  WriteLn;
  INCL(tief,4);
  IF 4 IN tief THEN
    WriteString("4 ist enthalten")
  END;
  WriteLn;
  EXCL(hoch,2);
  IF (2 NOT IN hoch) THEN
    WriteString("2 ist nicht in hoch enthalten")
  END
END bl1.
```

Eigentlich sind `INCL` und `EXCL` Spezialfälle von `+` und `-`. Beispielsweise bedeutet

```
INCL(hoch,1);
EXCL(hoch,4);
```

das gleiche wie

```
hoch := hoch + {1};
hoch := hoch - {4};
```

Weitere Beispiele:

Gegeben sind die Mengen:
[A = {a, b, c, d}]
[B = {d, e, f, g}]
Dabei bedeutet:
[A + B = {a, b, c, d, e, f, g}] [A * B = {d}] [A - B = {a, b, c}]
[A / B = {a, b, c, e, f, g}]
Da doppelte Elemente in einer Menge nicht erlaubt sind, wird *d* im ersten Beipiel nicht doppelt eingefügt.

3.14.2 Das Array

In den meisten Programmiersprachen existiert die Möglichkeit, Variablen gleichen Typs in einer Variablen zusammenzufassen, die man sich als ein- oder mehrdimensionale Tabelle vorstellen kann. Man spricht hier von *Arrays*, Vektoren oder Feldern. Mittels eines Index können die einzelnen Elemente eines Arrays angesprochen werden.

Allgemein wird ein Array so festgelegt:

```
TYPE <Bezeichner> = ARRAY <TypDeklaration> OF
                          <TypDeklaration>;
```

Beispiel:

```
TYPE IntArray = ARRAY [1..10] OF INTEGER;
```

Hiermit wird eine eindimensionale Tabelle definiert, welche 10 `INTEGER`-Zahlen aufnehmen kann (10 Zeilen $\times$ 1 Spalte = 10 Elemente). Die „Zellen" sind von 1 bis 10 durchnumeriert.

Nun kann eine Variable dieses Typs deklariert werden:

```
VAR int :  IntArray;
```

Die einzelnen Elemente dieser Variablen sind über die Indizes 1 bis 10 ansprechbar. Z. B. ließe sich dem ersten Feld des Arrays der Wert 100 wie folgt zuordnen:

```
int[1]:=100;
```

Auch Operationen wie die folgenden sind möglich:

```
int[1]:=int[2];
int[3]:=int[1]+int[7];
int[4]:=int[a+1]*int[int[8]];
```

Wie man am letzten Beispiel sieht, kann der Wert eines Feldes als Index für ein anderes Feld verwendet werden.

Probieren Sie diese Dinge auch selber aus und spielen Sie ein wenig herum. Aus den eigenen Fehlern lernt man bekanntlich am meisten.

Man kann auch Arrays von Arrays bilden, also zwei- oder mehrdimensionale Tabellen erstellen. Für ein zweidimensionales Array könnte das so aussehen:

```
TYPE
   Matrix = ARRAY [1..10] OF ARRAY [1..10] OF INTEGER;
VAR   mat :  Matrix;
```

Dieses Array besteht aus 10 Zeilen je 10 Spalten, also 100 indizierbaren Feldern. Vereinfacht kann man auch folgende Schreibweise nutzen:

```
TYPE
  Matrix = ARRAY [1..10],[1..10] OF INTEGER;
VAR   mat :  Matrix;
```

Zugegriffen werden kann auf die Elemente auf zwei verschiedene Arten, die die gleiche Bedeutung haben:
`Mat[2,7]` ist gleichbedeutend mit `Mat[2][7]`.
In **Cluster** darf man beliebig dimensionale Arrays deklarieren. Ein vierdimensionales Feld könnte so aussehen:

```
TYPE
  Matrix = ARRAY [1..10],[1..5],[1..7],[1..7] OF INTEGER;
VAR   mat :  Matrix;
```


Um ein Array zu verarbeiten, wird meistens die FOR-Struktur gebraucht. Im folgenden Beispiel wird ein Array mit Nullen belegt:

```
...
VAR feld :  ARRAY [0..9] OF INTEGER;
    i:  INTEGER;
...
FOR i:=0 TO 9 DO
  feld[i]:=0
END;
...
```

Man kann zwei Arrays einander zuweisen, wenn sie den gleichen Typ haben, nicht jedoch, wenn sie nur aus den gleichen Typen zusammengesetzt sind. Folgendes Beispiel verdeutlicht dies:

```
...
TYPE Field = ARRAY [0..10] OF INTEGER;

VAR f1,f2 :  Field;
    f3:  ARRAY [0..10] OF INTEGER;
    f4:  Field;
...
```

Zulässige Zuweisungen wären hier:

```
f1:=f2;
f3:=f3;
f1:=f4;
f3[4]:=f1[5];
```

Fehlerhafte und damit falsche Zuweisungen wären:

```
f1:=f3;
f3:=f4;
f1[4]:=f2;
```

Man kann übrigens die Grenzen einer Arrayvariablen auch dann ermitteln, wenn die Typdeklaration unbekannt ist. Programme können damit unabhängiger gestaltet werden, d. h., daß sich alle Programmroutinen automatisch anpassen, wenn die Typdeklaration modifiziert wird. Die Grenzen eines Array sind in Form von Attributen verfügbar und können mit den nachgestellten Attributnamen ermittelt werden[34]:

`field'MIN`	liefert den kleinsten Index des Arrays
`field'MAX`	liefert den größten Index des Arrays
`field'RANGE`	liefert die Anzahl der Elemente im Array

Beispiel:

Löschen eines Arrays „`Feld`", d. h. Auffüllen desselben mit Nullen:

```
...
FOR i:=Feld'MIN TO Feld'MAX DO
  Feld[i]:=0
END;
...
```

Man kann auch einen Arraytypen festlegen, dessen Größe noch nicht feststeht. In diesem Fall spricht man von einem „offenen Array". Bei der Variablendeklaration ist dies allerdings nicht möglich. Hier muß bei der Erzeugung eine feste Länge angegeben werden. Es ist jedoch möglich, den offenen Typen als Übergabeparametertypen zu verwenden. An diesen lassen sich dann alle Arrays fester Länge zuweisen, die über diesen offenen Typen definiert worden sind. Innerhalb der Prozedur läßt sich dann mittels der oben beschriebenen Attribute feststellen, wieviele Elemente das übergebene Array wirklich hat.

[34] Das Hochkomma „ ' " erreicht man durch Betätigen der Tastenkombination `Alt`-`ä`

Beispiel:

```
...
TYPE Vector = ARRAY OF REAL; |offenes Array

VAR vec3:  Vector(3);
| bei der Variablendeklaration muß
    vec10 :  Vector(10);
| eine feste Länge angegeben werden
...
```

Der kleinste Index eines offenen Arrays ist Null, kann also niemals negativ werden.

Wichtig im Zusammenhang mit Funktionsprozeduren ist, daß diese komplexe Rückgabetypen haben können, u. a. auch Arrays. Diese werden dann mit `RESULT` zurückgegeben. Bei `RESULT` handelt es sich um eine Pseudovariable, die den Typ des Rückgabewertes hat und wie eine normale Variable verwendet werden kann:

```
TYPE
  Vector3 = ARRAY [1..3] OF REAL;

PROCEDURE VAdd(VAR a,b:  Vector3):  Vector3;

VAR
  i:  INTEGER;

BEGIN
  FOR i := 1 TO 3 DO
    RESULT[i] := a[i] + b[i]
  END
END VAdd;
```

Hierbei ist es wichtig zu wissen, daß die Funktionsprozedur erst verlassen wird, wenn das Ende der Prozedur erreicht ist, oder ein Return ohne folgenden Wert auftritt.

3.14.3 LIST OF

Sie haben die Möglichkeit eine *variable* Anzahl von Parametern an Prozeduren zu übergeben. Voraussetzung dafür ist, daß die Prozedur als letzten Parameter einen vom Typ „`LIST OF Type`“ hat, s. u. .

Beim Aufruf kann man dann für diesen Parameter beliebig viele Argumente diesen Typs übergeben.

In der Prozedur wird ein solcher Parameter wie ein offenes Array[35] behandelt. Mit `'RANGE`[36] erhält man die Anzahl der übergebenen Parameter. Man kann auch einfach überhaupt nichts an dieser Stelle übergeben.

Das nachfolgende Beispiel sollte den Sachverhalt deutlicher machen:

```
MODULE ListOfTest;

FROM InOutIMPORT All;
FROM ConversionsIMPORT IntToString;

PROCEDURE WriteLine(REF str : LIST OF STRING);

VAR
  i : INTEGER;

BEGIN
  FOR i:=0 TO str'MAX DO
    WriteString(str[i]);
  END
END WriteLine;

BEGIN
  WriteLine("Dies"," ist eine Zahl ",
            IntToString(12)," als Test");
  WriteLn;
END ListOfTest.
```

Werden überhaupt keine Parameter übergeben, ist `str'MAX = -1`, das heißt, die Schleife wird nie durchlaufen. `str'RANGE` ist in diesem Fall Null.

[35] Mehr zu offenen Arrays erfahren Sie im Kapitel für Fortgeschrittene Programmierer. Im Moment reicht es zu wissen, daß man darauf wie auf ein normales Array zugreifen kann.

[36] Über `RANGE` mehr auf Seite 94 unter 52

3.14.4 Der Record

Die zweite Struktur, mit welcher mehrere verschiedene Datentypen zu einem neuen zusammengefaßt werden können, ist der Record. Im Unterschied zum Array müssen die hier gebündelten Typen jedoch nicht unbedingt vom selben Typ sein. Außerdem erfolgt hier der Zugriff nicht über einen Index, sondern über einen zusätzlichen Bezeichner.

```
TYPE <Bezeichner> = RECORD
                      <Bezeichner> {,
                      <Bezeichner> }:  <TypDeklaration> {;
                      <Bezeichner> {,
                      <Bezeichner> }:  <TypDeklaration> }
                    END;
```

Für eine Adressenkartei könnten z. B. folgende Personendaten zu einem Record zusammengefaßt werden:

```
...
TYPE Adresse = RECORD
                 Name,
                 Vorname :  STRING(32);
                 Telefon :  STRING(10);
                 Alter:  INTEGER;
                 PLZ: STRING(5);
                 END;

VAR adr :  Adresse;
...
```

Um hier auf ein bestimmtes Element zuzugreifen, muß man es – wie bei einem unqualifizierten Import; erinnern Sie sich? – mit einem zusätzlichen Punkt qualifizieren:

```
adr.Name:="Richter";
adr.Vorname:="Tom";
adr.Telefon:="095165873";
usw.
```

Man kann zwei Records desselben Typs einander zuweisen. Das erlaubt ein einfacheres Arbeiten, da man nicht, wie bei anderen Programmiersprachen, jedes Element einzeln dem anderen zuweisen muß.

Wegen des geschlossenen Typkonzepts in **Cluster** können Records auch in Arrays gesammelt werden oder in anderen Records auftauchen. Somit eignen sich Records in idealer Weise als Datensammler, die ein Objekt genau beschreiben.

Beim Amiga finden diese objektbeschreibenden Records einen großen Anwendungsbereich. So wird zum Beispiel jedes Fenster durch einen eigenen Record des Typs „Window" beschrieben. Mehr dazu an einer anderen Stelle in diesem Handbuch.

Wollen Sie einen Record von einer Funktionsprozedur zurückgeben lassen, müssen Sie wie bei Arrays `RESULT` verwenden.

Lassen Sie uns die Adressenkartei noch einmal betrachten. Wollen Sie eine Adressendatei programmieren, in der Sie die Daten Ihrer Freunde und Bekannten ablegen wollen, läge doch nichts näher, als ein Array von Records zu verwenden:

```
VAR AdressKartei :  ARRAY [1..100] OF Adresse;
                   (* Platz für 100 Adressen *)
```

Es wäre jedoch günstiger, wenn Sie wissen würden, wieviele Elemente wirklich im Array enthalten sind. Nachstehende Zeilen berücksichtigen dies:

```
   VAR AdressKartei : RECORD
                         NumEintrag :  INTEGER;
                         Daten:  ARRAY [1..100] OF Adresse;
                       END;
```


Um einen weiteren Eintrag („NeuePerson") anzuhängen, sind folgende Zeilen notwendig:

```
 INC(AdressKartei.NumEintrag);
            (* Zähler um eins erhöhen *)
 AdressKartei.Daten[AdressKartei.NumEintrag]:=NeuePerson;
            (* Neuen Eintrag zuweisen *)
```

Soll eine bestimmte Person nach ihrem Namen („SuchName") gesucht werden, benötigt man die Funktion „`Equal`" aus dem Modul „`Strings`". Sie liefert `TRUE`, wenn zwei Zeichenketten gleich sind.

```
i:=1;
WHILE (i<=AdressKartei.NumEintrag) | Letztes Element erreicht?
 AND_WHILE NOT Equal(AdressKartei.Daten[i].Name,SuchName) DO
    | Beide Zeichenketten gleich?
    INC(i);
  ELSE
    WriteString("Gefunden");WriteLn;
  END
ELSE
  WriteString("Ende des Feldes erreicht");WriteLn;
END;
```

Bitte nehmen Sie die hier aufgeführten Beispiele als Anregung für Ihre eigenen Programme, schreiben Sie sie ab, verändern und erweitern Sie sie.

3.14.5 STRING

Bei Strings handelt es sich in **Cluster** um einen besonderen Datentyp.

Grundmenge: Beliebige Zeichenketten[37], die sich aus der Grundmenge der für den Typ `CHAR` zulässigen Zeichen zusammensetzen.

Operatoren: „+": Zusammenfügen von Zeichenketten (Konkatenation), ist nur für Stringkonstanten definiert.

Beschreibung: Einer Variablen vom Typ `STRING` kann eine Zeichenkette zugewiesen werden, deren maximale Länge in der Variablendeklaration angegeben werden muß. Es ist empfehlenswert, diese Länge etwas größer als nötig zu wählen. C-Programmierer sollte nun ein Licht aufgehen, warum. Allen anderen sei hier nur kurz erwähnt, daß unter bestimmten Umständen ein Zeichen mehr als nötig verwendet wird. Außerdem läßt sich ein großzügig bemessener String, falls erforderlich, leicht mit zusätzlichen Zeichen „bestücken"[38]. Allerdings sollte man auch hierbei an den verfügbaren Speicherplatz denken und die Dimensionierung vernünftig vornehmen.

Es gibt mehrere Möglichkeiten, Informationen über einen String herauszubekommen:

Attribut	Beschreibung
`RANGE`	Anzahl der Elemente eines Arrays
`MIN`	Kleinstes Element eines einfachen Typs oder kleinster Index eines Arrays
`MAX`	Größtes Element eines einfachen Typs oder gröster Index eines Arrays

[37]einfach gesagt, eine Aneinanderreihung von beliebigen Zeichen

[38]Noch ein Wort an die C-Programmierer: **Cluster** verwaltet die Strings in einer Art struct (hier RECORD). Hierbei wird die Länge des Strings explizit in einem Datenfeld eingetragen. Außerdem wird der String zusätzlich nullterminiert, um das Arbeiten mit Betriebssystemroutinen zu erleichtern

Weitere Attribute existieren, welche auch im Zusammenhang mit anderen Objekten Verwendung finden können:

Attribut	Beschreibung
`SIZE`	Größe eines Objektes in Bytes
`PTR`	Zeiger auf das Objekt (Erklärung an einer anderen Stelle dieses Handbuchs
`ADR`	Adresse des Objektes

Nachfolgendes Beispiel sollte den Einsatz dieser Attribute deutlich machen:

```
TYPE
  Array = ARRAY [7..22] OF INTEGER;

VAR
  a: Array;
```

Dies bedeutet dann[39]

```
a'MIN   = Array'MIN   = 7
a'MAX   = Array'MAX   = 22
a'RANGE = Array'RANGE = 16
```

Zur Bearbeitung von STRING-Variablen stellt das Modul „`Strings`" zahlreiche Operationen zur Verfügung, mit denen Zeichenketten verbunden, zertrennt oder Teile von ihnen ausgeschnitten werden können (siehe Kapitel 7 Beschreibung der Standardmodule).
Auf die einzelnen Elemente eines Strings kann auf eine bestimmte Weise zugegriffen werden. Angenommen die Variable `Str` ist mit `Str: STRING(32)` als String mit 32 Elementen definiert. Dies entspricht dann etwa:

[39]Das Hochkomma „ ' " erreicht man durch Betätigen der Tastenkombination `Alt`-`ä`

```
Str :  RECORD
         len   :  INTEGER;
         data  :  ARRAY [0..31] OF CHAR
       END;
```

Nun kann im Programm die Länge des Strings mit `Str.len` und ein bestimmtes Element des Strings mit `Str.data` erfahren werden. Achtung, solange man einem String nur Konstanten zuweist, oder die Funktionen aus `Str` und `Strings` verwendet, ist sichergestellt, daß `str.len` immer richtig gesetzt ist, und daß der String ein Nullbyte angehängt hat. Wenn Sie selbst auf das Datenfeld eines Strings zugreifen, sind Sie dazu verplichtet, danach `len` entsprechend Ihrer Änderung zu setzen und ein Nullbyte hinten anzufügen.

Das schon mehrfach erwähnte Nullbyte am Stringende dient dazu, Clusterstrings einfacher an Betriebssystemroutinen übergeben zu können, da das Betriebssystem keine Strings mit einem Längeneintrag kennt, sondern nur solche, die mit einem Nullbyte (&0) terminiert sind. Außerdem gibt es Ausnahmen, bei denen die Verwendung des Nullbytes anstatt der Länge Geschwindigkeitsvorteile bietet. Z. B. wenn man einen String zeichenweise durchgeht, ist es unnötig auch noch die augenblickliche Position mit der Länge zu vergleichen, wenn man sowieso auf das Nullbyte stoßen wird. Umgekehrt ist es sinnvoll, wenn man z. B. einen String kopieren will, dies mit einer FOR-Schleife über die Länge zu machen.

Wichtig ist im Zusammenhang mit dem Nullbyte ist, daß dieses Zeichen bei der maximalen Stringlänge bei der Stringdefinition, sowie bei `str'RANGE`, berücksichtigt wird. In `str.len` dagegen ist es nicht enthalten.

Nun noch ein Beispiel für die konstante Stringaddition. Sie kann benutzt werden, wenn man eine Stringkonstante definieren will, die länger als eine Zeile im Editor ist, oder wenn man nichtdruckbare Steuerzeichen in einen String einfügen möchte.

```
CONST
  Str1 = "Dies ist ein Beispiel für ein Steuerzeichen"+&10+
         "in diesem Fall ein Zeilenvorschub".
```


Weitere Informationen zur Stringverarbeitung finden Sie in der Beschreibung des Moduls `Strings` Kapitel 7.

3.14.6 Typisierte Konstanten

Wie Sie einfache Konstanten deklarieren und verwenden konnten, wurde unter 3.6.1 ab Seite 31 schon besprochen. Nachfolgend wird gezeigt, wie Sie konstante Arrays oder Records erzeugen können. Eine derartige Konstante kann an einer beliebigen Stelle im Programm auftauchen. Dies ist eine Eigenschaft von **Cluster**, welche Sie in vergleichbaren Programmiersprachen wie Pascal oder Modula 2, nicht finden werden.

Um ein konstantes Array[40] zu deklarieren und zu initialisieren, geben Sie einfach

```
TYPE
  Matrix = ARRAY [0..2],[0..2] OF REAL;
CONST
  StdMatrix = Matrix:((5,3,2),
                      (1,0,2),
                      (3,3,3));
```

ein. Nachfolgendes Beispiel deklariert ein konstantes Record:

```
TYPE
  Wertemenge = RECORD
                 richtung :  Matrix; |Siehe oben
                 xposition:  INTEGER;
                 yposition:  INTEGER;
                 fractal:  REAL
               END;
CONST
  Werte = Wertemenge:(richtung=((0,1,1),
                                (1,1,1),
                                (0,0,0)),
                      xposition=10,
                      yposition=20,
                      fractal=22.5);
```

[40] Feld

Läßt man bei der Definition eines konstanten Records ein Element aus, wird dieses mit „0“ initialisiert. Es besteht auch die Möglichkeit, die Elementbezeichner wegzulassen, und die einzelnen Werte durch Kommata getrennt anzugeben. Allerdings darf man dann keine Elemente weglassen. Da diese Schreibweise wesentlich weniger aussagekräftig ist als die mit Bezeichnern, sollte man sie nur in Ausnahmefällen, z. B. bei großen Mengen solcher Records verwenden.
Mit derartigen Konstanten kann man natürlich auch Variablen und Parameter vordefinieren:

```
VAR
 v : Matrix:=Matrix:((1,0,1),
                     (0,1,0),
                     (1,0,1));
```

Um das zweimalige Schreiben des Typs zu vermeiden, existiert für typisierte Konstanten auch folgende Schreibweise:

```
VAR
 v :=Matrix:((1,0,1),
             (0,1,0),
             (1,0,1));
```


Übungen 4:

1. Worin unterscheiden sich Unterbereichs- von Aufzählungstypen?
2. Was ist der Unterschied zwischen einer Menge und einer Aufzählung?
3. Was ist bei folgendem Codefragment falsch?

```
TYPE
  T = [1..10];
VAR
  X: INTEGER;
  Y: T;
BEGIN
  X := 11;
  Y := X;
```

4. Bestimmen Sie mit den angegebenen Mengen

   `A = {a, b, c, d}`
   `B = {d, e, f, g}`
   die Ergebnisse von `A + B`, `A * B`, `A - B`, `A / B`.
5. Ist eine **PROCEDURE** ein Programm?
6. Welche Variablen im nachfolgenden Programm sind bezüglich der Prozedur `A` global und welche lokal?

```
MODULE T;

  VAR
    B,C: REAL;

  PROCEDURE A(X:REAL);
    VAR
      ZAEHLER: INTEGER;
  BEGIN
    ..
  END A;
```

```
BEGIN
  ..
END T.
```

7. Erklären Sie ein dreidimensionales Integerfeld `Willi` mit den Dimensionen 10, 100 und 9. Setzen Sie den Anfangsindex für alle Dimensionen auf 0.
8. Was ist am folgenden Codefragment falsch?

```
VAR
  A: ARRAY [1..10] OF CHAR;
  B: ARRAY [0..9]  OF CHAR;

BEGIN
  .
  .
  A := B;
```

9. Wieviele Byte Speicher belegen die folgenden Felder? Nehmen Sie an, daß ein CHAR-Wert ein Byte lang ist.
   (a) `ARRAY [0..100],[0..1] OF CHAR;`
   (b) `ARRAY [0..10],[0..1],[1..10] OF CHAR;`
   (c) `ARRAY [0..1] OF CHAR;`
10. Welcher Unterschied besteht zwischen einem ARRAY und einem RECORD?
11. Definieren Sie einen Triebwerke, die Anzahl der Passagiere, die Reichweite und den Namen.
12. Schreiben Sie zur vorigen Übung eine Prozedur, die die Inhalte des Records auf dem Bildschirm ausgibt.

3.15 Das Modul / Das Modulkonzept

Cluster erlaubt, ähnlich wie Modula 2, ein Programm in mehrere Module zu unterteilen. Dies ist sinnvoll, da bei großen Projekten ein einzelnes Modul doch bald unübersichtlich wird. Was liegt also näher, als alle von ihrer Funktion zusammengehörigen Prozeduren in ein Modul zu kapseln. Dies bringt neben der größeren Übersichtlichkeit den Vorteil der Wiederverwendbarkeit mit sich, da von jetzt an auch bei anderen Projekten einfach auf die schon bestehenden Prozeduren zugegriffen werden kann. Mit der Zeit kann man sich so eine eigene Bibliothek von Modulen zusammenstellen, aus der man sich nur noch bedienen, sprich importieren muß. Zuletzt läßt sich bei der Arbeit im Team durch Module eine sehr gute Aufgabentrennung erreichen, wobei die Schnittstellen[41], über die die einzelnen Module zusammenarbeiten, genau definiert sind.

3.15.1 Das einfache Modul (Hauptprogrammodul)

Dieses Modul wird durch das Schlüsselwort „**MODULE**” eingeleitet. Es enthält immer das Hauptprogramm. Aus diesem Modul kann nichts importiert werden. Beim Abspeichern muß dem Text des Hauptmoduls ein „`.mod`“ an den Dateinamen gehängt werden.

3.15.2 Das Bibliotheksmodul

Was eine Bibliothek ist, dürfte jedem klar sein. Die hier gemeinte jedoch beherbergt nicht Bücher, sondern Standardmodule, die feste Funktionen beinhalten, die von jedem Programm verwendet werden können – eben die Bibliotheksmodule.

In Kapitel 7 finden Sie alle Standardmodule aufgelistet, die zusammen mit **Cluster** ins Haus kommen. Der Zugriff erfolgt über die IMPORT-Zeilen im Programm. Selbstverständlich können auch Sie Ihre eigenen Prozeduren in Bibliotheksmodulen zusammenfassen und diese dann in Ihren Programmen verwenden.

[41] Damit ist gemeint, welche Prozeduren importiert werden können, und welche Parameter diese haben.

3.15.3 Das Definitionsmodul

Im Definitionsmodul wird die Schnittstelle zwischen einem Bibliotheksmodul und dem übergeordneten, aufrufenden Modul definiert. Variablen, Typen und Konstanten werden hier genauso wie in anderen Modulen deklariert, lediglich bei Prozeduren erscheint nur der zugehörige Prozedurkopf.

Beispiel Standardmodul `InOut`:

```
DEFINITION MODULE InOut;
...
PROCEDURE Write(c:CHAR);
PROCEDURE WriteString(VAR s:STRING);
PROCEDURE WriteLn;
...
END InOut.
```

Die im Definitionsmodul enthaltenen Objekte können von anderen Modulen importiert werden. Dazu gehören auch alle Objekte, die in diesem Definitionsmodul importiert werden. Beim Abspeichern des Textes eines Definitionsmoduls muß ein „`.def`“ an Stelle des „`.mod`“ angehängt werden.

3.15.4 Das Implementationsmodul

Im Implementationsmodul findet sich das zu einem Definitionsmodul gehörige Programm. Alle Prozeduren werden hier implementiert[42]. Da ein Implementationsmodul ebenfalls einen BEGIN- und/oder `CLOSE`-Teil haben kann, können auch Initialisierungen vorgenommen werden, bzw. in diesem Modul allozierte (belegte) Resourcen (Speicher, Fenster, etc.) wieder freigegeben werden. Auf Variablen, Konstanten, Exceptions und Typen, die im Definitionsmodul definiert worden sind, kann innerhalb des Implementationsmodules direkt zugegriffen werden. Um die Prozeduren, die im Definitionsmodul definiert wurden, zu implementieren,

[42]Die eigentliche Programmierung der Prozeduren geschieht hier.

schreibt man die komplette Prozedur einschließlich des Prozedurkopfes innerhalb des Implementationsmoduls. Dabei ist darauf zu achten, daß der Prozedurkopf identisch zu dem im Definitionsmodul ist. Implementationsmodule müssen beim Abspeichern wie Hauptmodule ein „`.mod`“ an den Dateinamen gehängt bekommen.

Beispiel:

```
IMPLEMENTATION MODULE InOut;
...
PROCEDURE Write(c:CHAR);
BEGIN
...  (Betriebssystemaufruf)
END Write;

PROCEDURE WriteString(VAR s:STRING);
  VAR i:INTEGER;
BEGIN
  FOR i:=0 TO s.len-1 DO
    Write(s.data[i])
  END;
END WriteString;

PROCEDURE WriteLn;
BEGIN
  Write(&10);
END WriteLn;
...
BEGIN
...  (* Consolenfenster öffnen *)
CLOSE
...  (* Consolenfenster schließen *)
END InOut.
```

3.15.5 Das Modulkonzept

Die importierten Module werden beim Linken[43] zu einem lauffähigen Programm verbunden. Dabei werden die BEGIN- und CLOSE-Teile so geschachtelt, daß der BEGIN-Teil eines importierten Moduls vor, der CLOSE-Teil nach dem des importierenden Moduls aufgerufen wird. Damit wird sichergestellt, daß alle Strukturen vor ihrem Zugriff initialisiert werden. Andererseits wird am Programmende zuerst der **CLOSE**-Teil des Haupmoduls, und als letzter der des Moduls, das von allen anderen Modulen direkt oder indirekt über ein anderes Modul importiert wird, ausgeführt. Somit ist sichergestellt, daß keine Resource aus einem Modul freigegeben wird, solange ein anderes Modul sie noch benötigt.

Der Definitionsteil eines Moduls muß immer vor dem Implementationsteil compiliert werden, da dieses Informationen aus dem ersten benötigt.

Um ein Programm compilieren zu können, muß bei allen importierten Modulen der Definitionsteil bereits übersetzt sein. Der Implementationsteil hingegen muß nicht unbedingt in übersetzter Form vorliegen, da zum Compilieren daraus keine Informationen benötigt werden.

Bei der Übersetzung eines Implementationsmoduls muß man kein anderes Modul neu compilieren, das Programm muß lediglich neu gebunden (gelinkt) werden. Hierbei sieht man schon einen Vorteil eines

[43] linken, Linker: Bei der Erstellung von Programmen sind nur selten alle zur Ausführung des Programms erforderlichen Prozeduren und Funktionsprozeduren auch in diesem Hauptmodul enthalten. Vielmehr werden in der Regel eine Anzahl von Standardprozeduren und -funktionen verwendet. Diese sind häufig schon übersetzt und in speziellen Dateien (Objektdateien) abgespeichert. Das gleiche gilt für Module, die getrennt erstellt und übersetzt wurden. Benutzt daher ein Programm Standardfunktionen oder Standardprozeduren, so wird bei der Übersetzung im Objektcode vermerkt, daß an bestimmten Stellen die Adressen der anderen Prozeduren eingesetzt werden müssen. Die Aufgabe des Linkers besteht nun darin, die Objektcodes verschiedener Programme zu einem Programm zusammenzufassen und dabei die Adressen der sonstigen und Standardprozeduren in die einzelnen Objektcodes einzusetzen.

Definitionsmoduls, man kann beliebig viele Änderungen an der Implementation vornehmen, ohne dabei immer auch die abhängigen Module compilieren zu müssen.

Wird dagegen ein Definitionsmodul geändert, müssen auch alle Module, die darauf zurückgreifen, sowie der eigene Implementationsteil neu compiliert werden. Würde das nicht geschehen, käme es zu Inkonsistenzen zwischen den Modulen. Man stelle sich nur einmal vor was passieren würde wenn man eine Prozedur um einen Parameter erweitert, aber alle Module die diese benützen noch einen Parameter zu wenig übergeben. Um derartige Probleme zu verhindern, erzwingt der Compiler nach einer Änderung im Definitionsmodul die Compilierung aller abhängigen Module. Unterläßt man dies, erhält man schon bald eine Fehlermeldung, in der einem ein Versionkonflikt gemeldet wird. Handelt es sich dann nur um wenige Module, kann man nun das Compilieren von Hand vornehmen. Sind es jedoch sehr viele mit vielen Abhängigkeiten, dann empfielt es sich, das Make zu verwenden, welches feststellt, welche Module neu übersetzt werden müssen, und dann alle nötigen Module in der richtigen Reihenfolge compiliert.

Ein weiterer Vorteil der Verwendung eines Definitionsmoduls ist die übersichtliche Inhaltangabe eines Moduls, sowie einer klar definierten Schnittstelle zur Benutzung der darin definierten Prozeduren. Außerdem kommt dieses Konzept mehr dem Geheimprinzip entgegen, d. h. der Programmierer, der ein Modul benutzt, braucht von dessen Implementation nichts mehr zu sehen, er kann die Objekte einfach benutzen, ohne sich darum zu kümmern, wie sie funktionieren. Arbeitet man mit einer Gruppe an einem Projekt, tauscht man nur die Texte der Definitionsmodule und die Symbol- und Objektdateien der Module untereinander aus. So ist sichergestellt, daß immer nur ein Programmierer an der Implementation etwas verändert, die anderen aber trotzdem die Module verwenden können. Desweiteren stellt eine solche Trennung auch einen „Kow How“-Schutz dar, denn schließlich will man nicht immer alle eigenen Algorithmen preisgeben.

Das Kapitel für fortgeschrittene Programmierer beschäftigt sich mit abstrakten Typen, die dazu dienen, Definitionsmodule möglichst implementationsunabhängig zu machen. Dazu werden Zeigervariablen benötigt.

Näheres schlagen Sie bitte dort nach.

3.15.6 Importgruppen

Cluster bietet die Möglichkeit, Importgruppen zu verwenden, um die Schreibarbeit beim Importieren von vielen Elementen eines Moduls zu verringern.

Diese werden durch das Schlüsselwort **GROUP** eingeleitet, gefolgt von den Bezeichnern, die zu einer Importgruppe zusammengefaßt werden sollen. Schauen Sie einfach einmal in das Modul „InOut“, speziell die Datei „`InOut.def`“. Hier existiert eine „`WriteGrp`“:

```
WriteGrp = WriteString, WriteMString, WriteEsc,
           WriteLn, WriteBuffer, WriteInt,
           WriteHex, WriteReal, WriteExpReal;
```

Um einige oder alle Bezeichner dieser Gruppe in Ihrem Programm verwenden zu können, importieren Sie einfach den Bezeichner der Gruppe. In diesem Beispiel ist es „WriteGrp“:

```
FROM InOut IMPORT WriteGrp;
```

Eine Importgruppe kann aus anderen Gruppen bestehen. Sie kann auch Bezeichner oder Importgruppen aus anderen Modulen enthalten.

Außer den einzelnen Gruppen eines Moduls existiert noch eine Gruppe mit der Bezeichnung „`All`“, welche alle Bezeichner dieses Moduls enthält und ebenso importiert werden kann.

Werden zwei Importgruppen mit gleichem Namen importiert, so gibt dies keine Namenskonflikte, solange nicht zwei gleiche Bezeichner innerhalb der Gruppen existieren.

3.16 Rekursion

Rekursion ist das Verfahren, etwas durch sich selbst zu definieren, was manchmal auch *Kreisdefinition* genannt wird und auch im täglichen Leben sehr verbreitet ist. Beispielsweise ist die natürliche Sprache rekursiv, da in einem Wörterbuch die meisten Wörter durch andere Wörter definiert sind, die selbst wieder mit den ursprünglichen Wörtern beschrieben sind. Weiterhin gibt es optische Rekursionen: In einer Zeitung sehen wir eine Werbeanzeige einer Computerfirma. Eine (hübsche?) Frau (sorry an alle Programmiererinnen, aber es ist nun mal so) beugt sich über einen Computermonitor, in diesem Monitor ist das Bild einer Frau zu sehen, welche sich über einen Computermonitor beugt, in diesem ist das Bild ....

Sind nun dann unendlich viele Frauen mit Computermonitoren (immer kleiner werdend) auf dem Bild? Natürlich nicht. Denn irgendwann ist das Auflösungsvermögen des Monitors und der Druckmaschine, welche diese Anzeige gedruckt hat, erschöpft und es ist nur ein Punkt zu erkennen.

In der Computersprache bedeutet Rekursion, daß eine Funktion oder Prozedur sich selbst aufruft. Man spricht in diesem Fall von einer rekursiven Funktion bzw. Prozedur. Diese Vorgehensweise ist natürlich nur von Fall zu Fall sinnvoll. Nicht alle Sprachen ermöglichen Rekursionen, so kennen bespielweise BASIC oder FORTRAN keine Rekursion.

Rekursionen sind sehr mächtige Programmierwerkzeuge, wenn sie richtig eingesetzt werden. Nachfolgend ein einfaches Beispiel zum Thema Rekursion, welches in die Materie einführen soll:

```
PROCEDURE R(i:INTEGER):INTEGER;
BEGIN
  IF i=0 THEN RETURN i END;
  R(i-1);
  WriteInt(i,5)
END R;
```

Diese Prozedur gibt die Zahlen 1 bis `i` auf dem Bildschirm aus. Beim Aufruf von `R` wird der Wert von `i` auf 0 geprüft. Falls `i` gleich 0 ist, springt `R` zurück und nichts weiter geschieht. Wenn jedoch `i` größer 0 ist, wird `R` erneut mit `i-1` aufgerufen, bis `i` schließlich den Wert 0 hat. Dann beginnt sich die Kette von Aufrufen zu lösen, indem jeder zum vorherigen Aufruf zurückspringt, wobei dann die Zahlen auf dem Bildschirm ausgegeben werden.

Der Schlüssel zu dieser rekursiven Prozedur lautet: Der zweite Aufruf von `R` beendet den ersten Aufruf **nicht**, sondern unterbricht seine Ausführung nur zeitlich. Deshalb geht der Wert der lokalen Variablen `i` nicht verloren und bleibt gespeichert, bis die Ausführung fortgesetzt wird. Das zweite `R` erstellt ein neues, eigenes `i` und die Variablen überschreiben ihre Werte nicht.

Nun ein weiteres Beispiel:

```
MODULE Was_macht_das;

FROM InOut IMPORT Read, Write;

PROCEDURE Zeichen;

VAR
  ch:  CHAR;

BEGIN
  Read(ch);
  IF ch # " " THEN
    Zeichen
  END;
  Write(ch)
END Zeichen;

BEGIN
  Zeichen
END Was_macht_das.
```

Daß es sich hierbei um eine Rekursion handelt, kann man an dem Aufruf der Prozedur „`Zeichen`" innerhalb der Prozedur „`Zeichen`" sehen. Die Prozedur ruft sich also selbst auf.

Angenommen, wir geben die Buchstaben „`abcd`" ein und danach eine Leerstelle. Die Eingabe eines Zeichens wird solange fortgesetzt, bis ein Leerzeichen eingegeben wurde, danach werden alle Zeichen rückwärts wieder ausgegeben.

Nun die Programmbeschreibung: Bei Eingabe von „`a`" ist „`ch`" kein Leerzeichen, also wird die Prozedur „`Zeichen`" erneut aufgerufen. „`ch`" ist nicht leer („`b`", also erneuter Aufruf von „`zeichen`". Dies geschieht solange, bis ein Leerzeichen eingegeben wurde, dann ist ja „`ch`" = „`" "`" und die Bedingung wird wahr. Nun werden in umgekehrter Reihenfolge die „`Write`"-Anweisungen ausgeführt.

In dieser Art ist die Rekursion natürlich nur dann möglich, wenn bei jedem Aufruf der Prozedur „`Zeichen`" eine neue Variable „`ch`" bereitgestellt wird. Dies ist dadurch gewährleistet, daß „`ch`" eine lokale Variable und somit nur innerhalb der Prozedur definiert ist. Obwohl die Variablen gleiche Namen haben, sind sie doch unterschiedlich.

Wälzt man die Fachliteratur, so findet man häufig zum Thema Rekursion ein Beispielprogramm, welches sich mit der Berechnung der Fakultät beschäftigt. Es ist bisher offensichtlich nicht gelungen, ein ähnlich anschaulich und zugleich übersichtliches Beispiel ausfindig zu machen. Der Klassiker wird hier einfach übernommen.

Er ist einfach zu schön! Das soll jedoch nicht darüber hinwegtäuschen, daß die Rekursion ein mächtiges Programmierwerkzeug ist, was anderen Lösungen an Geschwindigkeit meistens in nichts nachsteht, jedoch vom Speichervolumen der Basismaschine oft eine Menge abverlangt.

Das folgende Beispiel ist ein komplettes **Cluster**-Programm zur Berechnung der Fakultät auf rekursivem Wege. Geben Sie es auf Ihrem Rechner ein, probieren Sie es aus, verbessern Sie es:

Denkbar wäre eine einfache Abfrage am Ende, ob eine weitere Fakultät berechnet werden soll. Benutzen Sie dazu z. B. die **REPEAT**... **UNTIL**-Struktur.

```
MODULE Fakultaet_rekursiv;
FROM InOut IMPORT ClearWindow, WriteString, ReadInt,

                  WriteInt, WriteLn;
VAR n:INTEGER;

PROCEDURE fak(n:INTEGER):INTEGER;

 | Dies ist die Funktionsprozedur
BEGIN

    IF n=0 THEN RETURN 1 | Funktionswerte werden mit
    ELSE RETURN n*fak(n-1) | RETURN an den aufrufenden
    END | Programmteil zurückgegeben
END fak;
BEGIN | Hier beginnt das Hauptprogramm

    ClearWindow;
    WriteString("Eine natürliche Zahl eingeben:  ");
    ReadInt(n);
    WriteLn;
    WriteString("Fakultät:  ");
    WriteInt(fak(n));
| Hier wird die Funktion aufgerufen
END Fakultaet_rekursiv.
```

Zum Abschluß noch ein Programm für unsere Ungeduldigen, bzw. die Graphikfreunde unter Ihnen. Probieren Sie unter anderem die Werte (5, 121, 2) oder (10, 89, 1) aus.

```
MODULE Spirale;

FROM GfxScreen IMPORT OpenScreen, CloseScreen, Screen;
FROM GfxTurtle IMPORT Start, Down, Forward, Left;
FROM GfxInput IMPORT WaitUser, UserAct, UserActions;
FROM InOut IMPORT ReadFFP, WriteLn, WriteString;

VAR
  l,w,z:  FFP;
```



```
  meinSchirm:  Screen;
  Action:  UserAct;

PROCEDURE Seite(lang, winkel, zunahme:  FFP);

BEGIN
  IF lang < 200.0 THEN
    Forward (lang);
    Left(winkel);
    lang := lang + zunahme;
    Seite(lang, winkel, zunahme)
  END
END Seite;

BEGIN
  Action := userMouse;
  WriteString("Spiralen - Demo für Rekursion");
  WriteLn;
  WriteLn;
  WriteString("Eingabe der Anfangsseite:  ");
  ReadFFP(l);
  WriteLn;
  WriteString("Eingabe der Winkeldrehung:  ");
  ReadFFP(w);
  WriteLn;
  WriteString("Eingabe der Seitenzunahme:  ");
  ReadFFP(z);

  OpenScreen(meinSchirm,2,TRUE,FALSE);
  Start(meinSchirm,320.0,128.0,0.0,3);
  Down;
  Seite(l,w,z);
  WaitUser(meinSchirm,Action);
CLOSE
  CloseScreen(meinSchirm)
END Spirale.
```

In dem Programm werden einige Prozeduren der Standardmodule verwendet, deren Beschreibung Sie im Kapitel 7 finden können.

Das Programm wartet nach Ausführung der Zeichenoperationen auf einen Klick mit der linken Maustaste. Weitere Erläuterungen sollen zu dem Programm nicht gegeben werden. Probieren Sie es aus und veränderen Sie es.

Der Sinn und Einsatz von Rekursionen sollte nun erheblich klarer als vorher sein.

Wichtig bei rekursiven Algorithmen ist, daß die Rekursion irgendwann abgebrochen wird. Sonst würde sich die Prozedur immer weiter selbst aufrufen, bis ihr der Speicher zur Einrichtung von neuen lokalen Variablen ausgeht. Dies führt zu einem Laufzeitfehler wegen „Stacküberlauf“.

Übungen 5:

1. Was ist an der folgenden rekursiven Routine falsch?

```
PROCEDURE A(I: INTEGER): INTEGER;
BEGIN
  RETURN A(I-1);
  WriteInt(I,5);
END A;
```

2. Geben Sie zwei Gründe an, warum getrennte Compilierung nützlich ist.

3. Erklären Sie den Unterschied zwischen einem Definitons- und einem Implementationsmodul.

4. Was ist bei dem angegebenen Codefragment falsch?

```
MODULE BSP1; | Das ist falsch!
  IMPORT InOut;
BEGIN
  WriteString("Cluster la vista!");
  WriteLn;
END A.
```

3.17 Pointer

Neben dem offenen Array, das wir bisher als einzige dynamische Datenstruktur, d. h. eine Datenstruktur mit variabler Größe, kennengelernt haben, gibt es ferner auch die sogenannten Zeiger (Pointer). Hier zeigt eine Zeigervariable, z. B. `x`, auf einen Wert, der mittels „`x^`“ angesprochen wird. Der Computer reserviert also nicht einen bestimmten Speicherbereich für die Daten, sondern legt evtl. anfallende Daten dort ab, wo zuvor vom System Platz angefordert wurde.

Weitere Informationen zu diesem Thema finden Sie im Kapitel für den fortgeschrittenen Programmierer.

3.18 Methoden

Hat man mehrere Prozeduren, die die gleiche Funktion für verschiedene Typen erfüllen, ist es unangenehm, jeder Prozedur einen anderen Namen geben zu müssen. Daher besteht die Möglichkeit, beliebig viele Methoden mit dem gleichen Namen zu definieren, wenn Sie einen `RECORD` oder einen Zeiger auf einen `RECORD` als ersten Parameter erwarten. C++-Programmierer sollten spätestens jetzt glänzende Augen bekommen.

Hier einige Beispiele aus dem Modul DosSupport:

```
METHOD Get(VAR data:  FileData;
           REF path:  STRING)

METHOD Get(VAR list:  DirList;
           REF path:  STRING;
           REF pattern:  STRING:="#?";
               type:= DirSelectType:{selectDirs,selectFiles};
               context:  ContextPtr := NIL);
```

Diese Methoden werden nicht direkt importiert und aufgerufen, sondern sie werden qualifiziert über eine Variable des Typs des ersten Parameters benutzt z. B.:


```
VAR
  Dir      :  DirList;
  FileInfo :  FileData;
BEGIN
  FileInfo.Get("s:startup-sequence");
  Dir.Get("DH0:");
```

3.19 Prozedurtypen, Prozedurvariablen

Neben den bisher beschriebenen Typen gibt es noch einen weiteren, nämlich den Prozedurtypen. Einer Variablen eines solchen Typs kann man eine Prozedur zuweisen, und von da an diese Prozedur über die Variable aufrufen.

Wahrscheinlich werden Sie sich nun fragen, wozu das gut sein soll, man kann die Prozedur doch auch direkt aufrufen. Damit haben Sie natürlich recht, aber denken Sie doch einmal an folgendes Problem: An mehreren Stellen in Ihrem Programm wird eine bestimmte Prozedur aufgerufen. Nun soll abhängig von einer Benutzereingabe anstelle dieser Prozedur eine andere aufgerufen werden. Mit dem was Sie bisher kennen, würden Sie wahrscheinlich ein Bool-Flag definieren, dieses abhängig von der Eingabe setzen, und dann bei jedem Aufruf durch eine IF-Abfrage dieses Flag auswerten und die richtige Prozedur aufrufen.

Das heißt bei jedem Aufruf wäre ein zusätzlicher Vergleich nötig. Falls es mehr als nur zwei mögliche Prozeduren geben soll, wird das ganze dann schnell aufwendig. Viel einfacher wäre es hier doch, bei jedem Aufruf eine Prozedurvariable zu verwenden, und dieser nach der Benutzereingabe die richige Prozedur zuzuweisen. Danach wird dann immer diese Prozedur aufgerufen. Wichtig: Prozedurvariablen können nur globale Prozeduren zugewiesen werden, keine Prozedurlokalen.

Eine anderes Anwendungsbeispiel wäre ein `ARRAY` von Prozedurvariablen, so daß man abhängig von einer Zahl als Index eine bestimmte Prozedur aufrufen kann.

Der einfachste Prozedurtyp ist `PROC`, der im Modul `System` definiert ist. Ihm können alle Prozeduren, die keine Parameter erwarten, und keinen Rückgabewert haben zugewiesen werden.

Formale Prozedurparameter werden folgendermaßen definiert:

```
TYPE
 NewProc  = PROCEDURE(VAR a : INTEGER;b : CHAR);
 Proc2    = PROCEDURE(r : REAL):BOOLEAN;
```

Dabei können als Übergabe- und Rückgabeparameter alle Typen verwendet werden, die auch sonst bei Prozeduren möglich sind. Einer Variablen vom Typ `NewProc` kann man alle Prozeduren zuweisen, die als ersten Parameter einen Varparameter vom Typen `INTEGER` und als zweiten einen Werteparameter vom Typen `CHAR` haben. Einer Variablen vom Typ `Proc2` kann man alle Prozeduren zuweisen, die als ersten Parameter einen Parameter vom Typen `REAL` haben, und einen Boolwert zurückgeben. Die Parameternamen haben dabei keine Bedeutung, sie dienen lediglich der besseren Lesbarkeit und der Dokumentation.

Da sichergestellt ist, daß an eine Prozedurvariable nur Prozeduren zugewiesen werden können, die der Typdefinition entsprechen, erweitert sich deren Anwendungsgebiet noch einmal. Denn nun kann man davon ausgehen, daß der Benutzer der Prozedurvariablen beim Aufruf die richtigen Parameter übergeben, bzw. ihr nur richtige Prozeduren zuweisen wird, da sonst der Compiler einen Fehler meldet.

Damit eignen sich Prozedurtypen auch ideal als Übergabeparameter für Prozeduren. Man kann der Prozedur beim Aufruf eine Prozedur mitgeben, mit der diese dann arbeiten kann. Ein Beispiel ist im Modul `DosSupport` zu finden, die Prozedur `WorkOnFiles`. Diese bekommt eine Prozedur bestimmten Typs übergeben, die sie dann nacheinander mit alle Dateinamen eines Verzeichnises aufruft.

Bei der Verwendung von Prozedurvariablen als *Übergabeparameter* haben Sie in **Cluster** sogar die Möglichkeit, eine lokale Prozedur zu übergeben, dies geht bei keiner anderen Sprache. Der Vorteil hier ist dadurch gegeben, daß die lokale Prozedur auch auf die lokalen Variablen und Übergabeparameter der ihr übergeordneten Prozedur zugreifen kann, was bei der Übergabe einer globalen Prozedur nicht möglich wäre. Hierzu als Beispiel ein Ausschnitt aus dem Modul DosSupport:

```
PROCEDURE Delete(REF name : STRING;all : BOOLEAN);
```



```
VAR list        : DirList;
    path        : STRING(120);
    PatternStr  : STRING(32);
    deleteRoot  : BOOLEAN:=FALSE;
    c           : CHAR;

  PROCEDURE Kill2(REF path,name : STRING;data : FileData);
  BEGIN
    TRY
      DosDeleteFile(path,name);
    EXCEPT
      OF DirectoryNotEmpty THEN
        WorkOnFiles(SubDir(path,name),"",
                    Kill2,TRUE,
                    {selectDirs,selectFiles},TRUE);
        DosDeleteFile(path,name);
      END;
    END
  END Kill2;

  PROCEDURE Kill(REF path,name : STRING;data : FileData);
  BEGIN
    TRY
      DosDeleteFile(path,name)
    EXCEPT
      OF DirectoryNotEmpty THEN
        IF all THEN  | Achtung hier ist die interessante Stelle
          WorkOnFiles(SubDir(path,name),"",
                      Kill2,TRUE,
                      {selectDirs,selectFiles},TRUE);
          DosDeleteFile(path,name);
        END;
      END;
    END
  END Kill;
```

```
BEGIN
  .
  .
  WorkOnFiles(name,"",Kill,FALSE,{selectDirs,selectFiles},TRUE);
  .
  .
END Delete;
```

Im **BEGIN**-Teil wird der Prozedur `WorkOnFiles` die Prozedur `Kill` übergeben. Da die Prozedur `Kill` eine lokale Prozedur ist, kann Sie auf den Parameter `all` der Prozedur `Delete` zugreifen. Die Konstruktion **TRY**...**EXCEPT** wird nun im nächsten Kapitel erklärt.

3.20 Ausnahmebehandlung, Exceptions

Um anzuzeigen, ob eine Prozdur ihre Aufgabe ordnungsgemäß verrichtet hat, hatte man in Modula 2 nur die Möglichkeit, einen Boolwert zurückzugeben. Dieses Verfahren geht solange gut, wie man alleine arbeitet, und man sich selbst dazu zwingt, diesen Boolwert auch immer abzufragen, und ihn nicht im Vertrauen darauf, daß es schon gut gehen wird, einer Dummyvariablen zuweist oder ein `FORGET` davor schreibt. Sonst kann es nämlich leicht passieren, daß eine Fehlermeldung verloren geht und dadurch großer Schaden angerichtet wird.

Um derart versteckte Fehler von vornherein auszuschließen, sollte man im Falle eines Fehlers eine Exception auslösen. Bei diesem Konzept hat nämlich das Nichtbeachten einer möglichen Fehlermeldung einen Programmabbruch mit Meldung zur Folge.

Bevor man eine Exception benützen kann, muß man sie definieren. Dies kann an jeder Stelle[44] des Programms geschehen. Eingeleitet wird diese Definition durch das Schlüsselwort **EXCEPTION**. Darauf folgt der Bezeichner, unter dem man in Zukunft auf diese Exception zugreifen

[44] Außer innerhalb eines BEGIN-Teils.

möchte und, durch einen Doppelpunkt getrennt, ein String oder eine Nummer. Im Falle, daß diese Exception ausgelöst und nicht behandelt wird, wird dieser String bzw. die Nummer als Laufzeitfehlermeldung ausgegeben. Dabei sollte man immer Strings verwenden. Die Nummern finden lediglich in Verbindung mit Dos-Fehlermelungen eine Verwendeung, da diese als Zahlen definiert sind.

```
EXCEPTION
  UngueltigerName  : "Es wurde ein ungültiger Name eingegeben";
  Fehler1          : 1;
```

Die Exceptiondefinition wird durch das Auftreten eines anderen Programmelements beendet, wie z. B. eine Variablen- oder Typdefinition. Nachdem Sie nun wissen, wie man Exceptions definiert, fragen Sie sich warscheinlch, was man nun damit machen kann. Eine Anwendungsmöglichkeit ist `ASSERT`.

3.20.1 ASSERT

Mit der Prozedur `ASSERT` wird eine Bedingung kontrolliert, die für das korrekte Weiterlaufen des Programms unbedingt erforderlich ist. So könnte kontrolliert werden, ob die letzte Speicheranforderung vom Betriebssystem erfüllt worden ist, oder ob ein Fenster geöffnet wurde. Nachfolgendes Beispiel zeigt eine einfache Anwendung für ASSERT:

```
MODULE Was_macht_das;
FROM InOut IMPORT Read, Write;

VAR
  x,y:  INTEGER;

EXCEPTION
  diverror : "Division durch Null nicht definiert!";

BEGIN
  x := 5;
  x := x -5;
  ASSERT(x#0,diverror);
  y := 5 DIV x
END Was_macht_das.
```

Wenn die angegebene Bedingung *nicht* erfüllt, also `x = 0` ist, wird die Exception ausgelöst. In diesem speziellen Beispiel wird die Fehlermeldung `"Division durch Null nicht definiert!"`[45] ausgegeben, zusammen mit der Zeilennummer, in der die Exception auftrat. Außerdem wird das Programm an dieser Stelle durch einen Sprung in den **CLOSE**-Teil beendet. Die Ausgabe geschieht allerdings nur wenn Sie das Programm mit Run gestartet, oder die Prozedur `WriteException` im **CLOSE**-Teil ihres Hauptmoduls aufgerufen haben. Falls Sie sich nun fragen, daß es wohl kaum sinnvoll sein kann ein Programm bei jedem Fehler zu beenden, haben Sie natürlich Recht. Der Programmabbruch ist nur die Folge, wenn man die Exception nicht abfängt, dies ist durch **TRY**...**EXCEPT** möglich.

3.20.2 TRY..EXCEPT

Unter Ausnahmesituationen versteht man Zustände beim Ablauf eines Programmes, die gesondert berücksichtigt werden müssen.

Ausnahmesituationen (Exceptions) können durch Programm- bzw. Datenfehler, systembedingte Probleme wie Speichermangel oder durch benutzerdefinierte Fehlersituationen entstehen. Beispiele dafür sind: Mathematische Fehler (Division durch Null), Dateifehler (Ende der Datei), Diskettenfehler (Disk schreibgeschützt, Diskette voll, Schreib/Lesefehler, ...). All dies sind Beispiele für Situationen, unter bei denen eine Exception ausgelöst werden kann, um auf den Fehler aufmerksam zu machen.

Ist kein Exception-Handler[46] installiert, kann das Programm nur durch einen Abbruch reagieren.

Im Zusammenhang mit Ausnahmebehandlungen bekommt die ansonsten verpönte LOOP-Schleife (siehe unter 3.12.4.2 ab Seite 68) wieder eine Berechtigung. Hier ist die Abbruchbedingung gut zu sehen und damit ein sauberes Programm gewährleistet.

[45] Dieses Beispiel ist allerdings nicht aus der Praxis gegriffen, da der Compiler automatisch einen Check durchführt, ob durch 0 geteilt wurde, und falls dem so ist, eine im Modul `Exceptions` definierte Exception auslöst.

[46] Exception-Handler= Ein bestimmter Programmteil, welcher Ausnahmesituationen (Ende einer Datei, Division durch Null, ...) behandelt

Nachfolgende Programmfragmente sind aus einem größeren Programm, welches bestimmte ASCII-Codes in die Codes eines anderen Computers wandelt. In beiden Beispielen werden Exception-Handler (mittels TRY...EXCEPT) verwendet:

Im ersten Beispiel wird ein File geöffnet. „`Q_OpenInFile`“ stammt aus dem Standardmodul „`FileSystem`“. Näheres über die Funktionsweise dieser Prozedur können Sie im Kapitel 7 nachlesen. Wichtig ist nur, diese Funktion löst im Fehlerfall eine Exception aus.

Die potentielle Ausnahmesituation ist zwischen **TRY** und **EXCEPT** aufgeführt. Nach den Schlüsselwörtern **OF** stehen die vom Programmier vorgesehenen Ausnahmezustände[47] und die entsprechenden Ausnahmebehandlungsroutinen und -anweisungssequenzen. Tritt nun ein Ausnahmezustand auf, der nicht vorgesehen war, so wird der ELSE-Zweig ausgeführt. Danach wird der nächst höher gelegene Exception-Handler aufgerufen. Ist keiner vorhanden, bricht das Programm mit einem Laufzeitfehler ab.

```
TRY
  Q_OpenInFile(FileName,4096);
EXCEPT
  OF ObjectNotFound THEN
     WriteString("Kein Quellfile gefunden");
     WriteLn;
  END;
  OF ReadProtected THEN
     WriteString("Quellfile Readprotected");
     WriteLn;
  END;
END;
```

Im zweiten Beispiel wird ein Exception-Handler mit `ELSE`-Zweig verwendet. In diesem Programmfragment werden einzelne Zeichen aus einer Datei gelesen und in eine andere Datei geschrieben, bis das Ende der einen Datei erreicht ist (`EOF`[48]). Tritt ein anderer Fehler auf, wird dies gemeldet, die Exception aber nach außen weitergegeben.

[47] Es ist auch möglich hinter einem **OF** mehere Exceptions durch Komma getrennt anzugeben.

[48] End Of File= Ende der Datei

```
TRY
  LOOP
    Q_Read(c);
    IF (c = &10) OR (c > &128) THEN
      IF KEY c
        OF "ü" THEN c := &129 END;
        OF "ö" THEN c := &148 END;
        OF "ä" THEN c := &132 END;
        OF "Ü" THEN c := &154 END;
        OF "Ö" THEN c := &153 END;
        OF "Ä" THEN c := &142 END;
        OF "ß" THEN c := &225 END;
        OF &10 THEN Q_Write(&13) END;
      END;
    END;
    Q_Write(c);
  END;
EXCEPT
  OF EOF THEN Q_CloseInFile; Q_CloseOutFile END;
ELSE
  Q_CloseInFile; Q_CloseOutFile;
END;
```

3.20.2.1 RAISE, RAISE2

Außer mit den vorgenannten Methoden läßt sich eine eigene Exception auch mit Hilfe von `RAISE(x)` auslösen.

Angenommen Sie haben eine Exception „`divError`“ mit einem Fehlertext definiert („Division durch Null ist nicht definiert!“). Wenn Sie nun beispielsweise innerhalb einer Prozedur feststellen, daß eine Division nicht möglich ist (der Dividend ist gleich Null), so können Sie mit `RAISE(divError)` die Exception auslösen. Haben Sie um diese Prozedur einen Exception-Handler installiert wie oben angegeben, so kann diese normal behandelt werden. Wird die Exception nicht behandelt oder ist kein Exception-Handler installiert, so wird ein Laufzeitfehler ausgelöst. Hierbei wird der Modulname und die Zeilennummer angegeben, wo der Fehler auftrat.

Besonders im Zusammenhang mit Bibliotheksmodulen, in denen Exceptions meist nicht Fehler in einer darin definierten Routine, sondern

Fehler im Zusammenhang mit deren Aufruf anzeigen, interessiert es einen Programmierer nicht, daß eine Exception in dieser Zeile in jenem Modul aufgetreten ist. Er möchte wissen, an welcher Stelle er diese Prozedur aufgerufen hat, die die Exception ausgelöst hat. Hierfür ist `RAISE2()` vorgesehen. Wenn `RAISE2()` verwendet wurde, in den Standardmodulen ist dies häufig der Fall, bekommen Sie im Falle einer Exception angezeigt, in welchem Ihrer Module und in welcher Zeilennummer die Prozedur aufgerufen worden ist, die die Exception ausgelöst hat, und nicht die Zeilennummer in dem Modul, in dem die Prozedur definiert wurde. Zur gleichen Verwendung existiert auch ein `ASSERT2`.

3.20.2.2 Exception-Gruppen

In manchen Fällen kann ein Fehler durch eine Vielzahl von Exceptions angezeigt werden. Ein Beispiel hierfür ist das Modul `T_Dos`, da es viele Möglichkeiten gibt, warum auf eine Diskette nicht schreibend zugegriffen werden kann: Keine Diskette im Laufwerk, Schreibgeschützt, etc. . Nun interessiert oft allerdings nur, daß etwas schiefgegangen ist; so wäre es sehr aufwendig, jedesmal alle möglichen Exceptions abzufragen. Aus diesem Grund besteht bei **Cluster** die Möglichkeit, Exceptiongruppen zu definieren.

Diese werden genau wie normale Exceptions nach dem Schlüsselwort `EXCEPTION` definiert. Dabei wird ein Bezeichner angegeben, über den man in Zukunft auf die Exceptiongruppe zugreifen will, gefolgt von den Bezeichnern, die zusammengefaßt werden sollen.
Beispiel:

```
ErrorGrp = WriteError, ReadError, EOF,
           SeekError, ObjectNotfound;
```

Innerhalb einer `TRY...EXCEPT` Anweisung können Sie nun durch Verwendung des Gruppennamens an Stelle einer Exception alle darin enthaltenen Exceptions abfangen. Wollen Sie eine oder mehrere davon getrennt behandeln, dann sollten Sie so vorgehen:

```
  ...
EXCEPT
  OF WriteError THEN
                  <Anweisung1>
                END
  OF ErrorGrp   THEN
                  <Anweisung2>
                END
  ...
```

Da die einzelnen Exception-Handler von oben nach unten durchgegangen werden, wird im Falle eines `WriteError`s `Anweisung1` ausgeführt, obwohl `WriteError` auch in `ErrorGrp` enthalten ist.

3.20.3 Resourcenverwaltung

Cluster bietet ein Kontextsystem[49], mit dem Resourcen sehr komfortabel und sicher verwaltet werden können.

Alle Resourcen (Speicher, Files, etc.) können zu sogenannten Kontexten alloziert[50] werden, welche danach mit einem Schlag wieder freigegeben werden können. Man kann nun zum einen bei jeder Allozierung einen eigenen Kontext angeben. Gibt man diesen nicht an, wird zum aktuellen Kontext alloziert, welcher am Programmende wieder freigegeben wird.

Weiß man nun, daß man über einen bestimmten Bereich Resourcen benötigt und diese danach wieder freigegeben werden sollen, bietet sich die **TRACK**...**CLOSE**...**END** Konstruktion an:

[49]Kontext= Zu einem bestimmten Zusammenhang gehörig
[50]alloziert= angelegt werden, einfacher: im Speicher Platz reservieren


```
TRACK
  Allozierungen
CLOSE
  Anweisungen
END
```

Diese Anweisung bewirkt, daß ein neuer Kontext erzeugt und dieser zum aktuellen Kontext erklärt wird. Am Ende des Bereiches wird er wieder freigegeben und der vorherige aktuelle Kontext wird zum aktuellen Kontext gemacht.

Der Gag ist nun, daß der Kontext auch dann zurückgesetzt wird, wenn innerhalb von **TRACK**...**CLOSE** eine Exception oder ein Laufzeitfehler auftritt. Der **CLOSE**-Teil wird immer durchlaufen, ob eine Exception auftrat oder nicht. Dies dient dazu, Resourcen, die nicht über die Kontexte verwaltet werden, wieder freizugeben. Er kann auch entfallen.

Eine sehr beliebte Konstruktion ist ein **TRACK**...**END**, das von einem **TRY**...**EXPECT** umgeben ist, da man im Falle einer Exception nicht alle Resourcen selbst freigeben muß.

Dieses System der Kontexte funktioniert dabei ohne Performanceverlust, im Gegensatz zu einem Garbagecollector[51].

Wichtiger Hinweis:

Sie sollten sehr vorsichtig sein, daß Sie innerhalb eines **TRACK**...**END**-Konstruktes keine Strukturen allozieren, welche außerhalb benötigt werden.

Ist dies nötig, so sollte man bei der Allozierung einen anderen Kontext als den aktuellen verwenden. Näheres hierzu ist im Modul `Resources` im Kapitel 7 zu finden.

[51]Verschiedene, z. T. neuere Programmiersprachen verwenden einen sog. Garbagecollector, welcher allen „Speichermüll“ aufsammelt, den der Programmierer „vergessen“ hat. In anderen Worten: Es werden noch belegte aber nicht mehr benutzte Speicherstücke aufgespürt und wieder freigegeben. Diese Aufgabe ist aufwendig und benötigt deshalb Zeit.

3.20.4 Laufzeitchecks

Eine weitere Möglichkeit, wodurch eine Exception ausgelöst werden kann, sind Laufzeitchecks, die der Compiler durchführt, soweit man diese nicht abgeschaltet hat. Dabei wird unter anderem kontrolliert, ob man versucht durch Null zu teilen, ob das Ergebnis einer Multiplikation noch in den verwendeten Variablentyp paßt, oder ob bei einer Zuweisung an einen Unterbereichstypen eine Bereichverletzung vorkam. Welche Exception im Einzelnen ausgelöst wird, lesen Sie in der Beschreibung des Moduls `Exceptions`, Kapitel 7, nach. Wenn man eine solche Exception abfängt, sollte man sehr vorsichtig sein, da diese Exceptions normalerweise schwerwiegende Fehler anzeigen. Außerdem ist wichig zu wissen, daß alle Exceptions, deren Check sich durch einen Compilerswitch ausschalten läßt, nur dann ausgelöst werden, wenn dieser eingeschaltet ist. Wenn es also unbedingt nötig ist, schalten Sie den Check bereichsweise ein.

3.20.5 HALT

Will man ein Progamm ganz einfach nur abrechen, kann man diese Standardprozedur verwenden. Sie führt dazu, daß das Programm augenblicklich abgebrochen wird, und der **CLOSE**-Teil ausgeführt wird. Außerdem hat man die Möglichkeit, einen `DOS`-Returncode (0, 5, 10 bis 20 je nach schwere des Fehlers) auf diese Weise zurückzugeben. Aufruf: `HALT(<Returncode>);`

3.21 FORWARD für Konstanten

Nicht nur Prozeduren, sondern auch komplexe Konstanten – also Records und Arrays – lassen sich `FORWARD`-deklarieren. Dies hat folgende Anwendungen: Zum einen kann man Konstanten in einem Definitionsmodul `FORWARD`-deklarieren, und die eigentliche Wertzuweisung erst im Hauptmodul vornehmen. Dies hat den Vorteil, daß man nun im Implementationsteil den Wert der Konstanten ändern kann, ohne danach alle abhängigen Module neu compilieren zu müssen. Eine weitere Möglichkeit

ist die Verwendung zur Definition von konstanten, durch Zeiger verketteten Strukturen. Ein Beispiel dafür folgt gleich. Eine Konstante wird dadurch vordefiniert, daß man statt dem Wert einfach nur den Typ der Konstanten angibt:

```
TYPE
  ErrField = ARRAY [10] OF STRING(20);
CONST
  Meldung = ErrField;
```

Weiter hinten, oder im Implementationsteil eines Programmes muß man dann die selbe Konstante noch einmal definieren, diesmal jedoch mit einer Wertzuweisung. Nun noch ein Beispiel für eine verkettete konstante Struktur; in ähnlicher Weise kann man auch Gadgetlisten aufbauen:

```
TYPE
  ElemPtr    = POINTER TO Elem;
  Elem       = RECORD
                 prev,
                 next  : ElemPtr;
                 data  : INTEGER;
               END;
CONST
  Elem2     = Elem;
  Elem1     = Elem:(prev=NIL,next=Elem2'PTR,data=3);
  Elem2     = Elem:(prev=Elem1'PTR,next=NIL,data=4);
```

Sicherlich werden Sie in den meisten Fällen auch ohne die `FORWARD`-Deklarierung von Konstanten auskommen, jedoch ist es machmal sehr praktisch, wenn man diese Möglichkeit hat.

3.22 Variante Records

Dieses Sprachelement stammt noch aus der Zeit, in der man um jedes Byte Speicher kämpfen mußte, und möglichst wenig verschwenden durfte.

Zwar besitzen heutige Rechner meist genug Speicher, denoch sollte man nicht zu verschwenderisch damit umgehen. Dennoch sollte man dieses Verfahren nur in Extremfällen verwenden. Oft kommt es nämlich vor, daß man in einem Record verschiedene Information speichert, jedoch nicht alle Elemente zur gleichen Zeit benötigt. Da jedoch auch nicht benutzte Elemente Speicher belegen, besteht die Möglichkeit sich mehrere Recordelemente ein Speicherstück teilen zu lassen, so daß der Record maximal so groß ist wie die Summme der größten gleichzeitig aktiven Elemente. Da jedoch nicht kontrolliert werden kann, ob auf einen solchen Record korrekt zugegriffen wird, sollte man diese Konstruktion nur dann verwenden wenn man jedes einzelne Byte benötigt. Ansonsten sollte man aufgrund der großen Gefahr von Fehlern bei der Benutzung darauf verzichten.
In der Hoffnung alle Fragen damit zu klären folgt nun ein Beispiel:

```
TYPE
  Geschlecht = (maennlich,weiblich);
  Person     = RECORD
                 namen,
                 vornamen : STRING(20);
                 IF KEY geschl : Geschlecht
                   OF maennlich THEN dienstgrad   : STRING(20);
                   OF weiblich  THEN maedchenname : STRING(20);
                 END;
               END;
```

Entschuldigen Sie dieses etwas chauvinistische Beispiel, aber daran kann das Prinzip sehr gut gezeigt werden. Da eine Person im Normalfall entweder nur mänlich oder nur weiblich sein kann, ist es unnötig immer sowohl das Element `dienstgrad` und das Element maedchename in dem Record zu haben. Daher gilt in diesem Beispiel abhängig von dem Element `geschl` entweder das eine oder das andere Element. Dabei können sich beide den gleichen Platz teilen. Bei der Wertzuweisung muß man nur darauf achten, abhängig von `geschl` auf das richtige Element zuzugreifen. In unserem Beispiel würde zwar nicht viel passieren, doch stellen Sie sich einmal vor, was passieren würde, wenn sich ein String

und eine Integerzahl sich einen Platz teilen, und man das falsche Element ausliest. Also immer erst den Schlüsselwert prüfen, bevor man einen varianten Record ausliest.

In machen Systemstrukturen findet man auch variante Records ohne einen explizieten Schlüsselwert, was bei „C“ sehr beliebt ist. In Cluster sieht dies folgendermaßen aus:

```
TYPE
  MemEntry     = RECORD
    IF KEY :INTEGER
      OF 1 THEN reqs : MemReqSet
      OF 2 THEN addr : ANYPTR
    END;
    length : LONGCARD;
  END;
```

Bei diesen kann man einfach den Elementen etwas zuweisen, die in diesem Falle benötigt werden. Dabei ist jedoch wieder darauf zu achten, daß man nur auf die Elemente eines Falles zugreift, da man sonst die erstaunlichsten Ergebnise erhält. Diese Records werden normalerweise jedoch meist Prozeduren übergeben, denen man noch ein zusätzliches Flag übergibt, an dem diese erkennen können, um welchen Typ von Record es sich handelt. Beim Auslesen eines solchen Records sollte normalerweise immer aus einem anderen Element des Record hervorgehen um welche Art Record es sich handelt.

3.23 Schlußbemerkung

Sie sollten inzwischen in der Lage sein, die Informationen dieses Handbuchs selbständig zu interpretieren und für Ihre Zwecke zu verwenden. Gemeint ist damit insbesondere der Sprachreport und die Modulbibliothek, deren Dokumentation Sie weiter hinten in diesem Buch finden. Nun, dies wird wohl eher der Idealfall sein. Meistens werden Sie sich das eine oder andere Kapitel nochmals vornehmen müssen. Weiterhin können Sie auch den ausführlichen Index im Anhang dieser Dokumentation zur Hilfe nehmen, wenn Sie etwas bestimmtes suchen.

Wir hoffen, daß wir Ihnen wenigstens ansatzweise behilflich sein konnten, einen sanften Einstieg in die Welt der Programmierung zu finden. Möglicherweise habe wir doch zuviele Fremdwörter gebraucht und unseren Stil zu trocken gewählt.

Sollten Sie Verbesserungsvorschläge oder Wünsche haben, die dieses Kapitel betreffen, so scheuen Sie sich nicht davor, uns zu kontaktieren. Wir sind für jede Resonanz dankbar.

3.24 Beispiele:

Nun zum Abschluß möchten wir Ihnen anhand eines größeren Beispiels die Möglichkeit geben, Ihre Clusterkenntnisse zu kontrollieren, und Ihnen vielleicht einige Anregungen für eigene Programme geben.

Beim ersten Beispiel handelt es sich um eine Computerversion der bekannten Schiebepuzzles, bei denen man innerhalb eines Quadrates kleine Puzzleteile, durch Verschieben, in die richtige Reihenfolge zu bringen hat. In unserem Beispiel erfolgt dies durch Anklicken der entsprechenden Teile mit der Maus. Das Programm wird durch einen Tastendruck abgebrochen.

Das Programm verwendet für seine Graphikausgaben Routinen der `Gfx`-Module aus den Standardmodulen, die eine sehr einfache Benutzung der Graphikhardware des Amigas erlauben, ohne daß man sich mit dem Betriebssystem auseinandersetzen muß. Wenn Sie nähere Informationen zu den einzelnen Prozeduren haben möchten, dann lesen Sie bitte im

Kapitel 7 nach.

Nur zu einem soll hier noch ein Hinweis gegeben werden, nämlich zu den hier verwendeten Shapes. Unter einem Shape versteht man einen rechteckigen Bildschirmausschnitt, den man an jeder Stelle wieder in den Screen einfügen kann. Änlich einer *Brush* in einem Malprogramm. Bei der Verwendung der Shapes müssen Sie nichts über deren inneren Aufbau wissen, man benötigt nur einen Zugriff auf ihn, damit der Computer weiß, auf welchen Shape sich eine Operation bezieht. Zum gleiche Zweck dienen die Variablen `font`, `BackScreen`, `GameScreen`, es handelt sich auch hier nur auf Zugriffe auf die einzelnen Objekte, und nicht um die Objekte selbst (Für die Experten, es handelt sich hier um Zeiger).

```
MODULE Puzzle;

(* $V- $R- $S- $N- *) | Compilerswitches

FROM GfxScreen   IMPORT Screen,OpenScreen,Palette,SetPalette,
                        FadeIn,FadeOut;
FROM GfxDraw     IMPORT ClearScreen,SetDrawMode,DrawModes,
                        AreaRectangle,SetAPen;
FROM GfxShape    IMPORT Shape,GetShape,PutShape,FreeShape,Sync;
FROM GfxText     IMPORT Font,OpenFont,SetFont,CharWidth;
FROM GfxPseudo3D IMPORT Write3D,Box3D,Color3D,Circle3D;
FROM GfxInput    IMPORT UserActions,UserAct,WaitUser,MouseClick,
                        CheckUser;
FROM Random      IMPORT RND;

CONST
  PuzzleTop   = 40; | Abstand vom oberen Bildschirmrand zum Puzzle
VAR
  GameScreen,
  BackScreen  : Screen; | Zugriffe auf die beiden Screens
  font        : Font;   | Zugriff auf den Zeichensatz
  chips       : ARRAY [0..15] OF Shape; | Feld mit Zugriffen auf
                                        | einen Shape.
  backfields  : ARRAY [0..3],[0..3] OF Shape; | Hier werden die einzelnen
                                              | Teile des Hintergrundes
                                              | abgelegt.
  Field       : ARRAY [0..3],
                      [0..3] OF SHORTINT; | In dieser Tabelle wird
                                          | die aktuelle Position der
                                          | einzelnen Puzzelteile
                                          | eingetragen
  FreeX,FreeY : INTEGER;| Position des freien Feldes

PROCEDURE CreateChips;  | Erzeugt die einzelnen Puzzleteile
CONST ChipChars = ARRAY OF CHAR:
                   ("0","1","2","3", | Beschriftung der einzelnen Zeichen.
                    "4","5","6","7", | Die Zeichen werden in einem Array
                    "8","9","A","B", | abgelegt, damit man das Erzeugen
```



```
                    "C","D","E","F");| der Teile in einer Schleife
                                     | abhandeln kann

  ChipColor = Color3D:(topLeft=1,     | Farben Puzzelteils,
                       bottomRight=3, | sie sind so gewählt,
                       normal=2);     | daß die Teile erhaben wirken.
  CharColor = Color3D:(topLeft=3,     | Die Farbe der Schrift
                       bottomRight=1, | dagegen bewirkt, daß
                       normal=0);     | die Schrift wie ausgestanzt
                                      | wirkt.
VAR
  i  : INTEGER; | Schleifenzähler
  c  : CHAR;

BEGIN
  FOR i:=0 TO 15 DO | 16 Teile sollen gesetzt werden.
    Box3D(BackScreen,ChipColor,0,0,31,31); | Hier wird erstmal ein
                                           | Rechteck auf die Hilfs
                                           | screen gemalt.
    c:=ChipChars[i];
    Write3D(BackScreen,CharColor,          | Schreibt ein Zeichen auf das
                                           | Rechteck
            16-CharWidth(font,c) DIV 2,24, | Position, an die geschrieben
                                           | wird. Dabei wird mittels
                                           | "Charwidth" dafür gesorgt,
                                           | daß die Zeichen zentriert
                                          | auf den Puzzelteilen erschei-
                                          | nen, auch wenn ein Propor-
                                          | tionaler Zeichensatz verwendet
                                          | wird.
                                c]); | Zeichen, das geschrieben werden soll
    GetShape(BackScreen,chips[i],0,0,31,31); | Kopiert den Bereich, in
                                             | den gezeichnet worden ist,
                                             | und merkt sich einen
                                             | Verweis darauf in dem
  END;                                       | Array "chips"
END CreateChips;
```

```
PROCEDURE CreateBackground;
CONST
  BackColor = Color3D:(topLeft=4,bottomRight=6,normal=5);
VAR
  x,y : INTEGER;
BEGIN
  SetAPen(BackScreen,5); | Setze Hintergrundfarbe
  AreaRectangle(BackScreen,0,0,127,127); | Zeichne ein ausgefülltes
                                         | Rechteck in dieser Farbe
                                         | als Hintergrund
  | Die folgenden Schleifen malen nun aud den Hintergrund
  | 4 Kreise die vertieft liegend wirken
  FOR x:=14 TO 123 BY 20 DO
    FOR y:=14 TO 123 BY 20 DO
      Circle3D(BackScreen,BackColor,x,y,7);
    END;
  END;
  | Die Folgenden  Schleifen sichern jetzt jeweils ein Viereck
  | von der Größe eines Puzzleteils, diese werden gebraucht,
  | um später beim Verschieben der Puzzleteile den Hintergrund
  | wieder herstellen zu können.
  FOR x:=0 TO 3 DO
    FOR y:=0 TO 3 DO
      GetShape(BackScreen,backfields[x,y],x*32,y*32,x*32+31,y*32+31);
    END;
  END;
END CreateBackground;

PROCEDURE DrawPicture; | Malt den Rahmen des Spielfeldes,
                       | indem ein vertieft wirkendes Rechteck
                       | in ein erhabenes gezeichnet wird.
CONST Frame1 = Color3D:(topLeft=7,bottomRight=9,normal=8);
      Frame2 = Color3D:(topLeft=9,bottomRight=7,normal=8);
VAR x,y : INTEGER;
```



```
BEGIN
  Box3D(GameScreen,Frame1,0,PuzzleTop,159,PuzzleTop+159);
  Box3D(GameScreen,Frame2,14,PuzzleTop+14,145,PuzzleTop+145);
END DrawPicture;

PROCEDURE InitField;| Gibt jedem Feld eine Steinnummer
VAR i : INTEGER;
BEGIN
  FOR i:=0 TO 14 DO
    Field[i MOD 4,i DIV 4]:=i;
  END;
  FreeX:=3;FreeY:=3;| Setzt das anfangs leere Feld.
END InitField;

PROCEDURE DrawField; | Zeichnet die Puzzleteile auf den
VAR x,y : INTEGER;   | Spielscreen
BEGIN
  SetDrawMode(GameScreen,drawAPen); | Setzt normalen Zeichenmodus
  FOR x:=0 TO 3 DO | Jede Spalte
    FOR y:=0 TO 3 DO | Jede Zeile
      PutShape(GameScreen,backfields[x,y],  | Setzt die Hintergrund
               x*32+16,y*32+(PuzzleTop+16));| Kacheln
      IF (x#FreeX) OR (y#FreeY) THEN | Wenn die Position ungleich
                                     | der Leeren ist,
        PutShape(GameScreen,chips[Field[x,y]],| Zeichne das
                 x*32+16,y*32+(PuzzleTop+16));| Puzzleteil, das
                                              | sich gerade an dieser
                                              | Position befindet
      END;
    END;
  END;
END DrawField;

CONST
  | Dieses Array wird dazu benötigt, um eine Realistischere
  | Bewegung zu erzielen.
  Para    = ARRAY OF SHORTINT:(1,3,6,10,14,19,24,29,32,
                               29,27,28,30,32,31,32);
```

```
| Bewegt den x. Stein in der Zeile, in der sich der freie
| Platz befindet nach rechts
PROCEDURE MoveRight(x : INTEGER);
VAR i,j : INTEGER;
    s   : Shape;
BEGIN
  FOR i:=0 TO Para'MAX DO | Innerhalb dieser Schleife
                          | werden die Puzzleteile stückchenweise
                          | verschoben.
    FOR j:=x TO FreeX DO | Baut auf der Hilfsscreen den Hintergrund
                         | des Bereiches auf, der vom Verschieben
                         | betroffen ist.
      PutShape(BackScreen,backfields[j,FreeY],j*32,0);
    END;
    FOR j:=x TO FreeX-1 DO | Setzt die Chips, die verschoben werden
                           | umd den Betrag "Para[i]" verschoben
                           | auf das Hintergrundmuster auf dem
                           | Hilfsscreen
      PutShape(BackScreen,chips[Field[j,FreeY]],j*32+Para[i],0);
    END;
    GetShape(BackScreen,s,
             x*32,0,FreeX*32+31,31);| Schneidet den neu gezeichneten
                                    | Bereich aus der Hilfsscreen aus
    Sync(GameScreen); | Sorgt dafür, daß der Bildschirm dabei nicht
                      | flackert.
    PutShape(GameScreen,s,
             x*32+16,FreeY*32+(PuzzleTop+16)); | Und Kopiert ihn
                                               | an die richtige
                                               | Stelle auf den Spielscreen
    FreeShape(s); | Gibt den Shape wieder frei.
  END;
  FOR j:=FreeX TO x+1 BY -1 DO  | Verschiebt die Puzzleteile
                                | auch innerhalb des Arrays, in dem
                                | deren Position abgelegt ist.
    Field[j,FreeY]:=Field[j-1,FreeY]
  END;
  FreeX:=x; | Setze neue freie x-Position
```



```
END MoveRight;

| Bewegt den x. Stein in der Zeile, in der sich der freie
| Platz befindet nach links.
| Beschreibung, siehe MoveRight
PROCEDURE MoveLeft(x : INTEGER);
VAR i,j : INTEGER;
    s   : Shape;

BEGIN
  FOR i:=0 TO Para'MAX DO
    FOR j:=FreeX TO x DO
      PutShape(BackScreen,backfields[j,FreeY],j*32,0);
    END;
    FOR j:=FreeX+1 TO x DO
      PutShape(BackScreen,chips[Field[j,FreeY]],j*32-Para[i],0);
    END;
    GetShape(BackScreen,s,FreeX*32,0,x*32+31,31);
    Sync(GameScreen);
    PutShape(GameScreen,s,FreeX*32+16,FreeY*32+(PuzzleTop+16));
    FreeShape(s);
  END;
  FOR j:=FreeX TO x-1 DO
    Field[j,FreeY]:=Field[j+1,FreeY]
  END;
  FreeX:=x;
END MoveLeft;

| Bewegt den y. Stein in der Spalte, in der sich der freie
| Platz befindet nach unten
| Beschreibung, siehe MoveRight
PROCEDURE MoveDown(y : INTEGER);
VAR i,j : INTEGER;
    s   : Shape;
BEGIN
  FOR i:=0 TO Para'MAX DO
    FOR j:=y TO FreeY DO
      PutShape(BackScreen,backfields[FreeX,j],0,j*32);
```

```
    END;
    FOR j:=y TO FreeY-1 DO
      PutShape(BackScreen,chips[Field[FreeX,j]],0,j*32+Para[i]);
    END;
    GetShape(BackScreen,s,0,y*32,31,FreeY*32+31);
    Sync(GameScreen);
    PutShape(GameScreen,s,FreeX*32+16,y*32+(PuzzleTop+16));
    FreeShape(s);
  END;
  FOR j:=FreeY TO y+1 BY -1 DO
    Field[FreeX,j]:=Field[FreeX,j-1]
  END;
  FreeY:=y;
END MoveDown;

| Bewegt den y. Stein in der Spalte, in der sich der freie
| Platz befindet nach oben
| Beschreibung, siehe MoveRight
PROCEDURE MoveUp(y : INTEGER);
VAR i,j : INTEGER;
    s   : Shape;
BEGIN
  FOR i:=0 TO Para'MAX DO
    FOR j:=FreeY TO y DO
      PutShape(BackScreen,backfields[FreeX,j],0,j*32);
    END;
    FOR j:=FreeY+1 TO y DO
      PutShape(BackScreen,chips[Field[FreeX,j]],0,j*32-Para[i]);
    END;
    GetShape(BackScreen,s,0,FreeY*32,31,y*32+31);
    Sync(GameScreen);
    PutShape(GameScreen,s,FreeX*32+16,FreeY*32+(PuzzleTop+16));
    FreeShape(s);
  END;
  FOR j:=FreeY TO y-1 DO
    Field[FreeX,j]:=Field[FreeX,j+1]
  END;
  FreeY:=y;
```



```
END MoveUp;

PROCEDURE MixUpField | Mischt das Spiel zufällig
VAR i : INTEGER;
BEGIN
  FOR i:=1 TO 64 DO | es werden 65 Mischversuche unternommen
    IF KEY RND(4)  | Ziehe eine Zufallszahl zwischen 0 und 3
      OF 0 AND_IF FreeX>0 THEN | Wenn die freie Stelle nicht am linken
                               | Rand liegt,
          MoveRight(RND(FreeX))| bewege einen Stein zwischen dem linken
                               | Rand und der freien Stelle nach rechts.
                               | Eventuell dazwischenliegende Steine
                               | werden mitverschoben.
        END
      OF 1 AND_IF FreeX<3 THEN | Wenn die freie Stelle nicht am rechten
                               | Rand liegt,
          MoveLeft(3-RND(3-FreeX))| bewege einen Stein zwischen
                                  | rechten Rand und der freien
                                  | Stelle nach links.
        END
      OF 2 AND_IF FreeY>0 THEN | Entsprechendes für oben und unten.
          MoveDown(RND(FreeY))
        END
      OF 3 AND_IF FreeY<3 THEN
          MoveUp(3-RND(3-FreeY))
        END
    END;
  END;
END MixUpField;
```

```
VAR
  x,y : INTEGER; | Hierin werden später die Koordinaten des letzten
                 | Mausklick abgelegt.

BEGIN
  OpenScreen(BackScreen,4,FALSE,FALSE); | Hier wird eine Screen mit
                                        | 16 Farben mit den Auflösungen:
                                        | 320x256 geöffnet. Er wird noch
                                        | zum Ablegen der Puzzleteile
  SetDrawMode(BackScreen,drawAPen);     | benötigt Zeichenmodus auf
                                        | normales Malen einstellen.
  OpenScreen(GameScreen,4,FALSE,FALSE); | Hier wird der Spielscreen
                                        | geöffnet

  | Die nächste Anweisung setzt alle Farben des Screens auf Schwarz.
  SetPalette(GameScreen,Palette:((0,0,0),(0,0,0),(0,0,0),(0,0,0),
                                 (0,0,0),(0,0,0),(0,0,0),(0,0,0),
                                 (0,0,0),(0,0,0),(0,0,0),(0,0,0),
                                 (0,0,0),(0,0,0),(0,0,0),(0,0,0)));
  OpenFont(font,"diamond",20); | Hier öffnen wir einen eigenen
                               | Zeichensatz.
  SetFont(BackScreen,font);    | Setze den Font für den Backscreen.
  CreateChips;                 | Erzeuge Puzzleteile.
  CreateBackground;            | Erzeuge Puzzlehintergrund.
  DrawPicture;                 | Zeichne Puzzlerahmen.
  InitField;DrawField;         | Initialisiere Puzzle und zeichne es.
  FadeIn(GameScreen,Palette:((0,0,0),
                             (0,12,15),(0,8,10),(0,4,5),
                             (0, 0,15),(0,0,10),(0,0,5),
                             (15,0,0),(10,0,0),(5,0,0)),16);
  MixUpField; | Mische Puzzle
  REPEAT | Diese Schleife wird solange ausgeführt,
         | bis eine Taste gedrückt wird.
    WaitUser(GameScreen,UserAct:{userKey,userMouse});| Wartet auf
                                                     | eine Eingabe
    IF CheckUser(GameScreen,UserAct:{userMouse}) THEN
      | Falls die Eingabe ein Mausklick war,
```



```
      | wird die Position des Klicks abgefragt.
      MouseClick(GameScreen,x,y); | Dies geschieht hier.
      x:=(x-16) DIV 32;           | Welche Spalte im Puzzle ?
      y:=(y-(PuzzleTop+16)) DIV 32;| Welche Reihe ?
      IF (x OF 0..3) AND (y OF 0..3) THEN | Check, ob der Klick
                                          | innerhalb des Puzzles
                                          | stattfand.
        IF x=FreeX | Wenn der Stein in der angeklickten Spalte bewegt
                   | werden kann,
          AND_IF y>FreeY THEN MoveUp(y)| Und y größer als die nächste
                                       | freie Position ist, bewege
                                       | den Stein nach oben (bedenken
                                       | Sie dabei, daß Bildschirmkoor-
                                       | dinaten nach oben hin abnehmen.
          OR_IF  y<FreeY THEN MoveDown(y) | Im anderen Fall wird nach
                                          | unten bewegt
          END
        OR_IF y=FreeY | Falls der Stein nur in der gewählten Reihe
                      | bewegt werden kann,
          AND_IF x>FreeX THEN MoveLeft(x) | und sich der freie Platz
                                          | links von der Klickposition
                                          | befindet, bewege den Stein
                                          | nach links.
          OR_IF  x<FreeX THEN MoveRight(x)| Sonst nach rechts.
          END
        END;
      END;
    END;
  UNTIL CheckUser(GameScreen,UserAct:{userKey});
  FadeOut(GameScreen,16); | Ausblenden des Screens
CLOSE
  | Normalerweise sollte man hier den Screen und den Font wieder
  | Schließen bzw. freigeben, da die Gfx-Module dies jedoch von selbst
  | am Programmende machen, können wir darauf verzichten
END Puzzle.
```

Vielleicht erweitern Sie ja das Programm dahin gehend, daß das Puzzle auf Tastendruck automatisch gelöst wird.

Hier noch zwei weitere Beispiele:

```
MODULE Groesster_gemeinsamer_Teiler;
(* euklidischer Algorithmus *)
FROM InOut IMPORT ClearWindow, WriteString, WriteLn,
                   ReadInt, WriteInt;
VAR a,b,r :  INTEGER;
BEGIN
    ClearWindow;
    WriteString("Geben Sie 2 natürliche Zahlen ein:");
    WriteInt;
    WriteString("a:  ");
    ReadInt(a);
    WriteLn;
    WriteString("b:  ");
    ReadInt(b);
    WriteLn;
    REPEAT
        r:=a MOD b;
        a:=b;
        b:=r
    UNTIL r=0;
    WriteString("Der größte gemeinsame Teiler ist ");
    WriteInt(a);
END Groesster_gemeinsamer_Teiler.
```



```
MODULE Mittelwert;
FROM InOut IMPORT ClearWindow, WriteString, ReadReal,
 WriteReal;
VAR a,b,mittelwert:REAL;
BEGIN
  ClearWindow;
  WriteString("Bitte geben Sie eine Zahl a ein:  ");
  ReadReal(a);
  WriteString("Bitte geben Sie eine Zahl b ein:  ");
  ReadReal(b);
  mittelwert:=(a+b)/2;
  WriteString("Der Mittelwert aus ");
  WriteReal(a,5,2);
  WriteString(" und "); WriteReal(b,5,2);
  WriteString(" lautet:  ");
 WriteReal(mittelwert,6,2);
END Mittelwert.
```

Bildschirmbild dazu:

Bitte geben Sie eine Zahl a ein: 2.5
Bitte geben Sie eine Zahl b ein: 3.7
Der Mittelwert aus 2.50 und 3.70 lautet: 3.10

Kapitel 4

Fortgeschrittenes Programmieren in Cluster

Dieses Kapitel wendet sich an den schon etwas fortgeschritteneren Programmierer, der bereits einfache Programme schreiben kann. Hier werden erweiterte Datenstrukturen, wichtige Algorithmen, Programmierkonventionen und -techniken behandelt. Die hier vorgestellten Dinge gehören ins Nähkästchen eines jeden Programmierers und sollten auch bei eigenen Projekten berücksichtigt werden.

4.1 Der `POINTER`

Haben Sie sich nicht schon oft darüber geärgert, daß Sie bei der Benutzung von Feldern (ARRAYS) schon bei Programmbeginn wissen und angeben mußten, wie viele Elemente das Feld hat?

Die Datenstrukturen, die Sie bisher kennengelernt haben, sind allesamt statisch, d. h. im Deklarationsteil des Programms sind sie der Größe nach festgelegt.

In diesem Abschnitt sollen Ihnen sogenannte Zeigertypen nähergebracht werden. Später werden Sie mehr über dynamische Datenstrukturen erfahren. Dazu gehören auch Listen. Hierbei handelt es sich um eine Art eindimensionales Feld, allerdings ist die Länge einer Liste nicht beschränkt. Kommt ein Element zur Liste hinzu, so erhöht sich die Anzahl der Elemente. Die Elemente einer Liste sind miteinander durch sogenannte Zeiger verkettet. Sie werden mehrere Arten von Verkettung kennenlernen.

Es gibt schon merkwürdige Typen in **Cluster** ...

```
TYPE Zeiger = POINTER TO Objekt;
```

Eine Variable dieses Typs ist nicht selber von Typ Objekt, denn dann müßte es ja `TYPE Zeiger = Objekt` heißen, sondern es ist eine Variable, die auf das Objekt zeigt.

Deklarieren wir uns zwei Zeigervariablen:


```
TYPE Zeiger = POINTER TO CHAR;
  VAR
    a, b: Zeiger;
|VAR a, b: POINTER TO CHAR;
| wäre auch möglich gewesen!
```

Beachten Sie, daß die Variablen `a` und `b` *nicht* vom Typ `CHAR` sind!

Unsere Zeiger zeigen nun so in der Weltgeschichte herum. Dies ist ein ziemlich unbefriedigender Zustand, weil undefiniert. Aus diesem Grunde existiert eine Konstante mit der Bezeichnung

```
NIL (engl. Nichts),
```

die jedem Zeiger beliebigen Typs zugeordnet werden kann. Die Konstante `NIL` wird auch häufig *Erdung* genannt. Also kann man mit

```
a := NIL;
b := NIL;
```

beiden Zeigern einen definierten Zustand zuordnen. Sie sind geerdet.

Der Sinn dieser „Erdung" liegt darin, daß wir später beispielsweise das Ende einer Liste durch eine solche Erdung kennzeichnen können oder entscheiden können, ob ein Zeiger auf ein Element zeigt oder nicht.

Um unsere Zeiger auf Objekte vom Typ `CHAR` zeigen zu lassen benötigen wir die Prozedur

```
New (zeiger);
```

Mit dieser Prozedur wird eine vorerst leere Speicherstelle (Variable) erzeugt, auf die der Zeiger zeigt. Der Datentyp dieser Variablen ist durch den angegebenen Zeigerdatentyp (hier `CHAR`) festgelegt.

```
New (a);
New (b);
```

erzeugt Variablen mit den Namen `a^` und `b^` vom Typ `CHAR`. Erinnern Sie sich, der Zeigertyp ist `POINTER TO CHAR`. Betrachten Sie nun Abbildung 4.1.

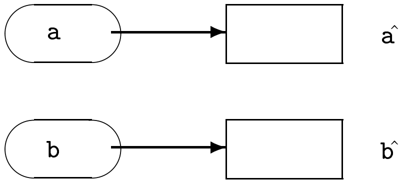


Abbildung 4.1: Speicher nach dem Erzeugen der Zeiger

Die Kästchen (Variablen) mit den Namen `a^` und `b^` sind noch leer. Nun wollen wir ihnen Speicherinhalte zuweisen:

```
a^ := "x";
b^ := "y";
```

Wenn Sie Abbildung 4.2 betrachten, werden Sie erkennen, daß die Zeiger `a` und `b` nun auf die Variablen `a^` und `b^` zeigen, mit den Inhalten `"x"` und `"y"`.

Wichtig ist, daß Sie folgende Zuweisungsarten zwischen diesen Zeigern jetzt peinlich unterscheiden müssen:

```
a := b;
```

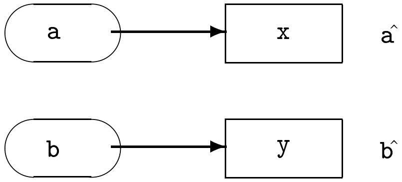



Abbildung 4.2: Zuweisung von Speicherinhalten

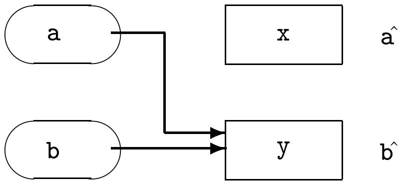


Abbildung 4.3: Zeiger abgeknipst

ist ein korrekter Befehl, wie sieht aber unser Diagramm jetzt aus? (Abbildung 4.3)

Mit dieser Zuweisung wurde der Zeiger `a` so verbogen, daß er auf dasselbe Objekt zeigt, wie der Zeiger `b`.

Achtung: Auf das Objekt `a^` zeigt nun kein Zeiger mehr. Dieses Objekt ist verloren! Es existiert praktisch keine Möglichkeit mehr, auf dieses Objekt zuzugreifen.

Werden Zeigervariablen einander zugewiesen, so müssen die Objekte, auf die die Zeiger zeigen, vom selben Typ sein.

Hätten Sie die Zuweisungsart anders gewählt, nämlich:

```
a^ := b^;
```

so ergäbe sich folgendes Bild (Abbildung 4.4):

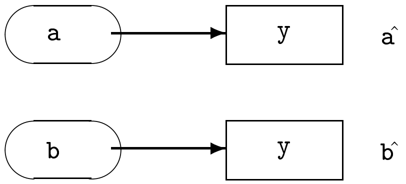


Abbildung 4.4: Andere Zuweisungsart

Hier haben wir nur den Inhalt der Speicherstelle geändert, auf die `a` zeigt. Im Speicher des Rechners gibt es nun den Buchstaben „`y`“ zweimal.

Sie sehen also, beim Arbeiten mit Zeigern ist Vorsicht angebracht. Denken Sie beim Programmieren lieber etwas länger nach, dies wird die Fehlersuchzeiten erheblich reduzieren.

Nun zur Wiederholung und Festigung des eben gelernten ein etwas umfangreicheres Beispiel. Außerdem werden Sie erfahren, wie man die mit `New` erzeugten Speicherstücke wieder freigeben kann.


```
TYPE
  ZahlZeiger = POINTER TO INTEGER;
VAR
  aPtr, bPtr : ZahlZeiger;
  zahl       : INTEGER;

BEGIN
  aPtr := zahl'PTR;
  bPtr := NIL;
  zahl := 12;
     | "bPtr" zeigt nach *nirgendwo*
     | "aPtr" zeigt jetzt auf den Speicherinhalt
     | von "zahl", "zahl" hat den Wert 12 angenommen
  New(bPtr);
     | fuer "bPtr" wurde ein Stueck Speicher erzeugt
     | "bPtr" zeigt auf dieses Speicherstueck
  bPtr^ := 4 * zahl;
     | in diesem Speicherstueck steht jetzt 48
  aPtr := bPtr;
     | auch "aPtr" zeigt jetzt auf dasselbe Speicher
     | stueck wie "bPtr", also "aPtr"="bPtr"
  zahl := aPtr^ - zahl;
     | "zahl" hat den Wert 36 angenommen
  Dispose(aPtr);
  bPtr := NIL;
     | Das mit New() erzeugte Speicherstueck ist
     | dem System zurueckgegeben worden,
     | es gilt: "bPtr=NIL", "aPtr=NIL", "zahl=36"
  ...
END;
```

Die Prozeduren `New()` und `Dispose()` erzeugen bzw. vernichten ein Speicherstück von der Größe desjenigen Objekts, auf das sie zeigen. Diese zwei Prozeduren werden vom Modul `Ressources` bereitgestellt. Man beachte, hier wurden vier verschiedene Methoden vorgestellt, wie man

Zeigervariablen „initialisieren“ kann, und zwar durch:

- `bPtr := NIL`, die Variable `bPtr` zeigt nirgendwohin, `NIL` ist das Symbol hierfür,
- `aPtr := zahl'PTR`, die Zeigervariable zeigt auf ein Speicherstück, das vom geforderten Typ ist – hier `INTEGER`,
- `aPtr := bPtr`, ein gültiger Zeigerinhalt wird dupliziert,
- `New(bPtr)`, ein passendes Speicherstück wird vom Laufzeitsystem angefordert, die Variable zeigt anschließend darauf.

Beim Hantieren mit Zeigern ist Vorsicht geboten! In unserem Beispiel zeigt die Variable `bPtr` – direkt nach Ausführung der Anweisung `Dispose(aPtr)` – auf einen Speicherbereich, der dem System bereits zurückgegeben worden ist. Wir dürfen keine Annahmen darüber machen, wofür das Laufzeitsystem dieses Stück Speicher als nächstes benutzen wird; falls wir diesen Speicher dennoch modifizieren, obwohl er schon zurückgegeben worden ist, sorgen wir für die tollsten Überraschungseffekte! Deshalb wird auch die Variable `bPtr` auf `NIL` gesetzt. So können wir später ganz einfach feststellen, ob unser Zeiger auf etwas gültiges zeigt oder nicht.

Es sollte auch darauf geachtet werden, daß man diejenigen Speicherstücke, die man nicht mehr braucht, wieder mit `Dispose()` zurückgibt. Denn sonst würde ja unser „Adreßverwaltungs-Programm“ unnütz viel mehr Speicher verbrauchen als nötig, und ein riesiger Vorteil der dynamischen Speicherung unserer Daten wäre dahin. Aber wie man mit dynamischen Datenstrukturen umgeht, das wird im Abschnitt „Dynamische Strukturen“ unter 4.6 ausführlich behandelt.

Es existiert ein Pointertyp, der zu allen Pointertypen und zum Datentyp `LONGINT` kompatibel ist, `ANYPTR`. Dieser Pointer hat kein typisiertes Ziel. In diesem Typ gibt es die Konstante `NIL`, die die nichtvorhandene Adresse bedeutet.

Sie haben eben gelernt, daß man, um an den Speicherinhalt, auf den ein Pointer zeigt, zugreifen zu können, den Zeiger mittels eines

„`^`“ dereferenzieren muß. Zeigt ein Pointer auf eine Recordvariable, und will man auf eine Element des Records zugreifen, muß man nicht `RecPtr^.element` schreiben, sondern es genügt `RecPtr.element`, da der „`.`“ wie bei allen modernen Hochsprachen die Dereferenzierung impliziert.

4.2 Offene Typen

Cluster stellt neben den üblichen Strukturen auch noch offene Typen zur Verfügung. Diesen fehlt bei der Definition eine Eigenschaft – eine feste Länge. Es gibt drei Arten dieser Typen, offene Strings, offene Arrays und offene Records[2]. Da diese immer für eine ganze Klasse von Typen stehen, heißen sie auch Klassen. Es ist nicht möglich, Variablen dieser Typen zu erzeugen, da ihre Länge nicht bekannt ist. Sie können jedoch als VAR-Parameter übergeben oder als Pointerziele existent werden. Da diesen Typen eine wichtige Information in ihrer Typbeschreibung fehlt, muß diese irgendwo im Zeiger darauf vorhanden sein. Zu diesem Zweck haben offene Typen eine eigene Art Zeiger, den `CLASSPTR`. Dieser unterscheidet sich vom `POINTER` nur dadurch, daß er zu der Adresse eines Objekts noch Daten darüber enthält.

4.2.1 Offene Strings

Diese sind ihnen als Übergabeparameter schon begegnet. Das fehlende Datum bei offenen Strings ist die maximale Länge. Diese wird entweder bei der Übergabe an eine Prozedur mit übergeben, oder wird im `CLASSPTR` auf den String gespeichert.

```
PROCEDURE FreeChars(VAR s : STRING):INTEGER;
BEGIN
  RETURN s'RANGE-s.len
END FreeChars;
```

Diese Prozedur liefert die Zahl der noch freien Zeichen in einem String. Soll ein String einer Länge erzeugt werden, die erst während der Laufzeit bekannt ist, so muß dies über einen Zeiger geschehen.

[2] Offnene Records werden in dieser Compiler-Version zwar noch unterstützt, sollten aber nicht mehr verwendet werden. Mittlerweile ist es nämlich möglich, aus normalen Records erweiterte Records zu erzeugen. Dieses Feature war anfangs nur mit offenen Records möglich.


```
VAR s : CLASSPTR TO STRING;

s^'RANGE:=MaxLength;
New(s);
```

Durch diese Anweisung wird ein String der Länge `MaxLength` erzeugt, auf den über `s^` zugegriffen werden kann.

4.2.2 Offene Arrays

Sie sind stark mit den Strings verwandt, da auch bei ihnen die Größenangabe fehlt. Offene Arrays beginnen immer mit null, und enden bei `'MAX`.

```
TYPE
  Koord    = RECORD x,y : INTEGER END;
  Polygon  = ARRAY OF Koord;
```

Auch sie können als Übergabeparameter verwendet werden, oder durch Pointer erzeugt werden.

```
PROCEDURE DrawPoly(VAR p : Polygon);
VAR i : INTEGER
BEGIN
  Move(p[0]);
  FOR i:=1 TO p'MAX DO
    DrawTo(p[i])
  END;
  DrawTo(p[0])
END DrawPoly;
```

Es ist wie bei Strings ebenfalls möglich, Ausprägungen dieses Types zu erzeugen, indem eine Größe angegeben wird.

```
TYPE
  Triangle = Polygon(3)
```

Wird ein offenes Array als Typ einer Konstante benutzt, wird die Länge aus der Anzahl der Werte genommen.

```
CONST
  Quadrat  = Polygon:((x=-10,y=-10),
                      (x= 10,y=-10),
                      (x= 10,y= 10),
                      (x=-10,y= 10));
```

Zur Vereinfachung ist es auch erlaubt, `ARRAY OF` bei Konstanten zu benutzen.

```
CONST
  Sequenz = ARRAY OF INTEGER:(4,5,6,7,8);
```

Bei offenen Arrays als Pointerziel wird wie bei Strings verfahren:


```
VAR p : CLASSPTR TO Polygon;
...
p^'RANGE:=...;
New(p);
```

Zuweisen lassen sich Arrays aber nur, wenn sie die gleiche Länge haben. Natürlich ist es aber möglich, dem Zeiger auf ein offenes Array den Zeiger auf ein normales Array diesen Typs zu übergeben.

```
VAR
  theTriangle : Triangle;
...
  p:=theTriangle'PTR;
```

Dabei wird auch automatisch die Länge mit übernommen. Soll nur die Adresse, aber nicht die Länge geändert werden, muß ein Umweg beschritten werden:

```
  ANYPTR(p):=theTriangle'PTR;
```

4.2.3 ALLOC_RESULT

Auch der Ergebnistyp einer Funktion kann ein offnenes Array bzw. ein offener String sein. Jedoch dann muß in der Funktion selbst die Größe des Ergebnisses bestimmt werden. Dies wird durch den Aufruf von `ALLOC_RESULT` geregelt. Die Funktion `ALLOC_RESULT` erzeugt bei unklarer Größe des Rückgabetyps ein entsprechendes Objekt im Heapspeicher. Wie bei komplexen Ergebnistypen erforderlich, muß auch auf dieses Objekt über die Variable `RESULT` zugegriffen werden. Die Argumente der Funktion `ALLOC_RESULT` enthalten die Anzahl der Elemente des Ergebnis-Arrays bzw. die Anzahl der Zeichen des Ergebnis-Strings und ermittelt daraus selbsttätig die die echte Größe.

```
$$OwnHeap:=TRUE
  PROCEDURE Reverse(REF str : STRING):STRING;
  VAR
    i : INTEGER;
  BEGIN
    ALLOC_RESULT(str.len);
    ASSERT(RESULT'RANGE>str.len,RangeChk);
    FOR i:=0 TO str.len-1 DO
      RESULT.data[i]:=str.data[str.len-i-1];
    END;
    RESULT.len:=str.len;
    RESULT.data[RESULT.len]:=&0;
  END Reverse;

  PROCEDURE Concat(REF str1,str2 : STRING):STRING;
  VAR
    i : INTEGER;
  BEGIN
    ALLOC_RESULT(str1.len+str2.len);
    ASSERT(RESULT'RANGE>(str1.len+str2.len),RangeChk);
    FOR i:=0 TO str1.len-1 DO
      RESULT.data[i]:=str1.data[i];
    END;
    FOR i:=0 TO str2.len-1 DO
      RESULT.data[str1.len+i]:=str2.data[i];
    END;
    RESULT.len:=str1.len+str2.len;
    RESULT.data[RESULT.len]:=&0;
  END Reverse;
```

Somit können auch Rückgabewerte, die vom Typ her offen gehalten sind, wieder direkt als Eingabewert von anderen Funktionen benutzt werden. Dies ist zum Beispiel bei Ausdrücken wie
`palindrom:=Concat(Reverse(wort),wort);`
der Fall. Ohne Verwendung von `ALLOC_RESULT` wäre die Größe des zu

allokierenden Speichers für das Ergebnis von `Reverse(wort)` unklar. Durch `ALLOC_RESULT` wird die Größe des Ergebnisses festgelegt. Erst dadurch kann eine vernünftige Parameterübergabe an die Funktion `Concat` erfolgen.

4.2.4 Der absolut offene Typ ANYTYPE

Der Typ `ANYTYPE` ist völlig offen, das heißt, von ihm ist nur Adresse und Länge bekannt. `ANYTYPE` kann nur als VAR-Übergabeparameter in einer Prozedur verwendet werden.

```
TYPE
  NodePtr = POINTER TO Node;
  Node    = RECORD
              next : NodePtr;
              size : INTEGER;
            END;
  ShortPtr= POINTER TO SHORTINT;

VAR
  queue : NodePtr;

PROCEDURE PutQueue(VAR data : ANYTYPE);
VAR p     : NodePtr;
    s1,s2 : ShortPtr;
    i     : INTEGER;
BEGIN
  Allocate(p,Node'SIZE+data'SIZE);
  s1:=ShortPtr(LONGINT(p)+Node'SIZE);
  s2:=data'PTR;
  FOR i:=1 TO data'SIZE DO
    s1^:=s2^;INC(s1);INC(s2);
  END;
  p^.size:=data'SIZE;
```

```
    IF queue=NIL THEN
      p^.next:=queue
    ELSE
      p^.next:=queue^.next;
      queue^.next:=p;
    END;
    queue:=p;
  END PutQueue;
```

Mit diesem Typen lassen sich also Prozeduren schreiben, die typunabhängig sind. Dies ist meist bei Lade- und Speicherroutinen der Fall.

4.3 Erweiterte Records

Records lassen sich um weitere Elemente erweitern, dabei bleiben alle vorherigen Felder erhalten. Ein Record, der sich auf einen bereits bestehenden gründet, wird durch `RECORD OF <ident>` eingeleitet.

Ein Beispiel aus Exec:

```
MinNodePtr = POINTER TO MinNode;
MinNode    = RECORD
               succ,pred : MinNodePtr;
             END;
Node       = RECORD OF MinNode
               type : NodeTypes;
               pri  : SHORTINT;
               name : SysStringPtr;
             END;

Message    = RECORD OF Node
               replyPort : MsgPortPtr;
               length    : CARDINAL;
             END;
```



```
IORequest  = RECORD OF Message
               device  : DevicePtr;
               unit    : UnitPtr;
               command : CARDINAL;
               flags   : IOFlagSet;
               error   : SHORTCARD;
             END;
```

So lassen sich Hierarchien von Typen aufbauen, die nach unten zuweisungskompatibel sind.

Neben allgemeinen Pointertypen exisitiert noch ein Typ, der speziell für die Probleme, die bei der Vererbung von Records entstehen, eingeführt wurde – der `SAMEPTR`. Ein `SAMEPTR` kann nur als Element eines Records verwendet werden, er stellt einen Zeiger auf den Typen dar, den der aktuelle Record bildet.

Beispiel:

```
TYPE
  Node    = RECORD
              prev,
              next  : SAMEPTR;
            END;
  Node2   = RECORD OF Node
              data  : INTEGER
            END;
```

Die Elemente `prev` und `next` haben in Records des Typs `Node` den Typ `POINTER TO Node`, in Records des Typs `Node2` aber den Typen `POINTER TO Node2`. Der Record bleibt trotzdem nach unten kompatibel, da ja ein `POINTER TO Node2` auch ein Nachfahre von `POINTER TO\index{POINTER`
darstellt. Der Vorteil ist aber, daß Ausdrücke der Form:
`Node2^.next^.data`
möglich sind, was bei der Verwendung des Typs `POINTER TO Node` anstatt eines `SAMPETR`s nicht möglich wäre.

4.4 Generische Module

Viele Datenstrukturen und dazugehörige Algorithmen werden häufig in verschiedenen Kontexten und in Verbindung mit anderen Datenstrukturen benötigt (z. B. Listen, AVLbäume, Stacks etc.). In einer streng und vor allem statisch getypten Sprache tritt das Problem auf, daß diese Strukturen jedesmal neu definiert und implementiert werden müssen. Diesem Problem wird durch *Generizität* abgeholfen.

Ein *generisches Modul* besitzt einen generischen Parameter. Dieser Parameter muß ein Zeigertyp sein. Über diesen Parameter kann das Modul bereits Annahmen machen.

Beispiel:

```
DEFINITION MODULE BiList(NodePtr : POINTER TO Node);

  TYPE
    Node     = RECORD
                 pred,
                 succ   : NodePtr
               END;
    List     = RECORD
                 first,
                 last   : NodePtr
               END;
    ParseProc = PROCEDURE(n : NodePtr);

  PROCEDURE Init(VAR l : List);

  PROCEDURE InsertTop(VAR l : List;n : NodePtr);

  PROCEDURE Parse(VAR l : List;parse : ParseProc);

END BiList;
```



```
IMPLEMENTATION MODULE BiList;

  PROCEDURE Init(VAR l : List);
  BEGIN
    l.first:=NIL;
    l.last :=NIL
  END Init;

  PROCEDURE InsertTop(VAR l : List;n : NodePtr);
  BEGIN
    n.pred:=NIL;
    n.succ:=l.first;
    IF l.first=NIL THEN
      l.last:=n
    ELSE
      l.first.pred:=n
    END;
    l.first:=n
  END InsertTop

  PROCEDURE Parse(VAR l : List;parse : ParseProc);
  VAR n : NodePtr;
  BEGIN
    n:=l.first;
    WHILE n#NIL DO
      parse(n);n:=n.next
    END;
  END Parse;

END BiList.
```

Das Implementationsmodul kann über die bekannten Elemente des Knotentyps (`pred`, `succ`) verfügen, da durch die Definition des generischen Parameters sichergestellt wird, daß nur ein Zeiger auf einen Nachfolger von Node in Frage kommt.

Ein generisches Modul muß, bevor es durch ein anderes Modul verwendet werden kann, durch einen aktuellen Parameter ausgeprägt werden. Dieser Typ muß zum formalen Parametertyp passen.

```
MODULE ZweiListen;

  FROM Resources IMPORT New, Dispose;
  ...

  TYPE
    NamePtr   = POINTER TO NameNode;
    StadtPtr  = POINTER TO StadtNode;

  DEFINITION MODULE NameList  = BiList(NamePtr);
  DEFINITION MODULE StadtList = BiList(StadtPtr);

  TYPE
    NameNode  = RECORD OF NameList.Node
                  nachname,
                  vorname : STRING(100);
                  alter   : INTEGER;
                END;
    StadtNode = RECORD OF StadtList.Node
                  city   : STRING(100);
                  plz    : [1000..9999];
                END;
```

Das Modul `NameList` verwaltet nun eine Liste derartiger `NameNode`s, das Modul `StadtList` eine Liste von `StadtNode`s. Die Typsicherheit ist voll gegeben, da keine anderen Knotentypen zugelassen sind.

Die Prozeduren der Module `NameList` und `StadtList` können wie üblich importiert und benutzt werden. Werden jedoch mehrere Ausprägungen desselben generischen Moduls verwendet, treten Namenskonflikte auf. Daher importiert man nicht die einzelnen Prozeduren aus einem solchen

Modul, sondern benutzt Sie wie Methoden. D. h. da normalerweise alle Prozeduren eines Generischen Moduls als ersten Parameter einen Zeiger auf eine Struktur, oder einen Record haben, in unserem Beispiel immer die Liste auf die sich die Operation bezieht, kann man die Prozedur über den ersten Parameter qualifizieren:

```
VAR
   myNames  : NameList.List;  | Da NameList im selben Modul,
                              | ausgeprägt wurde, muß das Modul
                              | nicht mehr importiert werden.
   name     : NameNode;
   myStadt  : StadtList.List;
   stadt    : StadtNode;
  ...

BEGIN
  myNames.Init; | an Stelle von NameList.Init(myNames);
  New(name);
  myName.InsertTop(name);

  myStadt.Init;
  New(stadt);
  WITH stadt DO
    city := "Karlsruhe";
    plz  := 7500
  END;
  myStadt.InsertTop(stadt);
```

Für alle Ausprägungen eines generischen Moduls wird in einem Programm derselbe Code benutzt, es wird also kein Code vervielfältigt. Aus diesem Grund können auch nur Zeiger als generische Parameter dienen.

Werden für eine Implementation einer Datenstruktur gewisse Fähigkeiten des generischen Parameters (wie eine Ordnungsrelation) benötigt, so kann dies über Prozedurvariablen erreicht werden.

Es ist möglich, lokale Prozeduren als Parameter zu verwenden, es ist allerdings nicht möglich, einer Prozedurvariablen eine lokale Prozedur zuzuweisen.

```
PROCEDURE LasseAltern(VAR l : NameList.List;um : INTEGER);

  PROCEDURE Altere(n : NamePtr);
  BEGIN
    INC(n.alter,um)
  END Altere;

BEGIN
  l.Parse(Altere);
END LasseAltern;
```

Wichtig: Auch wenn bei der Definition des Prozedurtypen `ParseProc` als Übergabeparametertypen nur ein `NodePtr` angegeben ist, muß man nach der Ausprägung an seiner Stelle den Typ verwenden, mit dem das Modul ausgeprägt worden ist. Nur so kann die Prozedur `Altere` auf das Element `alter` zugreifen.

Weitere Beispiele finden Sie auch bei der Beschreibung der generischen Module `AVLTrees` und `Lists` in Kapitel 7.

4.5 Objektorientiertes Programmieren

4.5.1 Grundlegende Struktur

Objekte entsprechen in **Cluster** in vielen Eigenschaften den Records. Dies drückt sich auch in der Definition aus, wie folgendes Beispiel zeigt:

```
TYPE
  Node    = POINTER TO NodeObj;
  NodeObj = OBJECT
              prev,next : Node;
            END;
```

Der deutlichste syntaktische Unterschied liegt im Schlüsselwort `OBJECT` an der Stelle von `RECORD`. Der wichtigste semantische Unterschied besteht darin, daß ein Objekt nicht nur einen statischen Typ, sondern auch einen dynamischen Typ besitzt. Dies wird bei Vererbungen deutlich:

```
TYPE
  BigNode = POINTER TO OBJECT OF Node;
              name : STRING(10);
            END;
VAR n : Node;
    b : BigNode;

...
  n:=b
...
```

Die Variablen `n` und `b` haben zwei verschiedene statische Typen, nämlich `Node` und `BigNode`. Sie haben nach der Zuweisung allerdings denselben dynamischen (d. h. zur Laufzeit) Typen, `BigNode`. Dies kann auch durch den relationalen Operator `IS` geprüft werden:

```
IF n IS BigNode THEN ... END;
```

Wären `BigNode` und `Node` Records gewesen, wäre der dynamische Typ `BigNode` bei der Zuweisung an `n` verloren gegangen. Eine umgekehrte Zuweisung `b:=n` wäre illegal oder zumindest sehr fraglich. Bei Objekten wird zu jedem Objekt ein dynamischer Typ mitgeführt, so daß die Legalität einer Zuweisung `b:=n` überprüft werden kann. Auch eine Typumwandlung wie:

```
WriteString(BigNode(n).name);
```

kann ordnungsgemäß ausgeführt werden, da der Compiler automatisch einen Typtest ins Programm einfügt (dies kann bei fertigen Programmen durch `$$TypeChk:=FALSE` unterbunden werden). Wie sich aber noch zeigen wird, ist eine derartige Zuweisung dank der Verwendung dynamischen Bindens sehr selten.

Der Typ eines Objekts wird auch als Klasse bzw. Klassenzugehörigkeit bezeichnet.

Einen Nachteil haben Objekte gegenüber Records: Objekte können nicht als Variable existieren, nur Zeiger auf Objekte sind erlaubt.

Der statische Typ eines Objektes ist immer durch den Zeiger auf dieses gegeben, der dynamische Typ (also der, den das Objekt nun wirklich hat) ist im Objekt selbst codiert. Der dynamische Typ eines Zeigers auf ein Objekt kann sich während der Programmausführung durch Zuweisungen ändern, der statische Typ ist immer fest und durch die Typdefinition im Programmtext festgelegt. Der Compiler achtet darauf, daß der dynamische Typ eines Objekts immer dem statischen entspricht, oder aber ein Nachfolger davon ist.

4.5.2 Einfacherben

Einfacherben gehorcht derselben Syntax und Semantik wie bereits von Records bekannt:


```
TYPE
  Fahrzeug  = POINTER TO OBJECT
                geschwindigkeit : REAL;
                gewicht         : REAL;
              END;
  Flugzeug  = POINTER TO OBJECT OF Fahrzeug;
                triebwerke      : [0..12];
              END;
  Auto      = POINTER TO OBJECT OF Fahrzeug;
                raeder          : [1..8];
              END;
  VW        = POINTER TO OBJECT OF Auto;
                typ             : (Kaefer,Golf,Polo,
                                   Passat,Jetta)
              END;
```

Der Nachfolger erbt alle Elemente und Fähigkeiten seines Vorgängers und kann diesem neue hinzufügen. Eine Zuweisungskompatibilität an seinen Vorgänger ist uneingeschränkt gegeben, der umgekehrte Fall wird durch eine Typüberprüfung während der Laufzeit gesichert.

Beispiele für Zuweisungen:

```
VAR fz,fz2 : Fahrzeug;
    fl     : Flugzeug;
    au,au2 : Auto;
    vw     : VW;
```

`fz:=fl` ⇒ richtig, hierbei ändert sich der dynamische Typ von `fz` zu `Flugzeug`, ist also ein Nachfolger des statischen Typs `Fahrzeug`.

`fz:=au` ⇒ richtig

`fz:=vw` ⇒ richtig

`au:=vw` ⇒ richtig

`fl:=fz` ⇒ Laufzeittest

`vw:=au` ⇒ Laufzeittest

`fl:=au` ⇒ falsch

`fl:=vw` ⇒ falsch

`vw:=fl` ⇒ falsch

Die Zugehörigkeit zu einer Klasse kann während der Laufzeit durch den relationalen Operator `IS` geprüft werden. Der Operator ist nur für Objekte gestattet, deren statischer Typ in linearer Nachfolge-/Vorgängerbeziehung zur getesteten Klasse stehen.

Beispiele für Tests:

`fz` sei `Fahrzeug`

`fz2` sei `VW`

`fl` sei `Flugzeug`

`au` sei `Auto`

`au2` sei `VW`

`vw` sei `VW`

`fz IS Fahrzeug` ⇒ statisch wahr

`fz2 IS Fahrzeug` ⇒ statisch wahr

`vw IS VW` ⇒ statisch wahr

`vw IS Fahrzeug` ⇒ statisch wahr

`fz2 IS Auto` ⇒ dynamisch wahr

`fz2 IS VW` ⇒ dynamisch wahr

`au2 IS VW` ⇒ dynamisch wahr

`fz2 IS Flugzeug` ⇒ dynamisch falsch

`au IS VW` ⇒ dynamisch falsch

`au2 IS Flugzeug` ⇒ fehlerhafte Anweisung, da der statische Typ von `au2` (also Auto) kein Nachfolger von Flugzeug ist, also sein dynamischer Typ unmöglich ein Nachfolger von Flugzeug sein kann. (Die Funktion könnte auch immer FALSE zurückgeben, doch deutet die Verwendung dieses Operators in diesem Zusammenhang eher auf einen Programmfehler hin, deshalb ist dies verboten).

Der statische Typ eines Objektes kann seinem dynamischen Typ für einen Bereich im Programm angepaßt werden:

```
Auto(fz).raeder:=4;
...
|oder für längere Zeit:
WITH VW(fz) DO
  fz.raeder:=4;
  fz.typ:=Kaefer;
END;
```

Damit ist im allgemeinen eine Typüberprüfung zur Laufzeit verbunden.

4.5.3 Methoden auf Objekte

Auch (oder gerade) auf Objekte können Methoden erklärt werden. Diese müssen bereits bei der Typdeklaration angemeldet werden (Begründung später). Andererseits ist die Angabe der Methoden in einem Definitionsmodul überflüssig, es reicht die Angabe in der Typdefinition.

Beispiel:

```
TYPE
  Lebewesen  = POINTER TO OBJECT
                 alter : INTEGER;
                 METHOD Altere(um : INTEGER := 1);
               END;

METHOD Lebewesen.Altere(um : INTEGER);
BEGIN
  ...
END Altere;
```

Der Methodenaufruf erfolgt analog zu dem für Records:

```
VAR le : Lebewesen;
...
  le.Altere;le.Altere(10);
...
```

Die Elemente (Instanzvariablen) eines Objektes sind in der Implementation der Methode unqualifiziert bekannt:

```
METHOD Lebewesen.Altere(um : INTEGER);
BEGIN
  INC(alter,um);
END Altere;
```


Das Objekt selbst, für das die Methode aufgerufen wurde, ist unter dem Bezeichner `SELF` verfügbar:

```
TYPE
  Lebewesen  = POINTER TO OBJECT
                 alter : INTEGER;
                 METHOD Altere(um : INTEGER := 1);
                 METHOD AltereStark;
               END;

METHOD Lebewesen.AltereStark;
BEGIN
  SELF.Altere(10);
END AltereStark;
```

4.5.3.1 Methoden und Erben (dynamisches Binden)

Methoden von Objekten können beim Erben redefiniert, d. h. durch neue Methoden, die sich dann auf die geerbte Klasse beziehen, ersetzt werden. Die Methode muß hierbei bei der Definition des neuen Typs mit angegeben werden. Der Typ der neuen Methode muß mit dem der bestehenden übereinstimmen.

```
TYPE
  Mensch     = POINTER TO OBJECT OF Lebewesen;
                 haarfarbe : (schwarz,dunkel,grau,weiss);
                 METHOD Altere(um : INTEGER);
               END;
```

```
METHOD Mensch.Altere(um : INTEGER);
BEGIN
  INC(alter,um);
  IF KEY alter
    OF  0..49 THEN haarfarbe:=schwarz END
    OF 50..59 THEN haarfarbe:=dunkel  END
    OF 60..69 THEN haarfarbe:=grau    END
  ELSE
    haarfarbe:=weiss
  END;
END Altere;
```

Welche der Methoden während der Laufzeit ausgeführt werden, bestimmt der dynamische Typ eines Objekts (im Gegensatz zu Methoden bei Records, wo der statische Typ entscheidend ist). So würde also für ein Objekt mit dem dynamischen Typ Mensch und dem statischen Typ Lebewesen immer die Methode mit der Haarfarbe aufgerufen. So kann man Prozeduren schreiben, die z. B. ein Lebewesen übergeben bekommen, und dessen Methoden verwenden. Je nachdem, um was für ein Lebenwesen es sich handelt, wird automatisch die richtige Methode verwendet (bei der Methode Altere und einem Menschen, diemit der Haarfarbe). Die Prozedur muß nicht wissen um welche Art von Lebewesen es sich handelt. Somit kann man auch im Nachhinein beliebige Lebewesen ergänzen, ohne diese Prozedur verändern zu müssen. Würde man hier Records verwenden, die eine Variable enthalten, der angibt um welche Art Lebewesen es sich handelt, müßte innerhalb der Prozedur ein **IF KEY** stehen, daß abhängig von der Art eine andere Prozedur „Altere“ aufruft. Fügt man bei diesem Ansatz ein neues Lebewesen hinzu, muß auch ein neuer **OF**-Zweig in das **IF KEY** eingefügt werden. Beim objektorientierten Ansatz ist dies nicht nötig.

Beispiel:


```
VAR le : Lebewesen;
    me : Mensch;
...
  le:=me;  | Lebewesen ist sicher ein Mensch
  le.Altere;
```

Auch der Aufruf von `AltereStark` würde letztendlich in einem Aufruf der Haarfarbmethode gipfeln, obwohl bei der Definition dieser Methode von der Existenz von Menschen (und der damit verbundenen Haarprobleme) noch nichts bekannt war.

Mit dem Schlüsselwort `SUPER` haben redefinierte Methoden Zugriff auf Methoden der statischen Vorgängerklasse. Dies ist sinnvoll, falls die neue Methode eigentlich eine Erweiterung der bestehenden Methode darstellt.

```
METHOD Mensch.Altere(um : INTEGER);
BEGIN
  SUPER.Altere(um); | Rufe Methode deines
                    | Vorgängers auf
  IF KEY alter
    OF  0..49 THEN haarfarbe:=schwarz END
    OF 50..59 THEN haarfarbe:=dunkel  END
    OF 60..69 THEN haarfarbe:=grau    END
  ELSE
    haarfarbe:=weiss
  END;
END Altere;
```

4.5.3.2 Aufgeschobene (deferred) Methoden und Klassen

Eine aufgeschobene Methode ist eine solche, die zwar vereinbart, aber absichtlich nicht implementiert wird. Ein Aufruf einer derartigen Methode führt zu einem Laufzeitfehler.

Aufgeschobene Methoden können später durch wirklich existierende Methoden redefiniert werden. Dies ergibt erst den Sinn dieser Methoden. Sie dienen dazu, ein Objekt mit Fähigkeiten zu schaffen, die auf anderen Fähigkeiten basieren, die sehr abstrakt gehalten werden können.

```
TYPE
  Stream  = POINTER TO OBJECT
              termChar : CHAR;
              DEFERRED METHOD Read(VAR c : CHAR);
              DEFERRED METHOD Write(c : CHAR);
              METHOD WriteString(REF s : STRING);
              METHOD ReadString(VAR s : STRING);
              METHOD WriteLn;
            END;

METHOD Stream.WriteString(REF s : STRING);
VAR i : INTEGER;
BEGIN
  FOR i:=0 TO PRED(s.len) DO
    SELF.Write(s.data[i]);
  END;
END WriteString;

METHOD Stream.WriteLn;
BEGIN
  SELF.Write(ASCII.lf);
END WriteLn;

METHOD Stream.ReadString(VAR s : STRING);
VAR i : INTEGER := 0;
    c : CHAR;
BEGIN
  SELF.Read(c);
  WHILE c NOT OF ASCII.lf," ",ASCII.tab DO
    ASSERT(i<s'MAX,RangeViolation);
```



```
    s.data[i]:=c;
    INC(i);
    SELF.Read(c);
  END;
  termChar:=c;
  s.data[i]:=ASCII.null;
END ReadString;
```

Ein Objekt dieser Klasse wäre ziemlich sinnlos, da jeder Aufruf einer seiner Methoden zu einem Laufzeitfehler führen würde. Ein Erbe dieser Klasse kann allerdings durch Implementation von `Read()` und/oder `Write()` voll funktional werden. Es erbt auch alle erweiterten Methoden wie `WriteString()` etc. Erst durch dieses Erben entsteht eine funktionsfähige Klasse.

```
TYPE
  DosStream = POINTER TO OBJECT OF Stream;
                fh : Dos.FileHandlePtr;
                METHOD Read(VAR c : CHAR);
                METHOD Write(c : CHAR);
              END;
METHOD DosStream.Read(VAR c : CHAR);
VAR i : INTEGER;
BEGIN
  i:=Dos.Read(fh,c'PTR,1);
  IF KEY i
    OF 0 THEN RAISE(Dos.EOF)       END
    OF 1 THEN RAISE(Dos.ReadError) END
  END
END Read;

METHOD DosStream.Write(c : CHAR);
BEGIN
  ASSERT(Dos.Write(fh,c'PTR,1)=1,Dos.WriteError);
END Write;
```

Ein weiterer Vorteil liegt darin, daß Objekte dieser Klasse zuweisungsfähig an Objekte (bzw. Objektzeiger) der ursprünglichen Klasse sind.

Beispiel: ein Filter, der alle Nennungen von „Gott“ in „jenes höhere Wesen, das wir verehren“ (frei nach „Murkes gesammeltes Schweigen“) ersetzt.

```
PROCEDURE MurkeFilter(in,out : Stream);
VAR s : STRING(100);
BEGIN
  TRY
    LOOP
      in.ReadString(s);
      IF Strings.Equal(s,"Gott") THEN
        out.WriteString("jenes höhere Wesen, das wir verehren")
      ELSE
        out.WriteString(s);
      END;
      out.Write(in.termChar);
    END;
  EXCEPT
    OF Dos.EOF THEN END
  END;
END MurkeFilter;
```

Dieser Filter arbeitet mit allen Arten von funktionsfähigen Streams zusammen, obwohl er kein Wissen über deren vollständige Implementierung trägt.

Im obigen Beispiel könnte die Methode `WriteString` in `DosStream` aus Performancegründen ebenfalls überdefiniert werden:


```
METHOD DosStream.WriteString(REF s : STRING);
BEGIN
  ASSERT(Dos.Write(fh,s.data'PTR,s.len)=s.len,
         Dos.WriteError);
END WriteString;
```

Grundsätzlich kann jede Methode bei jedem Erbvorgang durch eine neue ersetzt werden. Wird sie das nicht, wird beim Aufruf die des Vorgängers verwendet.

Eine aufgeschobene Klasse ist eine Klasse, die keine eigentliche Funktionalität besitzt, und nur durch Beerben und Redefinition der aufgeschobenen Methoden sinnvoll wird.

4.5.3.3 Die Methoden `Construct` und `Destruct`

Häufig benötigen Objekte Hilfsmittel, um ihre Fähigkeiten zu erhalten (z. B. das `FileHandle` des Objektes `DosStream`). Diese Hilfsmittel müssen bei der Erzeugung des Objektes alloziert bzw. initialisiert und bei der Vernichtung des Objektes wieder freigegeben werden. Dazu dienen die parameterlosen Methoden `Construct` und `Destruct`. Sie werden bei der Erzeugung bzw. Vernichtung eines Objektes aufgerufen.

Beispiel:

```
TYPE
  DosStream = POINTER TO OBJECT OF Stream;
                fh : Dos.FileHandlePtr;
                METHOD Read(VAR c : CHAR);
                METHOD Write(c : CHAR);
                METHOD Destruct;
              END;

METHOD DosStream.Destruct;
BEGIN
```

```
    IF fh#NIL THEN
      Dos.Close(fh);fh:=NIL
    END;
  END Destruct;
```

Oder ein Filterobjekt, das jedem ASCII-Zeichen ein anderes zuordnen kann:

```
TYPE
  Filter    = POINTER TO OBJECT;
                table : POINTER TO ARRAY CHAR OF CHAR;
                METHOD SetFilter(from,to : CHAR);
                METHOD Translate(c : CHAR):CHAR;
                METHOD Construct;
                METHOD Destruct;
              END;
METHOD Filter.SetFilter(from,to : CHAR);
BEGIN
  table[from]:=to
END SetFilter;

METHOD Filter.Translate(c : CHAR):CHAR;
BEGIN
  RETURN table[c]
END Translate;

METHOD Filter.Construct;
VAR c : CHAR;
BEGIN
  Resources.New(table);
  FOR c:=CHAR'MIN TO CHAR'MAX DO
    table[c]:=c;
  END;
END Construct;
```



```
METHOD Filter.Destruct;
BEGIN
  Resources.Dispose(table)
END Destruct;
```

Diese Methoden haben zwei große Unterschiede zu allen anderen normalen Methoden; erstens werden sie vom Laufzeitsystem aufgerufen, zweitens wird nicht die jüngste Methode aufgerufen, sondern alle, die sich im Stammbaum angesammelt haben. Eine Con/Destruct Methode sollte sich also nur um Dinge kümmern, die direkt mit der aktuellen Ausbaustufe des Objekts zusammenhängen. Spezielle Initialisierungen sollten im `CONSTRUCTOR` vorgenommen werden (siehe 4.5.4).

4.5.4 Erzeugung und Vernichtung von Objekten

Da bei der Erzeugung von Objekten im allgemeinen Initialisierungen vorgenommen werden müssen, scheidet ein einfaches Allozieren von vorneherein aus. Objekte können auf zwei Arten erzeugt werden, durch einen Constructor (eine besondere Methode) oder durch die Standardprozedur `NEW` (evtl. nicht mehr lange). Die Vernichtung erfolgt analog durch einen Destructor bzw. `DISPOSE`. Beispiel:

```
TYPE
  DosStream = POINTER TO OBJECT OF Stream;
                fh : Dos.FileHandlePtr;
                CONSTRUCTOR Create(REF name : STRING);
                CONSTRUCTOR Open(REF name : STRING);
                DESTRUCTOR Close;

                METHOD Read(VAR c : CHAR);
                METHOD Write(c : CHAR);
                METHOD Destruct;
              END;
```

```
METHOD DosStream.Create(REF name : STRING);
BEGIN
  fh:=Dos.Open(name,Dos.newFile)
  ASSERT(fh#NIL,Dos.ObjectNotFound);
END Create;

METHOD DosStream.Open(REF name : STRING);
BEGIN
  fh:=Dos.Open(name,Dos.oldFile)
  ASSERT(fh#NIL,Dos.ObjectNotFound);
END Open;

METHOD DosStream.Close;BEGIN END Close;
```

Wird für ein (im allgemeinen noch nicht existierendes Object) ein Constructor aufgerufen, wird dieses erzeugt (anhand des statischen Typs). Danach wird die Constructor Methode (in diesem Fall z. B. `Open`) aufgerufen, die weitere Initialisierungen vornehmen kann. Alle Instanzvariablen eines Objektes werden mit `0`, `NIL`, `FALSE` etc. vorinitialisiert. Die Vernichtung läuft umgekehrt ab, erst wird der Destructor aufgerufen, dann das Objekt vernichtet. Im obigen Beispiel ist `Close` mehr pro forma deklariert, da die eigentliche Vernichtung des FileHandles durch die Methode `Destruct` vorgenommen wird.

Eine Klasse kann beliebig viele Constructoren besitzen, auch dürfen diese als einzige Methode durch Constructoren mit anderen Parametern überdefiniert werden, da sie statisch (aus dem statischen Typ) ermittelt werden.

Beispiel:


```
VAR in,out : DosStream;

BEGIN
  in.Open("Rede die 1.");
  out.Create("Rede die 2.");
  MurkeFilter(in,out);
  out.Close;
  in.Close
END ...
```

Die Erzeugung mit `NEW` und `DISPOSE` sollte nur für sehr einfache Objekte verwendet werden, ihre weitere Existenz ist außerdem fraglich.

```
in:=NEW(DosStream);
out:=NEW(DosStream);
DISPOSE(in);
DISPOSE(out);
```

Eine weitere Möglichkeit, ein Objekt zu erzeugen, ist ein bestehendes zu verdoppeln (Klonen). Dies kann durch die Standardfunktion `CLONE` geschehen. (Allerdings wird wohl in Zukunft eine andere Technik verwendet werden).

```
p:=CLONE(q);
```

Der Unterschied zwischen `Con-/Destruct` und `CON-/DESTRUCTOR`

Die `CON-/DESTRUCTOR`en werden vom Programmierer zur Erzeugung bzw. Vernichtung eines Objektes eingesetzt. Da es mehrere Arten gibt, wie ein Objekt erzeugt werden kann, und auch evtl. Parameter für die Initialisierung benötigt werden, kann eine Klasse mehrere `CON-/ DESTRUCTOR`en mit verschiedenen Parametern besitzen.

Die Methoden `Con-/Destruct` werden vom Laufzeitsystem während der Initialisierung und Vernichtung aufgerufen[3]. Sie dienen einer grundlegenden Initialisierung bzw. einer Freigabe von Betriebsmitteln, wenn ein Objekt unter Notfallbedingungen (z. B. Laufzeitfehler, verlassen eines Kontexts) gerät. Sie können auch die Verwaltung von Betriebsmitteln an übergeordnete Schichten verheimlichen.

4.5.5 Mehrfacherben (multiple inheritance)

Oft ist es sinnvoll, wenn eine Klasse nicht nur von einer Vorgängerklasse, sondern von beliebig vielen erben kann.

Beispiel:

```
TYPE
  InputStream   = POINTER TO OBJECT
                    termChar : CHAR;
                    DEFERRED METHOD Read(VAR c : CHAR);
                    METHOD ReadString(VAR s : STRING);
                  END;
  OutputStream  = POINTER TO OBJECT
                    DEFERRED METHOD Write(c : CHAR);
                    METHOD WriteString(s : STRING);
                    METHOD WriteLn;
                  END;

  InOutStream   = POINTER TO OBJECT OF InputStream,
                                       OutputStream;
                  END;
```

Ein Objekt der Klasse `InOutStream` vereinigt die Fähigkeiten eines

[3] Also beim Aufruf von `New/Dispose` oder bei der Verwendung eines Constructors/Destructors

`InputStream` mit denen eines `OutputStream` und ist auch zuweisungskompatibel zu beiden.

Beispiele für legale Anweisungen und Ausdrücke:

```
VAR in  : InputStream;
    out : OutputStream;
    io  : InOutStream;
BEGIN
  ...
  in:=io;
  out:=io;
  IF in  IS InOutStream THEN ...
  IF out IS InOutStream THEN ...
  ...
  MurkeFilter(io,out);
```

falls dieser definiert wird als

```
PROCEDURE MurkeFilter(in  : InputStream;
                      out : OutputStream);
```

4.5.5.1 Probleme beim Mehrfacherben

Schwierig wird Mehrfacherben, wenn in der Hierarchie gleiche Bezeichner auftauchen. Dies kann auf zwei Arten geschehen, erstens, in einem der Väter taucht ein Bezeichner auf, der auch in einem anderen existiert, oder zweitens, eine Klasse ist zweimal Erbe derselben Klasse. (ACHTUNG, der Compiler führt zur Zeit noch keinen Test bei Mehrfacherben aus, es kann also unerkannt zu Problemen kommen. Dieser Mangel wird demnächst behoben.)

Diesem Problem kann durch Qualifizierung beigekommen werden, dabei werden einem oder mehreren Elternteilen qualifizierende Bezeichner beigestellt, die bei der Referenzierung ihrer Elemente benutzt werden können.

Beispiel: Knoten für eine Kreuzliste:

```
TYPE
  DoubleNode = POINTER TO OBJECT OF Node AS h,
                                    Node AS v;
               END;
VAR dn : DoubleNode;
```

Auf die Elemente der `DoubleNode` kann nun über die qualifizierenden Bezeichner `.h` und `.v` zugegriffen werden. Die volle Mächtigkeit kommt erst durch Generizität zum Tragen.

Eine andere Lösung wäre folgende: Wenn eine Klasse in der Hierarchie mehrmals auftaucht, sie dennoch nur einmal vorhanden ist. Dies wäre sehr sinnvoll für verschiedene Anwendungen, wirft aber neue Probleme auf, die im Moment eine Implementation noch verzögern (z. B. wenn aus der bestehenden Hierarchie zwei verschiedene Methodenredefinitionen für diese Klasse existieren).

4.5.6 Objekte und Generizität

Generizität bedeutet, daß Module mit abstrakten Datentypen definiert werden können, die dann für tatsächlich existierende Typen ausgeprägt werden. Die aktuellen Typen können dabei durch Forderungen (im allgemeinen geschieht dies durch eine Nachfolger-Vorgänger Bedingung) eingegrenzt werden.

Es stellt sich die Frage, wozu generische Module, wenn doch der Typ eines Objektes auch dynamisch ermittelt werden kann und somit also keine illegalen Zuweisungen oder Verwendungen möglich sind.

Die Antwort ist einfach, Generizität erhöht die statische Sicherheit eines Programmes, vermeidet somit sowohl unnötige Typchecks als auch mögliche Laufzeitfehler.

Die Regeln für generische Module sind im Prinzip dieselben wie die bei Records. Lediglich durch das Mehrfacherben ergibt sich eine Erweiterung.

Definition:

```
DEFINITION MODULE Lists ( Node : POINTER TO NodeObj);
  TYPE
    NodeObj = OBJECT
                prev,next : Node;
              END;
    List    = POINTER TO OBJECT
                first,last : Node;
                METHOD InsertFirst(n : Node);
                METHOD RemoveNode(n : Node);
              END;
END Lists;
```

Ausprägung:

```
TYPE
  TextNode = POINTER TO TextNodeObj;

DEFINITION MODULE TextLists = Lists(TextNode);

TYPE
  TextNodeObj = OBJECT OF TextLists.Node;
                  text : CLASSPTR TO STRING
                END;
  TextList    = TextLists.List;
```

Mehrfache Ausprägung:

```
TYPE
  Edge       = POINTER TO EdgeObj;

DEFINITION MODULE FLists     = Lists(Edge.fnode);
DEFINITION MODULE TLists     = Lists(Edge.tnode);

TYPE
  Vertex     = POINTER TO OBJECT OF FLists.List AS from,
                                    TLists.List AS to;
               END;
  EdgeObj    = OBJECT OF FLists.Node AS fnode,
                         TLIsts.Node AS tnode;
                 from,to : Vertex;
                 METHOD Add(from,to : Vertex);
                 METHOD Remove;
               END;

 METHOD Edge.Add(from,to : Vertex);
 BEGIN
   from.from.InsertFirst(SELF);SELF.from:=from;
   to.to.InsertFirst(SELF);SELF.to:=to;
 END Add;

 METHOD Edge.Remove;
 BEGIN
   from.from.Remove(SELF);from:=NIL;
   to.to.Remove(SELF);to:=NIL;
 END Remove;
```

Anhand der generischen Ausprägung der beiden Listenknoten in `Edge` wird beim Aufruf von `InsertFirst`/`Remove` der richtige der beiden Knoten automatisch erkannt. Andernfalls müßten die Knoten in der Kante über die Qualifizierung bezeichnet werden. Ein weiteres Beispiel wäre ein Knoten, der sowohl von einem Listenknoten, als auch von einem Baum erbt, und somit sowohl in eine Liste, als auch in einen Baum eingehängt

werden kann.

4.5.7 Resourcetracking mit Objekten

Objekte werden wie alle anderen Speicherstücke auch über Resources alloziert. Die dazu verwendeten Funktionen sind jedoch gegenüber dem Programmierer durch Constructoren und Destructoren versteckt. Objekte existieren relativ zu einem Kontext; bei dessen Vernichtung werden auch sie vernichtet. Um objektbezogene Resourcen (wie zusätzlichen Speicher oder Systemelemente) immer mit dem Objekt (also auch bei Freigabe des Objekts durch Kontextvernichtung) freizugeben, sollten `Destruct`-Methoden verwendet werden (siehe auch Beispiel in 4.5.3.3).

Ein anderes Problem besteht darin, daß Objekte immer zum aktuellen Kontext erzeugt werden. Dies ist in manchen Fällen nicht wünschenswert. Dafür besteht die Möglichkeit, den Kontext eines Objektes zu ändern, es also in einen anderen Existenzbereich umzuhängen. Dies wird vorläufig durch die Resources Funktion `ChangeObjContext` realisiert.

Beispiel:

```
TYPE
  DosStream = POINTER TO OBJECT OF Stream;
                fh : Dos.FileHandlePtr;

                CONSTRUCTOR Create(REF name : STRING;
                                       con : Context := NIL);
                CONSTRUCTOR Open(REF name : STRING;
                                     con  : Context := NIL);
                DESTRUCTOR Close;

                METHOD Read(VAR c : CHAR);
                METHOD Write(c : CHAR);
                METHOD Destruct;
              END;

METHOD DosStream.Create(REF name : STRING;con : Context);
BEGIN
  IF con#NIL THEN
    Resources.ChangeObjContext(SELF,con);
  END;
  fh:=Dos.Open(name,Dos.newFile)
  ASSERT(fh#NIL,Dos.ObjectNotFound);
END Create;

METHOD DosStream.Open(REF name : STRING;con : Context);
BEGIN
  IF con#NIL THEN
    Resources.ChangeObjContext(SELF,con);
  END;
  fh:=Dos.Open(name,Dos.oldFile)
  ASSERT(fh#NIL,Dos.ObjectNotFound);
END Open;
```



```
METHOD DosStream.Close;BEGIN END Close;
```

Die Funktion `ChangeObjContext` wird, wenn diese Lösung beibehalten wird, zu einer Standardfunktion erhoben.

Sie sehen, es sind noch nicht alle Probleme gelöst, wir sind daher sehr daran interessiert, Ihre Meinung zum momentanen Ansatz zu hören, damit wir Ihre Ideen und Anregungen in die endgültige Implemetation einfließen lassen können.

4.6 Dynamische Strukturen

Bei den wenigsten Problemen und Programmen kommt man mit *statischen* Datenstrukturen, wie Arrays oder Records aus. Meist ist die Größe der zu speichernden Daten nicht bekannt oder schwankt während der Laufzeit extrem. Doch selbst bei bekannter Größe kann eine *dynamische* Struktur sinnvoll sein, da sie sich fast ohne Aufwand umordnen läßt, so daß es nicht nötig ist, bei jeder Operation den ganzen Speicher hin und her zu schieben.

Die wichtigsten Elemente dynamischer Strukturen sind *Zeiger* (Pointer) und *Knoten* (Node).

Knoten sind die eigentlichen Träger der Daten, Zeiger stellen die Verbindungen zwischen ihnen dar, wie Städte, die durch Straßen verbunden sind. Ein Zeiger ist eine Verbindung eines Knotens zu einem anderen, dessen Richtung bekannt ist (Einbahnstraße). Die Repräsentanz eines Zeigers im Computer ist die Adresse des angepeilten Knotens.

Beispiel:

```
TYPE
  Ptr    = POINTER TO Node;
  Node   = RECORD
             ...
           END;
VAR
  p,q    : Ptr;
  n      : Node;
```

Variablen vom Typ `Node` sind die Informationsträger, die des Typs `Ptr` sind nur Verweise darauf. Man sagt, `Ptr` zeigt auf `Node`.

Ein Zeiger selbst trägt keine eigentliche Information, er erlaubt lediglich den Zugriff darauf. Ein nicht initialisierter Zeiger zeigt irgendwohin, ihm muß erst noch ein Ziel zugewiesen werden. Dies kann auf drei

Arten geschehen, ihm wird Platz für sein Informationspaket beschafft (alloziert) `New(p)`, ihm wird das Ziel eines anderen Zeigers zugewiesen `p:=q` oder er bekommt einen bereits vorhandenen Knoten als Ziel `p:=n'PTR`. Die Prozedur `New()` wird aus dem Modul `Resources` importiert.

Ein Zeiger kann auch bewußt nirgendwohin zeigen (man spricht auch von „geerdeten“ Zeigern), dazu weist man ihm als Ziel `NIL` zu. `NIL` ist eine Konstante für alle Zeiger, und bedeutet „nirgends“.

Ein noch nicht initialisierter Zeiger zeigt nicht nach `NIL`, sondern irgendwohin in den Speicher. Und die einfachste Möglichkeit einen Rechner nach Indien zu schicken ist, einen nicht initialisierten Zeiger zu benutzen.

Um von einem Zeiger auf den durch ihn bezeichneten Knoten zu kommen, muß man ihn dereferenzieren. Dies geschieht durch einen nachgestellten Hochpfeil „`^`“, also z. B. `p^`. Das Ergebnis dieser Operation ist der durch den Zeiger bezeichnete Knoten. Wenn also `p:=n'Ptr` gesetzt wird, ist `p^` genau `n`.

Um das Ziel, das durch einen Zeiger bezeichnet wird, wieder zu vernichten, d. h. den dadurch belegten Speicherplatz freizugeben, benutzt man `Dispose(p)`. Dabei wird `p` auch gleich auf `NIL` gesetzt. Die Freigabe eines nicht mit `New` besorgten Speicherstückes führt zu einem Laufzeitfehler.

Das Belegen und wieder Freigeben von Datenpaketen ist der Grund, warum auf diese Art erzeugte Strukturen als *dynamisch* bezeichnet werden, im Gegensatz zu *statischen* Strukturen, die immer dieselbe Ausdehnung besitzen.

Zur einfacheren Darstellung dynamischer Strukturen benutzt man meist ein *Pfeil-Kästchen-Diagramm*, in dem Zeiger durch Pfeile und Knoten durch Kästchen dargestellt werden. Ist der Zeiger `NIL`, dann wird statt des Pfeils ein Punkt gezeichnet.

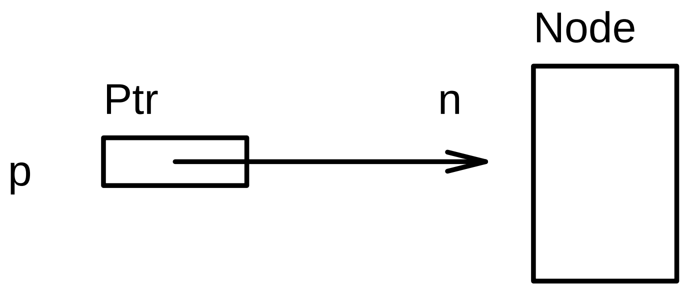


Abbildung 4.5: Pointer p zeigt auf Speicherinhalt n

4.6.1 Einfach verkettete Listen

Die einfach verkettete *Liste* ist die einfachste dynamische Datenstruktur. Sie ist relativ unflexibel, aber einfach zu erzeugen und zu vernichten. Das Prinzip ist, daß jeder *Knoten* der Liste einen *Zeiger* auf einen Nachfolgeknoten enthält. Der erste Knoten wird durch einen eigenen Zeiger „gehalten“, der letzte zeigt auf `NIL`, um das Ende zu markieren.

```
TYPE
  List  = POINTER TO Node;
  Node  = RECORD
            next  : List;
            data  : INTEGER
          END;

VAR
  myList: List;
BEGIN
  myList:=NIL;  (* Liste als leer kennzeichnen *)
...
```

Das Verhalten dieser Liste ist das einer Seilschaft bei Bergsteigern, der erste hält sich am Gipfel fest (`myList`), alle weiteren hängen an ei-

nem Seil an ihrem Vordermann (`next`). Am letzten hängt keiner mehr (`next=NIL`). Wird das Seil gekappt, und das Ende verloren, gehen alle Mitglieder dahinter den Bach, nein, Berg hinab; verliert man also den Zeiger auf ein Element der Liste, sind auch alle Nachfolger verloren.

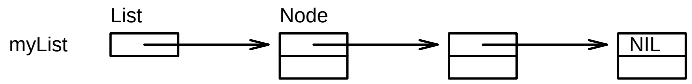


Abbildung 4.6: Eine einfach verkettete Liste

Das Einfügen am Anfang der Liste ist recht einfach, man muß nur einem neuen Element das bisher erste als Nachfolger zuweisen, und den Zeiger auf den Listenanfang auf das neue umbiegen.

```
PROCEDURE Insert(VAR list : List; data : INTEGER);
VAR p : List;
BEGIN
  New(p);
  p^.data:=data;
  p^.next:=list;
  list:=p;
END Insert;
```

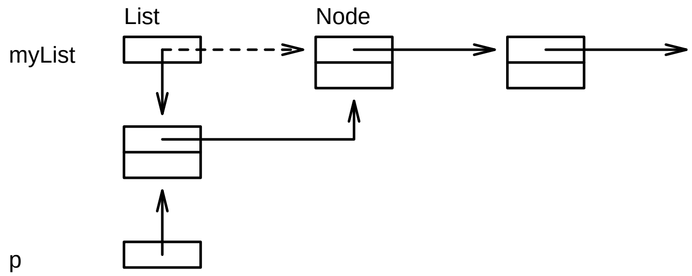


Abbildung 4.7: Einfügen am Anfang

Einfügen nach einem Element ist auch kein Hexenwerk, das Verfahren ist dasselbe wie eben, nur muß für `list` der Nachfolgezeiger des entsprechenden Elements übergeben werden, also z. B.: `Insert(e^.next,12)`.

Das Suchen in einer Liste erfolgt der Reihe nach, wobei das Ende der Liste, und somit der Mißerfolg der Suche, durch einen Zeiger auf `NIL` bezeichnet wird.

```
PROCEDURE Find(list : List; data : INTEGER):List;
BEGIN
  WHILE (list#NIL) AND (list^.data#data) DO
    list:=list^.next
  END;
  RETURN list
END Find;
```

Löschen des ersten Elements einer Liste ist ebenfalls recht einfach, man muß nur den Listenanfang ein Element weiterschieben, und dann das alte Anfangselement freigeben.


```
PROCEDURE DeleteFirst(VAR list : List);
VAR p : List;
BEGIN
  p:=list;
  list:=list^.next;
  Dispose(p)
END DeleteFirst;
```

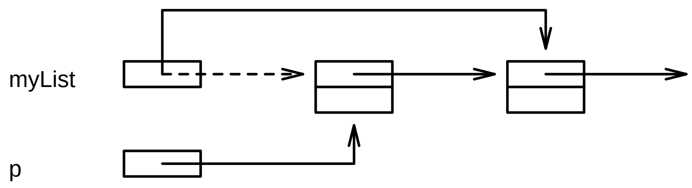


Abbildung 4.8: Löschen des ersten Elements

Das Löschen eines Elements aus der Mitte heraus ist nur dann möglich, wenn sein Vorgänger bekannt ist, da dessen Nachfolgezeiger umgebogen werden muß. Soll ein bestimmtes Element gelöscht werden, das durch sein Datenfeld bezeichnet wird, muß beim Suchen auch berücksichtigt werden, daß der Vorgänger ebenfalls noch zur Verfügung steht. Dafür existieren zwei grundlegende Methoden, *Indirektion* und *Schleppzeiger*.

Bei der Indirektion wird ein Element immer durch den Zeiger seines Vorgängers angesprochen `p^.next^.data`, so daß der Vorgänger des entsprechenden Elements automatisch bekannt ist.

```
PROCEDURE DeleteElem(VAR list : List;
                         data : INTEGER);
VAR p : List;
BEGIN
  IF list#NIL THEN
    IF list^.data=data THEN
      DeleteFirst(list)
    ELSE
      p:=list;
      WHILE (p^.next#NIL)
        AND_WHILE (p^.next^.data#data) DO
          p:=p^.next
        ELSE
          DeleteFirst(p^.next)
        END
      ELSE
        RAISE2(NotFoundError)
      END;
    END
  ELSE
    RAISE2(EmptyListError)
  END
END DeleteElem;
```

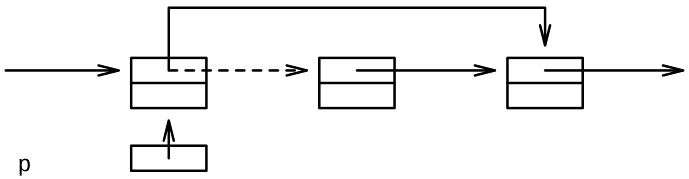


Abbildung 4.9: Löschen eines Nachfolgers

Bei der anderen Methode werden zwei Zeiger verwendet, wobei der eine, der *geschleppte* immer auf das Vorgängerelement zeigt.


```
PROCEDURE DeleteElem(VAR list : List;
                         data : INTEGER);
VAR s,p : List;
BEGIN
  p:=list;
  IF p#NIL THEN
    s:=NIL;
    WHILE p#NIL
      AND_WHILE p^.data#data DO
        s:=p;
        p:=p^.next
      ELSE
        IF s=NIL THEN
          DeleteFirst(list)
        ELSE
          DeleteFirst(s^.next)
        END
      END
    ELSE
      RAISE2(NotFoundError);
    END;
  ELSE
    RAISE2(EmptyListError);
  END
END DeleteElem;
```

Bei diesen beiden Verfahren ist immer der Fall zu berücksichtigen, daß das zu löschende Element schon das erste sein kann. Um dem zu entgehen verwendet man manchmal ein Hilfselement, daß als erstes Element in der Liste liegt, aber keine Daten enthält. Dieses Element wird nie gelöscht, neue Elemente werden immer nach ihm eingefügt.

Im allgemeinen, werden Sie jedoch nicht darauf angwiesen sein, selbst eine solche Liste zu programmieren. Für diese Anwendung existiert ein generisches Modul im Standardmodul `Lists`, das alle hier beschriebenen Routinen und noch einige mehr enthält.

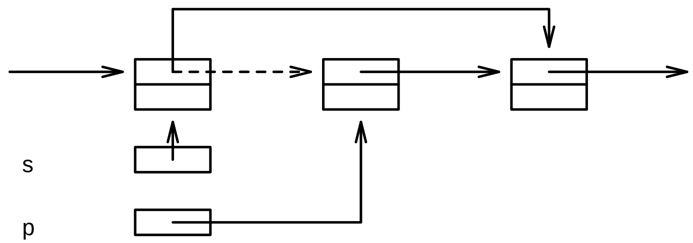


Abbildung 4.10: Löschen mit zwei Zeigern

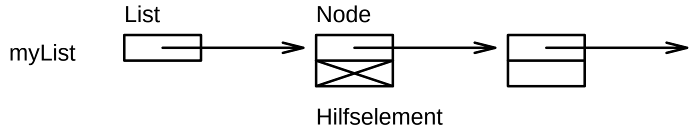


Abbildung 4.11: Liste mit Hilfselement

Um dem Problem, den Vorgänger zu finden, zu entkommen, verwendet man doppelt verkettete Listen.

4.6.2 Doppelt verkettete Listen

Bei diesem Listentyp zeigt jeder Knoten nicht nur auf seinen Nachfolger sondern auch auf seinen Vorgänger.


```
TYPE
  List  = POINTER TO Node;
  Node  = RECORD
            prev,
            next   : List;
            data   : INTEGER;
          END;
VAR
  myList: NodePtr;
```

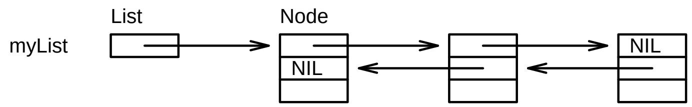


Abbildung 4.12: Doppelt verkettete Liste

Einfügen in diese Liste ist etwas komplizierter, als bei einer einfachen Liste, da ja beide Zeiger gesetzt werden müssen.

```
PROCEDURE InsertStart(VAR list : List;
                          data : INTEGER);
VAR p : List;
BEGIN
  New(p);
  p^.data:=data;
  p^.prev:=NIL;
  p^.next:=list;
  IF list#NIL THEN
    list^.prev:=p;
  END;
  list:=p;
END InsertStart;
```

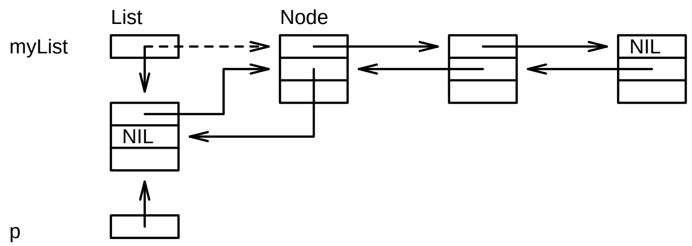


Abbildung 4.13: Einfügen am Anfang der Liste

```
PROCEDURE InsertAfter(prev : List;
                      data : INTEGER);
VAR p : List;
BEGIN
  New(p);
  p^.data:=data;
  p^.prev:=prev;
  p^.next:=prev^.next;
  IF prev^.next#NIL THEN
    prev^.next^.prev:=p;
  END;
  prev^.next:=p;
END InsertAfter;
```

Dafür ist aber das Löschen eines Elementes wesentlich einfacher, da der Vorgänger nicht mehr bekannt sein muß.

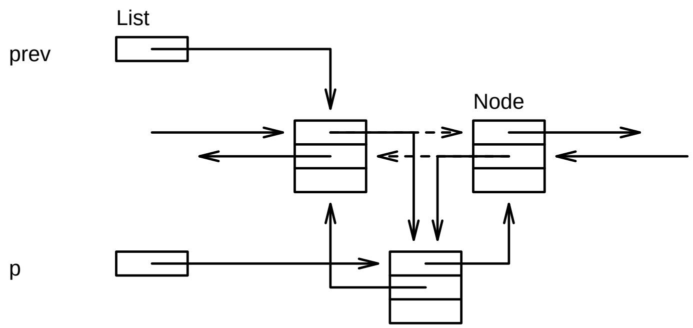


Abbildung 4.14: Einfügen hinter einem Element

```
PROCEDURE Delete(VAR list : List;
                     elem : List);
BEGIN
  IF elem^.prev=NIL THEN
    list^.next:=elem^.next;
  ELSE
    elem^.prev^.next:=elem^.next
  END;
  IF elem^.next#NIL THEN
    elem^.next^.prev:=elem^.prev
  END;
END Delete;
```

Auch für doppelt verkettete Listen exisiert ein generisches Modul im Standardmodul `Lists`.

Um dem Problem des ersten Knotens zu entkommen, kann auch hier ein Hilfselement am Anfang benutzt werden. Noch einfacher ist allerdings die Verwendung einer Ringliste.

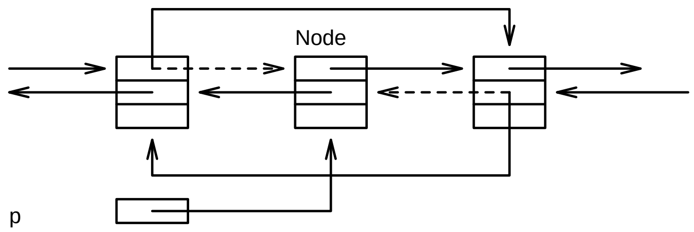


Abbildung 4.15: Löschen eines Elements

4.6.3 Ringlisten (zirkuläre Listen)

Bei einer *Ringliste* zeigt das letzte Element nicht nach `NIL`, sondern auf das erste Element der Liste (bei doppelter Verkettung auch das erste auf das letzte). Noch einfacher zu handhaben ist eine Ringliste mit Hilfselement.

Das Amiga Betriebssystem verwendet eine seltsame Abart davon in fast allen Strukturen des Systems. Die Prozeduren zur Verwaltung solcher Systemlisten sind im Kapitel 5.3.2 über das Programmieren des Betriebssystems beschrieben.

Hier ist unsere Form einer Ringliste:

```
TYPE
  List  = POINTER TO Node;
  Node  = RECORD
            prev,
            next   : List;
            data   : INTEGER;
          END;
VAR
  myList: List;
```

Das Erzeugen einer Liste dieses Typs verlangt etwas mehr Aufwand, als bei Listen ohne Hilfselemt, da dieses ja erzeugt werden muß.

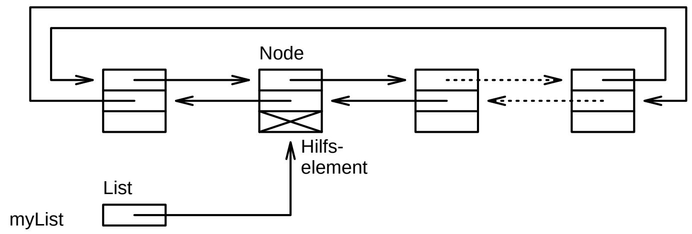


Abbildung 4.16: Ringliste

```
PROCEDURE Create(VAR list : List);
BEGIN
  New(list);
  list^.next:=list;
  list^.prev:=list;
END Create;
```

Die Bedingung für eine leere Liste ist auch nicht so einfach.

```
PROCEDURE Empty(list : List):BOOLEAN;
BEGIN
  RETURN list^.next=list
END Empty;
```

Dafür ist das Einfügen in so eine Liste an beliebiger Position nicht sehr aufwendig.

```
PROCEDURE InsertAfter(elem : List;data : INTEGER);
VAR p : List;
BEGIN
  New(p);
  p^.data:=data;
  p^.next:=elem^.next;
  p^.prev:=elem;
  elem^.next^.prev:=p;
  elem^.next:=p;
END InsertAfter;

PROCEDURE InsertBefore(elem : List;data : INTEGER);
VAR p : List;
BEGIN
  New(p);
  p^.data:=data;
  p^.next:=elem;
  p^.prev:=elem^.prev;
  elem^.prev^.next:=p;
  elem^.prev:=p;
END InsertBefore;
```

Auch das Entfernen eines Elements ist absolut problemlos.

```
PROCEDURE Delete(elem : List);
BEGIN
  elem^.prev^.next:=elem^.next;
  elem^.next^.prev:=elem^.prev;
  Dispose(elem);
END Delete;
```

Implementieren Sie doch eine solche Ringliste als generisches Modul, dann können Sie sie immer wieder für jeden beliebigen Typen verwenden.

Zum Schluß noch ein Beispiel, Einfügen in eine sortierte Liste, also eine Liste in der jedes Element kleiner oder gleich seinem Nachfolger ist (`elem^.data<=elem^.next^.data`). Um das Element in der Liste zu finden, vor dem das neue Element eingefügt werden muß, empfiehlt sich die Suche mit einem Wächterwert.

```
PROCEDURE InsertSorted(list : List;
                       data : INTEGER);
BEGIN
  list^.data:=data;  (* Wächterelement *)
  REPEAT
    list:=list^.next
  UNTIL list^.data<=data;
  InsertBefore(list,data);
END InsertSorted;
```

4.6.4 Bäume

Ein *Baum* ist eine andere dynamische Datenstruktur, im Unterschied zur Liste kann hier jeder Knoten über mehrere Nachfolger verfügen. Alle Knoten, bei Bäumen auch *Äste* und *Blätter* genannt, haben genau einen Vorgänger, bis auf einen, die *Wurzel*, der keinen Vorgänger besitzt. Ein Beispiel für einen Baum mit vier Ästen:

```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
            sons : ARRAY [0..3] OF Tree;
            data : INTEGER;
          END;
VAR
  myTree: Tree;
```

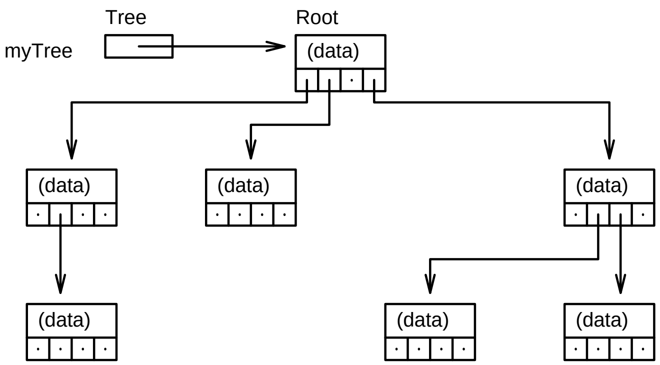


Abbildung 4.17: Baum mit vier Ästen

In einem Baum mit beliebig vielen Ästen pro Knoten, verwendet man am besten eine Liste, die die Nachfolger enthält.

```
TYPE
  Tree  = POINTER TO Root;
  List  = POINTER TO Node;
  Node  = RECORD
            next : List;
            data : Tree;
          END;
  Root  = RECORD
            sons : List;
            data : INTEGER;
          END;
```

Diese Mischstruktur läßt sich auch zu einer einzigen zusammenfassen.

```
  Tree  = POINTER TO Root;
  Node  = RECORD
            son,
```

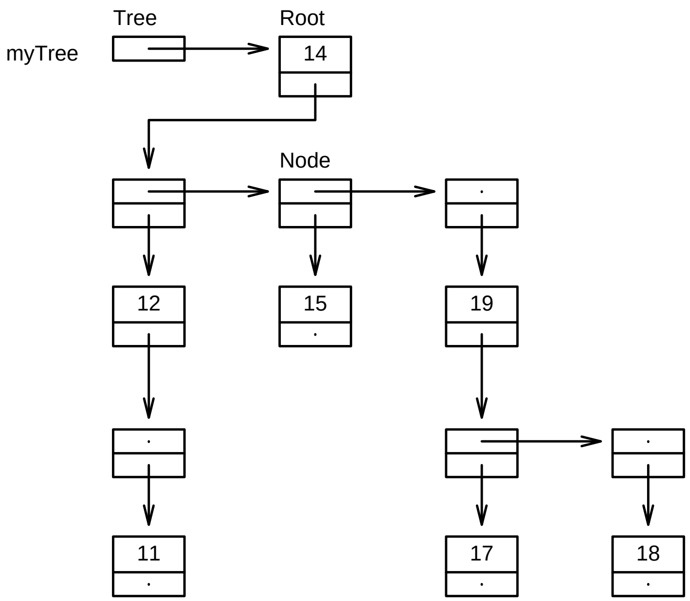



Abbildung 4.18: Baum als Mischstruktur

```
      brother : Tree;
      data    : INTEGER;
    END;
```

Funktionen zur Erzeugung und zur Verwaltung eines solchen Baumes, finden Sie in einem generischen Modul im Standardmodul `Trees`.

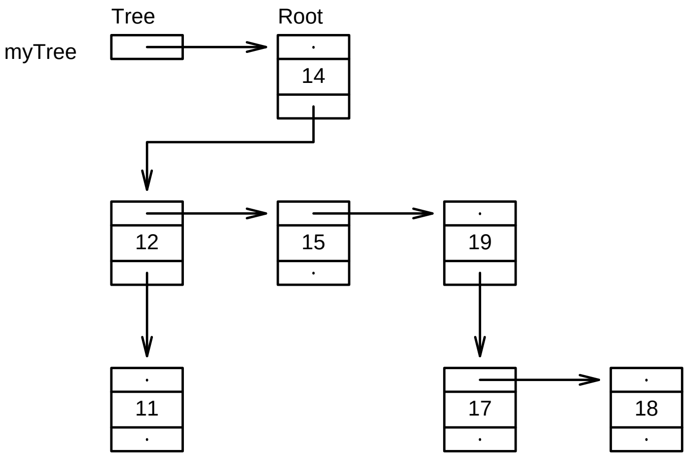


Abbildung 4.19: Vereinheitlichte Mischstruktur

Hierbei zeigt ein Knoten nicht nur auf seinen Nachfolger, sondern auch auf einen Mitnachfolger seines Vorgängers. Die wichtigste Art Baum ist der *Binärbaum*. Bei ihm hat jeder Knoten maximal *zwei* Nachfolger, wobei sich der *linke* vom *rechten* unterscheidet.

```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
      left,
      right : Tree;
      data  : INTEGER;
```



```
    END;
```

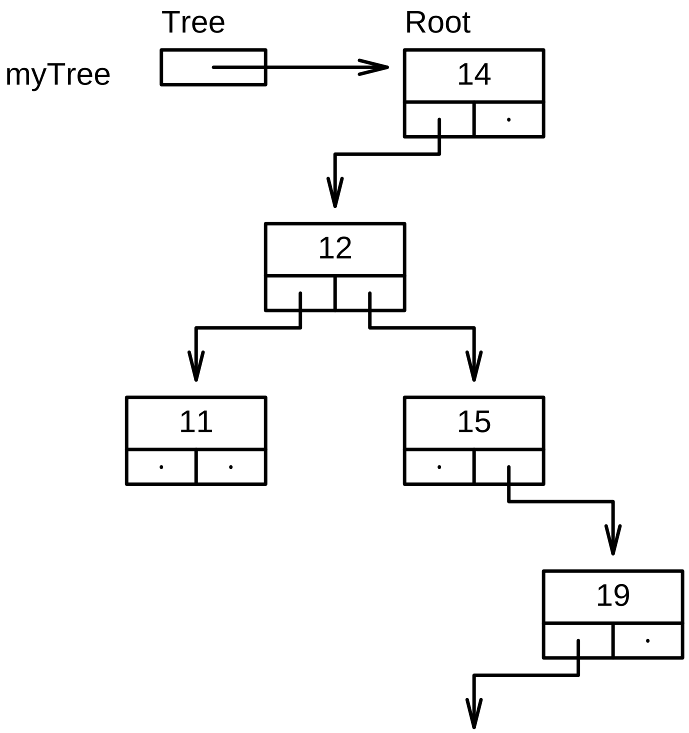


Abbildung 4.20: Binärbaum

Auch für die Anwendung dieses Baumtyps finden Sie Routinen im Modul `Trees`.

Ein Beispiel für die Verwendung eines Binärbaumes ist ein *Parsebaum* eines mathematischen Ausdrucks:

```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
            IF KEY type : CHAR
              OF "+","-","*","/"  THEN left,
                                       right  : Tree;
              OF "C"              THEN value  : INTEGER;
            END;
          END;
```

Der Ausdruck $3+4*(5/2+6)$ wäre in der Darstellung mit diesem Baum unter Berücksichtigung von Punkt vor Strich:

Die Ausgabe eines derartigen Baumes ist auf drei Arten möglich, *Prefix* (die Ausgabe der Wurzel erfolgt vor den Ästen), *Infix* (die Wurzel wird zwischen den beiden Ästen ausgegeben) und *Postfix* (die Wurzel wird am Schluß ausgegeben).

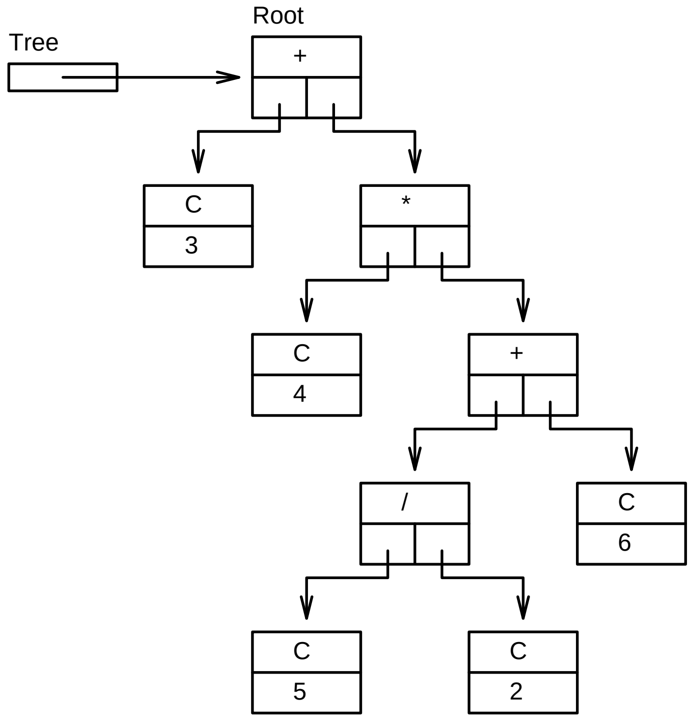


Abbildung 4.21: Parsebaum

```
PROCEDURE WritePrefix(tree : Tree);
BEGIN
  IF KEY tree^.type
    OF "+","-","*","/" THEN
                         Write(tree^.type);
                         Write(" ");
                         WritePrefix(tree^.left);
                         WritePrefix(tree^.right);
                       END
    OF "C"             THEN
                         WriteInt(tree^.value,0);
                         Write(" ")
                       END
  END
END WritePrefix;
```

```
=> + 3 * 4 + / 5 2 6
```

```
PROCEDURE WriteInfix(tree : Tree);
BEGIN
  IF KEY tree^.type
    OF "+","-","*","/" THEN
                         Write("(");
                         WriteInfix(tree^.left);
                         Write(tree^.type);
                         WritePrefix(tree^.right);
                         Write(")");
                       END
    OF "C"             THEN
                         WriteInt(tree^.value,0);
                       END
  END
END WriteInfix;
```

```
=> (3+(4*((5/2)+6)))
```



```
PROCEDURE WritePostfix(tree : Tree);
BEGIN
  IF KEY tree^.type
    OF "+","-","*","/" THEN
                         WritePostfix(tree^.left);
                         WritePostfix(tree^.right);
                         Write(tree^.type);
                         Write(" ");
                       END
    OF "C"             THEN
                         WriteInt(tree^.value,0);
                         Write(" ")
                       END
  END
END WritePostfix;
```

```
=> 3 4 5 2 / 6 + * +
```

So ein Parsebaum wird immer im Postfix ausgewertet, da ja, um eine Operation ausführen zu können, zuerst die beteiligten Operanden ermittelt sein müssen.

```
PROCEDURE Eval(tree : Tree):INTEGER;
BEGIN
  IF KEY tree^.type
    OF "+" THEN RETURN Eval(tree^.left)
                       +   Eval(tree^.right) END
    OF "-" THEN RETURN Eval(tree^.left)
                       -   Eval(tree^.right) END
    OF "*" THEN RETURN Eval(tree^.left)
                       *   Eval(tree^.right) END
    OF "/" THEN RETURN Eval(tree^.left)
                       DIV Eval(tree^.right) END
    OF "C" THEN RETURN tree^.value           END
  END
```

```
END Eval;
```

Auch ein Stammbaum ist ein Baum in unserem Sinne. Man muß hierbei zwei Typen unterscheiden, einmal die Rückverfolgung eines Familienmitgliedes zu seinen Ahnen hin, dies läßt sich mit einem Binärbaum realisieren.

```
TYPE
  Parent = POINTER TO Person;
  Person = RECORD
             name    : STRING(32);
             father,
             mother  : Parent;
           END;
```

Oder die Auflistung aller Nachkommen einer Person:

```
TYPE
  Child  = POINTER TO Person;
  Person = RECORD
             name     : STRING(32);
             children,
             brother  : Child;
           END;
```

An dieser Struktur ist zu erkennen, daß jeder Baum auch als Binärbaum dargestellt werden kann, wobei der linke Ast eines Knotens auf den ersten Nachfolger (ältester Sohn) und der rechte auf den nächsten Nachfolger seines Vorgängers (nächstjüngerer Bruder) zeigt.

Die gesamte Vererbungsgeschichte einer Familie läßt sich allerdings nicht als Baum darstellen (man denke nur an das Inzuchtverhalten europäischer Königshäuser), dafür verwendet man erweiterte Strukturen, Graphen.

4.6.5 Graphen

Graphen bilden eine Obergruppe über Bäume und Listen. Ein Graph besteht aus *Knoten* und *Kanten* (Verbindung zwischen zwei Knoten), wobei die Zahl der Kanten, die in einen Knoten hineinführen (Eingangsgrad), und die Zahl der herausführenden (Ausgangsgrad) nicht beschränkt sind. Es gibt auch Graphen, in denen die Kanten keine Richtung haben, man spricht dann von *ungerichteten* Graphen, im Gegensatz zu *gerichteten* Graphen, zu denen Listen und Bäume gehören. Es ist auch möglich, daß ein Knoten eine Kante mit sich selbst, oder mehrere mit dem selben Nachbarn gemeinsam hat. Es ist noch nicht einmal nötig, daß alle Knoten miteinander verbunden sind. Derartige Graphen, in denen *Knotenmengen* existieren, von denen kein Knoten eine Verbindung zu einem Knoten aus einer anderen Menge hat, nennt man *unzusammenhängend.*

Um einen gerichteten Graphen aufzubauen und zu bearbeiten, existiert ein generisches Standardmodul `Graphs`.

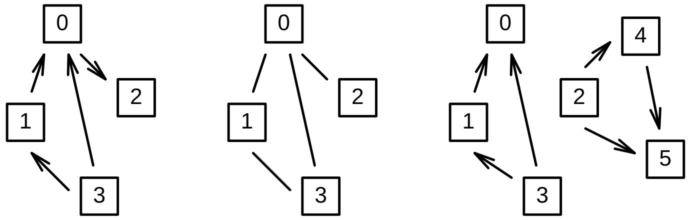
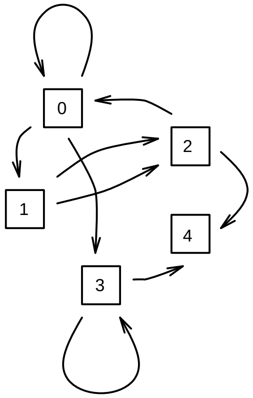


Abbildung 4.22: Verschiedene Graphen

Ist die Zahl der Knoten eines Graphen bekannt, kann man diese durchnumerieren, und die Kanten in einem zweidimensionalen Array ganzer Zahlen darstellen, wobei der Zahlenwert eines Elements des Arrays die Anzahl der Kanten vom Knoten mit der Nummer des ersten Index zu dem Knoten mit der Nummer des zweiten Index angibt. Das bedeutet, die Zahl der Kanten vom Knoten mit der Nummer 2 zum Knoten mit Nummer 3 befindet sich im Arrayelement [2,3].

```
TYPE
  Graph  = ARRAY [0..9],[0..9] OF SHORTCARD;
```



		nach				
		0	1	2	3	4
von	0	1	1	0	1	0
	1	0	0	2	0	0
	2	1	0	0	0	1
	3	0	0	0	1	1
	4	0	0	0	0	0

Abbildung 4.23: Zweidimensionales Array

Da eine derartige Struktur Speicher im Verhältnis zum Quadrat der Knotenzahl benötigt, ist sie allerdings für eine größere Zahl von Knoten nicht zu empfehlen. Hier bietet sich eine dynamische Struktur an, in der zu jedem Knoten die Kanten bzw. die Nummern deren Zielknoten in einer Liste gehalten werden. Eine derartige Listenstruktur nennt sich Adjazensliste.


```
TYPE
  KantePtr = POINTER TO Kante;
  Kante    = RECORD
               next : KantePtr;
               nach : INTEGER
             END;
  Knoten   = ARRAY [0..9] OF KantePtr;
```

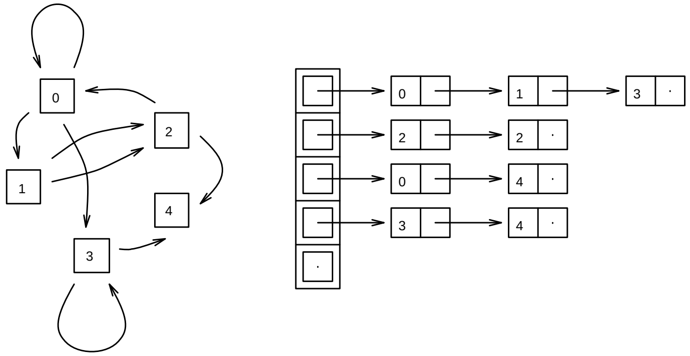


Abbildung 4.24: Adjazensliste

Ist die Zahl der Knoten nicht bekannt, müssen diese ebenfalls in einer Liste gehalten werden.

```
TYPE
  KnotenPtr = POINTER TO Knoten;
  KantePtr  = POINTER TO Kante;
  Kante     = RECORD
                next : KantePtr;
                nach : KnotenPtr
              END;
  Knoten    = RECORD
                kanten : KantePtr
              END;
```

Ist die Zahl der Kanten im Vergleich zu der der Knoten gering, empfiehlt sich eine andere Struktur, in der nur die Kanten in einer Liste gehalten werden.

```
TYPE
  KantenPtr = POINTER TO Kante;
  Kante     = RECORD
                next  : KantenPtr;
                von,
                nach  : INTEGER
              END;
```

Einfügen und Entfernen von Knoten und Kanten wird genauso wie bei Listen realisiert, es lassen sich auch komplexere Listen für die Verwaltung der Elemente verwenden.

4.6.6 Optimierung durch eigene Speicherverwaltung

Die Prozeduren `New`, `Dispose` und die darin benutzten Systemroutinen `AllocMem` und `FreeMem` sind nicht besonders schnell, da immer ein optimales Speicherfragment gesucht wird. Dies kann gerade bei „perforiertem“ Speicher relativ lange dauern. Benötigt man nun in einem Programm eine Struktur, die sich häufig ändert, ist es sinnvoll, die Verwaltung freier Knoten in die eigene Hand zu nehmen. Bei dieser Technik führt das Programm neben den normalen Strukturen noch eine Liste mit freien Knoten. Dieser wird ein freizugebender Knoten angehängt und bei Bedarf wieder entnommen. Nur wenn diese Liste leer ist, wird weiterer Systemspeicher angefordert. Dies geschieht am besten in größeren Paketen, da so die Anzahl der Anforderungen an das System nochmal vermindert und der Speicherperforierung vorgebeugt wird. Zur Verkettung der freien Knoten empfiehlt es sich einen Zeiger zu verwenden, der schon darin enthalten ist.


```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
            left,
            right  : Tree
          END;
VAR
  firstFree : Tree;

PROCEDURE NewRoot(VAR tree : Tree);
VAR mem : POINTER TO ARRAY [0..31] OF Root;
    i   : INTEGER;
BEGIN
  IF firstFree=NIL THEN
    New(mem);
    FOR i:=0 TO 30 DO
      mem^[i].left:=mem^[i+1]'PTR;
    END;
    mem^[31].left:=NIL;
    firstFree:=mem^[0]'PTR;
  END;
  tree:=firstFree;
  firstFree:=firstFree^.left;
END NewRoot;

PROCEDURE DisposeRoot(VAR tree : Tree);
BEGIN
  tree^.left:=firstFree;
  firstFree:=tree;
  tree:=NIL;
END DisposeRoot;

BEGIN
  firstFree:=NIL;
  ...
```

Diese Methode arbeitet wesentlich schneller als die normale Verwendung von `New` und `Dispose`. Allerdings ist zu bedenken, da der Speicher, der hier alloziert wird, nicht mehr freigegeben wird. Um dem zu entgehen, sollte man die belegten Blöcke verketten, und am Ende der Benutzung wieder freigeben.

4.6.7 Ausgefallene dynamische Strukturen

Angenommen Sie benötigen ein zweidimensionales Array mit Longreal-Zahlen und etwas exorbitanter Größe, sagen wir mal $1000 * 1000$. Viele Amigas wären mit einem Feld von 8MByte Größe leicht überfordert. Ist aber bekannt, daß das Array nur relativ locker gefüllt ist, z. B. nur 2000 Werte, empfiehlt sich eine andere Struktur.

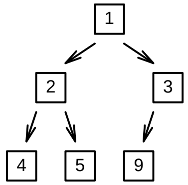
```
TYPE
  MatElemPtr = POINTER TO MatElem;
  MatElem    = RECORD
                 down,
                 right  : MatElemPtr;
                 row,
                 column : INTEGER;
                 data   : LONGREAL;
               END;
  Matrix     = RECORD
                 left,
                 top   : ARRAY [0..999] OF MatElemPtr;
               END;
```

Mit dieser Struktur ist es zwar nicht möglich, sofort von den Indizes auf das Element zu schließen, es ist aber sehr schnell möglich, die von Null verschiedenen Felder des Arrays durchzugehen. Aber gerade dies wird ja beim Rechnen mit Matrizen benötigt, da diese als einzige das Ergebnis beeinflussen. Für ein solches Problem bietet sich das generische Standardmodul `DynamicArrays` an.

4.6.8 Repräsentierung dynamischer Strukturen in einem Array

Manchmal ist nicht so sehr die variable Größe einer dynamischen Struktur nötig, sondern nur deren Flexibilität. In so einem Fall kann eine Struktur wie zum Beispiel ein Baum auch in einem Array gehalten werden.

```
TYPE
  Tree = ARRAY [0..9] OF
           RECORD
             left,
             right : INTEGER;
             data  : INTEGER;
           END;
```



left	1	2	-	-	5	-				
right	2	3	-	-	-	-				
data	1	2	4	5	3	9				

- ist ein ungültiger Index,
z.B.: -1

Abbildung 4.25: Ein Baum in einem Array

Diese Technik hat zwei Vorteile, erstens wird der Speicher durch häufiges Allozieren kleiner Speicherstücke nicht zum Emmentaler zergliedert, und zweitens ist der Speicherbedarf geringer, da ein Zeiger vier Bytes, eine Integer-Zahl dagegen nur zwei benötigt. Andererseits ist die maximale Größe einer derartigen Struktur durch das Array vorgegeben.

Auch die Adjazenzliste eines Graphen läßt sich recht geschickt in einem Array unterbringen.

```
TYPE
  List = ARRAY [0..20] OF INTEGER;
```

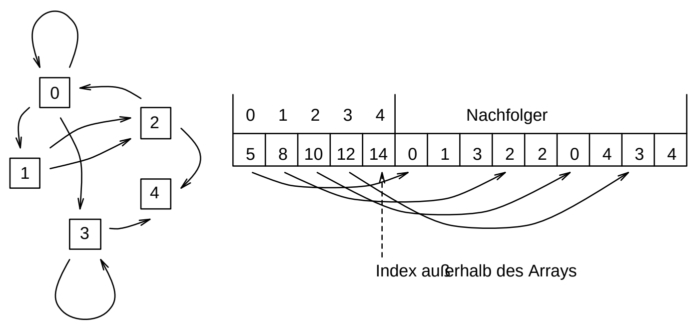


Abbildung 4.26: Eindimensionales Array

Bei einem Graphen mit n Knoten werden in den ersten n Arrayelementen die Anfänge der Kantenlisten eingetragen. Die Kantenlisten enthalten nacheinander die Nummern der Knoten, mit denen der entsprechende Knoten verbunden ist. Diese Struktur eignet sich vor allem für konstante Graphen, da sie nur sehr aufwendig zu erweitern ist.

4.6.9 Dateien als dynamische Strukturen

Auch eine *Datei* stellt eine dynamische Struktur dar, da ihre Größe fast unbeschränkt ist (besonders auf Festplatten). Man unterscheidet zwei grundlegend verschiedene Arten von Dateien. *Sequentielle* Dateien, sogennante *Ströme* (Streams), in die ein Element nach dem anderen geschrieben und gelesen wird und Dateien mit *freiem* Zugriff, in die beliebig geschrieben und gelesen wird.

Streams werden über `InOut` realisiert. Das Öffnen eines Eingabestroms geschieht über `OpenInput`, und eines Ausgabestroms über `OpenOutput`. Danach kann über die bekannten Write- und Read-Prozeduren auf den

Stream zugegriffen werden. Dem Modul `InOut` gegenüber ist sogar der Benutzer ein (bzw. zwei) Stream(s).

Dateien mit freiem Zugriff, meist nur Dateien genannt, werden über das FileSystem realisiert. Mit den Proceduren `SetPos` und `GetPos` kann beliebig in der Datei positioniert werden, und über `Write..` und `Read..` kann auch überall gelesen und geschrieben werden.

Aus- und Eingaben eines Programms werden im allgemeinen als Streams realisiert, z. B. ist der Quelltext der Eingabe- und das Objektfile der Ausgabestream eines Compilers. Das folgende Programm summiert immer zwei Zahlen eines Eingabestreams auf, und gibt die Summe auf den Ausgabestream aus.

```
MODULE Streams;

IMPORT InOut AS io;

VAR a,b : INTEGER;

BEGIN
  IF  io.OpenInput("Eingabe")
  AND io.OpenOutput("Ausgabe") THEN
    io.ReadInt(a);
    WHILE io.done DO
      io.ReadInt(b);
      io.WriteInt(a+b);
      io.ReadInt(a);
    END;
    io.CloseOutput;
    io.CloseInput;
  ELSE
    RAISE(NoStreamsError);
  END;
CLOSE
END Streams.
```

Dagegen werden Dateien, die ein Programm zur eigenen Benutzung anlegt, meist mit freiem Zugriff benutzt. Dazu wird die Datei in *Sätze* gleicher Länge zerteilt. Eine derartige Datei ist mit einem Array vergleichbar. Der Zugriff erfolgt über die *Satznummer*. Diese wird, um auf die echte Position in der Datei zu kommen, mit der *Satzlänge* multipliziert.

```
TYPE
  Satz   = RECORD
             Name,
             Vorname : STRING(40)
           END;
VAR f : File;

PROCEDURE ReadRecord(VAR satz : Satz;No : INTEGER);
BEGIN
  SetPos(f,No*Satz'SIZE);
  ReadBlock(f,satz);
END ReadRecord;
```

Es empfiehlt sich allerdings, um die Flexibilität zu erhöhen, dieses Array wie eine Liste zu führen (siehe 4.6.8).

4.6.10 Queue und Stack

Keine eigentliche dynamische Struktur, sondern mehr eine Anwendung von Listen, sind *Queues* (Warteschlangen) und *Stacks* (Kellerspeicher). Sie dienen als Puffer (Zwischenspeicher); in ihnen werden Informationen für eine Weile gehalten, um später wieder verwendet zu werden. Queues und Stacks haben zwei Funktionen, Information ablegen und Information wieder zurückgeben. Eine Queue gibt dabei die Information wieder in der Reihenfolge ab, in der diese übergeben wurden (first in first out, FIFO). Die typische Anwendung ist z. B. eine Tastaturwarteschlange, die von der Tastatur dauernd mit Zeichen gefüllt und auf der anderen

Seite vom Programm wieder geleert wird. Ein Stack dagegen gibt die zuletzt übergebenen Daten als erste wieder zurück (last in first out, LIFO). Auf einem derartigen Stack hält sich der Prozessor zum Beispiel die Rückkehradressen bei Prozeduraufrufen. Stacks und Queues werden entweder als Arrays oder als Listen realisiert.

```
TYPE
  Stack  = RECORD
             last  : INTEGER;
             data  : ARRAY [0..99] OF LONGINT
           END;
VAR
  s : Stack;

(* ein Element auf Stapel legen *)
PROCEDURE Push(VAR s : Stack; data : LONGINT);
BEGIN
  s.data[s.last]:=data;
  INC(s.last);
END Push;

(* Ein Element wieder herunter holen *)
PROCEDURE Pop(VAR s : Stack):LONGINT;
BEGIN
  DEC(s.last);
  RETURN s.data[s.last]
END Pop;

(* Testen, ob Stapel leer ist *)
PROCEDURE Empty(VAR s : Stack):BOOLEAN;
BEGIN
  RETURN s.last=0
END Empty;

BEGIN
```

```
  s.last:=0;
  ...
```

Nun dasselbe mit einer Liste:

```
TYPE
  Stack     = POINTER TO StackElem;
  StackElem = RECORD
                next : Stack;
                data : LONGINT
              END;
VAR
  s : Stack:=NIL;

PROCEDURE Push(VAR s : Stack; data : LONGINT);
VAR p : Stack;
BEGIN
  New(p);
  p^.next:=s;
  p^.data:=data;
  s:=p;
END Push;

PROCEDURE Pop(VAR s : Stack):LONGINT;
VAR p : Stack;
    l : LONGINT;
BEGIN
  p:=s;
  s:=s^.next;
  l:=p^.data;
  Dispose(p);
  RETURN l
END Pop;

PROCEDURE Empty(s : Stack):BOOLEAN;
```



```
BEGIN
  RETURN s=NIL;
END Empty;
```

Eine Queue ist etwas schwerer zu realisieren, da sowohl auf das letzte als auch das erste Element zugegriffen werden muß. Eine Queueliste wird der Einfachheit halber meist als Ringliste geführt.

```
TYPE
  Queue    = POINTER TO QueueElem;
  QueueElem= RECORD
               next : Queue;
               data : LONGINT
             END;
VAR
  q : Queue;

PROCEDURE Write(VAR q : Queue; data : LONGINT);
VAR p : Queue;
BEGIN
  New(p);
  p^.data:=data;
  IF q=NIL THEN
    p^.next:=q
  ELSE
    p^.next:=q^.next;
    q^.next:=p
  END;
  q:=p;
END Write;

PROCEDURE Read(VAR q : Queue):LONGINT;
VAR p : QueuePtr;
    l : LONGINT;
BEGIN
```

```
    p:=q^.next;
    l:=p^.data;
    IF p=q THEN
      q:=NIL
    ELSE
      q^.next:=p^.next
    END;
    Dispose(p);
    RETURN l
  END Read;

  PROCEDURE Empty(q : Queue):BOOLEAN;
  BEGIN
    RETURN q=NIL
  END Empty;

  BEGIN
    q:=NIL;
    ...
```

Für Queues und Stacks existiert das generische Modul `Buffers`, so daß Sie wahrscheinlich nie selbst die nötigen Routinen programmieren müssen.

4.6.11 Abspeichern dynamischer Strukturen in Dateien

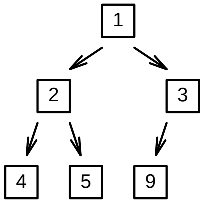
Es ist nicht besonders einfach, dynamische Strukturen abzuspeichern und wieder einzuladen, da man in einem System mit dynamischer Speicherverwaltung – und der Amiga ist so ein System – nie weiß, welchen Speicher einem das Betriebssystem zur Verfügung stellt. Somit sind abgespeicherte Zeiger völlig unbrauchbar, da sie beim Einladen bestimmt nicht dahin zeigen, wo die bezeichnete Struktur jetzt liegt.

Am besten ist es, die Strukturen beim Einlesen wieder neu aufzubauen. Am einfachsten ist dies noch bei Listen, da einfach nur Element

für Element gesichert und nachher wieder in der selben Reihenfolge eingehängt werden muß. Problematischer sind da schon Bäume. Am geschicktesten ist hier wohl Präfix, da einem Knoten recht einfach angesehen werden kann, wie viele Nachfolger er hat (Das Pointerziel `NIL` ist so einzigartig, daß es immer gleich bleibt). Da eine in einer Datei gespeicherte Pointer-Adresse unbrauchbar ist, reicht es vollkommen aus, sich zu merken, ob ein Pointer `NIL` ist oder nicht.

```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
            left,
            right : Tree;
            data  : ...
          END;
PROCEDURE WriteTree(f : File; t : Tree);
BEGIN
  IF t#NIL THEN
    WriteBlock(f,t^);
    WriteTree(f,t^.left);
    WriteTree(f,t^.right);
  END;
END WriteTree;

PROCEDURE ReadTree(f : File; VAR t : Tree);
BEGIN
  New(t);
  ReadBlock(f,t);
  IF t^.left#NIL THEN
    ReadTree(f,t^.left)
  END;
  IF t^.right#NIL THEN
    ReadTree(f,t^.right)
  END;
END ReadTree;
```



left	#	#	0	0	#	0		
right	#	#	0	0	0	0	EOF	
data	1	2	4	5	3	9		

EOF = Ende der Datei
0 entspricht NIL
ist ungleich NIL

Abbildung 4.27: Ein Baum in einem File

Bei einem Graphen empfiehlt es sich, die Kanten abzuspeichern, und die Knoten mit Nummern zu bezeichnen.

Eine dynamische Struktur in einem Array macht keinerlei Probleme, da sich die Indizes der Elemente, die als Zeiger fungieren, nicht verschieben können.

4.7 Laufzeitverhalten von Programmen

Ein wichtiges Kriterium für die Güte eines Algorithmus ist seine Geschwindigkeit. Hierbei ist allerdings nicht so sehr die echte Ausführungszeit von Bedeutung, da diese von zuvielen Faktoren abhängt, sondern sein Verhalten in Relation zur Eingabelänge. Unter der Eingabelänge versteht man zum Beispiel die Größe eines Arrays, mit dem der Algorithmus arbeitet, oder die Zahl der Sätze einer Datei. Nun ist es im allgemeinen fast unmöglich, die durchschnittliche Laufzeit eines Programmes zu errechnen, da meist zuviele verschiedene Möglichkeiten existieren, wie das Programm verläuft. Man beschränkt sich deshalb auf den schlechtesten Fall (worst case), und gibt auch nur eine Proportionalität zu einer Funktion der Eingabe an. Diese Funktion nennt man O, mit $O(n)$ sind zum Beispiel alle Algorithmen bezeichnet, deren Laufzeit gleichmäßig mit der Eingabelänge wächst.

Ein Programm, das keine Schleifen und Rekursionen enthält, ist in seiner Laufzeit von der Eingabe unabhängig, und hat $O(1)$.

Die Schleife:

```
FOR i:=1 TO n DO
  xxx
END
```

hat eine Laufzeit von $O(n*m)$, wenn `xxx` $O(m)$ hat;

```
FOR i:=1 TO n DO
  FOR j:=1 TO i DO
    ...
  END
END
```

hat $O(n^2)$, genau $O(n*(n+1)/2) = O(n*n)$.

Die folgende Tabelle zeigt wie sich verschiedene Funktionen bei wachsender Eingabelänge verhalten.

	5	10	20	1000	1E6	1E9
$\ln(\ln(n))$	0.5	0.8	1.1	1.9	2.6	3.0
$\ln(n)$	1.6	2.3	3.0	6.9	13.8	20.7
n	5.	10.	20.	1000.	1E6	1E9
$n\ln(n)$	8.	23.	60.	6900.	13E6	20E9
n^2	25.	100.	400.	1E6	1E12	1E18
n^3	125.	1000.	8000.	1E9	1E18	1E27
2^n	32.	1024.	1E6	1E301	1E301029	...

Es zeigt sich deutlich, daß exponentielle Algorithmen im allgemeinen abzulehnen sind, da die Laufzeit mit wachsender Länge ins Unerträgliche ansteigt. (Es gibt einen eigenen Zweig der Informatik, der sich mit dem

Problem beschäftigt, für jedes Problem einen nichtexponentiellen Algorithmus zu finden, oder zu beweisen, daß es keinen gibt. Gibt es keinen nichtexponentiellen Algorithmus, so bezeichnet man das Problem NP-Hard.)

4.8 Sortieralgorithmen

Das Sortieren von Objekten ist eine häufige Tätigkeit für Programme (und ein Steckenpferd vieler Informatiker). Es gibt eine ganze Reihe von Verfahren, die im Laufe der letzten Jahrzehnte entwickelt wurden. Manche wirken auf den ersten Blick völlig unverständlich, andere überzeugen durch ein schlechtes Laufzeitverhalten. Normale Hobbyprogrammierer beschränken sich meist auf das Sortierverfahren, das das schlechteste Laufzeitverhältnis hat, den Bubblesort. Im allgemeinen beschränkt sich das Sortieren auf Arrays, nur selten ist mal eine Liste fällig, da diese meist schon sortiert erstellt werden.

4.8.1 Bubblesort

```
...
FOR i:=1 TO n DO
  FOR j:=1 TO n-1 DO
    IF feld[i]>feld[i+1] THEN
      x        :=feld[i];
      feld[i]  :=feld[i+1];
      feld[i+1]:=x
    END;
  END;
END;
...
```

Das wohl bekannteste (und langsamste) aller Sortierverfahren. Der Grundgedanke dahinter ist, daß ein Feld dann sortiert ist, wenn jedes Element kleiner als sein Nachfolger ist. Es werden solange Nachbarn die diese Bedingung nicht erfüllen, miteinander vertauscht, bis das Feld sortiert ist. Es läßt sich beweisen, daß dies nach maximal n Vertauschungen der Fall ist (n ist die Zahl der Elemente). Die Laufzeit ist immer $O(n^2)$.

4.8.2 Insert & Select

Diese beiden Algorithmen benutzen ein anderes Verfahren. Sie teilen das Feld in zwei Teile, ein linkes, das bereits sortiert ist und ein rechtes, das die noch zu sortierenden enthält. Am Anfang ist der linke Teil noch ohne Elemente. Bei jedem Schritt wird nun ein Element aus der rechten Menge genommen, und so in die linke eingefügt, daß deren Sortierung erhalten bleibt. Dies kann auf zwei Arten durchgeführt werden. *Insert* nimmt immer das linkeste Element der rechten Menge und fügt es an der richtigen Stelle in den linken Teil ein. *Select* dagegen wählt immer das kleinste Element aus der rechten Menge aus, und hängt es ganz rechts an die linke an. Dies wird solange durchgeführt, bis die rechte Menge leer ist, was nach genau n Schritten der Fall ist.

Select:

```
...
FOR i:=1 TO n-1 DO
  j:=i;
  FOR k:=i+1 TO n DO
    IF feld[j]<feld[k] THEN
      j:=k
    END;
  END;
  x:=feld[i];feld[i]:=feld[j];feld[j]:=x
END;
...
```

Insert:

```
...
FOR i:=2 TO n DO
  x:=feld[i];
  j:=i;
  WHILE (j>1) AND (feld[j-1]>x) DO
```



```
    feld[j]:=feld[j-1];
    DEC(j)
  END;
  feld[j]:=x;
END;
```

Zwar haben auch diese Algorithmen eine Laufzeit von $O(n^2)$, doch sind sie in der Praxis meist doppelt so schnell wie Bubblesort, wobei sich `Select` mehr für Arrays und *Insert* für Listen empfiehlt.

4.8.3 Quicksort

Quicksort ist das im allgemeinen schnellste Verfahren, ein Feld zu sortieren. Im Schnitt braucht er $O(n \log(n))$. Er hat allerdings auch einen worst case, bei dem er $O(n^2)$ hat. Das Prinzip von Quicksort ist es, das Feld in zwei möglichst gleich große Teile zu zerlegen, so daß alle Elemente in der linken Hälfte kleiner sind als das kleinste Element aus der rechten Menge. Danach werden diese Teilmengen getrennt sortiert.

```
PROCEDURE Partition(left,right : INTEGER);
VAR m,i   : INTEGER;
    pivot : INTEGER;
BEGIN
  IF right>left THEN
    m:=left;
    pivot:=feld[m];
    FOR i:=m+1 TO right DO
      IF feld[i]<pivot THEN
        feld[m]:=feld[i];
        INC(m);
        feld[i]:=feld[m]
      END;
    END;
    feld[m]:=pivot;
```

```
      Partition(left,m-1);
      Partition(m+1,right);
    END;
  END Partition;

  ...
  Partition(feld'MIN,feld'MAX);
  ...
```

Das Feld wird in zwei Teile zerlegt, indem man sich einen beliebigen Grenzwert (Pivot) aus dem Feld nimmt. Danach sortiert man das Feld so um, daß alle Elemente, die kleiner als der Pivot sind, links von ihm, alle größeren rechts landen. Für diese Partitionierung gibt es die verschiedensten Techniken, alle haben aber eine Laufzeit von $O(n)$. Da das Feld immer in zwei Teile geteilt wird, ist die Anzahl der nötigen Partitionierungen im allgemeinen $\log(n)$, die Gesamtlaufzeit somit $O(n\log(n))$. Erwischt man allerdings als Grenzwert immer das kleinste oder größte Element, wird die Partitionierung n mal durchgeführt.

4.8.4 Heapsort

Ein anderes Sortierverfahren mit $O(n\log(n))$ ist Heapsort. Dieses Verfahren besitzt zwar keinen worst case, ist aber aufgrund etwas größeren Aufwands bei kleineren Feldern langsamer als Quicksort. Heapsort ist das vermutlich unter Programmierern meistgehaßte Sortierverfahren, da es ohne intime Kentnisse der Materie völlig unverständlich wirkt. Es scheint so, als würde das Programm das Feld absolut chaotisch durcheinanderwirbeln, um dann völlig unvermutet das sortierte Feld aus dem Hut zu ziehen. Heapsort verwendet eine seltsame Struktur, einen sogenannten Heap. Dies ist ein Baum, bei dem jeder Knoten maximal zwei Äste hat. Für jeden Knoten gilt, daß sein Wert größer ist als die Werte seiner Nachfolger.

Ein Heap läßt sich sehr geschickt in einem Array unterbringen, in dem man festlegt, daß die Nachfolger des Feldes i die Felder $2i$ und $2i+1$

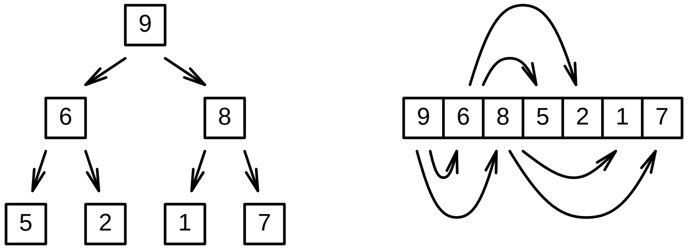


Abbildung 4.28: Heap in einem Array

sind. Legt man weiter fest, daß neue Blätter immer von links nach rechts angefügt werden, ist das Array ohne Lücken gefüllt.

Mit einem Heap kann man zwei Dinge tun, man kann ihm ein neues Element anfügen, oder man kann das größte Element des Heaps entnehmen. Bei diesen Aktionen ändern sich dessen Eigenschaften nicht. Angefügt wird immer an einem der Blätter. Dabei muß beachtet werden, daß der übergeordnete Knoten den höchsten Wert hat. Ist dies nicht der Fall, muß entsprechend getauscht werden. Dabei kann es vorkommen, daß der nach oben gewanderte Wert auch größer ist als der nächste Übergeordnete. Dies setzt sich solange fort, bis die Struktur wieder stimmt. Da dabei ein Wert nach oben steigen kann, nennt man dies Steigen.

```
VAR
  feld : ARRAY [1..n]OF INTEGER;
  heap : INTEGER; (* Anzahl der Elemente im Heap *)

PROCEDURE Erweitern;
VAR x,k,i : INTEGER;
BEGIN
  INC(heap);
  k:=heap;
  i:=k DIV 2;
```

```
    WHILE (k>1) AND (feld[k]>feld[i]) DO
      x:=feld[i]; feld[i]:=feld[k]; feld[k]:=x;
      k:=i;
      i:=k DIV 2;
    END;
  END Erweitern;
```

Nimmt man das größte Element aus dem Heap, bringt man das letzte Element aus dem Array an seine Position, und läßt es durchsinken. Dadurch bleibt der linke Teil des Arrays immer völlig durch den Heap gefüllt.

```
  PROCEDURE Entfernen;
  VAR x,k,i : INTEGER;
  BEGIN
    x:=feld[1]; feld[1]:=feld[heap]; feld[heap]:=x;
    DEC(heap);
    k:=1;
    WHILE k*2<=heap DO
      i:=k*2;
      IF (i<heap) AND (feld[i+1]>feld[i]) THEN
        INC(i)
      END;
      IF feld[i]>feld[k]THEN
        x:=feld[i]; feld[i]:=feld[k]; feld[k]:=x
      END;
      k:=i;
    END
  END Entfernen;
```

Das Einfügen oder Entfernen eines Elements dauert immer $O(\log(n))$. Zum Sortieren werden nun alle Elemente nacheinander in den Heap eingefügt, um dann der Größe nach entnommen zu werden.


```
  ...
  heap:=0;
  WHILE heap<n DO
    Erweitern
  END;
  WHILE heap>0 DO
    Entfernen
  END;
  ...
```

4.8.5 Mergesort

Mergesort stammt noch aus der guten alten Zeit der Lochkarten und Bänder, als man ohne großen Speicher zu besitzten (1960 hatte ein Großrechner noch stolze 4KByte) riesige Datenbestände sortieren mußte. Alles was man hatte waren Ein- und Ausgabeströme, meist Lochkartenleser, aber keine Möglichkeit größere Datenbestände im Speicher zu halten. Normale Sortiermethoden waren einfach undenkbar. Das Verfahren arbeitet um so besser, je mehr Eingabeströme vorhanden sind. Als Beispiel reicht es aber, die Technik mit nur zwei Eingabeströmen zu zeigen, da das Verfahren dabei schon ersichtlich wird. Die Eingabeschlange wird in zwei etwa gleich große Teilstücke zerlegt. Diese werden den beiden Lesegeräten zugeführt. Dann wird solange das kleinere Element der beiden Eingaben ausgegeben, bis dieses kleiner ist als das zuletzt ausgegebene. Danach werden noch alle Elemente der anderen Eingabe ausgegeben, solange sie in aufsteigender Reihenfolge vorliegen. Nach einem Durchlauf hat man eine Reihe von Stapeln, die alle in sich aufsteigend sortiert sind. Diese werden nun möglichst gerecht auf beide Leser verteilt, und das Programm wieder gestartet. Dies wird solange wiederholt, bis nur noch ein Stapel vorhanden ist. Diese Prozedur muß $\log(n)$ mal durchgeführt werden, da immer mindestens zwei Stapel zu einem neuen verschmolzen werden. Das Laufzeitverhalten ist also $O(n \log(n))$. Der Haken ist nur, daß nicht in einem Array gearbeitet werden kann, sondern mindestens zwei vorhanden sein müssen.

```
...
LOOP
  o:=INTEGER'MIN;
  Read(1,e1); Read(2,e2);
  WHILE (e1>=o) AND (e2>=o) DO
    IF e1<e2 THEN
      Write(e1);
      IF Eof(1) THEN EXIT END;
      o:=e1;
      Read(1,e1);
    ELSE
      Write(e2);
      IF Eof(2) THEN EXIT END;
      o:=e2;
      Read(2,e2);
    END;
  END; | WHILE
  IF e1>=o THEN
    WHILE e1>=o DO
      Write(e1);
      IF Eof(1) THEN EXIT END;
      o:=e1;
      Read(1,e1);
    END;
  ELSE
    WHILE e2>=o DO
      Write(e2);
      IF Eof(2) THEN EXIT END;
      o:=e2;
      Read(2,e2);
    END;
  END;
END;| LOOP
IF Eof(2) THEN
  Write(e1);
```



```
    WHILE NOT Eof(1) DO
      Read(1,e1);Write(e1);
    END;
  ELSE
    Write(e2);
    WHILE NOT Eof(2) DO
      Read(2,e2);Write(e2);
    END;
  END;
  ...
```

4.8.6 Radixsort

Radixsort ist die älteste und wohl ausgefallenste Art, ein Feld zu sortieren. Dieser Algorithmus stammt aus der Zeit mechanischer Lochkartenverarbeitung. Maschinen, die nach dieser Methode arbeiten, waren noch bis in die achtziger Jahre im Gebrauch. Sie besteht aus einem Eingabestapel und zehn Ausgabestapeln. Bei der Maschine wird eine Spalte der Karten eingestellt. Danach werden die Karten über ein Rollensystem über die zehn Fächer geführt, und fallen in das Fach, dessen Nummer in dieser Spalte auf der Karte steht. Mit diesem Verfahren lassen sich also Zahlen nach einer Ziffer sortieren. Sorgt man nun noch dafür, daß die Reihenfolge der Karten in den Sortierfächern sich nicht von der ursprünglichen Reihenfolge unterscheidet, kann man in sovielen Durchläufen, wie die Zahlen Ziffern hat, sortieren. Zuerst sortiert man nach der letzten Ziffer, dann legt man alle Stapel wieder aufeinander, und sortiert nach der vorigenen Ziffer. Dies wird fortgeführt, bis zur ersten. Danach ist der Stapel sortiert. Radixsort wird heute kaum noch angewendet, kann aber in einigen Spezialfällen durchaus noch nützlich sein. Ich möchte hier auf ein Beispiel verzichten, da der Aufwand das Ergebnis nicht lohnt. Das Verfahren sei nur der Vollständigkeit halber erwähnt.

4.9 Suchalgorithmen

Neben dem Sortieren spielt das Suchen bei vielen Programmen eine große Rolle. Für manche Suchtechniken müssen die Daten in speziellen Strukturen vorhanden sein, andere laufen mit beliebigen Strukturen. Im allgemeinen sind Suchalgorithmen mit speziellen Strukturen wesentlich effektiver, so daß es sich empfiehlt, Datensammlungen, in denen besonders oft gesucht wird, gleich in derartigen Strukturen abzulegen.

4.9.1 Lineares Suchen

```
TYPE
  List = POINTER TO Node;
  Node = RECORD
           next : List;
           data : INTEGER;
         END;

PROCEDURE Search(list : List; data : INTEGER) : List;
BEGIN
   WHILE (list#NIL) AND (list^.data#data) DO
     list:=list^.next
   END;
   RETURN list
END Search;
```

Die einfachste aller Möglichkeiten ein Element aus einer Liste oder einem Array herauszusuchen, ist lineares Suchen. Bei diesem Verfahren werden nacheinander alle Werte mit dem zu suchenden verglichen, bis Übereinstimmung eintritt oder kein Element mehr vorhanden ist. Dieses Verfahren sollte nur in Listen und unsortierten Arrays angewendet werden, da es mit $O(n)$ das langsamste Verfahren ist.

Und im Array:


```
PROCEDURE Search(REF feld : ARRAY OF INTEGER;
                     data : INTEGER) : INTEGER;
VAR i : INTEGER;
BEGIN
  i:=feld'MIN;
  WHILE (i<=feld'MAX) AND (feld[i]#data) DO
    INC(i)
  END;
  RETURN i
END Search;
```

4.9.2 Binärsuche

Hat man ein sortiertes Array zur Verfügung, gibt es ein wesentlich besseres Suchverfahren. Es beruht darauf: Nimmt man ein beliebiges Element heraus, kann man sicher sein, daß alle Elemente mit kleinerem Index kleiner und alle anderen größer sind. Nimmt man nun das mittlere Element aus dem Array heraus, kann man mit einem Vergleich schon die Hälfte des Feldes eliminieren. Um also das richtige Element zu finden sind nur $O(\log(n))$ Vergleiche nötig.

```
PROCEDURE BinSearch(REF feld : ARRAY OF INTEGER;
                        data : INTEGER) : INTEGER;
VAR l,r,m : INTEGER;
BEGIN
  l:=feld'MIN;r:=feld'MAX;
  WHILE l#r DO
    m:=(r+l) DIV 2;
    IF feld[m]<data THEN
      l:=m+1
    ELSE
      r:=m;
    END
  END;
  RETURN l;
END BinSearch;
```

4.9.3 Interpolationssuche

Weiß man, daß das Feld gleichmäßig ansteigt, die Werte der Feldelemente über ihrem Index aufgetragen, ziemlich genau eine Gerade ergeben, (z. B. Namen im Telefonbuch) bietet sich ein noch optimaleres Verfahren an.

Das Verfahren läßt sich am besten graphisch verdeutlichen. Man trägt die Werte des Feldes in einem Graphen über ihrem Index auf. Dann konstruiert man eine Gerade durch den ersten und letzten Punkt des Intervalls. Diese Gerade schneidet man mit einer Horizontalen in Höhe des gesuchten Wertes. Danach prüft man, ob der gefundene Wert kleiner oder größer als der Gesuchte ist, und wählt danach die neuen Intervallgrenzen.

Dies wird solange wiederholt, bis das gesuchte Element gefunden ist. Die Laufzeit des Verfahrens ist natürlich abhängig von der Verteilung der Elemente im Feld, es gibt auch einen schlimmsten Fall, in dem n Vergleiche nötig sind, aber bei einigermaßen gleichmäßiger Verteilung ist das Verfahren das schnellste.


```
PROCEDURE Interpolation(feld : ARRAY OF INTEGER;
                        data : INTEGER) : INTEGER;
VAR l,r,m : INTEGER;
BEGIN
  l:=feld'MIN;r:=feld'MAX;
  WHILE feld[l]#feld[r] DO
    m:=(data-feld[l])*(r-l) DIV (feld[r]-feld[l])+l;
    IF feld[m]<data THEN
      l:=m+1
    ELSE
      r:=m
    END;
  END;
  RETURN l;
END Interpolation;
```

4.9.4 Suchbäume

Ist die wichtigste Aufgabe einer Struktur, leichtes Erweitern und schnelles Suchen, empfiehlt sich eine der folgenden. Die einfachsten Strukturen dieser Art sind unter den Bäumen zu finden.

4.9.4.1 Binärbäume

Binärbäume lassen sich als Suchbäume verwenden, wenn man als Kriterium ansetzt, daß der linke Ast nur Knoten enthält die kleiner, der rechte nur solche die größer sind als der aktuelle Knoten.

Das Suchen in einem derartigen Suchbaum ist ähnlich wie die Binärsuche in einem Array. Zuerst wird der Wert mit der Wurzel verglichen. Ist er kleiner, wird als nächstes der linke, ist er größer, der rechte Knoten betrachtet. Die Suche ist dann beendet, wenn der Vergleich ergibt, daß ein Knoten der gesuchte ist.

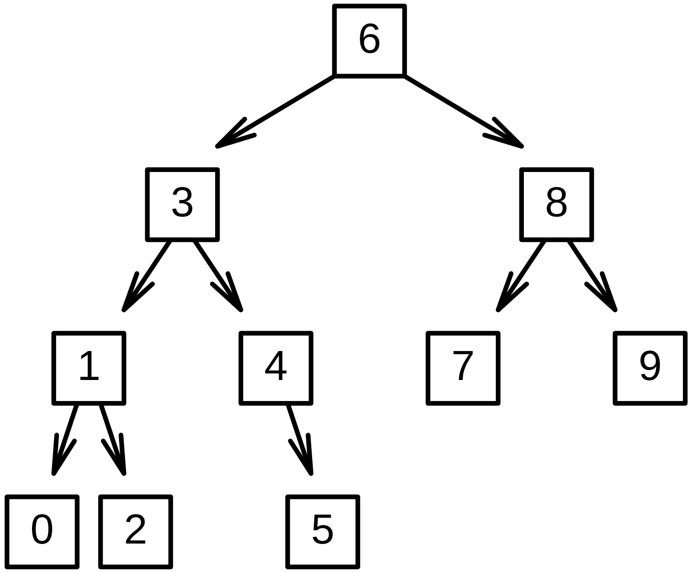


Abbildung 4.29: Binärbaum


```
TYPE
  Tree = POINTER TO Root;
  Root = RECORD
           left,
           right  : Tree;
           data   : INTEGER
         END;

PROCEDURE Search(tree : Tree; data : INTEGER):tree;
BEGIN
  WHILE tree#NIL
    AND_WHILE tree^.data>data DO
      tree:=tree^.left
    OR_WHILE tree^.data<data DO
      tree:=tree^.right
    END
  ELSE
    RETURN tree
  END;
END Search;
```

Die Suche ist dann erfolglos, wenn `NIL` zurückgegeben wird.

Einfügen in einen Suchbaum läuft ähnlich ab, wie das Suchen eines Elementes, nur wird dabei, wenn das Element nicht schon im Baum vorhanden ist, dieses an das Blatt angehängt, das als letztes besucht wurde.

```
PROCEDURE Insert(VAR tree : Tree; data : INTEGER);
VAR p,q : Tree;
BEGIN
  New(p);
  p^.data:=data;
  p^.left:=NIL;
  p^.right:=NIL;
```

```
    IF tree=NIL THEN
      tree:=p;
    ELSE
      q:=tree;
      WHILE q^.data>data
        AND_WHILE q^.left#NIL DO
          q:=q^.left
        ELSE
          q^.left:=p
        END
      OR_WHILE q^.data<data
        AND_WHILE q^.right#NIL DO
          q:=q^.right
        ELSE
          q^.right:=p
        END
      ELSE
        RAISE2(KeyExistsError)
      END;
    END;
  END Insert.
```

Die Laufzeit für Einfügen und Suchen beträgt für einigermaßen ausgeglichene Bäume $O(\log(n))$. Ein Baum ist dann ausgeglichen, wenn sich die Anzahl der Knoten im rechten Ast jedes Knotens sich nicht extrem von der des linken Astes unterscheidet. Im schlimmsten Fall hat der Baum nur linke (oder nur rechte) Äste. Die Laufzeit ist dann $O(n)$.

4.9.4.2 AVL-Bäume

Um entartete, also völlig unausgeglichene Bäume zu vermeiden, wurden schon riesige Mengen an Gehirnschmalz aufgewendet. Am bekanntesten und effektivsten sind sicher *AVL-Bäume* (nach den Anfangsbuchstaben

der Entwickler). Um einen Algorithmus zu entwickeln, der nur *ausgeglichene* Bäume produziert, muß erstmal genau festgelegt werden, was unter ausgeglichen zu verstehen ist. Dazu müssen wir die Höhe eines Baumes definieren: Die Höhe eines Baumes ist die Zahl der Knoten auf dem Weg von der Wurzel zum entferntesten Blatt.

Da jeder Knoten auch Wurzel eines neuen Baumes ist, kann die Höhe eines Astes als die Höhe dieses Unterbaums definiert werden. Jetzt können wir festlegen, wann ein Baum ausgeglichen sein soll: Ein Baum ist dann ausgeglichen, wenn sich für jeden Knoten die Höhe seiner Äste maximal um 1 unterscheidet. Die Differenz von linker und rechter Höhe bezeichnet man als Ausgeglichenheits-Faktor.

```
TYPE
  Tree  = POINTER TO Root;
  Root  = RECORD
            left,
            right  : Tree;
            a_fact : INTEGER;
            data   : INTEGER;
          END;
```

Um die A-Faktoren eines Baumes zu berechnen verwendet man folgende Prozedur, die gleichzeitig auch die Höhe des Baumes zurückliefert.

```
PROCEDURE GetAFact(tree : Tree):INTEGER;
VAR l,r : INTEGER;
BEGIN
  IF tree=NIL THEN
    RETURN 0
  ELSE
    l:=GetAFact(tree^.left);
    r:=GetAFact(tree^.right);
    tree^.a_fact:=r-l;
    IF r>l THEN
```

```
        RETURN r+1
      ELSE
        RETURN l+1
      END
    END
  END GetAFact;
```

Ein Baum ist dann *unausgeglichen*, wenn ein A-Faktor nicht -1, 0 oder 1 ist. Um das zu vermeiden, muß man eine veränderte Einfügeprozedur verwenden. Dabei kann eine Ausgleichstätigkeit nötig werden. Es gibt deren vier, wobei jeweils zwei symmetrisch sind.

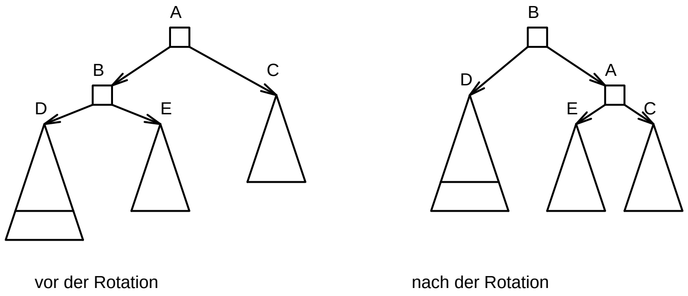


Abbildung 4.30: Einfachrotation links

Da nach diesen *Rotationen* die Höhe des beteiligten Baumteils wieder um eins vermindert wird, ist nur maximal eine Rotation nötig. Ich beschränke mich hier auf die beiden linken Rotationen.

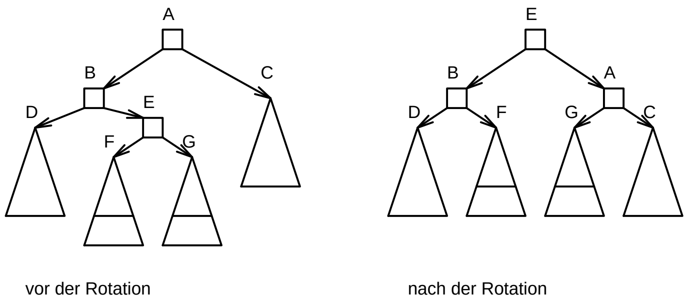


Abbildung 4.31: Doppelrotation links

```
PROCEDURE RotateLeft(VAR tree : Tree);
VAR help : Tree;
BEGIN
  help:=tree;
  tree:=tree^.left;
  help^.left:=tree^.right;
  tree^.right:=help;
  tree^.a_fact:=0;
  help^.a_fact:=0;
END RotateLeft;

PROCEDURE RotateDoubleLeft(VAR tree : Tree);
VAR help : Tree;
BEGIN
  help:=tree;
  tree:=tree^.left^.right;
  help^.left^.right:=tree^.left;
  tree^.left:=help^.left;
  help^.left:=tree^.right;
  tree^.right:=help;
  IF tree^.a_fact=-1 THEN
    tree^.left^.a_fact:=0;
```

```
      tree^.right^.a_fact:=1;
    ELSE
      tree^.left^.a_fact:=-1;
      tree^.right^.a_fact:=0;
    END;
    tree^.a_fact:=0;
  END RotateDoubleLeft;
```

Beim Einfügen wird noch ein Flag zurückgegeben, ob sich die Höhe vergrößert hat.

```
  PROCEDURE Insert(VAR tree : Tree; data : INTEGER);
  BEGIN
    IF tree=NIL THEN
      New(tree);
      tree^.left:=NIL;
      tree^.right:=NIL;
      tree^.data:=data;
      tree^.a_fact:=0;
    ELSE
      IF data<tree^.data THEN
        IF Insert(tree^.left,data) THEN
          DEC(tree^.a_fact);
          IF tree^.a_fact>-2 THEN
            RETURN TRUE
          ELSE
            IF tree^.left^.a_fact=-1 THEN
              RotateLeft(tree);
            ELSE
              RotateDoubleLeft(tree);
            END;
          END;
        END;
      ELSE
        IF Insert(tree^.right,data) THEN
```



```
          INC(tree^.a_fact);
          IF tree^.a_fact<2 THEN
            RETURN TRUE
          ELSE
            IF tree^.right^.a_fact=1 THEN
              RotateRight(tree);
            ELSE
              RotateDoubleRight(tree);
            END;
          END;
        END;
      END;
    END;
    RETURN FALSE
  END Insert;
```

Den erhöhten Aufwand, den man hier braucht, hat man gut investiert. Einmal bleibt das Zeitverhalten beim Einfügen bei $O(\log(n))$, auch die Zeit zum Suchen ist immer garantiert $O(\log(n))$.

Wir wollten Ihnen hier lediglich die Funktionsweise eines AVL-Baumes nahebringen, Sie brauchen jedoch nicht diese Routinen selbst implementieren, auch dafür gibt es ein generisches Standardmodul `AVLTrees`.

4.9.5 Hashing

Der Zeitaufwand beim Suchen entsteht durch das häufige Vergleichen mit verschiedenen Werten der Struktur, in der sich die Daten befinden. Optimal wäre es doch, wenn man direkt aus dem Schlüssel (z. B. einem Namen) auf den zugehörigen Datensatz schließen könnte, ohne erst noch viel vergleichen zu müssen. Dieses Verfahren ist leicht zu praktizieren, wenn die Schlüssel Zahlen in einem kleinen Intervall sind. Dann müßte man nur ein Array mit diesem Intervall als Indexbereich einrichten, und könnte sofort auf den Satz zugreifen.

```
TYPE
  Key    = [5..200];
  DPtr   = POINTER TO Data;
  Data   = RECORD
              ...
           END;
VAR
  Search : ARRAY Key OF DPtr;

PROCEDURE Insert(VAR d : Data; k : Key);
BEGIN
  Search[k]:=Data'PTR
END Insert;

PROCEDURE Search(k : Key):DPtr;
BEGIN
  RETURN Search[k]
END Search;
```

Bei größeren Schlüsseln, z. B. Namen, steigt die Größe des Arrays ins Unermeßliche, bei sechs Zeichen pro Wort auf 281'000'000'000'000 Felder. Dieses Feld wäre aber nur sehr dünn gefüllt, da die Zahl der üblichen Namen weit geringer ist. Könnte man nun jedem Namen einen eindeutigen Schlüssel zuordnen, wobei jeder Schlüssel nur einmal vorkommt, wäre man aus dem Schneider. Leider ist dies aber unmöglich (oder möchten Sie gerne Herr/Frau 123 245 heißen?). Eine zwar nicht ganz so gute aber dennoch annehmbare Lösung ist, jedem Namen einen eindeutigen Schlüssel zuzuordnen, so daß alle verwendeten Schlüssel möglichst gleich häufig benutzt werden. Einen derartigen Schlüssel ermittelt man am besten mit einer Formel aus den ASCII-Codes der Buchstaben. Dieses Verfahren nennt sich Hashing (zerhacken), der Index Hashwert.

Die einfachste Möglichkeit wäre, alle Codes aufzusummieren, und dann den Rest der Division dieser Summe mit der Anzahl der benutzten Schlüsseln als Index zu verwenden:


```
CONST MaxKey = 100;

PROCEDURE EvalKey(VAR s : STRING):INTEGER;
VAR i : INTEGER;
    j : LONGINT;
BEGIN
  j:=0;
  FOR i:=0 TO s.len-1 DO
    INC(j,INTEGER(s.data[i]))
  END;
  RETURN j MOD MaxKey
END EvalKey;
```

Dieses Verfahren hat allerdings den Nachteil, daß alle Namen, die sich nur durch die Position ihrer Buchstaben unterscheiden, den selben Index haben. Besser ist dieses Verfahren.

```
PROCEDURE EvalKey(VAR s : STRING):INTEGER;
VAR i : INTEGER;
    j : LONGINT;
BEGIN
  j:=0;
  FOR i:=0 TO s.len-1 DO
    j:=j*2+INTEGER(s.data[i]);
  END;
  RETURN j MOD MaxKey
END EvalKey;
```

Tauchen in einer Hashtabelle zwei Namen mit dem selben Schlüssel auf, spricht man von einer Kollision. Um dem zu begegnen, werden alle Namen mit dem selben Schlüssel in einer Liste gehalten, und nacheinander mit dem gesuchten verglichen. Es gibt auch die Möglichkeit diese Namen über noch freie Felder der Hashtabelle zu verteilen. Dies bringt aber einige Probleme mit sich, etwa wenn die Tabelle voll ist.

```
TYPE
  DataPtr = POINTER TO Data;
  Data    = RECORD
              next : DataPtr;
              name : STRING(10);
              data : INTEGER
            END;
VAR
  Hash    = ARRAY [0..MaxKey-1] OF DataPtr;
```

Um die Tabelle zu initialisieren, müssen alle Felder mit leeren Listen belegt werden.

```
PROCEDURE Init;
VAR i : INTEGER;
BEGIN
  FOR i:=0 TO MaxKey-1 DO
    Hash[i]:=NIL;
  END;
END Init;
```

Einfügen in ein Hash läuft ab, wie bei einer normalen Liste, hat also einen Zeitbedarf von $O(1)$.

```
PROCEDURE Insert(VAR name : STRING;data : INTEGER);
VAR p : DataPtr;
    i : INTEGER;
BEGIN
  New(p);
  p^.name:=name;
  p^.data:=data;
  i:=EvalKey(name);
  p^.next:=Hash[i];
  Hash[i]:=p;
END Insert;
```


Die Suche läuft ebenfalls ab, wie in einer Liste. Nur ist die Laufzeit nicht $O(n)$, da in den Listen im allgemeinen nicht n Elemente enthalten sind. Bei einer Hashgröße von m Schlüsseln, und n Namen ergibt sich $O(n/m)$. Sind also m und n einigermaßen gleich, ergibt sich $O(1)$ für das Suchen.

```
PROCEDURE Search(VAR name : STRING):INTEGER;
VAR p : DataPtr;
BEGIN
  p:=Hash[EvalKey(name)];
  WHILE p#NIL
    AND_WHILE NOT Equal(name,p^.name) DO
      p:=p^.next
    ELSE
      RETURN p^.data
    END
  ELSE
    RAISE2(NotFoundError);
  END;
END Search;
```

Hashing ist das, bei bekannter Zahl von Namen, wohl optimalste Verfahren, Daten wiederzufinden. Dies hat wohl auch die Entwickler des Amiga dazu inspiriert, im Gegensatz zu MS-DOS, die Namen in einem Verzeichnis in einem Hash abzulegen. Kollisionen werden dabei durch eine verkettete Liste abgefangen. Diese Liste besteht aus den Anfangsblöcken der Dateien und Unterverzeichnissen. Dies ist der Grund, warum ein `DIR` auf einem Amiga so viel länger dauert, als auf einem MS-DOS Maschinchen. Dafür ist aber der eigentliche Zugriff auf eine Datei wesentlich schneller, da nicht die Liste aller Dateien durchgegangen werden muß.

4.10 Graphenalgorithmen

Graphen spielen in der Informatik eine gewichtige Rolle. Viele auftretende Probleme lassen sich mit Graphen darstellen und lösen. Die verschiedenen Dateien, die zu einem Programm gehören, lassen sich in ihrer Abhängigkeit als Graph darstellen.

```
DEFINITION MODULE a;
END a.

IMPLEMENTATION MODULE a;
BEGIN
CLOSE
END a.


DEFINITION MODULE b;
IMPORT a;
END b.

IMPLEMENTATION MODULE b;
BEGIN
CLOSE
END b.


DEFINITION MODULE c;
IMPORT a;
END c.

IMPLEMENTATION MODULE c;
IMPORT b;
BEGIN
CLOSE
END c.
```



```
MODULE d;
IMPORT c;
IMPORT a;
BEGIN
CLOSE
END d.
```

Mit einem derartigen Graphen arbeitet die Make-Routine, die am Ende dieses Kapitels beschrieben wird.

4.10.1 Depth-First versus Breadth-First

Müssen alle Knoten und/oder Kanten eines Graphen bearbeitet werden (man sagt auch „besucht“), gibt es zwei grundlegende Methoden. Depth-First (Tiefensuche) besucht erst alle Nachfolger des ersten Nachfolgers des Knotens, bevor es sich dem zweiten zuwendet. Breadth-First (Breitensuche) besucht erst alle direkten Nachfolger eines Knotens, bevor es sich deren Nachfolger zuwendet. Um bei einem zyklischen Graphen einen Knoten nicht unendlich oft zu besuchen, werden die besuchten Knoten markiert.

```
TYPE
  KantePtr = POINTER TO Kante;
  Kante    = RECORD
               next  : KantePtr
               to    : KnotenPtr
             END;
  KnotenPtr= POINTER TO Knoten;
  Knoten   = RECORD
               next   : KnotenPtr;
               kanten : KantePtr
             END;
```

Für Depth-First bietet sich natürlich ein rekursiver Algorithmus an.

```
PROCEDURE DepthFirst(k : Knoten);
VAR p : KantePtr;
BEGIN
  ...WorkOn...;     (* hier wird was getan *)
  p:=k^.kanten;
  WHILE p#NIL DO
    DepthFirst(p^.to);
    p:=p^.next;
  END;
END DepthFirst;
```

Zur Breitensuche braucht man eine Queue, in der die zu besuchenden Knoten gehalten werden.

```
PROCEDURE BreadthFirst(k : Knoten);
VAR p : KantePtr;
BEGIN
  PutQueue(k);
  WHILE NOT QueueEmpty DO
    ...WorkOn...;   (* hier wird was getan *)
    GetQueue(k);
    p:=k^.kanten;
    WHILE p#NIL DO
      PutQueue(p^.to);
      p:=p^.next
    END;
  END;
END BreadthFirst;
```

Im allgemeinen ist die Tiefensuche vorteilhafter, da sie mit weniger Aufwand auskommt.

4.10.2 Abhängigkeiten

Häufig wird über Graphen die Abhängikeit von Objekten untereinander ausgedrückt (z. B. wie oben die Abhängigkeiten von Programmeinheiten). Aus diesen Abhängigkeiten müssen dann weitere Folgerungen gezogen werden. Als Beispiel möchte ich wieder auf das Make zurückgreifen. Alle Objekt- und Symboldateien werden durch einen eigenen Knoten repräsentiert, wobei neben den Abhängigkeiten auch noch ein Flag `toCompile` im Knotenrecord enthalten ist und angibt, ob dieses File neu kompiliert werden muß. Dies Eigenschaft vererbt sich auf alle abhängigen Knoten.

```
PROCEDURE IsToCompile(k : KnotenPtr):BOOLEAN;
VAR p : KantePtr;
BEGIN
  p:=k^.kanten;
  WHILE p#NIL DO
    k^.toCompile:=k^.toCompile OR IsToCompile(p^.to);
    p:=p^.next
  END;
  RETURN k^.toCompile
END IsToCompile;
```

Diese Funktion wird für den obersten Knoten (das fertige Programm `D`) aufgerufen und ermittelt alle Dateien, die compiliert werden müssen. Wenn die Dateien dann nacheinander compiliert werden, muß darauf geachtet werden, daß ein Modul erst dann übersetzt werden darf, wenn alle Module, von dem es abhängig ist, übersetzt sind.

```
PROCEDURE Compile(k : KnotenPtr);
VAR p : KantePtr;
BEGIN
  IF k^.toCompile THEN
    p:=k^.kanten;
```

```
      WHILE p#NIL DO
        Compile(p^.to);
        p:=p^.next
      END;
      Compiler.Compile(k);
      k^.toCompile:=FALSE
    END;
  END Compile;
```

Diese Prozedur wird für alle Knoten nacheinander aufgerufen und sorgt selbst dafür, daß weder ein Modul doppelt compiliert und eins das müßte, nicht oder zu früh compiliert wird.

4.10.3 Kürzester Pfad

Eine brauchbare Anwendung für BreadthFirst ist die Lösung der Aufgabe, den kürzesten Weg von einem Knoten zu einem anderen zu finden. Dabei ist die Länge eines Weges gleich der Anzahl der besuchten Knoten.

```
  PROCEDURE ShortestPath(from,to : KnotenPtr):INTEGER;
  VAR p : KantePtr;
      i : INTEGER;
  BEGIN
    PutQueue(from);
    PutQueue(0);
    WHILE NOT QueueEmpty DO
      GetQueue(from);
      GetQueue(i);
      IF from#to THEN
        INC(i);
        p:=from^.kanten;
        WHILE p#NIL DO
          PutQueue(p^.to);
          PutQueue(i);
```



```
          p:=p^.next
        END;
      ELSE
        RETURN i
      END
    END;
    RAISE2(NoWayError);
  END ShortestPath;
```

Sollen die kürzesten Entfernungen aller Knoten zu einem bestimmten ermittelt werden, geschieht dies wegen der geringeren Verwaltung mit DepthFirst.

```
  TYPE
    Knoten   = RECORD
                 ...
                 entf : INTEGER
               END;
  VAR
    Knoten0 : KnotenPtr;

  PROCEDURE Depth(k : Knoten);
    PROCEDURE Depth2(k : Knoten;entf : INTEGER);
    VAR p : KantePtr;
    BEGIN
      INC(entf);
      IF k^.entf>entf THEN
        k^.entf:=entf;
        p:=k^.kanten;
        WHILE p#NIL DO
          Depth2(p^.to,entf);
          p:=p^.next
        END;
      END;
    END Depth2;
```

```
VAR q : Knoten;
BEGIN
  q:=Knoten0;
  WHILE q#NIL DO
    q^.entf:=INTEGER'MAX;
    q:=q^.next
  END;
  Depth2(k,-1);
END Depth;
```


4.11 Strukturiertes Programmieren

Cluster ist eine strukturierte Programmiersprache der Pascal-Schule. Ein besonderes Merkmal einer strukturierten Sprache ist, daß Schleifen und Bedingungen nicht durch Sprünge sondern durch Strukturen beschrieben werden.

Eine Schleife, die ein Element in einem Array sucht, hat in Urbasic folgendes Aussehen:

```
1000 i=0
1010 IF a(i)<>wert THEN i=i+1 : GOTO 1010
```

Und in **Cluster**:

```
i:=0;
WHILE a[i]#wert DO
  INC(i)
END;
```

In einem unstrukturierten Programm ist der Programmfluß häufig nicht einfach zu erkennen, da mangels Strukturierungsmöglichkeiten wild hin und her gesprungen wird (Spaghetticode, da der Programmfluß vielfach verknotet ist). In einer strukturierten Programmiersprache gibt es keine Sprünge. Der Programmfluß ist deutlich an den Strukturen zu erkennen.

Beispiel: Bilden des GGT mit Basic

```
1000 INPUT a,b
1010 IF b>a THEN 1030
1020 x=b:b=a:a=x
1030 c=a/b
1040 IF c=INT(c) THEN 1070
1050 c=a-b*INT(c):a=b:b=c
```

```
1060 GOTO 1030
1070 PRINT b
```

Und jetzt dieses Programmfragment mit **Cluster**:

```
ReadInt(a);ReadInt(b);
IF b>a THEN
  x:=a; a:=b; b:=x
END;
c:=a MOD b;
WHILE c#0 DO
  a:=b; b:=c;
  c:=a MOD b
END;
WriteInt(b);
```

Der Programmfluß im oberen Programm ist nur sehr schwer zu erkennen, da man den Sprüngen nachlaufen muß. Im unteren Programm ist der Fluß wesentlich einfacher zu erkennen, da sich Bedingung und Schleife sofort unterscheiden lassen.

Strukturierte Programmiersprachen sind meist auch formatfrei, das heißt, es ist kein festes Format vorgegeben, in dem der Programmtext stehen muß. Somit kann der Programmtext so ausgerichtet werden, daß er einen Leser unterstützt.

Eine weitere wichtige Eigenschaft moderner Programmiersprachen ist die Strukturierbarkeit von Daten. In Basic und Fortran gibt es als einzige Typen Zahlen, Zeichen und Arrays. Dynamisches Programmieren ist also so gut wie unmöglich.

4.11.1 Äußere Form eines Programms

Das Ziel eines Programmierers ist es, einen Algorithmus richtig, lesbar und effizient (in dieser Reihenfolge) zu implementieren. Um die Lesbarkeit und damit die Bearbeit- und Überschaubarkeit eines Programms zu verbessern, sollten einige Regeln eingehalten werden.

Innere Programmteile, z. B. Schleifenkörper sollten gegenüber dem Kopf um etwa zwei Zeichen nach rechts eingerückt werden. Das Ende der Schleife (`END` oder `REPEAT`) sollte wieder auf gleicher Höhe mit dem Anfang stehen.

```
WHILE ... DO
  Anweisung 1;
  ...
  Anweisung n
END;
```

Gleichrangige Anweisungen sollten mit dem Kopf der Struktur auf gleicher Höhe erscheinen.

```
IF ... THEN
  Anweisung 1.1;
  ...
  Anweisung 1.n
OR_IF ... THEN
  Anweisung 2.1;
  ...
  Anweisung 2.n
ELSE
  Anweisung n.1
  ...
  Anweisung n.n
END;
```

Eine einschränkende Anweisung (`AND_IF`, `AND_WHILE`) sollte ebenfalls zwei Zeichen eingerückt werden. Anweisungen sollten nur dann hintereinander in eine Zeile geschrieben werden, wenn sie unmittelbar zusammengehören, z. B. bei einem Tausch zweier Variablen, oder der Ausgabe gemischter Daten.

```
...
x:=a; a:=b; b:=x;
...
WriteString(" a*b="); WriteInt(a*b,0); WriteLn;
```

Bei einer Anweisung mit dem Schlüsselwert KEY sollten die Schlüssel und die zugehörigen Anweisungen in verschiedenen Spalten stehen.

```
IF KEY a
  OF 1..2,3  THEN
               Anweisung 1.1;
               ...
               Anweisung 1.n
             END
  OF 4,7,8   THEN
               Anweisung 2.1;
               ...
               Anweisung 2.n
             END
ELSE
  Anweisung n.1
  ...
  Anweisung n.n
END;
```

Gehört zu jedem Wert nur eine Anweisung, können diese auch gemeinsam in der selben Zeile stehen, wobei auch alle END untereinander stehen sollten.

```
IF KEY a
  OF 1..2,3 THEN Anweisung 1   END
  OF 4,7,8  THEN Anweisung 2   END
  OF ...    THEN ...           END
END
```


Ein Ausdruck sollte nur dann geteilt werden, wenn es unbedingt nötig ist. In diesem Fall sollte man darauf achten, daß deutlich wird, daß die Zeilen zusammengehören.

```
det:=a[0,0]*a[1,1]*a[2,2]-a[2,0]*a[1,1]*a[0,2]+
     a[0,1]*a[1,2]*a[2,0]-a[2,1]*a[1,2]*a[0,0]+
     a[0,2]*a[1,0]*a[2,1]-a[2,2]*a[1,0]*a[0,1];
```

Hat eine Prozedur eine größere Anzahl Parameter, so sollte die Definition des Prozedurkopfes in mehrere Zeilen zerlegt werden, wobei wieder auf die Spalten geachtet werden sollte.

```
PROCEDURE OpenFile(VAR file   : File;
                   VAR name,
                       path   : STRING;
                       buffer : INTEGER;
                       new    : BOOLEAN):BOOLEAN;
```

Namen sollten so vergeben werden, daß die Bedeutung des beschriebenen Objekts deutlich wird. Namen sollten nicht völlig groß geschrieben werden, um sich von den Standardnamen zu unterscheiden. Großbuchstaben im Wort, oder der Unterstrich „_“ können in Bezeichnern verwendet werden, um deren Lesbarkeit zu erhöhen. Prozeduren und Typen sollten immer mit einem Großbuchstaben beginnen. Recordbezeichner und Namen in einem Aufzählungstyp sollten klein geschrieben werden.

Bei der Deklaration von Typen, Variablen und Konstanten sollte man sich ebenfalls an Spalten halten.

```
TYPE
  Koordinate = RECORD
                 x,
                 y  : INTEGER
               END;
```

```
    Polygon    = ARRAY OF Koordinate;
    PolygonPtr = POINTER TO Polygon;

  VAR
    myTriangle : Polygon(3);
    myPolygon  : PolygonPtr;

  CONST
    Quadrat    = Polygon:((x=-10,y=-10),
                          (x= 10,y=-10),
                          (x= 10,y= 10),
                          (x=-10,y= 10));
```

Zusammengehörende Objekte sollte man von anderen durch Leerzeilen abtrennen, um deren Gemeinsamkeit deutlich hervorzuheben.

4.11.2 Kommentierung

Dies ist das dunkelste Kapitel in der Geschichte der Informatik. Es gibt wohl keinen Programmierer, der seine Programme gerne kommentiert. (Es sollen aber auch schon Exzesse geschehen sein, in denen das Programm derart im Kommentar verschwand, daß der Programmierer es selbst nicht mehr entdecken konnte.) Aber Kommentierung muß sein; bei kleinen Programmen mag man ja noch den Überblick behalten, aber größere Projekte, an denen auch noch mehrere Programmierer arbeiten, sind ohne ausreichende Kommentierung nicht fertigzustellen.

Am wichtigsten sind Kommentare in Definitions-Modulen, da diese vermutlich auch von anderen benutzt werden. Dabei sollten alle Typen mit ihren Aufgaben kurz beschrieben werden. Jede Prozedur sollte einen Kommentarkopf erhalten.


```
 |
 |  AUFGABE       : Element an den Anfang der Liste einfügen
 |  PARAMETER     : Listenanfang, einzufügendes Element
 |  SEITENEFFEKTE : -
 |  BEMERKUNGEN   : Die Listenelemente werden nicht sortiert
 |                  eingefuegt!
 |
PROCEDURE Insert(VAR list : List;
                     node : NodePtr);
```

Es empfiehlt sich einen derartigen Kopf auch in Implementations-Modulen zu verwenden. Globale Variablen sind ebenfalls mit einem Kommentar zu versehen.

Grundsätzlich läßt sich sagen, daß eine Kommentierung immer so sein sollte, daß ein anderer Programmierer Sinn und Verwendung im Definitions-Modul und die Funktion in der Implementierung erkennen kann.

4.11.3 Datenstrukturierung

Cluster verfügt über die in modernen Programmiersprachen üblichen Typkonstruktoren, Verbund (`RECORD`), Reihung (`ARRAY`, `STRING`), Menge (`SET`) und Indirektion (`POINTER`, `CLASSPTR`). Diese Möglichkeiten sollten auch so genutzt werden, da Sinn und Zusammenhang deutlich werden. Sollen z. B. die Daten von 20 Personen bearbeitet werden, gibt es zwei Möglichkeiten:

```
  TYPE
    Daten = RECORD
              namen : ARRAY [0..19] OF STRING(20);
              alter : ARRAY [0..19] OF INTEGER;
              phone : ARRAY [0..19] OF STRING(12);
            END;
```

oder:

```
TYPE
  Person= RECORD
            namen : STRING(20);
            alter : INTEGER;
            phone : STRING(12);
          END;
  Daten = ARRAY [0..19] OF Person;
```

wobei die erste Lösung abzulehnen ist, da sie den Zusammenhang der Felder nicht deutlich macht. Der Sinn von Typen ist, die Datenobjekte abstrakter zu machen, um so das Programmieren zu erleichtern. Arbeitet ein Programm zum Beispiel mit Vektoren, so vereinfacht ein Typ `Vector` die Arbeit, im Vergleich zur Verwendung von immer drei Variablen.

```
TYPE
  Vector        = ARRAY [0..2] OF REAL;
VAR
  va, vb, vc, vc : Vector;

PROCEDURE VAdd(v1,v2 : Vector):Vector;
BEGIN
  ...
END VAdd;

BEGIN
  ...
  va:=VAdd(vb,VAdd(vc,vd));

...
```


4.11.4 Modularisierung

Ein wichtiges Konzept in **Cluster** ist die Modularisierung. Sie dient der Vereinfachung der Arbeit, da das Rad nicht immer neu erfunden werden muß. Die meisten Programmiersprachen verfügen über die Möglichkeit, Programmeinheiten getrennt zu übersetzten. Dies ist meist jedoch nur rudimentär gelöst, so daß die Typprüfung und ähnliches nicht über Modulgrenzen hinweg möglich ist. **Cluster** prüft Typen über Modulgrenzen hinweg, so daß importierte Objekte genauso sicher sind wie eigene. Durch die Zerlegung eines Moduls in Definitions- und Implementations-Teil wird die Implementations-Entscheidung nach hinten verlegt. Wird nur das Implementations-Modul neu compiliert, braucht kein abhängiges Modul neu übersetzt werden. Durch den Datentyp `HIDDEN` ist es möglich, Objekte und Prozeduren damit zu definieren, ohne daß im Definitions-Modul schon festgelegt ist, wie der Typ aussieht.

```
DEFINITION MODULE IntStack;
TYPE
  Stack = HIDDEN;

PROCEDURE CreateStack(VAR s : Stack;size : INTEGER);
PROCEDURE DeleteStack(VAR s : Stack;size : INTEGER);
PROCEDURE IsEmpty(s : Stack):BOOLEAN;
PROCEDURE Push(s : Stack;elem : INTEGER);
PROCEDURE Pop(s : Stack;VAR elem : INTEGER);

END IntStack;
```

Dieses Definitions-Modul funktioniert, egal, ob der Stack als Liste oder Array realisiert wird, und selbst wenn dies geändert wird, muß kein abhängiges Modul übersetzt werden. Dieses Verbergen von Informationen nach außen nennt sich “information hiding”, und sollte wann immer möglich Verwendung finden.

Module sollten möglichst allgemein sein, um ein breites Anwendungsspektrum zu besitzen. Dies kann besonders in Verbindung mit offenen Typen und Prozedurvariablen erreicht werden.

```
DEFINITION MODULE SortedList;
TYPE
  NodePtr = POINTER TO Node;
  Node    = RECORD
              prev,
              next   : NodePtr;
            END;
  Greater = PROCEDURE(VAR node1,node2 : Node):BOOLEAN;
  List    = HIDDEN;

PROCEDURE CreateList(VAR l : List; greater : Greater);
PROCEDURE Insert(l : List;elem : NodePtr);
...
END SortedList;
IMPLEMENTATION MODULE SortedList;
TYPE
  List    = POINTER TO
              RECORD OF Node
                greater : Greater
              END;
PROCEDURE CreateList...
...
  l^.prev:=l;
  l^.next:=l;
  l^.greater:=greater;
...
END CreateList;

PROCEDURE Insert(l : List; elem : NodePtr);
VAR p : List;
BEGIN
```



```
    p:=l^.next;
    WHILE (p#l) AND NOT l^.greater(p^,elem^) DO
      p:=p^.next
    END;
    ...
  END Insert;
```

Der wirkliche Typ, den diese Liste zu verwalten hat, ist dem Modul selbst völlig egal.

4.11.5 Dinge, die man niemals tut!!!

- In diesem Abschnitt möchte ich ein paar Beispiele dafür geben, was ein guter Programmierer vermeiden wird.
- Vermeiden Sie globale Variablen und Prozeduren mit Namen wie `i`, `p`, `do(...)` etc., denen wirklich nicht anzusehen ist, wofür sie gut sind.
- Benutzen Sie keine globalen Variablen für lokale Daten. Dies sorgt für die interessantesten Effekte, wenn das Programm wächst.
- Deklarieren Sie keine Objekte, die niemals Anwendung finden. Diese Objekte werden zwar durch den Linker entfernt, verwirren aber den Leser des Programmtextes.
- Verändern Sie keine importierten Variablen, wenn ihnen dies nicht ausdrücklich erlaubt wurde. Vermeiden Sie möglichst Variablen zu exportieren.
- Definieren Sie nichts in einem Definitions-Modul, was besser in die Implementation gehört.
- Schreiben Sie Anweisungen, die nicht zusammengehören, nicht in eine Zeile.
- Benutzen Sie keine nichtinitialisierten Variablen, besonders Pointer, obwohl diese meistens `NIL` enthalten.

- Vermeiden Sie `LOOP...EXIT...END`, soweit dies geht.
- Verzichten Sie auf Funktionen mit Nebeneffekten. Eine Funktion sollte einen Wert zurückliefern, nicht aber andere Daten verändern. Eine Ausnahme bilden hier Prozeduren, die mit einem Boolwert angeben, ob Sie funktioniert haben. Auf solche sollte man jedoch in der Zwischenzeit verzichten, und stattdessen Exceptions verwenden, um einen Fehler anzuzeigen.
- Vermeiden Sie wenn möglich die Übergabe größerer Typen an eine Prozedur als Werteparameter, da dies Stack und Laufzeit belastet.

4.12 Programmiertechniken

Die Informatik beschäftigt sich nun seit über dreißig Jahren ziemlich erfolglos damit, aus dem Programmieren eine Wissenschaft zu machen. Der größte Erfolg dieser Forschung ist wohl die strukturierte Programmiersprache. Einige Dinge, die bei diesen Versuchen noch abgefallen sind, habe ich hier zusammengefaßt. Sie mögen ihnen skuril, oder zumindest doch relativ weltfremd vorkommen, sind aber Bestandteil der modernen Informatik.

4.12.1 Top-Down, Bottom-Up

Ein Programm wird in zwei Richtungen entwickelt. Zuerst wird die Aufgabe in immer kleinere Teilaufgaben zerlegt (Top-Down). Danach werden diese wieder aufsteigend implementiert (Bottom-Up). Dies hat den Vorteil, daß der Bereich, an dem gearbeitet wird, klein und überschaubar wird. Auf den Punkt gebracht heißt das nichts anderes als: „Ein Programm besteht aus Programmen“ und „Viele kleine Programme ergeben ein großes“.

4.12.2 Programmentwicklung mit Induktion

Induktion ist ein Verfahren Programme zu entwickeln, das im allgemeinen zu rekursiven Lösungen führt, und Top-Down verherrlicht. Ein Induktions-Fanatiker geht davon aus, daß er jedes Problem lösen kann, wenn er es in kleinere zerlegt. Soll er in einem Feld ein Element suchen, wird er sich folgendes Überlegen:

Ich kann das Element finden, wenn ich es in einem kleineren Feld finden kann. In einem Feld mit einem Element, kann ich es finden, wenn es dieses ist. Um von einem Feld mit n Elementen auf eines mit einem Element weniger zu kommen, muß ich ein Element entfernen. Ist dieses Element das gesuchte, bin ich fertig, sonst muß ich das kleinere Feld prüfen.

```
PROCEDURE Search(VAR feld  : ARRAY OF INTEGER;
                     elem,
                     size  : INTEGER):INTEGER;
BEGIN
  IF feld[size-1]=elem THEN
    RETURN size-1
  ELSE
    RETURN Search(feld,elem,size-1)
  END;
END Search;
```

Dieses Verfahren ist durchaus geeignet, einfache Probleme zu erschweren, sorgt aber bei Aufgaben, die man zerlegen kann, für meist richtige Lösungen.

4.12.3 Divide and Conquer

Die beste Technik, die mit Induktion arbeitet, ist *Divide und Conquer* (Teile und Herrsche). Bei diesem Verfahren wird die Aufgabe in zwei möglichst gleich große Aufgaben zerlegt und diese getrennt sortiert. Beispiele dafür sind *Binärsuche* und *Quicksort*. Dieses Verfahren liefert meist optimale Algorithmen und sollte Berücksichtigung finden.

4.12.4 Programmentwicklung mit Deduktion

Dieses Verfahren versucht mit Regeln und logischen Überlegungen Algorithmen zu entwicklen. Ein Deduktions-Fanatiker wird das Suchproblem von oben etwa so angehen.

Es handelt sich um ein Feld, also wird wohl eine Schleife nötig sein.

```
WHILE ... DO ... END;
```


Die Schleife ist dann beendet, wenn das richtige Element gefunden ist.

```
WHILE feld[i]#such DO ... END;
```

Das beste wird es wohl sein, die Schleife am Anfang zu beginnen, und dann vorwärts laufen zu lassen.

```
i:=0;
WHILE feld[i]#such DO INC(i) END;
```

Mit Deduktion lassen sich viele Probleme so lösen, wie man es ohne auch machen würde.

4.12.5 Rekursion versus Iteration

Rekursive Verfahren bestehen darin, daß ein Problem in kleinere zerlegt wird, und die Lösungsroutine sich damit aufruft. Dies führt irgendwann einmal auf einen Basisfall, der sich einfach lösen läßt. Iterative Verfahren schreiten im Lösen einer Aufgabe in Richtung auf die Lösung hin. Diese erreichen sie nach einer endlichen Zahl Schritte.

Rekursive Algorithmen sind wegen der häufigen Prozeduraufrufe meist langsamer und speicher-verschwendender als iterative. Dafür sind sie aber häufig überschaubarer als diese. Theoretisch läßt sich jeder rekursive Algorithmus mit einem Stapel in einen iterativen umsetzen.

4.12.6 Programmbeweise

Dies ist der jüngste und abstrakteste Einfall der theoretischen Informatik. Das Problem beim Programmieren ist, daß man nie weiß, ob ein Programm auch fehlerfrei ist. Man kann noch so viel testen und probieren, früher oder später taucht doch wieder ein Fehler auf. Um dem abzuhelfen wurde eine Methode entwickelt, mit der sich die Richtigkeit eines Programmes im mathematischen Sinn beweisen läßt (Sie haben richtig gelesen, es gibt Leute die beweisen, daß ihr Programm auch funktioniert). Die Technik basiert auf der mathematischen Logik, und ist extrem abstrakt. Doch hat man ein Programm bewiesen, ist man völlig sicher, daß es korrekt ist. Es sei denn, im Beweis steckt ein Fehler. Dies, und der immense Zeitaufwand machen diese Technik für die Praxis unanwendbar. Doch in diese Forschung werden Millionen gepumpt. Der größte Sponsor ist das Dod (Department of Defence) und die NATO, und der Auslößer ist das SDI-Projekt. Das einzig interresante an der ganzen Forschung, ist mal wieder, daß sich die Größen der Forschung mit Geld durch die Militärs korrumpieren lassen.

Kapitel 5

Systemprogrammierung mit Cluster

Nachdem Sie bisher bei der Programmierung nur über die Standardmodule die Möglichkeit hatten, die phantastischen Fähigkeiten Ihres Amigas zu nutzen, wird Ihnen jetzt gezeigt, wie das Betriebssystem direkt programmiert wird, damit Sie noch flexibler mit Ihrer Maschine umgehen können.

5.1 Was ist ein Betriebssystem?

Das Betriebssystem ist das grundlegende Verwaltungssystem, das die ganze Zeit, die der Rechner eingeschaltet ist, wie ein öffentlicher Dienst (der nie streikt) den Programmen die Rechnerresourcen zur Verfügung stellt. So kann man einige Hauptaufgaben unterscheiden:

Prozesse Ein modernes Betriebssytem kann mehrere Prozesse (Programme) gleichzeitig ausführen. Dazu muß die zur Verfügung stehende Zeit ehrlich aufgeteilt werden und dafür gesorgt werden, daß alle Programme gleichzeitig die Resourcen des Betriebssystems in Anspruch genommen werden können, ohne daß dabei Konfliktsituationen auftreten.

Kommunikation Wenn man mehrere Programme gleichzeitig ausführen kann, dann möchte man auch diese Programme miteinander kommunizieren lassen, um optimale Flexibilität zu ermöglichen (z. B. Drucken im Hintergrund und Unterbrechung, wenn es Probleme gibt). Die Kommunikation zur Außenwelt (Graphik, Sound, Schnittstellen, Speichermedien, Tastatureingabe, etc.) wird ebenfalls vom Betriebssystem organisiert.

Hardware Eine jede Rechnerhardware unterliegt einer gewissen Evolution und da wäre es praktisch, wenn die alten Programme die neuen Möglichkeiten des Rechners nutzen können (z. B. neue Graphikmodi von neuen Graphikchips). Außerdem kann in einem Multitaskingsystem nicht jeder gleichzeitig auf die Hardware zugreifen, da diese sonst sicherlich sehr schnell durcheinander geraten würde.

Bibliotheken Ein jedes Betriebssystem bietet eine gewisse Menge an Standardroutinen, die von den Programmen genutzt werden können. Das spart zum einen Platz (sowohl auf der Platte, als auch im Speicher), zum anderen erleichtert es die Anpassung an eine neue Version, da das Programm nur neu gestartet werden muß, aber kein neues Programm hergestellt werden muß.

5.2 Cluster und Kickstart

Um sich die Organisation des Rechners besser vorstellen zu können, stelle man sich den Amiga als ein großes Fabrikunternehmen vor und das Betriebssystem als Firmenleitung.

Unser eigenes und alle anderen Programme sind lediglich kleine Abteilungen, die der Firmenleitung unterstehen und deren Anlagen nutzen können.

Nun stellt sich natürlich die Frage, wie man die Resourcen des Rechners, also die Anlagen der Firmenleitung, verwenden kann. Normalerweise muß man von Hand alle zu verwendenden Resourcen öffnen und schließen, wie es z. B. Assembler- oder C-Programmierer tun.

Als Clusterprogrammierer brauchen Sie sich nicht darum zu kümmern, da sich alle Schnittstellenmodule selbst darum kümmern, daß die Bibliotheken und Gerätetreiber bei Programmstart geöffnet und bei Programmbeendigung wieder geschlossen werden, sogar wenn das Programm unvorhergesehen abgebrochen wird.

Doch gerade bei der Nutzung der Bibliotheksfunktionen von der Sprache Cluster aus entsteht ein kleines Problem. Das Betriebssystem des Amigas wurde nämlich in „C“ geschrieben, und „C“ kennt zum einen nicht den Typ String, besitzt keine **VAR**-Parameter und kann keine komplexen Typen an Prozeduren übergeben.

5.2.1 Komplexe Typen

Das Problem komplexer Typen oder **VAR**-Parameter umgeht „C“, indem es Zeiger auf die übergebenen Werte übergibt, so auch bei Strings.

Außerdem sind Strings in „C“ einfach **ARRAY OF** CHAR, und ein Null-Byte kennzeichnet das Stringende.

Bei der Implementierung der Systemmodule sind wir so vorgegangen, daß wir, wenn sinnvoll, Pointer durch **VAR/REF**-Parameter ersetzt haben, wenn die zu verwendenden Parameter eher als Variable statt als dynamisch allozierte Struktur vorliegt.

5.2.1.1 Strings

Strings erhielten eine besondere Unterstützung, so daß der Unterschied von Cluster- zu C-Strings kaum noch ins Gewicht fällt. Voraussetzung für die sichere Verwendung von Clusterstrings im Zusammenhang mit Systemfunktionen ist, daß sie mit einem Nullbyte terminiert sind. Solange Sie nur Stringkonstanten sowie die Prozeduren aus den Modulen `Str` und `Strings` verwenden, ist dies immer sichergestellt. Modifizieren Sie jedoch einzelne Zeichen eines Strings von Hand, so müssen Sie sicherstellen, daß danach der Längeneintrag sowie das Nullbyte korrekt gesetzt worden sind. Um aus einem fehlenden Nullbyte resultierende Fehler zu vermeiden verfügt der Compiler über den Laufzeitcheck `StrZeroChk` der bei jeder Stringzuweisung überprüft, ob `str.data[str.len]= &0` ist.

Bei Library-Prozeduren, die einen **REF**-Parameter vom Typ String haben, kann man sowohl einen String[2] als auch auch einen `SysStringPtr` übergeben.

Bei Parametern vom Typ `SysStringPtr` kann man sowohl Zeiger als auch einen konstanten String übergeben; im letzteren Fall ermittelt der Compiler den Zeiger automatisch. Dies funktioniert auch bei Einträgen in Systemstrukturen vom Typ `SyStringPtr`. Der Typ „SysStringPtr“ ist im Modul System folgendermaßen definiert:

```
TYPE
  SysStringPtr : POINTER TO ARRAY OF CHAR;
```

Einen Clusterstring in einen SysStringPtr zu verwandeln ist sehr einfach[3]: `ptr:=str.data'PTR`, man ermittelt einfach den Zeiger des Datenfeldes des gewünschten Strings.

[2]Der Compiler übergibt automatisch einen Zeiger auf `str.data`

Um einen `SysStringPtr` in einen ClusterString umzuwandeln existiert im Modul `Strings` die Prozedur `Str()`:

```
$$OwnHeap:=TRUE
PROCEDURE Str(ptr IN A0 : SysStringPtr):STRING;
```

War `ptr = NIL`, wird ein String mit Länge Null zurückgegeben. Mit dieser Unterstützung sollte es leicht sein, mit den Systemstrings fertig zu werden.

5.2.1.2 Tags

Seit OS 2.0 verwendet das Betriebssystem eine neue Methode zur Übergabe von Parametern an Systemfunktionen, sogenannte „Tags“. Tags bieten den Vorteil, daß man leicht neue Parameter hinzufügen kann, ohne mit den alten in Konflikt zu kommen. Tags werden immer in Listen übergeben, dabei setzt sich ein Tag immer aus einer Tag-Id und einem 32-Bit Tag-Wert zusammen. Anhand der Tag-Id erkennt die Prozedur, der eine Tagliste übergeben wird, welche Bedeutung der Tag-Wert hat.

Im Gegensatz zu der Tag-Implementierung von „C“ verfügt Cluster über einen Typcheck für Tags, d. h. es wird überprüft, ob der Wert hinter der Tag-Id auch dem Typ der Tag-Id entspricht. Um dies zu gewährleisten, muß man für die einzelnen Ids die entsprechenden Typen definieren. Dabei ist daran zu denken, daß die Typen maximal 4 Bytes groß sein dürfen, für größere Objekte verwenden Sie bitte Zeiger.

Es ist möglich von bestehenden Tagtypen zu erben, somit enthält der neue Tagtyp die Elemente des Alten plus die neu definierten. Im Normalfall erbt man von `StdTags` die in `Utilities.def` definiert sind. Sie enthalten einige Standardtags, insbesondere den Tag `DONE`, durch welchen das Ende einer TagListe definiert wird.

[3] Seit die Nullbyte-Termination in die Sprachdefinition aufgenommen wurde, hat die Funktion `SysStr()` aus dem Modul Strings seine Bedeutung verloren und existiert nur noch aus Kompatibilitätsgründen.

Neben dem Typen trägt jede Tag-Id noch eine Nummer, an der Sie erkannt werden kann. Bei der Definition von neuen Tags muß nur der ersten Tag-Id ein Wert zugewiesen werden, die darauffolgenden werden automatisch durchnummeriert, ähnlich wie bei Aufzählungstypen. Am Beispiel der `ScreenTags`, die zum Öffnen eines Screens benötigt werden[4], soll die Definition von Tags gezeigt werden:

```
ScreenTags = TAGS OF StdTags
               width  = $80000020 : INTEGER;
               height             : INTEGER;
               depth              : INTEGER;
               name               : POINTER TO STRING;
               flags              : ScreenFlagSet
             END;
```

Tags werden in Form von TAG-Listen verwendet, das sind (nach Bedarf auch verkettete) Arrays von Tag-Typen.
Beispiel:

```
TYPE
  ScreenTagList = ARRAY OF ScreenTags;
```

Um nun einen Screen zu öffnen, muß eine Tag-Liste angegeben werden, in der die Elemente aufgeführt sind, die sich von den Werten eines Standardscreens unterscheiden. Der Vorteil auch hier, man muß keine unnötigen Werte angeben, die man gar nicht verändern möchte.

```
OpenScreenTagList(ScreenTagList:(width  : 320,
                                 height : 256,
                                 name   : "Name"'PTR,
                                 DONE));
```

Sicher ist Ihnen aufgefallen, daß bei Tags die Werte von der Tag-Id durch Doppelpunkte getrennt werden, da es sich hierbei um keine Zuweisung handelt. Bitte beachten Sie den letzten Tag-Eintrag `DONE`, er darf auf

[4] Die hier definierten entsprechen nicht den in `Intuition` definierten.

gar keinen Fall fehlen, da das System an ihm erkennt, daß die Liste hier endet.

Anstelle von einem konstanten Array können Tags auch als Typ eines **L**IST OF-Parameters verwendet werden, dann können als Argumente der Tags auch nicht konstante Ausdrücke verwendet werden. Die Funktion `Max` in unserem Beispiel gibt jeweils den größeren der Parameter zurück, `StandardWidth` sei eine Variable.

```
OpenScreenTags(ns : NewScreenPtr;tags : LIST OF ScreenTags);
...
OpenScreenTags(NIL, width  : Max(320,StandardWidth),
                    height : Max(256,StandardHeight),
                    name   : "Name"'PTR,
                    DONE);
```

Da ein Tag kein gewöhnlicher Clustertyp ist, kann man auch nicht direkt auf die einzelnen Elemente einer TagListe zugreifen. Hierfür sind zwei spezielle Standardfunktionen definiert, `TGET` und `TPUT`. `TGET` extrahiert einen Tag wertaus einer Liste, `TPUT` ändert einen Wert in einer Tagliste. `TGET` erhält drei Argumente: Die Liste, den Namen des Tags, dessen Wert ermittelt werden soll und einen Defaultwert, der zurückgeliefert werden soll, falls der Tag nicht in der Liste enthalten ist. `TPUT` hat eine ähnliche Argumentliste, lediglich der letzte Wert ist kein Defaultwert, sondern der neue Inhalt des Taglistenelements.

```
VAR tags := ScreenTagList:(width : 320, height : 256,
                           name : "Name"'PTR,DONE);
    w    : INTEGER;
...
   w:=TGET(tags,width,0);
   TPUT(tags,name,"Neuer name");
...
```

Da nur 32-Bit breite Tagtypen zugelassen sind, wurden zwei neue Typen für 32-Bit Zeichen und Booleanwerte eingeführt: `LONGBOOL` und `LONGCHAR`. Diese sind zu `BOOLEAN` bzw. `CHAR` vollkommen kompatibel, nur daß sie nicht 8 sondern 32 Bit belegen.

5.2.2 Weiterführende Literatur

Dieses Kapitel kann und will keine vollständige Abhandlung in das Amiga-Betriebssystem sein, vielmehr soll es als eine kleine Einführung dienen. Wer tiefer in diese Materie einsteigen will, dem seien die Rom-Kernel-Reference-Manuals (kurz RKMs), erschienen bei Addison Wesley, wärmstens empfohlen, sie ist die beste Literatur zu diesem Thema, leider nur in englischer Sprache. Desweiteren sei auf das Amiga-Guru-Buch von Ralph Babel hingewießen, in dem wohl die umfangreichste Dokumentation zum Thema Dos enthalten ist.

Außerdem empfiehlt es sich die Includes und Autodocs auf Diskette bei Hirsch & Wolf zu besorgen, da in den Autodocs eine kurze Beschreibung aller Library-Funktionen enthalten ist, auf die man dann direkt aus dem Editor zugreifen kann.

5.3 Exec, der Boss

Exec ist, wie schon die Überschrift sagt, der Teil des Betriebssystems, der über allem steht und alles koordiniert. Exec ist auch die einzige Library, die nicht geöffnet werden muß, sondern immer offen ist, denn schließlich enthält diese Library den Befehl zum Öffnen von Libraries. Schließlich kann man sich nicht wie Münchhausen an den eigenen Haaren aus dem Sumpf ziehen.

Daher steht die Basisadresse von Exec immer in Adresse $04 des Speichers. Folglich sollte man sich auch zurückhalten, den Inhalt dieser Adresse zu verändern, da sonst unvergleichliche Ausflüge nach Indien bevorstehen, die unerwünschte Amnesie im Kurz- und Langzeitgedächtnisses des Rechners hervorrufen können.

Doch Exec ist nicht nur dazu da Bibliotheken zu öffnen oder zu schließen. Von ihm wird auch das recht komplexe Multitasking gehandhabt (oder neudeutsch gehandelt). D. h., da auch beim Multitasking mehrere Programme nicht wirklich gleichzeitig ablaufen, sondern nur zwischen den einzelnen Programmen umgeschaltet wird, muß schließlich jemand bestimmen, welches Programm gerade läuft und welche Programme zu warten haben.

Die dritte wichtige Aufgabe von Exec ist die Verwaltung des Speichers, doch dazu später mehr.

5.3.1 Knoten, die allgegenwärtige Struktur

Man kann sich vorstellen, daß in einem derartig großen System die Verwaltung nicht gerade einfach ist, und ebenso wie in einem Großunternehmen sich der Chef nicht die Namen und Daten aller Angestellten merken kann, kann sich auch Exec nicht alle Verwaltungsstrukturen merken.

Wie ein Unternehmer, der Listen über seine Angestellten führt, macht dies auch Exec über verkettete Listen im Speicher, und weiß immer nur wo diese Listen „liegen“. Denn in der exec.library gibt es eine Struktur namens ExecBase, in der die Anfänge aller wichtigen Listen verzeichnet sind.

Listen bestehen neben einem Listenkopf (dazu gleich) aus sogenannten Knoten oder auch „Nodes“, die Records von folgendem Aufbau sind:

```
TYPE
  MinNodePtr    = POINTER TO MinNode;
  MinNode       = RECORD                        | 8 Bytes
                    succ,
                    pred  : SAMEPTR;
                  END;

  NodePtr       = POINTER TO Node;
  Node          = RECORD OF MinNode             | 14 Bytes
                    type  : NodeType;
                    pri   : NodePri;
                    name  : SysStringPtr;
                  END;
```

Da eine `Node` Nachfahre von `MinNode` ist, enthält sie natürlich auch deren Felder. Zur Erinnerung, ein `SAMEPTR` zeigt immer auf das Objekt, in dem es verwendet wird, also in einer `MinNode` auf eine `MinNode`, in einer `Node` auf ein `Node`. Die Felder im einzelnen:

succ Zeiger auf vorhergehenden Knoten

pred Zeiger auf nachfolgenden Knoten

type Wie Sie sicher schon denken können, erben alle möglichen Systemstrukturen, die in Listen verwaltet werden, von `Node` oder einem ihrer Nachfolger, so daß man immer eine Node am Anfang einer jeden Struktur hat. Um nun näher anzugeben, um welche Art von Node es sich handelt, dient das Feld `type` vom Aufzählungstyp `NodeType`. `typ` sollte bei der Initialisierung bereits angegeben werden.

```
TYPE
  NodeType = ( unknown,    task,         interrupt,
               device,     msgPort,      message,
               freeMsg,    replyMsg,     resoucre,
               library,    memory,       softInt,
               font,       process,      semaphore,
               signalSem,  bootNode,     kickMem,
               graphics,   deathMessage, user = 254,
               extended );
```


pri gibt die Priorität eines Knotens im Verhältnis zu den anderen Nodes an, manche Systemlisten werden abhängig von der Priorität geordnet, wobei +127 die größte und −128 niedrigste Priorität angibt, meistens wird dieses Feld jedoch gar nicht benutzt.

name Zeiger auf den Namen der Node (ist oft auch NIL), zeigt natürlich auf einen Systemstring mit „&0“ am Ende.

Nachdem wir nun die Einzelheiten einer Liste genau genug angesehen haben, wollen wir nun betrachten wie eine Liste daraus aufgebaut wird.

5.3.2 Aufbau von Systemlisten

Eine Systemliste besteht normalerweise aus einem Listenkopf und angehängten Knoten, so daß es ganz nach einer doppelt verketteten Liste aussieht. In Wirklichkeit sind Systemlisten aber eigentlich Ringlisten (zirkurläre Listen), die Sie schon aus Kapitel 5 her kennen. Allerdings ist das Hilfselement nicht ganz offensichtlich, denn dabei handelt es sich um den Listenkopf. Dieser ist folgendermaßen aufgebaut:

```
TYPE
  List = RECORD
           head,
           tail,
           tailPred : NodePtr;
           type     : NodeType;
           pad      : SHORTCARD
         END;
```

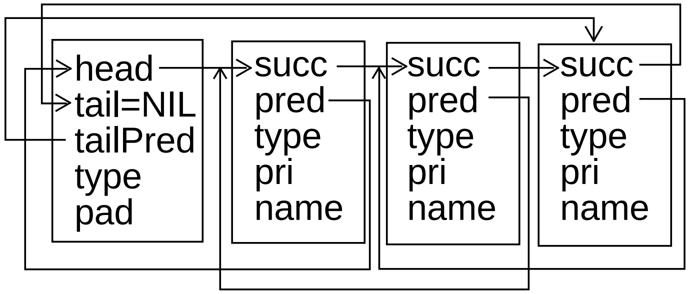
head Pointer auf die erste Node der Liste.

tail ist immer NIL;

tailPred Zeiger auf das letzte Element der Liste

type gibt an, welche Nodes in dieser Liste sind

pad hat keine Funktion, sondern ist nur da, damit die Länge stimmt

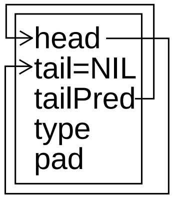


Die einzelnen Elemente sind über die Felder `succ` und `pred` miteinander verkettet, wobei `succ` des letzten Knotens auf den Eintrag `tail` des Listenkopfes zeigt. So wird innerhalb des Listenkopfes ein Schlußelement (Sentinel) gebildet, dessen `succ`-Feld, also das `tail`-Feld des Listenkopfes, NIL enthält. Das `type`-Feld des Listenkopfes ist das `type`-Feld des Sentinels und das scheinbar sinnlose `pad`-Feld des Listenkopfes entpuppt sich als `pri`-Feld des Sentinels. Somit könnte man den Listenkopf auch so definieren:

```
TYPE
  List2 = RECORD
            head     : NodePtr;
            sentinel : Node;
          END;
```

Die List2 hat lediglich das `name`-Feld zuviel, das bei einem Sentinel sowieso nicht verwendet wird. Obige Darstellung soll aber nur verdeutlichen, wie die Liste aussieht, wenn man versucht sie als normale, doppelt verkettete Liste zu interpretieren.

Achtung: Eine Liste ist nicht leer, wenn `head` und `tailPred` NIL sind, wie man vielleicht annehmen könnte; sie ist leer, wenn `head` auf den Eintrag `tail` des Listenkopfes zeigt und `tailPred` auf `head`. Das ist logisch, wenn man sich List2 anschaut, da die Systemlisten eben immer diesen inhaltslosen Sentinel am Ende haben. Siehe Skizze:



Da Listen dynamischer Art sind werden alle Strukturen, die auf Knoten und Listen aufbauen über Pointer angesprochen, also ohne **VAR**-Parameter. Sie sollten generell dynamisch über das Modul T_Exec arbeiten (siehe Kapitel T_Exec), da dieses Modul für automatische Beseitigung des Datenmülls sorgt und konsistente Konstruktionsfunktionen bereitstellt.

Da die Handhabung der Systemlisten nicht ganz einfach ist, stellt uns das Exec Modul und die exec.library (nicht alles stammt aus der library) einige Routinen für die Arbeit mit Listen zur Verfügung.

Um eine neue Node in eine Liste einzutragen, stehen Ihnen folgende Funktionen zur Verfügung:

```
PROCEDURE NewList(VAR list : List;
                      type : NodeType );

PROCEDURE Enqueue( list IN A0 : ListPtr;
                   node IN A1 : NodePtr);

PROCEDURE Insert( list     IN A0 : ListPtr;
                  node     IN A1,
                  listNode IN A2 : NodePtr );

PROCEDURE AddHead( list     IN A0 : ListPtr;
```

```
                     node      IN A1 : NodePtr );

PROCEDURE AddTail( list      IN A0 : ListPtr;
                     node      IN A1 : NodePtr);
```

NewList initialisiert eine Variable `list` vom Typ List als Listenkopf, so daß Sie danach eine leere Liste für Knoten vom typ `type` haben.

Enqueue fügt `node` prioritätssortiert in die Liste `list` ein. Praktisch bedeutet es, daß `node` vor den ersten Knoten in der Liste mit einer niedrigeren Priorität eingefügt wird.

Insert fügt den Knoten `node` in die Liste `list` hinter dem Knoten `listNode` ein.

AddHead fügt `node` als erstes Element von `list` ein.

AddTail fügt `node` am Listenende ein.

Um ein Knoten aus einer Liste zu entfernen existieren entsprechende Funktionen:

```
PROCEDURE Remove( minNode IN A1 : MinNodePtr );

PROCEDURE RemHead( list IN A0 : ListPtr ): NodePtr;

PROCEDURE RemTail( list IN A0 : ListPtr ): NodePtr;
```

Remove entfernt `minNode`[5] aus der Liste. Sie sollten sich trotzdem eine Referenz auf den Knoten behalten, da er sonst verlorener Speicher wird.

RemHead entfernt den ersten (`list^.head^`) Knoten aus der Liste `list`. Wenn `list` leer ist, wird NIL als Ergebnis zurückgegeben, ansonsten einen Zeiger auf den entfernten Knoten.

[5] Diese Funktion funktioniert auch mit Knoten vom Typ `Node`

RemTail wie RemHead, jedoch mit dem letzten Knoten in `list`.

Wollen Sie eine bestimmte Node nach ihrem Namen suchen, sollten Sie die Funktion `FindName` verwenden:

```
PROCEDURE FindName( list      IN A0 : ListPtr;
                      name      IN A1 : SysStringPtr ): NodePtr;
```

list Liste, in der gesucht werden soll

name Zeiger auf den Namen, nach dem gesucht werden soll.

Nach so viel Theorie wollen wir uns eine Liste in der Praxis ansehen. Wie schon erwähnt, gibt es im Modul Exec eine Struktur, die sogenannte ExecBase, in der die Köpfe einiger Systemlisten stehen. Unter anderem auch eine Liste aller Libraries. Mit dem untenstehenden Programm können Sie die Namen aller vorhandenen Libraries ausgeben.

```
MODULE WriteLibNames;
FROM Exec    IMPORT NodePtr, ExecBase,Forbid,Permit;
FROM InOut   IMPORT WriteString, WriteLn;
FROM Strings IMPORT Str;
VAR p  : NodePtr;
    ch : STRINGPTR;
BEGIN
  Forbid;
  p:=ExecBase^.libList.head;
  WHILE p^.succ#NIL DO
    WriteString( Str( p^.name ) );WriteLn;
    p:=p^.succ
  END;
  Permit;
END WriteLibNames.
```

Achtung: Dieses Programm soll lediglich dazu dienen, die Verwendung von Exec-Listen zu demonstrieren. Nur erfahrene Programmierer sollten direkt auf die ExecBasestruktur zugreifen, für alle andere bedeutet dies: Finger weg!

5.3.3 Prozesse

Auch wenn vorher schon erwähnt, möchte ich noch mal kurz erklären, worum es beim Ausführen von Prozessen im Multitasking eigentlich geht.

Multitasking bedeutet, daß mehrere Programme scheinbar gleichzeitig arbeiten. In Wirklichkeit schaltet das Betriebssystem aber nur zwischen den einzelnen Programmen hin und her, so daß jeder Prozeß eine seiner Priorität entsprechende Rechenzeit erhält. Das ganze geht jedoch so schnell vor sich, daß der Anwender in der Regel nichts davon merkt, es sei denn es laufen sehr viele Prozesse gleichzeitig.

Im Amiga gibt es zwei Arten von Prozessen, den Task und den Process (bitte auf die Schreibweise achten: Prozeß ist allgemein, Process ist eine Amiga Systemstruktur). Tasks sind die grundlegende Verwaltungsstruktur für das Multitasking, Processe sind eine Erweiterung davon, um Tasks den Zugang zum Dos zu ermöglichen (siehe Kapitel über Dos).

Nun gibt es prinzipiell zwei Möglichkeiten einen Prozeß zu starten. Zum einen durch Start eines Programmes von der Workbench aus oder von der SHELL mit `run` (auch wenn dabei eigentlich nicht nur ein Task sondern auch ein Process gestartet wird, doch dazu später). Zum anderen, indem Sie eine Prozedur ihres Programmes als eigenen Task erklären. Von den zwei Möglichkeiten, interessiert uns jetzt nur die letztere:

Um nun einen eigenen Task zu erzeugen, muß man eine entsprechende Struktur initialisieren und diese dann mittels `AddTask()` aus Exec ins System einbinden.

Da die meisten bei den ersten Versuchen einen eigenen Task zu erzeugen die erstaunlichsten Orientreisen unternommen haben oder der Amiga zeitweise gar nicht mehr aus dem Meditieren herauskam, waren wir der Meinung, daß man das keineswegs jemanden zumuten könne, der gerade erst mit dem Programmieren begonnen hat.

Aus diesem Grund bereicherten wir `T_Exec` um eine Funktion, die Ihnen die gesamte Initialisierung abnimmt:

```
PROCEDURE CreateTask(REF name      : STRING;
                         priority  : SHORTINT;
                         initPC    : ANYPTR;
```



```
                         stackSize : LONGINT    := 20000;
                         context   : ContextPtr := NIL):TaskPtr;
```

name Name des neuen Tasks. Da es sich hier nicht um eine Systemfunktion handelt, können Sie hier einen gewöhnlichen String übergeben.

priority Priorität des neuen Tasks. Bei einer Priorität von über 50 kommt im Prinzip kein anderer Task mehr zum Zuge, bei unter -50 bekommt ihr eigener Task nur Rechenzeit, wenn kein anderer Task Rechenzeit benötigt. Geben Sie Ihrem Task am besten eine Priorität von Null oder Eins.

initPC Hier tragen Sie ihre Prozedur ein, die Sie als eigenen Task wünschen. Achtung, die Prozedur darf keine Parameter erwarten.

stackSize Größe des Stacks in Bytes, den Ihr Task erhalten soll. Bei einem Wert kleiner 100 wird eine **`EXCEPTION`** ausgelöst. Am sinnvollsten ist es hier normalerweise einen Wert zwischen 500 und 1000 anzugeben. Hat ihr Task viele lokale Prozeduren oder ruft er Prozeduren rekursiv auf, kann es sein, daß Sie einen noch größeren Wert wählen müssen.

context Kontext, zu dem der Task erzeugt werden soll. Wird dieser Parameter nicht angegeben, wird der Task zum aktuellen Kontext erzeugt. Näheres zu Kontexten siehe Beschreibung des Moduls `Resources`

Ergebnis Zeiger auf die initialisierte und eingebundene Taskstruktur.

Die TaskStruktur, die von CreateTask erzeugt wird und die von Exec zur Verteilung der Systemzeit und zur grundlegenden Kommunikation zwischen Tasks verwendet wird, sieht folgendermaßen aus:

```
TYPE
  Task = RECORD OF Node
           flags      :TaskFlagSet; | Eigenschaften des Tasks
           state      :TaskState;   | Zustand des Tasks
           idNestCnt,               | Zähler für DISABLE
```

```
    tdNestCnt  :SHORTINT;    | Zähler für FORBID
    sigAlloc,                | allozierte Signale
    sigWait,                 | erwartet werdende Signale
    sigRecvd,                | empfangene Signale
    sigExcept  :TaskSigSet;  | Signale, die Exception auslösen
    trapAlloc,               | Traps die belegt sind
    trapAble   :BITSET;      | nicht belegte Traps
    exceptData :ANYPTR;      | Daten für Exception
    exceptCode :ExceptPROC;  | Code für Exception
    trapData   :ANYPTR;      | Daten für Traps
    trapCode   :PROC;        | Code für Traps
    spReg,                   | Stackpointer des Tasks
    spLower,                 | Stackuntergrenze
    spUpper    :ANYPTR;      | Stackobergrenze
    switch,                  | Abgaberoutine
    launch     :PROC;        | übernahmeroutine
    memEntry   :List;        | Speicherliste des Tasks
    userData   :ANYPTR;      | Frei für Userroutinen
  END;
```

flags Set mit Informationen über den Task, enthält er `except`, befindet er sich gerade in einem Ausnahmezustand, ist das Flag `switch` gesetzt, wird vor dem Umschalten zu einem anderen Task noch die Routine im Feld `switch` angesprungen, ist `launch` gesetzt, wird die Routine im Feld `launch` aufgerufen, bevor der eigentliche Task bearbeitet wird.

state Gibt den momentanen Zustand des Tasks an (siehe unten).

sigAlloc Zur Verständigung zwischen den Tasks existieren 32 Signale in einem Set, wovon die unteren 16 für das System reserviert sind. In diesem Feld sind die Signale, die von diesem Task belegt werden. Achtung: Jedem Task stehen also maximal 16 eigene Signale zur Verfügung.

sigWait Signale, auf die der Task im Moment wartet.

sigRecvd Hier werden alle empfangenen Signale eingetragen. Steht ein empfangenes Signal auch in `sigWait`, wird der Task in den `ready`-Zustand versetzt.

sigExcept Signale, die bei Empfang eine Exception auslösen (auch wenn der Task gerade läuft). Das bedeutet, daß die Funktion in `exceptCode` aufgerufen wird und `exceptData` als Zeiger auf private Daten verwenden kann (vor allem interessant, wenn der Code nicht zum geladenen Programm gehört).

trapAlloc Bits der Traps, die der Task für sich in Anspruch nimmt. Traps werden mit `AllocTrap` und `FreeTrap` wie Signale alloziert und freigegeben. Wenn der Prozessor auf eine Trap-Instruktion trifft, wird die eigene Fehlerroutine in `trapCode` aufgerufen. Traps werden von Cluster direkt verwaltet. Für Programmierer heißt dies: Finger weg!.

trapAble Wird vom System verwaltet.

switch Wenn in `flag` das `switch`-bit eingetragen ist, wird die hier eingetragene Prozedur vor dem Umschalten auf einen anderen Task angesprungen.

launch Wenn in `flag` das `launch`-bit eingetragen ist, wird die hier eingetragene Prozedur vor dem Aktivieren des eigenen Tasks angesprungen.

memEntry Speicherliste des Tasks, die am Ende wieder freigegeben wird.

Die Felder `sigAlloc`, `sigWait`, `sigRecvd`, `trapAlloc` und `trapAble` sind nicht vom Task zu verwenden, sondern die entsprechenden Funktionen (siehe Tasksignale).

Der Task wird vom System vom einem Zustand in den anderen versetzt, wobei das den Anwendertask nicht zu interessieren hat, sondern höchstens einen Debugger. Damit der Abschnitt über Tasks komplett ist, sind hier auch noch die möglichen Zustände eines Tasks aufgeführt.

inval Task ist ungültig, d. h. mit großer Wahrscheinlichkeit nicht mehr lauffähig.

added Der Task wurde soeben hinzugefügt und ist noch nicht ablaufbereit.

run Der Task wir gerade abgearbeitet (muß wohl der eigene Task sein, soweit es nur eine CPU im System gibt ...).

ready Der Task ist ablaufbereit, wartet aber, bis er dran kommt.

wait Der Task befindet sich im Wartezustand und wartet auf ein Signal.

except Der Task ist gerade in einer Ausnahmesituation.

removed Der Task wird gerade entfernt.

Nach dem Aufruf von `CreateTask()` beginnt die übergebene Routine selbständig zu arbeiten. Ein großer Vorteil gegenüber der Methode von Hand einen Task zu initialisieren ist, daß die mit `CreateTask` erzeugten Task nach Möglichkeit alle Gurus abfangen. Außerdem werden alle Task mit Beendigung des Hauptprogramms auch beendet.

Wollen Sie einen Task vor dem Ende des Hauptprogramms abschalten, dann benutzen Sie dazu `DeleteTask`:

```
PROCEDURE DeleteTask( tsk : TaskPtr );
```

Wichtig: Versuchen Sie nie einen mit `CreateTask()` erzeugten Task mit `RemTask()` aus Exec zu entfernen, für die Folgen übernehmen wir nämlich keinerlei Haftung. Dasselbe gilt, wenn man einen von Hand erzeugten Task mit `DeleteTask()` zu entfernen versucht.

Wenn Sie wissen wollen, wie man einen Task selbst initialisiert, dann schauen Sie sich am besten den Quelltext von `T_Exec.mod` an oder kaufen Sie sich das RKM und lesen nach, wie „C"-Programmierer von Hand im Speicher herumwühlen.

Wenn Sie Strukturen untersuchen oder verändern, die auch von anderen Prozessen verändert werden, empfiehlt es sich so lange das Multitasking mit der Funktion `Forbid()` auszuschalten. Wenn Sie fertig sind, schalten Sie es bitte mit der Funktion `Permit()` so schnell wie möglich wieder an (spätestens am Programmende), weil Sie sonst ihren Amiga zu einem Primitiv-PC degradieren, der immer nur ein Programm gleichzeitig ausführen kann.

Wenn Sie auch die Systeminterrupts[6] sperren müssen, benutzen Sie die Prozedur `Disable()`, und `Enable()`. Die Systeminterrupts brauchen Sie nur auszuschalten, wenn Sie an Strukturen arbeiten, die von Interrupts verändert werden, wie zum Beispiel die Taskliste.

Vielleicht werden Sie nun fragen, warum sollen zwei Prozeduren gleichzeitig arbeiten, wo ich doch eh nur eine zur gleichen Zeit bedienen kann.

Da mögen Sie vielleicht schon recht haben, aber stellen wir uns mal vor, wir wollten ein Textverarbeitungsprogramm schreiben. Wäre es nicht toll, wenn wir die Möglichkeit hätten, während wir einen Text eingeben, einen anderen auszudrucken?

Wichtig: Von einem Programm erzeugte Tasks unterliegen einer Einschränkung. Sie können nicht auf das Dos zugreifen. Also auch keine Ausgaben auf die SHELL mittels InOut machen. Wenn Sie dies dennoch versuchen, landen sie bald in Indien. Auf alle anderen Libraries können Sie jedoch zugreifen.

Wollen Sie von einem Task aus das Dos nutzen, reicht dazu kein normaler Task, sondern Sie brauchen einen Process, eine Erweiterung der Taskstruktur, doch dazu mehr im Kapitel über Dos.

5.3.3.1 Taskfunktionen

Hat man einen Task erzeugt, bietet Exec verschiedene Prozeduren an, um mit diesen zu arbeiten.

```
PROCEDURE FindTask( name IN A1 : SysStringPtr ): TaskPtr;

PROCEDURE SetTaskPri( task IN A1 : TaskPtr;
                      pri IN D0 : SHORTINT ): SHORTINT;
```

FindTask sucht den Prozeß (egal ob Task oder Process), der den Namen `name` trägt. Rückgabewert ist die Adresse des Prozesses oder

[6] Was ein Interrupt ist erfahren Sie in Kürze

NIL, wenn er nicht gefunden wurde. Wollen Sie die Adresse Ihres eigenen Tasks ermitteln, übergeben Sie für `name` einfach `NIL`.

SetTaskPri Setzt die Priorität des Prozesses `task` auf `pri` und gibt die bisherige Priorität als Rückgabewert zurück.

5.3.4 Das Nachrichtennetz der Tasks

Sie haben im letzten Kapitel gelernt, wie man eigene Tasks erzeugt. Tasks bringen jedoch nur dann etwas, wenn Sie sich untereinander verständigen oder Daten austauschen können.

Zwar könnte man dies über globale Variablen erreichen, aber stellen Sie sich einmal vor, ein Task wartet darauf, das ein anderer einen Wert in eine globale Variable schreibt und deshalb dauernd diese Variable ausliest. Damit würde eine Menge Rechenzeit vergeudet, die andere Tasks verwenden könnten. Aus diesem Grund stellt das Betriebssystem uns ein spezielles Nachrichtennetz zur Verfügung.

Die grundlegende Kommunikation läuft über die sogenannten Signale. Ein Task kann auf ein oder mehrere Signale warten, oder einem anderen Task ein Signal schicken und diesen somit „wecken". Die empfangenen Signale sind wie oben beschrieben, Bits in einem Set. Diese werden mit Bits in einem anderen Set, der die zu erwartenden Signale beschreibt, verglichen und entschieden, ob der Prozeß aus dem Wartezustand „aufgeweckt" werden kann. Kommt dabei ein Signal sehr oft nacheinander (z. B. von mehreren Prozessen höherer Priorität), dann wird der Prozeß nur einmal geweckt. Das heißt, daß man über Signale keine Informationen verschicken kann, sondern nur Ereignisse signalisieren (daher der Name . . .).

Aus diesem Grunde gibt es neben Signalen Messages, die in einer Liste einer Empfangstation (MsgPort) aufgefangen werden. Das Verschicken einer solchen Nachricht (mit Information) besteht also aus dem Eintragen der Nachricht in eine Empfangsliste und dem Aktivieren eines dafür vorgesehenes Signals. Der Empfänger liest dann die Empfangsliste, bis sie leer ist und wartet dann eventuell auf neue Nachrichten. So wird gewährleistet, daß pro verschickter Information der empfangende Prozeß eine Aktion unternimmt.

Diese beiden Teile der Kommunikation über die exec.library wird in den folgenden beiden Unterabschnitten ausführlich beschrieben.

5.3.4.1 Tasksignale

Im Amiga Betriebssystem besitzt jeder Prozeß 32 Signale, wovon die unteren 16 für das System reserviert sind. Die oberen 16 stehen jedem Prozeß zur freien Verfügung, was bedeutet, daß jeder Prozeß selber wissen muß, was für ihn ein bestimmtes Signalbit bedeutet. Man kann einen Task auf eine Menge von Signalen warten lassen oder eine Menge von Signalen bei sich selber oder einem anderen Prozeß setzen. Um sicherzugehen, das kein Signalbit doppelt verwendet wird, werden die Signalnummern für jeden Prozeß verwaltet. Wenn eine Signalnummer gebraucht wird, muß sie vorher angemeldet werden und wenn sie nicht mehr verwendet wird, kann sie abgemeldet werden. So wird sichergestellt, daß immer auf das richtige Ereignis gewartet wird. Folgende Signale gibt es:

```
TaskSignals   = ( noSignal  = -1,
                  anySignal = -1,
                  abort,        child,        ts2,          ts3,
                  blit,         single=4,     intuition,    ts6,
                  ts7,          dos,          ts9,          ts10,
                  ts11,         ctrlC,        ctrlD,        ctrlE,
                  ctrlF,        user16,       user17,       user18,
                  user19,       user20,       user21,       user22,
                  user24,       user25,       user26,       user27,
                  user28,       user29,       user30,       user31 );
TaskSigSet    = SET OF TaskSignals;
```

Dabei sind `user16`–`user17` zur eigenen Verfügung.

Die Verwaltung der Signalnummern wird mit zwei einfachen Funktionen bewerkstelligt, die nicht aufgerufen werden dürfen, während der Prozeß im „exception"-Zustand ist.

```
PROCEDURE AllocSignal( signalNum IN D0 : TaskSignals ):TaskSignals;

PROCEDURE FreeSignal( signalNum IN D0 : TaskSignals );
```

AllocSignal Versucht das Signal `signalNum` oder ein beliebiges, freies Signal (`anySignal`) zu belegen und gibt die Nummer des belegten Signals oder `noSignal` zurück, wenn der Anmeldeantrag abgelehnt wurde. Normalerweise übergibt man immer -1, da man dadurch sich nicht darum zu kümmern braucht, welche Signale schon belegt sind und welche nicht.

FreeSignal Gibt das Signal `signalNum` wieder frei.

Die Kommunikation läuft nun über das Verschicken von und warten auf Signale.

```
PROCEDURE SetSignals( newSignals  IN D0,
                      signalMask  IN D1 : TaskSigSet ): TaskSigSet;

PROCEDURE Signal( task    IN A1 : TaskPtr;
                  signals IN D1 : TaskSigSet );

PROCEDURE Wait( signals IN D0 : TaskSigSet ): TaskSigSet;
```

SetSignals setzt die Signale im Feld `sigRecvd` des Tasks, die in `signalMask` gesetzt sind, auf den entsprechenden Wert in `newSignals`. Damit kann man gezielt Signale setzen (sich selber ein Signal schicken) oder löschen (nach Bearbeitung). Als Ergebnis erhält man die vorherige Signalbelegung von `sigRecvd`. Ein beliebter Aufruf ist `SetSignals({},{})`, um alle empfangenen Signale zu lesen, jedoch keines zu verändern. Um beispielsweise alle empfangenen Signale zu löschen muß man nur für `newSignal` ein leeres Set und für `signalMask` die empfangenen Signale übergeben.

Signal schickt dem Prozeß `task` alle Signale in `signals`.

Wait wartet auf alle Signale im Set `signals`. Das heißt natürlich, daß der Prozeß geweckt wird, sobald irgendeins der Signale ankommt. Das Ergebnis ist die Menge der empfangenen Signale. War vor dem Aufruf von `Wait` im Feld `SigRcvd` der Taskstruktur schon ein zu erwartendes Signal gesetzt, wird der Prozeß nicht erst in den Wartezustand versetzt, sondern darf gleich weiterarbeiten. Die empfangenen Signale werden nicht automatisch gelöscht, sondern müssen mit `SetSignals` explizit gelöscht werden, damit der Prozeß beim nächsten Warten nicht in eine Endlosschleife verfällt.

Wenn ein selbsterzeugter Task an den Hauptask Signale senden will, benötigt er dazu dessen TaskPtr. Dieser sollte vom HauptTask mittels `FindTask(NIL)` ermittelt werden und dem Untertask mittels einer Message übermittelt werden.

5.3.4.2 Messages

Mit Signalen können wir nun die Tasks untereinander signalisierend kommunizieren lassen. Was aber, wenn man mehr Informationen benötigt und man nicht über globale Variablen gehen kann (sowieso nicht die feine Art), z. B. wenn zwei getrennte Programme sich verständigen sollen. Ja, Sie haben ganz richtig gelesen, zwei Programme können recht einfach miteinander in Verbindung treten. Dies geschieht immer, wenn man Informationen vom Betriebssystem erhalten will, z. B. wenn man die Tastatur oder die Maus abfragen möchte.

Hierzu dienen sogenannte MessagePorts oder Nachrichtenempfangsstationen, mit denen man richtige Datenkanäle aufbauen kann. Auch hierzu benötigt man eine Systemstruktur, die erst erzeugt werden muß.

```
TYPE
  MsgPort = RECORD OF Node
              IF KEY flags : MsgPortAction
              OF signal  THEN
                sigBit  : TaskSignals;
                sigTask : TaskPtr
              OF softint THEN
                softInt : InterruptPtr
              END;
              msgList : List;
            END;
```

flags Gibt an, ob der Port auf Signal- oder Interruptbasis arbeitet. Der Rest der Struktur sieht je nach Wert verschieden aus.

sigBit Wenn `flags=signal`, enthält dieses Feld die Signalnummer, die der wartende Prozeß bekommt, wenn eine Nachricht am Port ankommt.

sigTask Der Prozeß der `sigBit` bekommt, wenn eine Nachricht ankommt.

softint Der Interrupt, der ausgelöst wird, wenn eine Nachricht ankommt und `flags=softInt` ist.

msgList Liste der erhaltenen Nachrichten (FIFO). Die Knoten dieser Liste sollten alle vom Typ `message` oder `replyMsg` sein.

Bei den MsgPorts muß man unterscheiden zwischen privaten und öffentlichen MsgPorts. In der ExecBase gibt es eine Liste von öffentlichen MsgPorts, mit denen jeder Prozeß kommunizieren kann. Private MsgPorts können nicht von anderen Prozessen gefunden werden, können also nur für eigene Zwecke verwendet werden (z. B. IDCMP, siehe hierzu Intuition).

Exec bietet seit V36 Funktionen an, um MsgPorts zu initialisieren und zu zerstören. Da es sich dabei immer nur um private MsgPorts handelt, sind diese nur begrenzt zu verwenden. Das Modul `T_Exec` bietet dafür eine Funktion zur Erzeugung von privaten als auch öffentlichen MsgPorts. Der Unterschied besteht im Aufruf der Funktion.

```
PROCEDURE CreatePort(REF portName : STRING     :="";
                         priority : SHORTINT   := 0;
                         context  : ContextPtr := NIL):MsgPortPtr;
```

portname Name des Ports. Hier kann man auch einen leeren String angeben. Will man aber zwei Programme miteinander in Verbindung treten lassen, sollte man einen möglichst eindeutigen Namen angeben, denn anhand des Namens kann man den Port im System wiederfinden, wenn man den Pointer auf ihn nicht kennt.

priority Priorität des Ports. Die Portliste wird ebenso wie die Prozeßliste der Priorität geordnet.

context Kontext, zu dem der Port erzeugt wird.

Ergebnis Zeiger auf die initialisierte Portstruktur.

Da für die meisten Anwendungen sowieso Ports nur mit Signalen arbeiten, haben wir uns dafür entschieden, daß `CreatePort` einen Port auf Signalbasis und immer für den gerade laufenden Task erzeugt. Haben Sie mit `CreatePort()` einen Port erzeugt, können Sie diesen mit `DeletePort` aus `T_Exec` wieder vernichten.

Nachdem wir uns die Empfangsstation für Nachrichten angesehen haben, wenden wir uns der Nachricht selber zu. Die eigentlichen Daten, die mit der Message verschickt werden sollen, werden einfach hinter die Messagestruktur angehängt. In Cluster erbt man einfach von `Message`. Folgendes Beispiel zeigt dies deutlich:

```
TYPE
  Message  = RECORD OF Node
               replyPort : MsgPortPtr;
               length    : CARDINAL;
             END;

  MyString = STRING(20);

  MyMsg    = RECORD OF Message
               str : MyString
             END;
```

replyPort MsgPort, an die die Message nach Erhalt zurückgeschickt werden soll. Wenn man also eine Nachricht verschicken möchte, benötigt man auch einen Port, um die Nachricht wieder aufzufangen. Dieser muß vor der Versendung in dieses Feld eingetragen werden. Erst wenn die Message zurückgeschickt wurde, darf man davon ausgehen, daß der andere Task sie bearbeitet hat. Um sie z. B. erneut zu verschicken.

length Länge der Nachricht inklusive der eigentlichen Daten (maximal 64KB). Im Beispiel MyMsg also `Message'SIZE + str'SIZE`, bzw. `MyMsg'SIZE`.

Beide Felder müssen vor dem Verschicken unbedingt initialisiert werden.

Die Nachrichten werden mit `PutMsg()` zu einem MsgPort geschickt. Ist der bearbeitende Prozeß fertig mit dem Lesen (und eventuell Verändern) der Nachricht, so kann dieser die Nachricht mit `ReplyMsg` beantworten. Die Nachricht geht dann zurück an den Sender, wobei in das `type`-Feld der Node der Message ein `freeMsg` eingetragen wird. So weiß der erste Prozeß, daß er mit der Nachricht tun und lassen kann, was er will.

Wichtig: Die Nachricht wird nicht in den Speicher des empfangenden Tasks kopiert, dieser kann lediglich auf den Speicherbereich des sendenden Tasks zugreifen. Daher sollten immer nach Empfang einer Nachricht die benötigten Daten in eigene Variablen kopiert werden und die Nachricht zurückgeschickt werden. Es könnte sonst sein, daß der sendende Task unnötig wartet.

```
PROCEDURE PutMsg( port IN A0 : MsgPortPtr;
                  msg  IN A1 : MessagePtr );

PROCEDURE GetMsg( port IN A0 : MsgPortPtr ): MessagePtr;

PROCEDURE WaitPort( port IN A0 : MsgPortPtr ): MessagePtr;

PROCEDURE ReplyMsg( msg IN A1 : MessagePtr );
```

PutMsg schickt die Nachricht `msg` zu dem MsgPort `port`, reiht die Nachricht in die Nachrichtenliste ein und schickt dem empfangenden Prozeß das zugehörige Signal.

GetMsg holt eine Nachricht von der Empfangstation `port` und gibt die Adresse der Nachricht zurück. Gibt es keine neuen Nachrichten, so wird NIL zurückgegeben. Will man auf eine Nachricht warten, sollte man jedoch nicht mit einer WHILE-Schleife den MsgPort abfragen, hierfür existiert `WaitPort()`, welches keine Rechenzeit vergeudet.

WaitPort wartet auf eine neue Nachricht und gibt die Adresse der ersten neuen Nachricht als Ergebnis zurück. Die Nachricht muß dennoch mit GetMsg aus der Nachrichtenliste entfernt werden. Wenn schon eine neue Nachricht da war, wird der Prozeß nicht erst in den Wartezustand versetzt, sondern bekommt gleich das Ergebnis zurück.

ReplyMsg schickt `msg` zum replyPort zurück. Wenn kein replyPort eingetragen ist (also NIL), dann wird `freeMsg` in das `type`-Feld der Node der Message eingetragen. Danach darf man nicht mehr auf die Nachricht zugreifen.

Wollen Sie an einen Port eine Nachricht senden, von dem Sie nur den Namen kennen, nicht aber dessen PortPtr, dann können Sie diesen über FindPort erhalten:

```
PROCEDURE FindPort( REF name IN A1 : STRING ): MsgPortPtr;
```

In Verbindung mit Intuition werden Sie noch einmal mit Messageports konfrontiert werden.

5.3.5 Semaphoren

In jedem Multitaskingsystem ist es notwendig Daten unter unabhängig arbeitenden Tasks zu teilen. Solange die Daten statisch sind (d. h. sie werden nicht verändert) ist dies kein Problem. Sobald sie aber veränderbar sind, muß es eine Möglichkeit für den Task geben, der die Daten bearbeitet, andere Tasks daran zu hindern, auf die Daten gleichzeitig zuzugreifen.

Z. B. um einen Knoten in eine Liste einzuhängen würde ein Task normalerweise diese einfach einhängen. Können jedoch mehrere Tasks auf diese Liste zugreifen, kann dies sehr gefährlich sein: Ein anderer Task könnte gerade dabei sein, sich an der Liste entlangzuhangeln und einen ungültigen Zeiger auslesen. Schlimmer wird es, wenn zwei Tasks an der selben Stelle zur gleichen Zeit einen Knoten einhängen wollen. Für dieses Problem stellen Semaphoren die ideale Lösung da.

Man kann sich eine Semaphore ungefähr wie einen Schlüssel zu verschlossenen Daten vorstellen. Wenn Sie den Schlüssel (die Semaphore) haben, können sie mit den Daten machen was Sie wollen, ohne daß Ihnen ein anderer Task in die Quere kommen kann. Jeder andere Task, der diese Semaphore beansprucht, wird augenblicklich in den Wait-Status versetzt wird, bis die Semaphore wieder freigegeben wird. Wenn Sie Ihre Arbeit beendet haben, müssen Sie deshalb sobald als möglich die Semaphore wieder freigeben.

Achtung: Sempahoren können nur dann zuverlässig arbeiten, wenn folgende zwei Bedingungen von allen Tasks eingehalten werden:

- Alle Tasks, die gemeinsame Daten haben, die durch eine Semaphore gesichert sind, müssen diese zuerst beantragen, bevor sie auf die Daten zugreifen können. Wenn irgendein Task direkt auf die Daten zugreift, ohne über die Semaphore zu gehen, können die Daten ungültig werden, kein Task kann dann mehr sicher darauf zugreifen.
- Es können Deadlocks (Kurzschlüsse) entstehen, wenn ein Task, der eine exklusive Semaphore innehat, auf einen anderen Task wartet, der auf die selbe Semaphore wartet. Es ist also äußerste Vorsicht geboten, damit Deadlocks vermieden werden, sie können das gesamte System lahmlegen.

Um eine eigene Semaphore zu erzeugen, muß man folgende Struktur initialisieren:

```
SignalSemaphore     = RECORD OF Node
                            nestCount    : INTEGER;
                            waitQueue    : MinList;
                            multipleLink : SemaphoreRequest;
                            owner        : TaskPtr;
                            queueCount   : INTEGER;
                          END;
```

In dieser Struktur müssen Sie nur das `name` und `pri` Feld der Node initialisieren, den Rest erledigt die Prozedur `InitSemaphore()`:


```
PROCEDURE InitSemaphore( sigSema IN A0 : SignalSemaphorePtr );
```

Nachdem die Semaphore initialisiert wurde, kann man sie der Öffentlichkeit zugänglich machen, indem man sie mittels `AddSemaphore()` in die Liste der Semaphoren einbindet:

```
PROCEDURE AddSemaphore( sigSema IN A1 : SignalSemaphorePtr );
```

Dabei ist darauf zu achten, daß sich noch keine andere Semaphore mit gleichem Namen im System befindet. Dies läßt sich mit der Funktion `FindSemaphore()` herausfinden, mit der man normalerweise eine Semaphore, von der nur der Namen bekannt ist, finden läßt:

```
PROCEDURE FindSemaphore(name IN A1 : SysStringPtr):SignalSemaphore;
```

Befindet sich keine Semaphore mit Namen `name` im System, wird `NIL` zurückgegeben.

Bevor man nun auf die Daten, die von einer Semaphore geschützt werden, zugreifen kann, muß man die Semaphore belegen. Dafür dient die Funktion `ObtainSemaphore()`:

```
PROCEDURE ObtainSemaphore(sigSema IN A0 : SignalSemaphorePtr);
```

War die Semaphore schon belegt, wird Ihr Task in den Wartezustand versetzt. Will man nur lesend auf die Daten zugreifen, kann man auch `ObtainSemaphoreShared()` verwenden. Bei dieser Art der Belegung können auch noch andere Tasks die Semaphore Shared obtainen und die Daten lesen, jedoch kann kein Task mehr ein exklusives Obtain machen:

```
PROCEDURE ObtainSemaphoreShared(sigSema IN A0 : SignalSemaphorePtr);
```

Manchmal möchte man jedoch nicht warten, sondern nur feststellen, ob die Semaphore gerade frei ist. Wenn dies nicht der Fall ist, kann man in der Zwischenzeit etwas anderes machen. In einem solchen Fall hilft die Funktion `AttemptSemaphore()`

```
PROCEDURE AttemptSemaphore(sigSema IN A0 : SignalSemaphorePtr): BOOLEAN;
```

Ist man mit der Bearbeitung der Daten fertig, muß man die Semaphore wieder freigeben, damit die eventuell wartenden Tasks weiterarbeiten können. Hierzu dient `ReleaseSemaphore()`:

```
PROCEDURE ReleaseSemaphore(sigSema IN A0 : SignalSemaphorePtr);
```

Benötigt ein Task mehrere Semaphoren gleichzeitig, kann es sehr leicht zu Deadlocks kommen.

Beispiel: Task A belegt Semaphore X, Task B belegt Semaphore Y. Nun versucht Task A Semaphore Y zu belegen und wird auf *Wait* gesetzt, Task B versucht seinerseits Semaphore X zu belegen und wird auch auf *Wait* gesetzt. Beide Tasks habe keine Möglichkeit aus diesem Zustand herauszukommen. Um etwas derartiges zu vermeiden, existiert die Funktion `ObtainSemaphoreList()`:

```
PROCEDURE ObtainSemaphoreList( list IN A0 : ListPtr );
```

Wenn nicht alle Semaphoren der Liste belegt werden konnten, wird Ihr Task auf *Wait* gesetzt. Achtung, bitte hängen Sie keine öffentlichen Semaphoren in eine eigene Liste ein, um sie dieser Funktion zu übergeben, da hierfür die normalen Node-Felder verwendet werden und die öffenliche Semaphorenliste nicht zerstört werden darf. Also nur mit privaten Semaphoren verwenden bzw. auf die gesamte Semaphorenliste.

```
PROCEDURE ReleaseSemaphoreList( list IN A0 : ListPtr );
```

Gegenstück zu `ObtainSemaphoreList()`.

Bevor man eine Semaphore freigibt, muß man sichergehen, daß sie nicht mehr belegt wird. Dafür sollte man sie zuerst aus der Semaphorenliste entfernen, damit kein anderer Tasks sie mehr finden kann. Dies geschieht mit `RemSemaphore()`:

```
PROCEDURE RemSemaphore( sigSema IN A1 : SignalSemaphorePtr );
```


Danach sollte man sie exklusiv obtainen, damit man sicher ist, daß kein anderer Task sie noch belegt hat. Hat man den Zugriff auf die Semaphore erhalten, gibt man Sie wieder frei und kann danach den Speicher der Semaphorestruktur freigeben. Sie sehen, man sollte vor jedem Zugriff auf eine fremde Semaphore, diese mittels `FindSemaphore()` suchen, anstatt sich die Adresse zu merken. Auf diese Weise geht man sicher, daß die Semaphore auch noch existiert und nicht schon wieder aus dem System entfernt worden ist.

5.3.6 Interrupts in Cluster

An dieser Stelle soll nicht erklärt werden, was ein Interupt ist und wie man sie verwendet, dies würde den Rahmen dieses Handbuches sprengen. Außerdem empfiehlt es sich nur für erfahrenere Programmierer, sich mit Interupts zu beschäftigen. Was hier gezeigt werden soll ist, wie man einen Interuptserver in Cluster in das System einbindet. Hierzu ein Programmfragment als Beispiel:

```
|Checks abschalten
$$StackChk   :=FALSE
$$RangeChk   :=FALSE
$$NilChk     :=FALSE
$$OverflowChk:=FALSE
$$TypeChk    :=FALSE
PROCEDURE MyIntCode;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A4,A6});|Register retten
  SETREG(REG(A1),A4);             |Setzen von A4
  IF -- ist es  mein interrupt ? -- THEN
    |
    | hier liegt der Interruptcode
    |
    SETREG(1,D0)| Der Interupt wird nicht weitergeben
  ELSE
    SETREG(0,D0)| Interupt wird an andere Handler weitergereicht
  END
  POP(RegSet:{D2..D7,A2..A4,A6});| Register wiederherstellen
```

```
END MyIntCode;
|Checks wieder auf alten Zustand
$$StackChk   :=OLD
$$RangeChk   :=OLD
$$NilChk     :=OLD
$$OverflowChk:=OLD
$$TypeChk    :=OLD


VAR MyInt : Exec.Interrupt;

PROCEDURE AddMyInt;
BEGIN
  MyInt.name:="MyInterrupt";
  MyInt.pri :=0; | oder je nach Bedarf
  MyInt.code:=MyIntCode;
  MyInt.data:=REG(A4);

| Interupt einhängen, hier in den vertical Blank
  Exec.AddIntServer(Hardware.vertb,MyInt'PTR);
END AddMyInt;

PROCEDURE RemMyInt;
BEGIN
  Exec.RemIntServer(MyInt'PTR);
END RemMyInt;
```

Im Prinzip funktioniert es wie auch in anderen Programmiersprachen, interessant sind lediglich die Zeilen:

```
  MyInt.data:=REG(A4);
```

Hier wird die Localbase `A4` ausgelesen, über die die Variablen adressiert werden.

```
  SETREG(REG(A1),A4);
```


Hier wird innerhalb des Interrupts die Localbase gesetzt, so daß man auch innerhalb des Interrupthandlers auf alle Variablen des Haupprogramms zugreifen kann. Ansonsten ist darauf zu achten, daß innerhalb des Interrupthandlers alle Compilerchecks ausgeschaltet sind. Weitere Informationen zu Interrupts entnehmen Sie bitte den RKMs.

5.3.7 Speicherverwaltung

Im Gegensatz zu herkömmlichen Computern wie dem C64 kann man beim Amiga nicht davon ausgehen, daß das eigene Programm immer an derselben Stelle im Speicher liegt. Man stelle sich nur einmal vor was passieren würden, wenn beim Programmstart durch daß Multitasking schon ein anderes Programm an dieser Stelle im Speicher liegen würde.

Außerdem kann man auch nicht einfach seine Daten irgendwo in den Speicher schreiben, da auch dort schon ein anderes Programm liegen könnte.

Um das erste Problem brauchen Sie sich nicht zu kümmern, das übernimmt beim Programmstart Dos und Exec.

Benötigen wir jedoch Speicher für dynamische Strukturen, müssen wir diesen von Exec anfordern, wozu Exec uns mehrere Routinen anbietet. Hier das wichtigste und verbreiteste Paar:

```
TYPE
  MemReqs       = ( public,   chip,     fast,
                    mr3,      mr4,      mr5,
                    mr6,      mr7,      local,
                    dma24,    mr10,     mr11,
                    mr12,     mr13,     mr14,
                    mr15,     clear, largest,
                    reverse,  total );

PROCEDURE AllocMem( byteSize     IN D0 : LONGCARD;
                    requirements IN D1 : MemReqSet ):ANYPTR;

PROCEDURE FreeMem(  memoryBlock IN A1 : ANYPTR;
                    byteSize    IN D0 : LONGCARD );
```

MemReqs Über diese Flags kann man angeben, welcher Art der belegte Speicher sein soll:

public „Öffentlich“, der Speicherbereich kann von mehreren Tasks benutzt werden. Er darf nicht umgemapped, ausgelagert oder unadressierbar gemacht werden. Aller Speicher, der von Interrupts oder von anderen Tasks benutzt werden kann (z. B. Messages), muß diesen Typ haben.

chip Der zu belegende Speicher muß im ChipMem liegen, da nur dort die Customchips darauf zugreifen können. Dies ist wichtig für Sound- und Grafikdaten.

fast Der Bereich muß im Fastram liegen. Da die Customchips auf diesen Speicher nicht zugreifen können, wird der Prozessor nicht gebremst.

Wenn der Amiga keinen Fastram besitzt, kann natürlich kein Fast-Speicher belegt werden! Deshalb geben Sie in der Regel weder chip noch fast an. Exec wird ihnen FastMemory geben, und falls keines mehr frei ist, auf ChipMemory ausweichen.

clear Der Bereich wird beim Belegen erst noch gelöscht.

largest Es wird der größte Speicherbereich, für den die angegebenen Bedingungen zutreffen, belegt.

dma24 Belegt Speicher im 24-Bit-Adreßraum, der DMA-Fähig ist.

total Für `AvailMem()`, gibt die Summe des freien Speichers zurück.

AllocMem Belegt einen Speicherblock. Sie geben in `byteSize` die Größe des gewünschten Blocks an und mittels `requirements` die benötigte Art und Eigenschaften, siehe oben.

Als Ergebnis erhalten Sie einen Zeiger auf den Anfang des reservierten Speicherblocks oder NIL, wenn es nicht möglich war, einen Block dieser Größe mit diesen Bedingungen zu belegen.

FreeMem Gibt belegten Speicherplatz wieder frei. Übergeben wird ein Zeiger auf den freizugebenden Block und die Anzahl der Bytes, die er umfaßt. Der Block darf nicht geteilt werden, also muß die Anzahl der Bytes gleich groß sein wie beim Belegen des Blockes.

Wichtig: Spätestens am Programmende müssen Sie alle belegten Blöcke wieder freigeben, denn sonst würde bei mehrmaligem Starten solcher Programme der Speicher bald zu Ende sein.

Vielleicht möchten Sie wissen, wieviel Speicher überhaupt noch zu Verfügung steht oder was für Speicher Sie tatsächlich bekommen haben? Diese Funktionen geben darüber Auskunft:

```
PROCEDURE AvailMem( requirements IN D1 : MemReqSet ):LONGCARD;

PROCEDURE TypeOfMem( address IN A0 : ANYPTR ):MemReqSet;
```

AvailMem Ermittelt den noch freien Speicher für den bestimmte Bedingungen zutreffen. Ergebnis ist die entsprechende Anzahl freier Bytes, entweder des größten Blocks (`largest`), oder aller freien Speicherblöcke zusammen.

TypeOfMem Anhand einer Speicheradresse ermittelt diese Funktion, welcher Art (etwa chip, public) dieser Bereich ist.

Nach Möglichkeit sollten Sie, wenn Sie Speicher allozieren wollen, nicht die Funktionen von Exec, sondern besser die Routinen aus `Resources` verwenden, da dieses Modul darauf achtet, daß am Programmende auch wirklich alle Speicherstücke wieder freigegeben werden.
Achtung: Versuchen Sie nie ein mit `Resources` alloziertes Speicherstück mit `FreeMem()` wieder freizugeben, ein Absturz ist sicher.

5.3.8 Die Execbase-Struktur

Innerhalb von Exec gibt es die sogenannte ExecBase. Dies ist eine Systemstruktur, in der für die Systemprogrammierung wichtige Felder enthalten sind.

```
ExecBaseType = RECORD OF Library
                 softVer       : CARDINAL;
                 lowMemChkSum  : INTEGER;
                 chkBase       : LONGCARD;
                 coldCapture,
                 coolCapture,
                 warmCapture   : PROC;
                 sysStkUpper,
                 sysStkLower   : ANYPTR;
                 maxLocMem     : LONGCARD;
                 debugEntry    : PROC;
                 debugData,
                 alertData     : ANYPTR;
                 maxExtMem     : ANYPTR;
                 chkSum        : CARDINAL;
                 intVects      : ARRAY IntFlags OF IntVector;
                 thisTask      : TaskPtr;
                 idleCount,
                 dispCount     : LONGCARD;
                 quantum,
                 elapsed       : CARDINAL;
                 sysFlags      : CARDINAL;
                 idNestCnt,
                 tdNestCnt     : SHORTCARD;
                 attnFlags     : AttnFlagSet;
                 attnResched   : CARDINAL;
                 resModules    : ANYPTR;
                 taskTrapCode,
                 taskExceptCode,
                 taskExitCode   : PROC;
                 taskSigAlloc   : LONGSET;
                 taskTrapAlloc  : BITSET;
                 memList,
                 resourceList,
                 deviceList,
                 intrList,
                 libList,
                 portList,
```



```
                 taskReady,
                 taskWait      : List;
                 softInts      : ARRAY [0..4] OF SoftIntList;
                 lastAlert     : ARRAY [0..3] OF LONGINT;
                 vBlankFrequency      : SHORTCARD;
                 powerSupplyFrequency : SHORTCARD;
                 semaphoreList        : List;
                 kickMemPtr,
                 kickTagPtr           : ANYPTR;
                 kickCheckSum         : LONGCARD;
                 execBaseReserved,
                 execbaseNewReserved  : ARRAY 9 OF SHORTCARD;
                 exPad0                : CARDINAL;
                 exReserved0           : LONGCARD;
                 RamLibPrivate         : ANYPTR;
                 eClockFrequency,
                 cacheControl,
                 taskID,
                 puddleSize,
                 poolThreshold         : LONGCARD;
                 publicPool            : MinList;
                 mmuLock               : ANYPTR;
                 exReserved1           : ARRAY[ 0..11 ] OF SHORTCARD;

                 execBaseReserved,
                 execbaseNewReserved   : ARRAY [0..9] OF SHORTCARD;
               END;
```

An dieser Stelle werden nur die Einträge behandelt, auf die man recht unproblematisch zugreifen kann und die im Alltag von Interesse sind:

thisTask Zeiger auf den momentan aktiven Task. In der Vergangenheit wurde dieses Feld oft dazu benutzt die Adresse seines eigenen Tasks herauszubekommen. Dies ist jedoch aus Kompatibilitätsgründen untersagt, also Finger weg! Will man seine Taskadresse ermitteln, geschieht dies systemkonform durch `FindTask(NIL)`.

attnFlags Attention-Flags. An diesen Flags kann man erkennen, welche

CPU und FPU im System sind. Die Namen der Flags sind so eindeutig, das eine Erklärung überflüssig ist.

resourceList Liste aller im System befindlichen Resourcen.

deviceList Liste aller gemounteten Devices.

libList Liste aller im Speicher befindlichen Libraries.

portList Liste der öffentlichen MessagePorts im System.

taskReady Liste der zum Ablauf bereiten Taks.

taskWait Liste der wartetenden Tasks.

Da Sie nun die Anfänge dieser Systemlisten kennen, schreiben sie doch unser Beispielprogramm `WriteLibNames` um, so daß es als TaskMonitor oder zur Ausgabe aller Devices dient. Achtung: Wenn Sie auf diese Listen zugreifen, tun Sie dies bitte nur nach Aufruf von `Forbid()` oder noch besser einem `Disable()`.

Alle anderen Felder sind eigentlich nur für sehr erfahrene Systemprogrammierer von Bedeutung und man sollte besser die Finger von ihnen lassen.

Weiterhin kann ich nur abraten, in den Feldern der `ExecBase` etwas zu verändern, da dies meist mit einer Lektion im Meditieren endet.

5.3.9 T_Exec

Wie Sie sicher schon der Beschreibung von `Resources` entnommen haben, enthält das Modul `T_Exec` einige Funktionen aus Exec, mit dem Unterschied, daß hier dafür gesorgt wird, daß alle Resourcen am Programmende bzw. am Ende eines Kontextes wieder freigegeben werden. Dies ist in vielen Fällen sicherer und komfortabler. Einige Funktionen wie z. B. `CreateTask()` existieren in `Exec` gar nicht. Folgende Funktionen wurden getrackt:

OpenLibrary() Öffnet eine Library, diese wird am Programmende wieder geschlossen.

CloseLibrary() Schließt eine Library, mit Berücksichtigung der Resourceverwaltung.

OldOpenLibrary() Wie `OpenLibrary()`, jedoch ohne Versionsnummer.

CreateTask() Startet eine Clusterprozedur als Task, sorgt dafür, daß der Task am Programmende wieder beendet wird.

DeleteTask() Entfernt einen mit CreateTask erzeugten Task.

CreatePort() Erzeugt einen MessagePort inklusive Signalen. Beide werden am Programmende wiederfreigegeben.

DeletePort() Gibt einen mit `CreatePort` erzeugten Port inklusive Signal frei.

GetMsg() Sorgt dafür, daß erhaltene Nachrichten am Programm- oder Kontextende wieder zurückgeschickt werden.

ReplyMsg() Schickt eine erhaltene Nachricht zurück, entfernt sie dabei aus dem Resourcepool.

OpenDevice() Öffnet ein Device und trägt es in die Resourceverwaltung ein.

CloseDevice() Gegenstück zu `OpenDevice()`.

Achtung: Haben Sie eine Resource mit einer Funktion von `T_Exec` alloziert, dann müssen Sie sie auch mit der entsprechenden Funktion aus diesem Modul freigeben[7], auf keinen Fall mit der entsprechenden Funktion aus `Exec`, da diese Funktionen die Resourcen nicht aus der Resourceverwaltung austragen. Am Programmende würden sie dann ein zweites mal freigegeben, was zu einem Absturz führen würde. Genauso sollten Sie eine Resource, die mit `Exec` erzeugt worden ist, mit `T_Exec` freigeben, da dann das Resourcesystem etwas freizugeben versucht, welches gar nicht eingetragen wurde.

Doch damit genug von Exec, nun wenden wir uns der zweiten nicht weniger wichtigen Library zu: Dos.

[7] Bzw. im Falle von Messages zurückschicken.

5.4 Dos, der Ein- Ausgabespezialist

Dos ist eine Abkürzung für Disk Operating System, was soviel bedeutet wie Datenträger-Betriebssystem. Damit wird auch die Aufgabe der dos.library beschrieben. Sie ist die Schnittstelle zu den Datenträgern wie Diskette und Festplatte. Doch darüber hinaus enthält diese Library auch Funktionen zum Starten von Prozessen. Es ist leicht einzusehen, daß Dos ein zentraler Teil des Amiga-Betriebssystems ist. Ohne Dos könnten zwar Exec-Tasks laufen, aber diese könnten mit der Umwelt nicht in Kontakt treten. Ohne Dos läuft also nichts.

Als CLUSTER-Programmierer fragt man sich natürlich, wofür man überhaupt die Dos-Funktionen zur Dateiverwaltung braucht. Schließlich gibt es ja das Standardmodule FileSystem und DosSupport. Sie haben nicht ganz unrecht, dennoch enthält die dos.library einige interessante Funktionen, die in diesen Modulen nicht enthalten sind, die jedoch sehr nützlich sind.

5.4.1 Datei-Funktionen

```
PROCEDURE Open( REF name       IN D1 : STRING;
                    accessMode IN D2 : OpenMode ): FileHandlePtr;
```

Mit dieser Funktion kann man eine Datei öffen. Dabei stehen vier verschiedene Modi für `accessmode` zur Verfügung:

readWrite Eine Datei wird zum Lesen und Schreiben geöffnet. Falls sie noch nicht existiert, wird sie neu erzeugt.

oldFile Eine bereits existierende Datei wird zum Lesen und Schreiben geöffnet.

newFile Es wird eine neue Datei zum Schreiben geöffnet. Falls schon eine Datei mit diesem Namen existiert, wird diese gelöscht.

Falls Open() fehlschlägt, bekommt man kein FileHandle, sondern NIL zurück.


```
PROCEDURE Close( file IN D1 : FileHandlePtr);
```

Mit dieser Funktion wir die zuvor mit `Open()` geöffnete Datei wieder geschlossen.

```
PROCEDURE Read( file      IN D1 : FileHandlePtr;
                buffer    IN D2 : ANYPTR;
                length    IN D3 : LONGINT ): LONGINT;
```

In den `buffer` werden `length` Bytes eingelesen. Der Rückgabewert ist die Anzahl der tatsächlich gelesenen Bytes. Falls ein Fehler auftritt, wird `-1` zurück gegeben. Da diese Funktion ungepuffert arbeitet, ist es empfehlenswert, sie nur für größere Datenmengen anzuwenden. Für kleinere „Reads“ stehen ab OS 2.0 gepufferte Funktionen wie `FRead()` und `FGets()` zur Verfügung. Will man Funktionsaufrufe von gepufferten und ungepufferten Funktionen mischen, so muß man zwischenzeitlich `Flush()` aufrufen.

```
PROCEDURE Write( file      IN D1 : FileHandlePtr;
                 buffer    IN D2 : ANYPTR;
                 length    IN D3 : LONGINT ): LONGINT;
```

Diese Funktion ist das logische Gegenstück zu `Read()` und muß deshalb nicht genauer erklärt werden.

```
PROCEDURE Seek( file      IN D1 : FileHandlePtr;
                position  IN D2 : LONGINT;
                mode      IN D3 : SeekMode ): LONGINT;
```

Mit `Seek()` kann man den internen Datenzeiger[8] der FileHandles verändern, um Daten zu überspringen oder nochmals zu lesen. Es gibt drei verschidene Modi für `position`:

[8] Dieser gibt die Position im File an, an der das nächste Zeichen gelesen/geschrieben wird.

beginning Gibt an, an welche Stelle – gezählt in Bytes vom Dateianfang – der Zeiger gestellt werden soll.

current Es wird relativ zur momentanen Position verschoben, man kann also positive und negative Werte für `position` angeben.

end Spricht für sich: Positionierung relativ zum Dateiende.

Der Rückgabewert entspricht der neuen absoluten Position in der Datei oder `-1`, falls ein Fehler aufgetreten ist.

```
PROCEDURE SetFileSize( fh   IN D1 : FileHandlePtr;
                       pos  IN D2 : LONGINT;
                       mode IN D3 : SeekMode): LONGINT;
```

Wie schon oben erwähnt, kann man mit dieser Funktion die Größe einer Datei verändern. Sie kann sowohl vergrößert, als auch verkleinert werden. Dabei gibt `pos` die Position des neuen Dateiendes an, `mode` bestimmt dabei wie bei `Seek()`, wie `pos` zu interpretieren ist. Wichtig: Vergrößert man eine Datei, darf man keinerlei Annahmen über die Daten in dem vergrößerten Teil machen. Verkleinert man eine Datei und die aktuelle Fileposition befindet sich hinter dem neuen Dateiende, befindet sich diese danach auf diesem.

Als Rückgabewert erhält man die Position des neuen Dateiendes oder -1, falls ein Fehler auftrat. Achtung: Sie sollten diesen Rückgabewert auf gar keinen Fall ignorieren, da es möglich ist, daß einige Devices diese neue Funktion noch nicht unterstützen.

Nach Möglichkeit sollte man diese Dateifunktionen nicht verwenden, sondern die gepufferten Funktionen aus FileSystem benutzen.

Als Beispiel für die Benutzung der Dos-Funktionen kann das Modul FileSystem und das Beispielprogramm DosFileTest dienen:

```
MODULE DosFileDemo;

IMPORT Dos AS d;

CONST
```



```
  BufSize = 1024;

VAR
  File      : d.FileHandlePtr;
  Buffer    : ARRAY [BufSize] OF CHAR;
  Size, Err : LONGINT;

BEGIN
  File := d.Open ("S:Startup-Sequence", d.oldFile);
  IF File # NIL THEN
    REPEAT
      Size := d.Read (File, Buffer'PTR, BufSize);
      IF Size > 0 THEN
        IF d.Write (d.Output (), Buffer'PTR, Size) # Size THEN
          Size := -1;
        END;
      END;
    UNTIL Size < BufSize;
    IF Size = -1 THEN
      FORGET d.PrintFault (d.IoErr(), "IO-Error: ");
      HALT (20);
    END;
  END;
CLOSE
  IF File # NIL THEN
    FORGET d.Close (File);
    File := NIL;
  END;
END DosFileDemo.
```

Es gibt seit OS 2.0 noch einige weitere Funktionen, so wie die oben erwähnten Funktionen zur gepufferten Ein-/Ausgabe. Der Sinn dieser Prozeduren erschließt sich aber aus ihrem Namen und sollte deshalb nach einem Blick in die Definitionsdatei `Dos.def` klar sein.

Neben `Open()` gibt es noch weitere Funktionen, um einen FileHandle zu erhalten. Nämlich die Funktionen `Input()` und `Output()`, die einen FileHandle auf den Standard-Ein/Ausgabekanal zurückliefern. Dabei handelt es sich normalerweise um das Console-Device des Shellfensters, von dem das Programm gestartet wurde, oder das File/Device, auf den die Standard-Ein/Ausgabe mittels </> umgelenkt wurde. Wurde das Programm von der Workbench gestartet, erhält man NIL. Hier die zwei Funktionen:

```
PROCEDURE Input(): FileHandlePtr;
PROCEDURE Output(): FileHandlePtr;
```

Da wir gerade bei Consolefenstern sind, es ist ganz einfach über Dos ein solches zu öffnen:

```
fh:=Open("CON:20/10/100/50/MyWindow",readWrite);
```

Verwendet man statt `CON:` `RAW:`, erhält man neben normalen Zeichen auch Cursor- und andere Steuertasten. Man hat jedoch keine Editierunterstützung wie bei `CON:`

Gerade bei einem Konsolefenster möchte man eventuell nur eine bestimmte Zeit auf die Eingabe eines Zeichens warten. Hierfür existiert die Funktion `WaitForChar()`

```
PROCEDURE WaitForChar( File    IN D1 : FileHandlePtr;
                       Timeout IN D2 : LONGINT ): LONGBOOL;
```

Diese Funktion wartet `TimeOut` Microsekunden auf die Eingabe eines Zeichens auf dem FileHandle `file`. Sie gibt TRUE zurück, wenn innerhalb des Zeitraums ein Zeichen eingegeben wurde. Dieses kann dann mittels `Read()` gelesen werden.

Wichtig, diese Funktion ist nur auf interaktive FileHandles anwendbar, d. h. FileHandles, die einem virtuellen Terminal, von dem Benutzereingaben gelesen werden können, zugeordnet sind. Um zu testen, ob ein Filehandle interaktiv ist existiert folgende Funktion, die TRUE zurückliefert, falls es sich um ein interaktives File handelt:

```
PROCEDURE IsInteractive( File IN D1 : FileHandlePtr ): LONGBOOL;
```


5.4.2 Argumentparsing / ReadArgs()

Um einen Computer möglichst einfach bedienen zu können, sollte sich jedes Programm an einen bestimmten Standard halten. Für den Amiga ist dieser Standard im „User Interface Style Guide“ festgelegt. Falls ein Programm Parameter von der Kommandozeile erwartet, so sollten diese nach einem bestimmten Schema behandelt werden. Dieses Schema ist ausführlich im Benutzerhandbuch des Amigas erklärt. Mit der Prozedur `ReadArgs()` wird es sehr einfach, solche Kommandozeile zu parsen.

```
PROCEDURE ReadArgs( REF template IN D1 : STRING;
                        array    IN D2 : ANYPTR;
                        args     IN D3 : RDArgsPtr ): RDArgsPtr;
```

Als `args` übergibt man im Normalfall NIL. Für hier nicht behandelte Sonderfälle kann es sinnvoll sein, hier eine mithilfe von `AllocDosObject()` allozierte RDArgs-Struktur, die mit eigenen Werten gefüllt wurde, einzutragen.

In `template` wird eine Argumentenschablone angegeben, anhand der der Array, auf den ärray”zeigt, ausgefüllt wird. Die Schablone besteht aus durch Kommas getrennten Optionen, die möglichst im Klartext angegeben werden sollten. Abkürzungen sind mit Hilfe eines Gleichheitszeichens möglich (z.B.: „`Q=QUICK`“).

Jede Option entspricht einem Langwort im Array. Die Langwörter sind im Normalfall SysStringPtrs. Bei bestimmten Steuerkennungen, sind jedoch Abweichungen möglich. Wird ein Argument nicht angegeben, so wird das korrespondierende Element in dem Array nichts geändert, deshalb sollte man das Array vor dem Aufruf von ReadArgs() mit Vorgabewerten füllen.

Folgende Steuerkennungen gibt es:

- /S Schalter. Bei dieser Option kommt es nur darauf an, ob sie gesetzt ist, oder nicht. Falls die Option angegeben wird hat das Langwort einen Wert ungleich `0`.
- /K Schlüsselwort. Bei dieser Option muß immer der Name der Option mit angegeben werden. Wenn die Option z. B. „`TEST`“ heißt, so wird diese nur ausgefüllt, falls die Eingabe entweder durch

„`TEST=<beliebiger string>`“ oder

„`TEST <beliebiger string>`“ erfolgt.

/N Zahl. Die Option ist eine Zahl. Zurückgegeben wird ein Zeiger auf einen LONGINT, der die Zahl enthält.

/T Schalter. Entspricht `/S`, aber negiert den Vorgabewert.

/A Bedeutet, daß das Argument angegeben werden muß.

/F Der Rest der Kommandozeile wird als Parameter ausgewertet, selbst wenn sich darin Schlüsselwörter anderer Optionen befinden.

/M Durch diese Steuerkennung kann eine Option mehrere Strings bekommen. Das entsprechende Langwort in `array` zeigt dann auf ein Array von SysStringPtrs. Das Ende dieses Arrays erkennt man an einem `NIL`.

Beispiel: Bei einem Template „`DIR/M,ALL/S`“ und einer Kommandozeile „`foo bar all qwe`“ wird der Schalter ‘ÄLL“ gesetzt und für „`DIR`“ ein Array zurückgegeben, der „`foo`“, „`bar`“, „`qwe`“ und `NIL` enthält.

`ReadArgs()` gibt eine RDArgs-Struktur zurück, die man erst wieder mit `FreeArgs()` freigeben darf, wenn man nicht mehr auf `array` zugreift.

Bei einem Fehlschlag von `ReadArgs()` wird `NIL` zurückgegeben. In einem solchen Fall sollte man das richtige Format ausgeben (in diesem Fall das `template` ausgeben und das Programm beenden.

Das Beispielprogramm zur Dos Funktion `ExAll()` zeigt auch, wie man `ReadArgs()` verwendet. **Achtung:** Ein Programm, daß `ReadArgs()` verwendet, kann nicht aus dem Editor heraus gestartet werden.

5.4.3 Dateiinformationen

Falls man zu einem Dateinamen nähere Informationen braucht, bekommt man diese durch die Funtion `Examine()`. Diese füllt eine FileInfoBlock-Struktur aus. Hier ist sie:

```
FileInfoBlock    =  RECORD
                      diskKey      : LONGINT;
                      dirEntryType : EntryType;
                      fileName     : ARRAY [0..107] OF CHAR;
                      protection   : ProtectionFlagSet;
                      entryType    : LONGINT;
                      size         : LONGINT;
                      numBlocks    : LONGINT;
                      date         : Date;
                      comment      : ARRAY [0..79] OF CHAR;
                      reserved     : ARRAY [0..35] OF CHAR;
                    END;
```

Es ist wichtig, daß diese Struktur ab OS 2.0 mit `AllocDosObject()`, ansonsten mit `New` alloziert wird, da die Adresse auf einer durch 4 teilbaren Adresse liegen muß; daher keine Variable diesen Typs verwenden. Ansonsten sind unvorhersehbare Abstürze möglich.

Die wichtigsten Felder der Struktur im einzelnen (alle anderen Felder sind privat und sollten deshalb ignoriert werden):

dirEntryType Interessant sind hier nur die Werte: `file`/`linkFile`, es handelt sich um eine Datei; `root`, es handelt sich um ein Root-Verzeichnis und `userDir`/`linkDir` es handelt sich um ein Verzeichnis.

fileName Der Filename, sonst nichts.

protection Die bei den Shell-Befehlen „List“ und „Protect“ bekannten Schutzbits: `delete`, `execute`, `writeProt`, `readProt`, `archive`, `pure`, und `script`. Alle anderen Bits sind reserviert und sollten deshalb nicht beachtet und nicht verändert werden. Die Bits kann man übrigens mit der `SetProtection()` Funktion verändern.

size Die Dateigröße in Bytes, falls es sich um eine Datei handelt. Ab OS 2.0 kann man mit `SetFileSize()` die Dateigröße ändern (also auch verkleinern). Ansonsten kann man eine Datei nur durch `Write()` und verwandte Funktionen vergrößern.

numBlocks Die Anzahl der Blocks, die die Datei belegt.

date Das Datum der Datei, welches ab OS 2.0 mit `SetFileDate()` geändert werden kann.

comment Der Kommentar der Datei, dieses Feld wird jedoch bei den wenigsten Dateien benutzt. Mit der `SetComment()` Funktion kann man den Kommentar jederzeit ändern.

Doch wie kommt man an all diese Informationen? Zuerst braucht man einen sog. FileLock[9], den man dann an `Examine()` übergeben kann. Diesen Lock besorgt die gleichnamige Funktion:

```
PROCEDURE Lock( REF name       IN D1 : STRING;
                    accessMode IN D2 : LockMode ): FileLockPtr;
```

Es gibt zwei verschiedene Arten von FileLocks, die man über den `accessMode` anfordern kann. Einen `exlusiveLock` kann nur ein Programm alleine halten. Es können jedoch beliebig viele Programme einen `sharedLock` haben.

Falls die gewünschte Datei existiert und kein anderes Programm einen `exclusiveLock` hält, bekommt man einen BCPL-Zeiger auf eine FileLock-Struktur zurück, ansosten `NIL`.

Eine andere Möglichkeit an einen Lock zu kommen ist, einen bestehenden `sharedLock` zu kopieren:

```
PROCEDURE DupLock( lock IN D1 : FileLockPtr ): FileLockPtr;
```

Unter OS 2.0 kann man sogar einen Lock von einem mit `Open()` geöffneten FileHandle bekommen:

[9] Einen Zugriff auf die Datei


```
PROCEDURE DupLockFromFH( fh IN D1: FileHandlePtr ): FileLockPtr;
```

Den so erhaltenen Lock muß man nach dem Gebrauch auf jeden Fall so schnell wie möglich wieder freigeben, damit ein reibungsloser Ablauf im Multitaskingsystem möglich ist. Dazu dient:

```
PROCEDURE Unlock( Lock IN D1 : FileLockPtr );
```

Jetzt kann man die oben besprochene Funktion `Examine()` aufrufen, die diese Syntax hat:

```
PROCEDURE Examine( Lock      IN D1 : FileLockPtr;
                   InfoBlock IN D2 : FileInfoBlockPtr ): LONGBOOL;
```

Eine weitere häufig benötigte Funktion ist das Einlesen von Verzeichnissen. Seit OS 2.0 wurde die umständliche und fehlerträchtige Methode über `Examine()` und `ExNext()` durch eine viel komfortablere abgelöst, die sich der Funktion `ExAll()` bedient.

Um `ExAll()` verwenden zu können, muß man sich zuerst mit `AllocDosObject()` eine ExAllControl-Struktur besorgen.

```
ExAllControlPtr =  POINTER TO ExAllControl;
ExAllControl    =  RECORD
                     entries    :  LONGCARD;
                     lastkey    :  LONGCARD;
                     matchString:  SysStringPtr;
                     matchFunc  :  HookPtr;
                   END;
```

Im Feld `entries` steht nach einem erfolgreichen Aufruf von `ExAll()` die Anzahl der zurückgegebenen Files. Es ist möglich, daß `ExAll()` mehrmals aufgerufen werden muß, dies sogar, wenn `entries` gleich `0` ist. Dazu später.

`lastkey` muß man auf jeden Fall vor dem ersten Aufruf auf `0` setzten. Ansonsten sollte man das Feld in Ruhe lassen, da es von Dos intern benötigt wird.

Trägt man bei `matchString` nicht `NIL`, sondern ein AmigaDOS-Pattern-String ein, so werden nur Files zurückgegeben, die auf dieses Pattern passen.
Achtung: Dieser String muß mit `ParsePatternNoCase()` zuvor behandelt worden sein. Siehe Patternmatching 5.4.6

Eine weitere Möglichkeit, bestimmte Files auszuwählen, hat man, wenn man bei `matchFunc` mit Hilfe eines Hooks eine Prozedur installiert, die für jedes File entscheidet, ob es angenommen werden soll. Diese Vorgehensweise wird hier aber nicht beschrieben. Näheres findet man in den AutoDocs und im AmigaDOS-Manual.

Um jetzt `ExAll()` verwenden zu können, braucht man nur noch einen Lock auf das Verzeichnis, das man untersuchen will und einen Puffer, in dem die Informationen zu den Files gespeichert werden. Den Puffer holt man sich am besten mit `AllocMem()`, wobei man beachten sollte, daß bei einem kleinen Puffer ExAll sehr oft aufgerufen werden muß. Ein vernünftiger Wert ist hier 2 KB.

Jetzt endlich kann der Aufruf von `ExAll()` stattfinden.

```
PROCEDURE ExAll( lock    IN D1 : FileLockPtr;
                 buffer  IN D2 : ANYPTR;
                 size    IN D3 : LONGINT;
                 type    IN D4 : ExAllType;
                 control IN D5 : ExAllControlPtr ): LONGBOOL;
```

Im Feld `type` gibt man an, welche Informationen zu einem File man gerne hätte. Es gibt diese Möglichkeiten:

```
ExAllType     =  (name=1, type, size, protection,
                  date, comment);
```

Dabei ist zu beachten, daß wenn man z.B. `size` verwendet auch die Felder `name` und `type` der unten erklärten ExAllData-Struktur gültige Werte besitzen. Alle felder ab `prot` enthalten jedoch keine gültigen Informationen.

Wenn nach dem Aufruf von `ExAll()` in der Control-Struktur mehrere Einträge angezeigt werden, so wurde der Puffer mit mehreren ExAllData-Strukturen gefüllt.


```
ExAllDataPtr    =  POINTER TO ExAllData;
ExAllData       =  RECORD
                     next       :  ExAllDataPtr;
                     name       :  SysStringPtr;
                     type       :  EntryType;
                     size       :  LONGCARD;
                     prot       :  ProtectionFlagSet;
                     date       :  Date;
                     comment    :  SysStringPtr;
                   END;
```

Über das `next` kann man die Liste nacheinander durchgehen. Alle anderen Felder sind selbsterklärend.

Falls der Aufruf von `ExAll()` erfolgreich war (mit `IoErr()` zu überprüfen), so muß man diesen solange wiederholen, bis `ExAll() FALSE` zurückgibt und damit signalisiert, das alle Einträge abgearbeitet wurden.

Die Feinheiten sollten aus dem Beispielprogramm ersichtlich sein:

```
MODULE ExAllDemo;

FROM InOut     IMPORT WriteString,WriteLn;
FROM Strings   IMPORT Str;
FROM System    IMPORT SysStringPtr;
FROM Resources IMPORT Allocate,Dispose,MemReqs;
               IMPORT Dos     AS d;

CONST
  BufSize  = 2048;
  Template = "DIR,ALL/S";

VAR
  RD        : d.RDArgsPtr;
  ArgArray  : ARRAY [2] OF ANYPTR;
  Dir       : STRING(256);
  Level     : INTEGER;

PROCEDURE ShowDir (Dir: STRING; All: BOOLEAN);
VAR
```

```
  ObjLock, OldDir : d.FileLockPtr;
  res2            : d.IoErrors;
  more            : BOOLEAN;
  Buffer, ead     : d.ExAllDataPtr;
  control         : d.ExAllControlPtr;
  i               : INTEGER;
BEGIN
  TRACK
    | control MUST be allocated by AllocDosObject!
    control := d.AllocDosObject (d.exAllControl, DONE);
    Allocate(Buffer, BufSize, mem:={public, clear});

    IF (control # NIL) THEN
      ObjLock := d.Lock (Dir, d.sharedLock);
      IF ObjLock # NIL THEN
        OldDir := d.CurrentDir (ObjLock);
        control.lastkey := 0;  (* paranoia *)
        REPEAT
          more := d.ExAll (ObjLock, Buffer, BufSize, d.type, control);
          res2 := d.IoErr();
          IF (NOT more) AND (res2 # d.noMoreEntries) THEN
            FORGET d.PrintFault (res2,"Abnormal exit, error = ");
            HALT (20);
          END;
          IF control.entries # 0 THEN
            ead := Buffer;
            REPEAT
              i := Level;
              WHILE i > 0 DO
                WriteString("   ");
                DEC (i);
              END;
              WriteString(Str(ead.name));
              IF ead.type OF d.root,d.userDir,d.linkDir THEN
                WriteString(" (dir)");
                WriteLn;
                IF All THEN
                  INC (Level);
```



```
                  ShowDir (Str(ead.name), All);
                  DEC (Level);
                END;
              ELSE
                WriteLn;
              END;
              ead := ead.next;
            UNTIL ead = NIL;
          END;
        UNTIL NOT more;

        OldDir := d.CurrentDir (OldDir);
        d.Unlock (ObjLock);
      ELSE
        WriteString("Couldn't find ");WriteString(Dir);
        WriteLn;
      END;
    END;
  CLOSE | Sorgt dafür, daß auch im HALT-Fall
        | alles wieder freigegeben wird.
        | Buffer wird automatisch freigegeben
    IF control # NIL THEN
      d.FreeDosObject (d.exAllControl, control);
      control := NIL
    END;
  END;
END ShowDir;

VAR
  c : CHAR;
BEGIN
  ArgArray[0] := NIL; ArgArray[1] := 0;
  RD := d.ReadArgs (Template, ArgArray'PTR, NIL);
  IF RD = NIL THEN
    WriteString("Usage: ");WriteString(Template);
    WriteLn;
    HALT (20);
  END; (* IF *)
```

```
  IF ArgArray[0] # NIL THEN
    Dir:=Str(SysStringPtr(ArgArray[0]));
  ELSE
    Dir := "";
  END;
  Level := 0;
  ShowDir (Dir, ArgArray[1] # 0);
CLOSE
  IF RD # NIL THEN d.FreeArgs (RD); RD := NIL END;
END ExAllDemo.
```

5.4.3.1 Weitere Lock-Funktionen

Hat man einen Lock auf ein Directory, kann man dieses Directory zum Current-Directory für den eigenen Prozess machen. Dies entspricht dem Kommando CD in der Shell. Alle Dateipfade, sofern sie kein Device oder Volume enthalten, beziehen sich immer auf das Current-Directory. Hat man viele Zugriffe auf ein Verzeichnis zu machen, empfiehlt es sich, dieses zum aktuellen Verzeichnis zu machen, da dadurch die Zugriffe erheblich beschleunigt werden. Hierzu dient die Funktion `CurrentDir()`:

```
PROCEDURE CurrentDir( Lock IN D1 : FileLockPtr ): FileLockPtr;
```

Als Ergebnis erhält man einen Lock auf das vorherige aktuelle Verzeichnis, den man sich merken und am Programmende wieder setzen sollte. Übergibt man für Lock NIL, wird das Volume, von dem gebootet wurde zum aktuellen Verzeichnis (entspricht `SYS:`. Wichtig: Einen Lock, den man zum aktuellen Verzeichnis gemacht hat, darf man nicht mittels Unlock freigeben.

Mittels der Funktion `ParentDir()` erhält man den Lock auf das Übergeordnete Verzeichnis eines Verzeichnisses oder NIL, wenn es sich um ein Root-Verzeichnis gehandelt hat:

```
PROCEDURE ParentDir( Lock IN D1 : FileLockPtr ): FileLockPtr;
```

Für `Lock` übergibt man den Lock auf das Verzeichnis, dessen Parent-Directory man ermitteln will.

Bisher war es immer schwierig, wenn ein Programm auf zusetzliche Dateien zugreifen wollte, aber nicht wußte, wo es sie suchen soll. Bisher wurde meist ein `assign` auf das Programmverzeichnis verwendet, das vorhanden sein mußte. Seit OS 2.0 gibt es die Möglichkeit das Verzeichnis, in dem ein Programm abgelegt ist, zu ermitteln. Hierfür dient die Funktion `GetProgramDir()`:

```
PROCEDURE GetProgramDir(): FileLockPtr;
```

Möchte man zu einem Lock dessen Pfad erhalten, kann man die Funktion `NameFromLock()` verwenden:

```
PROCEDURE NameFromLock(     lock   : FileLockPtr;
                        VAR buffer : STRING;
                            len    : LONGINT:=0): LONGBOOL;
```

Für `buffer` übergibt man einen String, in den der Pfad eingetragen werden soll. `len` gibt die maximale Länge von `buffer` an, ist aber Überflüssig, da nicht direkt die Dos-Funktion aufgerufen wird und die Clusterfunktion die Länge des Strings einträgt, wenn für `len` Null eingetragen wird.
Für FileHandles existiert eine entsprechende Funktion:

```
PROCEDURE NameFromFH(     fh     : FileHandlePtr;
                      VAR buffer : STRING;
                          len    : LONGINT:=0 ): LONGBOOL;
```

Beide Funktionen liefern FALSE zurück, falls ein Fehler auftrat, z. B. wenn der String zu kurz war. `IoErr()` auslesen, um die genaue Fehlerursache festzustellen.

Will man feststellen, ob sich zwei Locks auf dem selben Device befinden, z. B. um sicher zu sein, daß man ein File mit `Rename()` von einem Directory in ein anderes bewegen kann, kann man dies mit der Funktion `SameDevice()` tun, die TRUE zurückliefert, wenn sich beide Locks auf dem selben Device befinden:

```
PROCEDURE SameDevice( lock1 IN D1 : FileLockPtr;
                      lock2 IN D2 : FileLockPtr ): LONGBOOL;
```

5.4.4 Diskinformationen

Um Informationen über einen Datenträger zu erhalten, kann man die Funktion `Info()` verwenden:

```
PROCEDURE Info( Lock           IN D1 : FileLockPtr;
                ParameterBlock IN D2 : InfoDataPtr ): LONGBOOL;
```

Für `Lock` übergibt man einen Lock auf den Datenträger oder irgendein File auf ihn, für `ParameterBlock` einen Zeiger auf einen Record vom Typ `InfoData`. Dabei ist wichtig, daß dieser Record auf einer durch 4 teilbaren Adresse liegt, am einfachsten alloziert man ihn mit Allocate, bitte nicht als Variable definieren. Der Record hat folgenden Aufbau:

```
  InfoData        =  RECORD
                       numSoftErrors   : LONGINT;
                       unitNumber      : LONGINT;
                       diskState       : DiskStatus;
                       numBlocks,
                       numBlocksUsed   : LONGINT;
                       bytesPerBlock   : LONGINT;
                       diskType        : DiskType;
                       volumeNode      : DeviceListPtr;
                       inUse           : LONGBOOL;
                     END;
```

Nach dem Aufruf der Funktion ist der Record mit folgenden Daten gefüllt:

numSoftErrors : Die Anzahl der Diskettenfehler

unitNumber : Die Nummer des Laufwerks

diskState : Status der Disk:

> **writeProtected :** Disk ist schreibgeschützt.
>
> **validating :** Disk wird validiert, es kann nicht darauf geschrieben werden.

> **validated :** Disk ist ok und beschreibbar.

numBlocks : Anzahl der Blöcke auf der Disk.

numBlocksUsed : Anzahl der belegten Blöcke.

bytesPerBlock : Größe eines Blocks in Bytes.

diskType : Art der Disk:

> **noDiskPresent :** Es befindet sich keine Diskette im Laufwerk.
>
> **unreadableDisk :** Diskette ist nicht lesbar.
>
> **dosDisk :** Normale Dos-Disk mit dem alten FileSystem.
>
> **ffsDisk :** Dos-Disk mit FastFileSystem.
>
> **notReallyDos :** Zwar Dos-Format, jedoch keine Dos-Disk.
>
> **kickstartDisk :** Es handelt sich um eine Kickstart-Diskette, kommt eigentlich nur beim A1000 vor.
>
> **msdosDisk :** Diskette, die im MS-Dos-Format formatiert ist.

volumeNode : BCPL-Zeiger auf die VolumeNode der Disk.

inUse : Ist FALSE, wenn keiner darauf zugreift.

5.4.5 Dateiverwaltung

Neben den Funktionen zum Anlegen und Beschreiben von Dateien, gibt es auch eine Reihe von Verwaltungsfunktionen, die mit vielen Shellfunktionen identisch sind und daher einfach zu benutzen sein sollten:

```
PROCEDURE DeleteFile( REF Name IN D1 : STRING ): LONGBOOL;
```

Löscht die Datei `Name`. Gibt FALSE zurück, falls ein Fehler auftrat.

```
PROCEDURE Rename( REF OldName IN D1,
                      NewName IN D2 : STRING ): LONGBOOL;
```

Benennt die Datei `OldName` in `NewName` um. Dabei kann `NewName` auch in einem anderen Verzeichnis liegen, sofern sich dieses auf dem selben Volume befindet. Es findet dann ein „Move“ statt. Gibt FALSE zurück, falls ein Fehler auftrat.

```
PROCEDURE SetComment( REF Name    IN D1,
                          Comment IN D2 : STRING ): LONGBOOL;
```

Setzt für die Datei `Name` den Kommentar `Comment`. Gibt FALSE zurück, falls ein Fehler auftrat.

```
PROCEDURE SetFileDate( REF name IN D1 : STRING;
                           date IN D2 : DatePtr ): LONGBOOL;
```

Setzt für die Datei `name` das neues Datum `date`. Gibt FALSE zurück, falls ein Fehler auftrat.

```
PROCEDURE CreateDir( REF Name IN D1 : STRING ): FileLockPtr;
```

Erzeugt ein Verzeichnis mit Pfad `Name` und gibt einen FileLockPtr darauf zurück oder NIL wenn ein Fehler auftrat.

```
PROCEDURE Relabel( REF drive   IN D1 : STRING;
                   REF newname IN D2 : STRING ): LONGBOOL;
```

Benennt das Volume (Diskett/Festplatte) `drive` auf den Namen `newname` um. Gibt FALSE zurück, falls ein Fehler auftrat.

5.4.6 Patternmatching

Bisher war die Bearbeitung von Dos-Pattern sehr aufwendig, da es keine spezielle Funktion hierfür in Dos gab. Seit 2.0 hat sich dies nun geändert. Um zu prüfen, ob ein String einem Dos-Pattern enspricht, gibt es nun die Funktion `MatchPatternNoCase()`:

```
PROCEDURE MatchPatternNoCase(     pat IN D1 : ANYPTR;
                              REF str IN D2 : STRING ): LONGBOOL;
```


Sie liefert TRUE zurück, falls `str` dem Dos-Pattern entspricht. Dabei ist zu beachten, daß Sie für `pat` nicht direkt einen Patternstring übergeben können, sondern es muß ein Zeiger auf einen vorbearbeiteten Patternsstring übergeben werden. Um diesen zu erzeugen, gibt es die Funktion `ParsePatternNoCase()`:

```
PROCEDURE ParsePatternNoCase( REF buf    IN D1 : STRING;
                                  pat    IN D2 : ANYPTR;
                                  buflen IN D3 : LONGINT ): LONGINT;
```

Diese Funktion erhält in `buf` einen String mit dem gewünschten Pattern, dabei sind alle Patternsymbole erlaubt, die man aus der Shell kennt wie z. B. `#?|` etc. Für `pat` übergibt man einen Zeiger auf einen Puffer, in dem das bearbeitete Pattern abgelegt werden soll. Dabei ist zu beachten, daß der Puffer mindestens doppelt so groß sein muß wie der Patternstring lang ist, zuzüglich zwei Bytes, da jedes Patternzeichen zu zwei Tokens verwandelt wird, eines sogar in drei, allerding nur einmal pro Pattern. In `buflen` übergibt man die Länge des Puffers. Als Ergebnisse kann man folgende Werte erhalten:

1 Bedeutet, daß innerhalb des Pattern sich *Wildcards*[10] befinden.

0 Es befinden sich keine *Wildcards* im Pattern.

-1 Es trat ein Fehler auf, möglicherweise war Ihr Puffer zu klein.

Diese Funktion ist auch zu verwenden, wenn man einen Patternstring für die `ExAllControl`-Struktur erhalten will.

Beide Funktionen sind nicht Case-Sensitiv, d. h. sie machen keinen Unterschied in der Groß/Kleinschreibung. Will man case-Sensitives Patternmatching, bietet Dos die Funktionen `MatchPattern()` und `ParsePattern()` an. Diese werden genau wie die oben beschriebenen verwendet. Da Dos keine Unterschiede bei den Filenamen in der Groß/Kleinschreibung macht, wurden nur diese Funktionen ausführlich beschrieben.

[10]Dies bedeutet, daß das Pattern auf mehr als einen String zutreffen kann, z. B.: `#?` für beliebige Strings.

5.4.7 Fehlerbehandlung

Nach jedem Dos-Kommando läßt sich überprüfen, ob sich ein Fehler ereignet hat und wenn ja, um welche Art von Fehler es sich handelt. Hierzu dient die Funktion `IoErr()`:

```
PROCEDURE IoErr():IoErrors;
```

`IoErrors` ist ein in Dos definierter Aufzählungstyp. Die Namen der Elemente geben ausreichend den Grund für den Fehler an, so daß es nicht nötig ist die Elemete im einzelnen zu beschreiben.

Will man einen Dos-Fehler ausgeben, kann man dazu die Funktion `PrintFault()` verwenden, die eine Ausgabe auf die Standardausgabe macht:

```
PROCEDURE PrintFault(     code   IN D1 : IoErrors;
                      REF header IN D2 : STRING ): LONGBOOL;
```

`code` ist dabei der Fehlercode, den `IoErr()` zurückgibt, `header` ein Vorspann, der vor der dem Code zugeordneten Fehlermeldung ausgegeben wird. Falls ein Fehler auftrat, wird FALSE zurückgegeben.

5.4.8 Prozessverwaltung

Manchmal kommt es vor, daß man aus einem Programm heraus ein anderes Programm starten muß. Seit OS 2.0 hat man dazu die komfortable Prozedur System() zur Verfügung, die das problematische Execute() ablöst.

```
PROCEDURE CallSystemTags(REF command IN D1 : STRING;
                             tags    IN D2 : LIST OF SystemTags):LONGINT;
```

Hier werden nur die wichtigsten Tags besprochen, die anderen kann man durch einen Blick in das Definitions-File leicht finden.

`input :  FileHandlePtr;`

`output :  FileHandlePtr;`
: Diese Tags bestimmen das Input- bzw. Output-Filehandle, also die Datei, die für die Ein- bzw. Ausgaben des gestarteten Programms zuständig sind. Läßt man diesen Tag weg, so wird das FileHandle des aufrufenden Programms vererbt.

`asynch :  LONGBOOL; (nicht ANYPTR!)`
: Hier kann bestimmt werden, daß das Programm asynchron ausgefürt wird. Man bekommt in diesem Fall von `CallSystemTags()` -1 als Ergebnis geliefert, anstelle des Returncodes des Programms. Achtung: Die FileHandles werden automatisch geschlossen! Man sollte also mit `sysInput` und `sysOutput` auf jeden Fall neu erzeugte FileHandles übergeben, da einem sonst die eigenen Handles geschlossen werden!

`priority :  LONGINT;`
: Die Priorität, die das Programm erhalten soll. Wie man sieht, kann man bei CallSystemTags() auch Tags der Prozedur CreateNewProc() übergeben.

Will man jedoch kein fremdes Programm, sondern eine Prozedur als eigenen Prozess starten, dient hierzu die Prozedur `CreateNewProc()`:

```
PROCEDURE CreateNewProc(tags IN D1 : LIST OF NewProcTags):ProcessPtr;
```

Wer sich für die Tags im einzelnen interessiert, sei das AmigaDos-Manual wärmstens ans Herz gelegt. Vielmehr soll hier die Lösung eines Problems gezeigt werden. Leider besteht keine Möglichkeit, einem neu erzeugten Prozess irgendwelche Daten zu übergeben. Damit jedoch der Prozess auf die Globalen Variablen des Moduls zugreifen kann, benötigt er den Inhalt des Adreßregisters `A4` des erzeugenden Programmes. Hierzu gibt es zwei Möglichkeiten: Das prozesserzeugende Programm schickt eine Message mittels `PutMsg()` an den Nachrichtenport des neuen Prozesses und der neue Prozess holt als allererstes diese Nachricht ab. Dies geschieht am besten in Assembler.

Die zweite Möglichkeit ist sehr viel einfacher, hat jedoch den Nachteil, daß das Programm die Fähigkeit der Multientranz verliert, da das Programm selbst verändert wird:

```
CONST
  MyA4    = ARRAY OF LONGINT:(0);

|Prozedur, die als Prozess laufen soll
PROCEDURE MyProcess;
BEGIN
  SETREG(MyA4[0],A4);
  ...
END MyProcess;

BEGIN
  $$ConstChk:=FALSE | Schreibschutz für Konstanten ausschalten
  MyA4[0]:=REG(A4)  | Patchen der Konstanten
  $$ConstChk:=OLD

  FORGET CreateNewProc(entry     : MyProcess,
                       name      : "Dummy",
                       stackSize : 20000,
                       DONE);
```

Da konstante Teile des Programmcodes absolut adressiert werden, sind sie sowohl vom erzeugenden Programm als auch vom neuen Prozess erreichbar[11]. In den meisten Fällen sollte diese Art vollkommen ausreichen, da eigene Prozesse meist nur von größeren Programmen erzeugt werden, die üblicherweise nicht mehrfach gleichzeitig laufen.

Eine weitere Funktion, die mit der Prozessverwaltung zu tun hat, ist die Dos-Funktion `Delay()`. Sie dient dazu, eine bestimmt Zeit zu warten, ohne anderen Programmen Rechenzeit zu stehlen, unabhängig, mit welchem Prozessor und Taktfrequenz der Amiga betrieben wird:

```
PROCEDURE Delay( ticks IN D1  : LONGCARD );
```

Dabei sind `ticks` die Anzahl 1/50 Sekunden, die gewartet werden soll. Bitte verwenden Sie immer diese Funktion anstelle einer Warteschleife.

[11] Es ist wichtig, die Konstante als Array zu deklarieren und nicht nur als LONGINT, das sonst nur eine textuelle Ersetzung stattfindet und keine echte Konstante erzeugt wird.

Die dos.library besitzt noch viele andere Funktionen, die Record-Locking (wichtig für Datenbanken), PatternMatching, Handler & Packets, Datum, Prozeßverwaltung u. a. ermöglichen. Alle diese Funktionen und die dazugehörigen Strukturen zu erklären, sprengt den Rahmen dieses Handbuchs. An dieser Stelle sei auf die Autodocs, das AmigaDOS-Manual und das Guru-Buch hingewiesen.

5.4.9 T_Dos

Falls Sie statt die Funktionen aus `Dos`, die aus dem Modul `T_Dos` verwenden, erhalten Sie bei jedem Dos-Fehler eine entsprechende Exception. Außerdem sorgt `T_Dos` dafür, das alle Resourcen, die mit diesem Modul geöffnet wurden, auch wieder geschlossen werden. Es folgt eine Auflistung der in `T_Dos` definierten Fuktionen, mit deren Unterschied zur jeweiligen Originalfunktion:

Open Öffnet Files, hängt sie in die Resourceverwaltung ein, so daß sie wieder automatisch geschlossen werden. Löst bei Fehlern Exceptions aus.

Close Schließt Files, trägt sie aus der Resourceliste aus.

Read Wie das normale Read, löst entsprechende Exception aus unter anderem `EOF`, wenn über das Fileende hinaus gelesen wird.

Write Löst Exceptions bei Fehlern aus.

CreateDir Sorgt dafür, daß der Lock, den man erhält wieder freigegeben wird. Löst Exceptions im Fehlerfall aus.

CurrentDir Trägt das neue aktuelle Verzeichnis aus der Resourceliste aus, den vorherigen in dieselbe ein, so wird dafür gesorgt, daß der Lock auf das aktuelle Verzeichnis nicht freigegeben werden kann. Löst Exceptions im Fehlerfall aus.

ParentDir, DupLock, Lock Sorgt dafür, daß der Lock, den man erhält wieder freigegeben wird. Löst Exceptions im Fehlerfall aus.

Unlock Gibt einen Lock wieder frei, und trägt ihn aus der Resourceverwaltung aus.

Sie können auch alle anderen Bezeichner aus `Dos` von `T_Dos` importieren, diese Verhalten sich jedoch genauso wie wenn Sie sie aus `Dos` importiert hätten.

Bei der Verwendung von`T_Dos` wird übrigens am Programmende wieder das aktuelle Verzeichnis gesetzt, das am Anfang gesetzt war, dies ist dann wichtig, wenn ein Programm direkt von der Shell gestartet wird.

Achtung: Mischen Sie niemals Prozeduren aus `Dos` und `T_Dos` miteinander, ein Kurztrip nach Indien wäre Ihnen sonst sicher. Das heist nicht, daß in einem Programm nicht beide Module verwendet werden dürfen, lediglich sollte man nie eine Resource, die mit einem Modul belegt wurde mit dem anderen wieder freigeben, da sonst die Resourceverwaltung ins schlingern kommt.

5.5 Intuition & GadTools, Herren der Maus und Gadgets

Dieses Kapitel versucht einen Überblick über die Programmierung der grafischen Benutzeroberfläche des Amiga zu geben. Mehr als ein Überblick kann es nicht werden, da dieses Thema im RKM Libraries über 400 Seiten einnimmt. Jedem, der sich intensiver damit befassen will, sei dieses Buch dringend empfohlen.

5.5.1 Screens

In einem Multitaskingsystem wie dem Amiga besteht das Problem, daß mehrere Programme gleichzeitig Ausgaben auf den Bildschirm machen können. Dies läßt sich im begrenzten Umfang mit Windows lösen, doch ist oft nicht erwünscht, daß man bei seinem Programm ständig noch Fenster von anderen Programmen im Blickfeld hat, außerdem wird es spätestens dann unmöglich, wenn ein Programm eine andere Auflösung als ein anderes beansprucht. Im Grunde genommen müßte man mehrere Bildschirme besitzen.

Dies ist jedoch kostenmäßig keinem normalen Benutzer zuzumuten, außerdem müßte dann auch die gesammte Videologik, die für den Bildaufbau zuständig ist, mehrmals vorhanden sein.

Das Zauberwort heißt virtuelle Bildschirme oder Screens. Das heißt, der Computer simuliert mehrere Bildschirme. Diese liegen hintereinander, lassen sich aber verschieben und können sich auch so überlappen, daß man von mehreren Screens jeweils einen Teil sieht.

Auf jedem Screen können Fenster geöffnet und unterschiedliche Auflösungen und Farben verwendet werden. Die meisten Screens lassen sich über Gadgets rechts oben wie Windows nach vorne oder hinten klicken. Durch Drücken von der rechten Amigataste und „n“ kann man die Workbenchscreen nach vorne, durch die linke Amigataste und „m“ nach hinten schalten. Es gibt einen Standard-Screen, den jedes Programm benutzen kann. Dieser Screen, der „Default Public Screen“, ist normalerweise der Workbench-Screen. Dies kann jedoch geändert werden. Darüber hinaus kann man auf jedem anderen Public Screen ein Fenster öffen. Mehr

dazu im Kapitel über Windows.
Seit OS 2.0 wird ein Screen mit der Funktion `OpenScreenTags()`[12] geöffnet und mit `CloseScreen()` wieder geschlossen.

```
PROCEDURE OpenScreenTags(newScreen IN A0 : NewScreenPtr;
                         tagList   IN A1 : LIST OF ScreenTags
                         ) : ScreenPtr;
```

Der erste Parameter ist ein Zeiger auf eine 1.3 kompatible NewScreen-Struktur, den man nicht benötigt und deshalb auf `NIL` setzen sollte. Diese Tags gibt es:

`left, top : LONGINT;`
: Diese beiden Tags beschreiben die linke und obere Ecke des Screens. Normalerweise gibt man diese Tags nicht an, da automatisch die richtigen Vorgabewerte benutzt werden.

`width, height : LONGINT;`
: Die Breite und Höhe des Screens muß auch nicht angegeben werden, da auch hier sinnvolle Vorgabewerte benutzt werden. Diese Werte hängen vom Overscan und der gewählten DisplayID ab.

`depth : LONGINT;`
: Die Anzahl der Bitplanes, die der Screen bekommen soll. Dadurch werden 2^{depth} Farben möglich (Ausnahme: HAM und EHB). Vorgabewert ist hier 1.

`detailPen, blockPen : LONGCARD;`
: Diese beiden Tags werden nicht benötigt, da ihre Aufgabe durch den `pens`-Tag übernommen wurde.

`title : SysStringPtr;`
: Zeiger auf den Titel des Screens.

[12] alternativ kann hier die Funktion `OpenScreenTagList()` verwendet werden, die ein Array von Tags übergeben bekommt

`colors: ColorPtr;`
: Hier kann man die Farben des Screens angeben. Der übergebene Wert ist ein Zeiger auf ein Array von ColorSpec-Strukturen, die von einer solchen Struktur mit dem `colorIndex` -1 beendet wird.

`errorCode : ErrorTypePtr;`
: Diese Tag erlaubt es, näheres über die Gründe zu erfahren, warum der Screen nicht geöffnet werden konnte. An dieser Stelle trägt man einen Zeiger auf eine Variable vom Typ ErrorType ein, in der nach einem fehlgeschlagenem `OpenScreenTags` die genaue Fehlerursache eingetragen wird.

`font : TextAttrPtr;`
: Der Font des Screens. Normalerweise sollte man jedoch die Wünsche des Benutzers respektieren und den Tag `sysFont` verwenden.

`sysFont : FontPrefs;`
: `oldDefaultFont` öffnet einen Screen mit einem nicht-proportionalen Font. `wbScreenFont` erlaubt auch proportionale Fonts.

`type : ScreenFlagSet;`
: Dieser Tag wird nicht benötigt.

`bitMap : BitMapPtr;`
: Durch diesen Tag kann man dem Screen eine selbsterstellte Bitmap mit auf den Weg geben.

`pubName : SysStringPtr;`
: Durch diesen Tag wird ein Screen zum „Public Screen“. Das bedeutet, daß auch andere Programme ihre Fenster auf diesem Screen öffnen dürfen. Man sollte einen Screen möglichst immer Public machen. Wenn der Screen geöffnet wurde, kann man ihn mit `PubScreenStatus(Screen,0)` wirklich öffentlich machen, nachdem man mit eigenen Initialisierungen (z. B. öffnen eines Backdrop-Windows) fertig ist. Der Name des Screens darf nicht zweimal vorkommen. Er sollte dem ARexx-Port-Namen und damit dem abgekürzten Programmnamen in Großbuchstaben entsprechen. Näheres findet man im User Interface Style Guide. Dieser Tag muß vor `pubSig` und `pubTask` angegeben werden.

`pubSig : TaskSignals;`
Die Nummer eines mit `AllocSignal()` allozierten Signals, das bei dem in `pubTask` angegebenen Task gesetzt werden soll, sobald das letzte Window auf dem Screen geschlossen wird.

`pubTask : TaskPtr;`
Der Task, an den das in `pubSig` angegebene Signal geschickt werden. Bitte mit `FindTask()` ermitteln, nicht aus der ExecBase auslesen.

`displayID : LONGINT;`
Dieser Tag ist sehr wichtig. Er gibt den Anzeigemodus an. Eine Liste der möglichen Modi kann man aus der Datenbank der graphics.library erhalten. Man sollte dem Benutzer ermöglichen, mit einem ScreenMode-Requester aus allen möglichen Modes auszuwählen. Einen solchen Requester bieten die `asl.library` und die `reqtools.library` jeweils ab Version 38. Die mit der jetztigen Harware möglichen Modi findet man in `Graphics.def`.

Beispiel: `ntscMonitorID + hireslaceKey` steht für einen Hires-Interlace-NTSC-Screen. Eine sinnvolle Alternative zu einem Screenmode-Requester ist die Übernahme des Modes des Default Public Screens. Wie man dies bewerkstelligt, sieht man im Demo-Programm.

`dClip : RectanglePtr;`
Mit diesem Tag kann man den Bildschirmausschnitt festlegen, in dem der Screen erscheint. Man sollte jedoch besser den `overScan`-Tag verwenden.

`overScan : OScanType;`
Hier hat man die vier Möglichkeiten `text`, `standard`, `max` und `video`. Die ersten beiden Größen kann man mit den Overscan-Preferences einstellen. Default ist `text`.

`showTitle : LONGBOOL;`
Mit diesem Tag kann man bestimmen, ob der Screen-Titelbalken vor oder hinter sogenannten `backdrop`-Windows erscheinen soll.

`behind : LONGBOOL;`
Mit diesem Tag kann man erreichen, daß der Screen hinter allen anderen geöffnet wird.

`quiet : LONGBOOL;`
Normalerweise zeichnet Intuition einen Titelbalken und das Tiefen-Gadget. Mit diesem Tag kann man dies verhindern.

`autoScroll : LONGBOOL;`
Durch diesen Tag scrollt ein übergroßer Screen automatisch, falls man mit dem Mauspfeil gegen den Rand des Bildschirms stößt.

`pens : PenArrayPtr;`
In dem hier anzugebenden Array werden die Pens in dieser Reihenfolge angegeben: detailPen, blockPen, textPen, shinePen, shadowPen, fillPen, fillTextPen, backgroundPen, highLightTextPen. Der Array wird von $FFFF abgeschlossen. Um die Vorgabewerte zu erhalten, sollte man einen Array übergeben, der nur aus einen einzigen Wert ($FFFF) besteht. Dies ist Voraussetzung für den seit 2.0 eingeführten 3D-Look.

`fullPalette: LONGBOOL;`
Normalerweise werden von Intuition nur die Farben 0–3 und 17–19 gesetzt. Durch diesen Tag wird es möglich, alle Farben setzen zu lassen.

Nach dieser zugegebenermaßen langen Beschreibung, ein kleines Beispielprogramm, das Licht ins Dunkel bringt und dabei auch noch zeigt, wie man die Eigenschaften des Default Public Screens kopieren kann. Außerdem wird die Funktion `EasyRequest()` vorgestellt, mit der man sehr einfach Requester erzeugen kann.

```
MODULE ScreenDemo;
FROM Strings IMPORT Str;
IMPORT Intuition AS I,
       Dos       AS d,
       Graphics  AS g,
       Utility   AS u;
```

```
VAR
  DefPubScreen,
  Screen        : I.ScreenPtr;
  DrawInfo      : I.DrawInfoPtr;
  ModeID        : LONGINT;
  Font          : g.TextAttr;
  FontName      : STRING(32);
PROCEDURE Assert (c: BOOLEAN; REF txt : STRING);
VAR
  es : I.EasyStruct;
BEGIN
  IF NOT c THEN
    es := I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Screen Demo",
                        gadgetFormat = "Abort");
    es.textFormat := txt.data'PTR;
    FORGET I.EasyRequest (NIL, es'PTR, NIL);
    HALT (20);
  END;
END Assert;

BEGIN
  DefPubScreen := I.LockPubScreen (NIL);
  Assert (DefPubScreen # NIL, "Unable to lock default Public Screen");

  DrawInfo := I.GetScreenDrawInfo (DefPubScreen);
  Assert (DrawInfo # NIL, "Unable to get DrawInfo");
  ModeID := g.GetVPModeID (DefPubScreen.viewPort'PTR);
  Assert (ModeID # g.invalidID, "Unable to get ModeID");

  Font.ySize := DrawInfo.font.ySize;
  Font.style := DrawInfo.font.style;
  Font.flags := DrawInfo.font.flags;
  FontName   :=Str(DrawInfo.font.name);
  Font.name  := FontName.data'PTR;
  IF DrawInfo # NIL THEN
```



```
    I.FreeScreenDrawInfo (DefPubScreen, DrawInfo); DrawInfo := NIL;
  END;

  Screen := I.OpenScreenTags (NIL,
                              width      : DefPubScreen.width,
                              height     : DefPubScreen.height,
                              depth      : DrawInfo.depth,
                              overScan   : I.text,
                              autoScroll : TRUE,
                              pens       : DrawInfo.pens,
                              font       : Font'PTR,
                              displayID  : ModeID,
                              title      : "Cloned Screen",
                              pubName    : "SCRDEMO",
                              DONE);
  Assert (Screen # NIL, "Unable to open Screen");

  | Hier wird der Screen öffentlich
  FORGET I.PubScreenStatus (Screen, 0)

  d.Delay (5 * 50); | Warten
CLOSE
  IF DefPubScreen # NIL THEN
    I.UnlockPubScreen (NIL, DefPubScreen); DefPubScreen := NIL;
  END;
  IF Screen # NIL THEN
    WHILE NOT I.CloseScreen (Screen) DO

      | Falls der Screen nicht geschlossen werden konnte:
      FORGET I.EasyRequest (NIL,
         I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                       title        = "Screen Demo",
                       textFormat   = "Please close all visitor windows",
                       gadgetFormat = "So I did")'PTR,
                       NIL);
    END;
    Screen := NIL;
  END;
```

```
END ScreenDemo.
```

Den so geöffneten Screen kann man mit ein paar Funktionen beeinflussen:

5.5.1.1 Screenfunktionen

```
PROCEDURE ShowTitle(screen IN A0: ScreenPtr;
                    showIt IN D0: BOOLEAN);
```

Mit dieser Funktion kann man nachträglich ändern, ob der Screen-Titelbalken vor oder hinter backdrop-Windows erscheinen soll.

```
PROCEDURE ScreenToFront(screen IN A0: ScreenPtr);
PROCEDURE ScreenToBack(screen IN A0: ScreenPtr);
```

Bringt den Screen vor bzw. hinter alle anderen Screens.

```
PROCEDURE MoveScreen(screen IN A0: ScreenPtr;
                     dx     IN D0: INTEGER;
                     dy     IN D1: INTEGER);
```

Mit dieser Prozedur kann man Screens unter Programmkontrolle verschieben.

Weitere wichtige Screen-Funktionen:

```
PROCEDURE DisplayBeep(screen IN A0: ScreenPtr);
```

Das sicher jedem bekannte Aufblitzen eines Screens kann mit dieser Prozedur ausgelöst werden.

```
PROCEDURE GetScreenDrawInfo(screen IN A0: ScreenPtr):DrawInfoPtr;
```

Mit dieser Funktion kann man Informationen über einen geöffneten Screen, wie z. B. den benutzten Font einholen. Die `DrawInfo`-Struktur findet man in `Intuition.def`. Es ist zu beachten, daß die enthaltenen Informationen nur solange gültig sind, wie der Screen geöffnet bleibt. Dies muß ggf. mit `LockPubScreen()` sichergestellt werden. Die DrawInfo-Struktur muß nach Gebrauch wieder freigegeben werden.


```
PROCEDURE FreeScreenDrawInfo(screen   IN A0: ScreenPtr;
                             drawInfo IN A1: DrawInfoPtr);
```

Gibt eine DrawInfo Struktur frei. Der Zeiger wird dadurch ungültig.

Speziell für Public Screens sind diese Funktionen von Bedeutung:

```
PROCEDURE LockPubScreen (name IN A0: SysStringPtr): ScreenPtr;
```

Um auf die Screen-Struktur eines Public Screens zugreifen zu können muß man sicherstellen, daß dieser Screen nicht geschlossen wird. Dies wird mit dieser Funktion erreicht. Sie bekommt den Namen des Public Screens oder NIL für den Default Public Screen übergeben. Falls das Ergebnis NIL ist, so existiert der Public Screen nicht. Um ein Window auf einem Public Screen zu öffnen, geht man normalerweise so vor:

```
LockPubScreen()
...
Untersuchung der Screen-Struktur und Anpassung
(Größe, Font) des Windows
an diesen Screen
...
OpenWindow()
UnlockPubScreen()
...
Arbeiten mit dem Window
...
CloseWindow()
```

```
PROCEDURE UnlockPubScreen(name   IN A0: SysStringPtr;
                          screen IN A1: ScreenPtr);
```

Hiermit wird der Public Screen wieder freigegeben. Der ScreenPtr ist danach ungültig und sollte auf NIL gesetzt werden.

5.5.2 Windows

Mit einem Screen an sich kann man eigentlich nichts vernünftiges anfangen. Er dient vielmehr als Grundlage für Windows. Der Screen bestimmt die Anzahl der Farben, die Farben selbst, sowie die Auflösung, die den Windows auf diesem Screen zur Verfügung steht.

Falls man ein Window öffnen will, muß man sich entscheiden, ob man dieses Window auf dem Default Public Screen (normalerweise die Workbench), einem frei wählbaren Public Screen oder einem eigenen Screen öffnen will. Normalerweise sollte man hier dem Benutzer, wie im Interface Style Guide vorgeschlagen, die freie Wahl lassen. Zum Öffnen eines Fensters dient die Prozedur `OpenWindowTags()`;

```
PROCEDURE OpenWindowTags(newWindow IN A0: NewWindowPtr;
                         tagList   IN A1: LIST OF WindowTags
                         ):WindowPtr;
```

Der erste Parameter ist analog zu `OpenScreenTags()` ein Zeiger auf eine 1.3 kompatible NewWindow-Struktur, den man auf `NIL` sezten sollte. Die möglichen Tags sind:

5.5.2.1 WindowTags

`left, top, width, height : LONGINT;`
: Mit diesen vier Werten bestimmt man die Ausmaße des Windows. Hierbei ist besonders darauf zu achten, daß man sich mit den Ausmaßen des Windows an den Font und die Auflösung des Screens anpaßt, auf dem das Window geöffnet wird. Windows, die den ganzen Screen ausfüllen ohne die Titelzeile zu verdecken, sollten diese Ausmaße haben:

  `left = 0`; `top = Screen^.barHeight+1`;

  `width = Screen^.width`;

  `height = Screen^.height-(Screen^.barHeight+1)`.

  Ansonsten muß man darauf achten, daß man mit verschiedenen Fonts zurecht kommt. Eine einfache Methode unabhängig vom

  Font im Fensterinneren einen verwendbaren Bereich einer bestimmten Größe zu erhalten bieten die Tags `innerWidth` und `innerHeight`, die man anstelle von `width` und `height` verwenden kann. Will oder kann man diese jedoch nicht verwenden, so erhält man mit `Screen^.barHeight-Screen^.barVBorder+Screen^.wBorTop` die Größe der Window-Titelzeile und mit `Screen^.wBorLeft`, `Screen^.wBorRight` und `Screen^.wBorBottom` die Größe der anderen Fensterränder.

`minWidth, minHeight : LONGINT; maxWidth, maxHeigth : LONGCARD;`
: Falls das Window in der Größe veränderbar ist, werden hier die minimalen und maximalen Ausmaße angegeben. Die Maximalwerte können auch -1 sein, was bedeutet, daß das Window beliebig groß werden darf.

`innerWidth, innerHeigth : LONGINT;`
: Diese beiden Tags kann man anstelle von `width` und `height` verwenden. Sie vereinfachen das fontsensitive Programmieren, da man mit ihnen die innere Größe des Windows angeben kann und sich somit nicht um die Breite und Höhe der Fensterrahmen kümmern muß. Durch diesen Tag wird die Benutzung von eigenen Gadgets im Fensterrand sehr eingeschränkt.

`detailPen, blockPen : LONGCARD;`
: Diese beiden Tags sind auf NewLook-Screens, welche unter OS 2.0 Standard sind, überflüssig.

`IDCMP : IDCMPFlagSet;`

`flags : WindowFlagSet;`
: Diese beiden FlagSets werden in einem eigenen Abschnitt erklärt.

`sizeGadget, dragBar, depthGadget, closeGadget,`

`backDrop, reportMouse,noCareRefresh,`

`borderless, activate, RMBTrap,`

`wBenchWindow,simpleRefresh, smartRefresh,`

`sizeBRight, sizeBBotton,gimmeZeroZero : LONGBOOL;`
Diese Tags entsprechen einzelnen Bits im WindowFlagSet. Sie werden in dem entsprechenden Abschnitt erklärt.

`autoAdjust : LONGBOOL;`
Mit diesem Tag erlaubt man Intuition, die Position und Größe des Windows zu ändern, damit es auf den Screen paßt.

`menuHelp : LONGBOOL;`
Durch diesen Tag wird des IDCMPFlag `menuHelp` zum Leben erweckt. Näheres im entsprechenden Abschnitt.

`gadgets : GadgetPtr;`
Hier wird das erste einer Liste eigener Gadgets eingetragen.

`checkmark : ImagePtr;`
Wenn es unbedingt sein muß, kann man dem Menühaken ein eigenes Aussehen geben.

`title : SysStringPtr;`
Der Titel des Windows.

`screenTitle : SysStringPtr;`
Der Titel, den der Screen bekommen soll, falls das Window aktiv ist.

`customScreen : ScreenPtr;`
Falls man das Window auf einem eigenen Screen öffnen will, so muß man hier einen Zeiger auf diesen angeben.

`pubScreenName : SysStringPtr;`
Soll das Window auf einem Public Screen geöffnet werden, so muß man hier dessen Namen angeben.

`pubScreen : ScreenPtr;`
Anstelle von `pubScreenName` kann man auch diesen Tag verwenden, falls man einen gültigen Zeiger auf diesen PubScreen hat. Der Zeiger ist nur gültig, falls man schon ein Window auf diesem Screen offen hat oder ihn mit `LockPubScreen()` gesichert hat.

`pubScreenFallBack : LONGBOOL;`
Dieser Tag ermöglicht es Intuition das Fenster auf dem Default Public Screen zu öffnen, falls der gewünschte Public Screen nicht vorhanden ist.

`superBitMap : BitMapPtr;`
Mit diesem Tag kann man dem Window eine selbsterstellte Bitmap zuweisen, die auch größer als die maximale Größe des Windows sein kann. Näheres kann aus Platzgründen hier nicht erklärt werden.

`zoom : IBoxPtr;`
Mit diesem Tag kann man die Position und Größe bestimmen, die das Window nach dem Anklicken des Zoom-Gadgets bekommen soll. Durch diesen Tag bekommen auch Windows ohne Sizing-Gadget ein Zoom-Gadget.

`mouseQueue, RPTQueue : LONGINT;`
Falls das Programm Mausbewegungen und Tastendrücke nicht schnell genug bearbeiten kann, so kann mit diesen Tags angegeben werden, wieviele dieser Nachrichten maximal aufgestaut werden sollen.

`backFill : HookPtr;`
Hier kann man eine Prozedur angeben, die verwendet wird, um den Hintergrund des Windows zu füllen. Die Einzelheiten werden in diesem Handbuch jedoch nicht geklärt. Näheres findet man wie immer im RKM Libraries.

5.5.2.2 Window-Flags

Folgende Flags in dem Tag `flags` können gesetzt werden:

`windowSizing, windowDrag, windowDepth, windowClose`
Mit diesen Bits kann man bestimmen, welche System-Gadgets das Window bekommen soll. Ohne guten Grund sollte ein Window mindestens alle bis auf `windowSizing` haben.

`sizeBRight, sizeBBottom`
: Falls das Window ein Size-Gadget besitzt kann man bestimmen, ob dieses den rechten, den unteren oder beide Rahmen des Fensters belegen soll.

`simpleRefresh`
: Normalerweise restauriert Intuition den Inhalt von Fenstern automatisch, wenn diese von anderen Fenstern überlagert werden. Durch dieses Bit spart man Speicher, ist aber selbst dafür verantwortlich, den Fensterinhalt zu restaurieren, falls man die entsprechenden IDCMP-Messages bekommt.

`superBitMap`
: Das Window hat eine mit dem Tag `superBitMap` angegebene Bitmap.

`backDrop`
: Das Window liegt immer hinter allen anderen Windows. Es darf kein Tiefen-Gadget haben. Dieses Flag sollte nur in Verbindung mit dem Flag `borderLess` und Fenstern, die den ganzen Hintergrund eines eigenen Screens abdecken, benutzt werden. Wird gerne verwendet, wenn ein Programm direkt auf einem eigenen Screen ohne Fenster arbeiten soll, aber dennoch Benutzereingaben verarbeitet werden sollen.

`borderless`
: Das Window hat keinen Rahmen. Es darf deshalb auch keine System-Gadgets haben. Weiteres siehe `backDrop`.

`reportMouse`
: Falls man die Mausbewegungen mitbekommen möchte muß man dieses Flag setzen.

`gimmeZeroZero`
: Ein Fenster, bei dem dieses Flag gesetzt ist, erhält einen getrennten RastPort für den Fensterrahmen und für den Fensterinhalt. Dadurch kann man einfach in das Fenster zeichnen, ohne sich um die Rahmengröße zu kümmern, denn die Position (0/0) ist

immer die linke, obere Ecke des Fensterinhaltes. Dieses Flag sollte man aus verschiedensten Gründen nach Möglichkeit nicht verwenden, vor allem, da dieser Fenstertyp viel Systemperformance verschlingt. Falls möglich sollte man die ClipRects der `graphics.library` verwenden.

`activate`
: Das Window wird beim Öffenen aktiviert.

`rmbTrap`
: Dieses Flag verhindert die Menüauswahl durch die rechte Maustaste. Man bekommt stattdessen eine `mouseButtons`-IDCMP-Message, falls dies gewünscht wird. Man sollte dieses Flag für alle Windows ohne Menüs setzten.

`noCareRefresh`
: Falls man die Grafik im Window nicht erneuern will, wenn sie durch andere Windows verdeckt wurde, so kann man dieses Flag setzten. Man ist dadurch auch davon befreit, sich um die IDCMP-Message `refreshWindow` zu kümmern.

Alle Flags existieren auch als getrennte Tags vom Typ `LONGBOOL`.

5.5.2.3 Der IDCMP

Die Kommunikation zwischen Intuition und einem Anwenderprogramm läuft normaleweise über den Message-Port eines Windows.

Um zu erfahren, was der Benutzer mit einem Fenster, sowie dessen Menüs und Gadgets anstellt, werden zu jeder dieser Aktionen Messages verschickt, falls dies gewünscht wird. Die Message hat immer die gleiche Struktur:

```
IntuiMessage = RECORD OF Message
                 class        : IDCMPFlagSet;
                 code         : CARDINAL;
                 qualifier    : QualifierSet;
                 iAddress     : ANYPTR;
                 mouseX,
                 mouseY       : INTEGER;
                 seconds,
                 micros       : LONGCARD;
                 idcmpWindow  : WindowPtr;
                 specialLink  : IntuiMessagePtr
               END;
```

Die Felder im einzelnen:

class Gibt an, um welche Art vom Ereignis es sich handelt. Die möglichen Bits werden weiter untern erklärt.

code Je nach `class` hat dieses Feld eine andere Bedeutung. Diese Bedeutung wird bei dem entsprechenden Flag erklärt.

qualifier Mit Hilfe dieses Feldes kann man herausfinden, ob z. B. die Shift- und/oder Ctrl-Taste gedrückt ist. Näheres s. u.

iAddress Ähnlich wie bei `code` hängt die Bedeutung dieses Feldes von der Art der Message ab. Man sollte deshalb nie auf diesen Zeiger zugreifen, bevor man nicht `class` geprüft hat. Bei `gadgetUp` und `gagdetDown` enthält `iAddress` einen `GadgetPtr`. Bei `rawKey` einen Zeiger auf Information für Dead-Keys und bei `idcmpUpdate` einen Zeiger auf eine Tag-Liste. Bei allen anderen Messages ist der `iAddress`-Zeiger ungültig. Also Vorsicht!

Wird jedoch `GetIMsg()` aus der `gadtools.library` verwendet, um die Messages abzuholen, so enthalten auch Messages vom Typ `mouseMove` in `iAddress` einen Zeiger auf ein Gadget. Dies ist jedoch nur im Falle von `GetIMsg()` der Fall. Also Vorsicht!

mouseX, mouseY Diese beiden Variablen beinhalten die Koordinaten des Mauspfeils relativ zur linken oberen Ecke des Windows. **Achtung:** Auch bei `gimmeZeroZero`-Fenstern sind die Koordinaten relativ zur linken, oberen Ecke des Fensters und nicht zu der der Zeichenfläche. Das IDCMPFlag `deltaMove` verändert dieses Verhalten. Näheres bei der Erklärung zu diesem Flag.

seconds, micros Die abgelaufenen Sekunden und Mikrosekunden seit dem 01.01.1978.

idcmpWindow Ein Zeiger auf das Fenster, zu dem diese Message gehört.

specialLink Dieses Feld ist nur für Intuition bestimmt.

Wie bereits angedeutet, enthält `class` die Art des Ereignisses. Dabei ist zu beachten, daß nur die Ereignisse dem Fenster gemeldet werden, deren Flags im Tag `IDCMP` gesetzt worden sind. Damit ist sichergestellt, daß das Programm nur die Benutzereingaben erhält, die für es von Interesse sind.

Hier eine geordnete Übersicht über alle IDCMPFlags:

5.5.2.3.1 Mausabfrage:

mouseButtons Diese Art von IntuiMessage informiert über Klicks mit den zwei (oder drei) Maustasten. In `code` ist die Information enthalten, welche Maustaste gedrückt oder losgelassen wurde. Die möglichen Werte sind in `Input.def` definiert:

lButton Die linke MausTaste wurde gedrückt.

mButton Die mittlere Maustaste wurde gedrückt.

rButton Die rechte Maustaste wurde gedrückt. Falls man auch etwas von der rechten Maustaste mitbekommen möchte, sollte man das WindowFlag `rmbTrap` setzen.

Addiert man zu diesen Konstanten die Konstante `upPrefix` aus `Input`, erhält man die Werte für Code, wenn die entsprechende Taste losgelassen wurde.

mouseMove Durch dieses Flag bekommt man alle Bewegenungen der Maus mit. Die aktuellen Koordinaten stehen in den entsprechenden Felder der IntuiMessage-Struktur.

deltaMove Dieses Flag steht für keine eigene IntuiMessage-Art. Es ändert nur Bedeutung der `mouseX-` und `mouseY`-Felder der IntuiMessages. Diese haben dann nur noch bei `mouseMove` oder `mouseButtons` Werte ungleich 0. Diese Werte beschreiben die Position der Maus reativ zur letzten Message dieser Art.

5.5.2.3.2 Gadget-Abfrage:

gadgetDown Ein Gadget wurde angeklickt. In `iAddress` steht die Adresse dieses Gadgets.

gadgetUp Ein Gadget wurde wieder losgelassen. In `iAddress` steht die Adresse dieses Gadgets.

closeWindow Der Benutzer hat das Close-Gadget des Windows angeklickt.

5.5.2.3.3 Menü-Abfrage:

menuPick Es wurde (möglicherweise) eines oder mehrere Menu-Items ausgewählt. Näheres dazu im Abschnitt über Menüs.

menuHelp Falls das WindowFlag `menuHelp` gesetzt wurde, bekommt man diese Message geschickt, wenn der Benutzer die Help-Taste drückt, während sich der Mauspfeil über einem Menü befindet. Das Menü findet man wie bei `menuPick` heraus.

menuVerify Ist dieses Flag gesetzt, erwartet Inituition eine Bestätigung des Programms, ob das Menü in diesem Moment gezeichnet werden darf oder nicht. Das `code`-Feld ist entweder `menuWaiting`, falls ein anderes Window auf dem selben Screen ein Menü aufklappen will, `menuHot`, falls das Menu des eigenen Windows ausgeklappt werden soll. Durch ändern dieses Feldes auf `menuCancel` wird die Menüoperation abgebrochen. Man bekommt dann ersatzweise eine `mouseButtons`-Message geschickt. Wie alle Verify-Messages muß diese Message sehr schnell bearbeitet und an Intuition zurückgeschickt werden. Während man sich Verify-Messages schicken läßt, darf man z. B. keine DOS-Funktionen aufrufen, da diese Requester hervorrufen können. Damit würden sich das Programm, da es auf die Verify-Message nicht antworten kann und Intuition, das keine Antwort bekommt, gegenseitig blockieren. Deshalb sollte man Verify-Messages nur verwenden, wenn sie wirklich nötig sind und sie mit `ModifyIDCMP()` ausschalten, falls man irgendwelche Aktionen durchführt, die DOS-Operationen und/oder das Öffnen von Requestern nach sich ziehen könnten.

5.5.2.3.4 Requester:

reqSet, reqClear, reqVerify Diese drei Flags beschäftigen sich mit echten Requestern, die viele Nachteile haben und deshalb besser nicht verwendet werden.

5.5.2.3.5 Window:

newSize Die Größe des Windows wurde möglicherweise verändert.

refreshWindow Falls es sich bei dem Fenster um kein Smartrefresh-Window handelt, und der Inhalt des Windows von einem anderen Window überlagert wurde, bekommt man diese Message geschickt. Man muß dann mindestens die Intuition Funktionen `BeginRefresh()` und `EndRefresh()` aufrufen. Falls nötig kann man zwischen diesen beiden Funktionen auch mit Funktionen der graphics.library den Inhalt des Windows restaurieren. Werden

GadTools-Gadgets verwendet, so müssen die entsprechenden Funktionen aus der `gadtool.library` verwendet werden. Durch das WindowFlag `noCareRefresh` kann man sich von der IDCMP-Message `refreshWindow` befreien, falls man keinen eigenen Window-Refresh durchführen will.

sizeVerify Durch diese Message bekommt man mittgeteilt, daß der Benutzer beabsichtigt, die Größe des Fensters zu verändern. Das Programm hat nun die Möglichkeit dies zuzulassen, indem es die Message einfach zurückschickt oder es zu verhindern, indem es zuvor in das `code`-Feld der Message `okCancel` einträgt. Einige wichtige Anmerkungen zu den Verify-Messages stehen unter `menuVerify`.

activeWindow, inactiveWindow So erfährt man, ob das Window aktiviert/deaktiviert wurde.

5.5.2.3.6 Tastaturabfrage:

vanillaKey Es wurde eine Taste gedrückt, der ein ASCII-Code zugeordnet werden kann. Dieser Code findet sich im `code`-Feld.

rawKey Mit diesem IDCMP-Flag kann man auch Tastendrücke für Tasten wie F1 oder ESC empfangen. In `code`-Feld seht der Tastencode der zu dieser Taste gehört. `ESC = $45, F1-F10 = $50-$59, DEL = $46, Help = $5F, Backspace = $41, Tab = $42`.

Die Cursortasten sind sind in Intuition als Konstanten definiert. Mit der Funktion `MapRawKey()` aus der keymap.library kann man diese Tastencodes, falls möglich, nach ASCII umwandeln. Beispiel folgt gleich.

Das folgende Beispiel zeigt, wie eine Tastaturabfrage aussehen könnte:

```
TYPE
  KeyType = (esc=$45,del,f1=$50,f2,f3,f4,f5,f6,f7,f8,f9,f10,
             help=$5F,normalkey,nokey);

VAR
  Code  : CHAR;
```



```
  event : InputEvent;
  Type  : KeyType;
  m     : IntuiMsgPtr;
...

IF m.code OF 0..13 THEN
  Type:=KeyType(m.code)
  Code:=&0;
ELSE
  event.code:=m.code;
  event.qualifier:=m.qualifier;
  event.eventAddress:=LongPtr(m.iAddress)^;
  IF MapRawKey(event'PTR,Code'PTR,1,NIL)=1 THEN
    Type:=NormalKey
  ELSE
    Type:=NoKey;
    Code:=&0;
  END;
END;
```

Der Funktion `MapRawKey()` wird ein InputEvent, ein Puffer, in den das konvertierte Zeichen geschrieben werden soll, die Länge des Puffers und eine Keymap übergeben. Übergibt man für die Keymap NIL, wird die Standardtastaturbelegung verwendet. Als Ergebnis erhält man die Anzahl der konvertierten Zeichen, ist diese in unserem Fall ungleich 1, ließ sich das Zeichen nicht konvertieren.

5.5.2.3.7 Verschiedenes:

newPrefs Ist seit OS 2.0 nicht mehr sinnvoll.

diskInserted, diskRemoved Es wurde eine Diskette eingelegt bzw. aus dem Laufwerk genommen.

intuiTicks Diese Messages bekommt man ungefähr zehn mal pro Sekunde geschickt, falls das Window aktiv ist. Es wird nur eine neue Message geschickt, falls die vorige bereits abgeholt wurde.

idcmpUpdate Diese Message hat mit sogenannten „BOOPSI"-Gadgets zu tun haben. Diese Art von Gadgets wird im RKM Libraries genauer erklärt.

changeWindow Diese Message bekommt man, falls sich das Window in Position oder Größe geändert hat.

Noch einmal zur Erinnerung: Man erhält nur die Ereignisse gemeldet, deren IDCMP-Flags man beim Öffnen des Fensters angegeben hat.

5.5.2.4 IDCMP-Abfrage

Der IDCMP-Port ist ein ganz normaler Exec-Messageport, für den man die Funktionen der exec.library verwenden kann. Eine Ausnahme gibt es, falls man GadTools-Gadgets oder Menüs verwendet. In diesem Fall muß man anstatt `GetMsg()` und `ReplyMsg()` aus Exec, die Funktionen `GetIMsg()` und `ReplyIMsg()` aus GadTools verwenden. Die Hauptschleife eines Programms sieht eigentlich immer gleich aus. Es ist sehr wichtig, daß zuerst mit `Wait()` oder `WaitPort()` auf das Signalbit der Window-UserPorts gewartet wird und dann solange mit `Get(I)Msg()` die Messages abgeholt werden, bis diese Funktion `NIL` zurück gibt.
Nach `Get(I)Msg()` sollte man die wichtigen Werte aus der IntuiMessage-Struktur kopieren und die Message dann so schnell als möglich mit `Reply(I)Msg()` zurückschicken. Achtung, nachdem man die Message zurückgeschickt hat, darf man nicht mehr auf sie zugreifen.
Da wir in unseren Beispielprogrammen nur GadTools verwenden, um Menus und Gadgets zu erstellen und Sie dies aus guten Gründen auch tun sollten, wird hier das Paar `GetIMsg()/ReplyIMsg()` verwendet.


```
REPEAT
  WaitPort(Window.userPort);

  IMsg:=GetIMsg(Window.userPort);
  WHILE Imsg # NIL DO
    |
    | Kopieren der wichtigen Felder der IntuiMessage
    |
    ReplyIMsg (IMsg)
    |
    | Auswerten der Message, ggf. Quit auf TRUE setzen
    |
    IMsg := GetIMsg(Window.userPort)
  END;
UNTIL Quit;
```

5.5.2.5 Window-Funktionen

Nachdem man ein Fenster geöffnet hat, bietet Intuition verschiedene Funktionen, um dieses zu bearbeiten. Hier die wichtigsten:

```
PROCEDURE CloseWindow(window IN A0 : WindowPtr);
```

`CloseWindow()` schließt ein Fenster.

```
PROCEDURE ActivateWindow(window IN A0 : WindowPtr);
```

Aktiviert ein Fenster.

```
PROCEDURE ModifyIDCMP(window   IN A0 : WindowPtr;
                      flags    IN D0 : IDCMPFlagSet);
```

Dient dazu die IDCMPFlags eines Windows zu verändern. `flags` sind dabei die neuen Flags.

```
PROCEDURE MoveWindow( window IN A0: WindowPtr;
                      dx     IN D0: LONGINT;
                      dy     IN D1: LONGINT );
```

Bewegt ein Fenster um `dx` Pixel in X-Richtung, `dy`-Pixel in Y-Richtung.

```
PROCEDURE RefreshWindowFrame( window IN A0: WindowPtr );
```

Zeichnet den Rahmen eines Fensters neu.

```
PROCEDURE SetWindowTitles(  window      IN A0: WindowPtr;
                            windowTitle IN A1: SysStrPtr;
                            screenTitle IN A2: SysStrPtr );
```

Setzt einen neuen Titel für das Fenster und einen neuen Screentitel, der dann angezeigt werden soll, wenn daß Fenster aktiv ist. Will man eine leere Titelleiste, muß man als Zeiger für den String `NIL` übergeben. Will man, daß einer der Titel beibehalten wird, muß man `ANYPTR(-1)` übergeben.

```
PROCEDURE SizeWindow( window IN A0: WindowPtr;
                      dx     IN D0: LONGINT;
                      dy     IN D1: LONGINT );
```

Verändert die Größe eines Windows um `dx` Pixel in der Breite und `dy` Pixel in der Höhe. **Achtung:** Diese Prozedur überprüft nicht die ihr übergebenen Werte. Sie müssen sich also selbst darum kümmern, daß diese möglich sind. Ansonsten kommt es zu einem Systemabsturz.

```
PROCEDURE ChangeWindowBox( window IN A0: WindowPtr;
                           left   IN D0: LONGINT;
                           top    IN D1: LONGINT;
                           width  IN D2: LONGINT;
                           height IN D3: LONGINT );
```

Mit dieser Prozedur kann man die Position und die Größe des Fensters auf einmal verändern. Zu beachten ist dabei, daß es sich diesmal um abslute Koordinaten handelt.


```
PROCEDURE WindowLimits( window    IN A0: WindowPtr;
                        widthMin  IN D0: LONGINT;
                        heightMin IN D1: LONGINT;
                        widthMax  IN D2: LONGCARD;
                        heightMax IN D3: LONGCARD ): LONGBOOL;
```

Bei Fenstern mit einem Größengadget, kann man mit dieser Funktion nachträglich die Minimal- und Maximalgröße verändern. Wollen Sie einen Wert nicht verändern, so übergeben Sie 0. Wollen Sie, daß das Fenster so groß wie die ganze Screen werden kann, übergeben Sie -1.

```
PROCEDURE WindowToBack( window IN A0 : WindowPtr );
```

Bringt ein Fenster hinter alle anderen auf den Screen, ausgenommen Backdropwindows.

```
PROCEDURE WindowToFront( window IN A0 : WindowPtr );
```

Bringt ein Fenster in den Vordergrund.

```
PROCEDURE MoveWindowInFrontOf( window       IN A0 : WindowPtr;
                               behindWindow IN A1 : WindowPtr );
```

Legt das Fenster `window` vor – hinsichtlich der Tiefenanordnung – das Fenster `behindWindow`.

```
PROCEDURE ReportMouse( flag   IN D0: LONGBOOL;
                       window IN A0: WindowPtr );
```

Mit dieser Prozedur kann man im nachhinein das Flag `reportMouse` in `WindowFlags` setzen oder löschen. TRUE, wenn Flag gesetzt, FALSE, wenn es gelöscht werden soll.

Folgendes Beispielprogramm zeigt den Umgang mit Windows:

```
MODULE WindowDemo;
IMPORT
  Intuition AS I,
  Graphics  AS g,
  Exec      AS e,
  Utility   AS u;

TYPE
  PenArray = ARRAY [1] OF CARDINAL;
CONST
  Pens = PenArray:($FFFF);
VAR
  Screen,
  PubScreen : I.ScreenPtr;
  Window    : ARRAY [2] OF I.WindowPtr;

PROCEDURE Assert (c : BOOLEAN; REF txt : STRING);
VAR es : I.EasyStruct;
BEGIN
  IF NOT c THEN
    es := I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Screen Demo",
                        gadgetFormat = "Abort");
    es.textFormat := txt.data'PTR;
    FORGET I.EasyRequest (NIL, es'PTR, NIL);
    HALT (20);
  END;
END Assert;

BEGIN
  Screen := I.OpenScreenTags (NIL,
                              depth     : 2,
                              pens      : Pens'PTR,
                              sysFont   : I.wbScreenFont,
                              displayID : g.hiresKey,
                              title     : "Window Demo",
                              pubName   : "WINDEMO",
```



```
                              DONE);
  Assert (Screen # NIL, "Unable to open Screen");

  Window[0] := I.OpenWindowTags
                    (NIL,
                     left         : 0,
                     top          : Screen.barHeight + 1,
                     width        : Screen.width,
                     height       : Screen.height -
                                    (Screen.barHeight + 1),
                     screenTitle  : "Backdrop-Window is active",
                     customScreen : Screen,
                     flags        : {I.noCareRefresh, I.backDrop,
                                     I.borderless, I.rmbTrap},
                     DONE);
  Assert (Window[0] # NIL, "Unable to open Backdrop-Window");

  FORGET I.PubScreenStatus (Screen, 0);

  | simulate to open on a pubscreen *)
  PubScreen := I.LockPubScreen ("WINDEMO");
  Assert (PubScreen # NIL, "Unable to lock PubScreen");
  | now it's time to adapt to the PubScreen *)

  Window[1] := I.OpenWindowTags
                       (NIL,
                        left        : 100,
                        top         : 50,
                        innerWidth  : 400,
                        minWidth    : 100,
                        maxWidth    : Screen.width,
                        innerHeight : 100,
                        minHeight   : 50,
                        maxHeight   : Screen.height,
                        title       : "Work-Window",
                        screenTitle : "Work-Window is active",
                        pubScreen   : Screen,
                        flags       : {I.noCareRefresh,I.windowClose,
```

```
                                      I.windowSizing,I.windowDrag,
                                      I.windowDepth,I.rmbTrap,
                                      I.activate},
                          IDCMP       : {I.closeWindow},
                          DONE);
  Assert (Window[1] # NIL, "Unable to open Work-Window");
  I.UnlockPubScreen ("WINDEMO", PubScreen); PubScreen := NIL;

  FORGET e.WaitPort (Window[1].userPort); | very simplified event-loop
CLOSE
  IF Window[1] # NIL THEN
    I.CloseWindow (Window[1]); Window[1] := NIL;
  END;
  IF PubScreen # NIL THEN
    I.UnlockPubScreen ("WINDEMO", PubScreen); PubScreen := NIL;
  END;
  IF Window[0] # NIL THEN
    I.CloseWindow (Window[0]); Window[0] := NIL;
  END;
  IF Screen # NIL THEN
    WHILE NOT I.CloseScreen (Screen) DO
      FORGET I.EasyRequest (NIL,
         I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                       title        = "Screen Demo",
                       textFormat   = "Please close all visitor windows",
                       gadgetFormat = "So I did")'PTR,
                       NIL);
    END;
    Screen := NIL;
  END;
END WindowDemo.
```


5.5.2.6 Mauszeigerdefinition

Intuition bietet die Möglichkeit, jedes Fenster mit einem eigenen Mauszeigersprite zu versehen, der sichtbar wird, sobald das Fenster aktiv ist. Will man einen eigenen Mauszeiger verwenden, so kann man sich diesen mit folgender Prozedur definieren:

```
PROCEDURE SetPointer( window  IN A0 : WindowPtr;
                      pointer IN A1 : ANYPTR;
                      height  IN D0 : LONGINT;
                      width   IN D1 : LONGINT;
                      xOffset IN D2 : LONGINT;
                      yOffset IN D3 : LONGINT );
```

Dabei gibt **window** das Fenster an, für das die Zeigerdefinition gilt, **pointer** einen Pointer auf die Graphikdaten, **height** die Höhe und **width** die Breite des neuen Mauszeigers. **xOffset/yOffset** gibt die Position des Hotspots (des Punktes, mit dem wirklich geklickt wird) relativ zur linken oberen Ecke des Mauszeigers an.

Bei den Pointerdaten handelt es sich um eine Sprite-Struktur. Da Sprites nur 16 Punkte (Bits) breit sein können, aber beliebig viele Punkte hoch, definieren Sie die Spritedaten am besten als konstantes **ARRAY OF** CARDINAL, da eine Cardinalzahl 16 Bits belegt. Da der Mauszeiger vierfarbig ist, besitzt er 2 Bitplanes ($2^2 = 4$). Für jede Zeile des Sprites brauchen wir also zwei Words (ein Word = 16 Bit), also zwei Cardinalzahlen. Jenachdem ob in beiden Bitplanes ein Bit gesetzt ist, bekommt der Punkt eine entsprechende Farbe (Siehe Graphics: Flächen und Muster).

Zu den Words, die die Grafikdaten enthalten benötigt das System am Anfang und am Ende des Arrays noch jeweils zwei Words, die man gleich Null setzen sollte.

Beispiel:

```
TYPE
  Data = ARRAY OF CARDINAL;
```

```
CONST             | erste Bitplane   zweite Bitplane
  Pointer = Data:(0,0                                (* System  *)
                  %0000001111000000,%0000011001100000, (* 1.Zeile *)
                  %0000011001100000,%0000100000010000, (* 2.Zeile *)
                  %0000001111000000,%0000011001100000, (* 3.Zeile *)
                  0,0);                              (* System  *)
..
BEGIN
  SetPointer(Window,Pointer'PTR,3,16,-7,-1);
```

Achtung: Alle Graphikdaten müssen im ChipMem liegen.

Will man einen selbstdefinierten Zeiger wieder löschen, verwenden Sie dazu `ClearPointer()`. An Stelle des eigenen tritt dann beim Anklicken ihres Fensters der vordefinierte Mauszeiger von Intuition, war ihr Fenster das aktive, vollzieht sich der Wechsel sofort.

```
PROCEDURE ClearPointer( window IN A0: WindowPtr );
```


5.5.3 Menüs

Nach langer und beschwerlicher Vorarbeit (Screen und Windows) ist es nun endlich möglich, zu den interessanteren Dingen von Intuition vorzustoßen, den Menüs und Gadgets. Bei der Erstellung dieser Hauptbestandteile der Benutzeroberfläche wird man seit OS 2.0 von Gadtools unterstützt. Durch sie wird die ursprünglich zeitaufwendige und umständliche Art der direkten Intuition-Programmierung umgangen.

Die Menüerstellung könnte wirklich nicht einfacher sein. Es gibt nur eine einzige Struktur, die bei der Erstellung von Menüs wichtig ist.

```
NewMenu = RECORD
            type          : NewMenuType;
            label         : SysStringPtr;
            commKey       : SysStringPtr;
            flags         : NewMenuFlagSet;
            mutualExclude : LONGSET;
            userData      : ANYPTR;
          END;
```

Mit einem Array dieser Struktur wird das gesamte Menü beschrieben. Die Felder der NewMenu-Struktur im einzelnen:

type Dieses erste Feld der NewMenu-Struktur enthält den Typ, den diese Struktur beschreibt. Diese Typen stehen zur Verfügung:

- **title** Die Struktur beschreibt den Anfang eines Menütitels, der dann in der Screen-Titelzeile erscheint.
- **item, imageItem** Diese beiden Typen erzeugen einen Menüpunkt im aktuellen Menü. Dabei hat man die Wahl, ob der Menüpunkt aus Text (item) oder einer Grafik (imageItem) besteht.
- **sub, imageSub** Diese beiden Typen erzeugen einen Unter-Menüpunkt zum aktuellen Menüpunkt. Einmal nur Text, einmal eine Grafik.
- **end** Das letzte Element der NewMenu-Arrays muß diesen Typ haben.

label Je nachdem, ob der NewMenu-Eintrag einen Menü-Text oder eine Grafik beschreibt, muß hier ein Zeiger auf einen SysString oder ein Zeiger auf eine Image-Struktur stehen. Der Zeiger muß für die gesamte Existenz des Menüs gültig bleiben. Anstelle eines `SysStringPtr` kann man auch die Konstante `barLabel` angeben, die zu einem Trennstrich führt.

commKey Hier kann ein Zeiger auf ein Zeichen, das der Shortcut für den Menüpunkt ist, angegeben werden. Achtung, hier darf nicht nur ein einzelnes Zeichen stehen, sondern ein `SysString` mit Nullbyte am Ende, z. B. `commKey: "c"+&0.data'PTR`.

flags Bei den Flags kann man `menuDisabled` bzw. `itemDisabled` angeben, falls das ganze Menü bzw. ein einzelner Menüpunkt nicht anwählbar sein darf.

Bei Menüpunkten, die mit einem Haken versehen werden können, muß man außerdem noch `checkIt` und `menuToggle` setzen. Mit `checked` kann man dann auch noch bestimmen, ob der Haken gleich am Anfang gesetzt sein soll.

mutualExclude Menüpunkte mit Haken können sich gegenseitig ausschließen. Mit diesem Set kann man die Menüpunkte angeben, die ausgeschlossen werden sollen. Beispiel: Dies ist der zweite Eintrag, er soll alle anderen Einträge ausschließen. mutualExclude muß auf `-LONGSET:`{2} gesetzt werden. Die Nummerierung startet in jedem Menustreifen mit Null für den obersten Menüpunkt.

userData In dieses Feld können Sie beliebige Daten eintragen, am besten eine eindeutige ID, um das Item wiederzuerkennen, wenn es angewählt wurde. Dadurch erspart man sich die umständliche Auswertung des `code`-Feldes der IDCMP-Message. Außerdem ist diese Art der Auswertung unabhängig vom geometrischen Aufbau des Menüs, d. h. auch wenn das Menü verändert wird, wird das Item eindeutig erkannt. Im folgenden Beispielprogramm ist dies deutlich gezeigt.

Um das Menu nun zu definieren, füllt man einfach ein Array mit NewMenu-Strukturen, dabei startet man mit einem `title`-Eintrag, gefolgt von den

Items des Menus. Die folgenden Einträge werden solange als Menüpunkte des zuletzt eingetragenen Titels betrachtet, bis ein weiterer `title-Eintrag` im Feld auftritt.

Ein Untermenü zu erzeugen ist genauso einfach, zuerst ein normaler `item`-Eintrag, der den Namen des Untermenüs enthält. Darauf folgen dann soviel `sub`-Einträge, wie man Untermenupunkte wünscht.

Nachdem man ein Array, das das gewünschte Menü beschreibt, initialisiert hat, kann man es mit `CreateMenus()` erzeugen. Diese Prozedur benötigt nur einen Zeiger auf das Array mit NewMenu-Strukturen und Tags als Parameter. Es gibt diese Tags:

`frontPen : LONGCARD`
: Mit diesm Tag kann man die Farbe angeben, mit der der Text geschrieben werden soll. Da der Vorgabewert (0) vernünftig ist, sollte man diesen Tag eigentlich nie benötigen.

`fullMenu : LONGBOOL`
: Durch diesen Tag erzwingt man, daß nur korrekte und vollständige Menüs erzeugt werden. Dadurch werden Fehler, wie zum Beispiel ein Sub-Item, das direkt auf einen Menü-Titel folgt, entdeckt.

`secondaryErr : POINTER TO CM2ndErr`
: Hier kann man einen Zeiger auf eine Variable angeben, in die der genaue Grund für ein Fehlschlagen von `CreateMenus()` geschrieben wird. Mögliche Fehlergründe sind:

menuOk Kein Fehler aufgetreten.

menuTrimmed Das Menü enthält zuviele Menütitel, Menüpunkte oder Unterpunkte. `CreateMenus()` schlägt in diesem Fall nicht fehl, sondern erzeugt eine zurechtgestutzte Version des Menüs.

menuInvalid Die Menübeschreibung war ungültig (z. B. `sub` direkt nach `title`).

noMem Zuwenig Speicher.

Nachdem die Menüs erzeugt wurden, muß man sie noch layouten lassen, bevor man sie mit `SetMenuStrip()` an ein Window bindet. Dies erledigt die Prozedur `LayoutMenus()`, die eine `VisualInfo`-Struktur benötigt. Diese Struktur kann man sehr einfach mit `GetVisualInfo()` erhalten. An Tags versteht `LayoutMenus()` nur `textAttr: TextAttrPtr`, mit welchen man den Font beschreiben kann, der für das Menü verwendet werden soll. Als Voreinstellung wird der Font des Screens verwendet, was sehr sinnvoll ist.

Bevor man das Window schließen kann, muß man die Menus mit `ClearMenuStrip()` wieder entfernen und kann sie dann mit `FreeMenus()` freigeben.

Das Beispielprogramm sollte alle Klarheiten beseitigen und auch zeigen, wie man richtig mit den IDCMP-Messages umgeht. Eine Besonderheit ist, daß man für mehrere Menüauswahlen nur eine IntuiMessage bekommen kann. Deshalb muß diese Schleife verwendet werden:

```
VAR
  item  : gt.GTMenuItemPtr;
  ...
OF {I.menuPick} THEN
  WHILE code # CARDINAL(I.menuNull) DO
    item := I.ItemAddress (MenuDemoMenus, code);

    | Test welches Item angewählt worden ist.
    IF KEY LONGINT(item.userData)
      OF MID_New THEN
      END
      OF MID_Open THEN
      END
       ...
       ...
    END;
    code := item.nextSelect;
  END; (* WHILE *)
END;
```


Vielleicht ist Ihnen aufgefallen, daß die Variable `item` nicht wie vielleicht erwartet vom Typ `MenuItemPtr` aus Intuition ist, sondern vom Typ `GTMenuItemPtr` aus GadTools ist. Da die ursprüngliche MenuItem-Struktur aus Intuition kein `userData`-Feld kennt, MenuItems, die mit GadTools erzeugt worden sind dieses aber enthalten, muß man, um darauf zugreifen zu können, einen Zeiger auf die von GadTools definierte erweiterte Struktur `GTMenuItem` verwenden.

Bei Menüs, die „abgehakt“ werden können, sollte man die IntuiMessage ignorieren und den Zustand des Hakens (gesetzt oder nicht gesetzt) erst dann abfragen, wenn man ihn wirklich benötigt. Einen gesetzen Haken erkennt man an dem Flag `checked` in den Flags der `GTMenuItem`-Struktur. Um die entsprechende Struktur zu finden, bleibt einem leider nichts anderes übrig, als sich nach Erzeugen der Menüs, sich durch den Menubaum zu hangeln und nach der entsprechenden ID zu suchen. Die einzelnen Elemente hängen in einfach verketteten Listen, die mit NIL terminiert sind.

Hier nun das komplette Beispielprogramm für Menüs:

```
MODULE MenuDemo;

IMPORT Exec      AS e,
       Intuition AS I,
       GadTools  AS gt,
       Graphics  AS g;

EXCEPTION
  NoVisualInfo       : "No visual info";
  NoScreen           : "Could not lock screen";
  NoWindow           : "Could not open window";
  NoMenus            : "Could not create menues";
  CouldNotLayout     : "Could not layout menues";
  CouldNotSet        : "Could not set menu strip";
  LeaveMsgLoop       : "Leave message loop";
VAR
  Scr                : I.ScreenPtr;
  VisualInfo         : ANYPTR;
  MenuDemoWnd        : I.WindowPtr;
```

```
  MenuDemoMenus      : I.MenuPtr;
  Font               : g.TextFontPtr;

CONST
  MID_New            = $01000000;
  MID_Open           = $01010000;
  MID_Save           = $01020000;
  MID_SaveAs         = $01020100;
  MID_Quit           = $01030000;

  MID_Cut            = $02000000;
  MID_Copy           = $02010000;
  MID_Paste          = $02020000;
  MID_Erase          = $02030000;
  MID_Undo           = $02040000;

  MID_StartLearning = $03000000;
  MID_StopLearning  = $03010000;
  MID_AssignMacro   = $03020000;

  MID_CreateIcons   = $04000000;
  MID_Raw           = $04010000;
  MID_Cooked        = $04010100;

  MenuDemoNewMenu = ARRAY OF gt.NewMenu:(
    (gt.title, "Project",         NIL,  {}, {}, NIL),
    (gt.item,  "New",             "N"*, {}, {}, MID_New),
    (gt.item,  "Open...",         "O"*, {}, {}, MID_Open),
    (gt.item,  gt.barLabel,       NIL,  {}, {}, NIL),
    (gt.item,  "Save",            "S"*, {}, {}, MID_Save),
    (gt.item,  "Save As...",      "A"*, {}, {}, MID_SaveAs),
    (gt.item,  gt.barLabel,       NIL,  {}, {}, NIL),
    (gt.item,  "Quit",            "Q"*, {}, {}, MID_Quit),

    (gt.title, "Edit",            NIL,  {}, {}, NIL),
    (gt.item,  "Cut",             "X"*, {}, {}, MID_Cut),
    (gt.item,  "Copy",            "C"*, {}, {}, MID_Copy),
    (gt.item,  "Paste",           "V"*, {}, {}, MID_Paste),
```



```
    (gt.item,  gt.barLabel,       NIL,  {}, {}, NIL),
    (gt.item,  "Erase",           NIL,  {}, {}, MID_Erase),
    (gt.item,  gt.barLabel,       NIL,  {}, {}, NIL),
    (gt.item,  "Undo",            "Z"*, {}, {}, MID_Undo),

    (gt.title, "Macro",           NIL,  {}, {}, NIL),
    (gt.item,  "Start Learning",  NIL,  {}, {}, MID_StartLearning),
    (gt.item,  "Stop Learning",   NIL,  {}, {}, MID_StopLearning),
    (gt.item,  "Assign Macro...", NIL,  {}, {}, MID_AssignMacro),

    (gt.title, "Settings",        NIL,  {}, {}, NIL),
    (gt.item,  "Create Icons?",   "I"*, {gt.checkIt,gt.checked,
                                          gt.menuToggle},
                                         {}, MID_CreateIcons),
    (gt.item,  gt.barLabel,       NIL,  {}, {}, NIL),
    (gt.item,  "Meal",            NIL,  {}, {}, NIL),
    (gt.sub,   "Raw",             NIL,  {gt.checkIt}, {1}, MID_Raw),
    (gt.sub,   "Cooked",          NIL,  {gt.checkIt,gt.checked}, {0},
                                                      MID_Cooked),
    (gt.end,   NIL,               NIL,  {}, {}, NIL));

PROCEDURE Warn (REF txt : STRING);
VAR es := I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Menu Demo",
                        gadgetFormat = "Abort");
BEGIN
  es.textFormat := txt.data'PTR;
  FORGET I.EasyRequest (MenuDemoWnd, es'PTR, NIL);
END Warn;


PROCEDURE SetupScreen;
BEGIN
  Scr := I.LockPubScreen (NIL);
  ASSERT(Scr#NIL,NoScreen);

  VisualInfo := gt.GetVisualInfo (Scr, DONE);
  ASSERT(VisualInfo#NIL,NoVisualInfo);
```

```
END SetupScreen;

PROCEDURE CloseDownScreen;
BEGIN
  IF VisualInfo # NIL THEN
    gt.FreeVisualInfo (VisualInfo);
    VisualInfo := NIL;
  END;
  IF Scr # NIL THEN
    I.UnlockPubScreen (NIL, Scr);
    Scr := NIL;
  END;
END CloseDownScreen;


PROCEDURE OpenMenuDemoWindow;
BEGIN
  MenuDemoMenus := gt.CreateMenusTags(MenuDemoNewMenu,
                                    frontPen : 0,
                                    DONE);
  ASSERT(MenuDemoMenus#NIL,NoMenus);

  ASSERT(gt.LayoutMenus (MenuDemoMenus,VisualInfo,DONE), CouldNotLayout);

  MenuDemoWnd := I.OpenWindowTags ( NIL,
                  left         : 50,
                  top          : 50,
                  innerWidth   : 400,
                  innerHeight  : 50,
                  IDCMP        : {I.menuPick,I.closeWindow,
                                  I.refreshWindow,I.menuHelp},
                  flags        : {I.activate,I.windowSizing,
                                  I.windowDrag,I.windowDepth,
                                  I.windowClose},
                  title        : "Menu Demo",
                  screenTitle  : "Menus in Cluster",
                  pubScreen    : Scr,
                  menuHelp     : TRUE,
```



```
                    minWidth     : 200,
                    maxWidth     : $FFFF,
                    minHeight    : 40,
                    maxHeight    : $FFFF,
                    DONE);
  ASSERT(MenuDemoWnd#NIL,NoWindow);

  ASSERT(I.SetMenuStrip (MenuDemoWnd, MenuDemoMenus),CouldNotSet);
  gt.GT_RefreshWindow (MenuDemoWnd, NIL);
END OpenMenuDemoWindow;


PROCEDURE CloseMenuDemoWindow;
BEGIN
  IF MenuDemoMenus # NIL THEN
    I.ClearMenuStrip (MenuDemoWnd);
    gt.FreeMenus (MenuDemoMenus);
    MenuDemoMenus := NIL;
  END;
  IF MenuDemoWnd # NIL THEN
    I.CloseWindow (MenuDemoWnd);
    MenuDemoWnd := NIL;
  END;
END CloseMenuDemoWindow;


PROCEDURE HandleIDCMP;
VAR
  imsg  : I.IntuiMessagePtr;
  item  : gt.GTMenuItemPtr;
  code  : CARDINAL;
  class : I.IDCMPFlagSet;
BEGIN
  imsg := gt.GT_GetIMsg (MenuDemoWnd.userPort);
  WHILE imsg#NIL DO
```

```
    code  := imsg.code;
    class := imsg.class;
    gt.GT_ReplyIMsg (imsg);
    imsg := NIL;

    IF KEY class
      OF {I.closeWindow} THEN
        RAISE(LeaveMsgLoop);
      END
      OF {I.refreshWindow} THEN
        gt.GT_BeginRefresh (MenuDemoWnd);
        (* Refresh Window *)
        gt.GT_EndRefresh (MenuDemoWnd, TRUE);
      END;
      OF {I.menuHelp} THEN
        IF code # CARDINAL(I.menuNull) THEN
          item := I.ItemAddress (MenuDemoMenus, code);
          (* give help for the item *)
          (* don't use item.nextSelect! *)
        END;
      END;
      OF {I.menuPick} THEN
        WHILE code # CARDINAL(I.menuNull) DO
          item := I.ItemAddress (MenuDemoMenus, code);
          IF KEY LONGINT(item.userData)
            OF MID_New THEN
            END
            OF MID_Open THEN
            END
            OF MID_Save THEN
            END
            OF MID_SaveAs THEN
            END
            OF MID_Quit THEN
              RAISE(LeaveMsgLoop);
            END
            OF MID_Cut THEN
```



```
            END
            OF MID_Copy THEN
            END
            OF MID_Paste THEN
            END
            OF MID_Erase THEN
            END
            OF MID_Undo THEN
            END
            OF MID_StartLearning THEN
            END
            OF MID_StopLearning THEN
            END
            OF MID_AssignMacro THEN
            END
            OF MID_CreateIcons THEN
            END
            OF MID_Raw THEN
            END
            OF MID_Cooked THEN
            END
          END;
          code := item.nextSelect;
        END; (* WHILE *)
      END;
    END;
    imsg := gt.GT_GetIMsg (MenuDemoWnd.userPort);
  END;
END HandleIDCMP;

BEGIN
  TRY
    SetupScreen;
    OpenMenuDemoWindow;
    TRY
      LOOP
        FORGET e.WaitPort(MenuDemoWnd.userPort);
        HandleIDCMP;
```

```
      END;
    EXCEPT
      OF LeaveMsgLoop THEN END
    END;
  EXCEPT
    OF NoVisualInfo THEN
      Warn("Could not get a visual info");
    END
    OF NoScreen THEN
      Warn("Could not lock the public screen");
    END;
    OF NoWindow THEN
      Warn("Could not open the window");
    END
    OF NoMenus THEN
      Warn("Could not create the menues");
    END
    OF CouldNotLayout THEN
      Warn("Could not layout the menues");
    END
    OF CouldNotSet THEN
      Warn("Could not set the new menu strip");
    END;
  END;
CLOSE
  CloseMenuDemoWindow;
  CloseDownScreen;
END MenuDemo.
```


5.5.4 Gadgets

Die Erstellung von Gadgets ist mit der `gadtools.library` fast so einfach wie die Erstellung von Menüs. Für alle Arten von Gadgets gibt es wieder eine Struktur:

```
NewGadget = RECORD
              leftEdge, topEdge : INTEGER;
              width, height     : INTEGER;
              gadgetText        : SysStringPtr;
              textAttr          : TextAttrPtr;
              gadgetID          : CARDINAL;
              flags             : NewGadgetFlagSet;
              visualInfo        : VisualInfo;
              userData          : ANYPTR;
            END;
```

Im Gegensatz zu Menüs muß man jedoch jedes Gadget einzeln mit `CreateGadgetTags()` erzeugen. Dabei übergibt man in einer Tag-Liste weitere wichtige Parameter für das Gadget. Normalerweise initialisiert man nur eine NewGadget-Struktur und ändert vor dem Aufruf von `CreateGadgetTags()` einige Felder. Die Felder der Struktur im einzelnen:

`leftEdge, topEdge` Position des Gadgets.

`width, height` Breite und Höhe des Gadgets.

`gadgetText` Der Text, der die Funktion des Gadgets erklärt. Die Position des Textes kann mit den Flags festgelegt werden. Wichtig ist, daß der Zeiger solange gültig bleibt, wie die Gadgets existieren.

`textAttr` Der Font, den das Gadget verwenden soll. Hier muß immer ein gültiger Zeiger (notfalls auf Topaz 8) stehen. Besser ist es natürlich, wenn hier ein Zeiger auf den vom Benutzer gewünschten Font steht und sich das Programm an diesen Font anpaßt.

`gadgetID` Eine frei wählbare Zahl, die verwendet werden kann, um das Gadget zu identifizieren.

`flags` Hiermit kann bestimmt werden, wo der zum Gadget gehörige Text plaziert werden soll und ob er mit einer besonderen (highlight) Farbe gezeichnet werden soll. Folgende Flags sind möglich:

placeTextLeft Links neben das Gadget.

placeTextRight Rechts neben das Gadget.

placeTextAbove Über das Gadget.

placeTextBelow Unter das Gadget.

highLabel In einer speziellen Farbe.

`visualInfo` Ein Zeiger auf die mit `GetVisualInfo()` erhaltene Struktur.

`userData` Ebenso wie `gadgetID` frei wählbar. Hier kann man z. B. eine Hookfunktion einhängen, die man bei drücken eines Gadgets aufruft.

Bevor man mit `CreateGadgetTags()` ein Gadget erzeugen kann, muß man zuerst mit `CreateContext()` für eine Umgebung sorgen, die die Gadgets aufnehmen kann.

```
PROCEDURE CreateContext(VAR GadgetPtr IN A0 : GadgetPtr):GadgetPtr;
```

`CreateContext()` bekommt als VAR-Parameter die Variable übergeben, in der der Zeiger auf das erste Gadget abgelegt werden soll. Der Rückgabewert ist ein Gadget-Zeiger, der dann `CreateGadgetTags()` übergeben werden muß. In Fehlerfall kann `CreateContext()` auch NIL zurückliefern.

```
PROCEDURE CreateGadgetTags(    kind     IN D0 : GadgetKind;
                               previous IN A0 : GadgetPtr;
                           REF newgad   IN A1 : NewGadget;
                               taglist  IN A2 : LIST OF GadgetTags
                           ): GadgetPtr;
```


Der Parameter `kind` bestimmt den Typ des Gadgets. In `previous` übergibt man den Zeiger, den man von `CreateContext()`- bzw. dem letzten `CreateGadgetTags()`-Aufruf erhalten hat. Wenn man `NIL` übergibt, wird kein Gadget erzeugt. Dadurch erspart man sich die Prüfung auf `NIL` nach jeden Aufruf und muß erst, nachdem man alle Gadgets erzeugt hat, prüfen, ob dies auch gelungen ist. `newgad` ist die bereits besprochene NewGadget-Struktur. Die Werte, die in der `taglist` angegeben werden können, hängen von der Art des Gadgets ab. Sie beeinflussen das Aussehen und die Funktion des Gadgets. Nachträglich kann man mit `GT_SetGadgetAttrs()` die Werte dieser Tags ändern. Dies ist jedoch nur bei speziellen Tags möglich und wird bei jedem Tag einzeln erklärt.

Den Tag `underScore` unterstützt jedes Gadget. Er gibt das Zeichen an, mit dem ein Zeichen im Gadgettext markiert werden soll. Dies ist normalerweise der Unterstrich „_“, kann aber auch jedes andere Zeichen sein, vorausgesetzt es kommt nicht im Gadgettext vor. Innerhalb des Gadgettextes muß dieses Zeichen dann direkt vor dem zu markierenden Zeichen stehen. Ein Beispiel für einen solchen GadgetText Label wäre „`_Ok`“. Hier wird dann da „`O`“ unterstrichen. Das Programm sollte, falls die Taste „`O`“ gedrückt wird, so reagieren, als hätte der Benutzer das Gadget angeklickt.

Die mit `CreateGadgetTags()` erzeugte Gadget-Struktur darf man auf keinen Fall von Hand verändern. Es gibt jedoch Ausnahmen, die bei dem jeweiligen GadgetTyp erklärt werden.

Nachdem man alle Gadgets erzeugt hat und dabei kein Fehler aufgetreten ist, werden sie mit dem Tag `gadgets` bei `OpenWindowTags()` eingebunden, indem man den Zeiger auf das erste Gadget dort einträgt. Der IDCMP sollte mindestens die Vereinigungsmenge aller für die verwendeten Gadgets vorgesehenen IDCMP-Flags enthalten. Nach `OpenWindowTags()` muß man noch die GadTools-Funktion `RefreshWindow()` aufrufen. Nachdem das Fenster geschlossen wurde, muß man die Gadgets mit `FreeGadgets()` freigeben. All dies wird im Beispielprogramm deutlich.

Hier eine Übersicht über alle Gadget-Arten, die für `kind` angegeben werden können. Eine Erklärung, wie man sie verwendet und eine Beschreibung der dazu gehörigen Tags:

button Dies ist ein Gadget, das einen einfachen Knopf darstellt. Man bekommt eine IDCMP-Message, falls er gedrückt wird. Diese Tags gibt es:

`buDisabled : LONGBOOL`
Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift" dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

checkbox Dieses Gadget ist ein Kästchen, das in der jetztigen GadTools-Version die feste Größe 26x11 hat. Durch anklicken kann man das Kästchen abhaken oder den Haken wieder entfernen. Den Zustand des Gadgets kann man durch Lesen des `selected` Bits in der Gadget-Struktur herausfinden. Diese Tags gibt es:

`cbDisabled : LONGBOOL`
Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift" dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`checked : LONGBOOL`
Gibt an, ob das Gadget abgehakt ist oder nicht. Kann mit `GT_SetGadgetAttrs()` verändert werden.

integer &string Mit diesen Gadget-Arten kann man Texte oder Zahlen einlesen. Den eingegebenen Text findet man im Element `buffer` der StringInfo-Struktur, auf die das Element `specialInfo` der Gadget-Struktur zeigt. Falls es sich um ein Integer-Gadget handelt, findet man die eingegebene Zahl im `longInt`-Feld der StringInfo-Struktur. Diese Felder dürfen nur ausgelesen und keinesfalls verändert werden. Veränderungen kann man mit `GT_SetGadgetAttrs()` vornehmen. Falls ein String- bzw. Integer-Gadget nicht mit RETURN verlassen wird, bekommt man auch keine `gadgetUp`-Message. Man sollte deshalb den Buffer immer erst auslesen, wenn man ihn benötigt. Diesen Tag gibt es nur beim String Gadget:

`stString : SysStringPtr`
Mit diesen Tag übergibt man den Text, der im Gadget stehen

soll. Dieser Tag kann mit `GT_SetGadgetAttrs()` verwendet werden.

Diesen Tag gibt es nur beim Integer Gadget:

`inNumber : LONGCARD`
Mit diesem Tag übergibt man die Zahl, die im Gadget stehen soll. Dieser Tag kann mit `GT_SetGadgetAttrs()` verwendet werden.

Die folgenden Tags können mit beiden Gadget-Arten verwendet werden, die Tags für die Stringgadgets haben anstelle eines `in` ein `st` vor dem Namen:

`inMaxChars : LONGCARD`

`stMaxChars : LONGCARD`
Die maximale Anzahl der Zeichen die im Gadget eingegeben werden dürfen. Default ist bei String-Gadgets 64 und bei Integer-Gadgets 10. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`inDisabled : LONBOOL`

`stDisabled : LONBOOL`
Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift" dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`inExitHelp : LONGBOOL`

`stExitHelp : LONGBOOL`
Das Gadget kann mit der HELP-Taste verlassen werden. Die gesendete „gadgetUp"-IDCMP-Message enhält in diesem Fall `$5F` im `code`-Feld. Es sollte dann ein Hilftext zu dem Gadget angezeigt werden. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`inTabCycle : LONBOOL`

`stTabCycle : LONBOOL`
Mit diesem Tag kann man das automatische Springen von String-Gadget zu String-Gadget mit Hilfe der TAB-Taste unterbinden, z. B. wenn nur ein String-Gadget vorhanden ist. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`stJustification : ActivationFlagSet`
Gibt an, ob der String linksbündig `stringLeft` (default), rechtsbündig `stringRight` oder zentriert `stringCenter` im Gadget erscheinen soll. Dieser Tag kann auch bei Integer-Gadgets zu verwenden. Nur bei der Erzeugung zu verwenden.

`stReplaceMode : LONGBOOL`
Wird hier TRUE übergeben, erhält man ein Gadget im Replace-Modus, also Überschreibmodus; übergibt man FALSE, eines im Einfügemodus. Nur bei der Erzeugung zu verwenden. Auch dieser Tag ist für Integer-Gadgets zu verwenden.

listview Dieses Gadget ist eine Scroll-Liste mit der mehrere Einträge angezeigt werden können. Bei Bedarf kann der ausgewählte Eintrag auch angezeigt werden. Die IDCMP-Message `gadgetUp` enthält im `code`-Feld die Nummer des angewählten Eintrags. Diese Tags gibt es:

`lvTop : LONGCARD`
Dieser Tag gibt die Nummer des Eintrags an, der als erster Eintrag im ListView sichtbar sein soll. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`lvLabels : ListPtr`
Hiermit gibt man die Liste an, deren Einträge angezeigt werden sollen. Will man die Liste nachträglich verändern, muß man sie vorher mit `SetGagetAttrs()` und `NIL` für den `lvLabels`-Tag vom Gadget entfernen. Der Inhalt des ListViews verschwindet dadurch jedoch auch. Für kurzfristige Änderungen kann man deshalb auch -1 als Parameter für den `labels`-Tag angeben, dann bleibt der Inhalt erhalten. Man muß dann jedoch die Liste wieder schnell mit diesem Tag einbinden.

`lvReadOnly : LONGBOOL`
Der Inhalt des Gadgets kann nur gelesen, jedoch nicht verändert werden. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`lvScrollWidth : LONGCARD`
Gibt die Breite des Scroll-Gadgets an der Seite der listviews an. Die (vernünftige) Voreinstelung beträgt 16. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`lvShowSelected : GadgetPtr`
Durch diesen Tag kann bestimmt werden, daß der angeklickte Eintrag in einem Feld (unterhalb) des Listviews angezeigt werden soll. Hier kann man den Zeiger auf ein vorher erstelltes String-Gadget angeben, dann wird der Text in dieses kopiert. Wird `NIL` angegeben, so wird der Text nur angezeigt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`lvSelected : LONGCARD`
Gibt die Nummer des angewählten Eintrags an, der bei der Erzeugung angezeigt werden soll. Hat nur eine Wirkung, wenn auch `lvShowSelected` verwendet wurde. Der Wert -1 bedeutet, daß kein Eintrag angewählt ist. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`lvSpacing : LONGCARD`
Gibt die Anzahl der Pixel an, die zwischen den einzelnen Einträgen sein sollen. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

mx Dieser Gadget-Typ (auch *radio button* genannt) wird verwendet, falls sich einem mehrere sich ausschließende Wahlmöglichkeiten bieten, ein Cycle-Gadget jedoch unangebracht ist. Bei sehr zahlreichen Wahlmöglichkeiten sollte ein ListView verwendet werden.

Die Flags in der NewGadget-Struktur geben an, ob der Text der Wahlmöglichkeiten links oder rechts von den Buttons stehen soll. Bitte nur `placeTextLeft` und `placeTextRight` verwenden. Der Text in der NewGadget-Struktur wird ingoriert. Die IDCMP-Message `gadgetUp` enthält im `code`-Feld die Nummer des angewählten Eintrags. Folgende Tags stehn zur Verfügung:

`mxLabels : ` **`POINTER TO ARRAY OF`** `SysStringPtr`
: In diesem Array werden die Texte der verschiedenen Wahlmöglichkeiten angegeben. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Als letzter Eintrag muß `NIL` stehen.

`mxActive : LONGCARD`
: Die Nummer der ausgewählten Möglichkeit. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`mxSpacing : LONGCARD`
: Gibt die Anzahl der Pixel an, die zwischen den einzelnen Einträgen sein sollen. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

text Dieses Gadget zeigt einen Text an. Der Benutzer kann den Text nicht ändern. Es werden keine IDCMP-Messages verschickt. Diese Tags gibt es:

`txText : SysStringPtr`
: Der Text, der durch das Gadget dargestellt werden soll. Er kann mit `GT_SetGadgetAttrs()` verändert werden.

`txCopyText : LONGBOOL`
: Der darzustellende Text wird kopiert. Dadurch darf der Zeiger auf ihn ungültig werden. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`txBorder : LONGBOOL`
: Um den darzustellenden Text wird ein Rahmen gezeichnet. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

number Dieses Gadget zeigt eine Zahl an. Der Benutzer kann die Zahl nicht ändern. Es werden keine IDCMP-Messages verschickt. Diese Tags gibt es:

`nmNumber : LONGINT`
: Die Zahl, die durch das Gadget dargestellt werden soll. Sie kann mit `GT_SetGadgetAttrs()` verändert werden.

`nmBorder : LONGBOOL`
: Um die darzustellende Zahl wird ein Rahmen gezeichnet. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

cycle Dieser Gadget-Typ wird verwendet, falls sich einem mehrere sich ausschließende Wahlmöglichkeiten bieten, ein Mx-Gadget jedoch unangebracht ist. Bei sehr zahlreichen Wahlmöglichkeiten sollte ein ListView verwendet werden.

Die Flags in der NewGadgetstruktur geben an, ob der Text der Wahlmöglichkeiten links oder rechts von Boxen stehen soll[13]. Der Text in der NewGadget-Struktur wird ingoriert. Die IDCMP-Message `gadgetUp` enthält im `code`-Feld die Nummer des angewählten Eintrags. Diese Tags gibt es:

`cyDisabled : LONGBOOL`
: Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift“ dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`cyLabels : ` **`POINTER TO ARRAY OF`** `SysStringPtr`
: In diesem Array werden die Texte der verschiedenen Wahlmöglichkeiten angegeben. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Achtung: Als lezter Eintrag muß `NIL` stehen.

`cyActive : LONGCARD`
: Die Nummer der ausgewählten Möglichkeit. Kann mit `GT_SetGadgetAttrs()` verändert werden.

[13] Siehe Mx-Gadgets.

palette Mit diesem Gadget läßt sich ein Farb-Auswahlfeld realisieren. Die IDCMP-Message `gadgetUp` enthält im `code`-Feld die Nummer der angewählten Farbe. Diese Tags gibt es:

`paDisabled : LONGBOOL`
Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift" dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`paDepth : LONGCARD`
Die Anzahl der Bitplanes, die die Palette repräsentieren. Es gibt 2^{depth} Farben. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`paColor : LONGCARD`
Die Nummer der zu beginn angewählten Farbe. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`paColorOffset : LONGCARD`
Die erste Farbe, die in der Palette angezeigt werden soll. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden.

`paIndicatorW : LONGCARD`
Durch diesen oder den nächsten Tag kann man ein Feld erzeugen, in dem die gerade angewählte Farbe angezeigt wird. Es wird die Breite dieses Feldes angegeben. Das Feld selbst befindet sich auf der linken Seite des Palette-Gadgets und ist so hoch wie die ganze Palette.

`paIndicatorH : LONGCARD`
Durch diesen oder den vorigen Tag kann man ein Feld erzeugen, in dem die gerade angewählte Farbe angezeigt wird. Es wird die Höhe dieses Feldes angegeben. Das Feld selbst befindet sich oberhalb des Palette-Gadgets.

slider Das Slider-Gadget ist eines der beiden proportionalen Gadgets, die GadTools anbietet. Mit einem Slider kann man einen Wert

zwischen zwei Grenzen (`min` und `max`) einstellen, vergleichbar einem Schieberegler an einer Stereoanlage. Klickt der Benutzer neben dem Reglerknopf in das Gadget, wandert der Knopf in Einzelschritten auf den Mauszeiger zu. Man bekommt `mouseMove`-IDCMP-Messages geschickt, die im `code`-Feld den aktuelle Wert enthalten. Falls man es wünscht, kann man sich mit den entsprechenden Tags auch `gadgetUp` und `gadgetDown` Messages schicken lassen. Als besonderen Service kann man sich den aktuellen Wert auch anzeigen lassen. Diese Tags gibt es:

`slMin : LONGINT`
Der minimale Wert, den das Gadget annehmen kann. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`slMax : LONGINT`
Der maximale Wert. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`slLevel : LONGINT`
Der aktuelle Wert. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`slMaxLevelLen : LONGCARD`
Die maximale Länge, die der String erreichen darf, der angezeigt wird. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Wenn dieser Tag verwendet wird, muß auch der folgende Tag `slLevelFormat` verwendet werden.

`slLevelFormat : SysStringPtr`
Das Format, in dem der Wert dargestellt werden soll, falls er ausgegeben werden soll. Es dürfen die Format-Strings wie bei Exec's `RawDoFmt()` oder der C-Funktion `printf()` verwendet werden. Beispiel: `"%2ld"` um eine maximal 2 Zeichen lange Nummer anzuzeigen, `"%02lx"` um eine 2 Zeichen lange Hexadezimalnummer anzuzeigen, die im Bedarfsfall führende Nullen enthält, oder auch `"%2ld Stunden"`. Damit die Darstellung sauber erfolgen kann, sollte man einen nicht-proportionalen Font für das Gadget verwenden. Dieser

Tag darf nur beim Erstellen des Gadgets verwendet werden. Wird dieser Tag verwendet, muß auch `slLevelMaxLen` gesetzt werden.

`slLevelPlace : NewGadgetFlagSet`
: Gibt den Ort an, an dem die Textausgabe erfolgt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Defaultmäßig steht der Text links vom Gadget.

`slDispFunc : DispFunc`
: Hier kann man eine Prozedur angeben, die den aktuellen Wert vor der Anzeige umrechnet. So kann z. B. ein Slider, mit dem man die Tiefe eines Screens (1-3) einstellt, die Anzahl der Farben (1, 2, 4, 8) anzeigen. Die Funktion muß vom Typ `DispFunc` sein:

```
DispFunc = PROCEDURE( gad IN A0: GadgetPtr;
                      org IN D0: INTEGER ): LONGINT;
```

Dabei bekommt sie in `gad` den Zeiger auf das Slidergadget und in `org` den Orginalwert des Sliders geliefert. Der Rückgabewert wird dann angezeigt. Die Umwandlung bezieht sich ausschließlich auf die Wertanzeige, nicht auf den echten Wert des Gadgets.

`slFreedom : Orientation`
: Hiermit kann man angeben, ob der Slider horizontal oder vertikal sein soll, indem man hier `horiz` bzw. `vert` einträgt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist `horiz`.

`slImmediate : LONGBOOL`
: Es werden `gadgetDown`-Messages verschickt, sobald der Benutzer den Knopf anklickt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist FALSE.

`slRelVerify : LONGBOOL`
: Es werden `gadgetUp`-Messages verschickt, sobald der Benutzer den Knopf losläßt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist FALSE.

`slDisabled : LONGBOOL`
: Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift“ dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

scroller Das Scroller-Gadget ist das zweite proportionale Gadgets, das GadTools anbietet. Ein Scroller repräsentiert einen Auschnitt aus einem größeren Bereich. Die Workbench benutzt z. B. Scroller-Gadgets in den Fensterrahmen, auch die Fenster des *Hitex-Editor* haben einen Scroller im rechten Fensterrahmen[14]. Dabei zeigt der Balken die Größe (relativ zum Gesamttext) und die Position des momentan sichtbaren Textbereich an.

Mit den drei Tags `scTotal`, `scVisible`und `scTop`, gibt man sowohl die Größe des gesamten Bereichs, die des sichtbaren Ausschnitts, sowie die Position des Ausschnitts an. Man bekommt `mouseMove`-IDCMP-Messages geschickt, die im `code`-Feld den aktuellen `scTop`-Wert enthalten. Falls gewünscht, kann man sich mit den entsprechenden Tags auch `gadgetUp` und `gadgetDown` Messages schicken lassen. Klickt man in einem Scroller neben dem Balken in das Gadget, blättert der Scroller seitenweise, d. h. `scTop` verschiebt sich um `scVisible-1`.

Diese Tags gibt es:

`scTop : LONGINT`
: Die oberste Zeile oder Position, die angezeigt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`scTotal : LONGINT`
: Die Anzahl der Zeilen oder Positionen, die es gibt. Kann mit `GT_SetGadgetAttrs()` verändert werden.

`scVisible : LONGINT`
: Die Anzahl der sichtbaren Zeilen oder Positionen, die es gibt. Kann mit `GT_SetGadgetAttrs()` verändert werden.

[14] Auch wenn diese nicht mit GadTools programmiert wurden, da man keine GadTools-Gadgets innerhalb von Fensterrahmen verwenden kann

`scArrows : LONGCARD`
: Bei Verwendung dieses Tags werden Pfeil-Gadgets zum Scroller hinzugefügt. Der Tagwert bestimmt die Breite bzw. Höhe dieser Pfeile. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Defaultmäßig ist dieser Tag nicht gesetzt.

`scFreedom : Orientation`
: Hiermit kann man angeben, ob der Scroller horizontal oder vertikal sein soll, indem man hier `horiz` bzw. `vert` einträgt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist `horiz`.

`scImmediate : LONGBOOL`
: Es werden `gadgetDown`-Messages verschickt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist FALSE.

`scRelVerify : LONGBOOL`
: Es werden `gadgetUp`-Messages verschickt. Dieser Tag darf nur beim Erstellen des Gadgets verwendet werden. Default ist FALSE.

`scDisabled : LONGBOOL`
: Gibt an, ob das Gadget anwählbar ist oder in „Geisterschrift“ dargestellt wird. Kann mit `GT_SetGadgetAttrs()` verändert werden.

generic Dieser Gadgettyp wird verwendet, falls man ein Gadget benötigt, das GadTools nicht zur Verfügung stellt. Beim `CreateGadgetTags()`-Aufruf darf man keine Flags für die Position des Gadget-Textes angeben. Eine IntuiText Struktur für den Text wird jedoch erzeugt. Man muß dann die Felder `flags`, `activation`, `gadgetRender`, `selectRender`, `mutualExclude` und `specialInfo` selbst setzen, um dem Gadget eine Funktion zu geben. Eine Besonderheit gibt es beim `gadgetType`. Dieses Feld darf nicht direkt überschrieben werden, sondern man muß den gewünschten mit dem bereits von GadTools gesetzten Type mit OR verknüpfen. Sie werden wahrscheinlich nie in die Verlegenheit kommen, diesen Typ zu verwenden, daher wird hier nicht näher darauf eingegangen.

Zu beachten ist bei GadTools-Gadgets, daß man unbedingt die Funktionen `GetIMsg()` und `ReplyIMsg()` der `gadtool.library` verwendet, um die IntuiMessages zu bearbeiten. Außerdem dürfen die Intuition-Funktionen `RefreshWindow()`, `RefreshGList()`, `OnGadget()`, `OffGadget()`, etc. nicht mit GadTools-Gadgets bzw. Windows, die diese Gadgets enthalten, verwendet werden. Man muß stattdessen die entsprechenden Funktionen aus der `gadtools.library` verwenden.

Das Beispielprogramm führt einige Gadgettypen vor und zeigt gleichzeitig, wie man sich an beliebige Fonts anpaßt.

```
MODULE GadgetDemo;

FROM System IMPORT SysStringPtr;
            IMPORT Exec      AS e,
                   Intuition AS I,
                   GadTools  AS gt,
                   Graphics  AS g;
EXCEPTION
  LeaveMsgLoop     : "Leave message loop";
  NoVisualInfo     : "No visual info";
  NoScreen         : "Could not lock screen";
  NoGadgetContext  : "No gadget context";
  NoGadgets        : "Could not allocate all gadgets";
  NoWindow         : "Could not open window";

CONST
  (* Window Size *)
  GadgetDemoLeft   = 60;
  GadgetDemoTop    = 30;
  GadgetDemoWidth  = 230;
  GadgetDemoHeight = 130;

  (* Gadget IDs *)
  GDListView       = 0;
  GDMX             = 1;
  GDCheckbox       = 2;
  GDCycle          = 3;
  GDSlider         = 4;
  GDString         = 5;

VAR
  Scr              : I.ScreenPtr;
  VisualInfo       : ANYPTR;
  GadgetDemoWnd    : I.WindowPtr;
  GadgetDemoGList  : I.GadgetPtr;
  ListViewList     : e.List;
  ListViewNodes    : ARRAY [7] OF e.Node;
  GadgetDemoZoom   : I.IBox;
```



```
  Font                : g.TextAttrPtr;
  Attr                : g.TextAttr;
  FontX, FontY        : INTEGER;
  OffX, OffY          : INTEGER;

CONST
  MX0Labels    = ARRAY OF SysStringPtr:("It's","mutually",
                                        "exclusive", NIL);
  Cycle0Labels = ARRAY OF SysStringPtr:("Mama","mia","let",
                                        "me ","go", NIL);

PROCEDURE Warn (REF txt: STRING);
VAR es := I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Gadget Demo",
                        gadgetFormat = "Abort");
BEGIN
  es.textFormat := txt.data'PTR;
  FORGET I.EasyRequest (GadgetDemoWnd, es'PTR, NIL);
END Warn;

PROCEDURE ComputeX (value: INTEGER): INTEGER;
BEGIN
  RETURN ((FontX * value) + 4 ) DIV 8;
END ComputeX;

PROCEDURE ComputeY (value: INTEGER): INTEGER;
BEGIN
  RETURN ((FontY * value) + 4 ) DIV 8;
END ComputeY;

PROCEDURE ComputeFont (width, height: INTEGER);
BEGIN
  Font       := Attr'PTR;
  Font.name  := Scr.rastPort.font.name;
  FontY      := Scr.rastPort.font.ySize;
  Font.ySize := FontY;
  FontX      := Scr.rastPort.font.xSize;
```

```
  OffX := Scr.wBorLeft;
  OffY := Scr.rastPort.txHeight + CARDINAL(Scr.wBorTop) + 1;

  | Wenn der aktuelle Font zu groß ist verwende Topaz 8
  IF (width # 0) AND (height # 0) AND
     ((ComputeX (width)  + OffX + Scr.wBorRight  > Scr.width) OR
      (ComputeY (height) + OffY + Scr.wBorBottom > Scr.height)) THEN
    Font.name  := "topaz.font";
    Font.ySize := 8;
    FontY      := Font.ySize;
    FontX      := Font.ySize;
  END;
END ComputeFont;

PROCEDURE SetupScreen;
BEGIN
  Scr := I.LockPubScreen (NIL);
  ASSERT(Scr#NIL,NoScreen);

  ComputeFont (0, 0);

  VisualInfo := gt.GetVisualInfo (Scr, DONE);
  ASSERT(VisualInfo#NIL,NoVisualInfo);
END SetupScreen;

PROCEDURE CloseDownScreen;
BEGIN
  IF VisualInfo # NIL THEN
    gt.FreeVisualInfo (VisualInfo);
    VisualInfo := NIL;
  END;
  IF Scr # NIL THEN
    I.UnlockPubScreen (NIL, Scr);
    Scr := NIL;
  END;
END CloseDownScreen;
```



```
PROCEDURE OpenGadgetDemoWindow;
VAR
  ng      : gt.NewGadget;
  gad     : I.GadgetPtr;
  lc, tc,
  lvc,
  offx,
  offy    : INTEGER;
  wleft,
  wtop,
  ww, wh  : INTEGER;

  PROCEDURE InitNewGadget (    left,
                               top,
                               width,
                               height   : INTEGER;
                           REF text     : STRING;
                               ID       : INTEGER;
                               flags    : gt.NewGadgetFlagSet);
  BEGIN
    ng.leftEdge   := OffX + ComputeX (left);
    ng.topEdge    := OffY + ComputeY (top);
    ng.width      := ComputeX (width);
    ng.height     := ComputeY (height);
    IF text.len>0 THEN
      ng.gadgetText := text.data'PTR;
    ELSE
      ng.gadgetText := NIL;
    END;
    ng.textAttr   := Font;
    ng.gadgetID   := ID;
    ng.flags      := flags;
    ng.visualInfo := VisualInfo;
    ng.userData   := NIL;
  END InitNewGadget;
```

```
BEGIN
  wleft := GadgetDemoLeft;
  wtop  := GadgetDemoTop;

  ComputeFont (GadgetDemoWidth, GadgetDemoHeight);

  ww := ComputeX (GadgetDemoWidth);
  wh := ComputeY (GadgetDemoHeight);

  IF wleft + ww + OffX + Scr.wBorRight > Scr.width THEN
    wleft := Scr.width - ww;
  END;
  IF wtop + wh + OffY + Scr.wBorBottom > Scr.height THEN
    wtop := Scr.height - wh;
  END;

  ListViewNodes[0].succ := ListViewNodes[1]'PTR;
  ListViewNodes[0].pred := ANYPTR(ListViewList.head'PTR);
  ListViewNodes[0].pri  := 0;
  ListViewNodes[0].name := "It's";

  ListViewNodes[1].succ := ListViewNodes[2]'PTR;
  ListViewNodes[1].pred := ListViewNodes[0]'PTR;
  ListViewNodes[1].pri  := 0;
  ListViewNodes[1].name := "a"*;

  ListViewNodes[2].succ := ListViewNodes[3]'PTR;
  ListViewNodes[2].pred := ListViewNodes[1]'PTR;
  ListViewNodes[2].pri  := 0;
  ListViewNodes[2].name := "Kind";

  ListViewNodes[3].succ := ListViewNodes[4]'PTR;
  ListViewNodes[3].pred := ListViewNodes[2]'PTR;
  ListViewNodes[3].pri  := 0;
  ListViewNodes[3].name := "of";

  ListViewNodes[4].succ := ListViewNodes[5]'PTR;
  ListViewNodes[4].pred := ListViewNodes[3]'PTR;
```



```
  ListViewNodes[4].pri  := 0;
  ListViewNodes[4].name := "Magic";

  ListViewNodes[5].succ := ListViewNodes[6]'PTR;
  ListViewNodes[5].pred := ListViewNodes[4]'PTR;
  ListViewNodes[5].pri  := 0;
  ListViewNodes[5].name := "it's";

  ListViewNodes[6].succ := ANYPTR(ListViewList.tail'PTR);
  ListViewNodes[6].pred := ListViewNodes[5]'PTR;
  ListViewNodes[6].pri  := 0;
  ListViewNodes[6].name := "Cluster";

  ListViewList.head     := ListViewNodes[0]'PTR;
  ListViewList.tail     := NIL;
  ListViewList.tailPred := ListViewNodes[6]'PTR;

  gad := gt.CreateContext (GadgetDemoGList);
  ASSERT(gad#NIL,NoGadgetContext);

  InitNewGadget (10, 15, 85, 55,
                 "ListView", GDListView, {gt.placeTextAbove});
  gad := gt.CreateGadgetTags (gt.listview,
                              gad, ng,
                              lvLabels       : ListViewList'PTR,
                              lvShowSelected : NIL,
                              DONE);
  InitNewGadget (120, 15, 17, 9,
                 "", GDMX, {gt.placeTextRight});
  gad := gt.CreateGadgetTags (gt.mx,
                              gad, ng,
                              mxLabels : MXOLabels'PTR,
                              DONE);
  InitNewGadget (10, 80, 26, 11,
                 "Check",GDCheckbox, {gt.placeTextRight});
  gad := gt.CreateGadgetTags (gt.checkbox,
                              gad,ng,
                              DONE);
```

```
InitNewGadget (120, 77, 70, 15,
               "", GDCycle, {});
gad := gt.CreateGadgetTags (gt.cycle,
                            gad, ng,
                            cyLabels : Cycle0Labels'PTR,
                            DONE);
InitNewGadget (10, 100, 70, 15,
               "", GDSlider, {});
gad := gt.CreateGadgetTags (gt.slider,
                            gad, ng,
                            slMaxLevelLen : 6,
                            slLevelFormat : "%2.2ld  ",
                            slLevelPlace  : {gt.placeTextRight},
                            slFreedom     : gt.horiz ,
                            slRelVerify   : TRUE,
                            DONE);
InitNewGadget (120, 100, 70, 15,
               "", GDString, {});
gad := gt.CreateGadgetTags (gt.string,
                            gad, ng,
                            stString   : "We are the champions",
                            stMaxChars : 256,
                            DONE);

ASSERT(gad#NIL,NoGadgets);

| Größe des Fensters nach drücken des ZoomGadgets
GadgetDemoZoom.left   := wleft;
GadgetDemoZoom.top    := wtop;
GadgetDemoZoom.width  := g.TextLength (Scr.rastPort'PTR,
                                       "Gadget Demo".data[0]'PTR,
                                        11) + 80;
GadgetDemoZoom.height:= Scr.wBorTop +
                        INTEGER(Scr.rastPort.txHeight) + 1;

GadgetDemoWnd := I.OpenWindowTags ( NIL,
                 left         : wleft,
                 top          : wtop,
```



```
                 width        : ww + OffX + Scr.wBorRight,
                 height       : wh + OffY + Scr.wBorBottom,
                 IDCMP        : gt.listViewIDCMP+gt.mxIDCMP+
                                gt.checkBoxIDCMP+gt.cycleIDCMP+
                                gt.sliderIDCMP+gt.stringIDCMP+
                                {I.closeWindow,I.refreshWindow},
                 flags        : {I.windowDrag,I.windowDepth,
                                 I.windowClose,I.activate,
                                 I.rmbTrap},
                 gadgets      : GadgetDemoGList,
                 title        : "Gadget Demo",
                 screenTitle  : "Gadgets in Cluster",
                 pubScreen    : Scr,
                 zoom         : GadgetDemoZoom'PTR,
                 DONE);

  ASSERT(GadgetDemoWnd#NIL,NoWindow);

  gt.GT_RefreshWindow (GadgetDemoWnd, NIL);
END OpenGadgetDemoWindow;

PROCEDURE CloseGadgetDemoWindow;
BEGIN
  IF GadgetDemoWnd # NIL THEN
    I.CloseWindow (GadgetDemoWnd);
    GadgetDemoWnd := NIL;
  END;
  IF GadgetDemoGList # NIL THEN
    gt.FreeGadgets (GadgetDemoGList);
    GadgetDemoGList := NIL;
  END;
END CloseGadgetDemoWindow;

PROCEDURE HandleIDCMP;
VAR
  imsg      : I.IntuiMessagePtr;
  item      : I.MenuItemPtr;
  class     : I.IDCMPFlagSet;
```

```
  iAddress  : ADDRESS;
  code      : INTEGER;
BEGIN
  imsg := gt.GT_GetIMsg (GadgetDemoWnd.userPort);
  WHILE imsg#NIL DO
    class    := imsg.class;
    iAddress := imsg.iAddress; | Immer erst class überprüfen,
                               | bevor iAddress dereferenziert wird
    gt.GT_ReplyIMsg (imsg);

    IF KEY class
      OF {I.closeWindow} THEN
        RAISE(LeaveMsgLoop);
      END
      OF {I.gadgetUp} THEN
        IF KEY I.GadgetPtr(iAddress).gadgetID
          OF GDListView THEN
            (* code enthält den ausgewählten Eintrag *)
          END
          OF GDMX THEN
            (* code enthält den ausgewählten Eintrag *)
          END
          OF GDCheckbox THEN
            IF (I.GadgetFlags.selected IN I.GadgetPtr(iAddress).flags)THEN
              (* angewählt *)
            ELSE
              (* nicht angewählt *)
            END;
            (* besser erst auslesen, wenn es gebraucht wird. *)
          END
          OF GDCycle THEN
            (* code enthält den ausgewählten Eintrag *)
          END
          OF GDSlider THEN
            (* code enthält den eingestellten Wert.
               Achtung: wenn der Slider auch negative Werte annehmen kann,
               muß man code in einen INTEGER casten *)
          END
```



```
          OF GDString THEN
            (* I.StringInfoPtr(I.GadgetPtr(iAddress).specialInfo).buffer
               enthält den eingegebenen Text. Erst auslesen, wenn
               es gebraucht wird.*)
          END
        ELSE
        END;
      END;
      OF {I.mouseMove} THEN
        (* iAddress in mouseMove ist nur dann gültig. wenn man
           GadTools' GT_GetIMsg() verwendet, also vorsicht! *)
        IF KEY I.GadgetPtr(iAddress).gadgetID
          OF GDSlider THEN
            (* code enthält den eingestellten Wert *)
          END
        ELSE
        END;
      END;
      OF {I.refreshWindow} THEN
        gt.GT_BeginRefresh (GadgetDemoWnd);
        (* refresh *)
        gt.GT_EndRefresh (GadgetDemoWnd, TRUE);
      END;
    END;

    imsg := gt.GT_GetIMsg (GadgetDemoWnd.userPort);
  END;
END HandleIDCMP;

BEGIN
  TRY
    SetupScreen;
    OpenGadgetDemoWindow;
    TRY
      LOOP
        FORGET e.WaitPort(GadgetDemoWnd.userPort);
        HandleIDCMP;
      END;
```

```
    EXCEPT
      OF LeaveMsgLoop THEN END
    END;
  EXCEPT
    OF NoVisualInfo THEN
      Warn("Could not get a visual info");
    END
    OF NoScreen THEN
      Warn("Could not lock the public screen");
    END;
    OF NoGadgetContext THEN
      Warn("Could not allocate the gadget context");
    END
    OF NoGadgets THEN
      Warn("Could not allocate all gadgets");
    END
    OF NoWindow THEN
      Warn("Could not open the window");
    END
  END;
CLOSE
  CloseGadgetDemoWindow;
  CloseDownScreen;
END GadgetDemo.
```

Mit dem hier vermittelten Wissen sollten Sie in der Lage sein, optisch ansprechende Benutzeroberflächen zu entwerfen. Weiterführende Informationen über die Programmierung findet man im RKM Libraries. Wie man seine Programme gestalten sollte, damit sie leicht bedienbar sind, steht im User Interface Style Guide.

5.6 Graphics, der Künstler

Dieses Abschnitt besfaßt sich nun mit der Library, die für die Graphikausgabe auf dem Bildschirm zuständig ist. Im Grunde sind alle Bildschirmausgaben beim Amiga Graphikausgaben, er besitzt nicht wie andere Rechner einen eigenen Textmodus.

5.6.1 Grundlegendes

Alle grafischen Ausgaben werden auf sog. Bitplanes gemacht. Diese bestehen aus Speicherbereichen, mit sovielen Bits wie Punkten auf dem Bildschirm. Je nachdem, ob ein Bit in so einer Plane gesetzt ist oder nicht, ist auf dem Bildschirm ein Punkt gesetzt oder nicht. Mit einer Bitplane, die nur Informationen über gesetzte und nicht gesetzte Punkte kennt, kann man logischerweise auch nur zwei Farben darstellen. Eine für gesetzte und eine für nicht gesetzte Bits.

Um mehr Farben zu erhalten, muß man mehrere Planes verwenden. Die Kombinationen der auf den einzelnen Planes gesetzten Bits ergibt dann die endgültige Farbe. Mit n Bitplanes kann man folglich 2^n Farben darstellen.

Im Normalfall muß sich der Programmierer nicht mit „Bitpopeln“ beschäftigen, das Betriebsystem unterstützt ihn dabei. Hierfür stellt es zwei Strukturen zur Verfügung: den Rastport. In ihn wird von vielen Funktionen gezeichnet. Und den ViewPort. In ihm stehen die Informationen über die Farben oder die Auflösung.

Screens und Windows haben beide den ViewPort gemeinsam, weshalb man in Fenstern nie andere Farben als auf dem Screen verwenden kann.

Sowohl Screen, als auch Fenster haben einen eigenen Rastport. In der Windowstruktur steht ein Zeiger auf dessen Rastport, ihn erhält man folgendermaßen:

```
VAR
  rastPort : RastPortPtr;
  window   : WindowPtr;
```

```
...
BEGIN
  | Öffnen des Fensters
  ...
  rastPort := window.rPort;
```

Um den an den Rastportpointer einer Screen zu kommen, muß man, da in einer Screen nicht nur ein Zeiger auf eine Rastport, sondern eine komplette Rastportstruktur enthalten ist, schreiben:

```
rastPort:= screen.rastPort'PTR;
```

Um den ViewPortPointer einer Screen zu ermitteln muß man im Prinzip genauso vorgehen:

```
view := screen.viewPort'PTR;
```

Um von einem Fenster aus dessen ViewPort zu ermitteln, muß man eine Stufe weiter gehen, da der ViewPort nicht direkt im Window verzeichnet ist:

```
view := window.wScreen.viewPort'PTR;
```

Bei diesen Beispielen muß `screen` vom Typ `ScrenPtr`, `window` vom Typ `WindowPtr`, `view` vom Typ `ViewPortPtr` und `rastPort` vom Typ `RastPortPrt` sein. Um den Inhalt dieser beiden Strukturen müssen Sie sich vorerst nicht kümmern, wir brauchen Sie nur, um sie an Prozeduren zu übergeben.

5.6.2 Farben, einfach einzustellen, auch für Farbenblinde

Sicher werden Sie sich über den Titel dieses Kapitels etwas wundern, aber es stimmt. Die Farbzusammensetzung ist sehr logisch, so daß man schon aus den Zahlenwerten recht gut abschätzen kann, wie die Farbe aussehen wird. Einer unserer Entwickler ist selbst farbenblind und stellt trotzdem

meist gute Farbwerte für seine Programme ein[15], er nennt es „Malen nach Zahlen“. Sie werden mir gleich zustimmen, denn alle Farben setzen sich aus drei Farbwerten zusammen: Aus Rot, Grün und Blau. Eine Farbe wird nun mit einer dreistelligen Hexzahl dargestellt:

```
Farbe : $RGB
```

Wobei `R` für den Rot-Anteil (0 – 15), `G` für den Grün-Anteil (0 – 15) und `B` für den Blau-Anteil (0 – 15) der Farbe steht. Ein Paar Beispiele:

```
$FFF = Weiß
$000 = Schwarz
$F00 = Rot
$0F0 = Grün
$00F = Blau
$FF0 = Gelb
```

Bei der Farbmischung ist zu beachten, das es sich nicht um eine additive Farbmischung wie bei Wasserfarben, sondern um die physikalische substraktive Farbmischung handelt. Wie schon gesagt, hat jeder Screen $2^{Screentiefe}$ Farbregister. Um nun die Farbe für ein solches Register[16] einzustellen, kann man den Befehl `SetRGB4():` benutzen.

```
PROCEDURE SetRGB4( vp IN A0 : ViewPortPtr;
                   n  IN D0 : CARDINAL;
                   r  IN D1 : CARDINAL;
                   g  IN D2 : CARDINAL;
                   b  IN D3 : CARDINAL );
```

Dabei gibt `vp` den Viewport, dessen Farbtabelle man verändern will, `n` die Nummer des Farbregisters und `r, g, b` den Rot-, Grün- und Blauanteil der neuen Farbe an.

Manchmal will man die Farbeinstellung eines Farbregisters auch auslesen, etwa um diese abzuspeichern. Hierzu kann `GetRGB4()` verwendet

[15] Ausnahmen bestätigen die Regel

[16] Beim Zeichnen gibt man nur noch die Registernummer an.

werden. Dieser Funktion muß man jedoch keinen ViewPort sondern einen Zeiger auf die Farbtabelle übergeben. Diesen finden Sie im ViewPort ihres Screens:

```
 colorMapPtr :=screen.viewPort.colorMap;
```

```
PROCEDURE GetRGB4( colorMap IN A0 : ColorMapPtr;
                   entry    IN D0 : LONGINT ): LONGINT;
```

Für `entry` übergibt man die Nummer des Farbregisters, das man auslesen möchte. Als Ergebnis erhält man einen codierten Farbwert: `Farbe = $00000RGB`. Um die einzelnen Farbwerte zu erhalten, muß man diese dekodieren:

```
Rot   = Farbe SHR 16 MOD 16
Grün  = Farbe SHR  4 MOD 16
Blau  = Farbe MOD 16
```

Wurde als Farbregisternummer ein ungültiger Wert gewählt, erhält man -1 zurück.

Will man alle Farbregister auf einmal verändern, kann man den Befehl `LoadRGB4()` verwenden:

```
PROCEDURE LoadRGB4( vp     IN A0 : ViewPortPtr;
                    colors IN A1 : ANYPTR;
                    count  IN D0 : INTEGER );
```

`pv` ist der ViewPort, der verändert werden soll. Bei `colors` handelt es sich um einen Zeiger auf eine Tabelle mit Farbeinträgen. Diese legt man am besten als Konstante **`ARRAY OF`** `CARDINAL` an, wobei jeder Eintrag der Form $0RGB entsprechen muß. `entries` schließlich gibt die Anzahl der Farbeinträge in der Tabelle an, die gesetzt werden sollen.

Hierzu ein kleines Beispiel, ich habe unser Beispielprogramm zu Screens aus Intuition etwas verändert. Dabei wird auch zuerst eine Screen geöffnet und dann nach kurzer Zeit mit `LoadRGB4()` die Farbregister geändert:


```
MODULE ScreenDemo;
FROM Strings IMPORT Str;
IMPORT Intuition AS I,
       Dos       AS d,
       Graphics  AS g,
       Utility   AS u;

CONST
  Colors = ARRAY OF CARDINAL:($F00,$FF0,$00F,$0F7); |Farbtabelle

VAR
  DefPubScreen,
  Screen        : I.ScreenPtr;
  DrawInfo      : I.DrawInfoPtr;
  ModeID        : LONGINT;
PROCEDURE Assert (c: BOOLEAN; REF txt : STRING);
VAR
  es : I.EasyStruct;
BEGIN
  IF NOT c THEN
    es := I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Screen Demo",
                        gadgetFormat = "Abort");
    es.textFormat := txt.data'PTR;
    FORGET I.EasyRequest (NIL, es'PTR, NIL);
    HALT (20);
  END;
END Assert;

BEGIN
  DefPubScreen := I.LockPubScreen (NIL);
  Assert (DefPubScreen # NIL, "Unable to lock default Public Screen");

  ModeID := g.GetVPModeID (DefPubScreen.viewPort'PTR);
  Assert (ModeID # g.invalidID, "Unable to get ModeID");


  Screen := I.OpenScreenTags (NIL,
```

```
                              width      : DefPubScreen.width,
                              height     : DefPubScreen.height,
                              depth      : DrawInfo.depth,
                              overScan   : I.text,
                              autoScroll : TRUE,
                              pens       : DrawInfo.pens,
                              sysfont    : I.wbSCreenFont,
                              displayID  : ModeID,
                              title      : "Cloned Screen",
                              pubName    : "SCRDEMO",
                              DONE);
  Assert (Screen # NIL, "Unable to open Screen");


  | Hier wird der Screen öffentlich
  FORGET I.PubScreenStatus (Screen, 0)

  d.Delay(50); | Warten

  g.LoadRGB4(Screen.viewPort'PTR,Colors'PTR,Colors'RANGE);

  d.Delay (5 * 50); | Warten
CLOSE
  IF DefPubScreen # NIL THEN
    I.UnlockPubScreen (NIL, DefPubScreen); DefPubScreen := NIL;
  END;
  IF Screen # NIL THEN
    WHILE NOT I.CloseScreen (Screen) DO

      | Falls der Screen nicht geschlossen werden konnte:
      FORGET I.EasyRequest (NIL,
          I.EasyStruct:(structSize   = I.EasyStruct'SIZE,
                        title        = "Screen Demo",
                        textFormat   = "Please close all visitor windows",
                        gadgetFormat = "So I did")'PTR,
                        NIL);
    END;
    Screen := NIL;
```



```
  END;
END ScreenDemo.
```

Alle Zeichenaktionen werden in der Farbe des APens ausgeführt, dieser ist im RastPort abgelegt. Um dem APen ein Farbregister zuzuweisen, existiert die Funktion `SetAPen()`:

```
PROCEDURE SetAPen( rp  IN A1 : RastPortPtr;
                   pen IN D0 : CARDINAL );
```

`rp` gibt dabei den Rastport an, dessen APen verändert werden soll. In `pen` steht das Farbregister, das dem APen zugeordnet werden soll.

Der BPen, der nicht wie man vielleicht vermuten könnte die Hintergrundfarbe enthält (diese steht immer in Farbregister 0), gibt die Farbe an, in der Lücken in Füllmustern oder hinter Text gefüllt werden soll. Um ihn zu verändern dient `SetBPen()`:

```
PROCEDURE SetBPen( rp  IN A1 : RastPortPtr;
                   pen IN D0 : CARDINAL );
```

Parameter siehe SetAPen.

Um einen ganzen RastPort einzufärben, können Sie den Befehl `SetRast()` verwenden:

```
PROCEDURE SetRast( rp  IN A1 : RastPortPtr;
                   pen IN D0 : CARDINAL );
```

`rp` gibt den Rastport an, der eingefärbt werden soll; `pen` das Farbregister, mit dessen Farbe gefärbt werden soll.

Bei Screens und GimmeZeroZero-Fenstern ist das kein Problem, da diese Art von Fenstern einen eigenen RastPort für den Rahmen haben. Bei normalen Fenstern muß man beachten, daß der Rahmen miteingefärbt wird.

5.6.3 Einfache Zeichenbefehle

Bevor Sie jedoch etwas zeichnen können, müssen Sie noch festlegen, in welcher Art die gesetzten Punkte mit der BitMap verknüpft werden. Hierzu dient `SetDrawMode()`:

```
PROCEDURE SetDrMd( rp   IN A1 : RastPortPtr;
                   mode IN D0 : DrawModeSet );
```

Für mode sind in Graphics zwei Konstanten definiert worden:

`jam1` Der Punkt wird direkt in den Rastport geschrieben, freibleibende Flächen, bei Textausgabe etwa der Raum im „O“, oder bei Füllmustern, lassen die schon existierende Graphic durchscheinen.

`jam2` Ebenfalls als Konstante definiert. Wie `jam1`, jedoch werden freie Flächen mit dem BPen gefüllt.

Die nächsten Modi arbeiten nur in Verbindung mit `jam1` oder `jam2`. Da `mode` ein Set ist, lassen sich folgende zusätzliche Flags setzen:

`complement` Alle Punkte werden, bevor sie gesetzt werden, exklusiv geodert oder „geXORt“. Dies ist manchmal praktisch, wenn man nur etwas markieren will. Zeichnet man nämlich noch einmal mit dem selben Modus über die selbe Stelle, kommt die ursprüngliche Graphic wieder zum Vorschein. Verwenden Sie dabei den `jam2`, werden auch noch APen und BPen vertauscht.

`inversvid` Ist eigentlich nur bei Textausgabe sinnvoll: Ausgegebener Text wird invertiert dargestellt. Wird dabei der `jam2` verwendet, ist dies so, als würden Sie nur jam2 alleine verwenden, lediglich APen und BPen vertauschen ihre Rollen.

Doch nun endlich zum Zeichnen, der erster Befehl dient dazu einen Punkt in der Farbe des APen in einen Rastport zu schreiben:

```
PROCEDURE WritePixel( rp IN A1 : RastPortPtr;
                      x  IN D0 : INTEGER;
                      y  IN D1 : INTEGER ): BOOLEAN;
```


`rp` gibt den Rastport an, in den gezeichnet werden soll. `x, y` sind die Koordinaten des Punktes (x/y) relativ zur linken, oberen Ecke des Rastport.

Mit der folgenden Funktion kann man die Farbe eines bestimmten Punktes abfragen:

```
PROCEDURE ReadPixel( rp IN A1 : RastPortPtr;
                     x  IN D0 : INTEGER;
                     y  IN D1 : INTEGER ): LONGINT;
```

`x, y` gibt die Position des Punktes an und als Ergebnis erhält man die Nummer des Farbregisters, mit der Farbe des Punktes oder -1, wenn ein Fehler auftrat.

Will man Linien zeichnen, muß man sich nicht selbst eine Routine mit `WritePixel()` schreiben, dafür gibt es `Draw()`. `Draw()` zeichnet eine Linie von der aktuellen Position des Graphic-Cursors zu einer angegebenen Position `x,y`:

```
PROCEDURE Draw( rp IN A1 : RastPortPtr;
                x  IN D0 : INTEGER;
                y  IN D1 : INTEGER );
```

Um den Graphic-Cursor zu versetzen, dient der Befehl `Move()`:

```
PROCEDURE Move( rp IN A1 : RastPortPtr;
                x  IN D0 : INTEGER;
                y  IN D1 : INTEGER );
```

Um einen Kreis oder Ellipse zu zeichnen, können Sie `DrawEllipse()` benutzen:

```
PROCEDURE DrawEllipse( rp IN A1 : RastPortPtr;
                       cX IN D0 : INTEGER;
                       cY IN D1 : INTEGER;
                       a  IN D2 : INTEGER;
                       b  IN D3 : INTEGER);
```

Diese Funktion zeichnet eine Ellipse mit dem horizontalen Radius `a` und dem vertikalen Radius `b` um den Mittelpunkt (`cX/cY`).

Um einen Kreis zu zeichnen, der eigentlich nur ein Sonderfall der Ellipse ist, muß man nur für `a` und `b` den gleichen Wert angeben. Theoretisch stimmt das, da jedoch die Bildpunkte des Monitors höher als breit sind (man hat ja auch keine 320*320 Punkte Auflösung), wird ein Kreis immer verzogen dargestellt.

Um dies zu verhindern, müssen die beiden Radien das gleiche Verhältnis zueinander haben, wie die horizontale zur vertikalen Auflösung der Screen.

Mit `RectFill()`, kann man gefüllte Rechtecke zeichnen:

```
PROCEDURE RectFill( rp   IN A1 : RastPortPtr;
                    xMin IN D0 : INTEGER;
                    yMin IN D1 : INTEGER;
                    xMax IN D2 : INTEGER;
                    yMax IN D3 : INTEGER );
```

Dabei wird das Rechteck über seine linke obere Ecke (`xMin/yMin` und seiner rechten unteren Ecke (`xMax/yMax`) definiert. Anmerkung: Das Rechteck wird mit dem momentan aktuellen Muster gefüllt.

Um beliebige Vielecke oder Polygone zu zeichnen dient der Befehl `PolyDraw()`:

```
PROCEDURE PolyDraw( rp    IN A1 : RastPortPtr;
                    count IN D0 : INTEGER;
                    array IN A0 : ANYPTR );
```

`count` gibt die Anzahl der Eckpunkte an, für `array` wird ein Zeiger auf eine Liste der Eckpunkte übergeben. Am besten, Sie verwenden hierfür ein konstantes **ARRAY OF** `INTEGER`, wobei immer abwechselnd die x- und die y-Koordinate angegeben wird.

Achtung: Da PolyDraw die erste Linie von der aktuellen Graphic-Cursor-Position aus zeichnet, sollte man den Cursor vor dem Prozeduraufruf auf den letzten Punkt seines Arrays setzen, da man ja einen geschlossenen Linienzug haben will.

Sie können mit Graphics auch Text ausgeben:


```
PROCEDURE TextStr(     rp     IN A1 : RastPortPtr;
                   REF string IN A0 : STRING;
                       count  IN D0 : LONGINT );
```

`string` ist der Clusterstring, der ausgegeben werden soll. Für `len` muß man die Anzahl der auszugebenden Zeichen angeben. Beispiel:

```
TextStr(rastPort,string,string.len);
```

Für alle C- und Assembler-Programmierer, die einen Pointer auf die Zeichenkette übergeben wollen, existiert die ursprüngliche Prozedur `Text()`.

Wichtig für die Verwendung von `TextStr()`: Der Text wird immer linksbündig ausgegeben. Dabei gibt der Cursor die Position der *baseline* des ersten Zeichens an.

5.6.4 Flächen und Muster

Neben den schon vorgestellten Zeichenbefehlen existieren noch die Area-Befehle. Diese sind eigentlich auch zum Zeichnen von Polygonen da, allerdings für wesentlich kompliziertere und außerdem ausgefüllte Polygone.

5.6.4.1 Flächen

Bevor man jedoch diese Befehle verwenden kann, muß man einige Vorbereitungen treffen. So muß man für all diese Befehle einen Speicherbereich allozieren, eine `TmpRas`- und eine `AreaInfo`-Struktur initialisieren.

Den Speicherbereich sollten Sie mit `Allocate()` aus `Resources` allozieren. Bitte setzen sie `chip` auf TRUE, da dieser Speicher im Chip-Memory liegen sollte.
Die Speichergöße errechnet sich folgendermaßen:

```
((Breite des Rastports+15)DIV16*2)*Höhe des Rastports
```

Um die `TmpRas`-Struktur zu initialisieren dient `InitTmpRas()`:

```
PROCEDURE InitTmpRas( VAR tmpras IN A0 : TmpRas;
                          buffer IN A1 : ANYPTR;
                          size   IN D0 : LONGINT );
```

Dieser Prozedur übergeben Sie eine Variable vom Typ `TmpRas`, einen Zeiger auf das Speicherstück, das Sie mit Allocate erhalten haben, sowie die Größe des Speicherstücks. Die Berechnung erfolgt wie bei der Allozierung.

Nachdem Sie eine TmpRas-Strucktur auf diese Weise vorbereitet haben, müssen Sie mit `InitArea()` noch eine AreaInfo-Struktur erzeugen:

```
PROCEDURE InitArea( VAR areainfo   IN A0 : AreaInfo;
                        buffer     IN A1 : ANYPTR;
                        maxvectors IN D0 : INTEGER );
```

Für `areainfo` übergibt man eine Variable des Typs `AreaInfo`, für `buffer` einen Zeiger auf den Puffer für die Punkt-Koordinaten. Am besten verwenden Sie hierfür eine Variable vom Typ `ARRAY OF [0..(Anzahl_Punkte*5)] OF SHORDCARD`. Wobei `Anzahl_Punkte` die maximale Anzahl der von Ihnen verwendeten Punkte sein sollte. Dabei muß man beachten, daß man für einen Kreis oder eine Ellipse (auch die ist mit Area-Befehlen machbar) 2 Punkte braucht. Am besten Sie planen immer etwas mehr Punkte ein, sicher ist sicher. `maxvectors` bezeichnet die maximale Anzahl von Punkten.

Hat man beide Variablen initialisiert, muß man nur noch jeweils einen Zeiger auf diese in die Rastportfelder `tmpRas` und `areaInfo` eintragen.

Hat man all diese Vorbereitungen getroffen, kann nun recht einfach ein Polygon abgesteckt werden. Mit `AreaMove()` setzen Sie den neuen Startpunkt für ihr Polygon und gleichzeitig werden noch offene andere Polygone geschlossen:

```
PROCEDURE AreaMove( rp IN A1 : RastPortPtr;
                    x  IN D0 : INTEGER;
                    y  IN D1 : INTEGER ): BOOLEAN;
```


(x/y geben dabei den neuen Startpunkt an. Falls ein Fehler auftrat, erhält man FALSE als Ergebnis.
Mit `AreaDraw()` kann man nun neue Punkte definieren:

```
PROCEDURE AreaDraw( rp IN A1 : RastPortPtr;
                    x  IN D0 : INTEGER;
                    y  IN D1 : INTEGER ): BOOLEAN;
```

Haben Sie alle Punkte definiert, müssen Sie es mit `AreaEnd()` schließen. Dabei muß man nicht wie bei `PolyDraw()` den ersten Punkt noch einmal anwählen. `AreaEnd()` verbindet automatisch den zuletzt gesetzten Punkt mit dem ersten und füllt die Fläche mit dem momentan aktuellen Muster. Ist kein Muster definiert, wird die Fläche ganz in der Farbe des APens gefüllt. Tritt ein Fehler auf, wird FALSE zurückgegeben.

```
PROCEDURE AreaEnd( rp IN A1 : RastPortPtr ): BOOLEAN;
```

`rp` gibt dabei den Rastport an, dessen Areadefinition abgeschlossen werden soll. Tritt ein Fehler auf, wird FALSE zurückgegeben.

Man kann jedoch nicht nur Polgone, sondern auch Kreise und Ellipse zeichnen:

```
PROCEDURE AreaEllipse( rp IN A1 : RastPortPtr;
                       cX IN D0 : INTEGER;
                       cY IN D1 : INTEGER;
                       a  IN D2 : INTEGER;
                       b  IN D3 : INTEGER): BOOLEAN;
```

Auf eine Erklärung der Parameter kann wohl verzichtet werden.

Möchte man, daß eine mit Area-Befehlen erzeugte Fläche eine Umrandung erhält, so muß man dazu die Nummer des gewünschten Farbregisters in das `aOlPen`-Feld des Rastports eintragen.

Ein weiterer Befehl, der eigentlich kein Area-Befehl ist, jedoch ebenfalls eine TempRas-Struktur benötigt, ist `Flood()`:

```
PROCEDURE Flood( rp   IN A1 : RastPortPtr;
                 mode IN D2 : LONGCARD;
                 x    IN D0 : INTEGER;
                 y    IN D1 : INTEGER ):BOOLEAN;
```

`Flood()` füllt eine Fläche mit dem aktuellen Muster von einem bestimmten Punkt an in der Farbe des APens.
die Parameter sind wahrscheinlich bis auf `mode` verständlich. Für diesen Parameter kann man zwei Werte übergeben:

1 Der gesamte zusammenhängende Bereich in der Farbe des Punktes (`x/y`) wird gefüllt.

2 Es wird solange gefüllt, bis `Flood()` auf die Farbe des `aOlPen`-Feldes im Rastort trifft.

All diese Funktionen geben zwar einen Boolwert zurück, diesen kann man jedoch meistens mittels **FORGET** vergessen.

5.6.4.2 Muster bei Linien

Nun war mehrmals vom aktuellen Füllmuster die Rede. Sie können nämlich nicht nur komplett ausgefüllte Flächen und Linien erzeugen, sondern diese mit einem Füllmuster versehen.

Ein Linienmuster ist 16 Bit groß, entspricht also einem Word oder einer Cardinal-Zahl, in der jedes gesetzte Bit für einen gesetzten Punkt steht. Gesetzte Punkte werden dabei in der Farbe des APens, nicht gesetzte in der Farbe des BPens gezeichnet. Normalerweise sind hier alle Bits gesetzt, also %1111111111111111. Um z. B. eine gepunktete Linie zu erhalten, müßte man folgendes Muster definieren: %1100110011001100. Damit ein solches Muster auch in einem Rastport verwendet wird, muß man das Muster in das `linePtrn`-Feld des Rastports eintragen. Beispiel:

```
window.rPort.linePtrn:=$1100110011001100;
```

5.6.4.3 Flächenmuster

Ein Flächenmuster ist ebenfalls 16 Bit breit, und eine beliebige Zweier-Potenz (also 1,2,4,8,16...) hoch. Als Höhe gibt man jedoch nicht die Anzahl, sondern nur deren Zweier-Potenz an, also 3 für 8 ($2^3 = 8$). Am besten definiert man es als konstantes **ARRAY OF** `CARDINAL`. Danach muß man einen Zeiger auf das Array sowie die Zweier-Potenz in den Rastport eintragen:


```
CONST
 pattern = ARRAY OF CARDINAL: ( %0011001100110011 ,
                                %1100110011001100 );

BEGIN
 ...
 window.rPort.areaPtrn:=pattern'PTR;
 window.rPort.areaPtSz:=1;
```

Um ein Muster zu löschen müssen Sie in `areaPtrn` NIL und in `areaPtSz` 0 schreiben.

Neben einfarbigen Füllmustern sind auch mehrfarbige möglich. Dazu muß man für jede BitPlane der BitMap, für die das Muster gelten soll, ein eigenes Muster definieren. Dabei wird aus der Kombination in den verschiedenen Planes die endgültige Farbe des Punktes ermittelt. Das folgende Beispiel ist gültig für zwei BitPlanes:

Plane 1	Plane 2	Farbe
0	0	0
1	0	1
0	1	2
1	1	3

Um dem System mitzuteilen, daß man im MultiColor-Mode arbeiten möchte, muß man die Musterhöhe negativ angeben. Zusätzlich sollte man

- Den APen auf 255 setzen, als Anzeige für MultiColor-Mode.
- Den BPen auf 0 setzen
- jam2 als Zeichenmodus wählen.

Beispiel:

```
TYPE
 Pattern = ARRAY [0..1],[0..3] OF CARDINAL;
```

```
CONST
  pat = Pattern:(%0011001100110011, (* Erste BitPlane *)
                 %0111100001111000,
                 %0111100001111000,
                 %0011001100110011),

                (%0100110011001100, (* Zweite BitPlane *)
                 %1011010010110111,
                 %1011010010110111,
                 %0100110011001100));

BEGIN
  ...
  WITH window.rPort AS r DO
    r.areaPtrn:=pattern'PTR;
    r.areaPtSz:=-2;
    SetAPen(r,255);
    SetBPen(r,0);
    SetDrMd(r,jam2);
    RectFill(r,100,100,150,170);
  END
  ...
END;
```

Vervollständigen Sie doch das obige Beispiel und probieren Sie es aus.

5.6.5 Weitere Befehle

Möchte man einen bestimmten Bereich des Bildschirm scrollen, so kann man dazu `ScrollRast()` benutzen:

```
PROCEDURE ScrollRaster( rp   IN A1 : RastPortPtr;
                        dx   IN D0 : INTEGER;
                        dy   IN D1 : INTEGER;
                        xMin IN D2 : INTEGER;
                        yMin IN D3 : INTEGER;
                        xMax IN D4 : INTEGER;
                        yMax IN D5 : INTEGER );
```


Der Inhalt des Rechtecks mit der linken oberen (`xMin/yMin`) und rechten unteren Ecke (`xMax/yMax`) wird um `dx` Pixel in horizontaler und um `dy` Pixel in vertikaler Richtung verschoben.

Bemerkung: Scrollraster verschiebt nicht einen Bereich auf einer Screen, sondern nur den Inhalt des Rechtecks innerhalb des selben. Dabei verringert dieser sich dementsprechend. Verschiebt man ihn um positive `dx/dy`-Werte, wird der Rechteckinhalt in Richtung (0/0) verschoben. Der freiwerdende Platz im Rechteck wird in der Farbe des BPens gefüllt.

Bei manchen Zeichenoperationen kann starkes Flackern auftreten. Um dies zu verhindern, sollte man diese Zugriffe auf die Bitmap nur innerhalb der Austastlücke des Zeilenstrahls machen. Um auf diese zu warten, dient:

```
PROCEDURE WaitTOF();

PROCEDURE WaitBOVP( vp IN A0 : ViewPortPtr );
```

`WaitTOF()` wartet, bis der Zeilenstrahl am unteren Bildschirmrand angekommen ist und den Rücklauf nach oben antritt. `WaitTOF()` erhält seine Information direkt aus einem Hardwareregister; man muß kein Parameter angeben.

`WaitBOVP()` wartet, bis der Elektronenstrahl am unteren Ende des übergebenen ViewPorts angekommen ist. Bei mehreren überlappenden Screens wartet er also bis zum Ende des sichtbaren Bereichs der Screen.

In einigen Fällen ist nötig, die aktuelle Zeilenposition des Elektronenstrahls zu wissen, hierzu dient `VBeamPos()`:

```
PROCEDURE VBeamPos(): LONGINT;
```

Als Erbegnis erhält man die aktuelle vertikale Position des Zeilenstrahls.

5.6.6 HAM und Extra-Halfbrite.

Bisher gingen wir immer davon aus, daß eine Screen höchstens 32 Farben haben kann. Es sind aber auch noch mehr möglich, nämlich 64 oder gar 4096. Für beide Modi wird eine Bitplane zusätzlich benötigt. Bei welchen Auflösungen die möglich ist, hängt von der Version der Graphikchips ab, die in Ihrem Amiga eingebaut sind.

Um auf 64 Farben im sogenannten Extra-Halfbrite-Mode zu gelangen, müssen Sie lediglich als Tiefe beim Öffnen 6 angeben und für den Tag `displayID palMonitorID + extrahalfbriteKey` eintragen. Dadurch erhalten wir 64 unabhängige Farbregister. Leider sind jedoch nicht alle 64 Farben unabhängig, sondern die oberen 32 sind die selben wie die unteren 32, jedoch mit der halben Helligkeit. Wenn man aber die Farbwahl geschickt trifft, kann man mit dieser Einschränkung ganz gut arbeiten.

Beim HAM-Modus müssen Sie ebenfalls 6 Bitplanes verwenden und `palMonitorID + hamKey` beim Öffnen des Screens angeben. Danach können Sie 4096 Farben verwenden, allerdings nicht ganz unabhängig. Zuerst müssen Sie 16 frei gewähle Farben in Farbregister 0–15 eintragen.

Um nun von 16 auf 4096 Farben zu kommen, verwendet das Betriebssystem einen kleinen Trick. Sie können entweder die Farben 0–15 verwenden, oder die Farbe des Punktes links von ihrer Position auf dem Bildschirm übernehmen und eine der drei Farbkomponenten Rot, Grün oder Blau verändern. Hierzu dienen die Register 16–63. Dabei ist 16–31 für die Veränderung der Blaukomponente, 32–47 für die Veränderung der Rotkomponente, 48–63 für die Veränderung der Grünkomponente zuständig.

Welche neue Farbkomponente welcher Registernummer zugeordnet ist, geht aus der nächsten Tabelle hervor:

Nummer	Blauwert	Nummer	Rotwert	Nummer	Grünwert
16	0	32	0	48	0
17	1	33	1	49	1
18	2	34	2	50	2
19	3	35	3	51	3
20	4	36	4	52	4
21	5	37	5	53	5
22	6	38	6	54	6
23	7	39	7	55	7
24	8	40	8	56	8
25	9	41	9	57	9
26	10	42	10	58	10
27	11	43	11	59	11
28	12	44	12	60	12
29	13	45	13	61	13
30	14	46	14	62	14
31	15	47	15	63	15

Noch einmal: Verwendet man die Register 16–63, werden zwei Farbkomponenten beibehalten und eine verändert. Um also von einer Farbe auf eine komplett neue, die nicht in Register 0–15 vorhanden ist, zu kommen, benötigt man immer mindestens drei Pixel, da bei jedem Schritt ja nur eine der drei Komponenten verändert werden kann.

Wichtig: Man kann keine Farbe von einer Bildschirmzeile in die nächste übernehmen. Der Punkt, der theoretisch links vom linken Bildschirmrand liegen würde, hat daher immer die Hintergrundfarbe, also die Farbe aus Register 0.

Selbstverständlich kann dieses Kapitel Graphics nicht vollständig abhandeln. Dies soll auch gar nicht Sinn dieses Kapitels sein. Es soll nur einen Abriß der wichtigsten Funktionen bieten, damit man die wunderbaren Grafikfähigkeiten des Amigas nutzen kann, ohne erst viele Bücher zu kaufen zu müssen.

5.7 Devices

In einem Multitaskingsystem kann nicht jeder Task frei über alle Resourcen des Systems verfügen. Alle Ein/Ausgaben werden über spezielle Gerätetreiber, sogenannte „Devices“ abgehandelt. Für den Bildschirm gibt es keine speziellen Gerätetreiber, da dies einfach zu langsam und unflexibel wäre.

Ein Gerätetreiber ist eine Art Library mit sehr wenigen Prozeduren. Diese werden im allgemeinen nicht durch ein Programm direkt angesprungen, sondern indirekt über Exec.

Ein Device muß, bevor darauf zugegriffen werden kann, erst geöffnet werden. Dies ist etwas komplexer als das Öffnen einer Library, da erst noch einige Strukturen erzeugt werden müssen. Devices können auch nicht im Begin-Teil des zugehörigen Schnittstellenmodus geöffnet werden, da manche Devices nur immer mit einem Task gleichzeitig zusammenarbeiten.

Um diesem Problem Herr zu werden, gibt es in jedem Schnittstellenmodul eine Open- und eine Close-Prozedur. Wird ein Device nicht wieder geschlossen, schließt der Close-Teil des Schnittstellenmoduls dieses automatisch am Ende des Programms.

Die Kommunikation mit einem Device erfolgt über IORequests, die diesem über Exec geschickt werden. Es handelt sich dabei um eine Erweiterung der Message-Struktur. Diese wird allerdings nicht mit `PutMsg()` an das Device übermittelt, sondern mit speziellen IO-Prozeduren von Exec. Man erhält einen Zeiger auf eine initialisierte IO-Struktur beim Öffnen eines Devices. Diese muß immer verwendet werden, wenn das Device benutzt wird.

Es gibt in Exec fünf Prozeduren, die sich mit Devices befassen und für uns interessant sind:

```
PROCEDURE SendIO( ioRequest IN A1 : IORequestPtr );
```

Diese Prozedur schickt eine Ein/Ausgabeanforderung an ein Device. Sie wartet nicht bis die Aktion beendet ist, sondern kehrt sofort zurück. Die Aktion läuft dann parallel im Hintergrund ab.

```
PROCEDURE CheckIO( ioRequest IN A1 : IORequestPtr ): IORequestPtr;
```

`CheckIO()` prüft, ob die in Auftrag gegebene Aktion bereits beendet ist. Als Ergebnis erhält man NIL, falls der Auftrag noch bearbeitet wird, sonst einen Zeiger auf den IORequest.

```
PROCEDURE WaitIO(ioRequest IN A1 : IORequestPtr):IOReturn;
```

`WaitIO()` setzt den Task solange in den Wartezustand, bis die Ein/Ausgabeaktion beendet ist.

```
PROCEDURE AbortIO(ioRequest IN A1 : IORequestPtr);
```

`AbortIO()` bricht die aktuelle Ein/Ausgabeaktion ab.

```
PROCEDURE DoIO(ioRequest IN A1 : IORequestPtr):IOReturn;
```

`DoIO()` beginnt eine Ein- / Ausgabeaktion und warte bis diese beendet ist. Als Rückgabewert erhält man, ob ein Fehler augetreten ist, dazu gleich mehr. Exec erkennt das gewünschte Device am IORequest, der übermittelt wird. Dieser unterscheidet sich leicht bei den meisten Devices. Gemeinsam sind die Felder:

```
IORequest     = RECORD OF Message
                  device  : DevicePtr;
                  unit    : UnitPtr;
                  command : IOCommand;     | CARDINAL
                  flags   : IOFlagSet;
                  error   : IOReturn;      | SHORTCARD
                END;
```


`device` : Zeiger auf das zugehörige Device.

`unit` : Nummer des angesprochenen Gerätes, falls ein Device für mehrere Geräte zuständig ist, wie z. B. das `trackdisk.device`: 0 für DF0:; 1 für DF1: etc.

`command` : Befehl, der ausgeführt werden soll. Eigentlich nur eine Nummer, in Cluster die Elemente eines Aufzählungstyps.

`flags` : Spezielle Flags, die von Device zu Device variieren. Gemeinsam ist Bit 0, das angibt, ob eine Operation im Quick-Mode durchgeführt werden kann.

`error` : Fehlerrückmeldung des Devices. Folgende Meldungen sind immer möglich:

`ioOk` Kein Fehler, alles ok.

`openFail` Das Device konnte nicht geöffnet werden.

`aborted` Der Befehl wurde abgebrochen.

`noCmd` Unbekanntes Kommando.

Diese Fehlermeldungen kann man auch von `DoIO()` als Ergebnis erhalten.

Viele Devices benutzen einen leicht erweiterten `IORequest`, den `IOStdRequest`. Dieser verfügt über folgende zusätzliche Felder:

```
IOStdReq      = RECORD OF IORequest
                  actual,
                  length  : LONGCARD;
                  data    : ANYPTR;
                  offset  : LONGCARD;
                END;
```

`actual` : Anzahl der bewegten Bytes.

`length` : Anzahl der zu bewegenden Bytes.

`data` : Zeiger auf Datenpuffer im Speicher, muß bei manchen Devices im Chip-Mem liegen.

`offset` : Position relativ zu `data`, ab der die Daten gelesen/geschrieben werden sollen.

Einige Ein/Ausgabebefehle sind für alle Devices standardisiert:

invalid Kein Befehl.

reset Device zurücksetzen.

read Vom Device lesen.

write In Device schreiben.

update Alle Puffer im Speicher ausschreiben.

clear Puffer leeren.

stop Alle Aktionen des Devices werden unterbunden.

start Die Aktionen, die durch `stop` unterbunden wurden, werden wieder weiter ausgeführt.

flush Alle Ein/Ausgabeaktionen von der Warteliste streichen.

Folgende Devices sind im Amiga standardmaßig vorhanden:

Audio : Soundausgaben.

Clipboard : Ablage von Datenblöcken.

Console : Zeichenorientierte Eingabe über ein Konsolfenster.

Gameport : Maus, Joysticks und Paddels.

Input : Vereinigt das Gameport- und das Keyboard-Device.

Keyboard : Eingabe über Tastatur.

Narrator : Sprachsynthesizer.

Parralel : Parallel-Schnittstelle.

Printer : Druckerausgabe.

Serial : Ein/Ausgabe über die serielle Schnittstelle.

Timer : Zeitgeber.

Trackdisk : interne und externe Diskettenlaufwerke.

Auf einge dieser Devices möchte ich nun etwas genauer eingehen. Dies soll keine Abhandlung des Themas „Devices“ sein, da dies dafür sicher nicht der richtige Ort ist, lediglich ihre Verwendung unter Cluster soll gezeigt werden. Für weitere Informationen seien der Band *Devices* der „Amiga ROM-Kernal Referance Manuals“ von Addison Wesley empfohlen.

5.7.1 Serial Device

Das Serial-Device steuert die Ein/Ausgabe über die interne serielle Schnittstelle. Dazu gibt es einige zusätzliche Befehle und einen erweiterten IO-Request. Folgende neuen Befehle gibt es:

query Stellt fest, wieviele Zeichen im Eingabepuffer enthalten sind.

break Abbruch der Übertragung.

setParams Einstellung für die Schnittstelle ändern.

Die Datenübertragung erfolgt über Befehle `read` und `write`.

Um das Device zu öffen, müssen noch einige zusätzliche Infomationen übergeben werden. Dies geschieht über Tags. Davon interessieren uns allerdings nur die Tags `name` und serFlags:

`name` : `SystStringPtr;`
Der Namen des Devices. Schließlich gibt es in der Zwischenzeit einige serielle Erweiterungskarten, die befehlskompatibel zum serial.device sind. Wird dieser Tag nicht angegeben, wird das serial.device geöffnet.

`serFlags : SerFlagSet;`
Einige Flags, die bestimmte Einstellungen bei dem Device bewirken. Hiervon sei hier nur das Flag `shared` erwähnt. Dieses Flag sollte man setzen, wenn man das Device auch anderen Tasks gleichzeitig zugänglich machen will. Will man von der seriellen Schnittstelle gleichzeitig lesen und schreiben, muß man das Device zweimal `shared` öffnen und verwendet zum Lesen und Schreiben jeweils eine eigene IOSerial-Struktur.

Das folgende Beispiel zeigt, wie man mit dem serial.device Daten ausgibt:

```
MODULE SerialWrite;
FROM Serial IMPORT IOSerialPrt,OpenSerial,CloseSerial,SerFlags,
                   SerFlagSet,IOSerialTags;
FROM Exec   IMPORT DoIO,write;

CONST
 Text = "Dies ist ein Text !"+&10;

VAR
 ioser : IOSerialPtr;

BEGIN
 | Versuchen, das Device zu öffnen
 ioser := OpenSerial(tags:= serFlags : {shared});
  | Befehl vorbereiten
  WITH ioser^ AS s  DO
    s.command := write;
    s.length := Text.len;
    s.data := Text.data'PTR;
  END;
  | Befehl ausführen *)
  FORGET DoIO(ioser);
  | Normalerweise müßte man hier prüfen, ob ein  Fehler auftrat.
  | Device wieder schliessen
  CloseSerial ( ioser );
END SerialWrite.
```

Soll vom Device gelesen werden, empfiehlt es sich nicht einfach stur zu warten, sondern eine Auszeit zu verwenden, nach der die Eingabe abgebrochen wird:

```
MODULE SerialRead;
FROM Serial IMPORT IOSerialPrt,OpenSerial,CloseSerial,SerFlags,
                   SerFlagSet,IOSerialTags;
FROM Exec   IMPORT SendIO,CheckIO,AbortIO,WaitIO,read;
FROM InOut  IMPORT WriteString;

VAR
 ioser  : IOSerialPtr;
 buffer : STRING(200);

BEGIN
  | Versuchen , das Device zu öffnen
  ioser := OpenSerial(tags:= serFlags : {shared});
  | Befehl vorbereiten
  WITH ioser^ AS s  DO
    s.command := read;
    s.length := -1; (* beliebige Länge, bis 0 Byte empfangen *)
    s.data := buffer.data'PTR;
    INCL ( s.serFlags, eofMode );
  END;
  |Befehl starten
  SendIO(ioser);
  |
  | hier irgentwas tun, oder ein bestimmte Zeit warten
  |
  IF CheckIO(ioser^)#NIL  THEN | Schon fertig ?
    | Eingabe beenden
    WaitIO ( ioser^ );| Warten bis das Device wirklich fertig ist.
    | Eigentlich hier Fehler abfragen !
    buffer.len := ioser^.actual;
    WriteString(buffer);
  ELSE
    | Eingabe Abbrechen
    AbortIO(ioser);
```



```
    WaitIO(ioser);
  END;
  | Device wieder schliessen
  CloseSerial(ioser);
END SerialRead.
```

Neben diesen Lese- und Schreiboperationen gibt es auch die Möglichkeit, die Parameter der seriellen Schnittstelle zu verändern. Dies läuft über den Befehl `setParams`. Die Felder des erweiterten IORequest müssen entsprechend gefüllt werden. Hier der erweiterte Request:

```
IOSerial       = RECORD OF IOStdReq
                   ctlChar   : LONGCARD;
                   rBufLen   : LONGCARD;
                   extFlags  : ExtSerFlagSet;
                   baud      : LONGCARD;
                   brkTime   : LONGCARD;
                   termArray : ARRAY [0..1] OF LONGCARD;
                   readLen   : SHORTCARD;
                   writeLen  : SHORTCARD;
                   stopBits  : SHORTCARD;
                   serFlags  : SerFlagSet;
                   status    : StatusSet;
                 END;
```

`ctlChar` Kontrollzeichen für XON/XOFF Protokoll.

`rBufLen` Länge des device-eigenen Eingabepuffers.

`extFlags` Flags für Paritätsprüfung.

`baud` Baudrate der Übertragung.

`brkTime` Zeit, die für ein Breaksignal nötig ist.

`termArray` 8 Bytes, die bei Empfang die Eingabe abbrechen.

`readLen` Anzahl der Bits pro Zeichen, die gelesen werden.

`writeLen` Anzahl der Bits pro Zeichen, die geschrieben werden.

`stopBits` Abzahl der Stop-Bits pro Zeichen.

`serFlags` Flags, die die Übertragung steuern. Es handelt sich um die selben Flags, die schon beim Öffnen des Devices verwendet wurden:

- `parityOn` Paritätsprüfung einschalten.
- `parityOdd` Auf ungerade Parität prüfen.
- `sevenWire` Erweiterte Schnittstelle nutzen.
- `queuedBrk` Break-Befehle werden in einer Warteschlange abgearbeitet.
- `radBoogie` Es wird auf alles Handshaking verzichtet, um maximale Übertragungsgeschwindigeit zu sichern.
- `shared` Andere Tasks können die Schnittstelle gleichzeitig benutzen.
- `eofMode` Veranlaßt, das die Eingabe bei Übereinstimmung eines Zeichens mit dem TermArray abgebrochen wird.
- `xDisabled` XON/XOFF abschalten.

`status` Status der Schnittstelle.

Für die einzelnen Felder existieren auch Tags, so daß man bereits beim Öffnen Einstellungen machen kann.

5.7.2 Timer Device

Das Timerdevice verwaltet die im Amiga enthaltenen Uhren. Dies sind die Zähler der CIAs, die Mikrosekunden zählen, und ein Zähler, der die VerticalBlanks des Bildschirms zählt. Er läuft mit 50 Hz bzw. 60 Hz (NTSC). Das Device wird mit eigenen Befehlen gesteuert:

getSysTime Holt die Systemzeit.

setSysTime Setzt die Systemzeit.

addRequest Fügt einen Warterequest in die Warteschlange ein.

Die Uhrzeit setzt sich aus Sekunden und Mikrosekunden zusammen (leider ist es mir noch nie gelungen, Zeiten mit einer größeren Genauigkeit als 20 Millisekunden zu erhalten).

Neben diesen Device typischen Befehlen gibt es noch Prozeduren, die eher an eine Library erinnern. Sie werden wie ganz normale Libraryfunktionen benutzt. Vorraussetzung ist allerdings, daß das Device geöffnet ist. Folgende Funktionen gibt es:

```
PROCEDURE AddTime(VAR dest   IN A0,
                      source IN A1  : TimeVal);
```

Addiert die Zeitdauer in `source` zur Zeit in `dest`.

```
PROCEDURE SubTime(VAR dest   IN A0,
                      source IN A1  : TimeVal);
```

Zieht von der Zeitdauer in `dest` die Zeit `source` ab.

```
PROCEDURE CmpTime(VAR time1 IN A0,
                      time2 IN A1  : TimeVal):INTEGER;
```

Vergleicht die Zeit `time1` mit der Zeit in `time2`. Das Ergebnis ist:

- -1 für `time1` ist später als `time2`
- 0 für `time1` ist gleich `time2`

1 für `time1` früher als `time2`

Beim Öffnen dieses Devices ist es nötig, die Art der Uhr, die verwendet werden soll, anzugeben.
Beispiel: Messen einer Zeitdauer

```
MODULE Duration;
FROM Exec  IMPORT DoIO;
FROM Timer IMPORT IOTimerPtr,OpenTimer,CloseTimer,getSysTime,vBlank,
                  SubTime,TimeVal;

VAR
 start,end : TimeVal;
 iotime    : IOTimePtr;

PROCEDURE GetTime (VAR time : TimeVal);
BEGIN
  iotimer.command := getSysTime;
  DoIO(iotimer);
  time := iotimer.time;
END GetTime;

PROCEDURE WriteTime (time : TimeVal);
BEGIN
  WriteReal(LONGREAL(time.secs)+LONGREAL(time.micro)/1.E6,10,2);
END WriteTime;

VAR
  i : INTEGER;

BEGIN
  iotimer:=OpenTimer(vBlank);
  GetTime(start);

  FOR i := 1 TO 100000 DO END;

  GetTimer(end);
  SubTime(end,start);
```



```
  WriteTime(end); WriteLn;
  CloseTimer(iotimer);
END Duration;
```

Und jetzt noch ein Beispiel 100 Millisekunden zu warten:

```
 ...
 iotimer.command := addRequest;
 iotimer.time.secs := 0;
 iotimer.time.micro := 1000000;
 DoIO ( iotimer );
 ...
```

Kapitel 6

Hardwarenahe Programmierung

6.1 Wozu hardwarenah programmieren?

Ein Prozessor, wie der 68000 (oder auch ein Prozessor höheren Typs) in ihrem Amiga ist kein Spezialist, sondern mehr ein Allroundtalent. Er kann keine komplizierten Dinge, ist aber in dem, was er kann, sehr schnell. So verfügt er über eine Anzahl von Befehlen, die nur ziemlich wenig bewirken, z. B. zwei INTEGER-Zahlen zu addieren. Daher ist das Programmieren in Maschinensprache extrem aufwendig, und nur schwer zu durchschauen. Eine Hilfe ist ein Assembler, der den Befehlen Namen gibt[2], und sonst noch einige Unterstützung bietet. Dennoch bleibt diese Art des Programmierens aufwendig und fehlerträchtig.[3]

In einer höheren Programmiersprache (wie **Cluster**) geht die Programmierung wesentlich schneller von der Hand, da man, um den gleichen Effekt wie in Assembler zu erreichen, eine viel geringere Zahl an Befehlen benötigt. Außerdem verfügt ein Compiler über eine Vielzahl von Kontrollen, die die Fehlersuche erleichtern. Obwohl ein von einem Compiler übersetztes Programm in den meisten Fällen schnell genug ist[4], gibt es doch Fälle, in denen ein rein in Maschinensprache geschriebenes Programm schneller oder durch das System bedingt notwendig ist.

Maschienennahes Programmieren bedeutet nun, daß man sich diesen untersten Ebenen annähert. Das bedeutet zum einen, maschienennahe Elemente der Sprache Cluster zu nutzen, oder gar direkt in Assembler[5] zu programmieren.

[2]Der Prozessor kennt schließlich nur Zahlen als Befehle

[3]Auch wenn dies von fanatischen Assembler-Programmierern bestritten wird. Es soll sogar Leute geben, die in Assembler komplette C++ Compiler entwickelt haben.

[4]Oft könnte nur ein sehr guter Assembler-Programmierer das erzeugte Maschinenprogramm noch verbessern

[5]Allerdings sehr komfortabel

6.2 Wie funktioniert ein Computer

Ein Computer ist eine Maschine, wie andere auch. Der Unterschied zu z. B. einem Auto ist, daß sich nichts sichtbar bewegt[6]. Doch erfüllt er viele Aufgaben, von denen man meinen könnte, daß sie Intelligenz verlangten. Dieses Image des Elektronengehirns, hat durch viele SF-Filme weitere Nahrung gewonnen[7].

So ist es nicht verwunderlich, daß viele Menschen denken, ein Computer sei so kompliziert, daß ein *normaler* Mensch ihn gar nicht verstehen könne. Denken Sie immer daran, ihr Rechner kann nicht einmal bis drei zählen.

Ein Computer ist ein extrem modulares Gebilde, d. h. er gliedert sich einem Organismus ähnlich, in immer kleinere Teile. Die Extremitäten sind die Peripherie, die Organe die Bausteine (Chips), die Zellen die Schaltkreise, die sich wieder in Transistoren u. ä. zergliedern lassen. Es ist nicht nötig, jederzeit den Computer bis ins kleinste zu kennen, es ist nur wichtig, die Übergänge der einzelnen Stufen zu verstehen.

6.2.1 Von Bits, Bytes, Hexen und Zeichen

Den einfachsten Computer, den es wohl gibt, halten Sie immer in Händen, ihre Finger. Nehmen wir an Sie können jeden Finger entweder ausstrecken oder zurückbeugen, also zwei Zustände anzeigen. Ein Object, daß zwei Zustände (ja/nein, 0/1, wahr/falsch, TRUE/FALSE) annehmen kann, bezeichnet man als Bit, genauer hat die Information ein Bit[8].

Ein Bit ist die kleinste mögliche Informationseinheit. Ihre Finger haben eine Kapazität von 10 Bit, ihr Amiga hat mindestens 8 Millionen davon.

Wie weit kann man mit zehn Fingern zählen? Bis zehn natürlich,

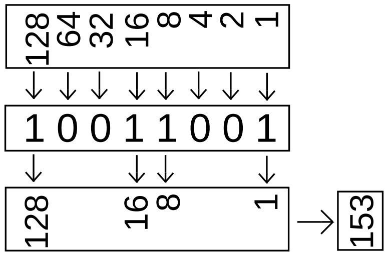
[6]Wenn man einmal von dem *Föhn* im `A 2000` absieht

[7] Es soll ja Leute geben, die sich bekreuzigen, wenn sie einen Computer sehen

[8]Von Binary Digit, engl. (extrem wörtlich) „Finger“, der zwei Zustände einnehmen kann. Sie sehen, auch ein Computer rechnet mit seinen Fingern, er hat allerdings einge Millionen davon

darum auch unser Dezimalsystem. Mit dem richtigen Verfahren kann man mit ihren Fingern allerdings auch wesentlich weiter zählen, etwa bis 1000[9].

Jedem Bit wird ein Wert zugeordnet, dem ganz rechts der Wert 1, dem nächsten 2 dann 4,8,16,32..., immer den vorigen Wert mal zwei. Der effektive Wert der Zahl ergibt sich aus der Summe der Werte der gesetzten Bits.



Die Bits einer Zahl werden von rechts nach links durchnumeriert, das niederwertigste Bit erhält die Nummer 0. So hat das Bit mit der Nummer n, den Wert 2^n (also $2 \cdot 2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2, n$ mal). Diese Schreibweise ist analog zur normalen Schreibweise im Zehnersystem, nur daß der Faktor von einer Ziffer zu nächsten nicht 10, sondern nur zwei ist.

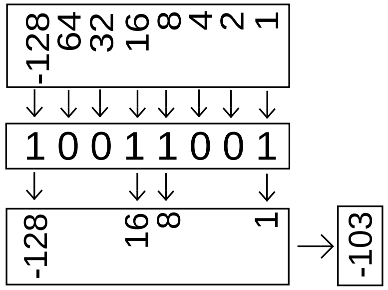
Es leuchtet so ein, daß es möglich ist, mit n Bits alle Zahlen von 0 bis $(2^n) - 1$ darzustellen. Eine Addition mit Binärzahlen (so der Fachausdruck) verläuft wie eine Addition im Dekadischen (Zehner-) System.

```
 123   01111011
+247  +11110111
----  ---------
 370  101110010
```

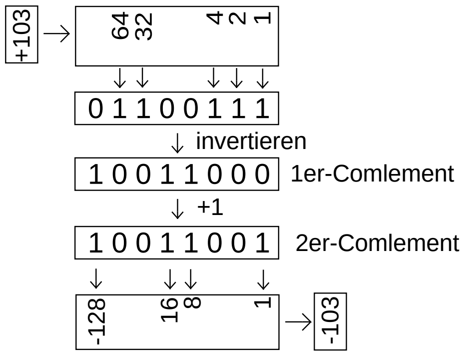
Um die Subtraktion zu erläutern, müssen zuerst die negativen Zahlen definiert werden. Um möglichst gleich viele positive wie negative Zahlen

[9] wenn auch ein wenig Biegsamkeit verlangt wird

zu haben, verwendet man ein zusätzliches Bit, um das Vorzeichen zu markieren. Dieses Bit ist jenes, welches ganz links liegt. Ist es 0, ist die Zahl positiv, sonst negativ. Dieses Bit hat bei n Bits einfach den Wert $-2^{(n-1)}$.



Um dieses Verfahren anwenden zu können, muß man eine feste Länge der Zahlen vorgeben, sonst könnte man das Vorzeichenbit nicht vom höchstwertigen einer positiven Zahl unterscheiden. Diese Art der Darstellung negativer Zahlen nennt man 2er Komplement. Warum, das wird deutlich, wenn man sich die Umwandlung einer Zahl in die entsprechende negative Zahl ansieht. Zuerst werden alle Bits umgedreht (aus 1 wird 0 und aus 0 wird 1), dies nennt man 1er Komplement. Danach wird zu der entstandenen Zahl 1 addiert.



Nachdem man die negativen Zahlen eingeführt hat, kann man eine Subtraktion auf eine Addition mit dem 2er Komplement zurückführen.

Da bei einer Darstellung dieser Art festgelegt wurde, daß alle Zahlen eine bestimmte Länge haben, kann es passieren, daß ein Wert nicht mehr in den darstellbaren Bereich hineinpaßt. Ein Fehler dieser Art nennt man Überlauf:
Beispiel, 8 Bits:

```
 112  01110000
+ 52 +00110100
---- ---------
 164  10100010 = -92
```

Passiert in Cluster eine derartige Bereichsverletzung, wird, sofern er nicht ausgeschaltet wurde, ein Laufzeitfehler ausgelöst, da ein derartiger Überlauf in den wenigsten Fällen erwünscht ist.

Da es ziemlich unpraktisch ist, immer in Bits zu rechnen, faßt man immer 8 Bits zu einem Byte zusammen. Weitere Zusammenfassungen sind beim 680XXer 16 Bits ein Wort, 32 Bits ein Langwort und 64 Bits ein Doppelwort[10]. In Cluster repräsentieren diese mit den skalaren, diskreten Zahlentypen:

Byte ohne Vorzeichen	:	0...255 SHORTCARD
Byte mit Vorzeichen	:	-127...128 SHORTINT
Wort ohne Vorzeichen	:	0...65535 CARDINAL
Wort mit Vorzeichen	:	-32768...32767 INTEGER
Langwort ohne Vorzeichen	:	0...4294967295 LONGCARD
Langwort mit Vorzeichen	:	-2147483648...2147483647 LONGINT

Da es ziemlich aufwendig ist, zwischen dezimal und binär hin und her zu rechnen, es andererseits aber extrem unpraktisch ist, immer mit Binärzahlen zu rechnen, hat man ein weiteres Zahlensystem eingeführt,

[10] Findet kaum verwendung

das Hexadezimalsystem. In diesem System ist die Basis die 16. Die fehlenden Ziffern von 10 bis 15 werden durch die ersten sechs Buschtaben des Alphabeths gebildet. Da 2^4 genau 16 ist, werden immer vier Bits in einer Hexadezimalziffer zusammengefaßt.

Dez	Bin	Hex	Dez	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Binärzahlen lassen sich recht einfach in Hexzahlen umwandeln, und sind so einfach zu handhaben.

1 0 0 1	0 1 1 1	1 1 0 1	0 0 1 0
↓	↓	↓	↓
9	7	D	2

Cluster versteht alle diese Zahlenformate. Dezimalzahlen werden ohne Vorsatz geschrieben, z. B. 153, Binärzahlen mit einem Vorgesetzten Prozentzeichen %10011001 und Hexzahlen mit einem Dollarzeichen $99[11].

Nun werden Sie vielleicht einwenden, daß ein Computer doch auch Zeichen verarbeiten kann. Falsch, er tut nur so! Alle Zeichen (Buchstaben, Ziffern etc.) werden einfach durchnumeriert, so daß der Computer wieder seine geliebten Zahlen vor sich hat. Für diese Numerierung gibt

[11] Diese Symbole haben sich bei Assemblerprogrammierern durchgesetzt

es einen Quasistandard: ASCII[12], und seine europäische Erweiterung ISO 8859-1, welcher auch im Amiga verwendet wird[13].

Beim diesen Codes werden immer 8-Bits, also ein Byte, für ein Zeichen verwendet. ASCII verwendet die Codes von 32 bis 127 für die darstellbaren Zeichen; ISO verwendet zusätzlich noch die Werte von 240 bis 255 für Europäische Sonderzeichen. Die freien Werte werden mit Sondercodes wie Zeilenvorschub, Seitenvorschub oder Tab belegt.

Die Ziffern liegen in den Codes von 48 bis 57 („0" ist 48, „1" ist 49 und „9" ist 57), so daß es recht einfach ist, ein String aus Ziffern in eine Zahl umzuwandeln.

```
PROCEDURE Convert(VAR s : STRING):INTEGER;
VAR i,sum : INTEGER;
BEGIN
  sum:=0;
  FOR i:=0 TO s.len-1 DO
    sum:=sum*10+INTEGER(s.data[i])-INTEGER("0")
  END;
  RETURN sum
END Convert;
```

6.2.2 Der Speicher

Ein normaler Amiga 2000 hat standardmäßig etwa 8 Millionen Bits Schreib/Lese Speicher (RAM). Diese bilden nun nicht eine einzige Zahl, sondern sind auf eine Million Bytes aufgeteilt. Immer zwei dieser Bytes sind wieder zu einem Wort zusammengefaßt. Die Bytes werden nun wiederum durchnumeriert, und zwar von 0 aufsteigend. Diese Nummer nennt man Adresse. Ein 68000 kann 16 Millionen[14] verschiedene Adressen verwenden, ein 68030/40 noch ein vielfaches mehr. Außer für das RAM werden einige Adressen auch noch für andere Aufgaben benötigt, so daß ein Amiga 2000 mit 68000er nicht mehr als 8 MB verwalten kann. Ab einem 68020, sind der Speichergröße praktisch keine Grenzen mehr gesetzt[15].

[12] „American Standard Code for Information Interchange"

[13] Nur eine große Computerfirma mit blauen Rechnern hält sich nicht daran, und verwendet einen total chaotischen Code, der sich auf eine spezielle Eigenart von Lochkarten stützt. Soviel zum Thema „Kompatibilität".

[14] D. h. er könnte 16 Mega-Byte Speicher bzw. Erweiterungen verwalten, im Gegensatz zu einer anderen Rechnerfamilie, die noch immer nicht ohne großen Aufwand über 640KBytes hinauskommt.

Auf diesen Adressen liegen das ROM (nur lesbarer Speicher) mit einem Teil des Betriebssystems, sowie Bausteine zur Ein- Ausgabe. Diese Bausteine haben vielfältige Aufgaben zu erfüllen. Dazu brauchen sie viele Daten, und liefern auch eine Menge Information zurück. Dazu haben sie gemeinsame Speicherzellen mit dem Prozessor.

Der RAM-Speicher des Amiga gliedert sich in zwei Teile, dem Chip-Mem und dem Fast-Mem. Auf das Chip-Mem können auch andere Bausteine des Computers zugreifen, in ihm liegen zum Beispiel die Daten, die nötig sind um ein Bild auf dem Bildschirm erscheinen zu lassen. Das Fast-Mem steht allein dem Prozessor zur Verfügung, und heißt so, weil der Zugriff darauf nicht durch andere Systemteile gebremst wird.

Der Prozessor ist mit mehreren Leitungen mit seinem Speicher verbunden. Diese sind grob unterteilt Daten und Adressleitungen. Der 68000 hat 16 Daten und 23 Adressleitungen. Somit greift er immer zwei Bytes auf einmal zu. Dies hat zur Folge, daß Daten, die Größer sind als ein Byte, immer auf geraden Adressen liegen müssen.

6.2.3 Der Prozessor

Der Prozessor (im Amiga ein 680XXer von Motorola) ist das Herz des Computers. Der Prozessor arbeitet ein Programm ab, das in einem speziellen Code im Speicher abgelegt ist. Dieser Code besteht aus einfachen Befehlen und dazugehörigen Daten. Er ist für jeden Rechnertyp verschieden. Da er speziell auf den Prozessor zugeschnitten ist, ist er nicht leicht zu verstehen.

Die Sprache besteht immer aus einem Befehl, mit impliziten, bzw. nachgestellten Daten. Diese Befehle holt sich der Prozessor nacheinander aus dem Speicher und führt sie aus.

[15] Außer vielleicht durch den Geldbeutel.

Ein Prozessor besteht grob aus drei Teilen, einem kleinen Speicher (Register), einer Recheneinheit (`ALU`) und einem Teil, der die Befehle ausführt (Decoder).

Die Register gliedern sich wieder in zwei Gruppen, solche, die für den Programmierer da sind, und solche, die der Prozessor für seine Arbeit benötigt. Der Vorteil solcher Register ist, daß ein Zugriff auf sie wesentlich schneller ist, als ein Zugriff auf den normalen Speicher.

Für den Benutzer stehen acht Daten-, und sieben Adreßregister zur Verfügung. Die Datenregister haben die Namen `D0`, `D1`, `D2...D7` und sind je nach bedarf 8,16 oder 32 Bit lang. Die Adreßregister heißen `A0` bis `A6` und sind immer 32 Bit breit. Der Unterschied zwischen diesen Registern ist der, daß viele arithmetische Befehle wie Multiplizieren nur mit Datenregistern funktionieren. Dagegen sind Adreßregister dazu da, auf Dinge im Speicher zu zeigen, sie entsprechen somit Pointern.

Die wichtigsten drei Register des Prozessors sind der Programmzähler `PC`, der Stapelzeiger `SP` und das Statusregister `SR`.

Im `PC` steht immer die Adresse des nächsten Befehls, den der Prozessor ausführt. Schreibt man in dieses Register, führt der Prozessor seine Arbeit an einer anderen Stelle weiter, darum nennt man dies einen Sprung.

Der `SP` ist ein Zeiger auf einen Stack im Speicher. Dieser Stack wächst nach unten, der Adresse 0 zu. Auf diesem Stack werden Daten abgelegt, um sie später wieder zu holen, so z. B. die Rücksprungadressen bei Unterprogrammaufrufen.

Das `SR` Register ist mit einem Set vergleichbar, es enthält mehrere Flags, die den Zustand des Prozessors wiederspiegeln.

Die `ALU` ist der Teil des Prozessors, der für ihn Operationen ausführt, wie etwa den Inhalt zweier Register zu addieren, eine Zahl mit einem Register zu multiplizieren oder ähnliches. Die `ALU` setzt nach den meisten Operationen einige Flags im `SR`. War das Ergebnis z. B. null, so setzt sie das Zero-Flag, tritt ein Überlauf auf, das Overflow-Flag usw.

Der Decoder verwaltet `ALU` und Register. Nacheinander führt er für jeden Befehl folgende Tätigkeiten aus: Zuerst holt er aus dem Speicher den nächsten Befehl. Dann wird er dekodiert und schließlich ausgeführt. Nebenbei werden noch nötige Daten, die dem Befehl folgen, aus dem

Speicher geholt. Diese Aktionen werden mehrere Millionen mal pro Sekunde ausgeführt.

Dies alles wird durch einen externen Taktgenerator am laufen gehalten. Jeder Befehl braucht eine gewisse Zahl an Takten. Generell läßt sich sagen, daß ein Befehl mit mehr Daten aus dem Speicher länger braucht, als einer mit weniger. Die Addition zweier Registerinhalte braucht 8 Zyklen, die Addition einer Konstanten zu einer Speicherstelle dagegen bis zu 36 Zyklen[16]. Es ist also ratsam, Daten, die häufiger benötigt werden, in einem Register zu halten.

Ein 68000er hat mehrere Möglichkeiten, die Adresse seiner Daten zu ermitteln, sogenannte Adressierungsarten.

Register: Die Daten liegen in einem der Register im Prozessor

Direkt oder Immediate: Die Daten liegen direkt hinter dem Befehl, oder sind in diesen hineincodiert.

Absolut: Dem Befehl folgt die Adresse, in der die Daten stehen. Diese Adressierungsart ist im allgemeinen die langsamste, da immer zwei Worte für die Adresse zusätzlich gelesen werden müssen.

Indirekt: Die Adresse der Daten liegen in einem der Adreßregister, entspricht der Adressierung mit Pointern.

Indirekt mit Displacement: Die Adresse der Daten ist der Inhalt eines Adreßregisters plus einer Konstanten, entspricht der Adressierung eines Elements in einem Record.

Indirekt mit Index: Die Adresse der Daten ist die Summe eines Adreßregisters plus dem Inhalt eines Datenregisters, entspricht der Adressierung in einem Record.

Postincrement: Die Adresse der Daten ist der Inhalt eines Adreßregisters. Nach der Operation wird der Inhalt dieses Adreßregisters um die Länge des Operators erhöht.

[16] bei einem 68000

Predekrement: Die Adresse der Daten ist der Inhalt eines Adreßregisters. Vor der Operation wird der Inhalt dieses Adreßregisters um die Länge des Operators vermindert.

PC-Relativ: Die Adresse ist die Summe aus dem Inhalt des PC und einer Konstanten. Diese Adressierung ist schneller, als die Absolute, da immer nur ein Wort für die Konstante verwendet wird. Diese Adressierung findet bei Sprüngen und Konstanten Anwendung.

6.2.4 Assembler

Da die Darstellung der Befehle in Maschinencode fast nicht zu lesen sind, gibt man ihnen, und den zugehörigen Adressierungsarten Namen bzw. Schlüsselzeichen. Die folgende kleine Einführung in 68000er Assembler soll Ihnen das Verstehen der in diesem Kapitel gezeigten Assemblersequenzen erleichtern.

Eine Assembleranweisung besteht immer aus einem Befehl und keinem, einem oder zwei Operanden. Hat der Befehl keine Operanden, so sind diese meist im Befehl enthalten.

Man unterscheidet Quell- und Zieloperanden. In den Zieloperanden wird das Ergebnis der Operation geschrieben. Er steht bei zwei Operanden immer rechts.
Beispiel:

```
ADD.W D0,D1

ADD-------------Befehlsnamen (Mnemonic)
   .W----------- Längenkennzeichen, B=Byte,W=Wort,L=Langwort
      D0-------- Quelloperand
         D1----- Zieloperand
```

Diese Anweisung bildet die Summe aus dem Inhalt von `D0` und `D1`, und speichert das Ergebnis nach `D1`. Diese Operation hat Wortbreite. Bei Befehlen mit nur einem Operand ist meist klar, ob es sich um Ziel (Destination) oder Quelle (Source) handelt.

```
NEG.L  D0
```


Diese Operation hat nur einen Zieloperanden, nämlich D0.

Adressierungsarten werden durch Spezielle Symbole gekennzeichnet:

```
D0          Register
(A0)        Indirekt
124(A0)     Indirekt mit Displacement
10(A0,D0)   Indirekt mit Index (und Displacement)
124         Absolut
124(PC)     PC-Relativ
(A0)+       Postincrement
-(A0)       Predecrement
```

Die Flags des `SR` werden als BOOLEAN-Variablen aufgefaßt, und haben die Namen:

- Z : Zero , zeigt Gleichheit oder Null an
- N : Negativ
- C : (Carry) Übertrag, für Vorzeichenlose Zahlen, d. h. bei einer Operation hatte das Ergebnis mehr Bits, als der Zieloperand groß war.
- V : Überlauf, für Vorzeichenbehaftete Zahlen

Diese Bits werden im allgemeinen durch die vorherige Operation nach deren Ergebnis gesetzt.

6.2.4.1 Operationen für beliebige Daten:

```
MOVE    <src>,<dst>    dst:=src
```

6.2.4.2 Operationen für Zahlen

```
ADD     <src>,<dst>    dst:=dst+src
SUB     <src>,<dst>    dst:=dst-src
```

```
MUL     <src>,<dst>   dst:=dst*src
DIV     <src>,<dst>   dst:=dst DIV src
NEG     <dst>         dst:=-dst
ASL     <src>,<dst>   dst:=dst SHL src
ASR     <src>,<dst>   dst:=dst SHR src (Vorzeichenbehaftet)
LSR     <src>,<dst>   dst:=dst SHR src (Vorzeichenlos)
EXT.W   <dst>         dst:=INTEGER(SHORTINT(dst))
EXT.L   <dst>         dst:=LONGINT(INTEGER(dst))
```

6.2.4.3 Operationen für Mengen

```
OR      <src>,<dst>   dst:=dst+src
AND     <src>,<dst>   dst:=dst*src
EOR     <src>,<dst>   dst:=dst/src
NOT     <dst>         dst:=-dst
BCLR    <src>,<dst>   EXCL(dst,src)
BSET    <src>,<dst>   INCL(dst,src)
BCHG    <src>,<dst>   FLIP(dst,src)
BTST    <src>,<dst>   Z:=NOT (src IN dst) (Z = Zero-Flag)
```

6.2.4.4 Sprünge

Anweisungen mit Sprüngen können nicht ohne weiteres in Cluster nachgebildet werden, da höhere Programmiersprachen bewußt auf Sprünge verzichten. Allerdings werden viele Clusterkonstrukte in Sprünge umgewandelt. Das Ziel eines Sprungs ist mit einem Namen (Label) bezeichnet, der früher oder später mit einem nachfolgenden Doppelpunkt am Ziel steht (Beispiele folgen).

`JMP,BRA`	`<dst>`	Setzen die Ausführung bei `dst` weiter
`Bcc`	`<dst>`	Führt den Sprung nur dann aus, wenn die Bedingung `cc` erfüllt ist.

Folgende Bedingungen sind für `cc` immer möglich:

```
CS := C IN SR
CC := NOT (C IN SR)
EQ := Z IN SR
NE := NOT (Z IN SR)
VS := V IN SR
VC := NOT (V IN SR)
```

und nach einem Vergleich,

```
CMP  <src>,<dst>

EQ := src=dst (Z IN SR)
NE := src#dst (NOT (Z IN SR))
```

Vorzeichenlos:

```
HI := dst>src
LS := dst<=src
CC := dst>=src
CS := dst<src
```

Mit Vorzeichen

```
GT := dst>src
LE := dst<=src
GE := dst>=src
LT := dst<src
```

```
Scc  <dst>    dst:=cc
```

Entspricht der Zuweisung an eine Booleanvariable. Die Bedeutung von `cc` ist wie oben bei `Bcc`.

Noch ein Beispiel zum Schluß, das Ermitteln der größeren zweier Zahlen. Die Zahlen befinden sich zu Begin in `D0` und `D1`, die größere von beiden soll nachher in `D1` sein.

```
IF D1>D0 THEN D0:=D1 END

CMP.W   D0,D1   ;D0 mit D1 vergleichen
BLE     end     ;wenn nicht D1>D0, dann nächsten Befehl
                ;überspringen
MOVE.W  D1,D0   ;D0:=D1
end:            ;nach hier wird gesprungen
```

Neben diesen Sprüngen, die hauptsächlich für Schleifen und `IF..THEN`-Verzweigungen verwendet werden, existiert noch ein Sprung, der beim Aufruf einer Prozedur verwendet wird. Er bietet eine Besonderheit, nämlich daß es eine einfache Methode gibt, das Programm am Ende der Prozedur wieder an der Stelle fortzusetzen, an der die Prozedur aufgerufen wurde:

`JSR`	`<dst>`	Legt den Inhalt des `PC` auf den Stapel und springt nach `dst`, Unterprogrammaufruf
`RTS`		Holt den `PC` wieder vom Stapel, Rücksprung.

Sie wissen zwar wahrscheinlich alle was ein Stapel ist, für den Fall, daß es Ihnen gerade entfallen ist, hier noch einmal die wichtigste Eigenschaft: Das Element, daß als letztes daraufgelegt wird, bekommt man auch als erstes wieder zurück. Mit den folgenden beiden Befehlen läßt sich auf den Stapel (Stack) Platz für lokale Variablen verwalten.

`LINK`	`Ax,<src>`	Erzeugt auf dem Stapel einen freien Bereich von der Größe `src` in Bytes, der dann über das Adreßregister Ax angesprochen werden kann.
`UNLK`	`Ax`	Macht den Vorgang wieder rückgängig.

Mit dem Stapelzeiger des Prozessors kann wie mit einem normalen Adreßregister adressiert werden. Der Stapel wird mit Hilfe des Postincrement und Predecrement realisiert.
Beisp: Pushen und Poppen eines Worts in `D0`

`MOVE.W D0,-(SP)`	Auf dem Stapel wird Platz geschaffen, d. h. der `SP` wird nach unten verschoben, und dann der Inhalt von `D0` eingetragen.
`MOVE.W (SP)+,D0`	Der letzte Wert des Stapels wird in `D0` gebracht. Danach wird der Platz wieder freigegeben.

Auf das aktuelle Element kann immer mit `(SP)`, und auf frühere Elemente mit `d(SP)` zugegriffen werden. Dabei ist zu beachten, das der Stapel nach unten wächst, also das letzte Element das niedrigste Displacement besitzt. Außerdem ist beim berechnen des Displacements zu beachten, daß auf dem Stack Elemente verschiedenster Größe liegen können.

Ein Problem ist, daß nicht bei jedem Befehl alle Adressierungsarten als Quelle und Ziel kombiniert werden können. So ist es bei den Arithmetischen Operationen nötig, daß mindestens ein Operand ein Register oder eine Konstante sein muß. Es gibt auch spezielle Versionen einiger Befehle mit Konstanten, die schneller arbeiten.

`MOVEQ`	`#Imm,<dst>`	Bringt eine Konstante im Bereich von -128 bis 127 als Langwort nach `dst`.
`ADDQ.x`	`#Imm,<dst>`	Addiert eine Konstante im Bereich von 1 bis 8 zu dem Wert in `dst`.
`SUBQ.x`	`#Imm,<dst>`	Zieht eine Konstante im Bereich von 1 bis 8 von dem Wert in `dst` ab.

6.3 Maschinennahe Techniken in Cluster

Wie schon Anfangs beschrieben, kann selbst ein guter Compiler nicht immer optimalen Code liefern. Für Cluster besteht noch ein zusätzliches Handicap, da er nur ein Singlepass-Compiler ist. Er kann folglich nicht sehen, was für ein Befehl als nächstes kommt, was für die Codegenerierung manchmal sehr wichtig wäre. Es gibt jedoch eine Fülle von Möglichkeiten, den Compiler bei der Codegenerierung zu unterstützen.

Dazu ist es aber notwendig zu wissen, wie Cluster die einzelnen Strukuren und Daten in Assembler umsetzt. Cluster bietet viele Möglichkeiten, maschinennahes Programmieren zu unterstützen. Wenn man diese anzuwenden weiß, erhält man in 99 von 100 Fällen ein Ergebnis, daß man auch in Assembler nicht mehr verbessern könnte.

6.3.1 Repräsentierung von Clustertypen in Bits & Bytes

Einfache Typen haben meist eine direkte Repräsentierung in Maschinengrößen.

`BOOLEAN, CHAR, SHORTINT, SHORTCARD`	: Byte
`CARDINAL, INTEGER`	: Wort
`LONGCARD, LONGINT, POINTER, REAL`	: Langwort
`LONGREAL`	: Doppelwort

Die Konstanten `TRUE` und `FALSE` zur Representierung von boolschen Werten haben die Länge Byte und folgende interne Werte:

`FALSE = 0,TRUE = 255` oder `-1`, je nach Betrachtung.

Aufzählungstypen sind je nach Anzahl ihrer Elemente als Byte, Wort oder Langwort repräsentiert:

1	bis	255 Elemente	: Byte
256	bis	65535 Elemente	: Wort
	sonst		: Langwort

Auch Sets werden je nach Anzahl der Elemente in einem Byte, Wort oder Langwort gehalten.

1 bis	8 Elemente	: Byte
9 bis	16 Elemete	: Wort
17 bis	32 Elemente	: Langwort

Die einzelnen Elemente des SETs entsprechen den Bits im Byte, Wort oder Langwort. Das erste Element ist immer das niederwertigste Bit.

Beisp:

```
set = SET OF [0..7];

INCL(set,4);
```

wird zu:

```
BSET    #4,set
```

Sie sehen, mittels Sets lassen sich die einzelnen Bits einer Speicherstelle manipulieren. Da die Bitbefehle bei Daten im Speicher immer ein Byte als Größe annehmen, wird bei längeren SETs die Operation in einem Datenregister vorgenommen.

Variablen, die länger als ein Byte sind, werden immer an geraden Adressen abgelegt.

Bei einem Record liegen die Elemente hintereinander im Speicher. Ihre Position wird durch ein Displacement (Verschiebewert) vom Anfang des Records bezeichnet.

Beisp:

```
Struc = RECORD
          x,y : INTEGER;
          c   : CHAR;
          p   : POINTER TO INTEGER;
        END;
```

```
x : Wort, Struc+0
y : Wort, Struc+2
c : Byte, Struc+4
p : Langwort, Struc+6
```

Bei varianten Records haben mehrere Elemte das selbe Displacement. Dies dient dazu Speicher zu sparen, indem Felder, die nie gleichzeitig belegt sein können, die selben Speicherteile nutzen.

Beisp:

```
Vector : RECORD
           IF KEY polar : BOOLEAN
             OF FALSE THEN
               x,y : REAL
             OF TRUE THEN
               rad,phi : REAL
           END
         END
```

```
polar : Byte,     Struc+0
x     : Langwort, Struc+2
y     : Langwort, Struc+6
rad   : Langwort, Struc+2
phi   : Langwort, Struc+6
```

Vorsicht: Es findet kein Check statt, in welcher Weise der Record gerade benutzt wird, es ist also problemlos möglich, erst in `Vector.y` und dann in `Vector.phi` etwas zu schreiben. Nur darf man sich dann nicht wundern, wenn in `Vector.y` nicht mehr das steht, was man zuvor dort hineingeschrieben hat.

In einem Array liegen die Elemente ebenfalls direkt hintereindander. Bei mehrdimensionalen Feldern kann dies als Feld von Feldern betrachtet werden.

```
Mat = ARRAY [0..2],[0..2] OF INTEGER;
```

```
Mat[0,0] : Wort, Mat[0]+0=Mat+0
Mat[0,1] : Wort, Mat[0]+2=Mat+2
Mat[0,2] : Wort, Mat[0]+4=Mat+4
Mat[1,0] : Wort, Mat[1]+0=Mat+6
..
Mat[2,2] : Wort, Mat[2]+4=Mat+16
```


Pointer enthalten die Adresse eines Elements im Speicher.

Beisp:

```
VAR p : POINTER TO INTEGER;
    i : INTEGER;

BEGIN
  i:=p^;
```

wird zu

```
MOVE.L  p,A0    Adresse von p^ nach A0
MOVE.W  (A0),i  Inhalt von p^ nach i
```

CLASSPTR sind sechs Bytes lang. Sie tragen außer der Adresse eines Objekts auch noch dessen zusätzlichen Daten, wie dessen Länge o. ä.

Beisp:

```
VAR p : CLASSPTR TO ARRAY OF CHAR;


p^[p^'MAX]:="A"
```

```
MOVE.L  p,A0        Adresse des Arrays nach A0
MOVE.W  p+4,D7      Anzahl der Elemente des Arrays nach D7
SUB.W   #1,D7       Anzahl um eins vermindern, um auf den
                    Index des letzten Elements zu kommen
MOVE.B  #65,(A0,D7)und "A" in Array eintragen.
```

Die globalen Variablen eines Moduls werden immer relativ zu `A4` (Modulebase) adressiert. Die Reihenfolge in der die Variablen liegen, stimmt nicht mit der Reihenfolge überein, in der sie deklariert wurden, da der selektive Linker nur benutzte Variablen erzeugt. Konstanten und Prozeduren im eigenen Modul werden relativ zum `PC` adressiert.

Variablen, Konstanten und Prozeduren fremder Module werden absolut angesprochen.

Lokale Variablen einer Prozedur werden auf dem Stapel erzeugt, und über A5 (Localbase) adressiert.

Beisp:

```
PROCEDURE x(a : INTEGER;b : LONGCARD);
VAR i : INTEGER;
    j : LONGCARD;
```

wird zu

```
SUB.W #8,A7   schafft Platz für 8 Bytes Variablen

a'PTR = 16(A5)
b'PTR = 12(A5)
i'PTR = (A7)
j'PTR = 4(A7)
```

6.3.2 Repräsentierung von Clusterstrukturen in Assembler

Ausdrücke werden von Cluster mit Punkt vor Strich umgesetzt, so daß eine total veränderte Struktur in Assembler entstehen kann.
Beisp:

```
VAR a,i,j,k : INTEGER;
    p       : POINTER TO INTEGER;

a:=i*j+k*(i+p^*j)
```

wird zu:

```
MOVE.W  i(A4),D7      Wert von i nach D7
MUL.W   j(A4),D7      i*j nach D7
```



```
MOVE.L  p(A4),A3      Adresse von p^ holen
MOVE.W  j(A4),D6      Wert von j nach D6
MUL.W   (A3),D6       p^*j nach D6
ADD.W   i(A4),D6      p^*j+i nach D6
MUL.W   k(A4),D6      k*(p^*j+i) nach D6
ADD.W   D6,D7         i*j+k*(p^*j+i) nach D7
MOVE.W  D7,a(A4)      und D7 nach a speichern.
```

Boolean Ausdrücke werden in einem speziellen Verfahren ausgewertet. Hierbei wird **AND** oder **OR** nicht wirklich ausgeführt, sondern ist in der Verzweigungsstruktur enthalten. Dabei wird davon ausgegangen, daß z. B., wenn bei einem **AND** schon daß erste Element FALSE ist, kein weiteres mehr geprüft werden muß.

Beisp:

```
VAR i,j : INTEGER;

IF (i>1) AND (i<10)OR (i=-1) AND (j>2)THEN
  ...
ELSE
  ...
END
```

wird zu:

```
CMP.W   #1,i(A4)
BLE     a1        wenn i<=1, dann kann erster AND nicht
                  erfüllt sein, also gleich zweiten prüfen
CMP.W   #10,i(A4)
BLT     then      ist i auch kleiner 10, dann ist die OR
                  Bedingung erfüllt, und der gesamte
                  Ausdruck wahr, also gleich zu then
a1:
CMP.W   #-1,i(A4)
BNE     else      ist i#-1, kann auch der zweite AND nicht
```

```
                     erfüllt werden, also gleich nach else
CMP.W    #2,j(A4)
BLE      else        der zweite Teil vom AND ist nicht erfüllt,
                     also zum else
...
BRA      end         ELSE-Teil überspringen
else:
...
end:
```

An diesem Beispiel wird auch gleich deutlich, wie ein **IF** in Assembler realisiert wird. **OR_IFs** erscheinen als weitere Überprüfung vor dem `else` Fall.
So sieht eine **WHILE**-Struktur aus:

```
VAR i : INTEGER;

...
WHILE i>1 DO
  ...
END

while:
CMP.W    #1,i
BLE      end      ist die Schleifenbedingung nicht erfüllt,
                  dann Schleife verlassen
...
BRA      while    und zurück zum Anfang
end:
```

und so ein **REPEAT UNTIL**:

```
REPEAT
  ...
UNTIL i=5;
```



```
repeat:
...
CMP.W    #5,i
BNE      repeat   wenn noch nicht erfüllt,
                  dann zurück an Anfang
```

FOR-Schleifen sind dann etwas schwieriger, wenn die Grenzen keine Konstanten sind, da sich die Ausdrücke ändern könnten, die Schleifengrenzen dies aber nicht dürfen. Darum wird bei variablem Ende dieses auf dem Stapel gesichert.

```
VAR i,j : INTEGER;

FOR i:=1 TO 10 DO
...
END

MOVE.W   #1,i     i:=1
for:
...
ADD.W    #1,i     i am Schleifenende erhöhen
CMP.W    #10,i    auf Ende prüfen
BLE      for:

FOR i:=j TO j+10 DO
...
END

MOVE.W   j,i          i:=j
MOVE.W   j,D7
ADD.W    #10,D7
MOVE.W   D7,-(A7)     TO j+10
BRA      end:         weil schon zu Beginn geprüft werden muß
for:
...
end:
```

```
ADD.W   #1,i          i am Schleifenende erhöhen
CMP.W   (A7),D7       mit dem Zielwert auf dem Stapel
                      vergleichen
BLE     for
ADD.W   #2,A7         Stack wieder bereinigen
```

Es ist also immer sinnvoll, eine **FOR**-Schleife auf eine Konstante hinlaufen zu lassen.

Bei einem Prozeduraufruf werden nacheinander die Parameter, sofern vorhanden, auf den Stapel gebracht, bevor die Prozedur mit `JSR` aufgerufen wird.

```
PROCEDURE a(x,y : INTEGER;VAR a : STRING)
...
END a;

VAR i : INTEGER;
    s : STRING(32);


...
  a(i,2,s);
...
```

wird zu:

```
MOVE.W  i(A4),-(SP)   Wert von i auf den Stapel
MOVE.W  #2,-(SP)      Konstante 2 auf den Stapel
MOVE.W  #32,-(SP)     Maximallänge von s auf den Stapel
PEA     s(A4)         Adresse von s auf den Stapel bringen.
JSR     a(PC)         Prozedur a als Unterprogramm anspringen
```


6.4 Optimierungen

Obwohl der Compiler versucht, möglichst guten Maschinen-Code aus dem Quelltext zu erzeugen, gelingt dies nicht immer optimal, da bei Cluster ein Kompromiß zwischen Compiliergeschwindigkeit und Code-Qualität eingegangen wurde. Aus diesem Grund wurde Cluster als Singlepass-Compiler[17] implementiert. Dies hat jedoch zur Folge, daß er nicht vorhersehen kann, welche Art von Anweisung als nächstes kommt. Dadurch schränkt sich allerdings die Zahl an automatischen Optimierungen etwas ein. Dennoch brauchen Sie nun nicht befürchten, daß Ihre Programme langsamer als die eines Mehrpass-Compilers sein müssen. Wenn man dem Compiler nämlich etwas unter die Arme greift, ist er in der Lage ganz hervorragende Programme zu erzeugen.
Achtung: Wir sind uns bewußt, daß vielen Informatikern die Haare zu Berge stehen werden, wenn Sie sehen, wie maschinennah manche dieser Optimierungen sind, daher sei hier noch einmal darauf hingewiesen, daß man derartige Routinen in spezielle Module kapseln soll. In einem Hauptmodul bitte nur in extrem zeitkritischen Fällen verwenden.

Damit ein Prozessor arbeiten kann, muß er von außen mit einem Takt gesteuert werden. Jeder Takt bewirkt einen Arbeitsschritt im Prozessor. Nun benötigt der Prozessor für jeden Befehl eine bestimmte Anzahl von Takten (Zyklen) und damit eine bestimmte Zeitspanne, um diesen auszuführen.

Um nun die Wirkungsweise und Effiziens der verschiedenen Optimierungen zu zeigen, werden wir immer den erzeugten Maschinencode vor und nach der Optimierung zeigen und die jeweils nötige Zyklenzahl[18] angeben.

6.4.1 Verwendung von Standardprozeduren

Standardprozeduren in Cluster sind im allgemeinen nicht als wirkliche Prozeduren realisiert, sondern werden als Assemblerbefehle ins Programm eingebaut. Mit den Prozeduren `INC`, `DEC`, `UNI`, `SEC`, `SHL` und `SHR` läßt

[17] Der Compiler bearbeitet den Quelltext in einem Durchlauf
[18] Die Zyklenzahlen beziehen sich hierbei auf einen `MC68000`

sich ein Laufzeitgewinn erzielen, da der Compiler weis, daß Quelle und Ziel der Operation identisch sind.

Beispiel:

```
VAR i : INTEGER;
...
  i:=i+2;
...
```

wird zu

```
MOVE.W  i(A4),D7        12 Zyklen
ADD.W   #2,D7            4 Zyklen
MOVE.W  D7,i(A4)        12 Zyklen
```

also 28 Zyklen

dagegen wird

```
...
INC(i,2)
...
```

zu

```
ADD.W   #2,i(A4)     16 Zyklen.
```

Selbiges gilt auch für die anderen der genannten Standardprozeduren. Selbst die Anweisung `i:=i*2` lässt sich optimieren zu: `INC(i,i)`.

6.4.2 Abfrage der Statusregister

Bei den meisten Operationen, die der Prozessor ausführt, setzt er hinterher das `SR`-Register. Diese Flags werden jedoch kaum beachtet. Selbst bei einem Vergleich wird meist nur ein Zustand erfragt. In Cluster ist

es möglich, den Zustand des `SR`-Registers zur Programmsteuerung einzusetzen. Dazu kann man einzelne Zustände abfragen. Dies geschieht durch die bereits bekannten relationalen Operatoren, die hier als eine Art Funktion eingesetzt werden, die einen Boolean-Wert zurückgeben.

- `=` : letzter Vergleich war gleich, oder Ergebnis war 0
- `#` : letzter Vergleich war ungleich, oder Ergebnis nicht 0
- `>` : letzter Vergleich war größer, oder Ergebnis größer 0
- `>=` : letzter Vergleich war größer/gleich, oder Ergebnis größer/gleich 0
- `<=` : letzter Vergleich war kleiner/gleich, oder Ergebnis kleiner/gleich 0
- `<` : letzter Vergleich war kleiner, oder Ergebnis kleiner 0
- `+` : Carry gesetzt, d. h. bei der letzten Operation gab es einen Überlauf.

Beispiel, suchen nach 0 in einem Array:

```
i:=100;
REPEAT
  DEC(i)
UNTIL < OR a[i]=0;
```

Mehrfacher Vergleich:

```
IF a>b THEN
  (* a ist größer b *)
OR_IF # THEN
  (* a ist kleiner b *)
ELSE
  (* a ist gleich b *)
END
```

Diese direkte Abfrage spart eine Menge Zeit, da ein erneuter Vergleich vermieden wird.

6.4.3 Registervariablen

Da der Prozessor wesentlich schneller (und auch mit kürzerem Code) auf seine Register als auf den Speicher zugreifen kann, empfiehlt es sich, einen Wert, der öfters benötigt wird, in einem der zahlreichen Register zu halten. Problematisch ist dies, wenn eine Prozedur aufgerufen wird, da diese ohne Rücksicht auf die Register des Aufrufers arbeitet. Die Register müssen also vor einem Prozeduraufruf gesichert werden.

Cluster bietet die Möglichkeit, Variablen für einen Bereich des Programms in Register zu legen. Der Compiler kümmert sich auch um das Sichern und Wiederholen von Registerinhalten, vor und nach Prozeduraufrufen. Der Geschwindigkeitsgewinn ist aber natürlich um so geringer, um so mehr Prozeduren in diesem Bereich aufgerufen werden.

Obwohl der Compiler auch von sich aus bemüht ist, immer möglichst viele Variablen in Registern zu halten, kann er nicht wissen, welche davon am meisten benötigt werden. Daher muß man ihm sagen, wenn bestimmte Variablen auf alle Fälle in Registern gehalten werden sollen. Die Struktur, die dem Compiler mitteilt, welche Variablen er verwenden soll, und in welchem Bereich, ist die **WITH** Struktur. Mit ihr können nicht nur komplexe Zugriffsstrukturen vereinfacht werden, sondern auch einfache Variablen in Register gelegt werden. Bei der Vereinfachung einer Zugriffsstruktur belegt der Compiler auch ein Register, ein Adreßregister, in dem die Adresse des Objekts gespeichert wird.

```
VAR i : INTEGER;

...
  FOR i:=1 TO 10000 DO ... END
...
```

wird zu:

```
MOVE.W  #1,i(A4)        16 Zyklen
for:
...
ADD.W   #1,i(A4)        16 Zyklen
```



```
CMP.W   #10000,i(A4)    24 Zyklen
BLE.S   for     10 Zyklen
```

ergibt 10000*(16+24+10)+16=500016 Zyklen, also 0,07 Sekunden[19]

```
...
  WITH i DO
    FOR i:=1 TO 10000 DO ... END
  END
...
```

wird zu:

```
MOVE.W  i(A4),D7        12 Zyklen
MOVE.W  #1,D7            4 Zyklen
for:
...
ADD.W   #1,D7            4 Zyklen
CMP.W   #10000,i(A4)     8 Zyklen
BLE.S   for             10 Zyklen
MOVE.W  D7,i(A4)        12 Zyklen
```

ergibt 10000*(4+8+10)+12+4+12=220028 Zyklen, also 0,03 Sekunden, eine Verbesserung der Laufzeit, um über 50%. Noch besser wird die Schleife, wenn man sie gegen 0 laufen läßt, dann tritt der spezielle Befehl `DBRA` in Aktion. Dieser Befehl macht drei Dinge auf einmal: Er veringert den Wert in dem dahinter angegebenen Register, vergleicht, ob er noch größer/gleich 0 ist und springt, wenn dem so ist an die angegebene Sprungmarke.

```
...
  WITH i DO
    FOR i:=9999 TO 0 BY -1 DO ... END
  END
...
```

[19] Alle Zeitangaben beziehen sich auf einen Amiga mit 68000er und 7,14 MHz

wird zu:

```
MOVE.W  i(A4),D7        12 Zyklen
MOVE.W  #9999,D7         8 Zyklen
for:
...
DBRA    D0,for          10 Zyklen
MOVE.W  D7,i(A4)        12 Zyklen
```

ergibt 10000*10+12+8+12=100032 Zyklen, also 0,014 Sekunden, und somit eine Verbesserung der Laufzeit um 80%.

Bei genügend freien Registern wird eine **FOR**-Schleife, die nicht gegen Null zählt, aber eine Registervariable als Zähler hat, ebenfalls mit einem `DBRA` implementiert.

Es ist auch möglich, anstatt eine vorhandene Variable in ein Register zu legen, auch eine bisher noch nicht bekannte Variable in einem Register zu erzeugen. Diese existiert nur innerhalb der WITH-Struktur. Anstatt des Variablennamens muß hierbei lediglich der Typnamen angegeben werden, ein **AS** ist allerdings erforderlich. Handelt es sich um einen einfachen Typ, wird dieser in ein Register gelegt, bei einem komplexen dagegen, wird er als lokale Variable angelegt.

```
WITH INTEGER AS i,
     INTEGER AS j DO
  ...
END
```

Auf diese Art erzeugte Variablen sind natürlich nicht initialisiert. Bei einfachen Typen belegt der Compiler eines der acht Datenregister, bei einem Zeiger oder einer komplexeren Variablen belegt der Compiler eines der fünf freien Adreßregister. Diese Art von Variablen eignet sich hervorragend als Zähler für Schleifen. Der Compiler meldet, wenn ihm die Register ausgehen.[20]

[20]Da es auch möglich ist, den Typ der zuWithenden Variablen über den ganzen Block zu ändern, kann man mit einem Trick auch mehr Registervariablen erreichen. Dazu benötigt man allerdings eine Registervariable, deren Inhalt nicht mehr nötig ist. Man witht sie dann einfach mit dem nun benötigten Typen neu. Diese Technik ist allerdings problembehaftet und fehlerträchtig und sollte nur in absoluten Notfällen verwendet werden, da danach nicht mehr auf die ursprüngliche Variable zugegriffen werden darf, der dies aber nicht merkt. Daß das dabei entstehende Programmstück besonders gut getestet wird, versteht sich von selbst.

Eine weitere Besonderheit der **WITH**-Anweisung, man kann innerhalb eines bestimmten Bereiches den Typ einer Variable ändern:

```
VAR
  ap   : ANYPTR;
...

WITH CharPtr(ap) AS p DO
  ...
END
```

6.4.4 Registerparameter

In Assembler ist es üblich, Parameter für eine Prozedur in Registern zu übergeben. Dies ist in Cluster ebenfalls möglich. Derartige Variablen verhalten sich in der Prozedur so, als wären sie von Beginn bis zum Ende durch **WITH** in ein Register gelegt worden. Es ist allerdings nötig, daß man hierbei die Nummer des zu verwendenden Registers angibt. Es empfiehlt sich hierbei den im Modul `System` definierten Aufzählungstypen `Regs` zu verwenden, denn `A0` ist verständlicher als `8`.

```
PROCEDURE Add(i IN D2,j IN D3 : INTEGER):INTEGER;
BEGIN
  RETURN i+j
END Add;
```

Die Datenregister haben die Nummern `D0` bis `D7`, die freien Adreßregister die Nummern `A0` bis `A3`. Verwendet sollten die Datenregister in der Reihenfolge: D2,D3,D4,D5,D6,D1,D7,D0 und die Adreßregister A0,A1,A2,A3. Zeiger und **VAR/REF**-Parameter sollten in Adreß- und einfache Typen in Datenregistern übergeben werden. Komplexe Typen kann man nur als **VAR/REF**-Parameter in Registern übergeben.

```
PROCEDURE Add(i,j : INTEGER):INTEGER;
BEGIN
  RETURN i+j
END Add;
```

wird zu

```
MOVE.W  i(A7),D7  12 Zyklen
ADD.W   j(A7),D7  12 Zyklen
MOVE.W  D7,D0      4 Zyklen
MOVE.L  (SP)+,A0  12 Zyklen
ADD     #4,SP      8 Zyklen
JMP     (A0)       8 Zyklen
```

und der Aufruf

```
...
  a:=Add(b,c)
...
```

```
MOVE.W  a(A4),-(SP)     16 Zyklen
MOVE.W  b(A4),-(SP)     16 Zyklen
JSR     Add(PC)         18 Zyklen
MOVE.W  D0,c(A4)        12 Zyklen
```

macht zusammen 118 Zyklen.

Mit Registerparametern:


```
SUBQ.W  #4,A7   4 Zyklen
MOVE.W  D2,D7   4 Zyklen
ADD.W   D3,D7   4 Zyklen
MOVE.W  D7,D0   4 Zyklen
ADDQ.W  #4,A7   4 Zyklen
RTS             16 Zyklen
```

```
MOVE.W  b(A4),D2        12 Zyklen
MOVE.W  c(A4),D3        12 Zyklen
JSR     Add(PC)         18 Zyklen
MOVE.W  D0,c(A4)        12 Zyklen
```

ergibt zusammen nur 90 Zyklen, also 38 Zyklen Einsparung.

Es ist immer zu bedenken, daß Prozeduraufrufe innerhalb einer Prozedur mit Registerparametern den Vorteil der Registerparameter stark mindern.

6.4.5 Postincrement/Predecrement

Der 68000er verfügt über zwei sehr interessante Adressierungsarten, die bei keinem mir bekannten Compiler unterstützt werden, Postincrement und Predecrement. Um diese mächtigen Adressierungsarten benützen zu können, ohne Assembler zu verwenden, existieren neben der normalen Dereferenzierung mit „^“ noch „+^“ für den Postincrement und „-^“ für den Predecrement. Diese Dereferenzierungsoperatoren funktionieren allerdings nur mit Variablen in Adressregistern, und auch da nicht in jeder Konstellation[21].

```
VAR a : ARRAY [0..49] OF INTEGER;
    i : INTEGER;

...
  FOR i:=0 TO 48 DO
    a[i]:=a[i+1]
```

[21] Keine Angst, der Compiler meldet den Fehler

```
  END
...
```

wird zu

```
MOVE.W  #0,i(A4)             16 Zyklen
for:
MOVE.W  i(A4),D7             12 Zyklen
ADD.W   D7,D7                 4 Zyklen
MOVE.W  i(A4),D6             12 Zyklen
ADD.W   D6,D6                 4 Zyklen
MOVE.W  a+2(A4,D6),a(A4,D7) 24 Zyklen
ADD.W   #1,i(A4)             16 Zyklen
CMP.W   #48,i(A4)            16 Zyklen
BLE.S   for                  10 Zyklen
```

Sind 49*(12+4+12+4+24+16+16+10)+16=4818 Zyklen

Und total optimiert:

```
TYPE
  IntPtr = POINTER TO INTEGER;

...
  (* $W- *)  | wird später erläutert
  WITH INTEGER AS i,
       IntPtr  AS src,
       IntPtr  AS dst  DO
    src:=a[1]'PTR;
    dst:=a[0]'PTR;
    FOR i:=48 TO 0 BY -1 DO
      dst+^:=src+^
    END;
  END;
```

wird zu


```
LEA     a+2(A4),A3      8 Zyklen
LEA     a(A4),A2        8 Zyklen
MOVE.W  #48,D7          4 Zyklen
for:
MOVE.W  (A3)+,(A2)+     12 Zyklen
DBRA    D7,for          10 Zyklen
```

Macht 48*(12+10)+8+8+4=1076 Zyklen, also weniger als ein viertel der nichtoptimierten Version.

Ein weiteres Beispiel: Kopieren bis zu einem Zeichen mit Code 0:

```
TYPE
  CharPtr = POINTER TO CHAR;

PROCEDURE Copy(from,to : CharPtr);
BEGIN
  DEC(from);DEC(to);
  REPEAT
    INC(from);INC(to)
    to^:=from^;
  UNTIL from^=&0;
END Copy;
```

wird zu

```
SUB.L   #1,from(A7)     24 Zyklen
SUB.L   #1,to(A7)       24 Zyklen
repeat:
ADD.L   #1,from(A7)     24 Zyklen
ADD.L   #1,to(A7)       24 Zyklen
MOVE.L  from(A7),A3     16 Zyklen
MOVE.L  to(A7),A2       16 Zyklen
MOVE.B  (A3),(A2)       12 Zyklen
MOVE.L  from(A7),A3     16 Zyklen
TST.B   (A3)             8 Zyklen
```

```
BNE.S   repeat          10 Zyklen
MOVE.L  (SP)+,A0        12 Zyklen
ADD.L   #8,SP            8 Zyklen
JMP     (A0)             8 Zyklen
```

Ergibt bei etwa 100 Durchläufen 100*(24+24+16+16+12+16+8+10)+ 24+24+12+8+8=12676 Zyklen

Und nun voll optimiert

```
(* $W- $E- *) | kommt später
PROCEDURE Copy(from IN A0,to IN A1 : CharPtr);
BEGIN
  REPEAT
    to+^:=from+^
  UNTIL =
END Copy;
```

wird zu

```
repeat:
MOVE.B  (A0)+,(A1)+     12 Zyklen
BNE.S   repeat          10 Zyklen
```

ergibt dann 100*(10+12)=2200 Zyklen, also etwa 5-fach schneller.

Der Postincrement und Predecrement eignet sich besonders zum Kopieren von Array oder Stringbereichen oder zur Verarbeitung von Tabellen.

6.4.6 Switches

Der Compiler versucht immer den sicheren Weg zu gehen, er überprüft Stapelüberlauf, meldet Bereichsüberschreitungen oder Überläufe bei Rechnungen. Diese Überprüfungen sind aber nicht immer nötig. Da sie Zeit und Speicher verbrauchen, ist es sinnvoll, diese Überprüfungen

bei gut ausgetesteten Programmen abzuschalten. Dafür und auch für andere Zwecke besitzt der Compiler sogenannte Switches, dies sind Schalter, die während der Compilierung gesetzt werden können und den Compilationsvorgang beeinflussen.

Die Beschreibung der einzelnen Switches und deren Syntax können Sie in Kapitel 2.5 nachlesen. hier soll lediglich Ihre Auswirkung auf die Laufzeit gezeigt werden.

Beispiel für Optimierung durch Switches:

```
PROCEDURE Sum(x IN D2,y IN D3 : INTEGER):INTEGER;
BEGIN
  RETURN x+y
END Sum;
```

wird zu:

```
Befehl           Zyklen  Funktion

SUB.W   #4,A7     4      Sicherungsbereich anlegen
MOVE.W  D2,D7     4      erstes Element holen
ADD.W   D3,D7     4      Summe bilden
TRAPV             4      Überlauf prüfen, evt. Fehlermeldung
MOVE.W  D7,D0     4      Ergebnis in Rückgaberegister
ADD.W   #4,A7     4
RTS              16      Rücksprung
```

ergibt 4+4+4+4+4+4+16=40 Zyklen.

Folgende Switches vor der Prozedur (* $V-$W- *) ergeben:

```
MOVE.W  D2,D7   4
ADD.W   D3,D7   4
MOVE.W  D7,D0   4
RTS            16
```

4+4+4+16=28 Zyklen

Der Stackcheck wurde in der Rechnung nicht berücksichtigt, da er als Prozedur im Modul System realisiert ist, und der Sprung wenig aussagekräfig ist. Allerdings braucht ein Stackcheck ziemlich viel Zeit und sollte immer als erstes ausgeschaltet werden.
Achtung: Bitte erst dann die Checks ausschalten, wenn das Programm sorgfältig getestet worden ist.

Für alle, die gerne den erzeugten Code vergleichen, sei hier gesagt, daß der Compiler bei noch angeschalteten Laufzeitchecks keine optimale Registerverwaltung bietet, also erst abschalten, dann vergleichen[22].

6.5 Bytes schaufeln und Bits rotieren

In diesem Abschnitt geht es darum, wie man in Cluster auf einzelne Bits und Bytes einer Variablen zugreifen kann, obwohl der normale Sprachumfang dafür keine Befehle zur Verfügung stellt.

Als erstes wollen wir betrachten, wie man bei einer Variablen, die länger als ein Byte ist, auf die einzelnen Bytes zugreift.

Wie Sie wahrscheinlich schon im Einführungskapitel gelesen haben, besteht die Möglichkeit, Typen, die die gleiche Länge haben, mit `CAST` in einander umzuwandeln. Dabei ändert sich lediglich der Typ, die Daten bleiben unverändert. Was läge also näher, als eine vier Byte lange Variable in ein Feld mit vier ein Byte langen Elementen umzuwandeln:

```
Arr = ARRAY [0..3] OF CHAR;

VAR l : LONGINT;

CAST(Arr,l)[3]:="C";
```

Mit dieser Anweisung wird dem niederwertigsten Byte des Langworts der Wert 65 zugewiessen. Ebenso wäre es möglich, an Stelle des Arrays einen

[22]Diese Bemerkung gilt besonders den Testern eines bekannten Computermagazins

entsprechend langen Record zu definieren, stellen Sie sich nur einmal vor, sie müßten auf das erste Wort und auf die letzen beiden Bytes eines Langwortes zugreifen.

Auf Maschinenebene und gerade bei maschinennaher Ein-/Ausgabe, geschieht viel auf Bitebene. Obwohl Cluster keine direkten Bitbefehle besitzt, haben Sie jedoch Befehle zur Manipulation von Sets zur Verfügung.

Man kann sich ein Byte im Speicher als ein SET der Zahlen 0 bis 7 vorstellen. Jedes Bit kann mit `INCL` gesetzt, mit `EXCL` gelöscht oder mit `FLIP` gekippt werden. Mit `IN` kann geprüft werden, ob es enthalten ist und einiges mehr. Viele Ein-/Ausgaberegister im Amiga sind in einzelne Bits aufgeteilt, die alle eine eigene Aufgabe haben.

Neben diesen Möglichkeiten, einzelne Bits zu setzten, kann man auch ganze Bytes und Words miteinander „verbitten“.

Um binäre Operationen mit Sets durchzuführen dienen die SET-Operatoren `+`,`-`,`*` und `/` sowie die Prozeduren `UNI` und `SEC`. Mit ihnen können Bits ausmaskiert werden, oder eine Anzahl Bits auf einen Schlag gesetzt werden. Dabei entspricht `+`/`UNI` einem binären `OR`, `*` einem binären `AND`, `/` einem `XOR` und ein`-` vor der Menge wirkt wie ein `NOT`.

Um nun eine Variable auf Bitebene zu manipulieren, „castet“ man einfach die gewünschte Variable in ein `SHORT`-, `BIT`- oder `LONGSET`, entsprechend der Variablenlänge und arbeitet dann mit Setoperatoren.

Neben diesen Bitsetz-/-löschoperationen gibt es auch noch solche, mit denen Bits verschoben werden können. Was hat man sich unter dem Schieben von Bits vorzustellen? Ein Wort hat 16 Bits, deren Wert von ihrer Position abhängig ist, das linke hat den höchsten, das rechte den niedrigsten Wert. Schiebt man nun die Bits nach links, heißt das, daß alle Bits den Wert und Platz ihres linken Nachbarn einnehmen, das ganz links stehende fällt heraus und rechts wird ein 0-Bit eingeschoben.

Beispiel:

```
0100 1001 0010 1011 => 1001 0010 0101 0110
```

Das herausgefallene Bit kommt ins Carryflag, und kann mit „`+`“ abgeprüft werden.

Man kann Zahlen auch nach rechts schieben. Bei Cardinal-Zahlen wird dann ebenfalls eine 0 eingeschoben, bei Integer-Zahlen, wird das vorderste Bit dagegen verdoppelt, um das Vorzeichen zu erhalten. Wie leicht zu erkennen ist, bedeutet eine Linksverschiebung eine Multiplikation mit 2, rechtsschieben dagegen eine Division mit 2.

Man kann eine Zahl auch mehrfach rechts oder linksschieben. Cluster verfügt dazu über die Operatoren `SHL` und `SHR`.
Beispiele:

```
1 SHL 1 = 2     1 SHL 2 = 4
1 SHL 3 = 8     1 SHL 4 = 16  200 SHR 2 = 50
```

Neben dem Schieben gibt es auch die Möglichkeit, eine Zahl zu rotieren. Dabei wird das herausfallende Bit auf der anderen Seite wieder eingeklebt. Cluster hat dazu die Standardprozeduren `ROL` und `ROR`.

```
ROR(a) oder ROL(a)
```

Eine weitere Anwendung für einen bytegenauen Zugriff ist die Ansteuerung von Hardwareerweiterungen. Sollten Sie also beabsichtigen, einen Treiber in Cluster zu programmieren[23], so gehen Sie folgendermaßen vor: Sie erhalten über die Expansionlibrary die Adresse, an der sich Ihre Erweiterungskarte einkonfiguriert hat. Weiterhin haben Sie von Ihrem Hardware entwickler eine genaue Beschreibung, in welchem Abstand die einzelnen Ein/Ausgaberegister liegen und welche Bitbreite diese haben. Nun definieren Sie einen Recordtypen, dessen Elemente jeweils den Ein/Ausgaberegisteren entsprechen. Nun müssen Sie nur noch eine Zeigervariable auf diesen Recordtypen definieren und dieser die Startadresse Ihrer Erweiterungskarte zuweisen. Nun können Sie durch einfaches Beschreiben der Recordfelder Werte in Ihre Hardwareregister schreiben und auslesen.

[23]Dies ist möglich, ich selbst habe ein Harddisk-Device in Cluster geschrieben, sowie die gesamten EGS-Libraries wurden in Cluster implementiert.

6.6 Assembler, wenn es unbedingt sein muß

Aufgrund der vielfältigen und mächtigen Möglichkeiten, die Cluster zur maschinennahen Programmierung liefert, ist es im allgemeinen unnötig, in Assembler zu programmieren. Für manche Zwecke kann allerdings nicht völlig auf direkte Verwendung von Maschinensprache verzichtet werden. Zum Beispiel bei Prozeduren, die einen Hardwareinterrupt abfangen, muß am Ende der Befehl `RTE` stehen, und nicht `RTS` wie üblich.

Zur direkten Maschinenprogrammierung bietet **Cluster** einen komfortablen Assembler[24] an. Dieser wird durch die Standardprozedur `ASSEMBLE` aufgerufen.
Hier ein Beispiel aus `Streams`:

```
ASSEMBLE( USE A0/A1/D0
          MOVE h.buff,A0
          ADD  h.buffPos,A0

          MOVE    pos,A1

          MOVE    h.buffLen,D0
          SUB     h.buffPos,D0
          BEQ     end
          SUB     #1,D0

    Loop:

          MOVE.B  (A0)+,(A1)+
          DBRA    D0,Loop

          MOVE    A1,pos
    end:
          )
```

[24]An dieser Stelle kann und soll kein Kurs in Assemblerprogrammierung stehen, zu diesem Thema gibt es Regale von guten Büchern. Hier soll lediglich auf die speziellen Eigenschaften des Clusterassemblers eingegangen werden.

Alle, die sich schon mit 68000er-Assembler beschäftigt haben, werden hier viel Vertrautes wiederfinden. Jedoch werden Sie wahrscheinlich über einiges gestolpert sein. Folgende Eigenschaften sind zu beachten:

- Die einzelnen Befehle müssen durch ein Leerzeichen oder einen Zeilenumbruch von einander getrennt sein.
- Hinter den Befehlen muß keine Längenangabe folgen, der Assembler versucht dann aus dem Typ der Operatoren die richtige Länge zu ermitteln. Ist eine andere Länge erwünscht, kann diese auf gewohnte Weise mit „`.L/W/B`" hinzugefügt werden.
- Sprungmarken werden durch einen Namen gefolgt von einem Doppelpunkt definiert.
- Wie sie sehen, kann man direkt auf Clustervariablen über deren Namen zugreifen. Jedoch nicht nur auf einfache Variablen, sondern auch auf einzelne Array oder Recordelemente kann man in gewohnter Clustersyntax zugreifen.
- Im Normalfall reserviert sich der Assembler alle Register für sich, so daß man innerhalb des Assemblerteils alle verwenden kann. Soll einem jedoch der Compiler den Zugriff auf komplexe Strukturen erleichtern, benötigt dieser ebenfalls freie Register. Daher sollte man zu begin mittels `USE` die Register angeben, die man benötigt. Die Parameterliste entspricht üblicher 68K-Syntax.
- Assembler verfügt über einen Typcheck, der mittels des Compilerswitches `$$AssTypeChk:=TRUE` eingeschaltet werden kann. Daraufhin überprüft der Assembler bei jeder Operation, ob die in den Registern enthaltenen Daten kompatible Typen enthalten.
- Am Ende des Assemblerteils muß der Stack wieder im selben Zustand wie davor sein. Wurden auf diesen z. B. für einen Prozeduraufruf Parameter abgelegt, so muß der Stack durch die Anweisung `STACK #num` wieder korigiert werden, wobei `num` die Anzahl Bytes ist, um die der Stack korigiert werden muß.

Daneben gibt es noch einige sehr rudimentäre Funktionen aus der Anfangszeit des Compilers:

SETREG(Register,Wert) weist einem Prozessorregister einen Wert zu. Dieses Register wird dabei nicht als belegt gekennzeichnet und kann bereits durch die nächste Anweisung überschrieben werden.

REG(Register) liefert den Inhalt eines Prozessorregisters.

INLINE(Wert,Wert..) schreibt die angegebenen Werte direkt in das erzeugte Programm. Damit können auf Zahlenebene Maschineninstruktionen in das Programm eingebunden werden. Ist Wert eine einfache Zahl, wird ein Wort geschrieben. Ist Wert dagegen eine Adresse, wird ein Langwort geschrieben.

6.7 Aufbau der Amiga Hardware

Als erstes eine Mitteilung an alle Clusterprogrammierer, die auch das letzte Handbuch kennen: In dieser neuen überarbeiteten Ausgabe des Handbuches verzichten wir nun auf eine Einführung in die direkte Programmierung der Graphikchips. In den ersten Tagen von Cluster mag das „Bitpopeln" legitim gewesen sein, hieß es doch aus einem 68000er das Letzte herauszuholen. Der Schock kam jedoch mit dem Erscheinen der neuen Graphikchips, durch das viele Programme auf einmal nicht mehr lauffähig waren. Da die weitere Entwicklung der Amiga-Hardware nicht vorauszusehen ist und jede Annahme in dieser Richtung die Flexibilität und Kompatibilität eines Programms herabsetzt, sollte man nach Möglichkeit das direkte Ansprechen der Custom-Chips vermeiden und die komfortablen und sicheren Routinen des Betriebssystems verwenden. Schließlich ist man bei einem 68040 nicht mehr auf jeden Zyklus angewiesen.

In der Geschichte der Computer gibt es drei Verfahren, mit denen der Prozessor mit seiner Umwelt in Verbindung tritt. Bei der ältesten Methode besitzt der Prozessor spezielle Ein/Ausgabebefehle („IN", „OUT" etc.). Diese Methode besitzt den Nachteil, daß meist die Anzahl der externen Bausteine stark begrenzt ist. Eine andere Methode funktioniert über Befehle, die nicht der Prozessor sondern der externe Baustein ausführt. Der Nachteil ist, daß diese Bausteine über eine gehörige Menge eigene „Intelligenz“ verfügen müssen und deshalb meist recht teuer sind. Darum wird diese Methode nur bei Großrechnern verwendet (Kanalprozessoren). Auf dem Amiga findet diese Art der Komunikation nur in Verbindung mit einem mathematischen Coprozessor (68881/68882) statt.

Bei der auf dem Amiga eingesetzten Methode, wird der Adreßraum des Prozessors ausgenutzt. Dazu liegt an bestimmten Adressen im System kein Speicher sondern ein externer Baustein. Diese Verfügen über spezielle Register, auf die so mit normalen Schreib- oder Lesebefehlen zugegriffen werden kann. Manche dieser Register lösen eine Aktion auf, wenn in sie geschrieben wird, andere lassen sich nur lesen.

Um von Cluster darauf zuzugreifen, sind für die Einzelnen Chips im Modul `Hardware` und `CIA` Records definiert, deren einzelne Elemente den einzelnen Registern entsprechen. Um auf diese zuzugreifen, muß nur auf das entsprechende Recordelement zugegriffen werden.

Informationen zu den einzelnen Registern entnehmen Sie bitte dem Hardware Reference Manual von Addison Wesley.

Teil II

Beschreibung der Standardmodule

Schnittstellenmodule zum System

Glossar

Index

Kapitel 7

Standardmodule

Die Standardmodule von Cluster stellen eine Sammlung von Bibliotheksmodulen dar, die einem das Programmieren erleichtern und verhindern sollen, daß das Rad mehrmals erfunden wird.
Außerdem heben Sie den Programmierer von der Hardware bzw. vom Betriebssystem ab. Sie bieten Funktionen, die man normalerweise nur durch direkte Programmierung des Betriebssystems erreichen könnte, die aber wesentlich einfacher als die des Betriebssystems zu benutzen sind. Sie beinhalten Prozeduren zur Stringhandhabung, zur Speicherverwaltung, für einfache Graphik und vieles mehr.

Wir sind ständig dabei, diese Sammlung zu erweitern. Falls Sie selbst Module geschrieben haben und diese der Allgemeinheit zugänglich machen wollen, senden Sie sie uns mit Dokumentation auf einer Diskette zu.

Bei der Beschreibung der einzelnen Module sind wir so vorgegangen, daß als erstes immer das Definitionsmodul des jeweiligen Moduls abgedruckt ist und anschließend die einzelnen Prozeduren kommentiert sind. Einen kurzen Kommentar finden Sie selbstverständlich auch in den Definitionsmodulen auf Diskette, dieser wurde jedoch hier beim Abdruck aus Platzgründen weggelassen.

Alle Quelltexte der Implementationsmodule finden Sie außerdem gepackt auf den Disketten. Es ist Ihnen gestattet, diese Module zu erweitern oder auch nach ihren Bedürfnissen zu verändern. Sollten Sie einen Fehler finden, würden wir es begrüßen, wenn Sie diesen beheben und die verbesserte Version an uns zurückschicken. Änderungen sind jedoch nur zu empfehlen, wenn diese nur im Implementationsmodul geschehen. Verändern Sie die Funktionsweise nicht gravierend, für den Fall, daß Sie einmal ein fremdes Programm übersetzen wollen, das von der ursprünglichen Version ausgeht.

Um zu verdeutlichen, ob es sich bei den Parametern um Ein- oder Ausgabeparameter handelt, wird dies durch folgende Symbole angezeigt:

- $\Leftarrow$ Parameter dient nur zur Dateneingabe.
- $\Rightarrow$ Parameter dient nicht zur Eingabe, sondern wird ausschließlich von der Prozedur beschrieben (verändert).

  Tritt dieses Symbol ganz am Ende einer Beschreibung auf, ohne vorangestellten Parameternamen, handelt es sich um einen Funktionsrückgabewert.
- $\Leftrightarrow$ Es handelt sich um einen Eingabeparameter, dessen Wert jedoch innerhalb der Prozedur verändert wird, so daß er auch als Ergebnis zu verstehen ist.

7.1 Arguments

Das Modul „**Arguments**“ dient dazu, Argumente abzufragen, die bei einem Aufruf vom CLI hinter dem Programmnamen, oder beim Start von der Workbench über erweiterte Auswahl angegeben wurden. Außerdem bietet es Prozeduren zur vereinfachten Bearbeitung der Tooltypes.

```
DEFINITION MODULE Arguments;

VAR
  NumArgs    : INTEGER;
  Wb         : BOOLEAN;

$$OwnHeap:=TRUE
PROCEDURE Arg(num : INTEGER):STRING;

PROCEDURE Exists(REF arg    : STRING;
                     start  : INTEGER:=0;
                     caps   : BOOLEAN:=TRUE):INTEGER;

$$OwnHeap:=TRUE
PROCEDURE FindToolType(REF path,
                           toolType : STRING):STRING;

PROCEDURE MatchToolValue(REF typeString,
                             val    : STRING):BOOLEAN;

GROUP
  All = NumArgs,Wb,Arg,Exists;

END Arguments.
```

`NumArgs` : Anzahl übergebener Argumente

`WB` : Flag für Workbenchstart


```
PROCEDURE Arg(num : INTEGER):STRING;
```

Funktion: Gibt Argumentstring zurück

Parameter:

`num` ⇐ Nummer des Arguments (`0..NumArgs-1`), das ausgegeben werden soll.

⇒ String mit dem `num`-ten Argument. Wurde das Programm von der Workbench gestartet, erhält man den kompletten Pfad der übergebenen Datei. Beim Workbenchstart gibt `Arg(-1)` den Namen des Programms zurück.

```
PROCEDURE Exists(REF arg    : STRING;
                     start  : INTEGER:=0;
                     caps   : BOOLEAN:=TRUE):INTEGER;
```

Funktion: Überprüft das Vorhandensein eines Arguments und liefert Position zurück.

Parameter:

`arg` ⇐ Argumentstring, nach dem gesucht werden soll.

`num` ⇐ Nummer des Arguments, von dem ab gesucht werden soll. Wichtig, falls ein Argument mehrfach vorkommen darf.

`caps` ⇐ Flag, ob ohne Rücksicht auf die Groß- oder Kleinschreibung verglichen werden soll.

⇒ Position des Arguments oder -1, falls kein solches Argument übergeben wurde

```
PROCEDURE FindToolType(REF path,
                           toolType : STRING):STRING;
```

Funktion: Liefert den Wert eines ToolTypes

Parameter:

`path`	⇐	Pfad der Datei, auf die sich der Befehl bezieht. Dabei ist zu beachten, daß nur der Name des eigentlichen Files ohne angehängtes „`.info`“ übergeben wird.
`toolType`	⇐	Name des Tooltypes, dessen Wert abgefragt werden soll.
	⇒	Wert des Tooltypes oder ein Leerstring, falls der Tooltype nicht gesetzt war.

```
PROCEDURE MatchToolValue(REF typeString,
                             val    : STRING):BOOLEAN;
```

Funktion: Vergleicht einen übergebenen ToolType-Wert mit einem vorgegebenen ToolType. Dabei wird nicht Case-Sensitiv vorgegangen. Außerdem wird unterschieden, ob in `TypeString` mehrere alternative Möglichkeiten existieren, die durch ein „|“ getrennt sind.

Parameter:

`typeString`	⇐	ToolTypewert, der verglichen werden soll.
`val`	⇐	Wert oder Muster, mit dem „`typestring`“ verglichen werden soll.

Zur Verdeutlichung von `MatchToolValue` hier einige Beispiele:

```
type1 = "text"
type2 = "a|b|c"

MatchToolValue( type1, "text" ) -> TRUE
MatchToolValue( type1, "TEXT" ) -> TRUE
MatchToolValue( type1, "data" ) -> FALSE
MatchToolValue( type2, "a" )    -> TRUE
MatchToolValue( type2, "b" )    -> TRUE
MatchToolValue( type2, "d" )    -> FALSE
MatchToolValue( type2, "a|b" )  -> FALSE
```

7.2 ASCII

Neben den druckbaren Zeichen gibt es noch einige nicht druckbare Steuerzeichen, die aber zum Beispiel zur Druckeransteuerung oder zur Textformatierung gebraucht werden. Die Zahlwerte dieser Zeichen sind in diesem Modul als Konstanten definiert.

```
DEFINITION MODULE ASCII;

CONST
 nul = &00;  soh = &01;  stx = &02;  etx = &03;
 eot = &04;  enq = &05;  ack = &06;  bel = &07;
 bs  = &08;  ht  = &09;  lf  = &10;  vt  = &11;
 ff  = &12;  cr  = &13;  so  = &14;  si  = &15;
 dle = &16;  dc1 = &17;  dc2 = &18;  dc3 = &19;
 dc4 = &20;  nak = &21;  syn = &22;  etb = &23;
 can = &24;  em  = &25;  sub = &26;  esc = &27;
 fs  = &28;  gs  = &29;  rs  = &30;  us  = &31;
 sp  = &32;
 del = &128;
 eol = lf;
 eof = fs;
 csi = &155;

END ASCII.
```

Die meisten dieser Steuerzeichen sind jedoch heute nicht mehr von so großer Bedeutung, als zu der Zeit, als sie vor allem zur Ansteuerung von Terminals verwendet wurden. Daher werden hier nur die Konstanten beschrieben, die gebräuchlich sind.

nul Das Nullbyte, wie es z. B. als Endekennzeichen von Strings verwendet wird.

bs *Backspace*, Rückschritt

lf *Linefeed*, bewirkt einen Zeilenvorschub und einen Wagenrücklauf.

ff *Formfeed* oder Seitenvorschub, bewirkt auf einem Ducker den Ausstoß der aktuellen und dem Einzug einer neuen Seite. Auf der Console wird das Fenster gelöscht.

cr Wagenrücklauf, auf manchen Druckern auch mit einem Linefeed verbunden.

esc *Escape*, dieses Zeichen leitet stets eine Kette Zeichen ein, die dann vom Ausgabegerät als Kommando interpretiert wird.

sp *Space* oder Leertaste.

del *Delete*, Zeichen löschen

eol *End of line*, entspricht dem Zeilenvorschub.

eof *End of File*, kennzeichnet das Ende einer Textdatei.

bel *Bell*, läßt Klingel ertönen, bzw. der Bildschirm blitzt.

ht *Horizontaler Tabulator*, bewirkt Vorlauf auf den nächsten Tabstop.

7.3 AVLTrees

Das Modul „`AVLTrees`“ ist ein generisches Modul[2], das Ihnen Routinen zur Verwaltung in AVL-Bäumen[3] zur Verfügung stellt. Der Unterschied der beiden internen Module „`AVLTrees`“ und „`AVLCursorTrees`“ besteht darin, daß bei einem Cursor-Baum nicht das Element selbst eingehängt wird, sondern nur eine Verweis darauf eingetragen wird. Das hat zur Folge, daß ein Element in beliebig vielen Cursor-Bäumen hängen kann.

```
DEFINITION MODULE AVLTrees;

FROM System IMPORT Equation;

  DEFINITION MODULE
    AVLTrees(AVLNodePtr : POINTER TO AVLNode);

    TYPE
      AVLNode    = RECORD
                     left,
                     right,
                     parent : AVLNodePtr;
                     a_fact : INTEGER;
                   END;
    Comparison = PROCEDURE(a,b : AVLNodePtr):Equation;

    AVLTree    = RECORD
                     root    : AVLNodePtr;
                     compare : Comparison;
                   END;

    EXCEPTION
      AllreadyExists : "Allready exists";
      NotFound       : "Not found";
      TreeEmpty      : "Tree is empty";

    PROCEDURE Init(VAR t       : AVLTree;
                       compare : Comparison);
```

[2] Falls Sie mit diesem Begriff noch nichts anfangen können, lesen Sie bitte im Kapitel 4 nach.

[3] Ein AVL-Baum ist ein Binärbaum (siehe Kapitel 4), der niemals unausgeglichen sein kann.


```
    PROCEDURE Insert(VAR t    : AVLTree;
                         data : AVLNodePtr);

    PROCEDURE Search(REF t    : AVLTree;
                         data : AVLNodePtr):AVLNodePtr;

    PROCEDURE Delete(VAR t    : AVLTree;
                         data : AVLNodePtr);

    PROCEDURE Remove(VAR t    : AVLTree;
                         data : AVLNodePtr);

    PROCEDURE Next(REF t    : AVLTree;
                       data : AVLNodePtr):AVLNodePtr;

    PROCEDURE Prev(REF t    : AVLTree;
                       data : AVLNodePtr):AVLNodePtr;

    PROCEDURE First(REF t : AVLTree):AVLNodePtr;

    PROCEDURE Last(REF t : AVLTree):AVLNodePtr;

  END AVLTrees;
```

```
DEFINITION MODULE AVLCursorTrees(type : ANYPTR);

  TYPE
    CNodePtr   = POINTER TO CNode;

  DEFINITION MODULE CTree = AVLTrees(CNodePtr);

  TYPE
    CNode      = RECORD OF CTree.AVLNode
                   data : type;
                 END;

    Comparison = PROCEDURE(a,b : type):Equation;
    Destructor = PROCEDURE(a : type);

    AVLTree    = RECORD OF CTree.AVLTree
                   cursor   : CNodePtr;
                   greater2 : Comparison;
                 END;

  FROM CTree IMPORT AllreadyExists,NotFound;

  PROCEDURE Init(VAR t       : AVLTree;
                 compare : Comparison);

  PROCEDURE Insert(VAR t : AVLTree;data : type);

  PROCEDURE Search(VAR t : AVLTree;data : type);

  PROCEDURE Get(VAR t : AVLTree):type;

  PROCEDURE Delete(VAR t : AVLTree);

  PROCEDURE Remove(VAR t : AVLTree);

  PROCEDURE Destruct(VAR t          : AVLTree;
                     destructor : Destructor);

  PROCEDURE Delete_All(VAR t : AVLTree);

  PROCEDURE Destruct_All(VAR t          : AVLTree;
                         destructor : Destructor);


  PROCEDURE Next(VAR t : AVLTree);

  PROCEDURE Prev(VAR t : AVLTree);

  PROCEDURE First(VAR t : AVLTree);
```



```
    PROCEDURE Last(VAR t : AVLTree);

  END AVLCursorTrees;

END AVLTrees.
```

7.3.1 AVLTrees

```
PROCEDURE Init(VAR t       : AVLTree;
                   compare : Comparison);
```

Funktion: Initialisiert einen AVL-Baum

Parameter:

`t` ⇒ Variable vom Typ `AVLTree`, die initialisiert werden soll. Sie wird bei allen zukünftigen Operationen auf den Baum als Zugriff auf den Baum gebraucht.

`compare` ⇐ Funktion vom Typ `Comparison`. Diese wird zur richtigen Einsortierung der Elemente benötigt. Die Funktion bekommt zwei Elemente vom Typ `AVLNodePtr` übergeben und muß festlegen, welche von beiden größer, gleich oder kleiner ist.

```
PROCEDURE Insert(VAR t    : AVLTree;
                     data : AVLNodePtr);
```

Funktion: Fügt einen neuen Knoten in den AVL-Baum ein. Enthält der Baum schon einen solchen Knoten, wird die Exception *AllreadyExists* ausgelöst.

Parameter:

`t` ⇐ Baum, in den der Knoten eingefügt werden soll.

`data` ⇐ Knoten, der eingefügt werden soll.

```
PROCEDURE Search(REF t    : AVLTree;
                     data : AVLNodePtr):AVLNodePtr;
```

Funktion: Sucht einen Knoten in einem Baum. Ist der Knoten nicht im Baum, wird die Exception *NotFound* ausgelöst.

Parameter:

`t`	⇐	Baum, in dem gesucht werden soll.
`data`	⇐	Knoten, nach dem gesucht werden soll.
	⇒	Zeiger auf den gefundenen Knoten.

```
PROCEDURE Delete(VAR t    : AVLTree;
                 VAR data : AVLNodePtr);
```

Funktion: Entfernt einen Knoten aus dem Baum und gibt dessen Speicher frei.

Parameter:

`t`	⇐	Baum, aus dem der Knoten entfernt werden soll.
`data`	⇐	Knoten, der gelöscht werden soll. Nach dem Aufruf enthält `data` den Wert `NIL`.

```
PROCEDURE Remove(VAR t    : AVLTree;
                     data : AVLNodePtr);
```

Funktion: Entfernt einen Knoten aus dem Baum, ohne jedoch seinen Speicher freizugeben.

Parameter:

`t`	⇐	Baum, aus dem der Knoten entfernt werden soll.
`data`	⇐	Knoten, der entfernt werden soll.


```
PROCEDURE Next(REF t    : AVLTree;
                   data : AVLNodePtr):AVLNodePtr;
```

Funktion: Da ein AVL-Baum ein geordneter Baum ist, bei dessen Initialisierung schließlich ein Ordnungskriterium übergeben wurde, kann man zu jedem Knoten einen Vorgänger und einen Nachfolger finden. `Next` liefert nun den Nachfolger eines Knotens oder `NIL`, wenn der höchstwertigste Knoten schon erreicht war.

Parameter:

`t`	⇐	Baum, auf den sich die Operation bezieht.
`data`	⇐	Knoten, dessen Nachfolger gefunden werden soll.
	⇒	Nachfolgerknoten oder `NIL`.

```
PROCEDURE Prev(REF t    : AVLTree;
                   data : AVLNodePtr):AVLNodePtr;
```

Funktion: Entsprechendes Gegenstück zu `Next`. `Prev` liefert den Vorgänger eines Knotens oder `NIL`, wenn der niedrigwertigste Knoten schon erreicht war.

Parameter:

`t`	⇐	Baum, auf den sich die Operation bezieht.
`data`	⇐	Knoten, dessen Vorgänger gefunden werden soll
	⇒	Vorgängerknoten oder `NIL`.

PROCEDURE `First(`**REF** `t : AVLTree):AVLNodePtr;`

Funktion: Liefert den Knoten eines Baumes, mit dem kleinsten Wert.

Parameter:

`t` ⇐ Baum, auf den sich die Operation bezieht.

⇒ Knoten mit dem niedrigsten Wert. Falls der Baum leer war, wird die Exception *TreeEmpty* ausgelöst.

PROCEDURE `Last(`**REF** `t : AVLTree):AVLNodePtr;`

Funktion: Liefert den Knoten eines Baumes mit dem größten Wert.

Parameter:

`t` ⇐ Baum, auf den sich die Operation bezieht.

⇒ Knoten mit dem höchsten Wert. Falls der Baum leer war, wird die Exception *TreeEmpty* ausgelöst.

7.3.2 AVLCursorTrees

Wie schon anfangs erwähnt, hängen die Knoten eines Cursor-Baumes nicht direkt im Baum, sondern nur jeweils ein Knoten mit einem Verweis auf die eigentlichen Daten. Dies hat zwei Folgen:

- Ein Datenfeld kann so in beliebig viele Cursor-Bäume eingfügt werden. Dies kann sinnvoll sein, wenn die einzelnen AVL-Bäume nach anderen Ordnungskriterien aufgebaut sind.
- Die Knoten müssen nicht Nachfolger eines Stammknotens sein wie dies in **`AVLTrees`** sein muß. Da sie nicht direkt eingehängt werden, muß das Modul nichts über den Typ wissen.

Wichtig jedoch: Bei der Freigabe eines solchen Knotens sollte man sicher gehen, daß er nicht noch in einem anderen Baum eingetragen ist. Daher erst aus allen Bäumen herausnehmen und dann freigeben.

Da die meisten Prozeduren identisch zu denen in **`AVLTrees`** sind, werde ich nur auf die Unterschiede aufmerksam machen. Sollte es Sie verwirren, daß bei allen Funktion nur der Baum und keine Knoten übergeben werden, dann liegt das daran, daß bei einem Cursor-Baum immer ein Knoten als aktueller Knoten gilt, der sich innerhalb des Baumes verschieben läßt. D. h. alle Operationen beziehen sich auf den aktuellen Knoten.

Um einen bestimmten Knoten zu bearbeiten, ruft man **`Search`** auf. Danach zeigt der aktuelle Knoten des Baumes auf den gesuchten Eintrag, sofern er im Baum enthalten war. Nun kann man jede andere Funktion aufrufen.

Remove löscht nur den Verweisknoten aus dem Baum. Das Datenfeld, auf den der Konten gezeigt hat, bleibt dabei unangetastet.

Delete löscht nicht nur den Verweisknoten aus dem Baum, sondern auch das Datenfeld wird durch Dispose freigegeben. Nur geeignet, wenn das Datenfeld nur aus Speicher besteht.

Destruct Falls zur Freigabe eines Knotens eine spezielle Routine nötig ist, z. B. wenn ein Knoten einen Zeiger auf ein Fenster enthält, das vorher noch geschlossen werden muß, verwendet man diese Prozedur. `Destruct` löscht den Eintrag aus dem Baum, und übergibt das Datenfeld der mitübergebenen Freigabeprozedur vom Typ `Destructor`.

Delete_All Vernichtet die gesamte Baumstruktur und gibt alle Datenfelder mit `Dispose` frei, es gilt die gleiche Einschränkung wie bei `Delete`.

Destruct_All Wie `Delete`, nur daß für jedes Datenfeld eine übergebene Freigabeprozedur aufgerufen wird. Siehe unter Destruct.

Nun noch ein kleines Beispielprogramm zur Verdeutlichung. Hierbei werden nacheinander mehrere Datensätze in einen nach Namen geordneten Baum eingehängt, anschließend wird nach einem Datensatz anhand des Namens gesucht. Sicher ist dieses Verfahren bei drei Datensätzen nicht sinnvoll, jedoch soll schließlich nur das Verfahren gezeigt werden.


```
MODULE AVLTreeTest;

FROM Strings   IMPORT Compare;
FROM System    IMPORT Equation;
FROM InOut     IMPORT WriteGrp;
FROM Resources IMPORT New;
IMPORT AVLTrees;

TYPE
  PersonPtr = POINTER TO Person;

DEFINITION MODULE PersonTrees = AVLTrees.AVLTrees(PersonPtr);

TYPE
  Person = RECORD OF PersonTrees.AVLNode;
             name    : STRING(20);
             alter   : SHORTINT;
             strasse : STRING(30);
           END;

PROCEDURE ComparePerson(a,b : PersonPtr):Equation;
BEGIN
  RETURN Compare(a^.name,b^.name);
END ComparePerson;


VAR
  PersonTree : PersonTrees.AVLTree;
  p,p2       : PersonPtr;

BEGIN
  TRACK
    PersonTree.Init(ComparePerson);
    New(p);
    p^.name:="Meier";
    p^.alter:=38;
    p^.strasse:="Neuer Weg 27";
    PersonTree.Insert(p);
    New(p);
    p^.name:="Müller";
    p^.alter:=75;
    p^.strasse:="Musterweg 80";
    PersonTree.Insert(p);
    New(p);
    p^.name:="Schulze";
    p^.alter:=21;
```

```
    p^.strasse:="Kaiserstr. 17";
    PersonTree.Insert(p);

    New(p);
    p^.name:="Müller";
    p2:=PersonTree.Search(p);
    WriteString(p2^.name);WriteLn;
    WriteInt(p2^.alter);WriteLn;
    WriteString(p2^.strasse);WriteLn;
  END;
END AVLTreeTest.
```


7.4 Buffers

Wie bei `AVLTrees` enthält das Modul Buffers zwei generische Module zur Bearbeitung von Stacks und Queues. Damit kann jeder Pointertyp verwendet werden.

```
DEFINITION MODULE Buffers;

EXCEPTION
  BufferEmpty : "Buffer empty";
  NotInBuffer : "Object not in buffer";

  DEFINITION MODULE Queues(type : ANYPTR);
    TYPE
      QueueNodePtr = HIDDEN;
      Queue        = RECORD
                       first : QueueNodePtr;
                     END;
      Destructor   = PROCEDURE(p : type);

    PROCEDURE Init(VAR q : Queue);

    PROCEDURE Remove_All(VAR q : Queue);

    PROCEDURE Remove(VAR q : Queue;data : type);

    PROCEDURE Destruct_All(VAR q    : Queue;
                               des  : Destructor);

    PROCEDURE Delete_All(VAR q : Queue);

    PROCEDURE Put(VAR q : Queue;data : type);

    PROCEDURE Get(VAR q : Queue):type;

    PROCEDURE IsEmpty(VAR q : Queue):BOOLEAN;

  END Queues;

  DEFINITION MODULE Stacks(type : ANYPTR);
    TYPE
      StackNodePtr  = HIDDEN;
      Stack         = RECORD
                        first : StackNodePtr;
                      END;
      Destructor    = PROCEDURE(p : type);
```

```
    PROCEDURE Init(VAR q : Stack);

    PROCEDURE Remove_All(VAR q : Stack);

    PROCEDURE Remove(VAR q : Stack;data : type);

    PROCEDURE Destruct_All(VAR q    : Stack;
                               des  : Destructor);

    PROCEDURE Delete_All(VAR q : Stack);

    PROCEDURE Put(VAR q : Stack;data : type);

    PROCEDURE Get(VAR q : Stack):type;

    PROCEDURE IsEmpty(VAR q : Stack):BOOLEAN;

  END Stacks;

END Buffers.
```


Exceptions:

BufferEmpty Kann durch `Get` ausgelöst werden, falls der Puffer bereits leer ist.

NotInBuffer Kann durch `Remove` ausgelöst werden, falls das Objekt nicht im Puffer ist.

7.4.1 Queues

Hierbei handelt es sich um einen FIFO-Puffer[4], Elemente können beliebig in den Puffer eingefügt werden und in derselben Reihenfolge wieder ausgelesen werden. Typische Anwendung ist eine Warteschlange, z. B. ein Tastaturpuffer.

```
PROCEDURE Init(VAR q : Queue);
```

Funktion: Initialisiert den Queue, muß vor der ersten Verwendung geschehen.

Parameter:
`q` ⇒ Queue, der initialisiert werden soll.

```
PROCEDURE Remove_All(VAR q : Queue);
```

Funktion: Vernichtet die Pufferstruktur. Die Elemente, die sich zu dieser Zeit noch darin befinden, werden nicht angetastet, da in der Struktur nur Verweise auf die Elemente enthalten sind.

Parameter:
`q` ⇔ Queue, der aufgelöst werden soll.

[4] FIFO= First In First Out

PROCEDURE `Remove(`**VAR** `q : Queue;data : type);`

Funktion: Entfernt ein Element aus dem Puffer, unabhängig von dessen Position im Puffer.

Parameter:

`q`	$\Leftrightarrow$	Queue, aus dem das Element entfernt werden soll.
`data`	$\Leftarrow$	Zeiger auf das Element, das entfernt werden soll.

```
PROCEDURE Destruct_All(VAR q   : Stack;
                           des : Destructor);
```

Funktion: Vernichtet den Puffer. Für jedes Element, das sich zu dieser Zeit noch darin befindet, wird eine spezielle Freigabeprozedur aufgerufen. Sinnvoll, falls das Element nicht nur aus Speicher besteht, sondern z. B. ein File, das noch geschlossen werden muß.

Parameter:

`q`	$\Leftrightarrow$	Queue, der vernichtet werden soll.
`des`	$\Leftarrow$	Freigabeprozedur vom Typ `Destructor`, die für jedes Element aufgerufen werden soll.

PROCEDURE `Delete_All(`**VAR** `q : Queue);`

Funktion: Vernichtet den Puffer. Die Elemente, die sich zu dieser Zeit noch darin befinden, werden durch `Dispose` freigegeben. Diese Funktion ist `Destruct_All` vorzuziehen, wenn keine besonderen Freigaberoutinen erforderlich sind.

Parameter:

`q`	$\Leftrightarrow$	Queue, der vernichtet werden soll.

PROCEDURE `Put(`**VAR** `q : Queue;data : type);`

Funktion: Fügt ein Element in den Queue so ein, daß es als oberstes im Stapel liegt.

Parameter:

`q`	$\Leftarrow$	Queue, in den das Element eingefügt werden soll.
`data`	$\Leftarrow$	Element, das eingefügt werden soll.

PROCEDURE `Get(`**VAR** `q : Queue):type;`

Funktion: Entfernt das Element aus dem Queue, das als erstes eingefügt worden ist, also ganz unten liegt.

Parameter:

`q`	$\Leftarrow$	Queue, aus dem das Element entfernt werden soll.
	$\Rightarrow$	Element, das als unterstes im Queue lag.

PROCEDURE `IsEmpty(`**VAR** `q : Queue):BOOLEAN;`

Funktion: Prüft, ob der Queue leer ist.

Parameter:

`q`	$\Leftarrow$	Queue, der überprüft werden soll.
	$\Rightarrow$	`TRUE`, falls er leer ist.
	$\Rightarrow$	`FALSE`, falls er nicht leer ist.

7.4.2 Stacks

Hierbei handelt es sich um einen LIFO-Puffer[5]. Das heißt, das Element, das als letztes daraufgelegt worden ist, wird auch wieder als erstes heruntergenommen. Typisches Beispiel ist z. B. ein Sicherungsspeicher für eine Undo-Funktion.

Da die Funktionen exakt die gleichen wie in Queues sind, werden sie nicht noch einmal aufgeführt.

[5] LIFO= Last In First Out

7.5 Conversions

`Conversions` enthält sowohl Funktionen zur Umwandlung von in Strings enthaltenen Ziffernfolgen in echte Zahlen[6], als auch solche zur Umwandlung von Zahlen in Zeichenfolgen, um Zahlen ausgeben zu können. Die Ein/Ausgabeprozeduren von `InOut` arbeiten ebenfalls mit `Conversions`.

```
DEFINITION MODULE Conversions;
FROM System       IMPORT Regs;
FROM Exceptions   IMPORT RangeViolation;

EXCEPTION
  NoPointInMask    : "No point in mask";
  IllegalChar      : "Illegal character in string";


$$OwnHeap:=TRUE
PROCEDURE RealToString(val      : LONGREAL;
                       field,
                       digits   : CARDINAL;
                       fillChar : CHAR :=" ";
                       point    : CHAR :=".")):STRING;

$$OwnHeap:=TRUE
PROCEDURE RealToExpString(val       : LONGREAL;
                          digits    : CARDINAL:=6;
                          expDigits : CARDINAL:=3;
                          point     : CHAR:=".")):STRING;
```

[6]Dies ist notwendig, da man über die Tastatur nur Zeichenfolgen erhält, jedoch keine Zahlwerte mit denen man rechnen kann


```
$$OwnHeap:=TRUE
PROCEDURE RealToMaskedString(    val      : LONGREAL;
                             REF mask     : STRING;
                                 fillChar : CHAR := " ";
                                 point    : CHAR := "."):STRING;

$$OwnHeap:=TRUE
PROCEDURE IntToString(val      IN D2 : LONGINT;
                      field    IN D3 : INTEGER:=0;
                      fillChar       : CHAR:=" "): STRING;

$$OwnHeap:=TRUE
PROCEDURE IntToHexString(val      IN D2 : LONGINT;
                         field    IN D3 : INTEGER:=0;
                         fillChar       : CHAR:=" ";
                         dollar         : BOOLEAN:=FALSE):STRING;

$$OwnHeap:=TRUE
PROCEDURE IntToBinString(val      IN D2 : LONGINT;
                         field    IN D3 : INTEGER:=0;
                         fillChar       : CHAR:=" ";
                         sign           : BOOLEAN:=FALSE):STRING;

$$OwnHeap:=TRUE
PROCEDURE CardToString(val      IN D2 : LONGCARD;
                       field    IN D3 : INTEGER:=0;
                       fillChar       : CHAR:=" "): STRING;

$$OwnHeap:=TRUE
PROCEDURE CardToHexString(val      IN D2 : LONGCARD;
                          field    IN D3 : INTEGER:=0;
                          fillChar IN D4 : CHAR:=" ";
                          dollar   IN D5 : BOOLEAN := FALSE):STRING;
```

```
$$OwnHeap:=TRUE
PROCEDURE CardToBinString(val      IN D2 : LONGCARD;
                          field    IN D3 : INTEGER:=0;
                          fillChar IN D4 : CHAR:=" ";
                          sign     IN D5 : BOOLEAN := FALSE):STRING;


PROCEDURE StringToReal(REF str   IN A0 : STRING;
                           point       : CHAR:="."):LONGREAL;

PROCEDURE StringToInt(REF str IN A0 : STRING):LONGINT;

PROCEDURE HexStringToInt(REF str IN A0 : STRING):LONGINT;

PROCEDURE BinStringToInt(REF str IN A0 : STRING):LONGINT;

PROCEDURE StringToCard(REF str IN A0 : STRING):LONGCARD;

PROCEDURE HexStringToCard(REF str IN A0 : STRING):LONGCARD;

PROCEDURE BinStringToCard(REF str IN A0 : STRING):LONGCARD;

GROUP
  All   = RealToString,RealToExpString,RealToMaskedString,
          IntToString,CardToHexString,StringToReal,StringToInt,
          HexStringToCard,IntToHexString,IntToBinString,
          CardToString,CardToHexString,CardToBinString,
          HexStringToInt,BinStringToInt,StringToCard,
          BinStringToCard,NoPointInMask;

END Conversions.
```


Exceptions

RangeViolation Die übergebene Zahl paßt nicht in den String, in das angegebene Feld, ist zu groß für das gewünschte Zahlenformat oder paßt vom Vorzeichen nicht.

NoPointInMask In einer Formatmaske muß ein Dezimalpunkt enthalten sein, der dem Zeichen entspricht, das als Parameter `point` übergeben wurde.

IllegalChar In dem übergebenen String befanden sich nicht konvertierbare Zeichen.

7.5.1 Value to String

```
PROCEDURE RealToString(val      : LONGREAL;
                       field,
                       digits   : CARDINAL;
                       fillChar : CHAR :=" ";
                       point    : CHAR :="."):STRING;
```

Funktion: Wandelt eine Realzahl in eine Zeichenkette.

Parameter:

`val`	$\Leftarrow$	Realzahl, die umgewandelt werden soll.
`field`	$\Leftarrow$	Gesamtzahl der Stellen, die umgewandelt werden sollen.
`digits`	$\Leftarrow$	Anzahl der gewünschten Nachkommastellen.
`fillChar`	$\Leftarrow$	Füllzeichen, mit dem `field` aufgefüllt werden soll.
`point`	$\Leftarrow$	Zeichen, das für den Dezimalpunkt verwendet werden soll.
	$\Rightarrow$	String mit der umgewandelten Zahl

.

```
PROCEDURE RealToExpString(val       : LONGREAL;
                          digits    : CARDINAL:=6;
                          expDigits : CARDINAL:=3;
                          point     : CHAR:="."):STRING;
```

Funktion: Wie `RealToString`, jedoch wird die Zahl in Exponentialdarstellung umgewandelt.

Parameter:

`val`	⇐	Realzahl, die umgewandelt werden soll.
`digits`	⇐	Anzahl der gewünschten Nachkommastellen.
`expDigits`	⇐	Anzahl der Exponentialstellen.
`point`	⇐	Zeichen, das für den Dezimalpunkt verwendet werden soll.
	⇐	String mit der umgewandelten Zahl.

```
PROCEDURE RealToMaskedString(    val      : LONGREAL;
                             REF mask     : STRING;
                                 fillChar : CHAR := " ";
                                 point    : CHAR := "."):STRING;
```

Funktion: Wandelt eine Longrealzahl in eine maskierte Realzahl. Siehe unten.

Parameter:

`val`	⇐	Zahl, die konvertiert werden soll.
`mask`	⇐	Maskierungsstring.
`point`	⇐	Zeichen, das als Dezimalpunkt verwendet werden soll.
	⇒	String, der die umgewandelte Zahl enthält.

Bemerkung:
Diese Funktion eignet sich hervorragend zum Aufbau von Tabellen. Die Maskierung sieht dabei folgendermaßen aus:

```
Mask : "ccccc+###c###c###.####
oder   "ccccc#.#########E****"
```


Wobei „`c`“ für ein beliebiges Zeichen, „`+`“ für die Position des Vorzeichens, „`#`“ für eine Ziffer und „`E***`“ für die Exponentialdarstellung steht. Ziffernstellen, die nicht von der Zahl ausgefüllt werden, werden mit `fillChar` gefüllt. `point` gibt das Zeichen für den Dezimalstring an. Wichtig ist, `point` darf zwischen dem ersten und dem letzten Nummernzeichen nur einmal vorkommen. Zur Verdeutlichung hier einige Beispiele:

```
val = -12533.7892
mask = "Dies ist eine Zahl #######,### als Beispiel."
fillChar = " "
point = ","
```

⇒„Dies ist eine Zahl -12533,789“ als Beispiel.“

```
val = -12533.7892
mask = "Wert + ##.###.###,#### "
fillChar = "0"
point = ","
```

⇒„Wert - 00.012.533,7892“

```
val = 0.00000056778
mask = "Größe +#.### E*** "
fillChar = " "
point = "."
```

⇒„Größe 5.677 E-07“

```
val = 12533.78
mask = "Betrag ####### DM ## Pf"
fillChar = " "
point = "D"
```

⇒„Betrag   12533 DM 78 Pf“

```
PROCEDURE IntToString(val      IN D2 : LONGINT;
                      field    IN D3 : INTEGER:=0;
                      fillChar       : CHAR:=" "): STRING;
```

Funktion: Wandelt eine Integerzahl in eine Zeichenkette.

Parameter:

`val`	⇐	Integerzahl, die umgewandelt werden soll.
`field`	⇐	Anzahl der Stellen, die umgewandelt werden sollen.
`fillChar`	⇐	Füllzeichen, mit dem `field` aufgefüllt werden soll.
	⇒	String mit der umgewandelten Zahl.

```
PROCEDURE IntToHexString(val      IN D2 : LONGINT;
                         field    IN D3 : INTEGER:=0;
                         fillChar       : CHAR:=" ";
                         dollar         : BOOLEAN:=FALSE):STRING;
```

Funktion: Wandelt eine Integerzahl in eine Zeichenkette in Hexadezimaldarstellung mit Vorzeichen um. Die Darstellung entspricht nicht dem 2er-Komplement.

Parameter:

`val`	⇐	Integerzahl, die umgewandelt werden soll.
`field`	⇐	Anzahl der Stellen, die umgewandelt werden sollen.
`fillChar`	⇐	Füllzeichen, mit dem `field` aufgefüllt werden soll.
`dollar`	⇐	Flag, ob ein `$`-Zeichen vor der Zahl eingefügt werden soll.
	⇒	String mit der umgewandelten Zahl.


```
PROCEDURE IntToBinString(val      IN D2 : LONGINT;
                         field    IN D3 : INTEGER:=0;
                         fillChar       : CHAR:=" ";
                         sign           : BOOLEAN:=FALSE):STRING;
```

Funktion: Wandelt eine Integerzahl in eine Zeichenkette in Binärdarstellung mit Vorzeichen. Negative Zahlen werden ebenfalls nicht im 2er-Komplement dargestellt.

Parameter:

`val`	⇐	Integerzahl, die umgewandelt werden soll.
`field`	⇐	Anzahl der Stellen, die umgewandelt werden sollen.
`fillChar`	⇐	Füllzeichen, mit dem `field` aufgefüllt werden soll.
`sign`	⇐	Flag, ob ein `%`-Zeichen vor der Zahl eingefügt werden soll.
	⇒	String mit der umgewandelten Zahl.

Die Funktionen `CardToString`, `CardToHexString` und `CardToBinString` entsprechen in Funktion und Parametern denen von `IntToString`, `IntToHexString` und `IntToBinString`. Daher werden sie hier nicht noch einmal alle aufgeführt. Übergibt man `CardToBinString` oder `CardToHexString` bei ausgeschaltetem
RangeCheck eine negative Zahl, wird diese im 2er-Komplement dargestellt.

7.5.2 String to Value

```
PROCEDURE StringToReal(REF str   IN A0 : STRING;
                           point       : CHAR:=".."):LONGREAL;
```

Funktion: Wandelt einen String in eine Realzahl, auch Strings in Exponentialdarstellung.

Parameter:

`str`	⇐ Zeichenkette, die umgewandelt werden soll.
`point`	⇐ Zeichen, welches als Dezimalpunkt verwendet wird.
	⇒ Realzahl.

```
PROCEDURE StringToInt(REF str IN A0 : STRING):LONGINT;
```

Funktion: Wandelt einen String in eine Longintzahl.

```
PROCEDURE HexStringToInt(REF str IN A0 : STRING):LONGINT;
```

Funktion: Wandelt einen HexString in eine Longintzahl. Keine 2er-Komplementdarstellung.

```
PROCEDURE BinStringToInt(REF str IN A0 : STRING):LONGINT;
```

Funktion: Wandelt einen BinärString in eine Longintzahl. Keine 2er-Komplementdarstellung

```
PROCEDURE StringToCard(REF str IN A0 : STRING):LONGCARD;
```

Funktion: Wandelt einen String in eine Longcardzahl.

```
PROCEDURE HexStringToCard(REF str IN A0 : STRING):LONGCARD;
```

Funktion: Wandelt einen Hex-String in eine Longcardzahl.

```
PROCEDURE BinStringToCard(REF str IN A0 : STRING):LONGCARD;
```

Funktion: Wandelt einen Binär-String in eine Longcardzahl.

7.6 Dates

Dieses Modul stellt Konvertierungsroutinen zur Umwandlung von Dos-Datumsstrukturen zur Verfügung, so daß die Daten in der uns gebräuchliche Form ausgegeben werden kann. Weiterhin auch Prozeduren, um Daten im Dosformat zu erzeugen.

```
DEFINITION MODULE Dates;
FROM Dos          IMPORT Date;
FROM Exceptions   IMPORT RangeViolation;

EXCEPTION
  NoLeapYear   :  "Only in leap-years there is a 29.th, february";
  TooMuchDays  :  "This month has less days, than passed";
  NotBefore78  :  "Year was earlier than 1978";

TYPE
  Months        = [1..12];
  Days          = [1..31];
  Hours         = [0..23];
  US_Hours      = [1..12];
  Min_Secs      = [0..59];
  WeekDays      = [1..7];
  MonthNameType = ARRAY Months OF STRING(10);
  ShortNameType = ARRAY Months OF STRING(3);
  DayNameType   = ARRAY WeekDays OF STRING(10);
  FullDate      = RECORD
                    year      : [1978..2099];
                    month     : Months;
                    day       : [1..31];
                    dayInWeek : WeekDays;
                    dayInYear : [1..366];
                    hour      : [0..23];
                    us_Hour   : [1..12];
                    pm        : BOOLEAN;
                    minute,
                    second    : [0..59];
                    ticks     : [0..49];
                  END;
```

```
CONST
  US_Months   = MonthNameType:("January","February","March",
                               "April","May","June","July",
                               "August","September","October",
                               "November","December");
  D_Months    = MonthNameType:("Januar","Februar","März",
                               "April","Mai","Juni","Juli",
                               "August","September","Oktober",
                               "November","Dezember");
  ShortNames  = ShortNameType:("Jan","Feb","Mar","Apr","May",
                               "Jun","Jul","Aug","Sep","Okt",
                               "Nov","Dez");
  US_DayNames = DayNameType:("Sunday","Monday","Tuesday",
                             "Wednesday","Thursday","Friday",
                             "Saturday");
  D_DayNames  = DayNameType:("Sonntag","Montag","Dienstag",
                             "Mittwoch","Donnerstag","Freitag",
                             "Samstag");

PROCEDURE FullDateOf(date : Date):FullDate;

PROCEDURE ValToDate(day     : Days;
                    month   : Months;
                    year    : CARDINAL;
                    hour    : Hours;
                    min,
                    second  : Min_Secs):Date;
```



```
CONST
  am  = FALSE; | Vormittag  1-12 Uhr
  pm  = TRUE;  | Nachmittag 13-0 Uhr

PROCEDURE US_ValToDate(month  : Months;
                       day    : Days;
                       year   : CARDINAL;
                       hour   : US_Hours;
                       min,
                       second : Min_Secs;
                       pm     : BOOLEAN:=FALSE):Date;

GROUP
  All    = Months,WeekDays,MonthNameType,ShortNameType,
           DayNameType,FullDate,US_Months,D_Months,ShortNames,
           US_DayNames,D_DayNames,,FullDateOf,ValToDate,Date,
           Dos.DateStamp,US_ValToDate;

END Dates.
```

Exceptions:

NoLeapYear Es wurde in einem Nicht-Schaltjahr ein 29. Februar übergeben.

TooMuchDays Das übergebene Tagesdatum ist größer als der Monat Tage hat.

NotBefore78 Es können keine Daten vor 1978 verarbeitet werden.

Typen: Die einzelnen Elemente von `FullDate` haben folgende Funktionen:

`year`	: Jahr
`month`	: Monat
`day`	: Tag im Monat
`dayInWeek`	: Tag in der Woche
`dayInYear`	: Tag im laufenden Jahr
`hour`	: Stunde im 24 Stunden-Tag
`us_Hour`	: Stunde im anglo-amerikanischen 12 Stunden Tag
`pm`	: `TRUE` für `hour OF 13..0`
`minute`	: Minuten der angebrochenen Stunde
`second`	: Sekunde der angebrochenen Minute
`ticks`	: 1/50 Sekunden der angebrochenen Sekunde

Die Konstanten `US_Months`, `D_Months`, `ShortNames`, `US_DayNames` und `D_DayNames` dienen zur einfachen Konvertierung der Felder `month` und `dayInWeek` zu Strings. Jeweils auf deutsch und auf englisch.


```
PROCEDURE FullDateOf(date : Date):FullDate;
```

Funktion: Wandelt ein Dos-Date in ein `FullDate` um.

Parameter:

`data`	⇐ Dos-Date, das umgewandelt werden soll.
	⇒ FullDate-Strukur, mit dem umgewandelten Datum.

```
PROCEDURE ValToDate(day     : Days;
                    month   : Months;
                    year    : CARDINAL;
                    hour    : Hours;
                    min,
                    second  : Min_Secs):Date;
```

Funktion: Erzeugt aus einem deutschen Standarddatum ein Dos-Date.

Parameter:

`day`	⇐ Tag im Monat
`month`	⇐ Monat
`year`	⇐ Jahr
`hour`	⇐ Stunde
`min`	⇐ Minute
`second`	⇐ Sekunde
	⇒ Dos-Date

```
PROCEDURE US_ValToDate(month  : Months;
                       day    : Days;
                       year   : CARDINAL;
                       hour   : US_Hours;
                       min,
                       second : Min_Secs;
                       pm     : BOOLEAN:=FALSE):Date;
```

Funktion: Erzeugt aus einem anglo-amerikanischen Standarddatum ein Dos-Date.

Parameter:

`day`	⇐	Tag im Monat
`month`	⇐	Monat
`year`	⇐	Jahr
`hour`	⇐	Stunde
`min`	⇐	Minute
`second`	⇐	Sekunde
`pm`	⇐	Flag ob am oder pm
	⇒	Dos-Date

7.7 DosSupport

Dieses Modul bietet eine Vielzahl von Prozeduren und Methoden zur einfachen Handhabung von Dos-Objekten.

```
DEFINITION MODULE DosSupport;

FROM System          IMPORT Regs;
FROM Exceptions      IMPORT RangeViolation;
FROM Resources       IMPORT NotEnoughMemory,ContextPtr;
FROM Lists           IMPORT BiLists,TextLists;
FROM Trees           IMPORT StdTrees;
                     IMPORT Dos;

TYPE
  DrivePtr    = POINTER TO Drive;
  DirNodePtr  = POINTER TO DirNode;
  DirTreePtr  = POINTER TO DirTree;

DEFINITION MODULE DriveLists = TextLists(DrivePtr);

DEFINITION MODULE DirLists   = BiLists(DirNodePtr);

DEFINITION MODULE DirTrees   = StdTrees(DirTreePtr);

TYPE
  Drive         = DriveLists.TextNode;
  DriveList     = RECORD OF DriveLists.TextList
                    context : ContextPtr;
                  END;

  FileData      = RECORD
                    dir    : BOOLEAN;
                    name   : CLASSPTR TO STRING;
                    comment: CLASSPTR TO STRING;
                    flags  : Dos.ProtectionFlagSet;
                    date   : Dos.Date;
                    length : LONGINT;
                  END;
```

```
  DirData       = RECORD
                    size,
                    fullSize    : LONGINT;
                    entries,
                    fullEntries,
                    dirs,
                    fullDirs    : INTEGER;
                  END;
  DirDataPtr    = POINTER TO DirData;
  DirNode       = RECORD OF DirLists.BiNode;
                    data : FileData;
                  END;
  DirTree       = RECORD OF DirTrees.Leaf;
                    data        : FileData;
                    dirData     : DirDataPtr;
                  END;
  Selection     = (selectFiles,selectDirs);
  DirSelectType = SET OF Selection;
  DirList       = RECORD OF DirLists.BiList
                    context : ContextPtr;
                    entries,
                    dirs    : INTEGER;
                    size    : LONGINT;
                  END;
  FileErrors    = (ok,unknownError,objectInUse,objectExists,
                   objectNotFound,objectWrongType,
                   diskNotValidated,writeProtected,
                   renameAcrossDevices,directoryNotEmpty,
                   deviceNotMounted,seekError,diskFull,
                   deleteProtected,readProtected,notADosDisk);
  FileTexts     = ARRAY FileErrors OF POINTER TO STRING;

CONST
  FileErrTexts = FileTexts; | Fehlermeldungen im Klartext

|------ Directory ----------
TYPE
  PutProc  = PROCEDURE(data : FileData);
PROCEDURE ScanDir(    put         : PutProc;
                  REF path        : STRING;
                  REF pattern     : STRING:="#?";
                      type        := DirSelectType:
                                       {selectFiles,
                                        selectDirs});


METHOD Get(VAR data       : FileData;
           REF path       : STRING)
```



```
METHOD Delete(VAR list : DirList);


METHOD Get(VAR list        : DirList;
           REF path        : STRING;
           REF pattern     : STRING:="#?";
               type        := DirSelectType:
                                {selectDirs,
                                 selectFiles};
               context     : ContextPtr := NIL);


METHOD Delete(VAR tree : DirTreePtr);

METHOD Get(VAR tree        : DirTreePtr;
           REF path        : STRING;
           REF pattern     : STRING:="#?";
               type        := DirSelectType:
                                {selectDirs,
                                 selectFiles};
               context     :  ContextPtr:=NIL);

TYPE
  CmpProc = PROCEDURE(REF d1,d2 : FileData):BOOLEAN;

PROCEDURE byName(REF d1,d2 : FileData):BOOLEAN;

PROCEDURE bySize(REF d1,d2 : FileData):BOOLEAN;

PROCEDURE byDate(REF d1,d2 : FileData):BOOLEAN;

METHOD Sort(VAR list : DirList;cmp : CmpProc);

METHOD Sort(father : DirTreePtr;cmp : CmpProc := byName);


PROCEDURE Exists(REF name : STRING;
                     retry : BOOLEAN := FALSE ):BOOLEAN;

PROCEDURE IsFile(REF name : STRING):BOOLEAN;
```

```
METHOD Get(VAR list     : DriveList;
               physical : BOOLEAN := TRUE;
               context  : ContextPtr := NIL);


METHOD Delete(VAR list : DriveList);

GROUP
  DirGrp  = DirTree,DirTreePtr,byName,bySize,byDate,
            DirNodePtr,DirNode,Selection,DirSelectType,
            Drive,DriveList,DrivePtr,DirList,
            ScanDir,FileData;

|----- Path ---------------


PROCEDURE SplitName(VAR path,name,extension : STRING);


PROCEDURE GetPathName(VAR path IN A0,name IN A1 : STRING);

$$OwnHeap:=TRUE
PROCEDURE Name(REF path IN A0 : STRING):STRING;

$$OwnHeap:=TRUE
PROCEDURE ExtendedName(REF path IN A0 : STRING):STRING;

$$OwnHeap:=TRUE;
PROCEDURE Path(REF fullPath IN A0 : STRING):STRING;

$$OwnHeap:=TRUE
PROCEDURE Extension(REF name IN A0 : STRING):STRING;
PROCEDURE GetSubDir(VAR path : STRING;
                    REF subDir : STRING);

$$OwnHeap:=TRUE;
PROCEDURE SubDir(REF path,subDir : STRING):STRING;

PROCEDURE GetParentDir(VAR path IN A0 : STRING);

$$OwnHeap:=TRUE
PROCEDURE ParentDir(REF path IN A0 : STRING):STRING;
```



```
PROCEDURE BuildName(VAR path      : STRING;
                    REF name,
                        extension : STRING);

$$OwnHeap:=TRUE
PROCEDURE FName(REF path,name,extension : STRING):STRING;


PROCEDURE GetFullPath(REF smallPath : STRING;
                      VAR path : STRING);

$$OwnHeap:=TRUE
PROCEDURE FullPath(REF smallPath : STRING):STRING;

PROCEDURE BuildConsolePath(VAR title      : STRING;
                               leftEdge,
                               topEdge,
                               width,
                               height     : CARDINAL);
$$OwnHeap:=TRUE
PROCEDURE ConsolePath(REF title     : STRING;
                          leftEdge,
                          topEdge,
                          width,
                          height    : CARDINAL):STRING;

GROUP
  PathGrp   = SplitName,GetFullPath,GetPathName,SubDir,
              GetSubDir,GetParentDir,ParentDir,BuildName,
              FName,FullPath,BuildConsolePath,ConsolePath,
              Path,Name,ExtendedName,Extension;
```

```
|----- Global files -------

PROCEDURE Delete(REF name : STRING;
                     all  : BOOLEAN := FALSE);


PROCEDURE MakeDir(REF name : STRING);

PROCEDURE Rename(REF oldName,
                     newName  : STRING)


TYPE
  CopyProc    = PROCEDURE(VAR name : STRING;
                              dir  : BOOLEAN);
  CheckProc   = PROCEDURE(REF path,
                              name : STRING):BOOLEAN;

PROCEDURE Copy(src,
               dest         : STRING;
               all          : BOOLEAN   :=FALSE;
               overWrite    : CheckProc :=NIL;
               apply        : CopyProc  :=NIL);
               clone        : BOOLEAN   :=TRUE);

PROCEDURE Move(REF src,dest : STRING);


TYPE
  SearchProc = PROCEDURE(REF path : STRING;
                             data : FileData);

PROCEDURE Search(REF startDir,
                     name       : STRING;
                     react      : SearchProc);
```



```
TYPE
  WorkProc   = PROCEDURE(REF path,name : STRING;
                             data      : FileData);
PROCEDURE WorkOnFiles(REF path,pattern : STRING;
                          apply        : WorkProc;
                          recursive    : BOOLEAN:=FALSE;
                          type         := DirSelectType:
                                              {selectDirs,
                                               selectFiles};
                          scanFirst    : BOOLEAN:=FALSE);

PROCEDURE SetProtect(REF name  : STRING;
                         flags : Dos.ProtectionFlagSet);

PROCEDURE GetProtect(REF name  : STRING;
                     VAR flags : Dos.ProtectionFlagSet);

 |---- Utilities -----------
TYPE
  WindowPtr = DEFERRED POINTER Intuition.WindowPtr;
PROCEDURE GetSystemRequest(REF window : WindowPtr);

PROCEDURE RestoreSystemRequest;

PROCEDURE SysReqOff;

PROCEDURE SysReqOn;

PROCEDURE FileError2(error : LONGINT):FileErrors;

(* $E-*)
PROCEDURE FileError():FileErrors;


GROUP
  GlobalGrp    = Delete,Rename,Copy,WorkOnFiles,
                 SetProtect,GetProtect,FileError,
                 FileError2,FileErrors,FileErrTexts,
                 Search;
  All          = DirGrp,PathGrp,GlobalGrp;

END DosSupport.
```

Exceptions:

RangeViolation Falls diese Exception von einer Prozedur in diesem Modul ausgelöst wurde, war wahrscheinlich eine der Pfad-Manipulations-Prozeduren die Ursache, da der zurückgegebene Pfad nicht in die Variable gepaßt hat.

Im Prinzip können in diesem Modul fast alle Exceptions aus dem Modul `DosSupport` ausgelöst werden. Ihre Beschreibung entspricht denen der FileError-Group, deren Elemente den gleichen Namen wie die Exception haben. Siehe dort.

Typen:

Drive, DriveList Generische Erweiterung von `DriveLists.TextNode` und `DriveList.TextLists`, auf sie sind also auch alle Methoden dieses Generischen Moduls anwendbar. Bei DriveList handelt es sich um eine alphabetisch geordnete Liste der Laufwerke. Dabei gibt das `special`-Feld der Textnode an, ob es sich um ein physikalisches oder um ein logisches Laufwerk handelt.

FileData Ein Datenblock, der einen Verzeichniseintrag näher beschreibt. Die einzelnen Felder sind:

- `dir` : Gibt an, ob es sich bei dem Eintrag um ein Verzeichnis handelt.
- `name` : Name des Eintrags.
- `comment` : Kommentar des Eintrags.
- `flags` : Protectionflags des Eintrags. Beschreibung der einzelnen Flags siehe Kapitel 5.4.3
- `date` : Datum der Erzeugung des Eintrags.
- `length` : Länge des Eintrags, falls es ein File ist.

DirData Datenblock, der nähere Information zu Verzeichnissen beinhaltet:

- `size` : Länge aller Einträge dieses Verzeichnisses in Bytes.
- `fullSize` : Länge aller Einträge dieses Verzeichnisses, inklusive der Länge aller Unterverzeichnisse, die darin enthalten sind.
- `entries` : Anzahl der Einträge dieses Verzeichnisses. Files und Directories
- `fullEntries` : Anzahl aller Einträge inklusive der Anzahl der Einträge eventueller Unterverzeichnisse.
- `dirs` : Anzahl der Unterverzeichnisse in diesem Verzeichnis.
- `fullDirs` : Anzahl der Unterverzeichnisse in diesem Verzeichnis, inklusive aller Unterverzeichnisse in Unterverzeichnissen.

DirTree Knoten eines Verzeichnisbaumes, ebenfalls eine Generische Ausprägung, es können also alle Methoden für Bäume auch auf diesen angewandt werden. Jeder Knoten enthält dabei einen FileData-Eintrag, und falls es sich um ein Verzeichnis handelt, auch einen Zeiger auf einen DirData-Block.

DirList Generisch ausgeprägte Liste für Elemente eines Verzeichnisses:

- `context` : Zeiger auf den Kontext, zu dem die Einträge alloziert wurden. Auf dieses Feld sollte nicht zugegriffen werden, sondern die entsprechende Freigabeprozedur verwendet werden.
- `entries` : Anzahl der Einträge.
- `dirs` : Anzahl der Unterverzeichnisse.
- `size` : Summe der Größen aller Einträge in Bytes, nicht rekursiv.

FileErrors Aufzählungstyp aller Rückgabewerte, die bei Dos-Operationen vorkommen können. Die meisten können, wenn man `T_Dos` verwendet, auch als Exceptions auftreten. Zum Abfangen eignen sich die Exceptiongruppen aus `T_Dos` besonders gut. Die Elemente im einzelnen:

`ok` : Es trat kein Fehler auf.

`unknownError` : Es trat ein Fehler auf, der nicht einem der anderen Fehler zuzuordnen ist.

`objectInUse` : Tritt beim Löschen eines Files oder Verzeichnisses auf, wenn dieses noch geöffnet ist oder darauf noch ein Lock existiert.

`objectExists` : Es wurde versucht, ein Verzeichnis zu erzeugen, das schon existiert.

`objectNotFound` : Angefordertes Dos-Object wurde nicht gefunden.

`objectWrongType` : Objekt hat nicht den richtigen Typ für diese Operation.

`diskNotValidated` : Es wurde schreibend versucht, auf einen nicht validierten Datenträger zuzugreifen.

`writeProtected` : Es wurde schreibend versucht, auf ein schreibgeschütztes File oder Datenträger zuzugreifen.

`renameAcrossDevices` : Es wurde bei einem Rename als Ziel ein anderes Laufwerk angegeben als bei der Quelle.

`directoryNotEmpty` : Es wurde versucht, ein nicht leeres Verzeichnis zu löschen.

`deviceNotMounted` : Es wurde auf ein Device zugegriffen, das nicht angemeldet ist. Ist meistens auf einen Schreibfehler zurückzuführen.

`seekError` : Bei einer Seekoperation trat ein Fehler auf.

`diskFull` : Es wurde versucht, auf eine volle Diskette zu schreiben.

`deleteProtected` : Es wurde versucht, ein löschgeschütztes File zu löschen.

`readProtected` : Es wurde versucht, von einem lesegeschützten File zu lesen.

`notADosDisk` : Es wurde auf eine Diskette zugegriffen, die nicht dem Format des Amiga-Dos entspricht.

Konstanten:

FileErrTexts Array, das als IndexTyp den Aufzählungstyp `FileErrors` hat, und dessen Einträge jeweils eine Klartextmeldung der entsprechenden Dos-Fehlernummer enthalten. Um z. B. die Meldung zu `diskNotValidated` zu bekommen, müssen Sie nur schreiben:

```
WriteString(FileErrTexts[diskNotValidated]);
```


7.7.1 Directory Funktionen

```
TYPE
  PutProc  = PROCEDURE(data : FileData);

PROCEDURE ScanDir(     put         : PutProc;
                   REF path        : STRING;
                   REF pattern     : STRING:="#?";
                       type        := DirSelectType:
                                        {selectFiles,
                                         selectDirs});
```

Funktion: Diese Funktion stellt wohl die flexibelste Funktion dar, ein Verzeichnis zu untersuchen. Sie geht alle Einträge eines Verzeichnisses durch und ruft für jeden Eintrag, auf den das Pattern zutrifft, eine übergebene Prozedur auf, der die Eintragsdaten übergeben werden.

Parameter:

`put`	⇐	Prozedur, die bei jedem Eintrag aufgerufen werden soll.
`path`	⇐	Pfad des Verzeichnisses, das untersucht werden soll.
`pattern`	⇐	Dos-Pattern, das angibt, welche Einträge untersucht werden sollen.
`type`	⇐	Set, der angibt, ob nur File-, nur Verzeichniseinträge oder beide untersucht werden sollen.

```
METHOD Get(VAR data       : FileData;
           REF path       : STRING)
```

Funktion: Untersucht ein File/Directory und füllt dazu einen FileDatablock aus.

Parameter:

`data`	⇐	FileDatablock, in den die Daten eingetragen werden sollen.
`path`	⇐	Pfad des Eintrags, der untersucht werden soll.

```
METHOD Delete(VAR list : DirList);
```

Funktion: Gibt eine DirListe frei.

Parameter:

`list`	⇐	freizugebende Liste


```
METHOD Get(VAR list       : DirList;
           REF path       : STRING;
           REF pattern    : STRING:="#?";
               type       := DirSelectType:
                               {selectDirs,
                                selectFiles};
               context    : ContextPtr := NIL);
```

Funktion: Untersucht ein Directory und gibt eine Liste der Einträge zurück.

Parameter:

`list`	⇒	Liste, in die die Einträge eingehängt werden sollen.
`path`	⇐	Pfad des Directorys, das untersucht werden soll.
`pattern`	⇐	Dos-Pattern der Einträge, die in die Liste aufgenommen werden sollen.
`type`	⇐	Gibt an, ob nach Files, Directories oder nach beiden gesucht werden soll.
`context`	⇐	Kontext, zu dem der Speicher für die Einträge alloziert werden sollen. Standardmäßig wird der aktuelle Kontext verwendet.

```
METHOD Delete(VAR tree : DirTreePtr);
```

Funktion: Gibt einen DirTree frei.

Parameter:

`tree`	⇐	freizugebender Baum.

```
METHOD Get(VAR tree       : DirTreePtr;
           REF path       : STRING;
           REF pattern    : STRING:="#?";
               type       := DirSelectType:
                               {selectDirs,
                                selectFiles};
               context    :  ContextPtr:=NIL);
```

Funktion: Untersucht ein Directory und die darin enthaltenen Einträge und gibt einen Baum der Einträge zurück.

Parameter:

`tree`	⇒	Baum, in den die Einträge eingehängt werden sollen.
`path`	⇐	Pfad des Directorys, das untersucht werden soll.
`pattern`	⇐	Dos-Pattern der Einträge, die in die Liste aufgenommen werden sollen.
`type`	⇐	Gibt an, ob Files,Directories oder beide in den Baum aufgenommen werden sollen.
`context`	⇐	Kontext, zu dem der Speicher für die Einträge alloziert werden soll. Standardmäßig wird der aktuelle Kontext verwendet.

```
TYPE
  CmpProc = PROCEDURE(REF d1,d2 : FileData):BOOLEAN;
```

```
PROCEDURE byName(REF d1,d2 : FileData):BOOLEAN;
```

```
PROCEDURE bySize(REF d1,d2 : FileData):BOOLEAN;
```

```
PROCEDURE byDate(REF d1,d2 : FileData):BOOLEAN;
```

Funktion: Diese drei Funktionen entsprechen alle dem Typ `CmpProc`, und dienen dazu den folgenden Sortprozeduren übergeben zu werden. Dabei sortiert `byName` nach Namen, `bySize` nach Größe und `byDate` nach Datum.

```
METHOD Sort(VAR list : DirList;cmp : CmpProc);
```


Funktion: Sortiert eine DirList abhängig von einer übergebenen Vergleichsprozedur.

Parameter:

`list`	⇔	Liste, die sortiert werden soll.
`cmp`	⇐	Vergleichsprozedur, nach der aufsteigend sortiert werden soll. In den meisten Fällen muß man nur eine der drei oben vordefinierten Funktionen hier eintragen.

Beispiel:

```
TestListe.Sort(byName);
```

```
METHOD Sort(father : DirTreePtr;cmp : CmpProc := byName);
```

Funktion: Sortiert einen DirTree nach obenstehenden Kriterien.

Parameter:

`father`	⇐	Knoten, von dem ab sortiert werden soll.
`cmp`	⇐	Vergleichsprozedur, nach der aufsteigend sortiert werden soll.

```
PROCEDURE Exists(REF name  : STRING;
                     retry : BOOLEAN := FALSE ):BOOLEAN;
```

Funktion: Prüft ob ein File/Verzeichnis, dessen Namen übergeben worden ist, existiert.

Parameter:

`name`	⇐	Name des Files/Verzeichnisses, das überprüft werden soll.
`retry`	⇐	Flag, ob im Falle, daß sich der Name auf einem nicht im Laufwerk liegenden Volume befindet, ein Systemrequester zum Einlegen der Diskette auffordern soll.
	⇒	`TRUE` wenn das Verzeichnis/File existiert.

```
PROCEDURE IsFile(REF name : STRING):BOOLEAN;
```

Funktion: Prüft, ob ein übergebener Pfad ein File bezeichnet.

Parameter:

`name` ⇐ Pfad, der überprüft werden soll.

⇒ `TRUE`, wenn es sich um ein File handelt.

```
METHOD Get(VAR list     : DriveList;
               physical : BOOLEAN := TRUE;
               context  : ContextPtr := NIL);
```

Funktion: Füllt eine übergebene Liste mit Einträgen der angeschlossenen Laufwerke (special-Feld der Node = `TRUE`), sowie der logischen Laufwerke (special = `FALSE`).

Parameter:

`list` ⇒ Liste, in die die Einträge eingehängt werden sollen.

`physical` ⇐ Flag, das angibt, ob nur physikalische Laufwerke in die Liste aufgenommen werden sollen.

```
METHOD Delete(VAR list : DriveList);
```

Funktion: Gibt eine gefüllte DriveList wieder frei.

Parameter:

`list` ⇐ Liste, die freigegeben werden soll.

7.7.2 Pfadoperationen

```
PROCEDURE SplitName(VAR path,name,extension : STRING);
```

Funktion: Zerlegt einen Filenamen in Pfad, Name und Erweiterung (Pfad/Name.Erweiterung)

Parameter:

`path` ⇐ Beim Aufruf der gesamte Filenamen.

⇒ Nach Prozedurende der reine Pfad des Files.

`name` ⇒ Name des Files.

`extension` ⇒ Erweiterung nach „.“ am Namensende z. B. „.iff“.

```
PROCEDURE GetPathName(VAR path IN A0,name IN A1 : STRING);
```

Funktion: Zerlegt einen String mit kombiniertem Pfad + Filenamen in einen getrennten Pfad und einen Filenamen.

Parameter:

`path` ⇐ Pfad kombiniert.

⇒ Reiner Pfad.

`name` ⇒ Abgtrennter Name.

```
$$OwnHeap:=TRUE
PROCEDURE Name(REF path IN A0 : STRING):STRING;
```

Funktion: Gibt den in einem Pfad enthaltenen Dateinamen ohne angehängte Erweiterung zurück.

Parameter:

`path` ⇐ Pfad.

⇒ Name.

```
$$OwnHeap:=TRUE
PROCEDURE ExtendedName(REF path IN A0 : STRING):STRING;
```

Funktion: Gibt den in einem Pfad enthaltenen Dateinamen, inklusive einer eventuell angehängten Erweiterung zurück.

Parameter:

`path`	⇐ Pfad.
	⇒ Name.

```
$$OwnHeap:=TRUE;
PROCEDURE Path(REF fullPath IN A0 : STRING):STRING;
```

Funktion: Gibt den in einem Pfad enthaltenen reinen Pfad zurück.

Parameter:

`fullPath`	⇐ Kompletter Pfad mit Dateinamen.
	⇒ Reiner Pfad.

```
$$OwnHeap:=TRUE
PROCEDURE Extension(REF name IN A0 : STRING):STRING;
```

Funktion: Gibt die in einem Pfad enthaltene Erweiterung zurück.

Parameter:

`name`	⇐ Kompletter Pfad.
	⇒ Darin enthaltene Erweiterung.

```
PROCEDURE GetSubDir(VAR path : STRING;
                    REF subDir : STRING);
```

Funktion: Fügt an einen Pfad ein Unterverzeichnis an. Dabei wird darauf geachtet, daß kein „/“ zuviel eingefügt wird.

Parameter:

`path`	⇔ Pfad, an den ein Subdir angefügt werden soll.
`subDir`	⇐ Namen des Verzeichnisses, das angefügt werden soll.


```
$$OwnHeap:=TRUE;
PROCEDURE SubDir(REF path,subDir : STRING):STRING;
```

Funktion: Wie GetSubDir, jedoch als Funktion.

```
PROCEDURE GetParentDir(VAR path IN A0 : STRING);
```

Funktion: Liefert den Pfad des übergeordneten Directorys.

Parameter:

`pfad`	⇔ Pfad, von dem das ParentDirectory ermittelt werden soll. Enthält nach dem Aufruf den Pfad des übergeordneten Verzeichnisses.

```
$$OwnHeap:=TRUE
PROCEDURE ParentDir(REF path IN A0 : STRING):STRING;
```

Funktion: Wie GetParentDir, jedoch als Funktion.

```
PROCEDURE BuildName(VAR path      : STRING;
                    REF name,
                        extension : STRING);
```

Funktion: Setzt aus einem Pfad, einem Namen und einer Erweiterung einen kompletten Pfad zusammen.

Parameter:

`path`	⇔ einzelner Pfad/zusammengesetzter Pfad.
`name`	⇐ Name.
`extension`	⇐ Erweiterung.

```
$$OwnHeap:=TRUE
PROCEDURE FName(REF path,name,extension : STRING):STRING;
```

Funktion: Wie BuildName, jedoch als Funktion.

```
PROCEDURE GetFullPath(REF smallPath : STRING;
                      VAR path : STRING);
```

Funktion: Gibt, nach Übergabe eines zum aktuellen Verzeichnis relativen Pfades, einen vollständigen Pfad zurück (einschließlich Volume-Namen).

Parameter:

`smallPath`	⇐	Name des UnterVerzeichnisses.
`path`	⇒	Kompletter Pfad.

```
$$OwnHeap:=TRUE
PROCEDURE FullPath(REF smallPath : STRING):STRING;
```

Funktion: Wie GetFullPath, jedoch als Funktion.

```
PROCEDURE BuildConsolePath(VAR title      : STRING;
                               leftEdge,
                               topEdge,
                               width,
                               height     : CARDINAL);
```

Funktion: Setzt aus den übergebenen Werten einen Pfad zusammen, wie man ihn zum Öffnen eines Consol-Fensters benötigt.

Parameter:

`title`	⇔	Titel des Fensters/zusammengesetzter Pfad.
`leftEdge`	⇐	x Position der linken, oberen Ecke des Fensters.
`topEdge`	⇐	y Position der linken, oberen Ecke des Fensters.
`width`	⇐	Breite des Fensters.
`heigth`	⇐	Höhe des Fensters.

```
$$OwnHeap:=TRUE
PROCEDURE ConsolePath(REF title     : STRING;
                          leftEdge,
                          topEdge,
                          width,
                          height    : CARDINAL):STRING;
```

Funktion: Wie BuilConsolePath, jedoch als Funktion.

7.7.3 Globale Filefunktionen

```
PROCEDURE Delete(REF name : STRING;
                     all  : BOOLEAN := FALSE);
```

Funktion: Löscht ein File/Verzeichnis.

Parameter:

`name`	⇐	Pfad des Files/Directories.
`all`	⇐	Gibt an, ob auch Unterverzeichnisse rekursiv gelöscht werden sollen. Hier ist äußerste Vorsicht geboten!

```
PROCEDURE MakeDir(REF name : STRING);
```

Funktion: Erzeugt ein Verzeichnis.

Parameter:

`name`	⇐	Pfad des neuen Verzeichnisses.

```
PROCEDURE Rename(REF oldName,
                     newName  : STRING)
```

Funktion: Benennt ein File um. Dabei muß sich der neue Name auf dem selben Gerät befinden wie der alte. Jedoch kann der neue Pfad innerhalb eines Gerätes in einem anderen Verzeichnis liegen als der alte.

Parameter:

`oldName`	⇐	Alter Name des Files.
`newName`	⇐	Neuer Name des Files.

```
TYPE
  CopyProc    = PROCEDURE(VAR name : STRING;
                              dir  : BOOLEAN);
  CheckProc   = PROCEDURE(REF path,
                              name : STRING):BOOLEAN;

PROCEDURE Copy(src,
               dest        : STRING;
               all         : BOOLEAN   :=FALSE;
```

```
                overWrite    : CheckProc :=NIL;
                apply        : CopyProc  :=NIL);
                clone        : BOOLEAN   :=TRUE);
```

Funktion: : Kopiert ein oder mehrere Files oder Verzeichnisse. Der Quellpfad, darf dabei ein Dos-Pattern enthalten z. B. „`#?.(def|mod)`“.

Parameter:

`src`	⇐	Quellpfad.
`dest`	⇐	Zielpfad.
`all`	⇐	Flag, das bestimmt, ob bei der Kopie eines Verzeichnisses darin liegende Verzeichnisse rekursiv mitkopiert werden sollen.
`overWrite`	⇐	Funktion, die mit Pfad und Namen des Files aufgerufen wird, wenn ein File gleichen Namens im Zielverzeichnis schon enthalten ist. Sie muß einen BOOLEAN-Wert zurückgeben, ob das File überschrieben werden soll. Ist existCheck = NIL, werden eventuell existierende Files gleichen Namens überschrieben. Achtung, von „`overWrite`“ darf weder Pfad noch Name verändert werden.
`apply`	⇐	Prozedur, der der Namen des als nächsten zu kopierenden Files/Verzeichnisses, sowie ein Flag, das angibt, ob es sich um ein Verzeichnis handelt, übergeben wird. Verändert „`apply`“ den Namen, wird beim Schreiben des Files dieser neue Namen verwendet.
`clone`	⇐	Flag, das angibt, ob beim Kopieren auch das Filedatum mitkopiert werden soll.


```
PROCEDURE Move(REF src,dest : STRING);
```

Funktion: Verschiebt ein File/Verzeichnis in ein anderes Verzeichnis, auch von einem Laufwerk auf ein anderes. Achtung, diese Prozedur ist mit Vorsicht zu genießen, da bei einem Move von einem Laufwerk auf ein anderes erst kopiert und dann gelöscht wird. Tritt beim Kopieren ein Fehler auf, kann nicht garantiert werden, daß in diesem Fall das Original nicht gelöscht wird.

Parameter:

`src`	⇐	File/Verzeichnis, das verschoben werden soll.
`dest`	⇐	Ziel der Operation.

```
TYPE
  SearchProc = PROCEDURE(REF path : STRING;
                             data : FileData);

PROCEDURE Search(REF startDir,
                     name      : STRING;
                     react     : SearchProc);
```

Funktion: Sucht ein File/Verzeichnis im Verzeichnis „`startDir`“ und dessen Unterverzeichnissen. Wird ein Eintrag mit passendem Namen gefunden, wird eine Prozedur aufgerufen, der der Pfad dieses Eintrages, sowie sein FileData-Block übergeben wird.

Parameter:

`startDir`	⇐	Name des Verzeichnisses, in dem gesucht werden soll.
`name`	⇐	Name oder Pattern, nach dem gesucht werden soll.
`react`	⇐	Prozedur, die beim Auffinden des gesuchten Namens aufgerufen werden soll.

```
TYPE
  WorkProc  = PROCEDURE(REF path,name : STRING;
                            data      : FileData);

PROCEDURE WorkOnFiles(REF path,pattern : STRING;
                          apply        : WorkProc;
                          recursive    : BOOLEAN:=FALSE;
                          type         := DirSelectType:
                                            {selectDirs,
                                             selectFiles};
                          scanFirst    : BOOLEAN:=FALSE);
```

Funktion: : Geht eine Verzeichnisstruktur rekursiv durch, und ruft für jeden Eintrag die Prozedur „apply" auf.

Parameter:

`path` ⇐ Pfad des Verzeichnisses, das bearbeitet werden soll.

`pattern` ⇐ Pattern für die Einträge, für die „`apply`" aufgerufen werden soll. Wird kein Pattern übergeben, werden alle Einträge bearbeitet.

`recursive` ⇐ Gibt an, ob auch in dem Verzeichnis liegende Unterverzeichnisse rekursiv untersucht werden sollen.

`apply` ⇐ Prozedur, die für jeden Eintrag aufgerufen wird, auf den „`pattern`" paßt. Ihr wird der Name des Eintrags, sein Pfad sowie dessen FileData-Block übergeben. Sie darf den Pfad und den Namen nicht verändern.

`type` ⇐ Gibt an, ob nur Files, nur Directories oder beide bearbeitet werden sollen.

`scanfirst` ⇐ Flag, das angibt, ob die Prozedur erst den gesamten Verzeichnisbaum durchgehen, sich die Einträge die in Frage kommen merken und danach bearbeiten soll, oder ob die Bearbeitung gleich beim Scanlauf geschehen soll. In manchen Fällen, kommt man ohne `scanFirst=TRUE` nicht aus.

```
PROCEDURE SetProtect(REF name  : STRING;
                         flags : Dos.ProtectionFlagSet);
```

Funktion: Setzt Protectionbits für ein File.

Parameter:

`name` ⇐ Name des Files.

`flags` ⇐ Neue Protectionbits.

```
PROCEDURE GetProtect(REF name  : STRING;
                     VAR flags : Dos.ProtectionFlagSet);
```

Funktion: Ließt Protectionbits eines Files.

Parameter:

`name` ⇐ Name des Files.

`flags` ⇐ Flags des Files.

7.7.4 Utilities

PROCEDURE `GetSystemRequest(`**REF** `window : WindowPtr);`

Funktion: Leitet Dos-Requester (z. B. Aufforderung zum Einlegen einer Diskette) auf den Screen mit dem angegebenen Fenster um.

Parameter:

`window` ⇐ Zeiger auf das Window, auf dessen Screen der Reqeuster erscheinen soll.

PROCEDURE `RestoreSystemRequest;`

Funktion: Sorgt dafür, daß Dos-Requester wieder auf der Screen erscheinen, die beim Programmbeginn eingetragen war.

PROCEDURE `SysReqOff;`

Funktion: Sorgt dafür, daß im Falle eines vom User korrigierbaren Dos-Errors (z. B. Disk is writeProtected) kein Systemrequester erscheint, sondern die Fehlermeldung unmittelbar vom Programm behandelt werden kann.

PROCEDURE `SysReqOn;`

Funktion: Sorgt dafür, daß Dos-Errors wieder mittles Systemrequestern behandelt werden. Achtung: Rufen Sie SysReqOn nicht auf, wenn Sie davor nicht SysReqOff benutzt haben.

PROCEDURE `FileError2(error : LONGINT):FileErrors;`

Funktion: Wandelt eine Dos-Fehlernummer in einen FileError um.

Parameter:

`error` ⇐ DosFehlernummer.

⇒ FileError.

PROCEDURE `FileError():FileErrors;`

Funktion: Gibt den zuletzt aufgetretenen Dos-Fehler als FileError

zurück.

Parameter:

⇒ Zuletzt aufgetretener Fehler.

7.8 DynamicArrays

Dieses generische Modul dient zur Verwaltung von dynamischen, also wachsenden Arrays. Dabei müssen die Elemente, die eingetragen werden, keine Nachfolger eines speziellen Typs sein.

```
DEFINITION MODULE DynamicArrays;

  EXCEPTION BadSubscript : "Bad subscript";

  DEFINITION MODULE DynamicArray(type : ANYPTR);

    TYPE
      Array = RECORD
                data   : ARRAY [256] OF POINTER TO
                         ARRAY [256] OF type;
                first,
                last   : CARDINAL;
              END;


    PROCEDURE Init(VAR a : Array);

    PROCEDURE Destruct(VAR a : Array);

    PROCEDURE Delete(VAR a : Array);


    PROCEDURE Put(VAR a     : Array;
                      index : CARDINAL;
                      data  : type);

    PROCEDURE Get(REF a     : Array;
                      index : CARDINAL):type;
    PROCEDURE Next(REF a   : Array;
                       old : CARDINAL):CARDINAL;

    PROCEDURE Prev(REF a   : Array;
                       old : CARDINAL):CARDINAL;

  END DynamicArray;


  DEFINITION MODULE DynamicArray2(type : ANYPTR);

    TYPE
```



```
      Array    = RECORD
                   data    : ARRAY [16],[16] OF POINTER TO
                             ARRAY [16],[16] OF POINTER TO
                             ARRAY [16],[16] OF POINTER TO
                             ARRAY [16],[16] OF type;
                   firstX,
                   firstY,
                   lastX,
                   lastY   : CARDINAL;
                 END;

    PROCEDURE Init(VAR a : Array);

    PROCEDURE Destruct(VAR a : Array);

    PROCEDURE Delete(VAR a : Array);

    PROCEDURE Put(VAR a     : Array;
                      x,
                      y     : CARDINAL;
                      data  : type);

    PROCEDURE Get(REF a : Array;
                      x,
                      y : CARDINAL):type;

    PROCEDURE FirstXatY(REF a : Array;
                            y : CARDINAL):CARDINAL;

    PROCEDURE LastXatY(REF a : Array;
                           y : CARDINAL):CARDINAL;

    PROCEDURE FirstYatX(REF a : Array;
                            x : CARDINAL):CARDINAL;

    PROCEDURE LastYatX(REF a : Array;
                           x : CARDINAL):CARDINAL;
```

```
    PROCEDURE NextX(REF a : Array;
                        x,
                        y : CARDINAL):CARDINAL;
    PROCEDURE NextY(REF a : Array;
                        x,
                        y : CARDINAL):CARDINAL;
    PROCEDURE PrevX(REF a : Array;
                        x,
                        y : CARDINAL):CARDINAL;
    PROCEDURE PrevY(REF a : Array;
                        x,
                        y : CARDINAL):CARDINAL;
  END DynamicArray2;
END DynamicArrays.
```

Exceptions:

BadSubscript Diese Exception wird ausgelöst, wenn man am Anfang oder Ende eines Arrays nach dem – nicht vorhandenen – vorherigen oder nächsten belegten Eintrag sucht.

7.8.1 DynamicArray

Dieses Modul stellt Methoden zur Bearbeitung von eindimensionalen Arrays variabler Größe zur Verfügung. Die Größe ist dabei nicht wirklich beliebig, sondern Arrays können maximal 65536 Elemente enthalten.

```
PROCEDURE Init(VAR a : Array);
```

Funktion: Initialisiert ein Array. Muß aufgerufen werden, bevor man ein Array verwenden kann.

Parameter:

`a` ⇒ Array, das initialisiert werden soll.

```
PROCEDURE Destruct(VAR a : Array);
```

Funktion: Zerstört die Arraystruktur; die darin enthaltenen Elemente bleiben jedoch erhalten.

Parameter:

`a` ⇔ Array, das zerstört werden soll.

```
PROCEDURE Delete(VAR a : Array);
```

Funktion: Zerstört die Arraystruktur; die darin enthaltenen Elemente werden ebenfalls mit „`Dispose`“ freigegeben.

Parameter:

`a` ⇔ Array, das gelöscht werden soll.

```
PROCEDURE Put(VAR a     : Array;
                  index : CARDINAL;
                  data  : type);
```

Funktion: Trägt einen Wert in das Array ein. Erst durch diese Anweisung wird das entsprechende Feld erzeugt. Somit belegen auch große Arrays, die nur wenig gefüllt sind, nur wenig Speicher.

Parameter:

`a`	⇐	Array, in das der Wert eingetragen werden soll.
`index`	⇐	Index des Array-Feldes, in das der Wert geschrieben werden soll.
`data`	⇐	Zeiger auf den Wert, der eingetragen werden soll. Dabei stellt das generische Konzept sicher, daß nur Elemente eines Typs in ein Array gelangen können.

```
PROCEDURE Get(REF a     : Array;
                  index : CARDINAL):type;
```

Funktion: Liest den Wert eines Array Feldes aus.

Parameter:

`a`	⇐	Array, aus dem gelesen werden soll.
`index`	⇐	Nummer des Feldes, das ausgelesen werden soll.
	⇒	Zeiger auf den Wert dieses Feldes. War unter diesem Index noch kein Wert eingetragen, wird `NIL` zurückgegeben.


```
PROCEDURE Next(REF a   : Array;
                   old : CARDINAL):CARDINAL;
```

Funktion: Da ein solches dynamisches Array sehr groß und trotzdem nur zum Teil gefüllt sein kann, bietet diese Funktion die Möglichkeit, den nächsten Index zu ermitteln, an dem ein Wert eingetragen ist.

Parameter:

`a`	⇐	Array, in dem der nächste Eintrag gesucht werden soll.
`old`	⇐	Position, ab der gesucht werden soll. Liegt `old` hinter dem letzten gültigen Eintrag, wird eine Exception ausgelöst.
	⇒	Index des nächsten Eintrags, der einen Wert enthält.

```
PROCEDURE Prev(REF a   : Array;
                   old : CARDINAL):CARDINAL;
```

Funktion: Ermittelt den vorherigen Index, an dem ein Wert eingetragen ist.

Parameter:

`a`	⇐	Array, in dem der vorherige Eintrag gesucht werden soll.
`old`	⇐	Position, ab der gesucht werden soll. Liegt `old` vor dem ersten gültigen Eintrag, wird eine Exception ausgelöst.
	⇒	Index des vorherigen Eintrags, der einen Wert enthält.

7.8.2 DynamicArray2

Dieses Modul stellt Methoden zur Bearbeitung von zweidimensionalen Arrays variabler Größe zur Verfügung. Die Größe ist dabei nicht wirklich beliebig, sondern Arrays können maximal 4.294.967.296 Elemente enthalten.

Die Funktionen `Init`, `Destruct` und `Delete` entsprechen in ihrere Funktion und Parametern den Funktionen aus `DynamicArrays`, siehe dort.

```
PROCEDURE Put(VAR a     : Array;
                  x,
                  y     : CARDINAL;
                  data  : type);
```

Funktion: Trägt einen Wert in ein Array-Feld ein.

Parameter:

`a`	⇐	Array, in das der Wert eingetragen werden soll.
`x`	⇐	Index in x-Richtung des Array-Feldes, in das der Wert geschrieben werden soll.
`y`	⇐	Index in y-Richtung des Array-Feldes, in das der Wert geschrieben werden soll.
`data`	⇐	Zeiger auf den Wert, der eingetragen werden soll. Dabei stellt das generische Konzept sicher, daß nur Elemente eines Typs in ein Array gelangen können.


```
PROCEDURE Get(REF a : Array;
                  x,
                  y : CARDINAL):type;
```

Funktion: Liest den Wert eines Array Feldes aus.

Parameter:

`a`	⇐	Array, aus dem gelesen werden soll.
`x`	⇐	Index in x-Richtung des Feldes, das ausgelesen werden soll.
`y`	⇐	Index in y-Richtung des Feldes, das ausgelesen werden soll.
	⇒	Zeiger auf den Wert dieses Feldes. War unter diesem Index noch kein Wert eingetragen, wird `NIL` zurückgegeben.

```
PROCEDURE FirstXatY(REF a : Array;
                        y : CARDINAL):CARDINAL;
```

Funktion: Sucht den ersten x-Index mit einem Eintrag in der Reihe mit Index „y“.

Parameter:

`a`	⇐	Array, in dem nach dem Index gesucht werden soll.
`y`	⇐	Index der Reihe, in der der erste Eintrag gesucht werden soll.
	⇒	Erster x-Index, an dem sich ein valider Eintrag befindet.

```
PROCEDURE LastXatY(REF a : Array;
                       y : CARDINAL):CARDINAL;
```

Funktion: Sucht den letzten x-Index mit einem Eintrag in der Reihe mit Index „y“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
y	⇐	Index der Reihe, in der der letzte Eintrag gesucht werden soll.
	⇒	Letzter x-Index, an dem sich ein valider Eintrag befindet.

```
PROCEDURE FirstYatX(REF a : Array;
                        x : CARDINAL):CARDINAL;
```

Funktion: Sucht den ersten y-Index mit einem Eintrag in der Spalte mit Index „x“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, in der der erste Eintrag gesucht werden soll.
	⇒	Erster y-Index, an dem sich ein valider Eintrag befindet.


```
PROCEDURE LastYatX(REF a : Array;
                       x : CARDINAL):CARDINAL;
```

Funktion: Sucht den letzten y-Index mit einem Eintrag in der Spalte mit Index „x“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, in der der letzte Eintrag gesucht werden soll.
	⇒	Letzter y-Index, an dem sich ein valider Eintrag befindet.

```
PROCEDURE NextX(REF a : Array;
                    x,
                    y : CARDINAL):CARDINAL;
```

Funktion: Sucht den x-Index des nächsten validen Eintrags in der Zeile „y“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, ab der der nächste Eintrag gesucht werden soll.
y	⇐	Index der Zeile, in der gesucht werden soll.
	⇒	Nächster x-Index in dieser Zeile, an dem sich ein valider Eintrag befindet.

```
PROCEDURE NextY(REF a : Array;
                    x,
                    y : CARDINAL):CARDINAL;
```

Funktion: Sucht dem y-Index des nächsten validen Eintrags in Spalte „x“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, in der gesucht werden soll.
y	⇐	Index der Zeile, ab der gesucht werden soll.
	⇒	Nächster y-Index in dieser Spalte, in der sich ein valider Eintrag befindet.

```
PROCEDURE PrevX(REF a : Array;
                    x,
                    y : CARDINAL):CARDINAL;
```

Funktion: Sucht den x-Index des vorherigen validen Eintrags in der Zeile „y“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, ab der der vorherige Eintrag gesucht werden soll.
y	⇐	Index der Zeile, in der gesucht werden soll.
	⇒	Vorheriger x-Index in dieser Zeile, an dem sich ein valider Eintrag befindet.


```
PROCEDURE PrevY(REF a : Array;
                    x,
                    y : CARDINAL):CARDINAL;
```

Funktion: Sucht den y-Index des vorherigen validen Eintrags in Spalte „x“.

Parameter:

a	⇐	Array, in dem nach dem Index gesucht werden soll.
x	⇐	Index der Spalte, in der gesucht werden soll.
y	⇐	Index der Zeile, ab der gesucht werden soll.
	⇒	Vorheriger y-Index in dieser Spalte, in der sich ein valider Eintrag befindet.

7.9 Exception

In diesem Modul finden Sie eine Sammlung von Systemexceptions, die zum Teil bei Laufzeitfehlern aufgerufen werden und zum Teil von unseren Modulen verwendet werden. Sollten Sie einmal eine Exception erhalten, die nicht in dem entsprechenden Modul definiert ist, finden Sie sie mit Sicherheit hier. Außerdem enthält es Prozeduren, mit denen man im Closeteil die letzte Exception abfragen oder auch die Exception ausgeben kann.

```
DEFINITION MODULE Exceptions;

EXCEPTION
  EverythingOk           : 0;
  BusError               : 2;
  AddressError           : 3;
  IllegalOpcode          : 4;
  DivisionByZero         : 5;
  RangeViolation         : 6;
  Overflow               : 7;
  PrivilegeViolation     : 8;
  Trace                  : 9;
  Line_A_Emulator        : 10;
  Line_F_Emulator        : 11;

  Trap_0                 : 32;
  Trap_1                 : 33;
  Trap_2                 : 34;
  Trap_3                 : 35;
  Trap_4                 : 36;
  Trap_5                 : 37;
  Trap_6                 : 38;
  Trap_7                 : 39;
  Trap_8                 : 40;
  Trap_9                 : 41;
  Trap_10                : 42;

  Function_No_Return     : 43;
  Local_Proc_Var         : 44;
  NIL_Dereferenced       : 45;
  Overflow2              : 46;
  RangeViolation2        : 47;

  StackUnderflow         : 99;
  UserBreak              : 100;
```



```
  CouldNotOpenLibrary    : "Could not open library";
  CouldNotOpenResource   : "Could not open resource";
  UnimplementedProcedure : "Unimplemented procedure";
  NilPassed              : "Nil-pointer passed to procedure";

PROCEDURE GetExceptionId():LONGINT;

$$OwnHeap:=TRUE;
PROCEDURE GetExceptionString():STRING;

PROCEDURE WriteException;

PROCEDURE ExceptionToSer;

END Exceptions.
```

EXCEPTION

EverythingOk Es liegt keine Exception vor, dies sollte normalerweiser der Wert sein, den man bei der Exceptionabfrage im Closeteil erhalten sollte. Wenn nicht, ist wohl etwas schief gegangen.

BusError Kann eigentlich nie auftreten, sie ist hier nur der Vollständigkeit halber erwähnt.

AdressError Wird ausgelöst, wenn man versucht, auf einem 68000 an einer ungeraden Adresse ein Wort oder ein Langwort zu lesen.

IllegalOpCode Kann durch nicht initialisierte Prozedur-Variablen, oder durch den Illegal-Befehl in Assembler ausgelöst werden.

DivisionByZero Es wurde versucht durch Null zu teilen.

RangeViolation Verletzung eines Zahlenbereichs. Dies kann ein Unterbereich sein oder ein Array-Index.

Overflow Das Ergebnis einer Rechenoperation lag außerhalb des Wertebereichs der verwendeten Typen.

PrivilegeViolation Kann durch Verwendung von Supervisor-Befehlen in Assembler auftreten oder, falls der Amiga jemals über einen Speicherschutz verfügen sollte, wenn man auf einen fremden Speicherbereich zugreift.

Trace Kann nur entstehen, wenn man mittels Inline oder Assembler das Trace-Flag setzt.

Line_A_Emulator Kann nur durch Assembler entstehen. Sehr unwahrscheinlich.

Line_F_Emulator Es wurde Code ausgeführt, der für eine FPU (68881/2) compiliert wurde, obwohl der Rechner über keine FPU verfügt.

Bei den oben aufgeführten Exceptions handelt es sich um Prozessorexceptions und würden normalerweise zu einem Systemalert führen. Werden sie nicht vom Programm abgefangen, werden sie als Laufzeitfehler gemeldet, da Cluster grundsätzlich einen Handler für Prozessorexceptions installiert.

Die folgenden Exceptions `Trap_0`...`Userbreak` sollte man nicht benutzen oder abfangen, da sie für interne Compilerchecks verwendet werden.

Bei den folgenden Exceptions handelt es sich nicht um Prozessorexceptions.

CouldNotOpenLibrary Wird von Systemmodulen, die eine Library öffnen aufgerufen, falls sich die Libary nicht öffnen ließ.

CouldNotOpenResource Wird von Systemmodulen, die eine Resource öffnen aufgerufen, falls sich die Resource nicht öffnen ließ.

UnimplementedProcedure Diese Exception sollte verwendet werden, falls man eine Prozedur definiert, aber noch nicht implementiert hat. Dies ist besser, als nur eine leere Prozedur zu verwenden, da man so nicht so leicht vergißt, sie zu implementieren.

NilPassed Sollte man in eigenen Prozeduren immer verwenden, wenn innerhalb ein Check läuft, ob `NIL` übergeben wurde. Auch eine Anzahl von Prozeduren der Standardmodule verwenden diese Exception.

Die folgenden Prozeduren sind nur zum Aufruf innerhalb eines Closeteil sinnvoll, da an anderen Stellen immer nur „`EverythingOk`" zurückgeliefert würde.

```
PROCEDURE GetExceptionId():LONGINT;
```

Funktion: Hiermit läßt sich der Wert der aktuellen Exceptionnummer abfragen. Der Wert ist jedoch nur dann als Nummer zu verstehen, wenn er im Bereich zwischen 0 und 4000 liegt. Sonst handelt es sich um einen Zeiger auf einen Cluster-String.

Parameter:

⇒ Wert der letzten Exception.

```
$$OwnHeap:=TRUE;
PROCEDURE GetExceptionString():STRING;
```

Funktion: Liefert einen String mit der letzten Exception zurück. War die Exception eine Zahl, wird diese automatisch in einen String gewandelt.

Parameter:

⇒ String mit der letzten Exception.

```
PROCEDURE WriteException;
```

Funktion: Gibt die letzte Exception aus. Wenn das Programm von der Shell gestartet wurde, wird die Meldung auf der Shell ausgegeben. Wurde es von der Workbench gestartet, erscheint ein Systemrequester. Diese Funktion kann auch in Libraries verwendet werden.

```
PROCEDURE ExceptionToSer;
```

Funktion: Im Falle, daß das Programm im CloseTeil selbst abstürzt, kann man diese Routine verwenden, die die zuletzt aufgetretene Exception auf die serielle Schnittstelle ausgibt. Wir verwenden dafür systemkonform die Routine `RawPutChar` aus Exec. Man kann also eines jener Tools verwenden, die diese Ausgabe auf die parallele Schnittstelle umlenkt.

7.10 FileSystem

Das Modul FileSystem stellt Prozeduren und Funktionen zur Dateiverwaltung zur Verfügung. Eine Datei ist normalerweise einfach eine Byte- oder Zeichenfolge auf einer Diskette/Festplatte. Sie können selbst eine solche Datei erzeugen und darin den Inhalt von Variablen oder allozierten Speicherstücken abspeichern und jederzeit wieder einlesen. Auf diese Weise bleibt die Information auch nach dem Verlassen des Programms oder nach dem Ausschalten des Computers erhalten.

Im Gegensatz zu Dos nimmt ihnen dieses Modul jedoch eine Menge Arbeit ab, außerdem schließt es automatisch alle geöffneten Files am Programmende wieder.

Man kann aber auch an eine bestehende Datei (im Folgenden auch File genannt) Daten anhängen, diese verändern, oder auch einfach nur Daten aus dieser einlesen und verarbeiten.

```
DEFINITION MODULE FileSystem;

FROM System     IMPORT Regs;
FROM Resources  IMPORT NotEnoughMemory,ContextPtr;
FROM T_Dos      IMPORT DosExceptionGrp;

EXCEPTION
  BadFile              : "Bad filehandle";
  NoInFile             : "InFile not opened";
  NoOutFile            : "OutFile not opened";
  OutFileOpenedTwice   : "OutFile allready open";
  InFileOpenedTwice    : "InFile allready open";

GROUP
  FsExceptionGrp  = NoInFile,NoOutFile,NotEnoughMemory,
                    T_Dos.DosExceptionGrp;

TYPE
  File          = HIDDEN;

|-------- Global File -------

PROCEDURE Open(VAR f        : File;
               REF name     : STRING;
```



```
                   new      : BOOLEAN := FALSE;
                   size     : LONGINT := $1000;
                   context  : ContextPtr := NIL);

PROCEDURE Close(VAR f : File);

PROCEDURE Length(f : File):LONGINT;

|------- Input --------------

PROCEDURE Read(f : File;VAR c : CHAR);

PROCEDURE ReadBlock(f : File;VAR block : ANYTYPE);

PROCEDURE ReadBytes(f : File;pos : ANYPTR;len : LONGINT);

|------- Output -------------

PROCEDURE Write(f : File;c : CHAR);

PROCEDURE WriteBlock(f : File;REF block : ANYTYPE);

PROCEDURE WriteBytes(f : File;pos : ANYPTR;len : LONGINT);

PROCEDURE Pos(f : File):LONGINT;

PROCEDURE SetPos(f : File;pos : LONGINT);

PROCEDURE Eof(f : File):BOOLEAN;

GROUP
  FileIOGrp  = Open,File,Close,Length,Read,ReadBlock,
               ReadBytes,Write,WriteBlock,WriteBytes,
               Pos,SetPos,Eof;

|------- Quickfiles ---------

PROCEDURE Q_OpenInFile(REF name     : STRING;
                           buffSize : LONGINT;
```

```
                          context  : ContextPtr:=NIL);

PROCEDURE Q_OpenOutFile(REF name     : STRING;
                            buffSize : LONGINT;
                            context  : ContextPtr:=NIL);

PROCEDURE Q_CloseInFile;

PROCEDURE Q_CloseOutFile;


PROCEDURE Q_Test(c : CHAR):BOOLEAN;

PROCEDURE Q_Read(VAR c IN A0 : CHAR);

PROCEDURE Q_ReadBlock(VAR block : ANYTYPE);

PROCEDURE Q_ReadBytes(pos : ANYPTR;len : LONGINT);

PROCEDURE Q_Write(c IN D4 : CHAR);

PROCEDURE Q_WriteBlock(REF block : ANYTYPE);

PROCEDURE Q_WriteBytes(pos : ANYPTR;len : LONGINT);


PROCEDURE Q_GetInPos():LONGINT;

PROCEDURE Q_SetInPos(pos : LONGINT);

PROCEDURE Q_GetOutPos():LONGINT;

PROCEDURE Q_SetOutPos(pos : LONGINT);


GROUP
  QuickIOGrp  = Q_OpenInFile,Q_OpenOutFile,Q_CloseInFile,
                Q_CloseOutFile,Q_Test,Q_Read,Q_ReadBlock,
                Q_ReadBytes,Q_Write,Q_WriteBlock,Q_WriteBytes,
                Q_GetInPos,Q_SetInPos,Q_GetOutPos,Q_SetOutPos;
  All         = FileIOGrp,QuickIOGrp;

END FileSystem.
```

Exception:

BadFile Ein übergebener Filehandle ist nicht gültig.

NoInFile Bei einer Quickfilefunktion wurde auf das Eingabefile zugegriffen, ohne daß eines geöffnet wurde.

NoOutFile Bei einer Quickfilefunktion wurde auf das Ausgabefile zugegriffen, ohne daß eines geöffnet wurde.

OutFileOpenedTwice Da sich immer nur ein Quick-Ausgabefile öffnen läßt, wird diese Exception aufgerufen, wenn versucht wird, ein zweites zu öffnen.

InFileOpenedTwice Da sich immer nur ein Quick-Eingabefile öffnen läßt, wird diese Exception aufgerufen, wenn versucht wird, ein zweites zu öffnen.

Im übrigen können Exceptions aus dem Modul T_Dos auftreten.

7.10.1 Global File

```
PROCEDURE Open(VAR f       : File;
               REF name    : STRING;
                   new     : BOOLEAN := FALSE;
                   size    : LONGINT := $1000;
                   context : ContextPtr := NIL);
```

Funktion: Bevor man in ein File schreiben oder davon lesen kann, muß man es erst öffnen. Dabei können alle Exceptions der Exceptiongruppe „`OpenErr`“ aus dem Modul `T_Dos` auftreten. Nach dem Öffnen steht der interne Positionszeiger des Files am Anfang des Files.

Parameter:

`f` ⇒ Zugriff auf das File. Da das FileSystem beliebig viele Dateien öffnen kann, benötigt man einen Zugriff auf das File, der bei allen weiteren Operationen übergeben wird, damit das Modul weiß, welches File es zu bearbeiten hat.

`name` ⇐ Pfad der Datei, die geöffnet werden soll.

`new` ⇐ Muß `TRUE` sein, wenn eine neue Datei erzeugt und eine bestehende mit gleichem Namen dabei gelöscht werden soll. Ist `new FALSE`, dann wird eine bestehende Datei geöffnet.

`size` ⇐ Gibt an, wie groß der Puffer sein soll, der für das File eingerichtet werden soll. Da ein Diskettenlaufwerk nicht unbedingt zu den schnellsten Datenträgern zählt, sollte man nur darauf zugreifen, wenn eine größere Datenmenge darauf geschrieben oder davon gelesen werden soll. Aus diesem Grund hält man sich immer einen Teil der Datei in einem Puffer im Speicher und arbeitet in diesem. Nur wenn man sehr große Datenmengen auf einmal schreiben oder lesen will, sollte man auf einen Puffer verzichten (`size=0`, da sonst die Daten unnötigerweise erst in den Puffer und dann an ihr Ziel kopiert werden, anstatt direkt dorthin. Als Faustregel sollte man den Puffer am besten zwischen einem und zehn KBytes wählen.

`context` ⇐ Context, zu dem das File geöffnet werden soll. Ist dieser NIL, wird der ActContext verwendet (siehe `Resources`).

```
PROCEDURE Close(VAR f : File);
```

Funktion: Schließt ein File. Alle von ihnen geöffneten Files müssen am Ende wieder geschlossen werden. Zwar schließt FileSystem die Files auch selbständig, jedoch zeugt es von schlechtem Stil, sich darauf zu verlassen, da man damit wahrscheinlich Files länger geöffnet hält als nötig. Außerdem wird anderen Programmen der Zugriff auf dieses File unnötigerweise eingeschränkt, was man in einem Multitaskingsystem stets vermeiden sollte.

Parameter:

`f` ⇐ Zugriff auf das File, das geschlossen werden soll.

PROCEDURE `Length(f : File):LONGINT;`

Funktion: Liefert die aktuelle Länge eines Files.

Parameter:

`f`	⇐	Zugriff auf das File, dessen Länge ermittelt werden soll.
	⇒	Länge des Files in Bytes.

7.10.2 Input

Hier finden sich alle Funktionen, um von einem File Daten zu lesen.

PROCEDURE `Read(f : File;`**VAR** `c : CHAR);`

Funktion: Liest ein Zeichen aus einem File. Der interne Positionzeiger wandert dabei eine Stelle weiter. War man schon am Ende eines Files, wird die Exception „`EOF`“ ausgelöst.

Parameter:

`f`	⇐	Zugriff auf das File, von dem gelesen werden soll.
`c`	⇒	Zeichen, das gelesen wurde.

PROCEDURE `ReadBlock(f : File;`**VAR** `block : ANYTYPE);`

Funktion: Dient dazu, die selbe Anzahl von Bytes zu lesen, die der Länge der übergebenen Variablen entspricht. Dieser Prozedur können Sie also im Grunde Variablen aller Typen übergeben (besonders für Records geeignet). Versuchen Sie jedoch nicht ein ASCII-File in einen String einzulesen, da dabei auch das Längenfeld des Strings mit Zeichen gefüllt würde.

Parameter:

`f`	⇐	Zugriff auf das File, von dem gelesen werden soll.
`block`	⇒	Gelesene Variable.

PROCEDURE `ReadBytes(f : File;pos : ANYPTR;len : LONGINT);`

Funktion: Liest eine beliebige Anzahl Bytes und schreibt sie an eine bestimmte Position in den Speicher. Dabei verschiebt sich die Leseposition um die Anzahl der gelesenen Bytes.

Parameter:

`f`	⇐	Zugriff auf das File, von dem gelesen werden soll.
`pos`	⇐	Zeiger auf die Position im Speicher, an die die Daten geschrieben werden sollen.
`len`	⇐	Anzahl Bytes, die gelesen werden sollen.

7.10.3 Output

Hier finden Sie alle Prozeduren, um Daten in ein File zu schreiben.

PROCEDURE `Write(f : File;c : CHAR);`

Funktion: Schreibt ein Zeichen in ein File. Der interne Positionszeiger wandert dabei eine Stelle weiter. Es wird dabei immer an der aktuellen Position weitergeschrieben.

Parameter:

`f`	⇐	Zugriff auf das File, in das geschrieben werden soll.
`c`	⇒	Zeichen, das geschrieben werden soll.

PROCEDURE `WriteBlock(f : File;`**REF** `block : ANYTYPE);`

Funktion: Eignet sich vor allem, um z. B. ganze Records oder Arrays auf einmal zu schreiben, die man später wieder mit ReadBlock einlesen kann. Achtung, wollen Sie nur den wirklich gefüllten Teil eines Strings als ASCII-Text abspeichern, so sollten Sie dazu nicht WriteBlock benutzen, da diese Routine immer den gesamten String, also mit Länge und nicht benutzten Feldern abspeichert.

Parameter:

`f`	⇐	Zugriff auf das File, in das geschrieben werden soll.
`block`	⇒	Variable, deren Inhalt geschrieben werden soll.

PROCEDURE `WriteBytes(f : File;pos : ANYPTR;len : LONGINT);`

Funktion: Schreibt von einer bestimmten Position im Speicher ab eine Anzahl Bytes in ein File. Eignet sich sehr gut, eine bekannte Anzahl Bytes abzuspeichern, so z. B. um dynamische Strukturen abzuspeichern, oder die Zeichen eines Strings.

```
WriteBytes(MyFile,Str.data'PTR,Str.len)
```

Parameter:

`f`	⇐	Zugriff auf das File, in das geschrieben werden soll.
`pos`	⇐	Zeiger auf die Position im Speicher, von der ab die Daten gelesen werden sollen.
`len`	⇐	Anzahl Bytes, die in das File geschrieben werden sollen.

7.10.4 Position

Hier finden Sie die Funktionen, um die aktuelle Position im File abzufragen oder neu zu setzen.

PROCEDURE `Pos(f : File):LONGINT;`

Funktion: Gibt die aktuelle Position des internen Schreib/Lesezeiger relativ zum Dateianfang zurück.

Parameter:

`f`	⇐	File, von dem man die Information erhalten möchte.
	⇒	Aktuelle Position.

PROCEDURE `SetPos(f : File;pos : LONGINT);`

Funktion: Daten am Stück zu lesen oder zu schreiben wäre nichts besonderes, denn das könnte man auch mittels InOut. Interessant wird es bei Dateien mit immer gleichgroßen Datenfeldern, z. B. eine Adreßdatei. Hier möchte man gerne direkt auf das gewünschte Datenfeld zugreifen, ohne die ganze Datei einzuladen. Daher hat man in Dateien eine Schreib/Lesemarke geschaffen, ähnlich einem Cursor im Editor. Um diese Marke zu versetzen, benutzt man SetPos. Um z. B. auf das 12. Datenfeld des Typs AdrRecord zuzugreifen, müßte man nur folgendes machen: `SetPos(MyFile,12*AdrRecord'SIZE)`.

Parameter:

`f`	⇐	File, dessen Marke versetzt werden soll.
`pos`	⇐	Neue Position in Bytes, vom Fileanfang aus gerechnet.

PROCEDURE `Eof(f : File):BOOLEAN;`

Funktion: Hiermit kann überprüft werden, ob man schon das Ende des Files erreicht hat. Diese Funktion hat jedoch viel von ihrer Bedeutung verloren, da nun alle Lesefunktionen die Exception „`EOF`“ auslösen, sobald man über das Fileende hinausliest.

Parameter:

`f` ⇐ File, das überprüft werden soll.

⇒ `TRUE`, wenn mit der letzten Operation das Fileende ereicht wurde. Versucht man dennoch weiterzulesen, wird eine Exception ausgelöst.

7.10.5 Quickfiles

Durch die Möglichkeit mit den bisherigen Prozeduren gleichzeitig lesend und schreibend auf eine Datei zuzugreifen, fällt ein enormer Verwaltungsaufwand an, der auch entsprechend Zeit benötigt. Bei vielen Anwendungen ist es jedoch nicht nötig, gleichzeitig auf ein File lesend und gleichzeitig schreibend zuzugreifen. Ein gutes Beispiel ist der Compiler; er muß beim Übersetzen auf Symbolmodule nur lesend zugreifen und schreibt auch nur in eine neu erzeugte Datei, ohne von dieser lesen zu müssen.

In solchen Fällen sollte man die Quickfileroutinen verwenden. Mit diesen läßt sich zwar nur auf jeweils ein File entweder lesend oder schreibend zugreifen, dafür sind sie einiges schneller als die normalen Routinen. Außerdem muß man nicht bei jedem Funktionsaufruf einen Filepointer übergeben, da immer klar ist, auf welches der beiden Files der Zugriff geschehen soll.

Q_OpenInFile Öffnet ein File, um davon zu lesen. Danach ist es nicht möglich, auch auf das File zu schreiben. Versucht man zweimal ein File auf diese Weise zu öffnen, ohne das letzte „Infile" geschlossen zu haben, wird eine Exception ausgelöst.

Q_OpenOutFile Öffnet ein File, um darauf zu schreiben. Davon kann dann nicht gelesen werden. Versucht man zweimal ein File auf diese Weise zu öffnen, ohne das letzte „Outfile" geschlossen zu haben, wird eine Exception ausgelöst.

Q_CloseInFile Schließt das LeseFile.

Q_CloseOutFile Schließt das Ausgabefile.

Die weiteren Routinen entsprechen in ihrer Funktion den oben erklärten, sie sind lediglich durch ein vorangesteltes „`Q_`" gekennzeichnet. Für SetPos und Pos existieren jeweils für In- und OutFile zwei Prozeduren `Q_SetInPos`, `Q_SetOutPos` und `Q_GetInPos`, `Q_GetOutPos`.

```
PROCEDURE Q_Test(c : CHAR):BOOLEAN;
```

Funktion: Oft möchte man wissen, welches Zeichen als nächstes gelesen würde. Hierfür dient `Q_Test`. Mit dieser Funktion können Sie testen, ob das nächste Zeichen gleich dem übergebenen ist. Diese Funktion ist viel schneller, als erst ein Zeichen zu lesen, und eventuell wieder die Marke zurückzusetzen. **Parameter:**

`c` ⇐ Zeichen, das getestet werden soll.

⇒ `TRUE`, wenn das nächste Zeichen „`c`" entspricht.

7.11 GfxDraw

Bevor Sie hier weiterlesen sollten Sie erst mal einen Blick in GfxScreen werfen, da Sie einige Dinge von dort benötigen. GfxDraw enthält Prozeduren, um einfach die graphischen Fäkigkeiten des Amigas nutzen zu können. Die GfxModule nehmen einem gegenüber dem Betriebssystem eine Menge Arbeit ab. GfxDraw enthält zum großen Teil alle Funktionen, die auch Graphics bietet, doch dürfen Sie nie Funktionen aus Graphics mit denen aus GfxDraw vermischen. Sollten Sie es doch tun, können wir keine Verantwortung für die Folgen übernehmen.

```
DEFINITION MODULE GfxDraw;

FROM System    IMPORT BITSET,SHORTSET;
FROM GfxScreen IMPORT Screen;

EXCEPTION
  AreaDrawFail        : "AreaDraw failed";
  AreaEllipseFail     : "AreaEllipse failed";
  AreaEndFail         : "AreaEnd failed";
  AreaMoveFail        : "AreaMove failed";
  FloodFail           : "Flood failed";
  WritePixelFail      : "WritePixel failed";

TYPE
  Point     = ARRAY [0..1] OF INTEGER;
  Polygon   = ARRAY OF Point;

  DrawModes = (drawAPen,drawABPen,drawInvers);

  Pattern   = ARRAY OF CARDINAL;


PROCEDURE SetAPen(s : Screen;color : INTEGER);

PROCEDURE SetBPen(s : Screen;color : INTEGER);

PROCEDURE SetDrawMode(s : Screen;mode : DrawModes);

PROCEDURE SetPlanes(s : Screen;plane : SHORTSET);
```



```
PROCEDURE SetDrawPt(s : Screen;pat : CARDINAL);

PROCEDURE SetAreaPt(s : Screen;VAR pat : Pattern);

PROCEDURE ClearScreen(s : Screen);

PROCEDURE WritePixel( s : Screen; x, y : INTEGER );

PROCEDURE ReadPixel( s : Screen;x,y : INTEGER ):INTEGER;


PROCEDURE CursorX( s : Screen ):INTEGER;

PROCEDURE CursorY( s : Screen ):INTEGER;

PROCEDURE Move(s : Screen;x,y : INTEGER);

PROCEDURE MoveTo(s : Screen;dx,dy : INTEGER);

PROCEDURE Draw(s : Screen;x,y : INTEGER);

PROCEDURE DrawTo(s : Screen;dx,dy : INTEGER);

PROCEDURE Line(s : Screen;x1,y1,x2,y2 : INTEGER);

PROCEDURE DrawRectangle(s : Screen;x1,y1,x2,y2 : INTEGER);

PROCEDURE DrawPolygon(s : Screen;VAR poly : Polygon);

PROCEDURE DrawCircle(s : Screen;x,y,r : INTEGER);

PROCEDURE DrawEllipse(s : Screen;x,y,a,b : INTEGER);

GROUP
  DrawGrp       =
            (*T*) DrawModes,Pattern,Point,Polygon,
                  Screen,SHORTSET,
            (*E*) WritePixelFail,
            (*P*) ClearScreen,CursorX,CursorY,Draw,
                  DrawCircle,DrawEllipse,DrawPolygon,
                  DrawRectangle,DrawTo,Line,Move,MoveTo,
```

```
                  ReadPixel,SetAPen,SetAreaPt,SetBPen,
                  SetDrawMode,SetDrawPt,SetPlanes,WritePixel;

PROCEDURE AreaMove( s : Screen;x,y : INTEGER );

PROCEDURE AreaDraw(s : Screen;x,y : INTEGER);

PROCEDURE AreaEnd( s : Screen );

PROCEDURE AreaRectangle(s : Screen;x1,y1,x2,y2 : INTEGER);

PROCEDURE AreaPolygon(s : Screen;VAR poly : Polygon);

PROCEDURE AreaCircle(s : Screen;x,y,r : INTEGER);

PROCEDURE AreaEllipse(s : Screen;x,y,a,b : INTEGER);

PROCEDURE Flood(s : Screen;x,y : INTEGER);

GROUP
  ExceptionGrp  = AreaDrawFail,AreaEllipseFail,
                  AreaEndFail,AreaMoveFail,FloodFail,
                  WritePixelFail;
  AreaGrp       =
            (*E*) AreaDrawFail,AreaEllipseFail,AreaEndFail,
                  AreaMoveFail, FloodFail,
            (*I*) Screen,
            (*P*) AreaCircle,AreaDraw,AreaEllipse,AreaEnd,
                  AreaMove,AreaPolygon,AreaRectangle,Flood,
                  SetAreaPt;
  All           = DrawGrp,AreaGrp;

END GfxDraw.
```

Exceptions:

AreaDrawFail Funktion `AreaDraw` konnte nicht ordnungsgemäß ausgeführt werden.

AreaEllipseFail Funktion `AreaEllipse` konnte nicht ordnungsgemäß ausgeführt werden.

AreaEndFail Funktion `AreaEnd` konnte nicht ordnungsgemäß ausgeführt werden.

AreaMoveFail Funktion `AreaMove` konnte nicht ordnungsgemäß ausgeführt werden.

FloodFail Funktion `Flood` konnte nicht ordnungsgemäß ausgeführt werden.

WritePixel Es wurden Punkkoordinaten übergeben, die außerhalb des Bildschirms liegen.

Typen:

Point Beschreibung eines Bildpunktes, dabei gibt der erste Eintrag[7] des Arrays die x, der zweite die y-Koordinate an.

Polygon Offenes Array von Punkten. Dient zur Definition von Polygonen für die Funktion `AreaPolygon`.

DrawModes Legt die Art fest, wie die Zeichenoperationen wirken:

- **drawAPen** Es wird mit dem Vordergrundstift, dem `APen` gezeichnet, bei einer Schrift oder einem Muster freie Stellen lassen den Hintergrund durchscheinen.
- **drawABPen** Es wird mit dem Vordergrundstift, dem `APen`, gezeichnet, bei einer Schrift oder einem Muster freie Stellen werden mit dem Hintergrundstift, dem `BPen`s, gefüllt.
- **drawInvers** Beim Zeichnen werden die betroffenen Stellen des Hintergrunds invertiert. Durch nochmaliges darüberzeichnen mit diesem Modus stellt man den ursprünglichen Hintergrund wieder her.

Pattern Muster für Flächen, dieses ist 16 Punkte (Bits) breit und eine zweierPotenz hoch, also 1,2,4,8,16,32 Punkte hoch. Jede Cardinalzahl in diesem Feld entspricht daher einer Zeile des Muster und jedes gesetzte Bit in einer Cardinalzahl einem gesetzen Punkt. Am besten legt man Pattern als konstantes Array an und

[7] Denken Sie daran, daß in einem Array mit Bereich `[0..x]` das erste Felde immer den Index „0“ hat.

gibt die einzelnen Felder direkt binär an : `%1100110011001100`. Bei `DrawMode=drawAPen` werden die gesetzen Punkte in der Farbe des APens gezeichnet und die nicht gesetzten Punkte lassen den Hintergrund durchscheinen; für `DrawMode=drawABPen` werden die nicht gesetzten Punkte in der Farbe des BPens gezeichnet. Wichtig ist, daß dieses Feld im ChipMem liegen muß. Daher derartige Konstanten in ein einzelnes Modul auslagern und dort am Anfang den Schalter `$$ChipMem:=TRUE` setzen.

PROCEDURE `SetAPen(s : Screen;color : INTEGER);`

Funktion: Setzt die Farbe des Vordergrundstiftes.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`color`	⇐	Farbregisternummer, die dem Vordergrundstift zugeordnet werden soll.

PROCEDURE `SetBPen(s : Screen;color : INTEGER);`

Funktion: Setzt die Farbe des Hintergrundstiftes.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`color`	⇐	Farbregisternummer, die dem Hintergrundstift zugeordnet werden soll.

PROCEDURE `SetDrawMode(s : Screen;mode : DrawModes);`

Funktion: Setzt den Zeichenmodus für einen Screen

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Zeichenmodus gesetzt werden soll.
`mode`	⇐	Neuer Zeichenmodus.

PROCEDURE `SetPlanes(s : Screen;plane : SHORTSET);`

Funktion: Mit dieser Prozedur kann man festlegen, auf welche Bitplanes eines Screens gezeichnet werden kann.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`plane`	⇐	Bitplanes, in die gezeichnet werden darf. Die Bitplanes entsprechen dabei direkt den Nummern der Bits (Denken Sie daran, daß Informatiker immer bei 0 mit dem Zählen anfangen) im `SHORTSET`. Um z. B. nur auf BitPlane 1,3,5 zuzugreifen müßten Sie für `plane` `SHORTSET:{1,3,5}` wählen. Standardmäßig ist `plane` bei vier Bitplanes `SHORTSET:{0,1,2,3}` gesetzt.

PROCEDURE `SetDrawPt(s : Screen;pat : CARDINAL);`

Funktion: Setzt das Linienmuster, daß bei Linienroutinen verwendet werden soll.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`pat`	⇐	Ist eine Cardinalzahl, in der wie bei Pattern jedes gesetzte Bit für einen gesetzten Punkt steht. Um z. B. eine gepunktete Linie zu erhalten müßten Sie `pat = %1100110011001100` setzen.

PROCEDURE `SetAreaPt(s : Screen;`**VAR**` pat : Pattern);`

Funktion: Setzt das Flächenmuster, das von nun an für alle Flächen- und Füllfunktionen verwendet werden soll.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`pat`	⇐	Muster, Aufbau siehe Erklärung der Typen dieses Moduls oben.

```
PROCEDURE ClearScreen(s : Screen);
```

Funktion: Füllt den Screen mit der Hintergrundfarbe.

Parameter:

s	⇐	Zugriff auf den Screen, der gelöscht werden soll.

```
PROCEDURE WritePixel( s : Screen; x, y : INTEGER );
```

Funktion: Setzt einen Punkt in der Vordergrundfarbe. Falls der Pixel außerhalb des Screens lag, wird die Exception `WritePixelFail` ausgelöst.

Parameter:

s	⇐	Zugriff auf den Screen, auf dem der Punkt gesetzt werden soll.
x	⇐	x-Position des Punktes.
y	⇐	y-Position des Punktes.

```
PROCEDURE ReadPixel( s : Screen;x,y : INTEGER ):INTEGER;
```

Funktion: Liest die Farbe eines Punktes ein.

Parameter:

s	⇐	Zugriff auf den Screen.
x	⇐	x-Position des Punktes.
y	⇐	y-Position des Punktes.
	⇒	Nummer des Stiftes, mit dem dieser Punkt gezeichnet wurde.


```
PROCEDURE CursorX( s : Screen ):INTEGER;
```

Funktion: Ermittelt die x-Koordinate des Graphikcursors[8].

Parameter:

s	⇐	Zugriff auf den Screen.
	⇒	x-Koordinate des Cursors.

```
PROCEDURE CursorY( s : Screen ):INTEGER;
```

Funktion: Ermittelt die y-Koordinate des Graphikcursors.

Parameter:

s	⇐	Zugriff auf den Screen.
	⇒	y-Koordinate des Cursors.

[8] Bei einigen Zeichenoperationen, wie dem Zeichnen von Linien oder Textausgabe, übergibt man nicht die Position direkt, sondern die Ausgabe wird an der Position eines unsichtbaren Graphikcursors gemacht. Nach der Operation befindet sich der Cursor am Ende der Linie / des Texts, so daß man direkt weiterzeichnen kann.

PROCEDURE `Move(s : Screen;x,y : INTEGER);`

Funktion: Setzt den Graphikcursor an die angegeben Position.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`x`	⇐	Neue x-Position des Cursors.
`y`	⇐	Neue y-Position des Cursors.

PROCEDURE `MoveTo(s : Screen;dx,dy : INTEGER);`

Funktion: Verschiebt den Graphikcursor relativ zur aktuellen Position.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`dx`	⇐	Betrag, um den der Cursor relativ in x-Richtung verschoben werden soll. Kann auch negativ sein.
`dy`	⇐	Betrag, um den der Cursor relativ in y-Richtung verschoben werden soll.

PROCEDURE `Draw(s : Screen;x,y : INTEGER);`

Funktion: Zeichnet vom Graphikcursor aus zur angegebenen Position. Der Graphik cursor steht danach am Ende der Linie.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`x`	⇐	x-Position des Zielpunktes der Linie.
`y`	⇐	y-Position des Zielpunktes der Linie

PROCEDURE `DrawTo(s : Screen;dx,dy : INTEGER);`

Funktion: Im Gegensatz zu `Draw` sind die Werte `dx`, `dy` relativ zur aktuellen Position des Graphikcursors.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`dx`	⇐	Relative x-Position des Zielpunktes der Linie.
`dy`	⇐	Relative y-Position des Zielpunktes der Linie

PROCEDURE `Line(s : Screen;x1,y1,x2,y2 : INTEGER);`

Funktion: Zieht eine Linien zwischen zwei Punkten, ohne daß `Move` verwendet werden muss.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`x1`	⇐	x-Position des Startpunktes der Linie.
`y1`	⇐	y-Position des Startpunktes der Linie
`x2`	⇐	x-Position des Zielpunktes der Linie.
`y2`	⇐	y-Position des Zielpunktes der Linie

PROCEDURE `DrawRectangle(s : Screen;x1,y1,x2,y2 : INTEGER);`

Funktion: Zeichnet die Umrißlinien eines Rechtecks.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`x1`	⇐	x-Position der linken oberen Ecke.
`y1`	⇐	y-Position der linken oberen Ecke.
`x2`	⇐	x-Position der rechten unteren Ecke.
`y2`	⇐	y-Position der rechten unteren Ecke.

PROCEDURE `DrawPolygon(s : Screen;`**VAR** `poly : Polygon);`

Funktion: Zeichnet ein Polygon.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`poly`	⇐	Feld mit den Koordinaten der Polygonecken.

PROCEDURE `DrawCircle(s : Screen;x,y,r : INTEGER);`

Funktion: Zeichnet einen Kreis. Der Kreis wird bei ungleicher horizontaler und vertikaler Auflösung entsprechend gestreckt.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Mittelpunktes.
y	⇐ y-Position des MittelPunktes.
r	⇐ Radius des Kreises.

PROCEDURE `DrawEllipse(s : Screen;x,y,a,b : INTEGER);`

Funktion: Zeichnet eine Ellipse.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Mittelpunktes.
y	⇐ y-Position des MittelPunktes.
a	⇐ Radius der Ellipse in x-Richtung.
b	⇐ Radius der Ellipse in y-Richtung.

PROCEDURE `AreaMove( s : Screen;x,y : INTEGER );`

Funktion: Um ein Ausgefülltes Polygon zu zeichnen, können Sie die Areabefehle verwenden. AreaMove setzt dazu den ersten Punkt, und schließt gleichzeitig noch nicht vollendete Polygone ab.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Anfangpunktes.
y	⇐ y-Position des Anfangpunktes.

PROCEDURE `AreaDraw(s : Screen;x,y : INTEGER);`

Funktion: Setzt eine weitere Ecke eines ausgefüllten Polygons. Vorher muß schon `AreaMove` ausgeführt worden sein, um das Vieleck zu initialisieren.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des neuen Eckpunktes.
y	⇐ y-Position des neuen Eckpunktes.

PROCEDURE `AreaEnd( s : Screen );`

Funktion: Hat man mit AreaDraw alle Polygonpunkte gesetzt, muß man das Polygon mit AreaEnd schließen, dabei wird automatisch der letzte Punkt mit dem ersten verbunden, und das Polygon mit dem aktuellen Füllmuster in der Farbe des APens gefüllt. Ist kein Muster gesetzt, so wird die gesamte Fläche gefüllt.

PROCEDURE `AreaRectangle(s : Screen;x1,y1,x2,y2 : INTEGER);`

Funktion: Zeichnet ein ausgefülltes Rechtecks.

Parameter:

s	⇐ Zugriff auf den Screen.
x1	⇐ x-Position der linken oberen Ecke.
y1	⇐ y-Position der linken oberen Ecke.
x2	⇐ x-Position der rechten unteren Ecke.
y2	⇐ y-Position der rechten unteren Ecke.

PROCEDURE `AreaPolygon(s : Screen;`**VAR** `poly : Polygon);`

Funktion: Zeichnet ein ausgefülltes Polygon.

Parameter:

s	⇐ Zugriff auf den Screen.
poly	⇐ Feld mit den Koordinaten der Polygonecken.

PROCEDURE AreaCircle(s : Screen;x,y,r : INTEGER);

Funktion: Zeichnet einen ausgefüllten Kreis. Der Kreis wird bei ungleicher horizontaler und vertikaler Auflösung entsprechend gestreckt.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Mittelpunktes.
y	⇐ y-Position des MittelPunktes.
r	⇐ Radius des Kreises.

PROCEDURE AreaEllipse(s : Screen;x,y,a,b : INTEGER);

Funktion: Zeichnet eine ausgefüllte Ellipse.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Mittelpunktes.
y	⇐ y-Position des MittelPunktes.
a	⇐ Radius der Ellipse in x-Richtung.
b	⇐ Radius der Ellipse in y-Richtung.

PROCEDURE Flood(s : Screen;x,y : INTEGER);

Funktion: Füllt eine Fläche in der Farbe des APens. `Flood` füllt solange, bis es auf eine andere Farbe als der des Startpunktes trifft.

Parameter:

s	⇐ Zugriff auf den Screen.
x	⇐ x-Position des Startpunktes.
y	⇐ y-Position des Startpunktes.

7.12 GfxInput

Bevor Sie weiterlesen, sollten Sie erst die Beschreibung von GfxScreen lesen. GfxInput ermöglicht direkt die Tastatur und die Maus abzufragen. Die Abfrage bezieht sich immer auf einen Screen. Der Screen, der angeklickt wurde ist aktiv, d. h. das Programm, das diesen Screen bei der Abfrage angibt, erhält die Nachrichten. Genauso, wie wenn Sie mehrere Fenster geöffnet haben, und die Tastendrücke nur in dem aktiven Fenster Wirkung zeigen. In Wirklichkeit liegt nämlich vor jedem Screen, der mit GfxScreen geöffnet wurde, ein unsichtbares Fenster.

```
DEFINITION MODULE GfxInput;

FROM GfxScreen IMPORT Screen;

TYPE
  UserActions  = (userKey,userMouse);
  UserAct      = SET OF UserActions;
  KeyTypes     = (KeyReturn,KeyBackspace,KeyDelete,
                   KeyTab,KeyHelp,KeyEsc,
                   KeyCsrUp,KeyCsrDown,KeyCsrLeft,
                   KeyCsrRight,
                   KeyF1,KeyF2,KeyF3,KeyF4,KeyF5,
                   KeyF6,KeyF7,KeyF8,KeyF9,KeyF10,
                   SKeyReturn,SKeyBackspace,SKeyDelete,
                   SKeyTab,SKeyHelp,SKeyEsc,
                   SKeyCsrUp,SKeyCsrDown,SKeyCsrLeft,
                   SKeyCsrRight,
                   SKeyF1,SKeyF2,SKeyF3,SKeyF4,SKeyF5,
                   SKeyF6,SKeyF7,SKeyF8,SKeyF9,SKeyF10,
                   AKeyReturn,AKeyBackspace,AKeyDelete,
                   AKeyTab,AKeyHelp,AKeyEsc,
                   AKeyCsrUp,AKeyCsrDown,AKeyCsrLeft,
                   AKeyCsrRight,
                   AKeyF1,AKeyF2,AKeyF3,AKeyF4,AKeyF5,
                   AKeyF6,AKeyF7,AKeyF8,AKeyF9,AKeyF10,
                   CKeyReturn,CKeyBackspace,CKeyDelete,
                   CKeyTab,CKeyHelp,CKeyEsc,
                   CKeyCsrUp,CKeyCsrDown,CKeyCsrLeft,
                   CKeyCsrRight,
                   CKeyF1,CKeyF2,CKeyF3,CKeyF4,CKeyF5,
                   CKeyF6,CKeyF7,CKeyF8,CKeyF9,CKeyF10,
                   MKeyReturn,MKeyBackspace,MKeyDelete,
                   MKeyTab,MKeyHelp,MKeyEsc,
                   MKeyCsrUp,MKeyCsrDown,MKeyCsrLeft,
```

```
                  MKeyCsrRight,
                  MKeyF1,MKeyF2,MKeyF3,MKeyF4,MKeyF5,
                  MKeyF6,MKeyF7,MKeyF8,MKeyF9,MKeyF10,
                  NoKey,NormalKey);

PROCEDURE WaitUser(s : Screen;act : UserAct);

PROCEDURE CheckUser(s : Screen;act : UserAct):BOOLEAN;

PROCEDURE LastAction(s : Screen):UserActions;

PROCEDURE GetKey(s : Screen;VAR c : CHAR);

PROCEDURE GetExtKey(s : Screen;VAR type : KeyTypes;
                               VAR code : CHAR);

PROCEDURE GetMousePos(s : Screen;VAR x,y : INTEGER);

PROCEDURE MousePressed(s : Screen):BOOLEAN;

PROCEDURE MouseClick(s : Screen;VAR x,y : INTEGER);

GROUP
  All  =  UserActions,UserAct,KeyTypes,WaitUser,CheckUser,
          LastAction,GetKey,GetExtKey,GetMousePos,
          MousePressed,MouseClick,WasDouble;

END GfxInput.
```


Typen:

UserActions, UserAct Aufzählungstyp und dazugehöriges Set, der die verschiedenen Eingabearten des Benutzers enthält. Dazu zählen:

- `userKey` ⇔Tastendruck.
- `userMouse` ⇔Mouseklick

KeyTypes Aufzählungstyp, der sämliche Funktionstasten enthält:

`KeyReturn`	:	Die Returntaste.
`KeyBackspace`	:	Die Backspacetaste (Rückschrittaste).
`KeyDelete`	:	Die Deletetaste.
`KeyTab`	:	Die Tabulatortaste.
`KeyHelp`	:	Die Helptaste.
`KeyEsc`	:	Die Escapetaste.
`KeyCsrUp`	:	Die Cursorhochtaste.
`KeyCsrDown`	:	Die Cursorruntertaste.
`KeyCsrLeft`	:	Die Cursorlinkstaste.
`KeyCsrRight`	:	Die Cursorrechtstaste.
`KeyF1`	:	Die Funktionstaste F1.
.	:	
.	:	
`KeyF10`	:	Die Funktionstaste F10.

Ein vorangestelltes „S“ bedeutet zusammen mit **SHIFT**.
Ein vorangestelltes „A“ bedeutet zusammen mit **Alt**.
Ein vorangestelltes „C“ bedeutet zusammen mit **Ctrl**.
Ein vorangestelltes „M“ bedeutet zusammen mit *A*.

Uns ist klar, das diese Verfahren nicht optimal und nicht mehr zeitgemäß ist. Diese Module stammen jedoch noch aus der Anfangszeit des Compilers, und da wir keine Zeit hatten sie zu ersetzen, und dennoch eine einfache Schnittstelle zur Graphik des Amigas bieten wollten, beschlossen wir sie noch einmal mitauszuliefern.

War die letzte Eingabe kein Tastendruck, erhält man den Wert `NoKey`. Handelte es sich um einen normalen Tastendruck, erhält man `NormalKey`.

PROCEDURE `WaitUser(s : Screen;act : UserAct);`

Funktion: Wartet, bis die angegebenen Eingabart ausgeführt wurde. Ist die Eingabe bereits geschehen, kehrt das Programm sofort zurück. Die Eingabe wird nicht gelöscht.

Parameter:

`s`	⇐	Screen, von dem die Eingabe erwartet wird.
`act`	⇐	Benutzereingabeart, auf die gewartet werden soll.

PROCEDURE `CheckUser(s : Screen;act : UserAct):BOOLEAN;`

Funktion: Prüft, ob eine der angegebenen Eingaben getätigt wurde, kehrt aber sofort wieder zurück. Diese Funktion sollte nur verwendet werden, wenn innerhalb einer Programmschleife kontroliert werden soll, ob in der Zwischenzeit eine Eingabe geschehen ist. Bitte warten Sie nicht mit dieser Funktion auf eine Eingabe, da solches „Polling“ in einem Multitasking-System anderen Programmen unnötig Rechenzeit stiehlt. Verwenden Sie stattdessen `WaitUser`.

Parameter:

`s`	⇐	Screen, der auf eine Eingabe geprüft werden soll.
`act`	⇐	Benutzereingabeart, auf die geprüft werden soll.
	⇒	`TRUE`, wenn eine solche Eingabe vorlag.

PROCEDURE `LastAction(s : Screen):UserActions;`

Funktion: Stellt fest, von welcher Art die letzte Eingabe war. Wurde die letzte Eingabe schon gelöscht, wartet die Funktion auf die nächste Eingabe.

Parameter:

`s`	⇐	Screen, der auf die letzte Eingabe geprüft werden soll.
	⇒	Typ der letzten Eingabe.

PROCEDURE `GetKey(s : Screen;`**VAR** `c : CHAR);`

Funktion: Wartet, bis der Benutzer eine Taste, der ein ASCII-Code zugeordnet ist, also keine Funktionstaste, drückt. War schon eine gedrückt, kehrt die Prozedur sofort zurück.

Parameter:

`s`	⇐	Screen, von dem die Eingabe gelesen werden soll.
`c`	⇒	Gelesenes ASCII-Zeichen.

```
PROCEDURE GetExtKey(s : Screen;VAR type : KeyTypes;
                               VAR code : CHAR);
```

Funktion: Wartet auf eine Tatstureingabe. Liegt schon eine vor, kehrt die Prozedur sofort zurück.

Parameter:

`s`	⇐	Screen, von dem die Eingabe gelesen werden soll.
`type`	⇒	Art der Taste.
`code`	⇒	ASCII-Code der Taste, falls es sich um eine normale Zeichentaste handelte.

```
PROCEDURE GetMousePos(s : Screen;VAR x,y : INTEGER);
```

Funktion: Liest die aktuelle Position der Maus ein.

Parameter:

s	⇐	Screen, von dem die Eingabe gelesen werden soll.
x	⇒	x-Position der Maus.
y	⇒	y-Position der Maus.

```
PROCEDURE MousePressed(s : Screen):BOOLEAN;
```

Funktion: Stellt fest, ob der linke Mausknopf gerade gedrückt ist.

Parameter:

s	⇐	Screen, von dem die Eingabe gelesen werden soll.
	⇒	`TRUE` falls der Knopf gedrückt ist.

```
PROCEDURE MouseClick(s : Screen;VAR x,y : INTEGER);
```

Funktion: Holt die Position des letzten Mausklicks. Hat noch kein Klick stattgefunden, wartet die Routine, bis geklickt wird.

Parameter:

s	⇐	Screen, von dem die Eingabe gelesen werden soll.
x	⇒	x-Position des letzten Mausklicks.
y	⇒	y-Position des letzten Mausklicks.

7.13 GfxPseudo3D

Dieses Modul ermöglicht es Zeichen, Strings, gefüllte Rechtecke, und gefüllte Kreise mit scheinbar dreidimensionalem Aussehen zu erzeugen. Das heißt, das die Objekte leicht erhaben oder vertieft wirken. Damit lassen sich dann so nette Spielerein wie z. B. das Programm Puzzle, das Sie auf der PD-Diskette finden, erzeugen.

```
DEFINITION MODULE GfxPseudo3D;

FROM GfxScreen IMPORT Screen;


TYPE
  Color3D  = RECORD
               topLeft,
               bottomRight,
               normal       : SHORTCARD;
             END;


PROCEDURE Write3D(s   : Screen;
                  col : Color3D;
                  x,y : INTEGER;
                  c   : CHAR);


PROCEDURE Write3DString(    s   : Screen;
                            col : Color3D;
                            x,y : INTEGER;
                        REF str : STRING);


PROCEDURE WriteQ3DString(   s   : Screen;
                            col : Color3D;
                            x,y : INTEGER;
                        VAR str : STRING);
```

```
PROCEDURE Box3D(s     : Screen;
                col   : Color3D;
                x1,y1,
                x2,y2 : INTEGER);

PROCEDURE Circle3D(s    : Screen;
                   col  : Color3D;
                   x,y,
                   r    : INTEGER);

GROUP
  All   = Color3D,Write3D,Screen,Write3DString,
          WriteQ3DString,Box3D,Circle3D;

END GfxPseudo3D.
```

Typen:

Color3D Farbstruktur für Pseudo3d-Darstellung:

topLeft	: Farbstift für linke und obere Kante.
bottomRight	: Farbe für untere und rechte Kante.
normal	: Farbe der Oberfläche.

Abhängig von der Farbwahl, erscheint ein Objekt erhaben oder vertieft.


```
PROCEDURE Write3D(s   : Screen;
                  col : Color3D;
                  x,y : INTEGER;
                  c   : CHAR);
```

Funktion: Schreibt ein Zeichen in 3D.

Parameter:

`s`	⇐ Screen, auf den gezeichnet werden soll.
`col`	⇐ Farbstruktur des Objekts.
`x`	⇐ x-Position, an der das Zeichen geschrieben werden soll.
`y`	⇐ y-Position, an der das Zeichen geschrieben werden soll.
`c`	⇐ Zeichen, das geschrieben werden soll.

```
PROCEDURE Write3DString(    s   : Screen;
                            col : Color3D;
                            x,y : INTEGER;
                        REF str : STRING);
```

Funktion: Schreibt einen String in 3D. Dabei wird für jedes Zeichen Write3D aufgerufen.

Parameter:

`s`	⇐ Screen, auf den gezeichnet werden soll.
`col`	⇐ Farbstruktur des Objekts.
`x`	⇐ x-Position, an der der String geschrieben werden soll.
`y`	⇐ y-Position, an der der String geschrieben werden soll.
`str`	⇐ String, der geschrieben werden soll.

```
PROCEDURE WriteQ3DString(   s   : Screen;
                            col : Color3D;
                            x,y : INTEGER;
                        VAR str : STRING);
```

Funktion: Schreibt einen String in 3D. Das Ergebnis sieht anders aus als das von `Write3DString`, außderdem ist diese Routine schneller.

Parameter:

`s`	⇐	Screen, auf den gezeichnet werden soll.
`col`	⇐	Farbstruktur des Objekts.
`x`	⇐	x-Position, an der der String geschrieben werden soll.
`y`	⇐	y-Position, an der der String geschrieben werden soll.
`str`	⇐	String, der geschrieben werden soll.

```
PROCEDURE Box3D(s     : Screen;
                col   : Color3D;
                x1,y1,
                x2,y2 : INTEGER);
```

Funktion: Zeichnet ein Rechteck in 3D.

Parameter:

`s`	⇐	Screen, auf den gezeichnet werden soll.
`col`	⇐	Farbstruktur des Objekts.
`x1`	⇐	x-Position der linken oberen Ecke.
`y1`	⇐	y-Position der linken oberen Ecke.
`x2`	⇐	x-Position der rechten unteren Ecke.
`y2`	⇐	y-Position der rechten unteren Ecke.


```
PROCEDURE Circle3D(s    : Screen;
                   col  : Color3D;
                   x,y,
                   r    : INTEGER);
```

Funktion: Zeichnet einen Kreis in 3D.

Parameter:

`s`	⇐	Screen, auf den gezeichnet werden soll.
`col`	⇐	Farbstruktur des Objekts.
`x`	⇐	x-Position, des Mittelpunktes.
`y`	⇐	y-Position, des Mittelpunktes.
`r`	⇐	Radius des Kreises.

7.14 GfxScreen

Dieses Modul ist Grundlage für alle GfxModule, da sie alle auf Screens zugreifen. Ein Screen ist ein virtueller Bildschirm, z. B. hat der Editor von Cluster einen eigenen Screen. GfxScreen ermöglicht Ihnen, einen Screen so einfach wie in Basic zu öffnen, ohne daß Sie erst viel initialisieren müssen. Außerdem, als Hinweis für die Spezialisten, richtet das Modul auch gleich eine TmpRas-Struktur für die Aera-Befehle ein. Am Programmende werden alle noch geöffneten Screens geschlossen. Automatisch.

```
DEFINITION MODULE GfxScreen;

TYPE
  Screen    = DEFERRED POINTER BasicGfx.ScreenPtr;

  ColorRGB  = (red,green,blue);
  Color     = ARRAY ColorRGB OF SHORTINT;
  Palette   = ARRAY OF Color;

CONST
  Black     = Color:( 0, 0, 0);
  White     = Color:(15,15,15);
  Red       = Color:(15, 0, 0);
  Green     = Color:( 0,15, 0);
  Blue      = Color:( 0, 0,15);
  Cyan      = Color:( 0,12,12);
  Purple    = Color:(12, 0,12);
  Yellow    = Color:(15,15, 0);

PROCEDURE OpenScreen(VAR s      : Screen;
                         depth  : INTEGER;
                         hires,
                         lace   : BOOLEAN);

PROCEDURE ScreenHeight(s : Screen):INTEGER;

PROCEDURE CloseScreen(VAR s : Screen);
```



```
PROCEDURE ScreenToFront(s : Screen);

PROCEDURE ScreenToBack(s : Screen);

GROUP
  ScreenGrp   = Screen,OpenScreen,ScreenHeight,
                CloseScreen,SaveScreen,ScreenToFront,
                ScreenToBack,LoadScreen;

PROCEDURE SetPalette(    s     : Screen;
                     VAR color : Palette);

PROCEDURE SetPalettePart(    s      : Screen;
                             reg    : INTEGER;
                         VAR color  : Palette);

PROCEDURE SetColor(s     : Screen;
                   reg   : INTEGER;
                   color : Color);

PROCEDURE SetRGB(s      : Screen;
                 reg,
                 red,
                 green,
                 blue   : INTEGER);
```

```
PROCEDURE GetPalette(    s      : Screen;
                     VAR color  : Palette);

PROCEDURE GetPalettePart(    s      : Screen;
                             reg    : INTEGER;
                         VAR color  : Palette);

PROCEDURE GetColor(s    : Screen;
                   reg  : INTEGER):Color;


PROCEDURE GetRGB(s     : Screen;
                 reg   : INTEGER;
                 color : ColorRGB):INTEGER;

GROUP
  ColorGrp =  ColorRGB,Color,Palette,Black,White,
              Red,Green,Blue,Cyan,Purple,Yellow,
              SetPalette,SetPalettePart,SetColor,
              SetRGB,GetPalette,GetPalettePart,
              GetColor,GetRGB;


PROCEDURE FadeIn(    s     : Screen;
                 VAR color : Palette;
                     time  : INTEGER);


PROCEDURE FadeOut(s     : Screen;
                  time  : INTEGER);


GROUP
  All  = ScreenGrp,ColorGrp,FadeIn,FadeOut;

END GfxScreen.
```

Typen:

Screen: Ein Hidden-Type, auf was er wirklich zeigt, ist für Sie nicht interessant, eine Variable dieses Typs ist nur ein Verweis auf die Screen, und muß einigen Prozeduren übergeben werden, damit diese wissen, mit welchem Screen sie arbeiten sollen. Achtung: Übergeben Sie nie eine Variable vom Typ Screen an eine Funktion von Intuition oder Graphics, es handelt sich dabei nämlich nicht um einen normalen ScreenPtr.

Color: Feld mit drei Einträgen, nämlich dem Rot-, dem Blau- und dem Grünanteil einer Farbe.

Palette: Feld vom einzelnen Farbe, wird zur einfacheren Screenfarbeinstellung benötigt.

```
PROCEDURE OpenScreen(VAR s      : Screen;
                         depth  : INTEGER;
                         hires,
                         lace   : BOOLEAN);
```

Funktion: Öffnet einen Screen mit den angegebenen Parametern.

Parameter:

`s`	⇒	Zugriff auf die Screen, er muß bei allen weiteren Operationen auf diesen Screen angegeben werden. Kann der Screen nicht geöffnet werden, wird eine Exception ausgelöst.
`depth`	⇐	Anzahl der Bitplanes des Screens. Die Zahl der möglichen Farben ist davon abhängig, sie beträgt `2^depth`.
`hires`	⇐	Falls `TRUE`, hat der Screen eine horizontale Auflösung von 640, sonst von 320 Punkten.
`lace`	⇐	Falls `TRUE`, hat der Screen eine vertikale Auflösung von 512, sonst von 256 Punkten, dann allerdings mit dem vertrauten Interlaceflimmern.

```
PROCEDURE ScreenHeight(s : Screen):INTEGER;
```

Funktion: Liefert die Höhe eines Screens.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Höhe erfragt werden soll.
	⇒	Höhe des Screen.

```
PROCEDURE CloseScreen(VAR s : Screen);
```

Funktion: Schließt einen mit `OpenScreen` geöffnetten Screen wieder. Zwar werden auch am Programmende alle noch geöffneten Screens automatisch geschlossen, doch sollte man keinen Screen länger auf haben als nötig, da andere Programme das Chipmem benötigen könnten.

Parameter:

`s`	⇐	Zugriff auf den Screen, der geschlossen werden soll. Auf diese Vaiable darf danach nicht mehr als Screen zugegriffen werden, es sei denn amn übergibt sie erneut `OpenScreen`.

```
PROCEDURE ScreenToFront(s : Screen);
```

Funktion: Bringt den angegebenen Screen in den Bildvordergrund.

Parameter:

`s`	⇐	Zugriff auf den Screen, der in den Vordergrund gebracht werden soll.

```
PROCEDURE ScreenToBack(s : Screen);
```

Funktion: Bringt den angegebenen Screen hinter alle anderen Screens.

Parameter:

`s`	⇐	Zugriff auf den Screen, der in den Hintergrund gebracht werden soll.


```
PROCEDURE SetPalette(    s     : Screen;
                     VAR color : Palette);
```

Funktion: Setzt die angegebene Palette für den Screen. Die Palette legt man am besten als konstantes Array an. Von der Palette werden höchstens `2^depth` Color-Einträge eingetragen, wobei `depth` die Tiefe ist, die man beim Öffnen des Screens angegeben hat. In der Reihenfolge, in der die Farben in der Palette stehen, werden sie auch in die Farbregister eingetragen, die Nummer des Farbregister muß man dann bei `SetAPen` bzw. `SetBPen` angeben, um die darin liegende Farbe anzuwählen.

Parameter:

`s`	⇐	Zugriff auf den Screen, für den die Palette gelten soll.
`color`	⇐	Palette, die verwendet werden soll.

```
PROCEDURE SetPalettePart(    s      : Screen;
                             reg    : INTEGER;
                         VAR color  : Palette);
```

Funktion: Setzt einen Teil der Palette eines Screens neu. Dabei werden so viele Farbregister verändert, wie Farbeinträge in der Palette sind.

Parameter:

`s`	⇐	Zugriff auf den Screen, für den die Palette gelten soll.
`reg`	⇐	Farbnummer, ab dem die Palette geändert werden soll.
`color`	⇐	Palette, die verwendet werden soll.

```
PROCEDURE SetColor(s     : Screen;
                   reg   : INTEGER;
                   color : Color);
```

Funktion: Setzt eine Farbe eines Screens neu.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`reg`	⇐	Farbnummer, die geändert werden soll.
`color`	⇐	Farbkombination, die eingetragen werden soll.

```
PROCEDURE SetRGB(s     : Screen;
                 reg,
                 red,
                 green,
                 blue  : INTEGER);
```

Funktion: Setzt eine Farbe des Screens in Komponenten neu.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`reg`	⇐	Farbnummer, die geändert werden soll.
`red`	⇐	Neuer Rotanteil.
`green`	⇐	Neuer Grünanteil.
`blue`	⇐	Neuer Bauanteil.

```
PROCEDURE GetPalette(    s     : Screen;
                     VAR color : Palette);
```

Funktion: Holt die Farben eines Screens. Es werden nur so viele Farben ausgelesen, wie das übergebene Array Felder hat.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Palette ausgelesen werden soll.
`color`	⇒	Feld, in das die Farbeinträge eingetragen werden sollen.


```
PROCEDURE GetPalettePart(    s     : Screen;
                             reg   : INTEGER;
                         VAR color : Palette);
```

Funktion: Holt einen Ausschnitt der Screenfarben.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Palette ausgelesen werden soll.
`reg`	⇐	Farbnummer, ab der die Farben ausgelesen werden sollen.
`color`	⇒	Feld, in das die Farbeinträge eingetragen werden sollen. Es werden nur soviele Farben geholt, wie `color` Elemente enthält

```
PROCEDURE GetColor(s   : Screen;
                   reg : INTEGER):Color;
```

Funktion: Holt eine Farbe eines Screens.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`reg`	⇐	Farbnummer, die ausgelesen werden soll.
	⇒	Farbkombination.

```
PROCEDURE GetRGB(s     : Screen;
                 reg   : INTEGER;
                 color : ColorRGB):INTEGER;
```

Funktion: Holt eine einzelnen Komponente einer Farbe des Screens.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`reg`	⇐	Farbnummer, die ausgelesen werden soll.
`color`	⇐	Farbkomponente, die ausgelesen werden soll.
	⇒	Intesität dieser Komponente dieser Farbe.

```
PROCEDURE FadeIn(    s     : Screen;
                 VAR color : Palette;
                     time  : INTEGER);
```

Funktion: Blendet einen Bildschirm von schwarz aus ein.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`color`	⇐	Palette, die der Screen am Ende haben soll.
`time`	⇐	Zeit in 1/50 Sekunden, die die Einblendung dauern soll.

```
PROCEDURE FadeOut(s     : Screen;
                  time  : INTEGER);
```

Funktion: Blendet einen Screen langsam nach schwarz hin aus.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`time`	⇐	Zeit in 1/50 Sekunden, die der Vorgang dauern soll.

7.15 GfxShape

Mit den Routinen dieses Moduls können Sie auf einfache Weise rechteckige Bereiche eines Screens, sogenannte Shapes, auschneiden und an anderer Stelle wieder einfügen. Die Routinen sind sicherlich recht rudimentär, jedoch kann man mit ihnen erstaunliches bewirken, wie man bei unserem Demo-Programm Puzzle sehen kann.

```
DEFINITION MODULE GfxShape;

FROM GfxScreen IMPORT Screen;

TYPE

  Shape     = HIDDEN;

PROCEDURE GetShape(    s      : Screen;
                   VAR shp    : Shape;
                       x1,y1,
                       x2,y2  : INTEGER);

PROCEDURE PutShape(s    : Screen;
                   shp  : Shape;
                   x,y  : INTEGER);

PROCEDURE FreeShape(shp : Shape);

PROCEDURE LoadShape(VAR shp   : Shape;
                    REF path  : STRING);

PROCEDURE SaveShape(    shp   : Shape;
                    REF path  : STRING);

PROCEDURE SizeShape(    sshp : Shape;
                    VAR dshp : Shape;
                        newx,
                        newy : INTEGER);
```

```
PROCEDURE Sync(s : Screen);

GROUP
  All   = Screen,Shape,GetShape,PutShape,FreeShape,
          LoadShape,SaveShape,Sync,SizeShape;

END GfxShape.
```

Typen:

Shape Da Sie mehrere Shapes auf einmal verwenden können, müssen Sie `GfxShape` mitteilen, auf welchen sich ein Befehl beziehen soll. Shape übt damit eine ähnliche Funktion wie „`Screen`" in `GfxScreen` aus.


```
PROCEDURE GetShape(     s       : Screen;
                    VAR shp     : Shape;
                        x1,y1,
                        x2,y2   : INTEGER);
```

Funktion: Holt einen Bildschirmausschnitt. Falls die übergebenen Koordinaten nicht der erwarteten Reihenfolge ensprechen, werden diese vertauscht.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`shp`	⇒	Zugriff auf das Shape. Diesen müssen Sie bei allen weiteren Operationen auf das das Shape übergeben.
`x1`	⇐	x-Position der linken oberen Ecke des Bildschirmausschnitts.
`y1`	⇐	y-Position der linken oberen Ecke des Bildschirmausschnitts.
`x2`	⇐	x-Position der rechten unteren Ecke des Bildschirmausschnitts.
`y2`	⇐	y-Position der rechten unteren Ecke des Bildschirmausschnitts.

```
PROCEDURE PutShape(s    : Screen;
                   shp  : Shape;
                   x,y  : INTEGER);
```

Funktion: Setzt ein Shape auf den Screen.

Parameter:

`s`	⇐	Zugriff auf den Screen.
`shp`	⇒	Zugriff auf das Shape, das gesetzt werden soll.
`x`	⇐	x-Position der linken oberen Ecke der gewünschten Position
`y`	⇐	y-Position der linken oberen Ecke der gewünschten Position

```
PROCEDURE FreeShape(shp : Shape);
```

Funktion: Gibt den Speicher eines Shapes wieder frei. Danach darf dieser keiner Operation mehr unterzogen werden, außer man übergibt ihn erneut `GetShape`.

Parameter:

`shp`	⇐	Zugriff auf das Shape, das freigegeben werden soll.

```
PROCEDURE LoadShape(VAR shp   : Shape;
                    REF path  : STRING);
```

Funktion: Lädt ein Shape. Dabei kann kein IFF-File geladen werden, sondern nur Shapes, die mit diesem Modul gespeichert wurden.

Parameter:

`shp`	⇒	Zugriff auf das Shape.
`path`	⇐	Pfad der Datei, die geladen werden soll.

```
PROCEDURE SaveShape(    shp   : Shape;
                    REF path  : STRING);
```

Funktion: Speichert ein Shape ab. Die Speicherung erfolgt nicht im IFF-Format.

Parameter:

`shp`	⇒	Zugriff auf das Shape.
`path`	⇐	Pfad, unter dem das Shape abgespeichert werden soll.


```
PROCEDURE SizeShape(    sshp : Shape;
                    VAR dshp : Shape;
                        newx,
                        newy : INTEGER);
```

Funktion: Erzeugt die Kopie eines Shapes, mit neuer Größe. Das übergebene Shape wird dabei entsprechend skaliert.

Parameter:

`sshp`	⇐	Quellshape.
`dshp`	⇒	Zielshape.
`newx`	⇐	Neue Breite.
`newy`	⇐	Neue Höhe.

```
PROCEDURE Sync(s : Screen);
```

Funktion: Um beim Setzen eines Shapes flackern zu vermeiden, sollte man zuvor diese Routine aufrufen. Sie wartet, bis der Kathodenstrahl den unteren Rand der Screen erreicht hat. Das Zeichnen geschieht dann in der Austastlücke.

Parameter:

`s`	⇐	Zugriff auf den Screen, auf den gezeichnet werden soll.

7.16 GfxText

In diesem Modul finden Sie eine Menge Funktionen, die zur Textausgabe auf einen mit `GfxScreen` erzeugten Screen gedacht sind. Außerdem verfügt GfxText auch über Funktionen um jeden der Diskfonts zu verwenden.

```
DEFINITION MODULE GfxText;

FROM GfxScreen IMPORT Screen;

TYPE
  Font     = HIDDEN;
  Styles   = (underlined,bold,italic,extended);
  StyleSet = SET OF Styles;

  CharPtr  = POINTER TO CHAR;


PROCEDURE OpenFont(VAR font   : Font;
                   REF name   : STRING;
                       height : INTEGER);


PROCEDURE CloseFont(VAR font : Font);


PROCEDURE SetFont(s     : Screen;
                  font  : Font);


PROCEDURE GetFont(s : Screen):Font;


PROCEDURE FontHeight(font : Font):INTEGER;


PROCEDURE FontWidht(font : Font):INTEGER;


PROCEDURE CharWidth(font : Font;
                    c    : CHAR):INTEGER;


PROCEDURE IsProportional(font : Font):BOOLEAN;
```



```
GROUP
  FontGrp   = Screen,Font,OpenFont,CloseFont,
              SetFont,GetFont,FontHeight,
              FontWidht,CharWidth,IsProportional;


PROCEDURE SetStyle(s     : Screen;
                   style : StyleSet);


PROCEDURE SubStyle(s : Screen;style : Styles);


PROCEDURE AddStyle(s : Screen;style : Styles);


GROUP
  StyleGrp = Styles,StyleSet,SetStyle,
             SubStyle,AddStyle;


PROCEDURE WriteAt(s   : Screen;
                  x,y : INTEGER;
                  c   : CHAR);


PROCEDURE WriteStringAt(    s   : Screen;
                            x,y : INTEGER;
                        REF str : STRING);

PROCEDURE WriteCharsAt(s    : Screen;
                       x,y,
                       len  : INTEGER;
                       pos  : CharPtr);


GROUP
  WriteGrp  = WriteAt,WriteStringAt,WriteCharsAt,CharPtr;


GROUP
  All   = FontGrp,StyleGrp,WriteGrp;

END GfxText.
```

Typen:

Font Zugriff auf einen Systemzeichensatz.

Styles Stilrichtungen eines Fonts:

- `underlined` : Unterstrichen.
- `bold` : Fettdruck.
- `italic` : Kursive Schrift.

StyleSet Menge der möglichen Stilarten.

7.16.1 Font-Funktionen

```
PROCEDURE OpenFont(VAR font   : Font;
                   REF name   : STRING;
                       height : INTEGER);
```

Funktion: Bevor Sie einen Font verwenden könne, müssen Sie ihn öffnen. Dies geschieht mit dieser Funktion. Falls der gewünschte Font nicht geladen werden konnte, wird ein Laufzeitfehler ausgelöst.

Parameter:

`font`	⇒	Zugriff auf den Font.
`name`	⇐	Namen des Fonts.
`height`	⇐	gewünschte Höhe.

```
PROCEDURE CloseFont(VAR font : Font);
```

Funktion: Schließt einen Font wieder.

Parameter:

`font`	⇐	Zugriff auf den Font, der geschlossen werden soll. Er darf danach nicht mehr verwendet werden.

```
PROCEDURE SetFont(s    : Screen;
                  font : Font);
```

Funktion: Setzt den Font für den angegebenen Screen. Danach werden alle Textausgaben auf diesen Screen in diesem Zeichensatz ausgeführt, auch die Schreibroutinen aus `GfxPseudo3D`.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Zeichensatz geändert werden soll.
`font`	⇐	Zeichensatz, der gesetzt werden soll.

```
PROCEDURE GetFont(s : Screen):Font;
```

Funktion: Holt den aktuellen Font eines Screens.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Font geholt werden soll.
	⇒	Zugriff auf den Font des Screens.

```
PROCEDURE FontHeight(font : Font):INTEGER;
```

Funktion: Ermittelt die Höhe eines Fonts in Bildpunkten.

Parameter:

`font`	⇐	Zugriff auf den Zeichensatz.
	⇒	Höhe des Fonts.

```
PROCEDURE FontWidht(font : Font):INTEGER;
```

Funktion: Ermittelt die Breite eines Fonts in Bildpunkten. Funktioniert nur bei nicht proportionalen Fonts.

Parameter:

`font`	⇐	Zugriff auf den Zeichensatz.
	⇒	Breite eines Zeichens des Fonts.

```
PROCEDURE CharWidth(font : Font;
                    c    : CHAR):INTEGER;
```

Funktion: Liefert die Breite eines Zeichens in einem Font. Funktioniert auch mit proportionalen Fonts.

Parameter:

`font`	⇐	Zugriff auf den Zeichensatz.
`c`	⇐	Zeichen, dessen Breite ermittelt werden soll.
	⇒	Breite dieses Zeichens in Punkten.

```
PROCEDURE IsProportional(font : Font):BOOLEAN;
```

Funktion: Stellt fest, ob ein Font proportional ist.

Parameter:

`font`	⇐	Zugriff auf den Font, der überprüft werden soll.
	⇒	`TRUE` wenn er proportional ist.

7.16.2 Style-Funktionen

```
PROCEDURE SetStyle(s     : Screen;
                   style : StyleSet);
```

Funktion: Setzt die Stilart des Font für die Ausgabe.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Font bearbeitet werden soll.
`style`	⇐	Neue Stile, die ab jetzt gelten sollen.

```
PROCEDURE SubStyle(s : Screen;style : Styles);
```

Funktion: Löscht eine Stilart.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Font bearbeitet werden soll.
`style`	⇐	Stil, der gelöscht werden soll.

```
PROCEDURE AddStyle(s : Screen;style : Styles);
```

Funktion: Fügt einem Font eine Stilart hinzu.

Parameter:

`s`	⇐	Zugriff auf den Screen, dessen Font bearbeitet werden soll.
`style`	⇐	Stil, der hinzugefügt werden soll.

7.16.3 Schreib-Funktionen

```
PROCEDURE WriteAt(s   : Screen;
                  x,y : INTEGER;
                  c   : CHAR);
```

Funktion: Schreibt ein Zeichen an eine Bildschirmposition.

Parameter:

`s`	⇐	Zugriff auf den Screen, auf den geschrieben werden soll.
`x`	⇐	x-Position des Zeichens, das geschrieben werden soll.
`y`	⇐	y-Position des Zeichens, das geschrieben werden soll.
`c`	⇐	Auszugebendes Zeichen.

```
PROCEDURE WriteStringAt(    s   : Screen;
                            x,y : INTEGER;
                        REF str : STRING);
```

Funktion: Schreibt einen String an eine Bildschirmposition.

Parameter:

s	⇐	Zugriff auf den Screen, auf den geschrieben werden soll.
x	⇐	x-Position des Strings, der geschrieben werden soll.
y	⇐	y-Position des Strings, der geschrieben werden soll.
str	⇐	Auszugebender String.

```
PROCEDURE WriteCharsAt(s    : Screen;
                       x,y,
                       len  : INTEGER;
                       pos  : CharPtr);
```

Funktion: Gibt von einer bestimten Position im Speicher eine Anzahl Zeichen aus.

Parameter:

s	⇐	Zugriff auf den Screen, auf den geschrieben werden soll.
x	⇐	x-Position, an der ausgegeben werden soll.
y	⇐	y-Position, an der ausgegeben werden soll.
len	⇐	Anzahl Zeichen, die ausgegeben werden sollen.
pos	⇐	Position im Speicher, ab der ausgebeben werden soll.

7.17 GfxTurtle

Auch dieses Modul ist zum Zeichnen gedacht, allerdings wie in „LOGO“ mit einer Schildkröte (turtle). Man stellt sich dabei einen kleinen Roboter (in Schildkrötenform) vor, den man mit einfachen Befehlen wie z. B. vorwärts, rückwärts, drehen etc. steuern kann. Dieser Roboter hat nun unter sich einen Stift befestigt, mit dem er seine Spur auf Papier in verschiedenen Farben hinterlassen kann. GfxTurtle simuliert solch einen Roboter auf dem Bildschirm.

```
DEFINITION MODULE GfxTurtle;

FROM GfxScreen IMPORT Screen;


PROCEDURE Start(scr   : Screen;
                xPos,
                yPos,
                Angle : FFP;
                color : INTEGER);

PROCEDURE Up;

PROCEDURE Down;


PROCEDURE Color(reg : INTEGER);


PROCEDURE Left(Angle : FFP);

PROCEDURE Right(Angle : FFP);


PROCEDURE Forward(Dist : FFP);


PROCEDURE Backward(Dist : FFP);

GROUP
  All   = Screen,Start,Up,Down,Color,Left,Right,
          Forward,Backward;

END GfxTurtle.
```

```
PROCEDURE Start(scr   : Screen;
                xPos,
                yPos,
                Angle : FFP;
                color : INTEGER);
```

Funktion: Setzt die Turtle auf einen Ausgangspunkt.

Parameter:

`scr`	⇐	Screen, auf dem gearbeitet werden soll.
`xPos`	⇐	Horizontale Koordinate des Startpunktes.
`yPos`	⇐	Vertikale Koordinate des Startpunktes.
`Angle`	⇐	Winkel, unter dem die Schildkröte startet. Bei 0° bewegt sie sich nach rechts, bei 90° nach oben, u. s. w.. Zwischenwerte sin natürlich auch möglich.
`color`	⇐	Farbe, mit der sie am Anfang zeichnet.

```
PROCEDURE Up;
```

Funktion: Hebt den Stift der Turtle vom Grund ab, so daß bei folgenden Bewegungen keine Spur hinterlassen wird.

```
PROCEDURE Down;
```

Funktion: Senkt den Stift der Turtle auf den Grund, so daß bei folgenden Bewegungen wieder eine Spur hinterlassen wird.

```
PROCEDURE Color(reg : INTEGER);
```

Funktion: Ändert die Farbe des Stiftes.

Parameter:

`reg` ⇐ Neue Farbe.

```
PROCEDURE Left(Angle : FFP);
```

Funktion: Dreht die Turtle nach links.

Parameter:

`Angle` ⇐ Drehwinkel.

```
PROCEDURE Right(Angle : FFP);
```

Funktion: Dreht die Turtle nach rechts.

Parameter:

`Angle` ⇐ Drehwinkel.

```
PROCEDURE Forward(Dist : FFP);
```

Funktion: Bewegt die Turtle vorwärts.

Parameter:

`Dist` ⇐ Strecke, um die die Schildkröte bewegt werden soll.

```
PROCEDURE Backward(Dist : FFP);
```

Funktion: Bewegt die Turtle rückwärts.

Parameter:

`Dist` ⇐ Strecke, um die die Schildkröte bewegt werden soll.

7.18 Graphs

Dieses Modul stellt Ihnen eine Vielzahl von Funktionen zur Bearbeitung von Graphen zur Verfügung. Da einerseits nur wenige von Ihnen sich mit Graphen beschäftigen werden, andererseits Graphen im Kapitel für Fortgeschrittene Programmierer ausreichend behandelt werden, verzichte ich hier auf eine Beschreibung der einzelnen Funktionen.

```
DEFINITION MODULE Graphs;

IMPORT Lists;

EXCEPTION
  CircleFound : "Circle in graph detected";

  DEFINITION MODULE DirectedGraphs(NodePtr : POINTER TO Node;
                                   EdgePtr : POINTER TO Edge);

    DEFINITION MODULE NodeLists = Lists.BiLists(NodePtr);
    TYPE
      Edge     = RECORD
                   from,
                   to     : RECORD
                              prev,
                              next  : EdgePtr;
                              with  : NodePtr
                            END;
                 END;
      Node     = RECORD OF NodeLists.BiNode;
                   from,
                   to     : RECORD
                              first,
                              last   : EdgePtr;
                            END;
                   mark   : BOOLEAN;
                 END;
```



```
      Graph    = RECORD
                   nodes : NodeLists.BiList;
                 END;

      ApplyN   = PROCEDURE(n : NodePtr);
      ApplyE   = PROCEDURE(e : EdgePtr);
      DestructN= PROCEDURE(n : NodePtr);
      DestructE= PROCEDURE(e : EdgePtr);

    PROCEDURE Init(VAR graph : Graph);

    PROCEDURE RemoveAll(VAR graph : Graph);

    PROCEDURE DeleteAll(VAR graph : Graph);

    PROCEDURE DestructAll(VAR graph : Graph;
                              desN  : DestructN;
                              desE  : DestructE);


    PROCEDURE AddNode(VAR graph : Graph;node : NodePtr);

    PROCEDURE RemoveNode(VAR graph : Graph;
                             node  : NodePtr;
                             des   : DestructE);

    PROCEDURE DeleteNode(VAR graph : Graph;
                             node  : NodePtr);


    PROCEDURE AddEdge(VAR graph : Graph;
                          src,
                          dst   : NodePtr;
                          edge  : EdgePtr);

    PROCEDURE RemoveEdge(VAR graph : Graph;
                             edge  : EdgePtr);

    PROCEDURE DeleteEdge(VAR graph : Graph;
                             edge  : EdgePtr);
```

```
    PROCEDURE FindEdge(VAR graph : Graph;
                           src,
                           dst   : NodePtr):EdgePtr;

    PROCEDURE SortTopological(VAR graph : Graph);

    PROCEDURE ApplyNodes(VAR graph : Graph;
                             apply : ApplyN);

    PROCEDURE ApplyToEdges(VAR graph : Graph;
                               node  : NodePtr;
                               apply : ApplyE);

    PROCEDURE ApplyFromEdges(VAR graph : Graph;
                                 node  : NodePtr;
                                 apply : ApplyE);

    PROCEDURE ApplyAllEdges(VAR graph : Graph;
                                apply : ApplyE);

    PROCEDURE ApplyDepthFirst(VAR graph  : Graph;
                                  root   : NodePtr;
                                  applyN : ApplyN;
                                  applyE : ApplyE);

    PROCEDURE ApplyBreathFirst(VAR graph  : Graph;
                                   root   : NodePtr;
                                   applyN : ApplyN;
                                   applyE : ApplyE);

    PROCEDURE MarkNodes(VAR graph : Graph);

    PROCEDURE ApplyMarked(VAR graph : Graph;
                              apply : ApplyN);

  END DirectedGraphs;

END Graphs.
```


7.19 IFFPictures

Dieses Modul stellt alle Routinen zur Verfügung, um einen Screen als IFF-Datei abzuspeichern, oder ein IFF-Bild zu laden.

```
DEFINITION MODULE IFFPictures;

FROM Intuition  IMPORT ScreenPtr;

TYPE
  PackedBitMap  = HIDDEN;

VAR
  SaveCompress,
  SaveIcon      : BOOLEAN;

PROCEDURE SetToolType(REF Name : STRING);

PROCEDURE SaveILBM(Screen : ScreenPtr;VAR Name : STRING);

PROCEDURE LoadILBM(REF Namen : STRING):ScreenPtr;

PROCEDURE LoadILBM_Packed(REF Namen : STRING):PackedBitMap;

PROCEDURE UnpackILBM(pack : PackedBitMap;scr : ScreenPtr);

PROCEDURE FreePacked(pack : PackedBitMap);

END IFFPictures.
```

Variablen:

SaveCompress Hiermit können Sie bestimmen, ob der Screen gepackt (=`TRUE`) oder ungepackt (=`FALSE`) abgespeichert werden soll.

SaveIcon Bestimmt, ob zu den Dateien ein Icon mit abgespeichert werden soll.

PROCEDURE `SetToolType(`**REF** `Name : STRING);`

Funktion: Setzt den DafaultTool-Eintrag, der beim Abspeichern gesetzt werden soll.

Parameter:

`Name`	⇐	Pfad des Programms, das als Standardprogramm gesetzt werden soll.

PROCEDURE `SaveILBM(Screen : ScreenPtr;`**VAR** `Name : STRING);`

Funktion: Speichert einen Screen in ein IFF-File.

Parameter:

`Screen`	⇐	Zeiger auf den Screen, der abgespeichert werden soll.
`Name`	⇐	Pfad, unter dem der Screen abgespeichert werden soll.

PROCEDURE `LoadILBM(`**REF** `Namen : STRING):ScreenPtr;`

Funktion: Öffnet einen Screen mit den Dimensionen des zu ladenden Bildes, und lädt die angegebene IFF-Datei.

Parameter:

`Namen`	⇐	Pfad der IFF-Datei, die geladen werden soll.
	⇒	Zeiger auf den neuen Screen.

PROCEDURE `LoadILBM_Packed(`**REF** `Namen : STRING):PackedBitMap;`

Funktion: Lädt ein IFF-Bild in den Speicher, stellt es jedoch noch nicht dar.

Parameter:

`Namen`	⇐	Pfad der IFF-Datei, die geladen werden soll.
	⇒	Zugriff auf das Bild im Speicher.

PROCEDURE `UnpackILBM(pack : PackedBitMap;scr : ScreenPtr);`

Funktion: Entpackt ein in den Speicher geladenes Bild auf einen existierenden Screen. Sind die Dimensionen des geladen Bildes größer als jene des Screens, wird der Rest abgeschnitten.

Parameter:

`pack`	⇐	Zugriff auf das Bild im Speicher, das auf den Screen gebracht werden soll.
`scr`	⇐	Zeiger auf den Screen, auf den das Bild entpackt werden soll.

PROCEDURE `FreePacked(pack : PackedBitMap);`

Funktion: Gibt den Speicher eines mit `LoadILBM_Packed` geladenen Bildes frei.

Parameter:

`pack`	⇐	Zugriff auf das freizugebende Bild im Speicher.

7.20 InOut

InOut bietet nun alle Funktionen zur einfachen Ein-/Ausgabe über ein Konsolenfenster. Wurde das Programm von der Workbench oder mit „run“ gestartet, öffnet `InOut` sein eigenes Console-Fenster. Man kann damit aber nicht nur Ausgaben auf den Bildschirm, sondern auch in Dateien machen. Allerdings nur in sogenannten Streams, d. h. man kann auf die Daten nur hintereinander und nicht direkt auf einzelne zugreifen. Für komplizierte Dateioperationen sollte man auf das `FileSystem` zurückgreifen. Wundern Sie sich nicht, wenn Ausgaben nicht sofort erscheinen, da dieses Modul normalerweise mit einer gepufferten Ausgabe arbeitet. Dabei wird der Puffer nur bei einem Zeilenvorschub, oder wenn der Puffer voll ist, ausgegeben.

```
DEFINITION MODULE InOut;
(* $ A- *)
FROM Streams IMPORT Stream,Termination;
FROM ASCII   IMPORT cr,lf,eof,sp,ff;


VAR
  Return         : BOOLEAN:=TRUE;

GROUP
  ParamGrp  = Return;

PROCEDURE OpenInput(REF name : STRING);

PROCEDURE CloseInput;

PROCEDURE OpenOutput(REF name : STRING);

PROCEDURE CloseOutput;
```



```
GROUP
  RedirectionGrp  = OpenInput,CloseInput,OpenOutput,
                    CloseOutput;

TYPE
  Style = (normal,italic,underlined,bold,invers);

PROCEDURE SetStyle(style : Style;handle : Stream:=NIL);

PROCEDURE ClearStyle(handle : Stream:=NIL);

PROCEDURE SetColor(foreground,
                   background : SHORTINT;
                   handle     : Stream:=NIL);

PROCEDURE ClearWindow(handle : Stream:=NIL);


GROUP
  StyleGrp  = SetStyle,ClearStyle,SetColor,
              ClearWindow;


PROCEDURE Write(c : CHAR;handle : Stream:=NIL);

PROCEDURE WriteString(REF s : STRING;handle : Stream:=NIL);

PROCEDURE WriteMString(REF s      : ARRAY OF CHAR;
                           handle : Stream:=NIL);

PROCEDURE WriteEsc(REF s : STRING;handle : Stream:=NIL);

PROCEDURE WriteLn(handle : Stream:=NIL);

PROCEDURE WriteBuffer(handle : Stream:=NIL);
```

```
PROCEDURE WriteInt(val    : LONGINT;
                   field  : INTEGER:=0;
                   handle : Stream:=NIL);

PROCEDURE WriteHex(val    : LONGINT;
                   field  : INTEGER:=0;
                   dollar : BOOLEAN:=FALSE;
                   handle : Stream:=NIL);

PROCEDURE WriteBin(val    : LONGINT;
                   field  : INTEGER:=0;
                   sign   : BOOLEAN:=FALSE;
                   handle : Stream:=NIL);

PROCEDURE WriteCard(val     : LONGCARD;
                    field   : INTEGER:=0;
                    handle  : Stream:=NIL);

PROCEDURE WriteCardHex(val    : LONGCARD;
                       field  : INTEGER:=0;
                       dollar : BOOLEAN:=FALSE;
                       handle : Stream:=NIL);

PROCEDURE WriteCardBin(val    : LONGCARD;
                       field  : INTEGER:=0;
                       sign   : BOOLEAN:=FALSE;
                       handle : Stream:=NIL);

PROCEDURE WriteReal(val     : LONGREAL;
                    field,
                    digits : CARDINAL;
                    handle  : Stream:=NIL);

PROCEDURE WriteExpReal(val    : LONGREAL;
                       digits: CARDINAL;
                       handle : Stream:=NIL);


GROUP
  WriteGrp   = WriteString,WriteMString,WriteEsc,
               WriteLn,WriteBuffer,WriteInt,WriteHex,
               WriteReal,WriteExpReal,Write;
```



```
PROCEDURE Read(VAR c :CHAR;handle : Stream:=NIL);


PROCEDURE ReadBool(handle          : Stream:=NIL;
                   returnEqualTrue : BOOLEAN:=FALSE):BOOLEAN;


PROCEDURE ReadString(VAR s      : STRING;
                         terms  := Termination:(sp,lf,cr,ff,eof,
                                               eof,eof,eof);
                         handle : Stream:=NIL);

PROCEDURE ReadMString(VAR s      : ARRAY OF CHAR;
                          terms  := Termination:(sp,lf,cr,ff,eof,
                                                eof,eof,eof);
                          handle : Stream:=NIL);


PROCEDURE ReadShortInt(VAR val : SHORTINT;handle : Stream:=NIL);


PROCEDURE ReadInt(VAR val : INTEGER;handle : Stream:=NIL);


PROCEDURE ReadLongInt(VAR val : LONGINT;handle : Stream:=NIL);


PROCEDURE ReadShortCard(VAR val    : SHORTCARD;
                            handle : Stream:=NIL);


PROCEDURE ReadCard(VAR val    : CARDINAL;
                       handle : Stream:=NIL);


PROCEDURE ReadLongCard(VAR val    : LONGCARD;
                           handle : Stream:=NIL);
PROCEDURE ReadReal(VAR val    : REAL;
                       handle : Stream:=NIL);

PROCEDURE ReadFFP(VAR val     : FFP;
                      handle  : Stream:=NIL);

PROCEDURE ReadLongReal(VAR val    : LONGREAL;
                           handle : Stream:=NIL);
```

```
GROUP
  ReadGrp   = Read,ReadBool,ReadString,ReadMString,
              ReadShortInt,ReadInt,ReadLongInt,
              ReadShortCard,ReadCard,ReadLongCard,
              ReadReal,ReadFFP,ReadLongReal;

  StreamGrp = Streams.ExceptionGrp,Streams.GlobalGrp;

  All       = ParamGrp,WriteGrp,StyleGrp,ReadGrp,StreamGrp,
              RedirectionGrp;

END InOut.
```

Variablen:

Return Gibt an, ob am Programmende auf ein Return gewartet wird. Standardeinstellung ist TRUE.

Typen:

Stream Alle Prozeduren in diesem Modul haben einen Handle als Parameter, der zu den Handles aus dem Modul Streams kompatibel ist. Sie können hier also einen eigenen Stream übergeben, den man mit Streams geöffnet hat. Übergibt man keinen Handle, wird der aktuelle Standardstrom aus Streams genommen, existiert noch keiner, wird ein Standard-ConsoleWindow als Stream geöffnet.

Style Stilarten, in denen auf einem Consolefenster geschrieben werden kann:

- `normal` : Normale Ausgabe.
- `italic` : Kursive Schrift.
- `underlined` : Unterstrichene Ausgabe.
- `bold` : Fettdruck.
- `invers` : Invertierte Ausgabe.

7.20.1 Ein-/Ausgabeumleitungs-Funktionen

`PROCEDURE OpenInput(REF name : STRING);`

Funktion: Dient dazu, den Standardeingabekanal (normalerweise das Con-Window) auf ein anderes Gerät oder eine Datei umzulenken. Alles Leseoperationen beziehen sich dann auf diesen Kanal.

Parameter:

`name` ⇐ Name des neuen Kanals.

`PROCEDURE CloseInput;`

Funktion: Schließt einen mit `OpenInput` geöffneten Kanal wieder. Der zuvor aktive Standardeingabestrom wird danach wieder zum aktuellen.

`PROCEDURE OpenOutput(REF name : STRING);`

Funktion: Dient dazu, den Standardausgabekanal (normalerweise das Con-Window) auf ein anderes Gerät oder eine Datei umzulenken. Alles Schreiboperationen beziehen sich dann auf diesen Kanal.

Parameter:

`name` ⇐ Name des neuen Kanals.

`PROCEDURE CloseOutput;`

Funktion: Schließt einen mit `OpenOutput` geöffneten Kanal wieder. Der zuvor aktive Standardausgabestrom wird danach wieder zum aktuellen.

7.20.2 Ausgabestil-Funktionen

`PROCEDURE SetStyle(style : Style;handle : Stream:=NIL);`

Funktion: Setzt einen der Schreibstile aus dem Aufzählungstyp `Style`.

Parameter:

`style` ⇐ Schreibstil, der gesetzt werden soll.

`handle` ⇐ Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

PROCEDURE ClearStyle(handle : Stream:=NIL);

Funktion: Setzt den Standardschreibstil.

Parameter:

handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE SetColor(foreground,
                   background : SHORTINT;
                   handle     : Stream:=NIL);
```

Funktion: Setzt Vorder- und Hintergrundfarbe für ein Consol-Fenster.

Parameter:

foreground	⇐	Vordergrundfarbe, die verwendet werden soll.
background	⇐	Hintergrundfarbe, die verwendet werden soll.
handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

PROCEDURE ClearWindow(handle : Stream:=NIL);

Funktion: Löscht das Ausgabefenster, ist der Pfad auf einen Drucker umgelenkt, führt dieser einen Seitenvorschub aus.

Parameter:

handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

7.20.3 Schreibfunktionen

PROCEDURE Write(c : CHAR;handle : Stream:=NIL);

Funktion: Gibt ein Zeichen aus. Ist `c=&10` und der verwendete Strom gepuffert[9], dann wird der Puffer ausgegeben, wenn einer vorhanden ist.

[9] Was standardmäßig so ist.

Parameter:

c	⇐	Zeichen das ausgegeben werden soll.
handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

PROCEDURE WriteString(**REF** s : STRING;handle : Stream:=NIL);

Funktion: Gibt eine Zeichenkette aus. Enthält der String einen Zeilevorschub (&10), wird der Puffer ausgegeben, falls vorhanden.

Parameter:

s	⇐	String, der ausgegeben werden soll.
handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteMString(REF s      : ARRAY OF CHAR;
                           handle : Stream:=NIL);
```

Funktion: Gibt einen Modula-String aus.

Parameter:

s	⇐	Modula-String, der ausgegeben werden soll.
handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

PROCEDURE WriteEsc(**REF** s : STRING;handle : Stream:=NIL);

Funktion: Wie WriteString, jedoch wird vor dem String noch ein „Esc-Zeichen“ ausgegeben. Daher ist diese Funktion besonders gut für Esc-Sequenzen zur Druckersteuerung zu gebrauchen.

Parameter:

s	⇐	String, der ausgegeben werden soll.
handle	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

PROCEDURE `WriteLn(handle : Stream:=NIL);`

Funktion: Führt einen Zeilenvorschub durch. Dabei wird der Puffer ausgegeben, sofern die Ausgabe gepuffert ist.

PROCEDURE `WriteBuffer(handle : Stream:=NIL);`

Funktion: Erzwingt die Ausgabe des Puffers. Wenn keiner existiert, entsteht ein Laufzeitfehler.

Parameter:

`s`	⇐	String, der ausgegeben werden soll.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteInt(val    : LONGINT;
                   field  : INTEGER:=0;
                   handle : Stream:=NIL);
```

Funktion: Gibt eine Integerzahl aus.

Parameter:

`val`	⇐	Intergerzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteHex(val    : LONGINT;
                   field  : INTEGER:=0;
                   dollar : BOOLEAN:=FALSE;
                   handle : Stream:=NIL);
```

Funktion: Gibt eine Integerzahl als Hexwert aus, sonst wie WriteInt. Bei negativen Werten wird die Zahl nicht im Zweier-Komplement, sondern mit Vorzeichen ausgegeben.

Parameter:

`val`	⇐	Intergerzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`dollar`	⇐	Flag, ob bei der Ausgabe der Zahl ein „$“ vorangestellt werden soll.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteBin(val    : LONGINT;
                   field  : INTEGER:=0;
                   sign   : BOOLEAN:=FALSE;
                   handle : Stream:=NIL);
```

Funktion: Gibt eine Integerzahl als Binärzahl aus, sonst wie WriteInt.

Parameter:

`val`	⇐	Intergerzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`sign`	⇐	Flag, ob bei der Ausgabe der Zahl ein „%“ vorangestellt werden soll.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteCard(val     : LONGCARD;
                    field   : INTEGER:=0;
                    handle  : Stream:=NIL);
```

Funktion: Gibt eine Cardinalzahl aus.

Parameter:

`val`	⇐	Cardinalzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteCardHex(val    : LONGCARD;
                       field  : INTEGER:=0;
                       dollar : BOOLEAN:=FALSE;
                       handle : Stream:=NIL);
```

Funktion: Gibt eine Cardinalzahl als Hexwert aus, sonst wie WriteCard.

Parameter:

`val`	⇐	Cardinalzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`dollar`	⇐	Flag, ob bei der Ausgabe der Zahl ein „$" vorangestellt werden soll.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.


```
PROCEDURE WriteCardBin(val    : LONGCARD;
                       field  : INTEGER:=0;
                       sign   : BOOLEAN:=FALSE;
                       handle : Stream:=NIL);
```

Funktion: Gibt eine Cardinalzahl als Binärzahl aus, sonst wie WriteCard.

Parameter:

`val`	⇐	Cardinalzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll. Ist `field = 0` wird die Zahl in einem genau passenden Feld positioniert. Ist `field > 0` wird die Zahl rechtsbündig, für `field < 0` linksbündig angeordnet.
`sign`	⇐	Flag, ob bei der Ausgabe der Zahl ein „%" vorangestellt werden soll.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteReal(val     : LONGREAL;
                    field,
                    digits : CARDINAL;
                    handle  : Stream:=NIL);
```

Funktion: Gibt eine Fließkommazahl aus.

Parameter:

`val`	⇐	Longrealzahl, die ausgegeben werden soll.
`field`	⇐	Größe des Feldes, in dem die Zahl positioniert werden soll, inklusive Dezimalpunkt und Vorzeichen. Die Zahl wird darin kommabündig positioniert
`digits`	⇐	Anzahl der auszugebende Nachkommastellen.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE WriteExpReal(val    : LONGREAL;
                      digits: CARDINAL;
                      handle : Stream:=NIL);
```

Funktion: Ausgabe einer Fließkommazahl in Exponetialdarstellung. Die Zahl wird linksbündig ausgegeben.

Parameter:

`val`	⇐	Longrealzahl, die ausgegeben werden soll.
`digits`	⇐	Anzahl der auszugebende Nachkommastellen.
`handle`	⇐	Alternativer Ausgabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

7.20.4 Lesefunktionen

```
PROCEDURE Read(VAR c :CHAR;handle : Stream:=NIL);
```

Funktion: Liest ein Zeichen ein. Ist der augenblickliche Eingabestrom ein interaktiver, kehrt diese Funktion erst zurück, nachdem eine Eingabe erfolgt ist, bei einem Consolefenster also erst, nachdem ein Return eingegeben wurde.

Parameter:

`c`	⇒	Gelesenes Zeichen.
`handle`	⇐	Alternativer Eingabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE ReadBool(handle          : Stream:=NIL;
                   returnEqualTrue : BOOLEAN:=FALSE):BOOLEAN;
```

Funktion: Liest einen Boolwert ein. `TRUE` wird bei folgenden Eingaben zurückgeliefert:

- „j“/„J“
- „1“
- „y“/„Y“
- „ja“/„Ja“
- „yes“, unabhängig von der Groß/Kleinschreibung.
- „true“, unabhängig von der Groß/Kleinschreibung.
- „wahr“, unabhängig von der Groß/Kleinschreibung.

FALSE wird für folgende Eingaben zurückgeliefert:

- „n“/„N“
- „0“
- „nein“, unabhängig von der Groß/Kleinschreibung.
- „no“, unabhängig von der Groß/Kleinschreibung.
- „false“, unabhängig von der Groß/Kleinschreibung.
- „falsch“, unabhängig von der Groß/Kleinschreibung.

Parameter:

`handle` ⇐ Alternativer Eingabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

`returnEqualTrue`

⇐ Wird hier „`TRUE`“ übergeben, wird ein einfaches Return auch als `TRUE` gewertet.

⇒ Eingegebener Boolwert.

```
PROCEDURE ReadString(VAR s      : STRING;
                         terms  := Termination:(sp,lf,cr,ff,eof,
                                                eof,eof,eof);
                         handle : Stream:=NIL);
```

Funktion: Liest einen String ein. Dabei wird solange gelesen, bis ein Terminationszeichen erreicht wird. Das Terminationszeichen selbst befindet sich nicht in dem String. Es werden nur Terminationszeichen erkannt, die nicht von ‘"’ eingeschlossen sind. Will man ein Anführungszeichen eingeben, kann dies nur innerhalb von Anführungsstrichen geschehen, und dort, indem man zwei ‘"’ direkt hintereinander ohne Leerzeichen eingibt.

Parameter:

`s` ⇒ Eingelesener String.

`terms` ⇐ Array mit 8 Einträgen, das die Termiationzeichen enthält, die zum Abbruch des Einlesens führen.

`handle` ⇐ Alternativer Eingabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

```
PROCEDURE ReadMString(VAR s      : ARRAY OF CHAR;
                          terms  := Termination:(sp,lf,cr,ff,eof,
                                                 eof,eof,eof);
                          handle : Stream:=NIL);
```

Funktion: Liest einen Modula-String ein. Sonst wie `ReadString`.

```
PROCEDURE ReadShortInt(VAR val : SHORTINT;handle : Stream:=NIL);
```

Funktion: Liest eine `SHORTINT`-Zahl ein. Die Zahl wird bis zum ersten Auftreten einer Nichtziffer ausgewertet. Gibt man vor der Zahl ein „$“ an, wird die darauffolgende Zahl als Hex-Zahl ausgewertet, durch ein „%“ wird sie als Binärzahl verstanden. Drückt man als Eingabe nur auf Return, wird die in `val` übergebene Variable nicht verändert.

Parameter:

`val` ⇒ Eingelesener Wert.

`handle` ⇐ Alternativer Eingabestrom. Wird dieser Parameter nicht übergeben, wird der Standardstrom verwendet.

Die Prozeduren `ReadInt, ReadLongInt, ReadShortCard, ReadCard und ReadLongCard` werden genauso aufgerufen, und dienen dazu, Zahlen der entsprechenden Typen einzulesen. Auf eine nähere Erklärung kann wohl verzichtet werden.

Für `ReadReal, ReadFP, und ReadLongReal` gilt im Prinzip das selbe, jedoch können hier auch Eingaben in Exponentialdarstellung gemacht werden wie z. B. `1.34553E4` oder `3.43355e-12`.

7.21 Lists

Bei diesem Modul handelt es sich um eine Zusammenstellung von generischen Modulen zur Verwaltung von verketteten Listen. Da Listen im Kapitel für Fortgeschrittene Programmierer ausführlich behandelt werden, wird hier auf eine nähere Erklärung verzichtet.

```
DEFINITION MODULE Lists;


FROM FileSystem IMPORT File;
FROM Resources  IMPORT ContextPtr;

EXCEPTION
  FileTypeMismatch : "File type and current type don't match";
  ListEmpty        : "List is allready empty";

  DEFINITION MODULE BiLists(BiNodePtr : POINTER TO BiNode);

    TYPE
      BiNode    = RECORD
                    prev,
                    next  : BiNodePtr;
                  END;
      BiList    = RECORD
                    first,
                    last   : BiNodePtr
                  END;

      ApplyProc  = PROCEDURE(n : BiNodePtr);
      Destructor = PROCEDURE(n : BiNodePtr);
      Relation   = PROCEDURE(a,b : BiNodePtr):BOOLEAN;
      Check      = PROCEDURE(n : BiNodePtr):BOOLEAN;
      SaveProc   = PROCEDURE(f : File;data : BiNodePtr);
      LoadProc   = PROCEDURE(f : File):BiNodePtr;

    PROCEDURE Init(VAR l : BiList);

    PROCEDURE InsertTop(VAR l : BiList;n : BiNodePtr);

    PROCEDURE InsertBottom(VAR l : BiList;n : BiNodePtr);

    PROCEDURE InsertBefore(VAR l : BiList;succ,n : BiNodePtr);

    PROCEDURE RemoveFirst(VAR l : BiList):BiNodePtr;
```



```
    PROCEDURE RemoveLast(VAR l : BiList):BiNodePtr;

    PROCEDURE Remove(VAR l : BiList;n : BiNodePtr);

    PROCEDURE Remove_IF(VAR l : BiList;if : Check);

    PROCEDURE Remove_All(VAR l : BiList);

    PROCEDURE DeleteFirst(VAR l : BiList);

    PROCEDURE DeleteLast(VAR l : BiList);

    PROCEDURE Delete(VAR l : BiList;n : BiNodePtr);

    PROCEDURE Delete_IF(VAR l : BiList;if : Check);

    PROCEDURE Delete_All(VAR l : BiList);


    PROCEDURE Destruct_IF(VAR l    : BiList;
                              if   : Check;
                              des  : Destructor);

    PROCEDURE Destruct_All(VAR l    : BiList;
                               des  : Destructor);


    PROCEDURE Apply(VAR l : BiList;proc : ApplyProc);

    PROCEDURE Sort(VAR l : BiList;greater : Relation);

    PROCEDURE n_th(VAR l : BiList;n : INTEGER):BiNodePtr;

    PROCEDURE Count(VAR l : BiList ): INTEGER;

    PROCEDURE isEmpty(VAR l : BiList):BOOLEAN;

    PROCEDURE Find(VAR l : BiList;equal : Check):BiNodePtr;

    PROCEDURE FindNext(VAR l      : BiList;
                           equal  : Check;
                           start  : BiNodePtr):BiNodePtr;

    PROCEDURE Append(VAR dest,arg : BiList);

    PROCEDURE Save(VAR l : BiList;f : File;part : SaveProc);

    PROCEDURE Load(VAR l : BiList;f : File;part : LoadProc);
```

```
END BiLists;

DEFINITION MODULE SingleLists(NodePtr : POINTER TO Node);

  TYPE
    Node      = RECORD
                  next : NodePtr
                END;
    List      = RECORD
                  first,
                  last   : NodePtr
                END;

    ApplyProc  = PROCEDURE(n : NodePtr);
    Destructor = PROCEDURE(n : NodePtr);
    Relation   = PROCEDURE(a,b : NodePtr):BOOLEAN;
    Check      = PROCEDURE(n : NodePtr):BOOLEAN;
    SaveProc   = PROCEDURE(f : File;data : NodePtr);
    LoadProc   = PROCEDURE(f : File):NodePtr;

  PROCEDURE Init(VAR l : List);

  PROCEDURE InsertTop(VAR l : List;n : NodePtr);

  PROCEDURE InsertBottom(VAR l : List;n : NodePtr);

  PROCEDURE InsertAfter(VAR l : List;pred,n : NodePtr);

  PROCEDURE RemoveFirst(VAR l : List):NodePtr;

  PROCEDURE Remove(VAR l : List;n : NodePtr);

  PROCEDURE Remove_IF(VAR l : List;if : Check);

  PROCEDURE Remove_All(VAR l : List);


  PROCEDURE DeleteFirst(VAR l : List);

  PROCEDURE Delete(VAR l : List;n : NodePtr);

  PROCEDURE Delete_IF(VAR l : List;if : Check);

  PROCEDURE Delete_All(VAR l : List);


  PROCEDURE Destruct_IF(VAR l   : List;
```



```
                            if  : Check;
                            des : Destructor);

  PROCEDURE Destruct_All(VAR l : List;des : Destructor);


  PROCEDURE Find(VAR l : List;equal : Check):NodePtr;

  PROCEDURE FindNext(VAR l      : List;
                         equal  : Check;
                         start  : NodePtr):NodePtr;


  PROCEDURE Apply(VAR l : List;proc : ApplyProc);

  PROCEDURE Sort(VAR l : List;greater : Relation);

  PROCEDURE n_th(VAR l : List;n : INTEGER):NodePtr;

  PROCEDURE Append(VAR dest,arg : List);


  PROCEDURE Save(VAR l : List;f : File;part : SaveProc);


  PROCEDURE Load(VAR l : List;f : File;part : LoadProc);
END SingleLists;
```

```
DEFINITION MODULE SortedBiLists(BiNodePtr : POINTER TO BiNode);

  EXCEPTION
    RelationMismatch : "Relations in sorted lists don't match";

  DEFINITION MODULE BiLists = BiLists(BiNodePtr);

  FROM BiLists IMPORT Remove,RemoveFirst,RemoveLast,
                      Remove_IF,Remove_All,
                      Delete,DeleteFirst,DeleteLast,
                      Delete_IF,Delete_All,
                      Destruct_IF,Destruct_All,
                      Relation;

  TYPE
    BiNode   = RECORD OF BiLists.BiNode END;
    BiList   = RECORD OF BiLists.BiList
                 greater : Relation
               END;

    SaveProc  = PROCEDURE(f : File;data : BiNodePtr);
    LoadProc   = PROCEDURE(f : File):BiNodePtr;

  PROCEDURE Init(VAR l : BiList;greater : Relation);

  PROCEDURE Insert(VAR l : BiList;n : BiNodePtr);

  PROCEDURE Mix(VAR dest,arg : BiList);

  PROCEDURE Save(VAR l : BiList;f : File;part : SaveProc);


  PROCEDURE Load(VAR l : BiList;f : File;part : LoadProc);
END SortedBiLists;
```



```
DEFINITION MODULE TextLists(TextNodePtr : POINTER TO TextNode);

  DEFINITION MODULE TL = SortedBiLists(TextNodePtr);

  TYPE
    TextList     = TL.BiList;
    TextNode     = RECORD OF TL.BiNode
                     name    : CLASSPTR TO STRING;
                     special : BOOLEAN;
                   END;


  PROCEDURE CmpStrings(a,b : TextNodePtr):BOOLEAN;

  PROCEDURE CmpStringsCaps(a,b : TextNodePtr):BOOLEAN;

END TextLists;

DEFINITION MODULE CursorLists(type : ANYPTR);

  EXCEPTION
    NoCursor : "No cursor set";
    NotFound : "Entry not found";

  TYPE
    PtrNodePtr = POINTER TO PtrNode;

  DEFINITION MODULE CList = BiLists(PtrNodePtr);

  TYPE
    PtrNode    = RECORD OF CList.BiNode
                   data  : type
                 END;
    List       = RECORD OF CList.BiList
                   mark,
                   cursor   : PtrNodePtr;
                 END;

    ApplyProc  = PROCEDURE(n : type);
    Destructor = PROCEDURE(n : type);
    Relation   = PROCEDURE(a,b : type):BOOLEAN;
    Check      = PROCEDURE(n : type):BOOLEAN;
    SaveProc   = PROCEDURE(f : File;data : type);
    LoadProc   = PROCEDURE(f : File):type;

  PROCEDURE Init(VAR l : List);

  PROCEDURE Get(VAR l : List):type;
```

```
PROCEDURE InsertTop(VAR l : List;t : type);
PROCEDURE InsertBottom(VAR l : List;t : type);
PROCEDURE InsertBefore(VAR l : List;t : type);
PROCEDURE InsertAfter(VAR l : List;t : type);

PROCEDURE RemoveFirst(VAR l : List):type;
PROCEDURE RemoveLast(VAR l : List):type;
PROCEDURE RemoveAct(VAR l : List):type;
PROCEDURE Remove(VAR l : List;t : type);
PROCEDURE Remove_IF(VAR l : List;if : Check);
PROCEDURE Remove_All(VAR l : List);

PROCEDURE DeleteFirst(VAR l : List);
PROCEDURE DeleteLast(VAR l : List);
PROCEDURE Delete(VAR l : List;t : type);
PROCEDURE DeleteAct(VAR l : List);
PROCEDURE Delete_IF(VAR l : List;if : Check);
PROCEDURE Delete_All(VAR l : List);

PROCEDURE Destruct_IF(VAR l   : List;
                          if  : Check;
                          des : Destructor);
PROCEDURE Destruct_All(VAR l : List;des : Destructor);

PROCEDURE Apply(VAR l : List;proc : ApplyProc);
PROCEDURE Sort(VAR l : List;greater : Relation);
PROCEDURE Append(VAR dest,arg : List);
```



```
  PROCEDURE Dup(VAR dest,arg : List);

  PROCEDURE Find(VAR l : List;equal : Check);
  PROCEDURE FindNext(VAR l : List;equal : Check);
  PROCEDURE Mark(VAR l : List);
  PROCEDURE GoMark(VAR l : List);
  PROCEDURE Next(VAR l : List);
  PROCEDURE Prev(VAR l : List);
  PROCEDURE First(VAR l : List);
  PROCEDURE Last(VAR l : List);
  PROCEDURE n_th(VAR l : List; n : INTEGER);
  PROCEDURE Go(VAR l : List;t : type);
  PROCEDURE HasCursor(VAR l : List):BOOLEAN;
  PROCEDURE Save(VAR l : List;f : File;part : SaveProc);
  PROCEDURE Load(VAR l : List;f : File;part : LoadProc);
  PROCEDURE GarbageCollect;
 END CursorLists;
END Lists.
```

Exceptions:

FileTypeMismatch Wird durch Load ausgelöst, wenn versucht wird eine Liste aus einem File zu laden, in dem keine abgespeichert war.

ListEmpty Falls versucht wurde ein Element aus einer leeren Liste zu entfernen.

7.21.1 BiLists

Enthält Routinen zur Verwaltung von doppelt verketteten Listen. Alle Elemente der Liste müssen Nachfolger des Typs „BiNode“ sein.

Typen:

BiList Listenkopf, der folgende Elemente enthält:

- `first` : Zeiger auf ersten Knoten der Liste, NIL, wenn die List leer ist.
- `last` : Zeiger auf letzten Knoten der Liste, NIL, wenn die Liste leer ist.

BiNode Listenknoten, der folgende Elemente enthält:

- `next` : Zeiger auf nächsten Knoten der Liste, NIL, wenn es selbst der letzte Knoten ist.
- `prev` : Zeiger auf vorherigen Knoten der Liste, NIL, wenn es selbst der erste Knoten ist.

```
PROCEDURE Init(VAR l : BiList);
```

Funktion: Initialisiert eine Liste. Das muß geschehen, bevor das erste Element in die Liste eingefügt wird.

Parameter:

`l` $\Rightarrow$ Liste, die initialisiert werden soll.


```
PROCEDURE InsertTop(VAR l : BiList;n : BiNodePtr);
```

Funktion: Fügt einen Knoten am Anfang der Liste ein.

Parameter:

`l` $\Leftrightarrow$ Liste, in die der Knoten eingefügt werden soll.

`n` $\Leftarrow$ Knoten, der eingefügt werden soll.

```
PROCEDURE InsertBottom(VAR l : BiList;n : BiNodePtr);
```

Funktion: Fügt einen Knoten am Ende der Liste ein.

Parameter:

`l` $\Leftrightarrow$ Liste, in die der Knoten eingefügt werden soll.

`n` $\Leftarrow$ Knoten, der eingefügt werden soll.

```
PROCEDURE InsertAfter(VAR l : BiList;pred,n : BiNodePtr);
```

Funktion: Fügt einen Knoten direkt nach einem bereits in der Liste vorhandenen ein.

Parameter:

`l` $\Leftrightarrow$ Liste, in die der Knoten eingefügt werden soll.

`pred` $\Leftarrow$ Knoten, hinter dem der neue Knoten eingefügt werden soll.

`n` $\Leftarrow$ Knoten, der eingefügt werden soll.

```
PROCEDURE InsertBefore(VAR l : BiList;succ,n : BiNodePtr);
```

Funktion: Fügt einen Knoten direkt vor einem bereits in der Liste vorhandenen ein.

Parameter:

`l` $\Leftrightarrow$ Liste, in die der Knoten eingefügt werden soll.

`succ` $\Leftarrow$ Knoten, vor dem der neue Knoten eingefügt werden soll.

`n` $\Leftarrow$ Knoten, der eingefügt werden soll.

PROCEDURE `RemoveFirst(`**VAR** `l : BiList):BiNodePtr;`

Funktion: Entfernt das erste Element aus der Liste, und gibt dieses zurück.

Parameter:

`l`	⇔	Liste, aus der das Element entfernt werden soll.
	⇒	Element, das entfernt wurde.

PROCEDURE `RemoveLast(`**VAR** `l : BiList):BiNodePtr;`

Funktion: Entfernt das letzte Element einer Liste und gibt dieses zurück.

Parameter:

`l`	⇔	Liste, aus der das Element entfernt werden soll.
	⇒	Element, das entfernt wurde.

PROCEDURE `Remove(`**VAR** `l : BiList;n : BiNodePtr);`

Funktion: Entfernt ein übergebenes Element aus der Liste.

Parameter:

`l`	⇔	Liste, aus der der Knoten entfernt werden soll.
`n`	⇐	Knoten, der entfernt werden soll.

```
TYPE
  Check      = PROCEDURE(n : BiNodePtr):BOOLEAN;
PROCEDURE Remove_IF(VAR l : BiList;if : Check);
```

Funktion: Entfernt alle Elemente aus der Liste, für die eine Bedingung erfüllt ist. Da nun dieses Modul nichts über die Eigenschaften seiner Knoten weiß, außer, daß sie Nachfolger von `BiNode` sind, muß der Benutzer hier eine Prozedur mit übergeben, die für jedes Listenelement aufgerufen werden kann und dann zurückgibt, ob dieses Element entfernt werden soll oder nicht.

Parameter:

`l`	⇔	Liste, die bearbeitet werden soll.
`if`	⇐	Funktion, der nacheinander die einzelnen Listenknoten übergeben werden und die `TRUE` zurückgeben muß, wenn dieses Element entfernt werden soll. Da diese Funktion als Übergabetypen nicht mehr den Typ `BiNode`, sondern den generisch ausgeprägten hat, kann sie auf alle Elemente des Knotens zugreifen.

PROCEDURE `Remove_All(`**VAR** `l : BiList);`

Funktion: Entfernt alle Elemente aus der Liste. Danach ist die Liste wieder im Zustand wie nach `Init`.

Parameter:

`l`	⇔	Liste, aus der alle Elemente entfernt werden sollen.

PROCEDURE `DeleteFirst(`**VAR** `l : BiList);`

PROCEDURE `DeleteLast(`**VAR** `l : BiList);`

PROCEDURE `Delete(`**VAR** `l : BiList;n : BiNodePtr);`

PROCEDURE `Delete_IF(`**VAR** `l : BiList;if : Check);`

PROCEDURE `Delete_All(`**VAR** `l : BiList);`

Diese Funktionen werden wie die entsprechenden `Remove...`-Funktionen aufgerufen. Der Unterschied ist, daß die Knoten anschließend mit `Dispose` freigegeben werden. Diese Funktionen eignen sich also nur dann zum Freigeben eines Knotens, wenn dieser nur aus Speicher besteht und keine anderen Resourcen enthält (wie z. B. ein `FileHandle`), die erst noch freigegeben werden müssen. In einem solchen Fall kann man jedoch auf die nahezu gleichartigen Destruct-Funktionen zurückgreifen:

```
TYPE
  Destructor = PROCEDURE(n : BiNodePtr);
PROCEDURE Destruct_IF(VAR l    : BiList;
                          if   : Check;
                          des  : Destructor);

PROCEDURE Destruct_All(VAR l    : BiList;
                           des  : Destructor);
```

Diese Funktionen arbeiten wie `Remove`, jedoch wird jeweils eine Prozedur mitübergeben, die die Freigabe des Knotens vornehmen muß.

```
TYPE
  ApplyProc  = PROCEDURE(n : BiNodePtr);
PROCEDURE Apply(VAR l : BiList;proc : ApplyProc);
```

Funktion: Will man alle Elemente einer Liste auf einmal bearbeiten, kann man diese Prozedur verwenden. Die angegebene Prozedur `proc` wird einmal für jedes Element aufgerufen, mit dem Element als Parameter.

Parameter:

`l`	⇔	Liste, die bearbeitet werden soll.
`proc`	⇐	Prozedur, die für jedes Element aufgerufen werden soll.

```
TYPE
  Relation   = PROCEDURE(a,b : NodePtr):BOOLEAN;
PROCEDURE Sort(VAR l : BiList;greater : Relation);
```

Funktion: Die Liste wird anhand einer Ordnungsrelation sortiert.

Parameter:

`l`	⇔	Liste, die sortiert werden soll.
`greater`	⇐	Funktion, die die Ordnung zwischen zwei übergebenen Knoten angibt. Sie gibt `TRUE` zurück, wenn `a` vor `b` sortiert werden soll, sonst `FALSE` – also auch bei Gleichheit: `(a<b)` ↔ `greater(a, b) = TRUE`


```
PROCEDURE n_th(VAR l : BiList;n : INTEGER):BiNodePtr;
```

Funktion: Liefert das n-te Element einer Liste.

Parameter:

`l`	⇐	Liste, auf die zugriffen werden soll.
`n`	⇐	Nummer des Elements, das gesucht werden soll.
	⇒	n-ter Knoten oder `NIL`, wenn die Liste kürzer war.

```
PROCEDURE Count(VAR l : BiList ): INTEGER;
```

Funktion: Liefert die Anzahl Elemente in einer Liste.

Parameter:

`l`	⇐	Liste, die untersucht werden soll.
	⇒	Anzahl Elemente.

```
PROCEDURE isEmpty(VAR l : BiList):BOOLEAN;
```

Funktion: Liefert `TRUE`, wenn die Liste leer ist.

```
TYPE
  Check      = PROCEDURE(n : BiNodePtr):BOOLEAN;
PROCEDURE Find(VAR l : BiList;equal : Check):BiNodePtr;
```

Funktion: Liefert das erste Element, für das eine Bedingung erfüllt ist. Da das Modul nichts über die einzelnen Elemente eines Knotens weiß, muß auch hier eine Prozedur mit übergeben werden.

Parameter:

`l`	⇐	Liste, in der gesucht werden soll.
`equal`	⇐	Prozedur, die `TRUE` zurückliefert, wenn es sich bei dem ihr übergebenen Knoten um den gesuchten handelt.
	⇒	Zeiger auf den ersten Knoten, auf den die Bedingung zutrifft oder NIL, wenn es keinen solchen Knoten gab.

```
PROCEDURE FindNext(VAR l      : BiList;
                       equal  : Check;
                       start  : BiNodePtr):BiNodePtr;
```

Funktion: Wie `Find`, jedoch kann hier ab einem bestimmten Knoten gesucht werden und nicht nur vom Listenanfang aus.

Parameter:

`l`	⇐	Liste, in der gesucht werden soll.
`equal`	⇐	Prozedur, die `TRUE` zurückliefert, wenn es sich bei dem ihr übergebenen Knoten um den gesuchten handelt.
`start`	⇐	Knoten, bei dem mit der Suche begonnen werden soll.
	⇒	Zeiger auf den ersten Knoten, auf den die Bedingung zutrifft, oder `NIL`, wenn kein Knoten gefunden wurde.

```
PROCEDURE Append(VAR dest,arg : BiList);
```

Funktion: Fügt eine Liste an eine andere an. Die angehängte Liste enthält danach keine Elemente mehr.

Parameter:

`dest`	⇔	Liste, an die angehängt werden soll.
`arg`	⇔	Liste, die angehängt werden soll und danach leer ist.


```
TYPE
  SaveProc   = PROCEDURE(f : File;data : BiNodePtr);
PROCEDURE Save(VAR l : BiList;f : File;part : SaveProc);
```

Funktion: Sichert eine Liste in eine File.

Parameter:

`l`	⇐	Liste, die gesichert werden soll.
`f`	⇐	Zugriff auf ein mittels `FileSystem` geöffnetes File.
`part`	⇐	Prozedur, die einen einzelnen Knoten in das übergebene File schreibt. Wird für jeden Knoten einzeln aufgerufen.

```
TYPE
  LoadProc   = PROCEDURE(f : File):BiNodePtr;
PROCEDURE Load(VAR l : BiList;f : File;part : LoadProc);
```

Funktion: Lädt eine Liste aus einem File.

Parameter:

`l`	⇐	Liste, die gesichert werden soll.
`f`	⇐	Zugriff auf ein mittels `FileSystem` geöffnetes File.
`part`	⇐	Prozedur, die einen einzelnen Knoten erzeugt, und seine Daten aus dem übergebenen File liest. Wird für alle Knoten einzeln aufgerufen.

7.21.2 SingleLists

Hierbei handelt es sich um ein Modul zur Verwaltung von einfach verketteten Listen (s. Kapitel 4). Da die Prozeduren in ihrer Funktion und Aufruf mit denen von `BiLists` übereinstimmen, wird hier auf eine Beschreibung verzichtet und auf das vorhergehende Modul verwiesen. Alle Elemente, die eingefügt werden sollen, müssen Nachfolger des Typs `Node` sein.

7.21.3 SortedBiLists

Hierbei handelt es sich wiederum um eine doppelt verkettete Liste, jedoch mit der zusätzlichen Eigenschaft, daß sie immer sortiert ist. Dies wird dadurch erzielt, daß jeder Knoten, der neu eingefügt wird, gleich richtig einsortiert wird. Auf diese Liste können alle Funktionen aus `BiLists` angewendet werden[10]. Des weiteren bietet das Modul folgende Funktionen an:

PROCEDURE `Init(`**VAR** `l : BiList;greater : Relation);`

Funktion: Initialisiert die Liste. Muß aufgerufen werden, bevor das erste Element eingehängt werden kann.

Parameter:

`l`	⇒	Liste, die initialisiert werden soll.
`greater`	⇐	Relationsprozedur, die angibt, welches von zwei übergebenen Elementen das größere ist.

PROCEDURE `Insert(`**VAR** `l : BiList;n : BiNodePtr);`

Funktion: Fügt ein Element in die Liste ein. Da es einsortiert wird, ist eine Positionsangabe nicht nötig.

Parameter:

`l`	⇔	Liste, in die eine Element eingefügt werden soll.
`n`	⇐	Knoten, der eingefügt werden soll.

[10]Interesant ist hierbei, daß dieses generische Modul selbst durch Ausprägung eines anderen generischen Moduls realisiert wurde.

PROCEDURE `Mix(`**VAR** `dest,arg : BiList);`

Funktion: Mischt zwei Listen zu einer neuen zusammen. Die Ordnung bleibt dabei erhalten.

Parameter:

`dest`	⇔	Liste, in die die andere eingefügt werden soll.
`arg`	⇔	Liste, die eingefügt werden soll und danach leer ist.

`Load` und `Save` siehe `BiLists`.

7.21.4 TextLists

`TextLists` verwalten sortierte Listen von Strings. Dieses Modul erbt alle Funktionen von `SortedBiLists`, Beschreibung siehe dort. Für die bei der Initialisierung einer solchen Liste notwendige Vergleichsprozedur werden gleich zwei Prozeduren `CmpStrings` (vergleicht zwei Strings, Groß/Kleinschreibung wird dabei beachtet[11]) und `CmpStringsCaps` (Vergleich ohne Berücksichtigung der Groß/Kleinschreibung) mitgeliefert. Man kann also einfach schreiben:

```
Init(MyList,CmpStrings);
```

Sicher ist Ihnen bei der Definition des Typs `TextNode` neben `name` der Eintrag `special` aufgefallen. Er bewirkt, daß innerhalb der Elemente, bei denen `special=TRUE` eine eigene Sortierung stattfindet. Ein Beispiel folgt später.

[11]Es gilt die Reihenfolge des ASCII-Zeichensatzes

7.21.5 CursorLists

Dies sind Listen, in denen die Elemente nicht direkt miteinander verkettet sind, sondern durch Zeiger einer anderen Liste erreicht werden. Eine Cursorliste ist eine doppelt verkettete Liste von Knoten, die jeweils einen Zeiger auf ein Element der Liste enthalten. Der Vorteil ist, daß in einer derartigen Liste beliebige Elemente verwendet werden können (also nicht nur Nachfolger eines Knotentyps) und ein Element auch in mehreren Listen erscheinen kann. Der Nachteil ist, daß man kaum vom Listenelement auf den zugehörigen Knoten schließen kann, was Listenoperationen erschwert. Die Lösung des Problems besteht darin, daß die Liste einen Cursor enthält, der auf den Knoten des aktiven Elements zeigt. Listenoperationen werden immer relativ zu diesem Cursor vorgenommen, der mit Hilfe mehrerer Prozeduren verschoben werden kann.

Exceptions

NoCursor Der Cursor zeigt auf kein Element der Liste.

NotFound Das gewünschte Element ist nicht in der Liste. Danach zeigt der Cursor nach `NIL`, also auf kein Element der Liste.

Da die Prozeduren den zuvor beschriebenen sehr ähnlich sind, wird auf eine Beschreibung der einzelnen Parameter verzichet.

`PROCEDURE Init(VAR l : List);`

Funktion: Initialisiert die Liste. Muß ausgeführt werden, bevor die Liste verwendet werden kann.

`PROCEDURE Get(VAR l : List):type;`

Funktion: Liefert das aktuelle Listenelement (nicht den Knoten), d. h. das Element, auf das der Cursor gerade zeigt.

`PROCEDURE InsertTop(VAR l : List;t : type);`

Funktion: Fügt ein Element als erstes in die Liste ein.

`PROCEDURE InsertBottom(VAR l : List;t : type);`

Funktion: Fügt ein Element als letztes in die Liste ein.

`PROCEDURE InsertBefore(VAR l : List;t : type);`

Funktion: Fügt ein Element vor dem Cursor ein.

`PROCEDURE InsertAfter(VAR l : List;t : type);`

Funktion: Fügt ein Element nach dem Cursor ein.

`PROCEDURE RemoveFirst(VAR l : List):type;`

Funktion: Entfernt das erste Element aus der Liste, gibt es aber nicht frei.

`PROCEDURE RemoveLast(VAR l : List):type;`

Funktion: Entfernt das letzte Element aus der Liste, gibt es aber nicht frei.

`PROCEDURE RemoveAct(VAR l : List):type;`

Funktion: Entfernt das aktuelle Element aus der Liste, d. h. das Element, auf das der Cursor zeigt.

`PROCEDURE Remove(VAR l : List;t : type);`

Funktion: Entfernt ein angegebenes Element aus der Liste. Diese Routinen benötigt mehr Zeit als die anderen, da es den Knoten des Elements erst suchen muß.

`PROCEDURE Remove_IF(VAR l : List;if : Check);`

Funktion: Entfernt alle Elemente aus der Liste, für die eine Bedingung erfüllt ist.

```
PROCEDURE Remove_All(VAR l : List);
```

Funktion: Entfernt alle Elemente aus der Liste.

```
PROCEDURE DeleteFirst(VAR l : List);
PROCEDURE DeleteLast(VAR l : List);
PROCEDURE Delete(VAR l : List;t : type);
PROCEDURE DeleteAct(VAR l : List);
PROCEDURE Delete_IF(VAR l : List;if : Check);
PROCEDURE Delete_All(VAR l : List);
```

Funktion: Wie die entsprechenden Remove-Prozeduren, jedoch werden hier die einzelnen Elemente durch `Dispose` freigegeben. Achtung: Kann nur verwendet werden, wenn die einzelnen Elemente nur aus Speicher bestehen, also keine Zeiger auf andere Objekte enthalten. Außerdem darf ein Element erst dann freigegeben werden, wenn es in keiner anderen Liste mehr hängt.

```
PROCEDURE Destruct_IF(VAR l   : List;
                          if  : Check;
                          des : Destructor);
PROCEDURE Destruct_All(VAR l : List;des : Destructor);
```

Funktion: Wie die entsprechenden Delete-Prozeduren, nur werden die Elemente durch eine Destructor-Prozedur vernichtet. Dabei ist auch darauf zu achten, daß die Elemente nicht mehr in einer anderen Liste enthalten sind.

```
PROCEDURE Apply(VAR l : List;proc : ApplyProc);
```

Funktion: Für jedes Element der Liste wird eine übergebene Prozedur aufgerufen.

```
PROCEDURE Sort(VAR l : List;greater : Relation);
```

Funktion: Sortiert die Liste anhand einer Ordnungsrelation.

```
PROCEDURE Append(VAR dest,arg : List);
```

Funktion: Hängt eine Liste an eine andere an. `arg` ist danach leer.


```
PROCEDURE Dup(VAR dest,arg : List);
```

Funktion: Erzeugt eine Kopie einer Liste. Dabei werden jedoch nicht die Elemente kopiert, sondern man erhält eine zweite Liste, deren Knoten auf dasselbe Elemente zeigen. Die Kopie finden Sie danach in `dest`.

```
PROCEDURE Find(VAR l : List;equal : Check);
```

Funktion: Sucht das erste Element in einer Liste, für die eine angegebene Bedingung erfüllt ist.

```
PROCEDURE FindNext(VAR l : List;equal : Check);
```

Funktion: Sucht das nächste Element (ab der Cursorposition), für das eine Bedingung erfüllt ist.

```
PROCEDURE Mark(VAR l : List);
```

Funktion: Markiert den aktuellen Knoten.

```
PROCEDURE GoMark(VAR l : List);
```

Funktion: Setzt den Cursor auf den zuletzt markierten Knoten.

```
PROCEDURE Next(VAR l : List);
```

Funktion: Bewegt den Cursor ein Element weiter.

```
PROCEDURE Prev(VAR l : List);
```

Funktion: Bewegt den Cursor ein Element zurück.

```
PROCEDURE First(VAR l : List);
```

Funktion: Setzt den Cursor auf das erste Element der Liste.

```
PROCEDURE Last(VAR l : List);
```

Funktion: Setzt den Cursor auf das letzte Element der Liste.

PROCEDURE n_th(**VAR** l : List; n : INTEGER);

Funktion: Setzt den Cursor auf das n-te Element der Liste.

PROCEDURE Go(**VAR** l : List;t : type);

Funktion: Bewegt den Cursor auf ein angegebenes Element, d. h. auf den Knoten, der auf das übergebene Element zeigt.

PROCEDURE HasCursor(**VAR** l : List):BOOLEAN;

Funktion: Liefert TRUE, wenn der Cursor auf ein Element der Liste zeigt. Bei einer Suchaktion kann es nämlich passieren, daß bei Nichtfinden der Cursor nach NIL zeigt.

PROCEDURE Save(**VAR** l : List;f : File;part : SaveProc);

Funktion: Sichert eine Liste in ein File.

PROCEDURE Load(**VAR** l : List;f : File;part : LoadProc);

Funktion: Liest eine Liste aus einem File.

PROCEDURE GarbageCollect;

Funktion: Die Knoten der Listen werden nach Benutzung nicht sofort freigegeben, sondern für andere Knoten weiterverwendet. Mit dieser Routine können die Knoten, die auf Weiterverwendung warten, freigegeben werden.

Nun noch ein Beispiel zur Verwendung einer TextListe:

```
MODULE TextListTest;
FROM InOut         IMPORT WriteGrp,ReadGrp;
FROM Lists         IMPORT TextLists;
FROM Resources     IMPORT New,Dispose;

TYPE
  ChildPtr = POINTER TO Child;

| Es folgt die Ausprägung
DEFINITION MODULE ChildrenLists = TextLists(ChildPtr);

TYPE
  Child    = RECORD OF ChildrenLists.TextNode
               age :  INTEGER;
             END;
VAR
  ChildrenList : ChildrenLists.TextList;
  TempStr      : CLASSPTR TO STRING;
  ActChild     : ChildPtr;
  done         : BOOLEAN:=FALSE;

PROCEDURE WriteChildren(child : ChildPtr);
BEGIN
  WriteString(child.name^);WriteLn;
  WriteInt(child.age);WriteLn;
  IF child.special THEN
    WriteString("Mädchen");
  ELSE
    WriteString("Junge");
  END;
  WriteLn;WriteLn;
END WriteChildren;

PROCEDURE Destruct(child : ChildPtr);
BEGIN
  Dispose(child.name);
  Dispose(child);
END Destruct;

BEGIN
  ChildrenList.Init(ChildrenLists.CmpStringsCaps);
  REPEAT
    TempStr'RANGE:=20;
    New(TempStr);
    WriteString("Namen: ");
```

```
    ReadString(TempStr^);
    IF TempStr^.len#0 THEN
      New(ActChild);
      ActChild.name:=TempStr;
      WriteString("Alter: ");
      ReadInt(ActChild.age);
      WriteString("Mädchen ? ");
      ActChild.special:=ReadBool();
      ChildrenList.Insert(ActChild);
    ELSE
      done:=TRUE;
      Dispose(TempStr);
    END
  UNTIL done;
  WriteLn;
  ChildrenList.Apply(WriteChildren);

| Die nachfolgende Destructanweisung könnte auch entfallen,
| da das Laufzeitsystem am Programmende allen Speicher wieder
| freigibt. Wie es geht soll aber trotzdem hier gezeigt werden,
| angenommen das Programm ginge hier noch weiter.
| Delete_All hätte man nicht verwenden können, da sonst bis zum
| Programmende der Speicher, der von den Namen belegt ist,
| verloren wäre.

  ChildrenList.Destruct_All(Destruct);
END TextListTest.
```


7.22 LongSets

```
DEFINITION MODULE LongSets;

FROM System IMPORT LONGSET;

TYPE
  LongSet = ARRAY OF LONGSET;
  LSetPtr = CLASSPTR TO LongSet;

PROCEDURE CreateSet(VAR set : LSetPtr;len : INTEGER);


$$OwnHeap:=TRUE
PROCEDURE Unite(REF s1,s2 : LongSet):LongSet;

$$OwnHeap:=TRUE
PROCEDURE Intersect(REF s1,s2 : LongSet):LongSet;


PROCEDURE isIn(i : INTEGER;REF s : LongSet):BOOLEAN;

PROCEDURE isEmpty(REF s : LongSet):BOOLEAN;

PROCEDURE isPartOf(REF s1,s2 : LongSet):BOOLEAN;

PROCEDURE Include(VAR s : LongSet;i : INTEGER);

PROCEDURE Exclude(VAR s : LongSet;i : INTEGER);


$$OwnHeap:=TRUE
PROCEDURE LConst(REF a : ARRAY OF INTEGER):LongSet;

$$OwnHeap:=TRUE
PROCEDURE Empty():LongSet;


END LongSets.
```

Normale Sets können bekanntlich maximal 32 Elemente enthalten. Dieses Modul dient nun dazu, fast beliebig lange Sets von `INTEGER`-Zahlen zu verwalten. Ein solches Set kann man entweder als Variable mit fester Länge oder dynamisch erzeugen. Auf die einzelnen Elemente kann man jedoch nicht direkt zugreifen, sondern muß die in diesem Modul enthaltenen Prozeduren verwenden.

`PROCEDURE CreateSet(VAR set : LSetPtr;len : INTEGER);`

Funktion: Erzeugt ein Set.

Parameter:

`set`	⇒	Zeiger auf das neue Set.
`len`	⇐	Länge des Sets.

```
$$OwnHeap:=TRUE
PROCEDURE Unite(VAR s1,s2 : LongSet):LongSet;
```

Funktion: Bildet die Vereinigungsmenge zweier Mengen.

Parameter:

`s1,s2`	⇐	Mengen, die vereinigt werden sollen.
	⇒	Vereinigungsmenge.

```
$$OwnHeap:=TRUE
PROCEDURE Intersect(VAR s1,s2 : LongSet):LongSet;
```

Funktion: Bildet die Schnittmenge zweier Mengen.

Parameter:

`s1,s2`	⇐	Mengen, die geschnitten werden sollen.
	⇒	Schnittmenge.

`PROCEDURE isIn(i : INTEGER;VAR s : LongSet):BOOLEAN;`

Funktion: Prüft, ob ein Element in der Menge enthalten ist.

Parameter:

`i`	⇐	Nummer des Elements, das geprüft werden soll.
`s`	⇐	Set, das überprüft werden soll.

`PROCEDURE isEmpty(REF s : LongSet):BOOLEAN;`

Funktion: Prüft, ob eine Menge leer ist.

Parameter:

`s`	⇐	Menge, due gerpüft werden soll.
	⇒	`TRUE` wenn sie leer ist.

`PROCEDURE isPartOf(REF s1,s2 : LongSet):BOOLEAN;`

Funktion: Prüft, ob eine Menge Teilmenge einer anderen ist.

Parameter:

tt s1,s2	⇐	Prüft, ob `s1` Teilmenge von `s2` ist.
	⇒	`TRUE`, wenn `s1` Teilmenge ist.

`PROCEDURE Include(VAR s : LongSet;i : INTEGER);`

Funktion: Fügt ein Element in eine Menge ein.

Parameter:

`s`	⇔	Set, in das das Element eingefügt werden soll.
`i`	⇐	Nummer des Elements, das eingefügt werden soll.

```
PROCEDURE Exclude(VAR s : LongSet;i : INTEGER);
```

Funktion: Entfernt ein Element aus einer Menge.

Parameter:

`s`	⇔	Set, aus dem das Element entfernt werden soll.
`i`	⇐	Nummer des Elements, das entfernt werden soll.

```
$$OwnHeap:=TRUE
PROCEDURE LConst(REF a : ARRAY OF INTEGER):LongSet;
```

Funktion: Erzeugt eine Menge, in der alle Mengenelemente enthalten sind, deren Nummer in einem der Arrayelemente enthalten ist, welches übergeben wird.

Parameter:

`a`	⇐	Feld mit den Elementnummern, welche gesetzt werden sollen.
	⇒	Menge mit diesen Elementen.

```
$$OwnHeap:=TRUE
PROCEDURE Empty():LongSet;
```

Funktion: Liefert eine leere Menge.

Parameter:

	⇒	Leere Menge.

7.23 MStr

Dieses Modul stellt die Funktionen zur Bearbeitung von Modula 2-Strings zur Verfügung. Man sollte es nur im Modula 2 Modus verwenden. Eine nähere Beschreibung der Prozeduren kann hier entfallen, da sie identisch mit denen des Moduls `Str` sind.

```
DEFINITION MODULE MStr;
(* $A- *)
FROM System IMPORT Regs;

TYPE
  Equation = (smaller,equal,greater);
  CapsArr  = ARRAY CHAR OF CHAR;

CONST
  CAP      = CapsArr;

VAR
  done : BOOLEAN;


PROCEDURE CapsString(VAR Str IN A0 : ARRAY OF CHAR);


PROCEDURE Equal(REF Str1 IN A0,
                    Str2 IN A1 : ARRAY OF CHAR): BOOLEAN;



PROCEDURE CapsEqual(REF Str1 IN A0,
                        Str2 IN A1 : ARRAY OF CHAR):BOOLEAN;


PROCEDURE Greater(REF Str1 IN A0,
                      Str2 IN A1 : ARRAY OF CHAR):BOOLEAN;


PROCEDURE Compare(REF Str1 IN A0,
                      Str2 IN A1 : ARRAY OF CHAR):Equation;
```

```
PROCEDURE Seg(REF Str     IN A0 : ARRAY OF CHAR;
              pos     IN D2,
              len     IN D3 : INTEGER;
          VAR Segment IN A1 : ARRAY OF CHAR);

PROCEDURE First(REF Str IN A0 : ARRAY OF CHAR;
                    Ch  IN D2 : CHAR):INTEGER;

PROCEDURE Last(REF Str IN A0 : ARRAY OF CHAR;
                   Ch  IN D2 : CHAR):INTEGER;

PROCEDURE Search(REF Find,In:ARRAY OF CHAR):INTEGER;

PROCEDURE Insert(VAR Dest   : ARRAY OF CHAR;
                 REF Source : ARRAY OF CHAR;
                     pos    : INTEGER);

PROCEDURE Replace(REF Str   : ARRAY OF CHAR;
                  VAR Into  : ARRAY OF CHAR;
                      pos   : INTEGER);

PROCEDURE Delete(VAR Str IN A0 : ARRAY OF CHAR;
                     pos IN D2,
                     len IN D3 : INTEGER);

PROCEDURE Concat(VAR Dest   IN A0 : ARRAY OF CHAR;
                 REF Source IN A1 : ARRAY OF CHAR);

PROCEDURE Length(REF Str IN A0 : ARRAY OF CHAR):INTEGER;

END MStr.
```


7.24 PatternMatcher

Mit diesem Modul können Sie prüfen, ob ein String einem Dos-Pattern[12] entspricht. Es verarbeitet in etwa die Namensmuster von OS 1.3, im einzelnen „`#?`", „`#`", „`?`", „`(|)`" und „`~`". Für höhere Ambitionen benutzen Sie bitte die erweiterten Pattern von `Dos` unter OS 2.0. Zahlreiche Anwendungsbeispiele für `PatternMatcher` finden Sie auch im Implementationsteil des Moduls `DosSupport`.

```
DEFINITION MODULE PatternMatcher;

EXCEPTION
  BadPattern : "Bad pattern";

TYPE
  Pattern      = HIDDEN;

PROCEDURE FreePattern(VAR pat : Pattern);

PROCEDURE CreatePattern(REF str : STRING;
                        VAR pat : Pattern);

PROCEDURE Matches(REF s   : STRING;
                      pat : Pattern):BOOLEAN;

END PatternMatcher.
```

`PROCEDURE FreePattern(VAR pat : Pattern);`

Funktion: Gibt ein Pattern wieder frei.

Parameter:

`pat` ⇔ Pattern, das freigegeben werden soll. Es darf danach nicht mehr verwendet werden.

[12] Allen, die mit diesem Begriff nichts anfangen können, können nähere Informationen dem Benutzerhandbuch entnehmen.

```
PROCEDURE CreatePattern(REF str : STRING;
                         VAR pat : Pattern);
```

Funktion: Erzeugt aus einem Patternstring ein Pattern. Dies beschleunigt später die Vergleiche enorm, da nicht vor jedem Vergleich der Patternstring geparst werden muß. Wird es nicht mehr gebraucht, sollte man es mittels `FreePattern` wieder freigeben.

Parameter:

`str`	⇐	String, der das Pattern enthält.
`pat`	⇒	Erzeugtes Pattern. Dieses kann nun zu Vergleichen verwendet werden.

```
PROCEDURE Matches(REF s   : STRING;
                      pat : Pattern):BOOLEAN;
```

Funktion: Prüft, ob ein Clusterstring einem Pattern entspricht.

Parameter:

`s`	⇐	String, der geprüft werden soll.
`pat`	⇐	Pattern, mit dem verglichen werden soll.
	⇒	`TRUE`, wenn der Sring dem Pattern entspricht.

7.25 Profiler

Mittels diesem Modul kann zu Optimierungszwecken festgestellt werden, wieviel Zeit zwischen zwei Punkten im Programm vergeht. Dabei wird, in Mikrosekunden gemessen, die maximale, die minimale Zeit und die Durchschnittszeit festgehalten. Außerdem wird die Zahl der Durchläufe, für rekursive Prozeduren die Schachtelungstiefe, sowie die momentane Stackposition ausgegeben.

```
DEFINITION MODULE Profiler;

TYPE
  Profile = HIDDEN;


PROCEDURE CreateProfile(REF Name : STRING):Profile;

PROCEDURE DestructProfile(p : Profile);


PROCEDURE PStart(p : Profile);

PROCEDURE PEnd(p : Profile);


PROCEDURE WriteProfile(p : Profile);

PROCEDURE WriteProfiles;


PROCEDURE ResetProfiler;

PROCEDURE ClearProfiles;

GROUP
  All = CreateProfile,DestructProfile,PStart,PEnd,WriteProfile,
        WriteProfiles,ResetProfiler,ClearProfiles;

END Profiler.
```

Dabei wird die Zeit immer in bestimmten Meßbereichen gemessen. Bevor man einen solchen Meßbereich verwenden kann, muß man diesen definieren:

PROCEDURE `CreateProfile(`**REF** `Name : STRING):Profile;`

Funktion: Erzeugt eine Meßbereichsmarke.

Parameter:

`Name`	⇐	Name des Meßbereichs, damit man ihn bei der Ausgabe wiedererkennt.
	⇒	Meßbereichsmarke.

PROCEDURE `DestructProfile(p : Profile);`

Funktion: Gibt eine Meßbereichsmarke wieder frei.

Parameter:

`p`	⇔	Marke, die freigeben werden soll. Sie darf danach nicht mehr verwendet werden.

PROCEDURE `PStart(p : Profile);`

Funktion: Setzt den Anfang eines Meßbereiches.

Parameter:

`p`	⇐	Marke, der dieser Bereich zugeordnet werden soll.

PROCEDURE `PEnd(p : Profile);`

Funktion: Setzt das Ende eines Meßbereiches.

Parameter:

`p`	⇐	Marke des Bereichs, der hier beendet werden soll.

PROCEDURE `WriteProfile(p : Profile);`

Funktion: Gibt die Daten eines Meßbereiches auf die Standard-Ausgabe aus.

Parameter:

`p`	⇐	Marke des Bereiches, dessen Daten ausgegeben werden sollen.

PROCEDURE `WriteProfiles;`

Funktion: Gibt die Daten aller Meßbereiche aus.

PROCEDURE `ResetProfiler;`

Funktion: Setzt die Datenfelder aller Meßbereiche auf Null zurück.

PROCEDURE `ClearProfiles;`

Funktion: Gibt alle Meßmarken wieder frei. Danach darf nicht mehr darauf zugegriffen werden, es sei denn, man definiert neue Marken.

7.26 Random

In diesem Modul finden Sie Prozeduren zur Erzeugung verschiedener Zufallswerte. Die Initialisierung geschieht auf Basis verschiedener interner Registerwerte, während nachfolgende Zufallszahlen algorithmisch erzeugt werden.

```
DEFINITION MODULE Random;

PROCEDURE Randomize;

PROCEDURE RealRND():LONGREAL;

PROCEDURE RND(Max IN 2 : LONGCARD):LONGCARD;

PROCEDURE BoolRND():BOOLEAN;

GROUP
  All   = Randomize,RealRND,RND,BoolRND;

END Random.
```

PROCEDURE `Randomize;`

Funktion: Mischt den Zufallsgenerator neu, wie es auch beim Programmstart geschieht.

PROCEDURE `RealRND():LONGREAL;`

Funktion: Gibt eine zufällige `REAL`-Zahl x zurück, wobei $0 \leq x < 1$ gilt.

Parameter:

⇒ Zufällige `REAL`-Zahl.

PROCEDURE `RND(Max IN 2 : LONGCARD):LONGCARD;`

Funktion: Gibt eine zufällige `LONGCARD`-Zahl n zurück, wobei $0 \leq n <$ `Max` gilt.

Parameter:

`Max` ⇐ Maximal mögliche Zufallszahl.

⇒ Zufallszahl von Null bis `Max-1`.

PROCEDURE `BoolRND():BOOLEAN;`

Funktion: Liefert einen zufälligen Boolwert.

Parameter:

⇒ Zufälliger Boolwert.

7.27 Resources

```
DEFINITION MODULE Resources;

TYPE
  ClassPtr    = HIDDEN;
  ContextPtr  = HIDDEN;

|============================================================
| Achtung !!!!!
| Diese Prozeduren dürfen nur vom Compiler benutzt werden, sie
| müssen an dieser Stelle im Quelltext stehen.

  PROCEDURE NewContext;

  PROCEDURE OldContext;

  PROCEDURE AllocObj(class : ClassPtr;
                     context : ContextPtr):ANYPTR;

  PROCEDURE FreeObj(obj : ANYPTR);

  PROCEDURE CloneObj(obj : ANYPTR;context : ContextPtr):ANYPTR;

  PROCEDURE ChangeObjContext(obj : ANYPTR;context : ContextPtr);

|============================================================
FROM System IMPORT PROC;

FROM Exec IMPORT MemReqs, MemReqSet;

CONST
  NoContext = NIL;|CAST(ContextPtr,NIL);

DEFINITION MODULE ResHandles(ResHandlePtr : POINTER TO ResHandle);

    TYPE
      ResourcePtr = HIDDEN;
      ResHandle   = RECORD END;
      Destructor  = PROCEDURE(res : ResHandlePtr);
```



```
    PROCEDURE AddResource(res      : ResHandlePtr;
                          destruct : Destructor;
                          context  : ContextPtr:=NIL);

    PROCEDURE RemResource(res : ResHandlePtr);

    PROCEDURE FreeResource(VAR res : ResHandlePtr);

    PROCEDURE IsResource(res : ResHandlePtr):BOOLEAN;

    (* PRIVAT !!! *)
    PROCEDURE FreeResources(res : ResourcePtr);


  END ResHandles;

EXCEPTION
  NotEnoughMemory      : "Not enough memory";
  MemoryNotAllocated   : "Memory not allocated";
  NILDisposed          : "Disposed a NIL-Pointer";
  ResourceFreedTwice   : "Resource freed twice";
  ContexFreedTwice     : "Context freed twice";
  NoOldContext         : "No old context accessable";
  ActContextFreed      : "Actcontext freed";
  ResourceNotAllocated : "Resource not allocated";
  NilResource          : "Passed ResourcePtr is NIL";

VAR
  ActContext    : ContextPtr;

PROCEDURE Allocate(VAR p       : ANYPTR;
                       size    : LONGINT;
                       clear,
                       chip    : BOOLEAN := FALSE;
                       mem     : MemReqSet := MemReqSet:{};
                       context : ContextPtr := NoContext );

PROCEDURE New( VAR p       : ANYPTR;
                   clear,
                   chip    : BOOLEAN := FALSE;
                   mem     : MemReqSet := MemReqSet:{};
                   context : ContextPtr:=NoContext );

PROCEDURE Dispose( VAR p : ANYPTR );
```

```
PROCEDURE DisposeAll( context : ContextPtr := NoContext );

PROCEDURE Avail( max,
                 chip : BOOLEAN := FALSE;
                 mem  : MemReqSet := MemReqSet:{} ): LONGINT;

PROCEDURE
   Create_Context(father : ContextPtr:=NoContext):ContextPtr;

PROCEDURE Create_Root():ContextPtr;

PROCEDURE Delete_Context(VAR context : ContextPtr);

END Resources.
```

Diese Modul bietet Prozeduren zur Resourcenverwaltung, insbesondere von Speicher. Es gliedert sich dabei in drei Teile:

- `ResHandles` : Ein generisches Modul zur Verwaltung beliebiger Resourcen[13]
- Routinen zur Allozierung und Freigabe von Speicher.
- Routinen zur Erzeugung und Freigabe von Kontexten.

[13] Unter einer Resource versteht man jede Art von Betriebsmittel, daß ein Programm vom Betriebsystem anfordern kann, z. B. Speicher, Files, Fenster, Screens, etc.

7.27.1 Resourcenverwaltung

Zur Verwaltung von Resourcen existiert das generische Modul `ResHandles`. Unter Verwaltung versteht man hierbei, daß man die Resource einem Kontext[14] zuordnen kann, und diese dann mit dem Kontext freigeben, bzw. daß die Resource am Programmende automatisch wieder freigegeben wird; man sagt auch, die Resource wird getrackt, oder man spricht von Resourcetracking. Voraussetzung, daß man eine Resource mit diesem Modul verwenden kann, ist daß die Resource ein Nachfolger des generischen Typen `ResHandle` ist. Die meisten wichtigen Strukturen des Betriebssystems sind als Nachfolger von `ResHandle` definiert. Da dieser Typ selbst keine Elemete enthält, verschiebt er auch nicht die Positionen der einzelnen Elemente der Nachfolgetypen. Er dient lediglich dazu, daß die Typsicherheit bei den verschiedenartigen Elementen, die eingehäng werden, gewähleistet ist.
Es stehen folgende Prozeduren zur Verfügung:

```
TYPE
  Destructor  = PROCEDURE(res : ResHandlePtr);
PROCEDURE AddResource(res      : ResHandlePtr;
                      destruct : Destructor;
                      context  : ContextPtr:=NIL);
```

Funktion: Fügt einen Nachfolger von `ResHandle` als Resource in einen Kontext ein.

[14] Das Kontextsystem wird weiter unten noch genauer erklärt.

Parameter:

`res`	⇐	Zeiger auf die Resource, die eingehängt werden soll.
`destruct`	⇐	Da es sich bei der Resource um die verschiedensten Dinge handeln kann, kann das Modul nicht wissen, wie diese freizugeben ist. Daher muß hier eine Freigabeprozedur übergeben werden, die den entsprechenden Typen übergeben bekommt, und die Resource freigibt.
`context`	⇐	Kontext, dem diese Resource zugeordnet werden soll. Wird dieser Parameter nicht, oder hier `NIL` übergeben, wird die Resource in den aktuellen Kontext (s. u.) eingehängt.

`PROCEDURE RemResource(res : ResHandlePtr);`

Funktion: Entfernt eine Resource wieder aus dem Kontext, in den sie eingefügt worden ist. Sie wird dann auch nicht mehr automatisch am Schluß freigegeben.

Parameter:

`res`	⇐	Zeiger auf die Resource, die entfernt werden soll.

`PROCEDURE FreeResource(VAR res : ResHandlePtr);`

Funktion: Gibt eine Resource frei, indem sie die bei `AddResource` übergebene Freigabeprozedur aufruft.

Parameter:

`res`	⇔	Zeiger auf die Resource, die freigegeben werden soll.

`PROCEDURE IsResource(res : ResHandlePtr):BOOLEAN;`

Funktion: Prüft, ob eine Resource bereits getrackt wird.

Parameter:

`res`	⇐	Zeiger auf die Resource, die geprüft werden soll.
	⇒	`TRUE` wenn die Resource bereits in einem Kontext hängt.

Exceptions:

NotEnoughMemory Es wurde versucht mehr Speicher anzufordern, als freier vorhanden war.

MemoryNotAllocated Es wurde versucht Speicher mit `Dispose` freizugeben, der nicht mit `Resources` alloziert wurde.

NILDisposed Es wurde versucht einen `NIL`-Pointer, d. h. ein nicht vorhandenes Speicherstück, freizugeben.

ResourceFreedTwice Es wurde versucht eine Resource freizugeben oder zu entfernen, die schon freigegeben wurde.

ContexFreedTwice Es wurde versucht einen Kontext freizugeben, der schon freigegeben wurde.

NoOldContext Nach einem **`TRACK...END`** kann der Vorgängerkontext nicht gefunden werden. Dies kann nur dadurch verursacht werden, daß jemand von Hand an den Kontexten gespielt hat, was man nicht tun sollte, oder daß irgendjemand über den eigenen Speicherbereich hinausgeschrieben und dabei die Kontextstruktur vernichtet hat.

ActContextFreed Es wurde versucht den aktuellen Kontext freizugeben. Dies ist untersagt, und darf nur vom Laufzeitsystem vorgenommen werden.

ResourceNotAllocated Es wurde versucht eine Resource freizugeben oder zu entfernen, die nicht mit `AddResource` in einem Kontext eingehängt wurde.

NilResource `RemResource` oder `FreeResource` wurde ein Zeiger übergeben, der auf `NIL` zeigte.

Ein Beispiel für die Verwendung von `ResHandles` finden Sie im Modul `Intuition` und `T_Intuition`, in denen unter anderen der Typ Window getrackt wird – wie jeder weiß, ein Fenster muß spätestens am Programmende wieder geschlossen werden. Sehen wir uns zunächst einmal die Definition des Typs `WindowPtr` aus `Intuition` an:

```
TYPE
  ..
  WindowPtr             = POINTER TO Window;

DEFINITION MODULE WinRes = Resources.ResHandles(WindowPtr);


TYPE
  Window           = RECORD  OF WinRes.ResHandle
       nextWindow  : WindowPtr;
       leftEdge    : INTEGER;
       topEdge     : INTEGER;
       width       : INTEGER;
       height      : INTEGER;
       mouseY      : INTEGER;
       mouseX      : INTEGER;
       minWidth    : INTEGER;
                       ...
                     END
```

Sie sehen, `Window` ist ein Nachfolger von `ResHandle`. Nun ein Ausschnitt aus der Definition der getrackten Öffnungs- und Schließprozeduren aus dem Modul `T_Intuition`:

```
| Dies ist die Freigabeprozedur, die bei
| AddResource mit übergeben wird.
```



```
|
PROCEDURE KillWindow(w : WindowPtr);
BEGIN
  In.CloseWindow(w);
END KillWindow;


PROCEDURE OpenWindow( REF newWindow : NewWindow;
                          context   : ContextPtr ): WindowPtr;
VAR
  w  : WindowPtr;
BEGIN
  w:=In.OpenWindow( newWindow ); | Fenster öffnen
  ASSERT( w#NIL, WindowNotOpen );| Test obs geklappt hat

| Einhängen in den Kontext:
  w.AddResource( KillWindow , context );
  RETURN w
END OpenWindow;

PROCEDURE CloseWindow(VAR window : WindowPtr);
BEGIN
  window.FreeResource;
  | Hierbei wird dann KillWindow aufgerufen
END CloseWindow;
```

Nun sollte die Verwendung des Moduls `ResHandles` um einiges klarer sein. Selbstverständlich können Sie damit auch beliebige eigene Kontruktionen in Kontexte einhängen, sofern es dafür noch kein Modul gibt.

7.27.2 Speicherverwaltung

Im Prinzip könnte man den Speicher mit `AllocMem` aus `Exec` allozieren, und dann mit `AddResource` in das ResourceSystem eingliedern. Da Speicher jedoch generell mit `FreeMem` freigegeben wird, wäre dies ein unnötiger Aufwand. Daher bietet `Resources` eigene Prozeduren zur Speicherverwaltung an, die sich um die Kontexte kümmern, und am Programmende allen allozierten Speicher wieder freigeben:

```
PROCEDURE Allocate(VAR p       : ANYPTR;
                       size    : LONGINT;
                       clear,
                       chip    : BOOLEAN := FALSE;
                       mem     : MemReqSet := MemReqSet:{};
                       context : ContextPtr := NoContext );
```

Funktion: Alloziert ein Speicherstück bestimmter Länge. Die Parameter `chip` und `clear` existieren lediglich noch aus Kompatibilitätsgründen, wenn in `mem` etwas anderes als „{}“ übergeben wird, werden diese Flags nicht mehr beachtet.

Parameter:

`p` ⇒ Zeiger, der nach dem Aufruf auf das allozierte Speicherstück zeigt.

`size` ⇐ Gewünschte Länge.

`clear` ⇐ Falls `TRUE`, wird das Speicherstück erst gelöscht. Sollte man bei der allozierung von Speicher für Betriebssystemstrukturen verwenden.

`chip` ⇐ Gibt an, ob das Speicherstück im Chipmem liegen soll.

`mem` ⇐ Hier können die in Exec definierten Memoryflags angegeben werden:

- **public** Speicher, der mit diesem Flag alloziert wird, darf nicht ausgelagert werden. Man sollte daher nur Strukturen damit allozieren, die immer erreichbar sein müssen wie z. B. Interupts, MessagePorts etc.. Alle anderen Strukturen sollte man nicht damit anforden.
- **chip** Das Speicherstück wird im Chipmem alloziert.
- **fast** Das Speicherstück muß im Fastmem liegen, sollte man eigentlich nur in Ausnahmefällen verwenden, da nicht alle Amigas Fastmem haben, und nach Möglichkeit immer zuerst versucht wird Fastmem zu allozieren, falls nicht ausdrücklich Chipmem gewünscht wird.
- **local** Ein solches Speicherstück ist auch noch nach dem Reset vorhanden. Erst ab Kickstart 2.0 verwendbar.
- **dma24** Speicher wird innerhalb des Zorro-II Adressbereichs alloziert, wichtig wenn der Speicher von einem Zorro-II-DMA-Gerät erreicht werden soll, und zusätzlich Speicher außerhalb des 24-Bit Adressraums existiert. Ebenfalls erst ab Kickstart 2.0.

clear Das Speicherstück wird gelöscht.

largest Die Längenangabe wird nicht beachtet, sondern das größte, freie Stück Speicher belegt. Sollte man nur machen, wenn es unbedingt nötig ist, um anderen Programmen nicht unötigerweise Speicher vorzuenthalten. Setzt man dieses Flag bei `Avail` zusammen mit einem anderen Flag, wird die Größe des größten Speicherstücks diesen Typs ermittelt.

reverse Darf unter Kickstart 2.0 noch nicht verwendet werde, da hier noch ein Fehler vorliegt, der die Maschine zum abstürzen bringt.

total Kann nicht mit `Allocate` verwendet werden. Bei der Verwendung mit `Avail` erhält man die Größe des gesamten freien Speichers des Systems.

Wird eines dieser Flags übergeben, werden die die Parameter `clear` und `chip` nicht mehr beachtet.

`context` ⇐ Kontext, zu dem der Speicher alloziert werden soll. Wird dieser Parameter nicht alloziert, was in den meisten Fällen der Fall sein wird, wird zum aktuellen Kontext alloziert.


```
PROCEDURE New( VAR p       : ANYPTR;
                   clear,
                   chip    : BOOLEAN := FALSE;
                   mem     : MemReqSet := MemReqSet:{};
                   context : ContextPtr:=NoContext );
```

Funktion: Alloziert ebenfalls wie `Allocate` Speicher, jedoch muß man hier keine Länge übergeben, es wird immer soviel Speicher belegt, wie der Typ, auf den der Zeiger zeigt groß ist. Will man also genau soviel Speicher belegen, wie eine Struktur belegt, dann verwendet man am besten `New` mit einem getypten Pointer (also keinem `ANYPTR`). Die Parameter entsprechen im übrigen denen von `Allocate`.

```
PROCEDURE Dispose( VAR p : ANYPTR );
```

Funktion: Gibt ein einzelnen Speicherstück wieder frei. **Achtung:** Geben Sie mit `Dispose` nur Speicherstücke frei, die mit `Allocate` oder `New` alloziert wurden, niemals solche, die direkt mit `AllocMem` aus `Exec` angefordert wurden.

Parameter:

`p` ⇔ Zeiger auf das Speicherstück, das freigegeben werden soll. Der Zeiger zeigt danach auf `NIL`.

```
PROCEDURE DisposeAll( context : ContextPtr := NoContext );
```

Funktion: Gibt allen Speicher eines Kontextes frei, wird kein Kontext übergeben, wird der Speicher des aktuellen Kontextes freigegeben.

Parameter:

`context` ⇐ Kontext, dessen Speicher freigegeben werden soll.

```
PROCEDURE Avail( max,
                 chip : BOOLEAN := FALSE;
                 mem  : MemReqSet := MemReqSet:{} ): LONGINT;
```

Funktion: Liefert die Größe des freien Speichers.

Parameter:

`max`	⇐	Falls `TRUE`, wird die Größe des größten freien Blocks zurückgegeben.
`chip`	⇐	Falls `TRUE`, bezieht sich der Rückgabewert nur auf Chipmemgröße.
`mem`	⇐	Auch hier können die `Exec` Memoryflags angegeben werden, um die Größe einer bestimmten Speicherart zu ermitteln. Beschreibung der Flags siehe `Allocate`. Wird hier ein Flag übergeben, werden die beiden vorhergehenden Parameter ignoriert.
	⇒	Größe des freien Speichers.

7.27.3 Kontext-Handling

Normalerweise werden alle Allozierungen relativ zum aktuellen Kontext gemacht. D. h., alle Resourcen werden erst am Ende eines **TRACK...END** oder am Programende wieder freigegeben. Oft benötigt man jedoch Resourcen nur über einen bestimmten Bereich, der aber nicht mit einem **TRACK...END** abgegrenzt werden kann, oder man möchte innerhalb eines **TRACK...END** eine Resource allozieren, und diese soll nicht am Ende von **TRACK** wieder freigegeben werden. In einem solchen Fall kann man bei allen Allozierungsprozeduren einen optionalen Kontext übergeben, zu dem dann die Resource alloziert wird. Ein weiteres Beispiel wäre eine Liste, die in ihrem Listenkopf einen Kontext eingetragen hat, und deren Elemente zu diesem alloziert werden. Durch Freigabe dieses Kontextes kann man alle Elemente der Liste auf einmal freigeben, ohne die Liste durchlaufen zu müssen.

Bevor man einen Kontext benutzen kann, muß er erst erzeugt werden. Dabei wird ein Kontext immer relativ zu einem Vaterkontext erzeugt, mit der Folge, daß bei der Freigabe des Vaterkontextes auch alle Söhne freigegeben werden.


```
PROCEDURE
  Create_Context(father : ContextPtr:=NoContext):ContextPtr;
```

Funktion: Erzeugt einen neuen Kontext.

Parameter:

`father`	⇐	Vaterkontext, wird dieser nicht angegeben, wird der aktuelle Kontext zum Vaterkontext erklärt.
	⇒	Zeiger auf den neuen Kontext.

```
PROCEDURE Create_Root():ContextPtr;
```

Funktion: Erzeugt einen neuen Kontext, der selbst keinen Vater hat. Dieser kann wie jeder andere Kontext verwendet werden. Wichtig ist jedoch, da dieser Kontext nicht als Sohn im aktuellen Kontext hängt, wird er nicht automatisch freigegeben. Man muß ihn daher am Programmende selbst freigeben.

Parameter:

⇒ Zeiger auf den neuen Kontext.

```
PROCEDURE Delete_Context(VAR context : ContextPtr);
```

Funktion: Gibt einen Kontext wieder frei. Dabei werden alle Speicherstücke und Resourcen, die zu diesem und seinen Söhnen alloziert wurden, wieder freigegeben.

Parameter:

`context`	⇔	Kontext, der freigegeben werden soll.

7.27.4 „Getrackte“ System-Module

Um einen sicheren Umgang mit Systemresourcen zu gewährleisten, existieren zu den drei wichtigsten Systemmodulen: `Exec`, `Dos` und `Intuition`

Module, bei denen die Freigabe der Resourcen automatisch geschieht, und bei denen Fehler durch Exceptions angezeigt werden. Wichtig jedoch ist, man darf niemals die Allozierungs- und Freigabeprozeduren von den ungetrackten und den getrackten Modulen mischen, ein Systemabsturz wäre die Folge. Auf die Funktionen der Prozeduren dieser Module wird hier nicht eingegangen, sie entsprechen denen der normalen Libraryfunktionen, die in einem eigene Kapitel beschrieben werden.

7.27.4.1 T_Dos

```
DEFINITION MODULE T_Dos;

FROM Dos   IMPORT actionEnd,actionNotKnown,badStreamName,beginning,
                  BPTR,BSTR,CommandLineInterface,CommandLineInterfacePtr,
                  commentTooBig,copyDir,createDir,CreateProc,
                  ctrlC,ctrlD,ctrlE,ctrlF,current,currentVolume,Date,DatePtr,
                  DateStamp,Delay,DeleteFile,deleteObject,
                  deleteProtected,DeviceList,DeviceListPtr,DeviceListType,
                  DeviceNode,DeviceNodePtr,deviceNotMounted,DeviceProc,
                  die,dirNotFound,diskChange,diskFull,
                  diskInfo,diskNotValidated,diskType,diskWriteProtected,
                  DosBase,dosDisk,DosEnvec,DosEnvecPtr,
                  DosInfo,DosInfoPtr,DosLibrary,DosLibraryPtr,
                  DosPacket,DosPacketPtr,end,error,
                  event,Examine,examineNext,examineObject,
                  LockMode,Execute,Exit,ExNext,
                  fail,FileHandle,FileHandlePtr,FileHandlePtr,
                  FileInfoBlock,FileInfoBlockPtr,FileLock,FileLockPtr,
                  fileNotObject,FileSysStartupMsg,FileSysStartupMsgPtr,
                  findInput,findOutput,findUpDate,flush,freelock,
                  getBlock,GetPacket,Info,info,
                  InfoData,InfoDataPtr,inhibit,Input,
                  invalidComponentName,invalidLock,invalidResidentLibrary,
                  IoErr,IsInteractive,kickstartDisk,lineTooLong,LoadSeg,
                  locateObject,moreCache,newFile,nil,
                  noDefaultDir,noDisk,noDiskPresent,noFreeStore,
                  noMoreEntries,notADosDisk,notReallyDos,objectExists,
                  objectInUse,objectNotFound,objectTooLarge,objectWrongType,
                  ok,(*oldFile,*)Output,parent,
                  PathInfo,PathInfoPtr,Process,Process,
                  ProcessId,ProcessPtr,ProtectionFlags,ProtectionFlagSet,
                  QueuePacket,read,readOnly,readProtected,
                  readReturn,readWrite,Rename,renameAcrossDevices,
                  renameDisk,renameObject,ResidentSegment,ResidentSegmentPtr,
                  RootNode,RootNodePtr,screenMode,Seek,
                  seek,seekError,setComment,SetComment,
                  setDate,setMap,setProtect,SetProtection,
                  StandardPacket,StandardPacketPtr,TaskArray,
```



```
                  TaskArrayPtr,taskTableFull,ticksPerSecond,timer,
                  tooManyLevels,truncate,TypeGrp,UnLoadSeg,
                  unreadableDisk,validated,validating,waitChar,
                  WaitForChar,warn,write,writeLock,
                  writeProtect,writeProtected,writeReturn;


FROM Resources  IMPORT ContextPtr;


EXCEPTION
  WriteError              : "WriteError";
  ReadError               : "ReadError";
  EOF                     : "End of File";
  NoParentDir             : "No parent-directory accessable";
  NoFreeStore             : 103;
  TaskTableFull           : 105;
  LineTooLong             : 120;
  FileNotObject           : 121;
  InvalidResidentLibrary  : 122;
  NoDefaultDir            : 201;
  ObjectInUse             : 202;
  ObjectExists            : 203;
  DirNotFound             : 204;
  ObjectNotFound          : 205;
  BadStreamName           : 206;
  ObjectTooLarge          : 207;
  ActionNotKnown          : 209;
  InvalidComponentName    : 210;
  InvalidLock             : 211;
  ObjectWrongType         : 212;
  DiskNotValidated        : 213;
  DiskWriteProtected      : 214;
  RenameAcrossDevices     : 215;
  DirectoryNotEmpty       : 216;
  TooManyLevels           : 217;
  DeviceNotMounted        : 218;
  SeekError               : 219;
  CommentTooBig           : 220;
  DiskFull                : 221;
  DeleteProtected         : 222;
  WriteProtected          : 223;
  ReadProtected           : 224;
  NotADosDisk             : 225;
  NoDisk                  : 226;
  NoMoreEntries           : 232;


  Protected     = DiskWriteProtected,DeleteProtected,ReadProtected,
                  WriteProtected;

  OpenErr       = ObjectNotFound,ObjectInUse,DirNotFound,NotADosDisk,NoDisk,
                  DiskNotValidated,DeviceNotMounted,ObjectWrongType;
```

```
  ReadErr      = ReadError,Protected;

  WriteErr     = WriteError,DiskNotValidated,DiskFull,
                 Protected,NotADosDisk,NoDisk;
  DirErr       = NoParentDir,ObjectExists,DirNotFound;

  DeleteErr    = Protected,ObjectInUse,DirectoryNotEmpty;

  InternalErr  = NoFreeStore,TaskTableFull,LineTooLong,FileNotObject,
                 InvalidResidentLibrary,NoDefaultDir,BadStreamName,
                 ObjectTooLarge,ActionNotKnown,InvalidComponentName,
                 InvalidLock,TooManyLevels,SeekError;

  AllDosErr    = Protected,OpenErr,ReadErr,WriteErr,DirErr,DeleteErr,
                 InternalErr;


GROUP
  DosExceptionGrp =
                 ActionNotKnown,              BadStreamName,
                 CommentTooBig,               DeleteProtected,
                 DeviceNotMounted,            DirectoryNotEmpty,
                 DirNotFound,                 DiskFull,
                 DiskNotValidated,            DiskWriteProtected,
                 EOF,                         FileNotObject,
                 InvalidComponentName,        InvalidLock,
                 InvalidResidentLibrary,      LineTooLong,
                 NoDefaultDir,                NoDisk,
                 NoFreeStore,                 NoMoreEntries,
                 NoParentDir,                 NotADosDisk,
                 ObjectExists,                ObjectInUse,
                 ObjectNotFound,              ObjectTooLarge,
                 ObjectWrongType,             ReadError,
                 ReadProtected,               RenameAcrossDevices,
                 SeekError,                   TaskTableFull,
                 TooManyLevels,               WriteError,
                 WriteProtected;


  DosExceptGroupGrp = Protected,OpenErr,ReadErr,WriteErr,DirErr,DeleteErr,
                      InternalErr;


GROUP
  DosErrorGrp  = writeProtect, validating, validated,DiskStatus,
                noDiskPresent, unreadableDisk, dosDisk, notReallyDos,
                kickstartDisk,DiskType,noFreeStore,taskTableFull,
                lineTooLong,fileNotObject,invalidResidentLibrary,
                noDefaultDir,objectInUse,objectExists,dirNotFound,
                objectNotFound,badStreamName,
                objectTooLarge,actionNotKnown,invalidComponentName,
                invalidLock,objectWrongType,diskNotValidated,
                diskWriteProtected,renameAcrossDevices,tooManyLevels,
                deviceNotMounted,seekError,commentTooBig,diskFull,
```



```
                 deleteProtected,writeProtected,readProtected,
                 notADosDisk,noDisk,noMoreEntries;


|------------------------------------------------------------------------------|
|  1. Ein- und Ausgabe                                                         |
|------------------------------------------------------------------------------|

PROCEDURE Close(VAR file : FileHandlePtr):LONGBOOL;

PROCEDURE Open(REF name       : STRING;
                   accessMode : OpenMode := oldFile;
                   context    : ContextPtr := NIL):FileHandlePtr;

PROCEDURE Read(    file       : FileHandlePtr;
                   buffer     : ANYPTR;
                   length     : LONGINT):LONGINT;


PROCEDURE Write(   file       : FileHandlePtr;
                   buffer     : ANYPTR;
                   length     : LONGINT):LONGINT;

GROUP
  DosIOGrp   = Open,Open,Close,Read,FileHandlePtr,FileHandlePtr,Seek,Write,
               readWrite,readOnly,oldFile,newFile,beginning,current,end;

|------------------------------------------------------------------------------|
|  2. Dateiverwaltung                                                          |
|------------------------------------------------------------------------------|

VAR
  CD_at_ProgrammEnd : FileLockPtr;

PROCEDURE CreateDir(REF name : STRING;
                        context : ContextPtr:=NIL):FileLockPtr;

PROCEDURE CurrentDir(lock : FileLockPtr):FileLockPtr;

PROCEDURE ParentDir(lock : FileLockPtr;
                    context : ContextPtr:=NIL):FileLockPtr;

PROCEDURE DupLock(lock : FileLockPtr;
                  context : ContextPtr:=NIL):FileLockPtr;
```

```
PROCEDURE Lock(REF name       : STRING;
                   accessMode : LockMode;
                   context    : ContextPtr:=NIL):FileLockPtr;

PROCEDURE Unlock(VAR lock : FileLockPtr);

GROUP
  DosFileGrp  =  CreateDir,FileLockPtr,CurrentDir,DeleteFile,Examine,ExNext,
                 FileInfoBlockPtr,FileInfoBlock,Info,InfoDataPtr,InfoData,
                 ParentDir,Rename,SetComment,SetProtection,ProtectionFlagSet,
                 ProtectionFlags,DupLock,Input,IoErr,IsInteractive,Lock,Output,
                 Unlock,sharedLock,exclusiveLock,accessRead,accessWrite;


  DosProcessGrp  = CreateProc,DateStamp,Delay,DeviceProc,Exit,
                   WaitForChar,ticksPerSecond,Date,Process,ProcessPtr,
                   ProcessId;

  DosSegmentGrp  = Execute,LoadSeg,UnLoadSeg,BPTR;

  PacketGrp      = DosPacket,DosPacketPtr,StandardPacket,StandardPacketPtr,
                   nil,getBlock,setMap,die,event,currentVolume,locateObject,
                   renameDisk,write,read,freelock,deleteObject,renameObject,
                   moreCache,copyDir,waitChar,setProtect,createDir,
                   examineObject,examineNext,diskInfo,info,flush,setComment,
                   parent,timer,inhibit,diskType,diskChange,setDate,
                   screenMode,readReturn,writeReturn,findUpDate,findInput,
                   findOutput,actionEnd,seek,truncate,writeLock,
                   GetPacket,QueuePacket;

  All            = DosIOGrp,DosFileGrp,DosProcessGrp,PacketGrp,DosSegmentGrp,
                   DosErrorGrp,DosExceptionGrp,DosExceptGroupGrp,
                   BSTR,DeviceListPtr,DatePtr,PathInfoPtr,PathInfo,
                   ok,warn,error,fail,ctrlC,ctrlD,ctrlE,ctrlF,
                   CommandLineInterfacePtr,Process,FileHandle,RootNodePtr,
                   DosLibrary,DosLibraryPtr,TaskArray,DosInfoPtr,TaskArrayPtr,
                   DosEnvecPtr,FileSysStartupMsgPtr,ResidentSegmentPtr,
                   ResidentSegment,RootNode,DosInfo,CommandLineInterface,
                   DeviceListType,DeviceList,DeviceNode,DeviceNodePtr,DosEnvec,
                   FileLock,FileSysStartupMsg,DosBase,LockMode;

END T_Dos.
```

Hier sind alle Exceptions und Exceptiongruppen definiert, die bei Dateioperationen auftreten können. Des weiteren kann man aus diesem Modul alle Bezeichner importieren, die auch in `Dos` definiert sind. Der Unterschied zu Dos ist, daß geöffnete Files automatisch wieder geschlossen, Locks wieder freigegeben werden. Falls bei einer Schreib- oder Leseoperation ein Fehler auftritt, wird eine Exception ausgelöst, wird versucht über ein Filende hinauszulesen, wird die Exception `EOF` ausgelöst. Außerdem wird dafür gesorgt, daß wenn mittels `CurrentDir` das aktuelle Verzeichnis geändert wurde, dieses am Programende wieder dem entspricht, das beim Programmbegin gesetzt war.

7.27.4.2 T_Exec

```
DEFINITION MODULE T_Exec;
(* $A- *)
FROM System    IMPORT BITSET,LONGSET,PROC,SysStringPtr,BPTR,Regs;
FROM Resources IMPORT ContextPtr;
FROM Hardware  IMPORT IntFlags;
FROM Exec      IMPORT NodeType,NodePtr,MinListPtr,MinNodePtr,MinNode,
                      Node,MinList,List,ListPtr,Interrupt,InterruptPtr,IntVector,
                      SoftIntList,MemReqs,MemReqSet,MemTypeSet,MemChunkPtr,
                      MemChunk,MemHeader,MemHeaderPtr,MemEntry,MemList,
                      MemListPtr,TaskFlags,TaskFlagSet,TaskState,
                      MsgPortAction,SigAbort,SigChild,SigBlit,SigSingle,
                      SigDos,vectSize,reserved,base,userDef,nonStd,extFunc,
                      expunge,close,open,LibFlags,LibFlagSet,Device,
                      TaskSignals,DevicePtr,UnitFlags,UnitFlagSet,UnitPtr,
                      Unit,invalid,reset,read,write,update,clear,stop,start,
                      flush,nonstd,abortIO,beginIO,IOFlagSet,quick,openFail,
                      aborted,noCmd,badLength,Semaphore,SemaphorePtr,
                      SemaphoreRequest,ResidentFlags,ResidentFlagSet,
                      ResidentPtr,Resident,matchWord,AttnFlags,AttnFlagSet,
                      ExecBaseType,ExecBasePtr,ExecBase,AddLibrary,
                      MakeFunctions,MakeLibrary,RemLibrary,SetFunction,
                      SumLibrary,InitCode,InitStruct,InitResident,
                      FindResident,Alert,Debug,RawDoFmt,RawIOInit,RawPutChar,
                      Disable,Enable,Forbid,Permit,SetSR,SuperState,UserState,
                      SetIntVector,Cause,Allocate,Deallocate,AllocMem,AllocAbs,
                      FreeMem,AvailMem,TypeOfMem,NewList,Insert,AddHead,AddTail,
                      Remove,RemHead,RemTail,Enqueue,FindName,AddTask,RemTask,
                      FindTask,SetTaskPri,SetSignal,SetExcept,Wait,Signal,
                      AddPort,AddPort,RemPort,PutMsg,WaitPort,FindPort,
                      AddDevice,RemDevice,DoIO,SendIO,CheckIO,WaitIO,AbortIO,
                      AddResource,RemResource,OpenResource,InitSemaphore,
                      AttemptSemaphore,FindSemaphore,AddSemaphore,RemSemaphore,
                      Procure,Vacate,SumKickData,AddMemList,CopyMem,
                      CopyMemQuick,ObtainSemaphore,ReleaseSemaphore,
                      ObtainSemaphoreList,ReleaseSemaphoreList,
                      SignalSemaphore,SignalSemaphorePtr,
                      TaskSignals,IOReturn,TaskSigSet,
                      IOCommand,nonstdVAL,IOFlags,IOReturn;

TYPE
  MsgPortPtr        = POINTER TO MsgPort;
  MessagePtr        = POINTER TO Message;
  LibraryPtr        = POINTER TO Library;
  IORequestPtr      = POINTER TO IORequest;
```

```
  IOStdReqPtr        = POINTER TO IOStdReq;
  TaskPtr            = POINTER TO Task;

DEFINITION MODULE PortRes = Resources.ResHandles(MsgPortPtr);
DEFINITION MODULE MsgRes  = Resources.ResHandles(MessagePtr);
DEFINITION MODULE LibRes  = Resources.ResHandles(LibraryPtr);
DEFINITION MODULE IoRes   = Resources.ResHandles(IORequestPtr);
DEFINITION MODULE TaskRes = Resources.ResHandles(TaskPtr);

TYPE
  MsgPort         = RECORD OF Exec.MsgPort,PortRes.ResHandle END;
  Message         = RECORD OF Exec.Message,MsgRes.ResHandle  END;
  Library         = RECORD OF Exec.Library,LibRes.ResHandle  END;
  IORequest       = RECORD OF Exec.IORequest,IoRes.ResHandle END;
  IOStdReq        = RECORD OF Exec.IOStdReq,IORequest        END;
  Task            = RECORD OF Exec.Task,TaskRes.ResHandle    END;

EXCEPTION
  NoFreeSignal         : "No signal available";
  NoMsgPort            : "Failed to create msg port";
  OpenError            : "Could not open Device";
  NotEnoughStackSpace  : "Not enough stack space";
  CantKillMainTask     : "Can not kill main task";
  TaskNeedsName        : "Task needs a name";

(*========================= Library ========================================*)

PROCEDURE OpenLibrary(REF name     : STRING;
                          version  : LONGINT):LibraryPtr;

PROCEDURE CloseLibrary(library : LibraryPtr);


PROCEDURE OldOpenLibrary(REF libName IN A1 : STRING):LibraryPtr;


GROUP
  LibGrp   = OpenLibrary,CloseLibrary,AddLibrary,MakeFunctions,MakeLibrary,
             RemLibrary,OldOpenLibrary,SetFunction,SumLibrary,LibraryPtr,BPTR;

(*--------------------------------------------------------------------------*)
(*  Listen-Funktionen                                                       *)
(*--------------------------------------------------------------------------*)

GROUP ListGrp = NodeType,Node,NodePtr,List,ListPtr,
                MinNode,MinNodePtr,MinList,MinListPtr,
                NewList,Insert,AddHead,AddTail,Remove,RemHead,RemTail,
                Enqueue,FindName;

(*--------------------------------------------------------------------------*)
(*  Task-Funktionen                                                         *)
(*--------------------------------------------------------------------------*)


PROCEDURE CreateTask(REF name      : STRING;
```



```
                         priority  : SHORTINT;
                         initPC    : ANYPTR;
                         stackSize : LONGINT    := 20000;
                         context   : ContextPtr := NIL):TaskPtr;

PROCEDURE DeleteTask(task : TaskPtr);



GROUP TaskGrp = TaskPtr,Task,TaskFlags,TaskFlagSet,TaskState,
                AddTask,RemTask,CreateTask,DeleteTask,
                FindTask,SetTaskPri;


(*--------------------------------------------------------------------------*)
(*  Message-Funktionen                                                      *)
(*--------------------------------------------------------------------------*)

PROCEDURE CreatePort(REF portName : STRING     :="";
                         priority : SHORTINT   := 0;
                         context  : ContextPtr := NIL):MsgPortPtr;

PROCEDURE DeletePort(port : MsgPortPtr);

PROCEDURE GetMsg(port : MsgPortPtr;context : ContextPtr := NIL):MessagePtr;

PROCEDURE ReplyMsg(msg : MessagePtr);

GROUP PortGrp = MsgPort,MsgPortPtr,MsgPortAction,MessagePtr,
                AddPort,RemPort,CreatePort,DeletePort,
                FindPort;

      MsgGrp  = MsgPortPtr,MessagePtr,
                PutMsg,GetMsg,ReplyMsg,WaitPort,FindPort;

(*--------------------------------------------------------------------------*)
(*  Device-Funktionen                                                       *)
(*--------------------------------------------------------------------------*)

|  Funktion  : Öffnet ein Device
|  Parameter : packet   <- Zeiger auf einen IORequest. Verwendet man hier
|                          mit einem Nachfolgertypen , so muß man die
|                          Customfelder selbst initialisieren.
|              name     <- Name des Devices
|              len      <- Wirkliche Länge des Verwendeteten Typen
|              unit     <- Unit des Devices, die geöffnet werden soll
|              context  <- Context, zu dem dieses Device geöffnet werden soll
|
PROCEDURE OpenDevice(    packet  : IORequestPtr;
                     REF name    : STRING;
                         len     : CARDINAL;
                         unit    : LONGCARD  :=0;
                         flags   :=LONGSET:{};
                         context : ContextPtr:=NIL);
```

```
|
| Funktion  : Schließt ein mit diesem Modul geöffnetes Device
| Parameter : packet <- Zeiger auf den IORequest des Devices, das
|                       geschlossen werden soll.
|
PROCEDURE CloseDevice(packet : IORequestPtr);


GROUP DeviceGrp = Device,DevicePtr,IORequest,
                  AddDevice,RemDevice;

      ExecIOGrp = IORequestPtr,IOStdReqPtr,DoIO,SendIO,CheckIO,WaitIO,
                  AbortIO,invalid,reset,read,write,update,clear,stop,
                  start,flush,quick;

(*------------------------------------------------------------------------------*)
(*  Resource-Funktionen                                                         *)
(*------------------------------------------------------------------------------*)


GROUP
  ResourceGrp   = AddResource,RemResource,OpenResource;


(*------------------------------------------------------------------------------*)
(*  Semaphore-Funktionen                                                        *)
(*------------------------------------------------------------------------------*)


GROUP SemaphoreGrp = SignalSemaphore,List,
                     InitSemaphore,ObtainSemaphore,ReleaseSemaphore,
                     AttemptSemaphore,ObtainSemaphoreList,
                     ReleaseSemaphoreList,
                     FindSemaphore,AddSemaphore,RemSemaphore;

GROUP
  KickGrp   = SumKickData,AddMemList,CopyMem,CopyMemQuick;

  All       = LibGrp,ListGrp,TaskGrp,PortGrp,
              MsgGrp,DeviceGrp,ExecIOGrp,ResourceGrp,SemaphoreGrp,KickGrp,
              SoftIntList,IntVector,MemChunkPtr,MemChunk,MemTypeSet,
              MemHeaderPtr,MemEntry,SigAbort,SigChild,SigBlit,SigSingle,
              SigDos,vectSize,reserved,base,userDef,nonStd,extFunc,expunge,
              close,open,LibFlags,LibFlagSet,Library,Device,DevicePtr,
              UnitFlags,UnitFlagSet,UnitPtr,Unit,openFail,aborted,noCmd,
              badLength,ResidentFlagSet,ResidentPtr,Resident,matchWord,
              AttnFlags,AttnFlagSet,ExecBaseType,ExecBase;


END T_Exec.
```

Libraries und Devices, die mit diesem Modul geöffnet worden sind, werden wieder automatisch geschlossen, eigene Tasks und MessagePorts werden am Programende wieder aus dem System entfernt, und Messages, die man vergessen hat zurückzuschicken, werden bei Freigabe des Kontextes an ihren Replyport geschickt.

7.27.4.3 T_Intuition

```
DEFINITION MODULE T_Intuition

IMPORT Exec;
IMPORT Input;

FROM Graphics  IMPORT DrawModes, ViewModes;

FROM Resources IMPORT ContextPtr;

FROM Intuition IMPORT
                  IntuiText,          IntuiTextPtr,        DrawModeSet,
                  autoFrontPen,       autoBackPen,         autoDrawMode,
                  autoLeftEdge,       autoTopEdge,         autoITextFont,
                  autoNextText,       BorderPtr,           Border,
                  DrawModeSet,        ImagePtr,            Image,
                  GadgetPtr,          PropInfoPtr,         StringInfoPtr,
                  GadgetFlags,        GadgetFlagSet,       ActivationFlags,
                  ActivationFlagSet,  gadgHighbits,        gadgHNone,
                  gadgHComp,          GadgetType,
                  GadgInfoPtr,        GadgInfo,            Gadget,
                  boolMask,           BoolInfo,            PropInfoFlags,
                  PropInfoFlagSet,    knobVmin,            knobHmin,
                  maxBody,            maxPot,              PropInfo,
                  BITSET,             StringInfo,          BorderGrp,
                  ImageGrp,           IntuiTextGrp,        IDCMPFlags,
                  IDCMPFlagSet,       selectUp,            selectDown,
                  menuUp,             menuDown,            menuNull,
                  noMenu,             noItem,              noSub,
                  keyCodeQ,           keyCodeX,            keyCodeV,
                  keyCodeB,           keyCodeN,            keyCodeM,
                  cursorUp,           cursorDown,          cursorRight,
                  cursorLeft,         menuHot,             menuCancel,
                  menuWaiting,        okOk,                okAbort,
                  okCancel,           wbenchOpen,          wbenchClose,
                  altLeft,            altRight,            amigaLeft,
                  amigaRight,         amigaKeys,           IntuiMessage,
                  IntuiMessagePtr,    ActivateWindow,      ModifyIDCMP,
                  MoveWindow,         RefreshWindowFrame,  SetWindowTitles,
                  SizeWindow,         WindowLimits,        WindowToBack,
                  WindowToFront,      WindowFlags,         WindowFlagSet,
                  otherRefresh,       NewWindow,
                  Window,             WindowPtr,           IDCMPGrp,
                  RastPortPtr,        ActivateGadget,      AddGadget,
                  AddGList,           ModifyProp,          NewModifyProp,
                  OffGadget,          OnGadget,            RefreshGadgets,
                  RefreshGList,       RemoveGadget,        RemoveGList,
```

```
ClearMenuStrip,     ItemAddress,        (*MenuNum,*)
OffMenu,
OnMenu,             menuEnabled,
miDrawn,            Menu,               MenuPtr,
MenuItemPtr,        MenuItemFlags,      MenuItemFlagSet,
highNone,           checkWidth,         commWidth,
lowCheckWidth,      lowCommWidth,       MenuItem,
LONGSET,            AutoRequest,        BuildSysRequest,
ClearDMRequest,     DisplayAlert,       EndRequest,
FreeSysRequest,     InitRequester,      Request,
SetDMRequest,       (*NewAlert,*)       RequesterFlagSet,
RequesterFlags,     RequesterPtr,       Requester,
deadendAlert,       recoveryAlert,
CloseWorkbench,     DisplayBeep,        GetScreenData,
MakeScreen,         MoveScreen,         OpenWorkbench,
ScreenToBack,       ScreenToFront,      ShowTitle,
WBenchToBack,       WBenchToFront,      ScreenFlags,
ScreenFlagSet,      stdScreenHeight,    customScreen,
NewScreen,          ViewModeSet,        Screen,
ScreenPtr,          ViewPortPtr,        RastPortPtr,
BitMap,             BitMapPtr,          ClearPointer,
DrawBorder,         DrawImage,          IntuiTextLength,
PrintIText,         SetPointer,         IntuiTextGrp,
BorderGrp,          ImageGrp,           AllocRemember,
FreeRemember,       Remember,           RememberPtr,
BeginRefresh,       EndRefresh,         RemakeDisplay,
RethinkDisplay,     CurrentTime,        DoubleClick,
GetDefPrefs,        GetPrefs,           SetPrefs,
filenameSize,       topazSixty,         PrinterPort,
CustomName,         AlphaP101,          Brother15XL,
CbmMps1000,         Diab630,            DiabAdvD25,
DiabC150,           Epson,              EpsonJX80,
Okimate20,          QumeLP20,           HpLaserjet,
HpLaserjetPlus,     SerParShk,          SerParShkSet,
Preferences,        baud110,            baud300,
baud1200,           baud2400,           baud4800,
baud9600,           baud19200,          baudMidi,
pica,               elite,              fine,
draft,              letter,             sixLPI,
eightLPI,           imagePositive,      imageNegative,
aspectHoriz,        aspectVert,         shadeGreyscale,
shadeColor,         usLetter,           usLegal,
nTractor,           wTractor,           custom,
fanfold,            single,             readBits,
writeBits,          stopBits,           bufSizeBits,
buf512,             buf1024,            buf2048,
buf4096,            buf8000,            buf16000,
ViewAddress,        ViewPortAddress,    LockIBase,
UnlockIBase,        ReportMouse,        IDCMPGrp,
DisplayMode,        dMountCode,         eventMax,
Res,                resCount,           Gadgets,
gadgetCount,        ILocks,             numILocks,
FatIntuiMessage,    IBox,               Point,
PenPair,
GadgetInfo,         numIEvents,         IntuitionBase,
```



```
                IntuitionBasePtr,   IntuitionBaseType,  pointerSize,
                topazEighty,                            shadeBW,
                WindowTags,         ScreenTags;

EXCEPTION
  WindowNotOpen   : "Couldnt open window";
  NilPassed       : "Nil passed to procedure";
  ScreenNotOpen   : "Screen couldnt be opened";

GROUP
  IntuiTextGrp  = IntuiText,          IntuiTextPtr,       DrawModeSet,
                  DrawModes,          autoFrontPen,       autoBackPen,
                  autoDrawMode,       autoLeftEdge,       autoTopEdge,
                  autoITextFont,      autoNextText;

  BorderGrp     = BorderPtr,          Border,             DrawModeSet,
                  DrawModes;

  ImageGrp      = ImagePtr,           Image;

  GadgetGrp     = GadgetPtr,          PropInfoPtr,        StringInfoPtr,
                  GadgetFlags,        GadgetFlagSet,      ActivationFlags,
                  ActivationFlagSet,  gadgHighbits,       gadgHNone,
                  gadgHComp,          GadgetType,
                  GadgInfoPtr,        GadgInfo,           Gadget,
                  boolMask,           BoolInfo,           PropInfoFlags,
                  PropInfoFlagSet,    knobVmin,           knobHmin,
                  maxBody,            maxPot,             PropInfo,
                  BITSET,             StringInfo,         BorderGrp,
                  ImageGrp,           IntuiTextGrp;

  IDCMPGrp      = IDCMPFlags,         IDCMPFlagSet,       selectUp,
                  selectDown,         menuUp,             menuDown,
                  menuNull,           noMenu,             noItem,
                  noSub,              keyCodeQ,           keyCodeX,
                  keyCodeV,           keyCodeB,           keyCodeN,
                  keyCodeM,           cursorUp,           cursorDown,
                  cursorRight,        cursorLeft,         menuHot,
                  menuCancel,         menuWaiting,        okOk,
                  okAbort,            okCancel,           wbenchOpen,
                  wbenchClose,        altLeft,            altRight,
                  amigaLeft,          amigaRight,         amigaKeys,
                  IntuiMessage,       IntuiMessagePtr;

PROCEDURE OpenWindow(REF newWindow : NewWindow;
                         context   : ContextPtr:=NIL ): WindowPtr;

PROCEDURE OpenWindowTags(context : ContextPtr := NIL;
                         tags    : LIST OF WindowTags):WindowPtr;

PROCEDURE CloseWindow(VAR window : WindowPtr);
```

```
GROUP
  WindowGrp     = OpenWindow,         ActivateWindow,     CloseWindow,
                  ModifyIDCMP,        MoveWindow,         RefreshWindowFrame,
                  SetWindowTitles,    SizeWindow,         WindowLimits,
                  WindowToBack,       WindowToFront,      WindowFlags,
                  WindowFlagSet,      otherRefresh,       (*superUnused,*)
                  NewWindow,          Window,             WindowPtr,
                  IDCMPGrp,           RastPortPtr,        WindowTags;

  GadgetProcGrp = ActivateGadget,     AddGadget,          AddGList,
                  ModifyProp,         NewModifyProp,      OffGadget,
                  OnGadget,           RefreshGadgets,     RefreshGList,
                  RemoveGadget,                           RemoveGList;

  MenueGrp      = ClearMenuStrip,     ItemAddress,        (*MenuNum,*)
                  (*ItemNum,*)        (*SubNum,*)         OffMenu,
                  OnMenu,             (*MenuStrip,*)      menuEnabled,
                  miDrawn,            Menu,               MenuPtr,
                  MenuItemPtr,        MenuItemFlags,      MenuItemFlagSet,
                  highNone,           checkWidth,         commWidth,
                  lowCheckWidth,      lowCommWidth,       MenuItem,
                  LONGSET;

  ReqGrp        = AutoRequest,        BuildSysRequest,    ClearDMRequest,
                  DisplayAlert,       EndRequest,         FreeSysRequest,
                  InitRequester,      Request,            SetDMRequest,
                  (*NewAlert,*)       RequesterFlagSet,   RequesterFlags,
                  RequesterPtr,       Requester,
                  deadendAlert,                           recoveryAlert;

PROCEDURE OpenScreen(REF newScreen : NewScreen;
                         context   : ContextPtr := NIL):ScreenPtr;

PROCEDURE OpenScreenTags(context   : ContextPtr := NIL;
                         tags      : LIST OF ScreenTags):ScreenPtr;

PROCEDURE CloseScreen(VAR screen : ScreenPtr);


GROUP

  ScreenGrp     = CloseScreen,        CloseWorkbench,     DisplayBeep,
                  GetScreenData,      MakeScreen,         MoveScreen,
                  OpenScreen,         OpenWorkbench,      ScreenToBack,
                  ScreenToFront,      ShowTitle,          WBenchToBack,
                  WBenchToFront,      ScreenFlags,        ScreenFlagSet,
                  stdScreenHeight,    customScreen,       NewScreen,
                  ViewModeSet,        ViewModes,          Screen,
                  ScreenPtr,          ViewPortPtr,        RastPortPtr,
                  BitMap,             ScreenTags,         BitMapPtr;

  GfxGrp        = ClearPointer,       DrawBorder,         DrawImage,
                  IntuiTextLength,    PrintIText,         SetPointer,
```



```
                  IntuiTextGrp,       BorderGrp,          ImageGrp;

  MemGrp        = AllocRemember,      FreeRemember,       Exec.MemReqSet,
                  Exec.MemReqs,       Remember,           RememberPtr;

  RefreshGrp    = BeginRefresh,       EndRefresh,         RemakeDisplay,
                  RethinkDisplay;

  TimeGrp       = CurrentTime,        DoubleClick;

  PrefGrp       = GetDefPrefs,        GetPrefs,           SetPrefs,
                  filenameSize,       pointerSize,        topazEighty,
                  topazSixty,         PrinterPort,        CustomName,
                  AlphaP101,          Brother15XL,        CbmMps1000,
                  Diab630,            DiabAdvD25,         DiabC150,
                  Epson,              EpsonJX80,          Okimate20,
                  QumeLP20,           HpLaserjet,         HpLaserjetPlus,
                  SerParShk,          SerParShkSet,       Preferences,
                  baud110,            baud300,            baud1200,
                  baud2400,           baud4800,           baud9600,
                  baud19200,          baudMidi,           pica,
                  elite,              fine,               draft,
                  letter,             sixLPI,             eightLPI,
                  imagePositive,      imageNegative,      aspectHoriz,
                  aspectVert,         shadeBW,            shadeGreyscale,
                  shadeColor,         usLetter,           usLegal,
                  nTractor,           wTractor,           custom,
                  fanfold,            single,             readBits,
                  writeBits,          stopBits,           bufSizeBits,
                  buf512,             buf1024,            buf2048,
                  buf4096,            buf8000,            buf16000;

  ViewGrp       = ViewAddress,        ViewPortAddress;

  LockGrp       = LockIBase,          UnlockIBase;

  IntuiIOGrp    = ReportMouse,        IDCMPGrp,           Exec.MsgGrp,
                  Input.EventGrp;


  All           = IntuiTextGrp,       BorderGrp,          ImageGrp,
                  GadgetGrp,          IDCMPGrp,           WindowGrp,
                  GadgetProcGrp,      MemGrp,             GfxGrp,
                  ReqGrp,             MenueGrp,           ReqGrp,
                  GadgetProcGrp,      MenueGrp,           ScreenGrp,
                  RefreshGrp,         TimeGrp,            PrefGrp,
                  ViewGrp,            LockGrp,            IntuiIOGrp,
                  DisplayMode,        dMountCode,         eventMax,
                  Res,                resCount,           Gadgets,
                  gadgetCount,        ILocks,             numILocks,
                  FatIntuiMessage,    IBox,               Point,
                  PenPair,
                  GadgetInfo,         numIEvents,         IntuitionBase,
                  IntuitionBasePtr,                       IntuitionBaseType;
```

```
END T_Intuition.
```

Fenster und Screen, die mit diesem Modul geöffnet worden sind, können in Kontexte eingehängt werden, und werden am Programmende wieder automatisch geschlossen.

7.28 Str

Dieses Modul enthält Prozeduren zur Bearbeitung von Strings.
A c h t u n g ! ! !
Keine der Prozeduren in `Str` führt einen Bereichs-Check aus. Sie sind dafür besonders schnell. Also nur benutzen, wenn sicher ist, daß der String, der verwendet wird, groß genug ist; sonst die Prozeduren aus `Strings` benutzen.

```
DEFINITION MODULE Str;

FROM System     IMPORT Regs,Equation;
FROM Exceptions IMPORT RangeViolation;



|================== Caps-Funktions ====================

TYPE
  CapsArr  = ARRAY CHAR OF CHAR;

CONST
  LOW      = CapsArr;
  CAP      = CapsArr;

PROCEDURE LowerString(VAR str IN A0 : STRING);


PROCEDURE CapsString(VAR str IN A0 : STRING);


|=============== Compare ==============================


PROCEDURE Equal(REF str1 IN A0,
                    str2 IN A1 : STRING): BOOLEAN;

PROCEDURE CapsEqual(REF str1 IN A0,
                        str2 IN A1 : STRING):BOOLEAN;


PROCEDURE Greater(REF str1 IN A0,
                      str2 IN A1 : STRING):BOOLEAN;
```

```
PROCEDURE CapsGreater(REF str1 IN A0,
                          str2 IN A1 : STRING):BOOLEAN;

PROCEDURE Compare(REF str1 IN A0,
                      str2 IN A1 : STRING):Equation;

|============== Search ================================

PROCEDURE First(REF str IN A0 : STRING;
                    ch  IN D2 : CHAR):INTEGER;

PROCEDURE Last(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR):INTEGER;

PROCEDURE Search(REF find IN A0,
                     in   IN A1 : STRING):INTEGER;

|================ Modify ==============================

PROCEDURE Seg(REF str     IN A0 : STRING;
                  pos     IN D2,
                  len     IN D3 : INTEGER;
              VAR segment IN A1 : STRING);

PROCEDURE Insert(REF str    IN A0 : STRING;
                 VAR into   IN A1 : STRING;
                     pos    IN D2 : INTEGER);
```



```
PROCEDURE Replace(REF str   IN A0 : STRING;
                  VAR into  IN A1 : STRING;
                      pos   IN D2 : INTEGER);

PROCEDURE Delete(VAR str IN A0 : STRING;
                     pos IN D2,
                     len IN D3 : INTEGER);

PROCEDURE Concat(VAR dest   IN A0 : STRING;
                 REF source IN A1 : STRING);

GROUP
  All  = Equation,CapsArr,CAP,LowerString,CapsString,Equal,
         CapsEqual,Greater,Compare,Seg,First,Last,Search,
         Insert,Replace,Delete,Concat,CapsGreater;

END Str.
```

7.28.1 Groß/Kleinschrift

Zur Umwandlung von kleinen in große Buchstaben, gibt es hier zwei konstante Arrays `CAP` und `LOW`. Ist `c` vom Typ `CHAR` dann ergibt:

- `CAP[c]`, `c` als großen Buchstaben z. B. `CAP["a"]="A"`
- `LOW[c]`, `c` als kleinen Buchstaben z. B. `LOW["A"]="a"`

Zur Bearbeitung von ganzen Strings in dieser Art stehen folgende zwei Prozeduren zur Verfügung:

```
PROCEDURE LowerString(VAR str IN A0 : STRING);
```

Funktion: Verwandelt alle Zeichen eines Strings in Kleinbuchstaben.

Parameter:

`str` ⇔ String, der umgewandelt werden soll.

```
PROCEDURE CapsString(VAR str IN A0 : STRING);
```

Funktion: Verwandelt alle Zeichen eines Strings in Großbuchstaben.

Parameter:

`str` ⇔ String, der umgewandelt werden soll.

7.28.2 Stringvergleiche

Wie Sie wissen, kann man Strings nicht einfach so vergleichen wie andere Variablen, hierfür stehen allerdings einige Prozeduren zur Verfügung:

```
PROCEDURE Equal(REF str1 IN A0,
                    str2 IN A1 : STRING): BOOLEAN;
```

Funktion: Prüft zwei Strings auf Gleichheit.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE` wenn sie gleich sind.


```
PROCEDURE CapsEqual(REF str1 IN A0,
                        str2 IN A1 : STRING):BOOLEAN;
```

Funktion: Wie `Equal`, jedoch ohne Beachtung der Groß/Kleinschreibung `(a=A)`.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE` wenn sie gleich sind.

```
PROCEDURE Greater(REF str1 IN A0,
                      str2 IN A1 : STRING):BOOLEAN;
```

Funktion: Prüft ob `str1` alphabetisch größer als `str2` ist.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE` wenn `str1` größer als `str2` ist.

```
PROCEDURE CapsGreater(REF str1 IN A0,
                          str2 IN A1 : STRING):BOOLEAN;
```

Funktion: Prüft ob `str1` größer als `str2` ist. Ohne Berücksichtigung der Groß/Kleinschreibung. Parameter wie `Greater`

```
PROCEDURE Compare(REF str1 IN A0,
                      str2 IN A1 : STRING):Equation;
```

Funktion: Vergleicht zwei Strings.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `smaller` : `str1` kleiner `str2`

`equal` : `str1` gleich `str2`

`greater` : `str1` größer `str2`

7.28.3 Suchfunktionen

```
PROCEDURE First(REF str IN A0 : STRING;
                    ch  IN D2 : CHAR):INTEGER;
```

Funktion: Sucht nach dem ersten Auftreten eines Zeichens in einem String.

Parameter:

`str` ⇐ Strings, in dem gesucht werden soll.

`ch` ⇐ Zeichen, das gesucht werden soll.

⇒ Position des Zeichens, oder `-1`, falls es nicht gefunden wurde.

```
PROCEDURE Last(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR):INTEGER;
```

Funktion: Sucht nach dem letzten Auftreten eines Zeichens

Parameter:

`str` ⇐ Strings, in dem gesucht werden soll.

`ch` ⇐ Zeichen, das gesucht werden soll.

⇒ Position des Zeichens, oder `-1`, falls es nicht gefunden wurde.

```
PROCEDURE Search(REF find IN A0,
                     in   IN A1 : STRING):INTEGER;
```

Funktion: Sucht nach dem ersten Auftreten eines Strings in einem anderen.

Parameter:

`find` ⇐ String, der gesucht werden soll.

`in` ⇐ String, in dem gesucht werden soll.

⇒ Position des Strings, oder `-1`, falls er nicht gefunden wurde.

7.28.4 Bearbeitungsfunktionen

```
PROCEDURE Seg(REF str     IN A0 : STRING;
                  pos     IN D2,
                  len     IN D3 : INTEGER;
              VAR segment IN A1 : STRING);
```

Funktion: Kopiert aus einem String ein Stück heraus.

Parameter:

`str` ⇐ QuellString.

`pos` ⇐ Position, ab der kopiert werden soll.

`len` ⇐ Länge des Bereichs, der kopiert werden soll.

`segment` ⇔ Zielstring, in ihn wird der Bereich hineinkopiert.

```
PROCEDURE Insert(REF str    IN A0 : STRING;
                 VAR into   IN A1 : STRING;
                     pos    IN D2 : INTEGER);
```

Funktion: Fügt in einen String ein Stück ein. Achten Sie bitte darauf, das im Zielstring genügend Platz ist.

Parameter:

`dest` ⇔ String, in den eingefügt werden soll.

`source` ⇐ String, der eingefügt werden soll.

`pos` ⇐ Position, ab der eingefügt werden soll.

```
PROCEDURE Replace(REF str    IN A0 : STRING;
                  VAR into   IN A1 : STRING;
                      pos    IN D2 : INTEGER);
```

Funktion: Ersetzt einen Teil eines Strings durch einen anderen. Paßt `str` nicht in `into`, bleibt `into` unverändert.

Parameter:

`str` ⇐ String, durch den ein Teil von `into` ersetzt werden soll.

`into` ⇔ String, in den ein Teil eingesetzt werden soll.

`pos` ⇐ Position, ab der ersetzt werden soll.

```
PROCEDURE Delete(VAR str IN A0 : STRING;
                     pos IN D2,
                     len IN D3 : INTEGER);
```

Funktion: Schneidet einen Teil des Strings heraus.

Parameter:

`str`	⇔	String, aus dem ausgeschnitten werden soll.
`pos`	⇐	Position, ab der geschnitten werden soll.
`len`	⇐	Länge des Stücks, das ausgeschnitten werden soll.

```
PROCEDURE Concat(VAR dest   IN A0 : STRING;
                 REF source IN A1 : STRING);
```

Funktion: Fügt zwei Strings zusammen.

Parameter:

`dest`	⇔	Linker Teil des neuen Strings, in dieser Variable findet sich danach auch der zusammengesetzte String.
`source`	⇐	Rechter Teil des neuen Strings.

7.29 Streams

Dieses Module bietet Schreib- und Lesefunktionen für sequentielle[15] Datenströme. Auch wenn diese nicht so flexibel zu handhaben sind wie die Dateien aus dem `FileSystem`, sind Streams jedoch sehr praktisch, wenn es darum geht eine `ASCII`-Datei zu parsen.

```
DEFINITION MODULE Streams;
FROM Resources IMPORT ContextPtr;
FROM ASCII     IMPORT cr,lf,eof,sp,ff;

EXCEPTION
  NoInputStream         : "Stream is not an inputstream";
  NoOutputStream        : "Stream is not an outputstream";
  StreamNotInteractive  : "Stream is not an interactive one";
  AppendNotPossible     : "Append not posssible";
  WrongNumOfQuotes      : "Incorrect number of quotation marks";


GROUP
  ExceptionGrp  = NoInputStream,NoOutputStream,
                  StreamNotInteractive,AppendNotPossible,
                  WrongNumOfQuotes;

TYPE
  Stream  = HIDDEN;


VAR
  CurrentInput,
  CurrentOutput  : Stream;

|===================== Global Streamhandling ================

PROCEDURE OpenInputStream(VAR handle    : Stream;
                          REF name      : STRING;
                              buffSize  : LONGINT    := 256;
                              context   : ContextPtr := NIL);

PROCEDURE OpenOutputStream(VAR handle       : Stream;
                           REF name         : STRING;
                               append       : BOOLEAN   := FALSE;
                               writeLnBuffer: BOOLEAN   := TRUE;
```

[15] sequentiell heißt, auf die einzelnen Bytes einer Datei kann nur hintereinander zugegriffen werden. Es ist nicht möglich wahlfrei darauf zuzugreifen.

```
                                buffSize     : LONGINT   := 256;
                                context      : ContextPtr:= NIL);

PROCEDURE OpenInOutStream(VAR handle         : Stream;
                          REF name           : STRING;
                              writeLnBuffer  : BOOLEAN   := TRUE;
                              buffSize       : LONGINT   := 256;
                              context        : ContextPtr:=NIL);

  DEFINITION MODULE CustomStreams(Handle : ANYPTR);

    TYPE
      ReadProc  = PROCEDURE(VAR c      : CHAR;
                                handle : Handle);
      WriteProc = PROCEDURE(c      : CHAR;
                            handle : Handle);
      CloseProc = PROCEDURE(handle : Handle);


    PROCEDURE OpenCustomStream(VAR handle    : Stream;
                                   customIn,
                                   customOut : Handle    := NIL;
                                   read      : ReadProc  := NIL;
                                   write     : WriteProc := NIL;
                                   close     : CloseProc := NIL;
                                   echo      : BOOLEAN   := FALSE;
                                   context   : ContextPtr:= NIL);
  END CustomStreams;

PROCEDURE CloseStream(VAR handle : Stream);

PROCEDURE SetCurrentInput(handle : Stream);

PROCEDURE SetCurrentOutput(handle : Stream);

PROCEDURE IsInterActive(handle : Stream):BOOLEAN;

GROUP
  GlobalGrp  = OpenInputStream,OpenOutputStream,OpenInOutStream,
               CustomStreams,CloseStream,SetCurrentInput,
               SetCurrentOutput,Stream,CurrentInput,
               CurrentOutput,Stream;


|====================== Read ==============================


TYPE
  Termination = ARRAY [8] OF CHAR;
```



```
  CharPtr     = POINTER TO CHAR;

PROCEDURE Read(VAR c      : CHAR;
                   handle : Stream :=NIL);

PROCEDURE NextChar(handle : Stream :=NIL):CHAR;


PROCEDURE ReadBytes(pos     : ANYPTR;
                    len     : LONGINT;
                    handle  : Stream :=NIL);

PROCEDURE ReadBlock(VAR block   : ANYTYPE;
                        handle  : Stream :=NIL);

PROCEDURE ReadCharsTerminated(pos    : CharPtr;
                              maxLen : LONGINT;
                              terms  := Termination:(sp,lf,cr,
                                                     ff,eof,eof,
                                                     eof,eof);
                              handle : Stream :=NIL):LONGINT;

PROCEDURE ReadString(VAR  str        : STRING;
                          terms      := Termination:(sp,lf,cr,
                                                     ff,eof,eof,
                                                     eof,eof);
                          quotedFlag : BOOLEAN :=TRUE;
                          handle     : Stream :=NIL);


PROCEDURE TermChar(handle : Stream := NIL):CHAR;

PROCEDURE WasQuoted(handle : Stream :=NIL):BOOLEAN;

GROUP
  ReadGrp    = Termination,CharPtr,Read,NextChar,ReadBytes,
               ReadBlock,ReadCharsTerminated,TermChar,WasQuoted;


|====================== Write ============================

PROCEDURE Write(c       : CHAR;
                handle  : Stream := NIL);

PROCEDURE WriteBytes(pos    : ANYPTR;
                     len    : LONGINT;
                     handle : Stream :=NIL);
```

```
PROCEDURE WriteBlock(REF block  : ANYTYPE;
                         handle : Stream := NIL);

PROCEDURE WriteString(REF str     : STRING;
                          handle  : Stream :=NIL);

PROCEDURE CustomWriteBuffer(handle : Stream);

GROUP
  WriteGrp    = Write,WriteBytes,WriteBlock,WriteString,
                CustomWriteBuffer;
  All         = WriteGrp,ExceptionGrp,GlobalGrp,ReadGrp;

END Streams.
```

Exceptions:

NoInputStream Es wurde versucht von einem reinen Ausgabestrom zu lesen.

NoOutputStream Es wurde versucht auf einen reinen Eingabestrom zu schreiben.

StreamNotInteractive Es wurde versucht einen nicht interaktiven (z. B. Datei) Strom als Ein/Ausgabestrom zu öffnen.

AppendNotPossible Nur bei Dateien kann man einen Ausgabestrom mit `append` öffnen.

WrongNumOfQuotes Beim Einlesen eines Strings wurde das Ende erreicht, ohne daß ein schließende Anführungszeichen gelesen wurde.

Weiterhin können die in `T_Dos` definierten Exceptions auftreten.

7.29.1 Stream-Handling

Bevor man einen Stream verwenden kann, muß dieser erst geöffnet werden. Im Gegensatz zu `InOut`, mit dem man ebenfalls einen eigenen Ein/Ausgabestrom öffnen kann, lassen sich hier beliebig viel Ströme gleichzeitig öffnen.


```
PROCEDURE OpenInputStream(VAR handle    : Stream;
                          REF name      : STRING;
                              buffSize  : LONGINT    := 256;
                              context   : ContextPtr := NIL);
```

Funktion: Öffnet einen Eingabestrom. Von diesem kann nur gelesen werden.

Parameter:

`handle`	⇒	Zugriff auf den Strom.
`name`	⇐	Name der Datei, die als Strom geöffnet werden soll. Dies kann auch ein `CON:`-Fenster oder `SER:` sein.
`buffSize`	⇐	Größe des Eingabepuffers.
`context`	⇐	Kontext, zu dem der Strom geöffnet werden soll, um sicherzustellen, daß der Strom auch wieder geschlossen wird. Wird dieser Parameter nicht angegeben, wird der Strom zum aktuellen Kontext geöffnet

```
PROCEDURE OpenOutputStream(VAR handle       : Stream;
                           REF name         : STRING;
                               append       : BOOLEAN   := FALSE;
                               writeLnBuffer: BOOLEAN   := TRUE;
                               buffSize     : LONGINT   := 256;
                               context      : ContextPtr:= NIL);
```

Funktion: Öffnet einen Ausgabestrom. Auf diesen kann nur geschrieben werden.

Parameter:

`handle` ⇒ Zugriff auf den Strom.

`name` ⇐ Name der Datei, die als Strom geöffnet werden soll. Dies kann auch ein `CON-Fenster` oder `SER:` sein.

`append` ⇐ Flag, das angibt, ob an eine alte Datei Zeichen angefügt werden sollen. Ist nur bei nicht interaktiven Strömen möglich. Wird hier `FALSE` angegeben, wird eine neue Datei geöffnet.

`writeLnBuffer`

⇐ Gibt an, ob der Puffer bei einem Zeilenvorschub ausgegeben werden soll.

`buffSize` ⇐ Größe des Eingabepuffers.

`context` ⇐ Kontext, zu dem der Strom geöffnet werden soll, um sicherzustellen, daß der Strom auch wieder geschlossen wird. Ohne diesen Parameter wird der Strom zum aktuellen Kontext geöffnet.


```
PROCEDURE OpenInOutStream(VAR handle        : Stream;
                          REF name          : STRING;
                              writeLnBuffer : BOOLEAN   := TRUE;
                              buffSize      : LONGINT   := 256;
                              context       : ContextPtr:=NIL);
```

Funktion: Öffnet einen Ein/Ausgabestrom. Auf diesem kann sowohl geschrieben als auch gelesen werden. Funktioniert nur mit interaktiven Strömen, also nicht mit Dateien.

Parameter:

`handle` ⇒ Zugriff auf den Strom.

`name` ⇐ Name der Datei, die als Strom geöffnet werden soll. Dies kann auch ein `CON-Fenster` oder `SER:` sein.

`writeLnBuffer`

⇐ Gibt an, ob der Puffer bei einem Zeilenvorschub ausgegeben werden soll.

`buffSize` ⇐ Größe des Eingabepuffers.

`context` ⇐ Kontext, zu dem der Strom geöffnet werden soll, um sicherzustellen, daß der Strom auch wieder geschlossen wird. Ohne diesen Parameter wird der Strom zum aktuellen Kontext geöffnet

Diese Ströme arbeiten alle mit den Schreib/Lesefunktionen von `Dos` zusammen. Für manche Spezialanwendungen benötigt man möglicherweise eigene Schreib/Lesefunktionen. Hierfür existiert ein generisches Modul `CustomStreams`. Nachdem man dieses für seinen Stromtypen ausgeprägt hat, kann man einen solchen Strom mit folgender Prozedur öffnen:

```
PROCEDURE OpenCustomStream(VAR handle     : Stream;
                               customIn,
                               customOut : Handle    := NIL;
                               read      : ReadProc  := NIL;
                               write     : WriteProc := NIL;
                               close     : CloseProc := NIL;
                               echo      : BOOLEAN   := FALSE;
                               context   : ContextPtr:= NIL);
```

Funktion: Öffnet einen Kundenstrom[16]

Parameter:

`handle`	⇒	Zugriff auf den Strom.
`customIn`	⇐	Lesezugriff, der Ihrer Leseroutine `read` übergeben wird.
`customOut`	⇐	Schreibzugriff, der Ihrer Schreibroutine `write` übergeben wird.
`read`	⇐	Leseroutine, mit der alle Lesefunktionen implementiert werden.
`write`	⇐	Schreibroutine, mit der alle Schreibfunktionen implementiert werden.
`close`	⇐	Wird beim Schließen dieses Stromes zweimal aufgerufen, das erste Mal mit „`customIn`“, das zweite Mal mit „`customOut`“.
`echo`	⇐	Gibt an, ob bei einem Aufruf von „`read`“, das gelesene Zeichen gleich darauf mit „`write`“, ausgegeben werden soll.
`context`	⇐	Kontext, zu dem der Strom geöffnet werden soll, um sicherzustellen, daß er zu gegebener Zeit auch wieder geschlossen wird. Wird dieser Parameter nicht angegeben, wird der Strom zum aktuellen Kontext geöffnet.

Nachdem ein solcher Strom mit `openCustonStream` geöffnet wurde, kann er wie jeder andere Strom verwendet werden.

[16]Nicht zu verwechseln mit gleichnamigen Datentypen aus dem Modul `SuperMarket :-)`.


```
PROCEDURE CloseStream(VAR handle : Stream);
```

Funktion: Schließt einen Strom.

Parameter:

`handle`	⇔	Zugriff auf den Strom, der geschlossen werden soll. Auf ihn darf danach nicht mehr zugegriffen werden.

```
PROCEDURE SetCurrentInput(handle : Stream);
```

Funktion: Um nicht bei jeder Lesefunktion den Strom angeben zu müssen, von dem gelesen werden soll, besteht die Möglichkeit einen Strom zum Standardeingabestrom zu erklären. Wird dieser geschlossen, wird automatisch der Strom zum Standardstrom erklärt, der vorher gesetzt war.

Parameter:

`handle`	⇐	Zugriff auf den Strom, der „CurrentInput“ werden soll.

```
PROCEDURE SetCurrentOutput(handle : Stream);
```

Funktion: Um nicht bei jeder Schreibfunktion den Strom angeben zu müssen, auf den geschrieben werden soll, besteht die Möglichkeit einen Strom zum Standardausgabestrom zu erklären. Wird dieser geschlossen, wird automatisch der Strom zum Standardstrom erklärt, der vorher gesetzt war.

Parameter:

`handle`	⇐	Zugriff auf den Strom, der „CurrentOutput“ werden soll.

Da auch das Modul `InOut`, mit diesem Modul arbeitet, werden alle Ausgaben von `InOut` sofern dort kein Strom angegeben ist, auf den Standardstrom gemacht, der mit diesen Funktionen gesetzt wurde.

```
PROCEDURE IsInterActive(handle : Stream):BOOLEAN;
```

Funktion: Ermittelt, ob es sich bei dem übergebenen Strom um einen interaktiven handelt.

Parameter:

`handle` ⇐ Zugriff auf den Strom, der geprüft werden soll.

⇒ `TRUE`, wenn er interaktiv ist.

7.29.2 Lesefunktionen

Verzichtet man beim Aufruf der Read-Prozeduren auf die Übergabe des Zugriffes auf einen Strom, so wird der aktuelle CurrentInput benutzt. Ist dieser `NIL`, also kein Standardeingabestrom gesetzt, wird ein Console-Fenster geöffnet.

```
PROCEDURE Read(VAR c      : CHAR;
                   handle : Stream :=NIL);
```

Funktion: Liest ein Zeichen aus einem Strom.

Parameter:

`c` ⇔ Gelesenes Zeichen.

`handle` ⇐ Strom, von dem gelesen werden soll.

```
PROCEDURE NextChar(handle : Stream :=NIL):CHAR;
```

Funktion: Liefert das jeweils nächste Zeichen im Strom, ohne die Leseposition zu verschieben.

Parameter:

`handle` ⇐ Strom, von dem gelesen werden soll.

⇒ Gelesenes Zeichen.


```
PROCEDURE ReadBytes(pos     : ANYPTR;
                    len     : LONGINT;
                    handle  : Stream :=NIL);
```

Funktion: Liest eine Anzahl Zeichen aus einem Strom an eine bestimmte Speicherstelle.

Parameter:

`pos` ⇐ Position im Speicher, ab der die gelesenen Zeichen abgelegt werden sollen.

`len` ⇐ Anzahl der zu lesenden Zeichen.

`handle` ⇐ Strom, von dem gelesen werden soll.

```
PROCEDURE ReadBlock(VAR block   : ANYTYPE;
                        handle  : Stream :=NIL);
```

Funktion: Liest einen beliebigen Datentyp fester Länge aus einem Strom.

Parameter:

`block` ⇐ Variable mit fester Länge, die gelesen werden soll.

`handle` ⇐ Strom, von dem gelesen werden soll.

```
PROCEDURE ReadCharsTerminated(pos    : CharPtr;
                              maxLen : LONGINT;
                              terms  := Termination:(sp,lf,cr,
                                                    ff,eof,eof,
                                                    eof,eof);
                              handle : Stream :=NIL):LONGINT;
```

Funktion: Liest eine Anzahl Zeichen aus einem Strom, und schreibt sie an eine bestimmte Speicherstelle, bis die Maximalzahl erreicht ist, oder ein bestimmtes Zeichen gelesen wurde. Führende Terminierungszeichen werden überlesen, Terminierungszeichen werden nicht in den Speicher kopiert.

Parameter:

`pos`	⇐	Position im Speicher, ab der die gelesenen Zeichen abgelegt werden sollen.
`maxLen`	⇐	Maximale Anzahl der zu lesenden Zeichen.
`terms`	⇐	Array mit den Zeichen, die das Einlesen abbrechen sollen.
`handle`	⇐	Strom, von dem gelesen werden soll.
	⇒	Anzahl der tatsächlich gelesenen Zeichen.

```
PROCEDURE ReadString(VAR str        : STRING;
                         terms      := Termination:(sp,lf,cr,
                                                   ff,eof,eof,
                                                   eof,eof);
                          quotedFlag : BOOLEAN :=TRUE;
                         handle     : Stream :=NIL);
```

Funktion: Liest einen String aus einem Strom, bis er gefüllt ist oder ein bestimmtes Zeichen gelesen wurde.

Parameter:

`str`	⇔	String, der gelesenen werden soll.
`terms`	⇐	Array mit den Zeichen, die das Einlesen abbrechen sollen. Will man ein solches Terminierungszeichen eingeben, muß dies innerhalb von Anführungszeichen geschehen. Anführungszeichen lassen sich nur innerhalb von Anführungszeichen eingeben, indem man sie darin doppelt `""` schreibt.
`quotedFlag`	⇐	Gibt an, ob Leerzeichen bei der Eingabe unabhängig von `terms`, als Terminierungszeichen gewertet werden soll.
`handle`	⇐	Strom, von dem gelesen werden soll.

```
PROCEDURE TermChar(handle : Stream := NIL):CHAR;
```

Funktion: Liefert das Zeichen, durch das die letzte Eingabe abgebrochen wurde.

Parameter:

`handle`	⇐	Strom, von dem zuletzt gelesen wurde.
	⇒	Zeichen, das den Abbruch verursachte.

```
PROCEDURE WasQuoted(handle : Stream :=NIL):BOOLEAN;
```

Funktion: Gibt an, ob in der letzten Eingabe ein ‘"’ enthalten war.

Parameter:

`handle`	⇐	Strom, von dem zuletzt gelesen wurde.
	⇒	`TRUE`, wenn ein ‘"’ enthalten war.

7.29.3 Schreibfunktionen

Verzichted man beim Aufruf der Write-Prozeduren auf die Übergabe des Zugriffes auf einen Strom, so wird der aktuelle CurrentOutput benutzt. Ist dieser `NIL`, wird ein Console-Fenster geöffnet.

```
PROCEDURE Write(c       : CHAR;
                handle  : Stream := NIL);
```

Funktion: Schreibt ein Zeichen in einen Strom.

Parameter:

`c` ⇐ Zu schreibendes Zeichen.

`handle` ⇐ Strom, auf den geschrieben werden soll.

```
PROCEDURE WriteBytes(pos    : ANYPTR;
                     len    : LONGINT;
                     handle : Stream :=NIL);
```

Funktion: Schreibt eine Anzahl Zeichen von einer bestimmten Speicherstelle aus in einem Strom.

Parameter:

`pos` ⇐ Position im Speicher, von der ab geschrieben werden soll.

`len` ⇐ Anzahl der zu schreibenden Zeichen.

`handle` ⇐ Strom, auf den geschrieben werden soll.

```
PROCEDURE WriteBlock(REF block  : ANYTYPE;
                         handle : Stream := NIL);
```

Funktion: Schreibt einen beliebigen Typ fester Länge in einen Strom.

Parameter:

`block` ⇐ Variable, deren Inhalt geschrieben werden soll.

`handle` ⇐ Strom, auf den geschrieben werden soll.


```
PROCEDURE WriteString(REF str     : STRING;
                          handle  : Stream :=NIL);
```

Funktion: Schreibt einen String in einem Strom.

Parameter:

`str` ⇐ String, der geschrieben werden soll.

`handle` ⇐ Strom, auf den geschrieben werden soll.

```
PROCEDURE CustomWriteBuffer(handle : Stream);
```

Funktion: Normalerweise wird der Puffer eines Streams erst dann ausgegeben, wenn er voll ist oder ein Zeilenvorschub[17] ausgegeben wird. Diese Prozedur führt zur augenblicklichen Ausgabe des Ausgabepuffers.

Parameter:

`handle` ⇐ Strom, dessen Puffer ausgegeben werden soll.

[17] Soweit dies beim öffnen des Streams angegeben wurde.

7.30 Strings

Ebenso wie das Modul `Str` stellt `Strings` Prozeduren und Funktionen zur Stringbearbeitung zur Verfügung, mit dem Unterschied, daß hier ein Bereichscheck durchgeführt wird, und daß viele Prozeduren aus `Str` hier als Funktionen vorhanden sind.

```
DEFINITION MODULE Strings;

FROM System     IMPORT Regs,SysStringPtr,Equation;
FROM Dos        IMPORT BSTR;
FROM Resources  IMPORT ContextPtr,NoContext;
FROM Exceptions IMPORT RangeViolation;

TYPE
  MString  = ARRAY OF CHAR;

|========================= Compare ========================

PROCEDURE Equal(REF str1 IN A0,
                    str2 IN A1 : STRING): BOOLEAN;


PROCEDURE CapsEqual(REF str1 IN A0,
                        str2 IN A1 : STRING):BOOLEAN;


PROCEDURE BCPL_Equal(str1 IN A0,
                     str2 IN A1 : BSTR):BOOLEAN;


PROCEDURE Greater(REF str1 IN A0,
                      str2 IN A1 : STRING):BOOLEAN;


PROCEDURE CapsGreater(REF str1 IN A0,
                          str2 IN A1 : STRING):BOOLEAN;
PROCEDURE Compare(REF str1 IN A0,
                      str2 IN A1 : STRING):Equation;
```



```
|======================= Search ============================


PROCEDURE First(REF str IN A0 : STRING;
                    ch  IN D2 : CHAR):INTEGER;



PROCEDURE Next(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR;
                   pos IN D3 : INTEGER):INTEGER;


PROCEDURE Last(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR):INTEGER;


PROCEDURE Prev(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR;
                   pos IN D3 : INTEGER):INTEGER;


PROCEDURE Search(REF find,in : STRING):INTEGER;


PROCEDURE SearchNext(REF find,in : STRING;
                         pos     : CARDINAL):INTEGER;

|===================== Modify ==============================


$$OwnHeap:=TRUE
PROCEDURE Seg(REF str IN A0 : STRING;
                  pos IN D2,
                  len IN D3 : INTEGER):STRING;


$$OwnHeap:=TRUE
PROCEDURE Concat(REF strs : LIST OF STRING):STRING;
```

```
$$OwnHeap:=TRUE
PROCEDURE Insert(REF str,into : STRING;
                     pos      : INTEGER):STRING;

$$OwnHeap:=TRUE
PROCEDURE Delete(REF str  : STRING;
                     pos,
                     len  : INTEGER):STRING;

$$OwnHeap:=TRUE
PROCEDURE Replace(REF str,
                      into      : STRING;
                      pos IN D2 : INTEGER):STRING;

$$OwnHeap:=TRUE
PROCEDURE Dup(REF str IN A0 : STRING;
                  num IN D2 : INTEGER):STRING;

$$OwnHeap:=TRUE
PROCEDURE Fill(REF str : STRING;
                   ch  : CHAR;
                   pos,
                   len : CARDINAL):STRING;

PROCEDURE CutOut(REF str     IN A0 : STRING;
                     pos     IN D2,
                     len     IN D3 : INTEGER;
                 VAR segment       : STRING);

PROCEDURE InsertIn(REF str    IN A0 : STRING;
                   VAR into         : STRING;
                       pos    IN D2 : INTEGER);

PROCEDURE ReplaceIn(REF str   IN A0 : STRING;
                    VAR into        : STRING;
                        pos   IN D2 : INTEGER);

PROCEDURE Erase(VAR str IN A0 : STRING;
                    pos IN D2,
                    len IN D3 : INTEGER);
```



```
PROCEDURE Append(VAR dest         : STRING;
                 REF source IN A1 : STRING);

|====================== Conversion ======================

PROCEDURE SysStr(REF str IN A0 : STRING):SysStringPtr;

$$OwnHeap:=TRUE
PROCEDURE Str(ptr IN A0 : SysStringPtr):STRING;

PROCEDURE CreateBSTR(REF str      IN A0 : STRING;
                         resident IN D2 : BOOLEAN := FALSE;
                         context  IN A1 : ContextPtr
                                            :=NoContext):BSTR;

$$OwnHeap:=TRUE
PROCEDURE BSTRtoString(ptr IN A0 : BSTR):STRING;

PROCEDURE StrToMStr(REF str IN A0 : STRING;
                    VAR mStr : MString);

$$OwnHeap:=TRUE
PROCEDURE MStrToStr(REF mStr : MString):STRING;

GROUP
  All   = Equation,MString,Equal,CapsEqual,BCPL_Equal,Greater,
          CapsGreater,Compare,First,Next,Last,Prev,Search,
          SearchNext,Seg,Concat,Insert,Delete,Replace,Dup,
          Fill,CutOut,InsertIn,ReplaceIn,Erase,Append,
          SysStr,Str,CreateBSTR,BSTRtoString,MStrToStr;

END Strings.
```

Exceptions:

RangeViolation Wird ausgelöst, wenn eine Stringoperation über die Grenze eines Strings hinausschreiben würde.

7.30.1 Stringvergleiche

Wie Sie wissen, kann man Strings nicht einfach so vergleichen wie andere Variablen, hierfür stehen allerdings einige Prozeduren zur Verfügung:

```
PROCEDURE Equal(REF str1 IN A0,
                    str2 IN A1 : STRING): BOOLEAN;
```

Funktion: Prüft zwei Strings auf Gleichheit.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE`, wenn sie gleich sind.

```
PROCEDURE CapsEqual(REF str1 IN A0,
                        str2 IN A1 : STRING):BOOLEAN;
```

Funktion: Wie `Equal`, jedoch ohne Beachtung der Groß/Kleinschreibung `(a=A)`.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE`, wenn sie gleich sind.

```
PROCEDURE BCPL_Equal(str1 IN A0,
                     str2 IN A1 : BSTR):BOOLEAN;
```

Funktion: Vergleicht zwei `BCPL`-Strings.

Parameter:

`str1`, `str2` ⇐ BCPL-Strings, die verglichen werden sollen.

⇒ `TRUE`, wenn sie gleich sind.

```
PROCEDURE Greater(REF str1 IN A0,
                      str2 IN A1 : STRING):BOOLEAN;
```


Funktion: Prüft ob `str1` alphabetisch größer als `str2` ist.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `TRUE`, wenn `str1` größer als `str2` ist.

```
PROCEDURE CapsGreater(REF str1 IN A0,
                          str2 IN A1 : STRING):BOOLEAN;
```

Funktion: Prüft ob `str1` größer als `str2` ist. Ohne Berücksichtigung der Groß/Kleinschreibung. Parameter wie `Greater`

```
PROCEDURE Compare(REF str1 IN A0,
                      str2 IN A1 : STRING):Equation;
```

Funktion: Vergleicht zwei Strings.

Parameter:

`str1`, `str2` ⇐ Strings, die verglichen werden sollen.

⇒ `smaller` : `str1` kleiner `str2`

`equal` : `str1` gleich `str2`

`greater` : `str1` größer `str2`

7.30.2 Suchfunktionen

```
PROCEDURE First(REF str IN A0 : STRING;
                    ch  IN D2 : CHAR):INTEGER;
```

Funktion: Sucht nach dem ersten Auftreten eines Zeichens in einem String.

Parameter:

`str` ⇐ String, in dem gesucht werden soll.

`ch` ⇐ Zeichen, das gesucht werden soll.

⇒ Position des gesuchten Zeichens, oder `-1`, falls es nicht gefunden wurde.

```
PROCEDURE Next(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR;
                   pos IN D3 : INTEGER):INTEGER;
```

Funktion: Sucht nach dem nächsten Auftreten eines Zeichens.

Parameter:

`str`	⇐ String, in dem gesucht werden soll.
`ch`	⇐ Zeichen, das gesucht werden soll.
`pos`	⇐ Position, ab der gesucht werden soll
.	⇒ Position des gesuchten Zeichens, oder `-1`, falls es nicht gefunden wurde.

```
PROCEDURE Last(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR):INTEGER;
```

Funktion: Sucht nach dem letzten Auftreten eines Zeichens

Parameter:

`str`	⇐ Strings, in dem gesucht werden soll.
`ch`	⇐ Zeichen, das gesucht werden soll.
	⇒ Position des Zeichens oder, `-1`, falls es nicht gefunden wurde.

```
PROCEDURE Prev(REF str IN A0 : STRING;
                   ch  IN D2 : CHAR;
                   pos IN D3 : INTEGER):INTEGER;
```

Funktion: Sucht nach dem voherigen Auftreten eines Zeichens.

Parameter:

`str`	⇐ String, in dem gesucht werden soll.
`ch`	⇐ Zeichen, das gesucht werden soll.
`pos`	⇐ Position, ab der gesucht werden soll
.	⇒ Position des gesuchten Zeichens, oder `-1`, falls es nicht gefunden wurde.


```
PROCEDURE Search(REF find,in : STRING):INTEGER;
```

Funktion: Sucht nach dem ersten Auftreten eines Strings in einem anderen.

Parameter:

`find`	⇐ String, der gesucht werden soll.
`in`	⇐ String, in dem gesucht werden soll.
	⇒ Position des Strings, oder `-1`, falls er nicht gefunden wurde.

```
PROCEDURE SearchNext(REF find,in : STRING;
                         pos     : CARDINAL):INTEGER;
```

Funktion: Sucht nach dem nächsten Auftreten eines Strings.

Parameter:

`find`	⇐ String, der gesucht werden soll.
`in`	⇐ String, in dem gesucht werden soll.
`pos`	⇐ Position, ab der gesucht werden soll.
	⇒ Position des Strings, oder `-1`, falls er nicht gefunden wurde.

7.30.3 Bearbeitungsfunktionen

```
$$OwnHeap:=TRUE
PROCEDURE Seg(REF str IN A0 : STRING;
                  pos IN D2,
                  len IN D3 : INTEGER):STRING;
```

Funktion: Kopiert aus einem String ein Stück heraus.

Parameter:

`str`	⇐ QuellString, aus dem kopiert werden soll.
`pos`	⇐ Position ab der kopiert werden soll.
`len`	⇐ Länge des Bereichs, der kopiert werden soll.
	⇒ Kopierter Bereich.

```
$$OwnHeap:=TRUE
PROCEDURE Concat(REF strs : LIST OF STRING):STRING;
```

Funktion: Setzt aus einer Liste von Strings einen einzelnen neuen zusammen.

Parameter:

`strs` ⇐ Beliebig lange Liste von Strings.

⇒ Zusammengesetzter String.

```
$$OwnHeap:=TRUE
PROCEDURE Insert(REF str,into : STRING;
                     pos      : INTEGER):STRING;
```

Funktion: Fügt in einen String ein Stück ein und gibt das Ergebnis zurück.

Parameter:

`str` ⇐ String, der eingefügt werden soll.

`into` ⇐ String, in den eingefügt werden soll.

⇒ Zusammengesetzter String.

```
PROCEDURE Delete(REF str  : STRING;
                     pos,
                     len  : INTEGER):STRING;
```

Funktion: Schneidet einen Teil des Strings heraus.

Parameter:

`str` ⇐ String, der als Vorlage dient.

`pos` ⇐ Position, ab der geschnitten werden soll.

`len` ⇐ Länge des Stücks, das ausgeschnitten werden soll.

⇒ Ergebnisstring.


```
$$OwnHeap:=TRUE
PROCEDURE Replace(REF str,
                      into      : STRING;
                      pos IN D2 : INTEGER):STRING;
```

Funktion: Ersetzt einen Teil eines Strings durch einen anderen. Paßt `into` nicht in `str`, wird ein `RangeCheck` ausgelöst.

Parameter:

`str` ⇐ String der als Vorlage dient.

`into` ⇐ String durch den ein Teil von `str` ersetzt werden soll.

`pos` ⇐ Position ab der geschnitten werden soll.

`len` ⇐ Länge des Stücks, das ausgeschnitten werden soll.

⇒ Ergebnisstring.

```
$$OwnHeap:=TRUE
PROCEDURE Dup(REF str IN A0 : STRING;
                  num IN D2 : INTEGER):STRING;
```

Funktion: Vervielfältigt einen String in einen anderen und gibt diesen zurück. Z. B. `Dup("Str",3)="StrStrStr"`.

Parameter:

`str` ⇐ String, der vervielfältigt werden soll.

`num` ⇐ Zahl der Vervielfältigungen.

⇒ Ergebnisstring.

```
$$OwnHeap:=TRUE
PROCEDURE Fill(REF str : STRING;
                   ch  : CHAR;
                   pos,
                   len : CARDINAL):STRING;
```

Funktion: Füllt einen Teil eines Strings mit Zeichen.

Parameter:

`str`	⇐	String, in dem gefüllt werden soll.
`pos`	⇐	Position, ab der gefüllt werden soll.
`len`	⇐	Anzahl Felder, die gefüllt werden sollen.
	⇒	Ergebnisstring.

```
PROCEDURE CutOut(REF str     IN A0 : STRING;
                     pos     IN D2,
                     len     IN D3 : INTEGER;
                 VAR segment       : STRING);
```

Funktion: Kopiert aus einem String ein Stück heraus.

Parameter:

`str`	⇐	QuellString.
`pos`	⇐	Position, ab der kopiert werden soll.
`len`	⇐	Länge des Bereichs, der kopiert werden soll.
`segment`	⇔	Zielstring, in ihn wird der Bereich hineinkopiert.

```
PROCEDURE InsertIn(REF str    IN A0 : STRING;
                   VAR into         : STRING;
                       pos    IN D2 : INTEGER);
```

Funktion: Fügt in einen String ein Stück ein.

Parameter:

`str`	⇐	String, der eingefügt werden soll.
`into`	⇔	String, in den eingefügt werden soll.
`pos`	⇐	Position, ab der eingefügt werden soll.


```
PROCEDURE ReplaceIn(REF str   IN A0 : STRING;
                    VAR into        : STRING;
                        pos   IN D2 : INTEGER);
```

Funktion: Ersetzt einen Teil eines Strings durch einen anderen. Paßt `str` nicht in `into`, bleibt `into` unverändert.

Parameter:

`str`	⇐	String, durch den ein Teil von `into` ersetzt werden soll.
`into`	⇔	String, in den ein Teil eingesetzt werden soll.
`pos`	⇐	Position, ab der ersetzt werden soll.

```
PROCEDURE Erase(VAR str IN A0 : STRING;
                    pos IN D2,
                    len IN D3 : INTEGER);
```

Funktion: Schneidet einen Teil des Strings heraus.

Parameter:

`str`	⇔	String, aus dem ausgeschnitten werden soll.
`pos`	⇐	Position, ab der geschnitten werden soll.
`len`	⇐	Länge des Stücke, das ausgeschnitten werden soll.

```
PROCEDURE Append(VAR dest         : STRING;
                 REF source IN A1 : STRING);
```

Funktion: Fügt zwei Strings zusammen.

Parameter:

`dest`	⇔	Linker Teil des neuen Strings, in dieser Variable findet sich danach auch der zusammengesetzte String.
`source`	⇐	Rechter Teil des neuen Strings.

7.30.4 Stringkonvertierung

```
PROCEDURE SysStr(REF str IN A0 : STRING):SysStringPtr;
```

Funktion: Gibt einen Zeiger auf einen Systemstring, der in den übergebenen Clusterstring verweist, zurück nach Übergabe eines ClusterStrings.

Parameter:

`str` ⇐ ClusterString.

⇒ Zeiger auf Systemstring.

```
$$OwnHeap:=TRUE
PROCEDURE Str(ptr IN A0 : SysStringPtr):STRING;
```

Funktion: Gibt einen Clusterstring nach Übergabe eines Zeigers auf einen Systemstring[18] zurück.

Parameter:

`ptr` ⇐ Zeiger auf Systemstring.

⇒ Clusterstring.

[18] Nähere Erkärungen zu den Systemstrings finden Sie im Kapitel 7.31


```
PROCEDURE CreateBSTR(REF str      IN A0 : STRING;
                         resident IN D2 : BOOLEAN := FALSE;
                         context  IN A1 : ContextPtr
                                          :=NoContext):BSTR;
```

Funktion: Gibt einen Zeiger auf einen BCPL-String zurück nach Übergabe eines ClusterStrings.

Parameter:

`str` ⇐ Clusterstring.

`resident` ⇐ Soll der String auch noch nach Beendingung des Programms existieren, muß hier `TRUE` übergeben werden.

`context` ⇐ Kontext, zu dem der Speicher für den BCPL-String alloziert werden soll. Hat natürlich nur eine Auswirkung, wenn `resident= FALSE` ist.

⇒ Zeiger auf BCPL-String.

```
$$OwnHeap:=TRUE
PROCEDURE BSTRtoString(ptr IN A0 : BSTR):STRING;
```

Funktion: Gibt einen ClusterString nach übergabe eines Zeigers auf einen BCPL-String zurück.

Parameter:

`ptr` ⇐ Zeiger auf BCPL-String.

⇒ Clusterstring.

```
PROCEDURE StrToMStr(REF str IN A0 : STRING;
                    VAR mStr : MString);
```

Funktion: Wandelt einen Clusterstring in einen ModulaString um.

Parameter:

`str`	⇐	Clusterstring.
`mStr`	⇔	Modulastring.

```
$$OwnHeap:=TRUE
PROCEDURE MStrToStr(REF mStr : MString):STRING;
```

Funktion: Gibt einen Clusterstring nach übergabe eines Modulastrings zurück.

Parameter:

`mStr`	⇐	Modulastring.
	⇒	Clusterstring.

7.31 System

Dieses Modul wird automatisch bei jedem Modul importiert. Es enthält unter anderem Prozeduren, die der Compiler für Laufzeitchecks, Typkonvertierungen und ähnliches. Daneben sind hier aber auch einige Typen, Variablen und Prozeduren definiert, die Sie in Ihren Programmen verwenden können. **Achtung:** Verändern Sie diese Datei auf keinen Fall, der Compiler würde danach keine lauffähigen Programme mehr generieren!

```
DEFINITION MODULE System;

(* $A- *)

TYPE
  SHORTSET     = SET OF [0..7];
  BITSET       = SET OF [0..15];
  LONGSET      = SET OF [0..31];
  BPTR         = BCPLPTR TO SHORTINT;
  STRINGPTR    = POINTER TO ARRAY OF CHAR;
  PROC         = PROCEDURE;

  SysStringPtr= POINTER TO ARRAY OF CHAR;
  SysWbMsg     = HIDDEN;
  SysTask      = HIDDEN;
  Register     = (D0,D1,D2,D3,D4,D5,D6,D7,
                  A0,A1,A2,A3,A4,A5,A6,A7,
                  FP0,FP1,FP2,FP3,FP4,FP5,FP6,FP7);
  RegSet       = SET OF Register;
  Equation     = (smaller,equal,greater);
  SysInfo      = RECORD
                   linkTime : RECORD
                                days,
                                minute,
                                tick     : LONGINT;
                              END;
                   initA4   : ANYPTR
                 END;

CONST
  GlobalBase  = A4;
  LibraryBase = A6;
  StackPtr    = A7;

GROUP
  Regs         = RegSet,Register,GlobalBase,LibraryBase,StackPtr;
```

```
VAR
  CloseProc     : PROC;
  OldGuru       : PROC;
  StartStack    : ANYPTR;
  GuruId        : LONGINT;
  CLIParamPtr   : STRINGPTR;
  CLIParamLen   : LONGINT;
  WBStartupMsg  : SysWbMsg;
  OwnTask       : SysTask;
  GuruPosition  : ANYPTR;
  OldExcept     : PROC;
  OldSigs       : LONGSET;
  CtrlC         : BOOLEAN;
  HeapStart,
  LastHeap      : ANYPTR;
  HeapSize      : LONGINT;
  HeapStackPtr  : RECORD
                    pos  : ANYPTR;
                    len  : INTEGER
                  END;
  LowerStack    : ANYPTR;
  OldExceptData : ANYPTR;
  OldTrapData   : ANYPTR;
  LocalLength   : LONGINT;
  CtrlCFlag     : BOOLEAN;
  OwnLibBase    : RECORD
                    succ,
                    pred     : ANYPTR;
                    type     : SHORTCARD;
                    pri      : SHORTINT;
                    Name     : SysStringPtr;

                    flags    : SHORTSET;
                    pad      : SHORTCARD;
                    negSize,
                    posSize  : CARDINAL;
                    version,
                    revision : CARDINAL;
                    idString : SysStringPtr;
                    sum      : LONGINT;
                    openCnt  : INTEGER
                  END;
   SegList      : ANYPTR;
   CloseGuruPos : ANYPTR;
   CloseGuruId  : LONGINT;

GROUP
  All  = SHORTSET,BITSET,LONGSET,BPTR,STRINGPTR,PROC,
```



```
         SysStringPtr,SysWbMsg,SysTask,Register,RegSet,
         GlobalBase,LibraryBase,StackPtr,CloseProc,OldGuru,
         StartStack,GuruId,CLIParamPtr,CLIParamLen,WBStartupMsg,
         OwnTask,GuruPosition,OldExcept,OldSigs,CtrlC,HeapStart,
         LastHeap,HeapSize,HeapStackPtr,
         LowerStack;

(* Nicht aufrufen, da Rückgabewert nicht stimmt !!! *)

PROCEDURE MULU32(x IN D0,y IN D1 : LONGINT):LONGINT;
PROCEDURE MULS32(x IN D0,y IN D1 : LONGINT):LONGINT;
PROCEDURE DIVU32(x IN D0,y IN D1 : LONGINT):LONGINT;
PROCEDURE DIVS32(x IN D0,y IN D1 : LONGINT):LONGINT;
PROCEDURE MODU32(x IN D0,y IN D1 : LONGINT):LONGINT;
PROCEDURE MODS32(x IN D0,y IN D1 : LONGINT):LONGINT;

(* auf keinen Fall aufrufen, sicherer Tod !!! *)


(* Nicht aufrufen, da automatisch aufgerufen wird *)

PROCEDURE RealToLong(val IN D0 : REAL):LONGREAL;
PROCEDURE LongToReal(high IN D0,low IN D1 : LONGINT):REAL

(* nicht aufrufen, sicherer Tod !!! *)

PROCEDURE SignalCtrlC;

(*  Bricht das Programm an der aktuellen Position ab,
 *  Rückgabewert ist Err, bei negativem Err wird dieses als
 *  Zeiger auf eine Fehlermeldung gedeutet.
 *
 *  Besser die Standardproceduren aufrufen !!!
 *)

PROCEDURE HALT(Err IN D0 : LONGINT);

PROCEDURE HALT_CALL(Err IN D0 : LONGINT);

PROCEDURE HALT_CALL_LOCAL(Err IN D0 : LONGINT);

PROCEDURE HALT_CALL_ENTRY(Err IN D0 : LONGINT);

(* Proceduren zur Heapverwaltung, besser nicht aufrufen *)
```

```
PROCEDURE InitHeap;

PROCEDURE DisposeHeap;

(* Procedure um Heapspeicher zu Allozieren *)

(* $P- *)
PROCEDURE AllocHeap(Size IN D0 : LONGINT):ANYPTR;

(* Proceduren für die Cardinalität einer Menge, sollten nicht
 * aufgerufen werden, da sie automatisch aufgerufen werden.
 *)

PROCEDURE ShortCardinal(Set IN D0 : SHORTSET):LONGINT;

PROCEDURE WordCardinal(Set IN D0 : BITSET):LONGINT;

PROCEDURE LongCardinal(Set IN D0 : LONGSET):LONGINT;

PROCEDURE StackCheck(Need IN D0 : INTEGER);

PROCEDURE CheckBreak;

PROCEDURE EndBEGIN;

CONST SystemInfo = SysInfo;

PROCEDURE AddExceptHandler(pos IN A0 : ANYPTR);

PROCEDURE RemExcept;

PROCEDURE RaiseAgain;

PROCEDURE ExceptHandler;

PROCEDURE DeferredMethod;

PROCEDURE PutTagData(tagList IN A0 : ANYPTR;
                     value   IN D0 : LONGINT;
                     tag     IN D1 : LONGINT);

PROCEDURE GetTagData(tagList IN A0 : ANYPTR;
                     value   IN D0 : LONGINT;
                     tag     IN D1 : LONGINT):LONGINT;

PROCEDURE IClose;

END  System.
```


7.31.1 Typen

SHORTSET Benötigt ein Byte Länge, ideal zur Bitmanipulation einzelner Bytes.

BITSET Benötigt zwei Byte Länge, ideal zur Bitmanipulation einzelner Worte.

LONGSET Benötigt vier Byte Länge, ideal zur Bitmanipulation einzelner Langworte.

BPTR `BCPLPTR`, wird in `Dos` verwendet. Ein BCPL-Pointer enthält nicht die wirkliche Adresse, um diese zu erhalten, muß er mit „4“ multipliziert werden.

STRINGPTR Zeiger auf ein **`ARRAY OF`** `CHAR`. Sollte man nur genau so verwenden, für „C“-Strings eignet sich der `SysStringPtr` besser.

PROC Häufig verwendeter Prozedurtyp, wenn Prozeduren ohne Parameter übergeben werden sollen.

SysStringPtr Zeiger auf einen „C“-String, wie er in vielen Systemstrukturen verwendet wird.

Register Aufzählungstyp, der die Prozessorregisternummern enthält. Er sollte aufgrund der besseren Dokumentation statt einfacher Zahlen verwendet werden.

Equation Ein Aufzählungstyp für Ordnungsrelationen. Er wird von einigen Vergleichsprozeduren der Standardmodule verwendet, und ist hier definiert, damit alle Module den gleichen Typ verwenden können.

7.31.2 Variablen

GuruId Hier steht die Nummer (0 ... 4000) bzw. die Adresse der Exceptionmeldung (> 4000) der letzten Exception. Besser ist es jedoch, wenn Sie das Modul „`Exceptions`“ verwenden.

CliParamPtr, CliParamLen, WBStartupMsg Für alle die nicht Arguments verwenden wollen.

CtrlC Weist man dieser Variable `FALSE` zu, dann kann das Programm nicht mehr durch `Ctrl-C` abgebrochen werden.

7.31.3 Prozeduren

CheckBreak Da innerhalb von Dos-Aufrufen `Ctrl-C` keine Wirkung hat, sollte man in Programmteilen, in denen viele Dos-Aufrufe stattfinden, zwischendurch diese Prozedur aufrufen.

RaiseAgain Ruft die letzte Exception noch einmal auf.

Bitte lassen Sie von allen anderen Elementen, die nicht hier aufgezählt wurden, die Finger, wir können sonst für keine Folgen garantieren!!

7.32 Trees

Hierbei handelt es sich ähnlich wie bei `Lists` um ein generisches Modul zur Verwaltung von Bäumen. Nähere Informationen zu dieser Datenstruktur finden Sie in Kapitel 4.

```
DEFINITION MODULE Trees;

FROM Lists      IMPORT FileTypeMismatch;
FROM FileSystem IMPORT File;

IMPORT Lists;

EXCEPTION
  NoFather   : "Leaf has no father";

  DEFINITION MODULE StdTrees(Tree : POINTER TO Leaf);

    DEFINITION MODULE LeafList = Lists.BiLists(Tree);

    TYPE
      Leaf       = RECORD OF LeafList.BiNode
                     father : Tree;
                     sons   : LeafList.BiList
                   END;

      Check      = PROCEDURE(t : Tree):BOOLEAN;
      Destructor = PROCEDURE(t : Tree);
      ApplyProc  = PROCEDURE(t : Tree);
      SaveProc    = PROCEDURE(f : File;leaf : Tree);
      LoadProc   = PROCEDURE(f : File):Tree;

    PROCEDURE Init(t : Tree);


    PROCEDURE AddFirstSon(father,son : Tree);

    PROCEDURE AddLastSon(father,son : Tree);

    PROCEDURE AddNextBrother(tree,brother : Tree);

    PROCEDURE AddPrevBrother(tree,brother : Tree);


    PROCEDURE Remove(tree : Tree);

    PROCEDURE RemoveSons(father : Tree);
```

```
PROCEDURE RemoveSons_IF(tree : Tree;if : Check);

PROCEDURE Remove_IF(tree : Tree;if : Check);


PROCEDURE Delete(VAR tree : Tree);

PROCEDURE DeleteSons(father : Tree);

PROCEDURE DeleteSons_IF(tree : Tree;if : Check);

PROCEDURE Delete_IF(VAR tree : Tree;if : Check);


PROCEDURE Destruct(VAR tree : Tree;des : Destructor);

PROCEDURE DestructSons(father : Tree;des : Destructor);

PROCEDURE DestructSons_IF(tree : Tree;
                          if   : Check;
                          des : Destructor);

PROCEDURE Destruct_IF(VAR tree : Tree;
                          if   : Check;
                          des  : Destructor);


PROCEDURE ApplyDepthFirst(tree : Tree;
                          work : ApplyProc);

PROCEDURE ApplyBreadthFirst(tree : Tree;
                            work : ApplyProc);

PROCEDURE ApplyDepthFirstBig(tree     : Tree;
                             pre,
                             between,
                             post     : ApplyProc);


PROCEDURE FindDepthFirst(tree   : Tree;
                         equal  : Check):Tree;

PROCEDURE FindBreadthFirst(tree  : Tree;
                           equal : Check):Tree;


PROCEDURE FindSon(father : Tree;
```



```
                      equal : Check):Tree;

  PROCEDURE FindNextSon(father : Tree;
                        equal : Check;
                        son   : Tree):Tree;


  PROCEDURE Save(tree : Tree;
                 f    : File;
                 part : SaveProc);

  PROCEDURE Load(VAR tree : Tree;
                     f    : File;
                     part : LoadProc);

END StdTrees;


DEFINITION MODULE CursorTree(type : ANYPTR);

  EXCEPTION
    NoCursor : "No cursor set";

  TYPE
    LeafPtr = POINTER TO Leaf;

  DEFINITION MODULE CTree = StdTrees(LeafPtr);

  TYPE
    Leaf    = RECORD OF CTree.Leaf
                data : type;
              END;
    Tree    = RECORD
                root,
                mark,
                cursor : LeafPtr;
              END;

    Check      = PROCEDURE(t : type):BOOLEAN;
    Destructor = PROCEDURE(t : type);
    ApplyProc  = PROCEDURE(t : type);
    SaveProc  = PROCEDURE(f : File;leaf : type);
    LoadProc   = PROCEDURE(f : File):type;


  PROCEDURE Init(VAR tree : Tree);
```

```
PROCEDURE Get(VAR tree : Tree):type;

PROCEDURE GetSubtree(VAR dest,arg : Tree);


PROCEDURE AddFirstSon(VAR tree : Tree;son : type);

PROCEDURE AddLastSon(VAR tree : Tree;son : type);

PROCEDURE AddNextBrother(VAR tree : Tree;brother : type);

PROCEDURE AddPrevBrother(VAR tree : Tree;brother : type);


PROCEDURE AddFirstSonSubtree(VAR tree,son : Tree);

PROCEDURE AddLastSonSubtree(VAR tree,son : Tree);

PROCEDURE AddNextBrotherSubtree(VAR tree,brother : Tree);

PROCEDURE AddPrevBrotherSubtree(VAR tree,brother : Tree);


PROCEDURE Remove(VAR tree : Tree);

PROCEDURE RemoveSons(VAR tree : Tree);

PROCEDURE RemoveSons_IF(VAR tree : Tree;if : Check);

PROCEDURE Remove_IF(VAR tree : Tree;if : Check);


PROCEDURE Delete(VAR tree : Tree);

PROCEDURE DeleteSons(VAR tree : Tree);

PROCEDURE DeleteSons_IF(VAR tree : Tree;if : Check);

PROCEDURE Delete_IF(VAR tree : Tree;if : Check);


PROCEDURE Destruct(VAR tree : Tree;des : Destructor);

PROCEDURE DestructSons(VAR tree : Tree;des : Destructor);

PROCEDURE DestructSons_IF(VAR tree : Tree;
                              if   : Check;
                              des  : Destructor);
```



```
PROCEDURE Destruct_IF(VAR tree : Tree;
                          if   : Check;
                          des  : Destructor);


PROCEDURE ApplyDepthFirst(VAR tree : Tree;
                              work : ApplyProc);

PROCEDURE ApplyBreadthFirst(VAR tree : Tree;
                                work : ApplyProc);

PROCEDURE ApplyDepthFirstBig(VAR tree     : Tree;
                                 pre,
                                 between,
                                 post     : ApplyProc);


PROCEDURE Dup(VAR dest,arg : Tree);

PROCEDURE DupSubtree(VAR dest,arg : Tree);


PROCEDURE Mark(VAR tree : Tree);

PROCEDURE GoMark(VAR tree : Tree);


PROCEDURE FindDepthFirst(VAR tree  : Tree;
                             equal : Check);

PROCEDURE FindBreadthFirst(VAR tree   : Tree;
                               equal  : Check);


PROCEDURE FindSon(VAR tree  : Tree;
                      equal : Check);

PROCEDURE FindNextSon(VAR tree  : Tree;
                          equal : Check);


PROCEDURE Father(VAR tree : Tree);

PROCEDURE FirstSon(VAR tree : Tree);

PROCEDURE LastSon(VAR tree : Tree);

PROCEDURE NextBrother(VAR tree : Tree);
```

```
    PROCEDURE PrevBrother(VAR tree : Tree);

    PROCEDURE Root(VAR tree : Tree);

    PROCEDURE HasCursor(VAR tree : Tree):BOOLEAN;

    PROCEDURE Save(VAR tree : Tree;
                       f    : File;
                       part : SaveProc);

    PROCEDURE Load(VAR tree : Tree;
                       f    : File;
                       part : LoadProc);

  END CursorTree;

END Trees.
```

Exceptions:

NoFather Dies Exception wird ausgelöst, wenn man einem Wurzelknoten versucht einen Bruderknoten[19] hinzuzufügen.

[19]Bruderknoten sind alle Knoten, die den selben Vater haben

7.32.1 StdTrees

Mit diesem Modul lassen sich Bäume mit beliebig vielen Söhnen erzeugen. Voraussetzung, um ein Element in einen solchen Baum einhängen zu können, ist, daß die Elemente Nachfolger von `Leaf` sind.

```
PROCEDURE Init(t : Tree);
```

Funktion: Bevor man einen Baum benützen kann, muß man den Basisknoten initialisieren.

Parameter:

`t`	⇐	Zeiger auf Wurzelknoten, der initalisiert werden soll.

```
PROCEDURE AddFirstSon(father,son : Tree);
```

Funktion: Fügt in die Liste der Söhne eines Knotens einen weiteren am Anfang ein.

Parameter:

`father`	⇐	Zeiger auf den Vaterknoten, dem ein neuer Sohnknoten angehängt werden soll.
`son`	⇐	Zeiger auf den Knoten, der als Sohn eingehängt werden soll.

```
PROCEDURE AddLastSon(father,son : Tree);
```

Funktion: Fügt in die Liste der Söhne eines Knotens einen weiteren am Ende ein.

Parameter:

`father`	⇐	Zeiger auf den Vaterknoten, dem ein neuer Sohnknoten angehängt werden soll.
`son`	⇐	Zeiger auf den Knoten, der als Sohn eingehängt werden soll.

```
PROCEDURE AddNextBrother(tree,brother : Tree);
```

Funktion: Fügt in einer Sohnliste hinter einem Knoten einen neuen Bruder ein.

Parameter:

`tree`	⇐	Zeiger auf den Knoten, dem ein neuer Bruderknoten angehängt werden soll.
`brother`	⇐	Zeiger auf den Knoten, der als Bruder eingehängt werden soll.

```
PROCEDURE AddPrevBrother(tree,brother : Tree);
```

Funktion: Fügt in einer Sohnliste vor einem Knoten einen neuen Bruder ein.

Parameter:

`tree`	⇐	Zeiger auf den Knoten, dem ein neuer Bruderknoten vorangestellt werden soll.
`brother`	⇐	Zeiger auf den Knoten, der als Bruder eingehängt werden soll.


```
PROCEDURE Remove(tree : Tree);
```

Funktion: Entfernt einen Knoten aus einem Baum. Dabei wird er nur aus der Baumstruktur entfernt, jedoch nicht freigegeben.

Parameter:

`tree`	⇐	Zeiger auf den Knoten, der entfernt werden soll.

```
PROCEDURE RemoveSons(father : Tree);
```

Funktion: Entfernt alle Söhne eines Knotens aus den Baum.

Parameter:

`father`	⇐	Zeiger auf den Knoten, dessen Söhne entfernt werden sollen.

```
TYPE
  Check      = PROCEDURE(t : Tree):BOOLEAN;
PROCEDURE RemoveSons_IF(tree : Tree;if : Check);
```

Funktion: Enfernt alle Söhne eines Knotens, für die eine Bedingung erfüllt ist.

Parameter:

`father`	⇐	Zeiger auf den Knoten, dessen Söhne entfernt werden sollen.
`if`	⇐	Testprozedur, die jeden Konten übergeben bekommt, und `TRUE` zurückliefern muß, wenn der Knoten entfernt werden soll.

```
PROCEDURE Remove_IF(tree : Tree;if : Check);
```

Funktion: Enfernt alle Knoten eines Baumes, für die eine Bedingung erfüllt ist. **Parameter:**

`tree`	⇐	Zeiger auf den Knoten, mit dem begonnen werden soll.
`if`	⇐	Testprozedur, die jeden Konten übergeben bekommt, und `TRUE` zurückliefern muß, wenn der Knoten entfernt werden soll.

```
PROCEDURE Delete(VAR tree : Tree);
```

```
PROCEDURE DeleteSons(father : Tree);
```

```
PROCEDURE DeleteSons_IF(tree : Tree;if : Check);
```

```
PROCEDURE Delete_IF(VAR tree : Tree;if : Check);
```

Funktion: Wie die entsprechenden `Remove...`-Funktionen, jedoch werden alle Knoten, die entfernt werden, mit `Dispose` freigegeben.

```
PROCEDURE Destruct(VAR tree : Tree;des : Destructor);
```

```
PROCEDURE DestructSons(father : Tree;des : Destructor);
```

```
PROCEDURE DestructSons_IF(tree : Tree;
                          if   : Check;
                          des : Destructor);
```

Funktion: Wie die entsprechenden `Remove...`-Funktionen, jedoch werden alle Knoten, die entfernt werden, mit einer zusätzlich übergebenen Destruktorprozedur freigegeben. Mehr Informationen zu diesem Verfahren finden Sie in der Beschreibung von `Lists`.

```
TYPE
  ApplyProc  = PROCEDURE(t : Tree);
PROCEDURE ApplyDepthFirst(tree : Tree;
                          work : ApplyProc);
```

Funktion: Wendet eine Funktion auf alle Knoten des Baumes an (Tiefensuche). Dabei werden erst alle Nachfolger des ersten Nachfolgers des Startknotens besucht, bevor es sich dem zweiten zuwendet.

Parameter:

`tree`	⇐	Startknoten.
`work`	⇐	Prozedur, mit der alle Knoten bearbeitet werden sollen.


```
PROCEDURE ApplyBreadthFirst(tree : Tree;
                            work : ApplyProc);
```

Funktion: Wendet eine Funktion auf alle Knoten des Baumes an (Breitensuche). Besucht erst alle direkten Nachfolger eines Knotens, bevor es sich deren Nachfolgern zuwendet.

Parameter:

`tree`	⇐	Startknoten.
`work`	⇐	Prozedur, mit der alle Knoten bearbeitet werden sollen.

```
PROCEDURE ApplyDepthFirstBig(tree     : Tree;
                             pre,
                             between,
                             post     : ApplyProc);
```

Funktion: Wie `ApplyDepthFirst`, jedoch mit dem Unterschied, daß man drei Prozeduren mitgeben kann. Die eine wird jeweils mit dem Knoten aufgerufen, bevor mit dessen Söhnen weitergemacht wird, die zweite für jeden Sohn, die Dritte mit dem Knoten nachdem alle seine Söhne bearbeitet wurden. Dies geschieht selbstverständlich rekursiv mit allen Knoten. Übergibt man bei einer der Prozeduren `NIL`, wird diese nicht aufgerufen.

Parameter:

`tree`	⇐	Startknoten.
`pre`	⇐	Prozedur, die vor der Bearbeitung der Söhne eines Knotens mit diesem aufgerufen wird.
`between`	⇐	Prozedur, die mit jedem Sohn aufgerufen wird.
`post`	⇐	Prozedur, die nach der Bearbeitung aller Söhne eines Knotens mit diesem aufgerufen wird.

```
PROCEDURE FindDepthFirst(tree   : Tree;
                         equal  : Check):Tree;
```

Funktion: Sucht einen Knoten (Tiefensuche). Da das Modul nichts über die Datenfelder der Knoten weiß, muß eine Vergleichsprozedur mitgegeben werden.

Parameter:

`tree`	⇐	Startknoten.
`equal`	⇐	Prozedur, die `TRUE` zurückliefert, wenn der gesuchte Knoten gefunden wurde.
	⇒	Zeiger auf den gefundenen Knoten, oder `NIL`, wenn er nicht gefunden wurde.

```
PROCEDURE FindBreadthFirst(tree  : Tree;
                           equal : Check):Tree;
```

Funktion: Wie `FindDepthFirst`, jedoch in Breitensuche.

```
PROCEDURE FindSon(father : Tree;
                  equal  : Check):Tree;
```

Funktion: Sucht einen Knoten in der Liste der Söhne eines Knotens.

Parameter:

`father`	⇐	Knoten, dessen Sohnliste durchsucht werden soll.
`equal`	⇐	Vergleichsprozedur.
	⇒	Gefundener Knoten, oder `NIL`, falls kein Knoten gefunden wurde, auf den die Bedingung zutraf.


```
PROCEDURE FindNextSon(father : Tree;
                      equal  : Check;
                      son    : Tree):Tree;
```

Funktion: Sucht in einer Sohnliste ab einem bestimmten Knoten.

Parameter:

`father`	⇐	Knoten, dessen Sohnliste durchsucht werden soll.
`equal`	⇐	Vergleichsprozedur.
`son`	⇐	Knoten, ab dem gesucht werden soll.
	⇒	Gefundener Knoten, oder `NIL`, falls kein Knoten gefunden wurde, auf den die Bedingung zutraf.

```
TYPE
  SaveProc   = PROCEDURE(f : File;leaf : Tree);
PROCEDURE Save(tree : Tree;
               f    : File;
               part : SaveProc);
```

Funktion: Speichert einen Baum oder einen Teil davon in einer Datei ab.

Parameter:

`tree`	⇐	Knoten, ab dem gespeichert werden soll.
`f`	⇐	Zugriff auf ein mittels `FileSystem` geöffnetes File.
`part`	⇐	Prozedur, die einen einzelnen Knoten in das übergebene File schreibt. Wird für jeden Knoten einzeln aufgerufen.

```
TYPE
  LoadProc   = PROCEDURE(f : File):Tree;
PROCEDURE Load(VAR tree : Tree;
                   f    : File;
                   part : LoadProc);
```

Funktion: Lädt einen Baum aus einer Datei.

Parameter:

`f`	⇐	Zugriff auf ein mittels `FileSystem` geöffnetes File.
`part`	⇐	Prozedur, die einen einzelnen Knoten erzeugt, und seine Daten aus dem übergebenen File liest. Wird für alle Knoten einzeln aufgerufen.
	⇒	Zeiger auf den Baum.

7.32.2 CursorTrees

Dieses Modul ähnelt `StdTrees` in seiner Funktion, der einzige Unterschied ist, daß die Elemente nicht direkt im Baum hängen, sondern nur Knoten mit einem Zeiger auf die eigentlichen Elemente. Dadurch kann man beliebige Typen in diesem Baum einhängen. Sie müssen keinerlei Nachfolger sein. Außerdem können auf diese Weise Elemente in meheren Bäumen gleichzeitig hängen. Der Nachteil, man kann nicht vom Element aus auf die Baumstruktur zurückgreifen. Aus diesem Grund verwendet man einen Cursor, den man innerhalb des Baumes verschieben kann. Siehe auch `AVLTrees`.

Da die Funktionen zum gößten Teil mit denen von `StdTrees` identisch sind, werden hier nur die wichtigsten beschrieben:

```
PROCEDURE Get(VAR tree : Tree):type;
```

Funktion: Liefert den Zeiger auf das Element, das an dem Knoten hängt, auf den der Cursor gerade zeigt.

```
PROCEDURE Mark(VAR tree : Tree);
```

Funktion: Markiert einen Knoten im Baum.


```
PROCEDURE GoMark(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf die Marke.

```
PROCEDURE Father(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den Vater des aktuellen Knotens.

```
PROCEDURE FirstSon(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den ersten Sohn des aktuellen Knotens.

```
PROCEDURE LastSon(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den letzten Sohn des aktuellen Knotens.

```
PROCEDURE NextBrother(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den nächsten Bruder des aktuellen Knotens.

```
PROCEDURE PrevBrother(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den vorherigen Bruder des aktuellen Knotens.

```
PROCEDURE Root(VAR tree : Tree);
```

Funktion: Setzt den Cursor auf den Wurzelknoten des Baumes.

```
PROCEDURE HasCursor(VAR tree : Tree):BOOLEAN;
```

Funktion: Prüft, ob der Cursor noch auf einen gültigen Wert zeigt, wenn nicht, muß er neu positioniert werden.

7.33 VectorLongReal / VectorReal

Dieses Modul stellt zwei neue Typen zur Verfügung:

- Vektor: Ein Vektor besteht aus drei Komponenten. Man kann ihn als Punkt im dreidimensionalen Raum oder als Pfeil vom Ursprung des Koordinatensystems zu einem Punkt auffassen.
- Matrix: Eine Matrix besteht aus drei Vektoren. Diese drei Vektoren können als Bilder der Einheitsvektoren entlang der Koordinatenachsen aufgefasst werden. Damit legt die Matix eine Abbildung des Koordinatensystems in ein anderes Koordinatensystem fest. Das Bild eines Punktes (oder Vektors) läßt sich durch Multiplikation mit der Matix errechnen.

Außerdem enthält es Prozeduren und Funktionen für die grundlegende Arbeit mit Vektoren und Matrizen. Es existiert jeweils ein Modul für Vektoren mit einfacher (`VectorReal`) sowie eines für Vektoren doppelter (`VectorLongReal`) Genauigkeit zur Verfügung. Da die Prozeduren bis auf den Typen jedoch identisch sind, wird hier nur eines von beiden abgedruckt:

```
DEFINITION MODULE VectorLongReal;

FROM Exceptions IMPORT DivisionByZero;

EXCEPTION
  DeterminanteZero : "Determinante is zero";
  AxisZero         : "Axis is zero";

TYPE
  Vector = ARRAY [0..2] OF LONGREAL;

  Matrix = ARRAY [0..2] OF Vector;
```



```
PROCEDURE VNorm(VAR v IN 10 : Vector);

PROCEDURE VNeg(v : Vector):Vector;

PROCEDURE VAbs(VAR v IN 10 : Vector):LONGREAL;

PROCEDURE VExt(v : Vector;a : LONGREAL):Vector;

PROCEDURE VAdd(REF v : LIST OF Vector):Vector;

PROCEDURE VSub(v1,v2 : Vector):Vector;

PROCEDURE VSMul(VAR v1 IN 10,v2 IN 11 : Vector):LONGREAL;

PROCEDURE VCMul(v1,v2 : Vector):Vector;

PROCEDURE VRotate(VAR v : Vector;VAR axis : Vector);

PROCEDURE VMMul(m : Matrix;v : Vector):Vector;

PROCEDURE MNorm(VAR m IN 10 : Matrix);

PROCEDURE MNeg(m : Matrix):Matrix;

PROCEDURE Det(VAR m IN 11 : Matrix):LONGREAL;

PROCEDURE MExt(m : Matrix;a : LONGREAL):Matrix;

PROCEDURE MAdd(REF m : LIST OF Matrix):Matrix;

PROCEDURE MSub(m1,m2 : Matrix):Matrix;
```

```
PROCEDURE MMul(m1,m2 : Matrix):Matrix;

PROCEDURE MTrans(m : Matrix):Matrix;

PROCEDURE MInvert(m : Matrix):Matrix;

PROCEDURE CalcRotM(VAR m : Matrix;VAR axis : Vector);

GROUP
  All   = Vector,Matrix,VNorm,VNeg,VAbs,VExt,VAdd,VSub,VSMul,
          VCMul,VRotate,VMMul,MNorm,MNeg,Det,MExt,
          MAdd,MSub,MMul,MTrans,MInvert,CalcRotM;

END VectorLongReal.
```

PROCEDURE `VNorm(VAR v IN 10 : Vector);`

Funktion: Normiert einen Vektor auf Absolutbetrag 1. Ist der Absolutbetrag von `v = 0`, so entsteht ein Laufzeitfehler.

Parameter:

`v`	⇔ Vektor, der normiert werden soll.

PROCEDURE `VNeg(v : Vector):Vector;`

Funktion: Negiert einen Vektor. Dabei wird jede Komponente mit einem Minus versehen.

Parameter:

`v`	⇐ Vektor, der negiert werden soll.
	⇒ Negierter Vektort.

PROCEDURE `VAbs(VAR v IN 10 : Vector):LONGREAL;`

Funktion: Berechnet den Absolutbetrag von `v`. Der Absolutbetrag ist immer >= 0.

Parameter:

`v`	⇐ Vektor, dessen Absolutbetrag berechnet werden soll.
	⇒ Betrag, eine reelle Zahl.

PROCEDURE `VExt(v : Vector;a : LONGREAL):Vector;`

Funktion: Multipliziert einen Vektor mit einer `REAL`-Zahl.

Parameter:

`v`	⇐ Vektor, der skaliert werden soll.
`a`	⇐ Betrag, um den der Vektor skaliert werden soll.
	⇒ Skalierter Vektor.

PROCEDURE `VAdd(REF v : LIST OF Vector):Vector;`

Funktion: Addiert beliebig viele Vektoren.

Parameter:

`v`	⇐ Liste von Vektoren, die addiert werden sollen. Die Vektoren werden komponentenweise addiert.
	⇒ Summe der Vektoren.

PROCEDURE `VSub(v1,v2 : Vector):Vector;`

Funktion: Subtrahiert zwei Vektoren.

Parameter:

`v1,v2`	⇐ `v2` wird von `v1` abgezogen. Die Vektoren werden komponentenweise subtrahiert.
	⇒ Ergebnisvektor.

PROCEDURE `VSMul(VAR v1 IN 10,v2 IN 11 : Vector):LONGREAL;`

Funktion: Bildet das Skalarprodukt zweier Vektoren. Das Skalarprodukt ist die Summe der Produkte der einzelnen Komponenten. (Es ist 0, wenn die Eingabevektoren senkrecht zueinander stehen.)

Parameter:

`v1,v2` ⇐ Bildet das Skalarprodukt von `v1` und `v2`.

⇒ reelle Zahl (entspricht dem Produkt der Absolutberäge der Eingabevektoren mal dem cosinus des eingeschlossenen Winkels).

PROCEDURE `VCMul(v1,v2 : Vector):Vector;`

Funktion: Bildet das Kreuzprodukt zweier Vektoren. Der Kreuzproduktvektor ist der Vektor, der senkrecht auf beiden Eingabevektoren steht und einen Absolutbetrag hat, der dem Produkt der Absolutbeträge der beiden Eingabevektoren mal dem sinus des von ihnen eingeschlossenen Winkels entspricht. (Er ist `0|0|0`, wenn `v1` ein Vielfaches von `v2` ist.)

Parameter:

`v1,v2` ⇐ Vektoren, die multipliziert werden sollen.

⇒ Kreuzproduktvektor.

PROCEDURE `VRotate(`**VAR** `v : Vector;`**VAR** `axis : Vector);`

Funktion: Rotiert einen Vektor um eine vorgegebene Achse. Der Winkel wird im Bogenmaß durch den Absolutbetrag der Achse angegeben.

Parameter:

`v` ⇔ Vektor, der rotiert werden soll.

`axis` ⇐ Achse, um die gedreht werden soll.

PROCEDURE `VMMul(m : Matrix;v : Vector):Vector;`

Funktion: Multipliziert einen Vektor mit einer Matrix.

Parameter:

`m` ⇐ Matrix, mit der der Vektor multipliziert werden soll.

`v` ⇐ Vektor, der multipliziert werden soll.

⇒ Ergebnisvektor.

PROCEDURE `MNorm(`**VAR** `m` **IN** `10 : Matrix);`

Funktion: Normiert eine Matrix auf $Determinante = 1$. Ist die Determinante von $m = 0$, so entsteht ein Laufzeitfehler.

Parameter:

`m` ⇔ Matrix, die normiert werden soll.

PROCEDURE `MNeg(m : Matrix):Matrix;`

Funktion: Negiert eine Matrix. Jede Komponente wird dabei mit einem Minus versehen.

Parameter:

`m` ⇐ Matrix, die negiert werden soll.

⇒ Negierte Matrix.

PROCEDURE `Det(`**VAR** `m` **IN** `11 : Matrix):LONGREAL;`

Funktion: Berechnet die Determinante einer Matrix. Die Determinante ist eine reelle Zahl, die die Volumensvergrößerung der Abbildung angibt.

Parameter:

`m` ⇐ Matrix, deren Determinante berechnet werden soll.

⇒ Reelle Zahl.

PROCEDURE `MExt(m : Matrix;a : LONGREAL):Matrix;`

Funktion: Erweitert eine Matrix um einen Faktor. Jede Komponente wird mit dem Faktor multipliziert.

Parameter:

`m` ⇐ Matrix, die erweitert werden soll.

`a` ⇐ Faktor, um den erweitert werden soll.

⇒ erweiterte Matrix.

`PROCEDURE MAdd(REF m : LIST OF Matrix):Matrix;`

Funktion: Addiert eine beliebige Anzahl von Matrizen.

Parameter:

`m`	$\Leftarrow$	Liste von Matrizen, die addiert werden sollen.
`a`	$\Leftarrow$	Faktor, um den erweitert werden soll.
	$\Rightarrow$	erweiterte Matrix.

`PROCEDURE MSub(m1,m2 : Matrix):Matrix;`

Funktion: Subtrahiert zwei Matrizen. Die Matrizen werden komponentenweise subtrahiert.

Parameter:

`m1,m2`	$\Leftarrow$	Matrizen, die subtrahiert werden sollen.
	$\Rightarrow$	Ergebnismatrix.

`PROCEDURE MMul(m1,m2 : Matrix):Matrix;`

Funktion: Multipliziert zwei Matrizen. Die Matrizen werden so multipliziert, daß für die Abbildungen gilt: `m1` nach `m2`.

Parameter:

`m1,m2`	$\Leftarrow$	Matrizen, die multipliziert werden sollen.
	$\Rightarrow$	Ergebnismatrix.

`PROCEDURE MTrans(m : Matrix):Matrix;`

Funktion: Transponiert eine Matrix. Transponieren bedeutet: Vertauschen von Zeilen und Spalten.

Parameter:

`m`	$\Leftarrow$	Matrix, die transponiert werden soll.
	$\Rightarrow$	Ergebnismatrix.

`PROCEDURE MInvert(m : Matrix):Matrix;`

Funktion: Bildet die inverse Matrix. Hat `m` $Determinante = 0$, kann keine inverse Matrix gebildet werden und es entsteht ein Laufzeitfehler.

Parameter:

`m`	$\Leftarrow$	Matrix, die invertiert werden soll.
	$\Rightarrow$	Ergebnismatrix.

`PROCEDURE CalcRotM(VAR m : Matrix;VAR axis : Vector);`

Funktion: Berechnet die Abbildungsmatrix bezüglich einer Drehung um eine beliebige Achse. Ist die Achse der Nullvektor, so entsteht ein Laufzeitfehler.

Parameter:

`m`	$\Leftrightarrow$	Matrix, die berechnet werden soll.
`axis`	$\Leftarrow$	Achse, um die gedreht werden soll.

Kapitel 8

Schnittstellenmodule zu den Libraries

Dieses Kapitel beschreibt die Module, die benötigt werden, um die Funktionen des Amiga-Betriebsystems nutzen zu können. Dabei müssen Sie sich nicht selbst um das Öffnen der Libraries kümmern, da die Module, sobald davon eine Prozedur, Variable oder Konstante importiert wird, die entsprechende Library automatisch öffnen, und am Programmende wieder schliessen. Wird lediglich ein Typ importiert, bleibt die entsprechende Library geschlossen.

Bei der Erstellung haben wir uns bemüht, die C-Includes von Commodore so genau wie möglich nach Cluster umzusetzen. Dabei werden Sie — der den Amiga bisher in „C" programmiert hat — feststellen, daß an einigen Stellen, an denen in „C" ein `APTR` oder ein `void *` steht, hier ein getypter[2] Pointer zu finden ist, auch wurden an einigen Stellen `LONGINT`s in Prozedurdefinitionen, falls an dieser Stelle nur diskrete Werte übergeben werden können (z. B. `Dos.Seek()`), in Aufzählungstypen umgewandelt. Dadurch ist nun eine noch größere Typsicherheit gewährleistet.

Der Typ `LONGBOOL` wurde zur „C"-Kompatibilität eingeführt, da „C" sinnigerweise auch 32-Bit Boolwerte kennt. Bei der Verwendung unterscheidet er sich nicht von einem normalen `BOOLEAN`. Ebenso ein `LONGCHAR`, womit wir entgültig für die Schrift jeder Lebensform dieses Universums gewappnet sein dürften.
Bei Prozeduren, bei denen ein

```
REF str :  STRING
```

angegeben ist, kann man auch einen `SysStringPtr`, als einen Zeiger auf einen „C"-String übergeben.

[2] Es wird bei einer Zuweisung ein Typcheck durchgeführt.

8.1 Asl

```
DEFINITION MODULE Asl;

FROM System   IMPORT SysStringPtr,Regs,PROC;
FROM Exec     IMPORT LibraryPtr;
FROM Graphics IMPORT TextAttr,DrawModeSet,FontStyleSet,FontFlagSet;
FROM Utility  IMPORT tagUser,TagListPtr,StdTags;

TYPE
  AslRequesterPtr = POINTER TO AslRequester;
  AslRequester    = RECORD END;

  WindowPtr  = DEFERRED POINTER Intuition.WindowPtr;
  WBArgPtr   = DEFERRED POINTER Workbench.WBArgPtr;

  | don't extend the following structure, as it will change in the future
  |
  FileRequesterPtr = POINTER TO FileRequester;
  FileRequester    = RECORD OF AslRequester
                       reserved1  :  ANYPTR;
                       file       :  SysStringPtr; | filename
                       dir        :  SysStringPtr; | directoryname
                       reserved2  :  ANYPTR;
                       reserved3  :  SHORTCARD;
                       reserved4  :  SHORTCARD;
                       reserved5  :  ANYPTR;
                       leftEdge   :  INTEGER;      | Preferred window pos
                       topEdge    :  INTEGER;
                       width      :  INTEGER;      | Preferred window size
                       height     :  INTEGER;
                       reserved6  :  INTEGER;
                       numArgs    :  LONGINT;      | for multiselects
                       arglist    :  WBArgPtr;     | a la WB Args
                       userData   :  ANYPTR;       | Applihandle (you may
                                                   | write!!)
                       reserved7  :  ANYPTR;
                       reserved8  :  ANYPTR;
                       pat        :  SysStringPtr; | Pattern match Pointer
                     END; | note - more reserved fields follow

  FileFuncFlags    = (
                     patGad,          | ask for pattern gadget
                     multiSelect = 3, | request multiple selections
                                      | (not for save)
                     newIDCMP,        | Force a new IDCMP (if window != NULL)
                     save,            | for a SAVE operation
                     doMsgFunc,       | Called with Object=IDCMP message
                                      | for other window of shared port.
                                      | You must return pointer to Object,
                                      | asl will reply the Object for you
                     doWildFunc,      | Called with an Object=AnchorPath,
                                      | ZERO return accepts.
                     fff31 = 31       | to make the SET a longword
```

```
                 );
  FileFuncFlagSet  =  SET OF FileFuncFlags;

|
| The following additional flags may be passed with the
| ASL_ExtFlags1 tag.
|

  FileExtFlags     = ( noFiles, matchDirs, fileefDummy=31 );
  FileExtFlagSet   = SET OF FileExtFlags;

  FontRequesterPtr = POINTER TO FontRequester;
  FontRequester    = RECORD OF AslRequester
                       reserved1  :  ARRAY [2] OF ANYPTR;
                       attr       :  TextAttr;  | Returned TextAttr
                       frontPen   :  SHORTCARD; | Returned Pens,
                                                | if selected
                       backPen    :  SHORTCARD;
                       drawMode   :  DrawModeSet;
                       userData   :  ANYPTR;
                     END;

  FontFuncFlags    = (frontColor, backColor, styles, drawMode, fixedWidth,
                      newIdcmp, doMsgFunc, doWildFunc, fontffDummy = 31);
  FontFuncFlagSet  = SET OF FontFuncFlags;

  RequestType      = (fileRequest, fontRequest);

  ModeArray        = ARRAY OF SysStringPtr;

  AslTags          = TAGS OF StdTags
                       dummy          =  tagUser +$80000;
                       hail           :  SysStringPtr;
                       window         :  WindowPtr;
                       leftEdge       :  LONGINT;
                       topEdge        :  LONGINT;
                       width          :  LONGINT;
                       height         :  LONGINT;
                       hookFunc       :  PROC;
                       file           :  SysStringPtr;
                       dir            :  SysStringPtr;
                       fontName       :  SysStringPtr;
                       fontHeight     :  LONGCARD;
                       fontStyles     :  FontStyleSet;
                       fontFlags      :  FontFlagSet;
                       frontPen       :  LONGCARD;       | SHORTCARD
                       backPen        :  LONGCARD;       | SHORTCARD
                       minHeight      :  LONGCARD;       | CARDINAL
                       maxHeight      :  LONGCARD;       | CARDINAL
                       okText         :  SysStringPtr;
                       cancelText     :  SysStringPtr;
                       fileFuncFlags  :  FileFuncFlagSet;
                       fontFuncFlags = tagUser+ $80000 + 20 : FontFuncFlagSet;
                       modeList       :  POINTER TO ModeArray;
                       extFlags1      :  FileExtFlagSet;
```



```
                       pattern       = tagUser+ $80000 + 10 :  SysStringPtr;
                     END;

  AslTagAPtr       = POINTER TO AslTagA;
  AslTagA          = ARRAY OF AslTags;

VAR
  AslBase          :  LibraryPtr;

LIBRARY AslBase BY -30
  PROCEDURE AllocFileRequest(): FileRequesterPtr;

LIBRARY AslBase BY -36
  PROCEDURE FreeFileRequest( fileReq IN A0 : FileRequesterPtr );

LIBRARY AslBase BY -42
  PROCEDURE RequestFile( fileReq IN A0: FileRequesterPtr ): BOOLEAN;

LIBRARY AslBase BY -48
  PROCEDURE AllocAslRequest(type    IN D0: RequestType;
                            tagList IN A0: LIST OF AslTags): AslRequesterPtr;

LIBRARY AslBase BY -48
  PROCEDURE AllocAslRequestA( type    IN D0: RequestType;
                              tagList IN A0: AslTagAPtr ): AslRequesterPtr;

LIBRARY AslBase BY -54
  PROCEDURE FreeAslRequest( request IN A0: AslRequesterPtr );

LIBRARY AslBase BY -60
  PROCEDURE AslRequest( request IN A0: AslRequesterPtr;
                        tagList IN A1: LIST OF AslTags ): BOOLEAN;

LIBRARY AslBase BY -60
  PROCEDURE AslRequestA( request IN A0: AslRequesterPtr;
                         tagList IN A1: AslTagAPtr ): BOOLEAN;


GROUP

  ProcGrp          = AllocAslRequest,AllocAslRequestA,AllocFileRequest,
                     AslRequest,AslRequestA,FreeAslRequest,
                     FreeFileRequest,RequestFile;

  TypeGrp          = AslTagA,AslTagAPtr,AslTags,ModeArray,
                     FileExtFlags,FileExtFlagSet,FileFuncFlags,
                     FileFuncFlagSet,FileRequester,FileRequesterPtr,
                     FontFuncFlags,FontFuncFlagSet,FontRequester,
                     FontRequesterPtr,RequestType;
  All              = TypeGrp,ProcGrp;

END Asl.
```

8.2 Audio

```
DEFINITION MODULE Audio;

| Groups in this module ( in this order ):
| AudioGrp All

(* $A- *)
FROM T_Exec       IMPORT NoFreeSignal,OpenError,IOCommand,IOFlagSet,
                         IOFlags,IORequest,IOReturn,Message,nonstdVAL;
FROM Resources    IMPORT ContextPtr;

CONST

  | System Constants
  hardChannels     = 4;

  minPrec          = -128;
  maxPrec          = 127;

  | values for IORequest.command
  free             = IOCommand( nonstdVAL + 0 );
  setPrec          = IOCommand( nonstdVAL + 1 );
  finish           = IOCommand( nonstdVAL + 2 );
  perVol           = IOCommand( nonstdVAL + 3 );
  lock             = IOCommand( nonstdVAL + 4 );
  waitCycle        = IOCommand( nonstdVAL + 5 );
  allocate         = 32;
  noUnit           = allocate;

  | values for IORequest.flags
  pervol           = IOFlagSet:{ IO4 };
  syncCycle        = IOFlagSet:{ IO5 };
  noWait           = IOFlagSet:{ IO6 };
  writeMessages    = IOFlagSet:{ IO7 };

  | errors returned in IORequest.error
  noAllcation      = IOReturn( $F6 );
  allocFailed      = IOReturn( $F5 );
  channelStolen    = IOReturn( $F4 );


TYPE

  IOAudioPtr =   POINTER TO IOAudio;
  IOAudio    =   RECORD OF IORequest
                   allocKey  : INTEGER;
                   data      : ANYPTR;
                   length    : LONGCARD;
                   period    : CARDINAL;
                   volume    : CARDINAL;
                   cycles    : CARDINAL;
                   writeMsg  : Message;
                 END;
```



```
|   OpenAudio
|
| try to open the audio.device.
| EXCEPTION NoFreeSignal OpenError
|
PROCEDURE OpenAudio(context : ContextPtr:=NIL) : IOAudioPtr;

|   CloseAudio
| close the audio device associated with the request.
| will be called if forgotten.
PROCEDURE CloseAudio(VAR request : IOAudioPtr);

GROUP
  AudioGrp =  hardChannels,minPrec,maxPrec,free,setPrec,finish,
              perVol,lock,waitCycle,noUnit,allocate,pervol,syncCycle,
              noWait,writeMessages,noAllcation,allocFailed,
              channelStolen,IOAudio,IOAudioPtr,OpenAudio,CloseAudio;

  All      = AudioGrp;

END Audio.
```

8.3 BattClockResource

```
DEFINITION MODULE BattClockResource;

(* $A- *)

|S. Herr, 01.10.1992

FROM Exec   IMPORT LibraryPtr;
FROM System IMPORT Regs;

VAR
  BattClockBase : LibraryPtr;

LIBRARY BattClockBase BY -6
  PROCEDURE ResetBattClock();

LIBRARY BattClockBase BY -12
  PROCEDURE ReadBattClock():LONGCARD;

LIBRARY BattClockBase BY -18
  PROCEDURE WriteBattClock(amigaTime IN D0 : LONGCARD);


END BattClockResource.
```


8.4 BattMemResource

```
DEFINITION MODULE BattMemResource;
(* $A- *)

|S. Herr, 01.10.1992

FROM Exec   IMPORT LibraryPtr;
FROM System IMPORT Regs;

|Bit-Offsets & Längen
CONST
  |Amiga-spezifisch : Bits 0-31
  amigaAmnesiaAddr  = 0;
  amigaAmnesiaLen   = 1;
  scsiTimeoutAddr   = 1;
  scsiTimeoutLen    = 1;
  scsiLUnsAddr      = 2;
  scsiLUnsLen       = 1;

  |AMIX-spezisfisch : Bits 32-63

  |Shared           : Bits 64 und höher
  sharedAmnesiaAddr = 64;
  sharedAmnesiaLen  = 1;
  scsiHostIdAddr    = 65;
  scsiHostIdLen     = 3;
  scsiSyncXferAddr  = 68;
  scsiSyncXferLen   = 1;

VAR
  BattMemBase : LibraryPtr;

LIBRARY BattMemBase BY -6
  PROCEDURE ObtainBattSemaphore();

LIBRARY BattMemBase BY -12
  PROCEDURE ReleaseBattSemaphore():LONGCARD;

|Achtung: ReadBattMem erfolgreich, wenn Rückgabewert FALSE ist!
LIBRARY BattMemBase BY -18
  PROCEDURE ReadBattMem( buffer IN A0 : ANYPTR;
                         offset IN D0 : LONGCARD;
                         len    IN D1 : LONGCARD):LONGBOOL;

LIBRARY BattMemBase BY -24
  PROCEDURE WriteBattMem( buffer IN A0 : ANYPTR;
                          offset IN D0 : LONGCARD;
                          len    IN D1 : LONGCARD):LONGBOOL;


END BattMemResource.
```

8.5 Bullet

```
      (* $A- *)
DEFINITION MODULE Bullet;

FROM Utility   IMPORT StdTags;
FROM System    IMPORT Regs,SysStringPtr;
FROM Exec      IMPORT LibraryPtr;
               IMPORT Resources;

TYPE
  Fixed          = RECORD
                     IF KEY : BOOLEAN
                       OF TRUE  THEN long  : LONGINT;
                       OF FALSE THEN int   : INTEGER;
                                     fract : CARDINAL;
                     END
                   END;

  GlyphMapPtr    = POINTER TO GlyphMap;
  GlyphMap       = RECORD
                     bytesPerRow : INTEGER;
                     rows        : INTEGER;
                     leftBlank,
                     topBlank,
                     widthUsed,
                     heightUsed  : INTEGER;
                     xorg,yorg   : Fixed;
                     x0,y0,
                     x1,y1       : INTEGER;
                     width       : Fixed;
                     map         : ANYPTR;
                   END;

  Coord          = RECORD x,y : INTEGER END;

  GlyphWidthNodePtr = POINTER TO GlyphWidth;
  GlyphWidth        = RECORD OF Exec.MinNode;
                        pad   : SHORTCARD;
                        code  : CHAR;
                        width : Fixed;
                      END;

  GlyphTags      = TAGS OF StdTags;
                     deviceDPI    = $80000001 : Coord;
                     dotSize      = $80000002 : Coord;
                     pointHeight  = $80000008 : Fixed;
                     pointHeightL = $80000008 : LONGINT;
                     setFactor    = $80000009 : Fixed;
                     setFactorL   = $80000009 : LONGINT;
                     shearSin     = $8000000A : Fixed;
                     shearSinL    = $8000000A : LONGINT;
                     shearCos     = $8000000B : Fixed;
                     shearCosL    = $8000000B : LONGINT;
                     rotateSin    = $8000000C : Fixed;
```



```
                     rotateSinL   = $8000000C : LONGINT;
                     rotateCos    = $8000000D : Fixed;
                     rotateCosL   = $8000000D : LONGINT;
                     emboldenX    = $8000000E : Fixed;
                     emboldenXL   = $8000000E : LONGINT;
                     emboldenY    = $8000000F : Fixed;
                     emboldenYL   = $8000000F : LONGINT;

                     pointSize    = $80000010 : LONGINT; | in 16tel Punkt
                                                         | = 1/72"

                     glyphCode    = $80000011 : LONGCHAR;
                     glyphCode2   = $80000012 : LONGCHAR;
                     glyphWidth   = $80000013 : Fixed;
                     glyphWidthL  = $80000013 : LONGINT;

                     tagPath      = $80008014 : SysStringPtr;
                     tagList      = $80008015 : ANYPTR;

                     glyphMap     = $80008020 : GlyphMapPtr;
                     widthList    = $80008021 : Exec.MinListPtr;
                     kernPair     = $80008022 : ANYPTR;
                     designKernPair           : ANYPTR;


                     fileIdent    = $80001001 : LONGINT;
                     engine       = $80009002 : SysStringPtr;
                     family       = $80009003 : SysStringPtr;
                     boldName     = $8000A005 : SysStringPtr;
                     italicName   = $8000A006 : SysStringPtr;
                     biName       = $8000A007 : SysStringPtr;
                     symbolSet    = $80001010 : LONGINT;
                     ySizeFactor  = $80001011 : LONGINT; | ???
                     spaceWidth   = $80002012 : LONGINT;
                     isFixed      = $80002013 : LONGBOOL;
                     serifFlag    = $80001014 : LONGBOOL;
                     stemWeight   = $80001015 : LONGINT;   | 128
                     slantStyle   = $80001016 : LONGINT;
                     horizStyle   = $80001017 : LONGINT;   | 128
                     spaceFactor  = $80002018 : Fixed;
                     dummy        = $80002019 : LONGINT;

                     availSizes   = $80009020 : POINTER TO ARRAY OF CARDINAL;
                     specCount    = $80001100 : LONGINT;
                   END;
```

```
ReqGlyphTags   = TAGS OF StdTags;
                   deviceDPI    = $80000001 : POINTER TO Coord;
                   dotSize      = $80000002 : POINTER TO Coord;
                   pointHeight  = $80000008 : POINTER TO Fixed;
                   setFactor    = $80000009 : POINTER TO Fixed;
                   shearSin     = $8000000A : POINTER TO Fixed;
                   shearCos     = $8000000B : POINTER TO Fixed;
                   rotateSin    = $8000000C : POINTER TO Fixed;
                   rotateCos    = $8000000D : POINTER TO Fixed;
                   emboldenX    = $8000000E : POINTER TO Fixed;
                   emboldenY    = $8000000F : POINTER TO Fixed;

                   pointSize    = $80000010 : POINTER TO LONGINT;
                                              | in 16tel Punkt = 1/72"

                   glyphCode    = $80000011 : POINTER TO LONGCHAR;
                   glyphCode2   = $80000012 : POINTER TO LONGCHAR;
                   glyphWidth   = $80000013 : POINTER TO Fixed;

                   tagPath      = $80008014 : POINTER TO SysStringPtr;
                   tagList      = $80008015 : POINTER TO ANYPTR;

                   glyphMap     = $80008020 : POINTER TO GlyphMapPtr;
                   widthList    = $80008021 : POINTER TO ANYPTR;
                   kernPair     = $80008022 : POINTER TO ANYPTR;
                   designKernPair         : POINTER TO ANYPTR;


                   fileIdent    = $80001001 : POINTER TO LONGINT;
                   engine       = $80009002 : POINTER TO SysStringPtr;
                   family       = $80009003 : POINTER TO SysStringPtr;
                   boldName     = $8000A005 : POINTER TO SysStringPtr;
                   italicName   = $8000A006 : POINTER TO SysStringPtr;
                   biName       = $8000A007 : POINTER TO SysStringPtr;
                   symbolSet    = $80001010 : POINTER TO LONGINT;
                   ySizeFactor  = $80001011 : POINTER TO LONGINT; | ???
                   spaceWidth   = $80002012 : POINTER TO LONGINT;
                   isFixed      = $80002013 : POINTER TO LONGBOOL;
                   serifFlag    = $80001014 : POINTER TO LONGBOOL;
                   stemWeight   = $80001015 : POINTER TO LONGINT; | 128
                   slantStyle   = $80001016 : POINTER TO LONGINT;
                   horizStyle   = $80001017 : POINTER TO LONGINT; | 128
                   spaceFactor  = $80002018 : POINTER TO Fixed;

                   availSizes   = $80009020 : POINTER TO POINTER TO
                                                 ARRAY OF CARDINAL;
                   specCount    = $80001100 : POINTER TO LONGINT;
                 END;

GlyphTagList   = ARRAY OF GlyphTags;
ReqGlyphTagList= ARRAY OF ReqGlyphTags;
GlyphTagListPtr= POINTER TO GlyphTagList;
ReqGlyphTagListPtr= POINTER TO GlyphTagList;

GlyphEnginePtr = POINTER TO GlyphEngine;
```



```
DEFINITION MODULE GlyphRes = Resources.ResHandles(GlyphEnginePtr);

TYPE
  GlyphEngine     = RECORD OF GlyphRes.ResHandle;
                    END;

  GlyphErrors     = (failure  = -1,
                     ok,
                     badTag,unknownTag,badData,noMemory,
                     noFace,noGlyph,badGlyph,noShear,
                     noRotate,tooSmall,
                     unknownGlyph,

                     makeMeLong = $10000);

VAR BulletBase : LibraryPtr;


LIBRARY BulletBase BY -36
  PROCEDURE CloseEngine(engineHandle IN A0  : GlyphEnginePtr);

LIBRARY BulletBase BY -60
  PROCEDURE GetGlyphMap(    engineHandle IN A0  : GlyphEnginePtr;
                            glyphCode    IN D0  : LONGCHAR;
                        VAR glyph        IN A1  : GlyphMapPtr):GlyphErrors;

LIBRARY BulletBase BY -48
  PROCEDURE ObtainInfoA(engineHandle IN A0 : GlyphEnginePtr;
                        tagList      IN A1 : ReqGlyphTagListPtr):GlyphErrors;

LIBRARY BulletBase BY -48
  PROCEDURE ObtainInfo(    engineHandle IN A0 : GlyphEnginePtr;
                       VAR tags         IN A1 : LIST OF GlyphTags):GlyphErrors;

LIBRARY BulletBase BY -30
  PROCEDURE OpenEngine():GlyphEnginePtr;

LIBRARY BulletBase BY -54
  PROCEDURE ReleaseInfoA(engineHandle IN A0 : GlyphEnginePtr;
                         tagList      IN A1 : GlyphTagListPtr):GlyphErrors;

LIBRARY BulletBase BY -54
  PROCEDURE ReleaseInfo(engineHandle IN A0 : GlyphEnginePtr;
                        tags         IN A1 : LIST OF GlyphTags):GlyphErrors;

LIBRARY BulletBase BY -42
  PROCEDURE SetInfoA(engineHandle IN A0 : GlyphEnginePtr;
                     tagList      IN A1 : GlyphTagListPtr):GlyphErrors;
```

```
LIBRARY BulletBase BY -42
  PROCEDURE SetInfo(engineHandle IN A0 : GlyphEnginePtr;
                    tagList      IN A1 : LIST OF GlyphTags):GlyphErrors;

GROUP
  All    = Fixed,GlyphMapPtr,GlyphTags,GlyphTagList,
           ReqGlyphTags,ReqGlyphTagList,GlyphEnginePtr,
           GlyphErrors,Coord,GlyphWidthNodePtr,

           CloseEngine,GetGlyphMap,ObtainInfoA,ObtainInfo,
           OpenEngine,ReleaseInfoA,ReleaseInfo,SetInfoA,SetInfo;

END Bullet.
```


8.6 CiaaResource

```
DEFINITION MODULE CiaaResource;

(* $A- *)
FROM Exec     IMPORT InterruptPtr,Resource,ResourcePtr,Interrupt,IntVector;
FROM Hardware IMPORT CiaIcrFlagSet,IntFlagSet,CiaIcrFlags;
FROM System   IMPORT Regs;

VAR
  CiaaBase  : ResourcePtr;

LIBRARY  CiaaBase   BY -18
  PROCEDURE AbleIcr(mask IN D0  : CiaIcrFlagSet): CiaIcrFlagSet;

LIBRARY  CiaaBase   BY -6
  PROCEDURE AddICRVector(icrBit    IN D0 : CiaIcrFlags;
                         interrupt IN A1 : InterruptPtr):InterruptPtr;

LIBRARY  CiaaBase   BY -12
  PROCEDURE RemICRVector(icrBit    IN D0  : CiaIcrFlags;
                         interrupt IN A1  : InterruptPtr);

LIBRARY  CiaaBase   BY -24
  PROCEDURE SetICR(mask IN D0  : CiaIcrFlagSet):CiaIcrFlagSet;


GROUP
  All  = CiaaBase,AbleIcr,AddICRVector,RemICRVector,SetICR;

END CiaaResource.
```

8.7 CiabResource

```
DEFINITION MODULE CiabResource;
(* $A- *)
FROM Exec     IMPORT InterruptPtr,Resource,ResourcePtr,Interrupt,IntVector;
FROM Hardware IMPORT CiaIcrFlagSet,IntFlagSet,CiaIcrFlags;
FROM System   IMPORT Regs;

TYPE

VAR
  CiabBase  : ResourcePtr;

LIBRARY  CiabBase   BY -18
  PROCEDURE AbleIcr(mask IN D0  : CiaIcrFlagSet): CiaIcrFlagSet;

LIBRARY  CiabBase   BY -6
  PROCEDURE AddICRVector(icrBit    IN D0 : CiaIcrFlags;
                         interrupt IN A1 : InterruptPtr):InterruptPtr;

LIBRARY  CiabBase   BY -12
  PROCEDURE RemICRVector(icrBit    IN D0  : CiaIcrFlags;
                        interrupt IN A1  : InterruptPtr);

LIBRARY  CiabBase   BY -24
  PROCEDURE SetICR(mask IN D0  : CiaIcrFlagSet):CiaIcrFlagSet;

GROUP
  All  = CiabBase,AbleIcr,AddICRVector,RemICRVector,SetICR;

END CiabResource.
```


8.8 Clipboard

```
DEFINITION MODULE Clipboard;

(* $A- *)

FROM T_Exec     IMPORT nonstdVAL,NoFreeSignal,OpenError,
                       Node,Message,IOStdReq,DevicePtr,IOFlagSet,
                       IOCommand,IOReturn,UnitPtr;
FROM Resources  IMPORT ContextPtr;

CONST
  primaryClip = 0;

  post            = IOCommand( nonstdVAL + 0 );
  currentReadId   = IOCommand( nonstdVAL + 1 );
  currentWrite    = IOCommand( nonstdVAL + 2 );
  changeHook      = IOCommand( nonstdVAL + 3 );

  obsoleteId      = IOReturn( 1 );

TYPE
  ClipboardUnitPartialPtr = POINTER TO ClipboardUnitPartial;
  ClipboardUnitPartial    = RECORD OF Node
                              unitNum : LONGCARD
                            END;

  IOClipboardPtr          = POINTER TO IOClipboard;
  IOClipboard             = RECORD OF IOStdReq
                              clipID : LONGINT;
                            END;

  SatisfyMsgPtr           =  POINTER TO SatisfyMsg;
  SatisfyMsg              =  RECORD OF Message
                               unit   : CARDINAL;
                               clipID : LONGINT;
                             END;

  ClipHookMsgPtr          = POINTER TO ClipHookMsg;
  ClipHookMsg             = RECORD
                              type      : LONGCARD;
                              changeCmd : LONGINT;
                              clipID    : LONGCARD;
                            END;

PROCEDURE OpenClipboard( unit    : CARDINAL  := primaryClip;
                         context : ContextPtr:=NIL): IOClipboardPtr;

PROCEDURE CloseClipboard( VAR request : IOClipboardPtr );
```

```
GROUP
  All     = post,                     currentReadId,
            currentWrite,             obsoleteId,
            ClipboardUnitPartial,     ClipboardUnitPartialPtr,
            IOClipboard,              IOClipboardPtr,
            primaryClip,              SatisfyMsg,
            SatisfyMsgPtr,
            OpenClipboard,            CloseClipboard,
            T_Exec.ExecIOGrp;

END Clipboard.
```


8.9 Commodities

```
DEFINITION MODULE Commodities;

FROM Exec   IMPORT LibraryPtr, MsgPortPtr, TaskPtr, TaskSignals;

FROM Input  IMPORT Qualifiers, QualifierSet, InputEventPtr;
FROM System IMPORT SysStringPtr, Regs, PROC;

TYPE
  KeyMapPtr       = DEFERRED POINTER KeyMap.KeyMapPtr;

  CxBrokerErr     = (ok, sysErr, dup, version);

CONST
  NBVersion       = 5;

TYPE
  UniqueFlags     = (unique, notify, dummy = 15);
  UniqueFlagSet   = SET OF UniqueFlags;

  NewBrokerFlags  = (showHide = 2, dummy = 15);
  NewBrokerFlagSet= SET OF NewBrokerFlags;

  NewBrokerPtr    = POINTER TO NewBroker;
  NewBroker       = RECORD
                      version           :  SHORTINT;       | := NBVersion
                      name              :  SysStringPtr;
                      title             :  SysStringPtr;
                      descr             :  SysStringPtr;
                      unique            :  UniqueFlagSet;
                      flags             :  NewBrokerFlagSet;
                      pri               :  SHORTINT;
                      port              :  MsgPortPtr;
                      reservedChannel   :  INTEGER;
                    END;


  CxObjPtr        = HIDDEN;
  CxMsgPtr        = POINTER TO RECORD OF Exec.Message END;

  PFL             = PROCEDURE():LONGINT;
```

```
  CxObjType       =  (invalid, filter, typefilter,
                      sender, signal, translate,
                      broker, debug, custom, zero);

 | invalid    : not a valid object (probably null)
 | filter     : input event messages only
 | typefilter : filter on message type
 | send       : sends a message
 | signal     : sends a signal
 | translate  : translates IE into chain
 | broker     : application representative
 | debug      : dumps kprintf to serial port
 | custom     : application provids function
 | zero       : system terminator node

  CxMsg           = (unique=4, iEvent, command);
  CxMsgTypes      = SET OF CxMsg;

  CxMsgCommands = (disable   = 15,  | disable yourself
                   enable    = 17,  | enable yourself
                   appear    = 19,  | open your window
                   disappear = 21,  | close the window
                   kill      = 23,  | terminate yourself
                   unique    = 25,  | someone tried to create
                                    | a unique broker
                   listChg   = 27); | someone changed broker list

  BrokerCommandErr= (noMem = -3, noPort, noBroker, ok);

  CxObjErrs       = (isNull, nullAttach, badFilter, badType);
  CxObjErrSet     = SET OF CxObjErrs;

  | isNull     : you called CxError(NULL)
  | nullAttach : someone attached NULL to my list
  | badFilter  : a bad filter description was given
  | badType    : unmatched type-specific operation


CONST
  IXVersion       =  2;

TYPE
  QualSameId      = (shift, caps, alt);
  QualSameSet     = SET OF QualSameId;
```



```
  InputXpressionPtr
                  = POINTER TO InputXpression;
  InputXpression  = RECORD
                      version    :  SHORTCARD; | must be set to IX_VERSION
                      class      :  SHORTCARD; | class must match exactly
                      code       :  CARDINAL;
                      codeMask   :  CARDINAL;  | normally used for UPCODE
                      qualifier  :  CARDINAL;
                      qualMask   :  CARDINAL;
                      qualSame   :  QualSameSet;| synonyms in qualifier
                    END;
  IX              = InputXpression;
  IXPtr           = InputXpressionPtr;

  IXErr           = (noDescription = -2, tokensAfterEnd, ok);

CONST
  shiftMask       = QualifierSet:{lShift,rShift};
  capsMask        = shiftMask + QualifierSet:{capsLock};
  altMask         = QualifierSet:{lAlt, rAlt};
  normalquals     = QualifierSet:{lShift..leftButton};


VAR
  CxBase :  LibraryPtr;


LIBRARY CxBase BY -30
  PROCEDURE CreateCxObj(type IN D0: CxObjType;
                        arg1 IN A0: ANYPTR;
                        arg2 IN A1: ANYPTR) : CxObjPtr;

LIBRARY CxBase BY -36
  PROCEDURE CxBroker(REF nb    IN A0: NewBroker;
                     VAR error IN D0: CxBrokerErr) : CxObjPtr;

LIBRARY CxBase BY -42
  PROCEDURE ActivateCxObj(co   IN A0: CxObjPtr;
                          true IN D0: LONGBOOL): LONGBOOL;
        (* TRUE: active, FALSE inactive *)


LIBRARY CxBase BY -48
  PROCEDURE DeleteCxObj(co IN A0: CxObjPtr);

LIBRARY CxBase BY -54
  PROCEDURE DeleteCxObjAll(co IN A0: CxObjPtr);

LIBRARY CxBase BY -60
  PROCEDURE GetCxObjType(co IN A0: CxObjPtr) : CxObjType;

LIBRARY CxBase BY -66
  PROCEDURE CxObjError(co IN A0: CxObjPtr): CxObjErrSet;
```

```
LIBRARY CxBase BY -72
  PROCEDURE ClearCxObjError(co IN A0: CxObjPtr);

LIBRARY CxBase BY -78
  PROCEDURE SetCxObjPri(co  IN A0: CxObjPtr;
                        pri IN D0: LONGINT) (*** Should be SHORTINT***);

LIBRARY CxBase BY -84
  PROCEDURE AttachCxObj(headObj IN A0: CxObjPtr;
                        co      IN A1: CxObjPtr);

LIBRARY CxBase BY -90
  PROCEDURE EnqueueCxObj(headObj IN A0: CxObjPtr;
                         co      IN A1: CxObjPtr);

LIBRARY CxBase BY -96
  PROCEDURE InsertCxObj(headObj IN A0: CxObjPtr;
                        co      IN A1: CxObjPtr;
                        pred    IN A2: CxObjPtr);

LIBRARY CxBase BY -102
  PROCEDURE RemoveCxObj(co IN A0: CxObjPtr);


LIBRARY CxBase BY -114
  PROCEDURE SetTranslate(translator IN A0: CxObjPtr;
                         events     IN A1: InputEventPtr);

LIBRARY CxBase BY -120
  PROCEDURE SetFilter(    filter IN A0: CxObjPtr;
                      REF text   IN A1: STRING);

LIBRARY CxBase BY -126
  PROCEDURE SetFilterIX(filter IN A0: CxObjPtr;
                        ix     IN A1: IXPtr);

LIBRARY CxBase BY -132
  PROCEDURE ParseIX(REF description IN A0: STRING;
                        ix          IN A1: IXPtr) :IXErr;

LIBRARY CxBase BY -138
  PROCEDURE CxMsgType(cxm IN A0: CxMsgPtr) : CxMsgTypes;

LIBRARY CxBase BY -144
  PROCEDURE CxMsgData(cxm IN A0: CxMsgPtr) : ANYPTR;

LIBRARY CxBase BY -150
   PROCEDURE CxMsgID(cxm IN A0: CxMsgPtr) : LONGINT;

LIBRARY CxBase BY -150
  PROCEDURE CxMsgCommand(cxm IN A0: CxMsgPtr) : CxMsgCommands;
```



```
LIBRARY CxBase BY -156
  PROCEDURE DivertCxMsg(cxm     IN A0: CxMsgPtr;
                        headobj IN A1: CxObjPtr;
                        ret     IN A2: CxObjPtr);

LIBRARY CxBase BY -162
  PROCEDURE RouteCxMsg(cxm IN A0: CxMsgPtr;
                       co  IN A1: CxObjPtr);

LIBRARY CxBase BY -168
  PROCEDURE DisposeCxMsg(cxm IN A0: CxMsgPtr);

LIBRARY CxBase BY -174
  PROCEDURE InvertKeyMap(ansicode IN D0: LONGCARD;
                         event    IN A0: InputEventPtr;
                         km       IN A1: KeyMapPtr) : BOOLEAN;

LIBRARY CxBase BY -180
  PROCEDURE AddIEvents(events IN A0: InputEventPtr);


PROCEDURE CxFilter(REF des : STRING):CxObjPtr;

PROCEDURE CxSender(port : MsgPortPtr;id : LONGINT):CxObjPtr;

PROCEDURE CxSignal(task : TaskPtr;sig : TaskSignals):CxObjPtr;

PROCEDURE CxTranslate(ie : InputEventPtr):CxObjPtr;

PROCEDURE CxCustom(action : PROC;id : LONGINT):CxObjPtr;


END Commodities.
```

8.10 Console

```
DEFINITION MODULE Console;
(* $A- *)
FROM T_Exec      IMPORT IOCommand,nonstdVAL,IOStdReqPtr,LibraryPtr,
                        IOStdReq;
FROM Input       IMPORT InputEventPtr;
FROM KeyMap      IMPORT KeyMapPtr;
FROM System      IMPORT Regs;
FROM Resources   IMPORT ContextPtr;

CONST
  askKeyMap          = IOCommand(nonstdVAL+0);
  setKeyMap          = IOCommand(nonstdVAL+1);
  askDefaultKeyMap   = IOCommand(nonstdVAL+2);
  setDefaultKeyMap   = IOCommand(nonstdVAL+3);

  primary            = 0;
  bold               = 1;
  italic             = 3;
  underScore         = 4;
  negative           = 7;

|(V36)
  normal             = 22;
  notItalic          = 23;
  notUnderscore      = 24;
  positive           = 27;

  black              = 30;
  red                = 31;
  green              = 32;
  yellow             = 33;
  blue               = 34;
  magenta            = 35;
  cyan               = 36;
  white              = 37;
  default            = 39;
  blackBg            = 40;
  redBg              = 41;
  greenBg            = 42;
  yellowBg           = 43;
  blueBg             = 44;
  magentaBg          = 45;
  cyanBg             = 46;
  whiteBg            = 47;
  defaultBg          = 49;

  clr0               = 30;
  clr1               = 31;
  clr2               = 32;
  clr3               = 33;
  clr4               = 34;
  clr5               = 35;
  clr6               = 36;
```



```
  clr7               = 37;
  clr0Bg             = 40;
  clr1Bg             = 41;
  clr2Bg             = 42;
  clr3Bg             = 43;
  clr4Bg             = 44;
  clr5Bg             = 45;
  clr6Bg             = 46;
  clr7Bg             = 47;

  dsrCpr             = 6;

  ctcHSetTab         = 0;
  ctcHClrTab         = 2;
  ctcHClrTabsAll     = 5;
  tbcHClrTab         = 0;
  tbcHClrTabsAll     = 3;

  mLnm               = 20;
  mAsm               = ">1";
  mAwm               = "?7";


|Definition der verschiedenen Console Units für OpenDevice()
CONST
  library  = -1;
  standard = 0;

|(V36)
  charMap  = 1;
  snipMap  = 3;

|Neue Flags für OpenDevice() - (V37)
TYPE
  ConFlags   = (nodrawOnNewsize,dummy=31);
  ConFlagSet = SET OF ConFlags;

VAR
  ConsoleBase  : LibraryPtr;

PROCEDURE OpenConsole(window  : ANYPTR;
                      unit    : LONGCARD   := standard;
                      flags   := ConFlagSet:{};
                      context : ContextPtr := NIL): IOStdReqPtr;

PROCEDURE CloseConsole(VAR request : IOStdReqPtr);

LIBRARY ConsoleBase BY -42
  PROCEDURE CDInputHandler(events  IN A0  : InputEventPtr;
                           device1 IN A1  : ANYPTR):InputEventPtr;
```

```
LIBRARY ConsoleBase BY -48
  PROCEDURE RawKeyConvert(events  IN A0  : InputEventPtr;
                          buffer  IN A1  : ANYPTR;
                          length  IN D1  : LONGINT;
                          keyMap  IN A2 : KeyMapPtr):LONGINT;

GROUP
  UnitGrp       =  library,standard,charMap,snipMap,ConFlags,
                   ConFlagSet;
  CommandGrp    =  askKeyMap,setKeyMap,askDefaultKeyMap,
                   setDefaultKeyMap;
  StyleGrp      =  primary,bold,italic,underScore,negative,normal,
                   notItalic,notUnderscore,positive;
  ColorGrp      =  black,red,green,yellow,blue,magenta,cyan,white,
                   default,blackBg,redBg,greenBg,yellowBg,blueBg,
                   magentaBg,cyanBg,whiteBg,defaultBg;
  ClrGrp        =  clr0,clr1,clr2,clr3,clr4,clr5,clr6,clr7,clr0Bg,
                   clr1Bg,clr2Bg,clr3Bg,clr4Bg,clr5Bg,clr6Bg,clr7Bg;
  ConstGrp      =  dsrCpr,ctcHSetTab,ctcHClrTab,ctcHClrTabsAll,
                   tbcHClrTab,tbcHClrTabsAll,mLnm,mAsm,mAwm;

  ProcGrp       =  T_Exec.ExecIOGrp,OpenConsole,CloseConsole,
                   CDInputHandler,RawKeyConvert;

  All           =  UnitGrp,CommandGrp,StyleGrp,ColorGrp,ClrGrp,
                   ConstGrp,ProcGrp;


END Console.
```


8.11 ConUnit

```
DEFINITION MODULE ConUnit;
(* $A- *)
FROM Console      IMPORT mLnm;
FROM Exec         IMPORT MsgPort;
FROM Graphics     IMPORT DrawModeSet,TextFontPtr;
FROM Input        IMPORT classMax;
FROM Intuition    IMPORT WindowPtr;
FROM KeyMap AS km IMPORT KeyMap;


|Bemerkung: Die V36/37 C CONU_xx- sowie die CONFLAG_xx-Definitionen sind in
|Console.def!

CONST
  pmbAsm   = mLnm+1;
  pmbAwm   = pmbAsm+1;
  maxTabs  = 80;

TYPE
  ConUnit     = RECORD
                  mp            : MsgPort;
                  window        : WindowPtr;
                  xCP           : INTEGER;
                  yCP           : INTEGER;
                  xMax          : INTEGER;
                  yMax          : INTEGER;
                  xRSize        : INTEGER;
                  yRSize        : INTEGER;
                  xROrigin      : INTEGER;
                  yROrigin      : INTEGER;
                  xRExtant      : INTEGER;
                  yRExtant      : INTEGER;
                  xMinShrink    : INTEGER;
                  yMinShrink    : INTEGER;
                  xcCP          : INTEGER;
                  ycCP          : INTEGER;
                  keyMap        : KeyMap;
                  tabStops      : ARRAY [0..maxTabs-1] OF CARDINAL;
                  mask          : SHORTCARD;
                  fgPen         : SHORTCARD;
                  bgPen         : SHORTCARD;
                  aolPen        : SHORTCARD;
                  drawMode      : DrawModeSet;
                  areaPtSz      : SHORTCARD;
                  areaPtrn      : ANYPTR;
                  minTerms      : ARRAY [0..7] OF SHORTCARD;
                  font          : TextFontPtr;
                  algoStyle     : SHORTCARD;
                  txFlags       : SHORTCARD;
                  txHeight      : CARDINAL;
                  txWidth       : CARDINAL;
                  txBaseLine    : CARDINAL;
                  txSpacing     : CARDINAL;
```

```
                 modes          : ARRAY [0..(pmbAwm+7) DIV 8-1] OF SHORTCARD;
                 rawEvents      : ARRAY [0..(classMax+7) DIV 8-1] OF SHORTCARD;
               END;
   ConUnitPtr = POINTER TO ConUnit

GROUP
  All  = pmbAsm,pmbAwm,maxTabs,ConUnit,ConUnitPtr;

END ConUnit.
```


8.12 DiskFont

```
DEFINITION MODULE DiskFont;
(* $A- *)
FROM System   IMPORT BITSET,BPTR,SysStringPtr,Regs;
FROM Exec     IMPORT Node,LibraryPtr;
FROM Graphics IMPORT FontGrp,TextAttr,TTextAttr;

CONST
  maxFontPath           = 256;
  maxFontName           = 32;
  fchId                 = $0F00;
  dfhId                 = $0F80;

TYPE
  AvailFontTypes        = (memory,disk,scaled,af15=15);
  AvailFontTypeSet      = SET OF AvailFontTypes;
  FontContents          = RECORD
                            fileName : ARRAY [maxFontPath] OF CHAR;
                            ySize    : CARDINAL;
                            style    : FontStyleSet;
                            flags    : FontFlagSet;
                          END;
  TFontContents         = RECORD
                            fileName : ARRAY [maxFontPath-2] OF CHAR;
                            tagCount : CARDINAL;
                            ySize    : CARDINAL;
                            style    : FontStyleSet;
                            flags    : FontFlagSet;
                          END;

  FontContentsHeaderPtr = POINTER TO FontContentsHeader;
  FontContentsHeader    = RECORD
                            fileId     : CARDINAL;
                            numEntries : CARDINAL;
                          END;


  DiskFontHeaderPtr     = POINTER TO DiskFontHeader;
  DiskFontHeader        = RECORD OF Node
                            fileId    : CARDINAL;
                            revision  : CARDINAL;
                            segment   : BPTR;
                            name      : ARRAY [maxFontName] OF CHAR;
                            tf        : TextFont;
                          END;

  AvailFont             = RECORD
                            type      : AvailFontTypeSet;
                            attr      : TextAttr;
                          END;
  TAvailFont            = RECORD
                            type      : AvailFontTypeSet;
                            attr      : TTextAttr;
                          END;
```

```
  AvailFontHeader       = RECORD
                            numEntries  : CARDINAL;
                          END;

  AvailFontHeaderPtr    = POINTER TO AvailFontHeader;

VAR
  DiskFontBase   : LibraryPtr;

LIBRARY DiskFontBase BY -36
  PROCEDURE AvailFonts(buffer IN A0 : ANYPTR;
                       len    IN D0 : LONGINT;
                       modus  IN D1 : BITSET):LONGINT;

LIBRARY DiskFontBase BY -48
  PROCEDURE DisposeFontContents(header IN A1 : FontContentsHeaderPtr);

LIBRARY DiskFontBase BY -42
  PROCEDURE NewFontContents(fontsLock IN A0 : ANYPTR;
                            name      IN A1 : SysStringPtr):FontContentsHeaderPtr;

LIBRARY DiskFontBase BY -30
  PROCEDURE OpenDiskFont(REF tAttr IN A0 : TextAttr):TextFontPtr;

LIBRARY DiskFontBase BY -54
  PROCEDURE NewScaledDiskFont(    font  IN A0 : TextFontPtr;
                              REF tAttr IN A1 : TextAttr):DiskFontHeaderPtr;

GROUP
  DiskFontGrp     =  AvailFonts,OpenDiskFont,Graphics.FontGrp,
                     AvailFontTypes,AvailFontTypeSet,DiskFontHeaderPtr;

  All             =  DiskFontGrp,maxFontPath,maxFontName,fchId,dfhId,
                     FontContents,FontContentsHeader,FontContentsHeaderPtr,
                     DiskFontHeader,AvailFont,AvailFontHeader,
                     AvailFontHeaderPtr,DisposeFontContents,NewFontContents,
                     NewScaledDiskFont;

END DiskFont.
```



```
DEFINITION MODULE DiskResource;
(* $A- *)

|2.0-Version 01.10.1992

FROM Exec       IMPORT Interrupt,Resource, ResourcePtr,LibraryPtr,List,
                       Message,TaskPtr;
FROM System     IMPORT Regs;


TYPE
  DiscResourceUnit       = RECORD OF Message;
                             discBlock : Interrupt;
                             discSync  : Interrupt;
                             index     : Interrupt
                           END;
  DiscResourceUnitPtr     = POINTER TO DiscResourceUnit;

  DiscResourceFlags       = (alloc0,alloc1,alloc2,alloc3,drf4,drf5,drf6,
                             active);
  DiscResourceFlagSet     = SET OF DiscResourceFlags;

  DiscResource            = RECORD OF Resource;
                              current     : DiscResourceUnitPtr;
                              flags       : DiscResourceFlagSet;
                              pad         : SHORTCARD;
                              sysLib      : LibraryPtr;
                              ciaResource : ResourcePtr;
                              unitId      : ARRAY [alloc0..alloc3] OF LONGCARD;

                              discBlock   : Interrupt;
                              discSync    : Interrupt;
                              index       : Interrupt;
                              task        : TaskPtr;
                            END;
  DiscResourcePtr         = POINTER TO DiscResource;

CONST
  |Hardware magic
  dskDmaOff     = $4000;

  |Resource-spezifische Kommandos


  |Drive Types
  amiga         = 0;
  drt37422D2S   = $55555555;
  empty         = $FFFFFFFF;
  drt150RPM     = $AAAAAAAA;
```

```
VAR
  DiskBase   : DiscResourcePtr;

LIBRARY  DiskBase  BY -6
  PROCEDURE AllocUnit(unitNum  IN D0  : LONGINT):BOOLEAN;

LIBRARY  DiskBase  BY -12
  PROCEDURE FreeUnit(unitNum  IN D0  : LONGINT):LONGINT;

LIBRARY  DiskBase  BY -18
  PROCEDURE GetUnit(unitPointer IN A1 : DiscResourceUnitPtr):DiscResourceUnitPtr;

LIBRARY  DiskBase  BY -30
  PROCEDURE GetUnitID(unitNum  IN D0  : LONGINT):LONGCARD;

LIBRARY  DiskBase  BY -24
  PROCEDURE GiveUnit;

GROUP
  All     = DiscResourceUnit,DiscResourceUnitPtr,DiscResourceFlags,
            DiscResourceFlagSet,DiscResource,DiscResourcePtr,dskDmaOff,
            amiga,drt37422D2S,empty,AllocUnit,FreeUnit,GetUnit,GetUnitID,
            GiveUnit;

END DiskResource.
```


8.13 Dos

```
(* $A- *)
DEFINITION MODULE Dos;

FROM System     IMPORT  BPTR,LONGSET,BITSET,PROC,SysStringPtr,Regs;
FROM Resources  IMPORT  ResHandles;
FROM Exec       IMPORT  Library,Message,MessagePtr,MsgPort,MsgPortPtr,
                        Task,LibraryPtr,MinList,MinNode,TaskPtr,Node,
                        SignalSemaphore;
FROM Utility    IMPORT  tagUser,TagListPtr,HookPtr,StdTags;


TYPE
  BSTR             =  BCPLPTR TO CHAR;
  FileHandlePtr    =  BCPLPTR TO FileHandle;
  FileHandleCPtr   =  POINTER TO FileHandle;
  FileLockPtr      =  BCPLPTR TO FileLock;
  FileLockCPtr     =  POINTER TO FileLock;
  ProcessId        =  POINTER TO MsgPort;
  DeviceListPtr    =  BCPLPTR TO DeviceList;

DEFINITION MODULE FileHandleRes = ResHandles( FileHandleCPtr );

DEFINITION MODULE FileLockRes = ResHandles( FileLockCPtr );

CONST
  lenDatString     =  16;

TYPE
  DatFlags         =  (Subst,Future);
  DatFlagSet       =  SET OF DatFlags;
  DateFormats      =  (DOS,INT,USA,CDN,MAX=CDN);

  DatePtr          =  POINTER TO Date;
  Date             =  RECORD
                        days       : LONGCARD;
                        minute     : LONGCARD;
                        tick       : LONGCARD;
                      END;

  DateTimePtr      =  POINTER TO DateTime;
  DateTime         =  RECORD OF Date;
                        format     :  DateFormats;
                        flags      :  DatFlagSet;
                        strDay     :  SysStringPtr;
                        strDate    :  SysStringPtr;
                        strTime    :  SysStringPtr;
                      END;


TYPE
  OpenMode         =  ( readWrite=1004,readOnly,
                        oldFile=1005,newFile,
                        makemelong=1000000 );
```

```
  SeekMode         =  ( beginning=-1,current,end,
                        makemelong=1000000 );


  LockMode         =  (sharedLock=-2,
                       accessRead=sharedLock,
                       exclusiveLock=-1,
                       accessWrite=exclusiveLock,
                       makemelong=1000000 );

  FileLock         =  RECORD OF FileLockRes.ResHandle
                        link            :  FileLockPtr;
                        key             :  LONGINT;
                        access          :  LockMode;
                        task            :  ProcessId;
                        volume          :  DeviceListPtr;
                      END;


CONST
  ticksPerSecond   =  50;

TYPE
  DiskStatus       =  (writeProtect=80,validating,validated,
                       makemelong=1000000);
  DiskType         =  (
                       noDiskPresent   =  -1,
                       unreadableDisk  =  $42414400, (* "BAD\0" *)
                       dosDisk         =  $444F5300, (* "DOS\0" *)
                       ffsDisk         =  $444F5301, (* "DOS\1" *)
                       notReallyDos    =  $4E444F53, (* "NDOS"  *)
                       kickstartDisk   =  $4B49434B, (* "KICK"  *)
                       msdosDisk       =  $4D534400  (* "MSD\0" *)
                      );

  InfoDataPtr      =  POINTER TO InfoData;
  InfoData         =  RECORD
                        numSoftErrors  : LONGINT;
                        unitNumber     : LONGINT;
                        diskState      : DiskStatus;
                        numBlocks,
                        numBlocksUsed  : LONGINT;
                        bytesPerBlock  : LONGINT;
                        diskType       : DiskType;
                        volumeNode     : DeviceListPtr;
                        inUse          : LONGBOOL;
                      END;
```



```
| Fehlermeldungen von IoErr()

TYPE
  IoErrors         = ( noFreeStore=103,
                       taskTableFull=105,      badTemplate=114,
                       badNumber,              requiredArgMissing,
                       keyNeedsArg,            tooManyArgs,
                       unmatchedQuotes,        lineTooLong,
                       fileNotObject,          invalidResidentLibrary,
                       noDefaultDir=201,       objectInUse,
                       objectExists,           dirNotFound,
                       objectNotFound,         badStreamName,
                       objectTooLarge,         actionNotKnown=209,
                       invalidComponentName,   invalidLock,
                       objectWrongType,        diskNotValidated,
                       diskWriteProtected,     renameAcrossDevices,
                       directoryNotEmpty,      tooManyLevels,
                       deviceNotMounted,       seekError,
                       commentTooBig,          diskFull,
                       deleteProtected,        writeProtected,
                       readProtected,          notADosDisk,
                       noDisk,                 noMoreEntries=232,
                       isSoftLink,             objectLinked,
                       badHunk,                notImplemented,
                       recordNotLocked=240,    lockCollision,
                       lockTimeout,            unlockError,
                       bufferOverflow=303,     break,
                       notExecutable,          makemelong=1000000 );

| Rueckgabewerte der Amiga DOS Kommandos
  ReturnCode       = (ok=0,warn=5,error=10,fail=20,makemelong=1000000 );


|Break Signale
TYPE
  BreakSignals     = (ctrlC=12,ctrlD,ctrlE,ctrlF);


|Rueckgabewerte von SameLock()
  SameLockType     = (lockDifferent=-1,lockSame,
                      lockSameHandler,makemelong=1000000 );


|Typen fuer ChangeMode()
  ChangeModeType   = ( changeLock,changeFH,makemelong=1000000 );


|Werte fuer MakeLink()
  LinkType         = ( linkHard,
                       linkSoft,              |Noch nicht voll unterstuetzt
                       makemelong=1000000 );

|Rueckgabewerte von ReadItem()
  ReadItemType     = ( itemEqual=-2,itemError,itemNothing,
                       itemUnquoted,itemQuoted,makemelong=1000000 );
```

```
|For SetMode
  ScreenType      = ( con,raw,makemelong=1000000);

|Typen fuer Alloc-/FreeDosObject()
TYPE
  DosObject       = ( fileHandle,exAllControl,FIB,stdPkt,
                      CLI,rArgs,makemelong=1000000 );


  ProtectionFlags =  (delete,execute,writeProt,readProt,archive,
                      pure,script,hidden,pf31=31);
  ProtectionFlagSet=  SET OF ProtectionFlags;

  EntryType       =  (linkFile=-4,file,root=1,userDir,
                      softLink,linkDir,makemelong=1000000);

  FileInfoBlockPtr =  POINTER TO FileInfoBlock;
  FileInfoBlock    =  RECORD
                        diskKey      : LONGINT;
                        dirEntryType : EntryType;
                        fileName     : ARRAY [0..107] OF CHAR;
                        protection   : ProtectionFlagSet;
                        entryType    : LONGINT;
                        size         : LONGINT;
                        numBlocks    : LONGINT;
                        date         : Date;
                        comment      : ARRAY [0..79] OF CHAR;
                        reserved     : ARRAY [0..35] OF CHAR;
                      END;

  ExAllType       =  (name=1,type,size,protection,date,comment);

  ExAllDataPtr    =  POINTER TO ExAllData;
  ExAllData       =  RECORD
                       next       :  ExAllDataPtr;
                       name       :  SysStringPtr;
                       type       :  EntryType;
                       size       :  LONGCARD;
                       prot       :  ProtectionFlagSet;
                       date       :  Date;
                       comment    :  SysStringPtr;
                     END;

  ExAllControlPtr =  POINTER TO ExAllControl;
  ExAllControl    =  RECORD
                       entries    :  LONGCARD;
                       lastkey    :  LONGCARD;
                       matchString:  SysStringPtr;
                       matchFunc  :  HookPtr;
                     END;

| Konstanten aus dosasl.h
  AChainFlags     =  (patternBit,examinedBit,completed,allBit,single);
  AChainFlagSet   =  SET OF AChainFlags;
```



```
  AChainPtr        =  POINTER TO AChain;
  AChain           =  RECORD
                        child      :  AChainPtr;
                        parent     :  AChainPtr;
                        lock       :  BPTR;
                        info       :  FileInfoBlock;
                        flags      :  AChainFlagSet;
                        string     :  CHAR; | nobody knows
                      END;

  AnchorPathFlags =  (doWild,itsWild,doDir,didDir,noMemErr,doDot,
                       dirChanged);
  AnchorPathFlagSet = SET OF AnchorPathFlags;

  AnchorPathPtr    =  POINTER TO AnchorPath;
  AnchorPath       =  RECORD
                        base       :  AChainPtr;
                        last       :  AChainPtr;
                        breakBits  :  LONGINT;
                        foundBreak :  LONGINT;
                        flags      :  AnchorPathFlagSet;
                        reserved   :  SHORTINT;
                        strLen     :  INTEGER;  | set to 0
                        info       :  FileInfoBlock;
                      END;

  FullAnchorPathPtr = POINTER TO FullAnchorPath;
  FullAnchorPath   = RECORD OF AnchorPath;
                       buffer : ARRAY [256] OF CHAR; |set strlen to 256
                     END;

CONST
  complexBit       =  1;
  examineBit       =  2;


TYPE
  ProcessFlags     =  (freeSegList, freeCurrDir, freeCLI, closeInput,
                       closeOutput, freeArgs,prf31=31);
  ProcessFlagSet   =  SET OF ProcessFlags;

  CommandLineInterfacePtr  =  BCPLPTR TO CommandLineInterface;

  ProcessPtr       =  POINTER TO Process;
  Process          =  RECORD OF Task
                        msgPort            :  MsgPort;
                        pad                :  CARDINAL;
                        segList            :  BPTR;
                        stackSize          :  LONGINT;
                        globVec            :  ANYPTR;
                        taskNum            :  LONGINT;
                        stackBase          :  BPTR;
                        result2            :  LONGINT;
                        currentDir         :  FileLockPtr;
```

```
                    cis                 :  FileHandlePtr;
                    cos                 :  FileHandlePtr;
                    consoleTask         :  ProcessId;
                    fileSystemTask      :  ProcessId;
                    cli                 :  CommandLineInterfacePtr;
                    returnAddr          :  ANYPTR;
                    pktWait             :  ANYPTR;
                    windowPtr           :  ANYPTR;
                    homeDir             :  BPTR;
                    flags               :  ProcessFlagSet;
                    exitCode            :  PROC;
                    exitData            :  LONGINT;
                    arguments           :  ANYPTR;
                    localVars           :  MinList;
                    shellPrivate        :  LONGCARD;
                    ces                 :  BPTR;
                  END;


TYPE
  | you MUST always specify at least read or write
  LockDosFlags    =  (read,write,devices,volumes,
                      assigns,entry,delete,ldf31=31);
  LockDosFlagSet  =  SET OF LockDosFlags;

CONST
  ldfAll          =  LockDosFlagSet:{devices,volumes,assigns};


| Error Report Typen fuer ErrorReport()
TYPE
  ErrorReportType =  (stream,task,lock,volume,insert);

CONST
   abortDiskError =  296;
   abortBusy      =  288;

TYPE
  RunType         =  (systemAsynch = -3,system,execute);


  Hunks           =  (hunkUnit=999, hunkName, hunkCode, hunkData,
                      hunkBss, hunkReloc32, hunkReloc16, hunkReloc8,
                      hunkExt, hunkSymbol, hunkDebug, hunkEnd,
                      hunkHeader, hunkOverlay=1013, hunkBreak,
                      hunkDRel32, hunkDRel16, hunkDRel8, hunkLib,
                      hunkIndex);
  External        =  (extSymb, extDef, extAbs, extRes, extRef32=129,
                      extCommon, extRef16, extRef8, extDExt32,
                      extDExt16, extDExt8);
```



```
CONST
  notifyClass     =  $40000000; | attention, these constants may be
                                | changeged
  notifyCode      =  $1234;     | next time, so don't use them

TYPE
  NotifyRequestFlags      =  (sendMessage,sendSignal,waitReply=3,
                              notifyInitial,nrf16=16,magic=31);
  NotifyRequestFlagSet    =  SET OF NotifyRequestFlags;

CONST
  handlerFlags    =  NotifyRequestFlagSet:{nrf16..magic};
  nrHandlerFlags  =  $FFFF0000;

TYPE
  NotifyRequestPtr=  POINTER TO NotifyRequest;
  NotifyRequest   =  RECORD
                       name        :  SysStringPtr;
                       fullName    :  SysStringPtr;
                       userData    :  LONGCARD;
                       flags       :  NotifyRequestFlagSet;
                       IF KEY : NotifyRequestFlags
                         OF message THEN port      : MsgPortPtr
                         OF signal  THEN task      : TaskPtr;
                                         signalNum : Exec.TaskSignals;
                                         pad       : ARRAY[0..2] OF SHORTCARD

                       reserved    :  ARRAY[0..3] OF LONGCARD;
                       msgCount    :  LONGCARD;
                       handler     :  MsgPortPtr;
                     END;

  NotifyMessagePtr=  POINTER TO NotifyMessage;
  NotifyMessage   =  RECORD OF Message
                       class       :  LONGCARD;
                       code        :  CARDINAL;
                       nReq        :  NotifyRequestPtr;
                       doNotTouch,
                       doNotTouch2:  LONGCARD;
                     END;


TYPE
  RDArgsFlags     =  (stdIn, noAlloc, noPrompt,rdaf31=31);
  RDArgsFlagSet   =  SET OF RDArgsFlags;

CONST
  maxTemplateItems=  100;
  maxMultiArgs    =  128;
```

```
TYPE
  CSourcePtr      =  POINTER TO CSource;
  CSource         =  RECORD
                       buffer     :  SysStringPtr;
                       length     :  LONGINT;
                       curChr     :  LONGINT;
                     END;

  RDArgsPtr       =  POINTER TO RDArgs;
  RDArgs          =  RECORD OF CSource
                       daList     :  LONGINT;
                       buffer     :  SysStringPtr;
                       bufSiz     :  LONGINT;
                       extHelp    :  SysStringPtr;
                       flags      :  RDArgsFlagSet;
                     END;

  RecordMode      = (recExclusive,recExclusiveImmed,recShared,
                     recSharedImmed,makemelong=1000000);

  RecordLockPtr   =  POINTER TO RecordLock;
  RecordLock      =  RECORD
                       fh         :  FileHandlePtr;
                       offset     :  LONGCARD;
                       length     :  LONGCARD;
                       mode       :  RecordMode;
                     END;
| for SetVBuf
  BuffType        =  (bufLine,bufFull,bufNone,makemelong=1000000);



TYPE
  PathInfoPtr     =  BCPLPTR TO PathInfo;

  SystemTags      =  TAGS OF StdTags
                       dummy           =  tagUser + 32;
                       input           :  FileHandlePtr;
                       output          :  FileHandlePtr;
                       asynch          :  ANYPTR;
                       userShell       :  ANYPTR;
                       customShell     :  SysStringPtr;
                     END;
  SystemTagList   =  ARRAY OF SystemTags;
  SystemTagListPtr=  POINTER TO SystemTagList;

  WindowPtr       =  DEFERRED POINTER Intuition.WindowPtr;;

  NewProcTags     =  TAGS OF StdTags;
                       dummy            =  tagUser + 1000;
                       segList          :  BPTR;
                       freeSegList      :  BPTR;
                       entry            :  BPTR;
                       input            :  FileHandlePtr;
```



```
                     output          :  FileHandlePtr;
                     closeInput      :  LONGBOOL;
                     closeOutput     :  LONGBOOL;
                     error           :  FileHandlePtr;
                     closeError      :  LONGBOOL;
                     currentDir      :  FileLockPtr;
                     stackSize       :  LONGINT;
                     name            :  BSTR;
                     priority        :  LONGINT;
                     consoleTask     :  ProcessId;
                     windowPtr       :  WindowPtr;
                     homeDir         :  FileLockPtr;
                     copyVars        :  LONGBOOL;
                     cli             :  LONGBOOL;
                     path            :  PathInfoPtr;
                     commandName     :  BSTR;
                     arguments       :  SysStringPtr;
                     notifyOnDeath   :  LONGBOOL;
                     synchronous     :  LONGBOOL;
                     exitCode        :  ANYPTR;
                     exitData        :  ANYPTR;
                   END;
NewProcTagListPtr= POINTER TO NewProcTagList;
NewProcTagList   = ARRAY OF NewProcTags;


DosObjectTags    =  TAGS OF StdTags;
                     dummy            =  tagUser + 2000;
                     fHMode           :  LONGINT;
                     dirLen           :  LONGINT;
                     commNameLen      :  LONGINT;
                     commFileLen      :  LONGINT;
                     promptLen        :  LONGINT;
                   END;
DosObjectTagListPtr= POINTER TO DosObjectTagList;
DosObjectTagList = ARRAY OF DosObjectTags;



PathInfo         =  RECORD
                     nextPath  : PathInfoPtr;
                     lock      : FileLockPtr;
                   END;


FileHandle       =  RECORD OF FileHandleRes.ResHandle
                     link           : MessagePtr;
                     port           : MsgPortPtr;
                     type           : ProcessId;
                     buf            : LONGINT;
                     pos            : LONGINT;
                     end            : LONGINT;
                     func1,
                     func2,
```

```
                  func3          : PROC;
                  arg1           : LONGINT;
                  arg2           : LONGINT;
                END;

  DosCommands     = ( nil=0,              startup=0,
                    getBlock=2,         setMap=4,
                    die,                event,
                    currentVolume,      locateObject,
                    renameDisk,         freelock=15,
                    deleteObject,       renameObject,
                    moreCache,          copyDir,
                    waitChar,           setProtect,
                    createDir,          examineObject,
                    examineNext,        diskInfo,
                    info,               flush,
                    setComment,         parent,
                    timer,              inhibit,
                    diskType,           diskChange,
                    setDate,            sameLock=40,
                    screenMode=994,     changeSignal,
                    readReturn=1001,    writeReturn,
                    findUpDate=1004,    findInput,
                    findOutput,         actionEnd,
                    seek,               format=1020,
                    makeLink,           truncate,
                    writeLock,          readLink,
                    fhFromLock=1026,    isFileSystem,
                    changeMode,         copyDirFh=1030,
                    parentFh,           examineAll=1033,
                    examineFh,          lockRecord=2008,
                    freeRecord,         addNotify=4097,
                    removeNotify,       read=LONGINT("R"),
                    write=LONGINT("W"), makemelong=1000000 );


TYPE
  DosPacketPtr    =  POINTER TO DosPacket;
  DosPacket       =  RECORD
                       link          : MessagePtr;
                       port          : MsgPortPtr;
                       type          : DosCommands;
                       res1,
                       res2,
                       arg1,
                       arg2,
                       arg3,
                       arg4,
                       arg5,
                       arg6,
                       arg7          : LONGINT;
                     END;
```



```
  StandardPacketPtr= POINTER TO StandardPacket;
  StandardPacket   =  RECORD OF Message
                        pkt  : DosPacket;
                      END;

  RootNodePtr      =  POINTER TO RootNode;

  ErrorStringPtr   =  POINTER TO ErrorString;
  ErrorString      =  RECORD
                        nums                :  ANYPTR;
                        strings             :  SysStringPtr;
                      END;

  DosLibraryPtr    =  POINTER TO DosLibrary;
  DosLibrary       =  RECORD OF Library
                        root                :  RootNodePtr;
                        gv                  :  ANYPTR;
                        a2,
                        a5,
                        a6                  :  LONGINT;
                        errors              :  ErrorStringPtr;
                        timeReq             :  ANYPTR;
                        utilityBase         :  LibraryPtr;
                        intuitionBase       :  LibraryPtr;
                      END;

  TaskArray        =  RECORD
                        maxCli              :  LONGINT;
                        cli                 :  ARRAY [1..99] OF ProcessId;
                      END;

  DosInfoPtr       =  BCPLPTR TO DosInfo;
  TaskArrayPtr     =  BCPLPTR TO TaskArray;
  DosEnvecPtr      =  BCPLPTR TO DosEnvec;
  FileSysStartupMsgPtr = BCPLPTR TO FileSysStartupMsg;

CONST
  | special values for ResidentSegment.usecount
  cmdSystem        =  -1;
  cmdInternal      =  -2;
  cmdDisabled      =  -999;

TYPE
  ResidentSegmentPtr       =  BCPLPTR TO ResidentSegment;
  ResidentSegment  =  RECORD
                        next        :  ResidentSegmentPtr;
                        usecount    :  LONGINT;
                        segment     :  BPTR;
                        name        :  ARRAY [32] OF CHAR
                      END;


  RootNodeFlags    =  (private1=1,wildStar=24,rnf31=31);
  RootNodeFlagSet  =  SET OF RootNodeFlags;
```

```
RootNode        = RECORD
                    taskArray          :  TaskArrayPtr;
                    consoleSegment     :  BPTR;
                    time               :  Date;
                    restartSeg         :  BPTR;
                    info               :  DosInfoPtr;
                    fileHandlerSegment :  BPTR;
                    cliList            :  MinList;
                    bootProc           :  MsgPortPtr;
                    shellSegment       :  BPTR;
                    flags              :  RootNodeFlagSet;
                  END;

MsgPortFieldPtr = POINTER TO MsgPortField;
MsgPortField    = ARRAY OF MsgPortPtr;
CliProcList     = RECORD OF MinNode
                    first              :  LONGINT;
                    array              :  MsgPortFieldPtr;
                  END;

DosInfo         = RECORD
                    mcName              :  BSTR;
                    devInfo             :  DeviceListPtr;
                    devices             :  BPTR;
                    handlers            :  BPTR;
                    netHand             :  ResidentSegmentPtr;
                    devLock,
                    entryLock,
                    deleteLock          : SignalSemaphore;
                  END;

CommandLineInterface  =  RECORD
                    result2       :  LONGINT;
                    setName       :  BSTR;
                    commandDir    :  PathInfoPtr;
                    returnCode    :  LONGINT;
                    commandName   :  BSTR;
                    failLevel     :  LONGINT;
                    prompt        :  BSTR;
                    standardInput :  FileHandlePtr;
                    currentInput  :  FileHandlePtr;
                    commandFile   :  BSTR;
                    interactive   :  LONGINT;
                    background    :  LONGINT;
                    currentOutput :  FileHandlePtr;
                    defaultStack  :  LONGINT;
                    standardOutput:  FileHandlePtr;
                    module        :  BPTR;
                  END;

AssignListPtr   = POINTER TO AssignList;
AssignList      = RECORD
                    next       :  AssignListPtr;
                    lock       :  FileHandlePtr;
```



```
                  END;

DeviceListType  = ( device,directory,volume,late,nonBinding,
                   private=-1,makemelong=1000000 );

DeviceList      = RECORD
                    next          :  DeviceListPtr;
                    type          :  DeviceListType;
                    task          :  ProcessId;
                    lock          :  FileLockPtr;
                    IF KEY : DeviceListType
                      OF device,
                        directory   THEN
                                      handler    :  BSTR;
                                      stackSize  :  LONGINT;
                                      priority   :  LONGINT;
                                      startup    :  FileSysStartupMsgPtr;
                                      segList    :  BPTR;
                                      globeVec   :  BPTR
                      OF volume     THEN
                                      volumeDate :  Date;
                                      lockList   :  FileLockPtr;
                                      diskType   :  LONGINT;
                      OF assign     THEN
                                      assignName :  SysStringPtr;
                                      assignList :  AssignListPtr
                    END;
                    name  :  BSTR;
                  END;
DosList         = DeviceList;
DosListPtr      = DeviceListPtr;
DeviceNode      = DeviceList;
DeviceNodePtr   = POINTER TO DeviceNode;


DevProcFlags    = (unlock,assign,dvp31=31);
DevProcFlagSet  = SET OF DevProcFlags;

DevProcPtr      = POINTER TO DevProc;
DevProc         = RECORD
                    port   :  MsgPortPtr;
                    lock   :  FileLockPtr;
                    flags  :  DevProcFlagSet;
                    devNode:  DosListPtr;
                  END;

DosEnvec        = RECORD
                    tableSize        :  LONGCARD;
                    sizeBlock        :  LONGCARD;
                    secOrg           :  LONGCARD;
                    surfaces         :  LONGCARD;
                    sectorsPerBlock  :  LONGCARD;
                    blocksPerTrack   :  LONGCARD;
                    reserved         :  LONGCARD;
```

```
                    preAlloc          :  LONGCARD;
                    interleave        :  LONGCARD;
                    lowCyl            :  LONGCARD;
                    highCyl           :  LONGCARD;
                    numBuffers        :  LONGCARD;
                    bufMemType        :  LONGCARD;
                    maxTransfers      :  LONGCARD;
                    mask              :  LONGSET;
                    bootPri           :  LONGINT;
                    dosType           :  ARRAY [0..3] OF CHAR;
                    baud              :  LONGCARD;
                    control           :  LONGCARD;
                    bootBlocks        :  LONGCARD;
                  END;

  FileSysStartupMsg= RECORD
                    unit     : LONGCARD;
                    device   : BSTR;
                    environ  : DosEnvecPtr;
                    flags    : LONGSET
                  END;
  ArgArrayPtr     =  POINTER TO ArgArray;
  ArgArray        =  ARRAY OF SysStringPtr;



  GetVarFlags     =  (var,alias,ignore=7,globalOnly,localOnly,
                     binaryVar,gvf31=31);
  GetVarFlagSet   =  SET OF GetVarFlags;

  LocalVarPtr     =  POINTER TO LocalVar;
  LocalVar        =  RECORD OF MinNode
                    type       :  GetVarFlags;
                    pri        :  SHORTINT;
                    name       :  SysStringPtr;
                    flags      :  BITSET;
                    value      :  SysStringPtr;
                    len        :  LONGCARD;
                  END;

GROUP
  DosErrorGrp     =    IoErrors;

VAR DosBase : DosLibraryPtr;

|------------------------------------------------------------------------
|  1. Ein- und Ausgabe
|------------------------------------------------------------------------

LIBRARY DosBase BY -36
  PROCEDURE Close( file IN D1 : FileHandlePtr):LONGBOOL;
```



```
LIBRARY DosBase BY -30
  PROCEDURE Open( REF name       IN D1 : STRING;
                      accessMode IN D2 : OpenMode):FileHandlePtr;


LIBRARY DosBase BY -42
  PROCEDURE Read( file       IN D1 : FileHandlePtr;
                  buffer     IN D2 : ANYPTR;
                  length     IN D3 : LONGINT ): LONGINT;

LIBRARY DosBase BY -66
  PROCEDURE Seek( file       IN D1 : FileHandlePtr;
                  position   IN D2 : LONGINT;
                  mode       IN D3 : SeekMode ): LONGINT;

LIBRARY DosBase BY -48
  PROCEDURE Write( file       IN D1 : FileHandlePtr;
                   buffer     IN D2 : ANYPTR;
                   length     IN D3 : LONGINT ): LONGINT;

LIBRARY DosBase BY -294
  PROCEDURE SelectInput( fh IN D1 : FileHandlePtr ): FileHandlePtr;

LIBRARY DosBase BY -300
  PROCEDURE SelectOutput( fh IN D1 : FileHandlePtr ): FileHandlePtr;

LIBRARY DosBase BY -306
  PROCEDURE FGetC( fh IN D1 : FileHandlePtr ): CHAR;

LIBRARY DosBase BY -312
  PROCEDURE FPutC( fh IN D1 : FileHandlePtr;
                   ch IN D2 : CHAR ): CHAR;

LIBRARY DosBase BY -318
  PROCEDURE UnGetC( fh        IN D1 : FileHandlePtr;
                    character IN D2 : CHAR);

LIBRARY DosBase BY -324
  PROCEDURE FRead( fh       IN D1 : FileHandlePtr;
                   buf      IN D2 : ANYPTR;
                   blocklen IN D3 : LONGCARD;
                   number   IN D4 : LONGCARD ): LONGCARD;

LIBRARY DosBase BY -330
  PROCEDURE FWrite( fh       IN D1 : FileHandlePtr;
                    buf      IN D2 : ANYPTR;
                    blocklen IN D3 : LONGCARD;
                    number   IN D4 : LONGCARD ): LONGCARD;

LIBRARY DosBase BY -336
  PROCEDURE FGets( fh     IN D1 : FileHandlePtr;
                   buf    IN D2 : ANYPTR;
                   buflen IN D3 : LONGCARD ): ANYPTR;
```

```
LIBRARY DosBase BY -342
  PROCEDURE FPuts(     fh  IN D1 : FileHandlePtr;
                   REF str IN D2 : STRING );

LIBRARY DosBase BY -348
  PROCEDURE VFWritef(     fh       IN D1 : FileHandlePtr;
                      REF format   IN D2 : STRING;
                          argarray IN D3 : ArgArrayPtr );

LIBRARY DosBase BY -354
  PROCEDURE VFPrintf(     fh       IN D1 : FileHandlePtr;
                      REF format   IN D2 : STRING;
                          argarray IN D3 : ArgArrayPtr );

LIBRARY DosBase BY -360
  PROCEDURE Flush( fh IN D1 : FileHandlePtr ): LONGBOOL;

LIBRARY DosBase BY -366
  PROCEDURE SetVBuf( fh   IN D1 : FileHandlePtr;
                     buff IN D2 : ANYPTR;
                     type IN D3 : BuffType;
                     size IN D4 : LONGINT ): LONGINT;

LIBRARY DosBase BY -942
  PROCEDURE WriteChars( REF buf    IN D1 : STRING;
                            buflen IN D2 : LONGINT ): LONGINT;

LIBRARY DosBase BY -948
  PROCEDURE PutStr( REF str IN D1: STRING ): LONGINT;

LIBRARY DosBase BY -954
  PROCEDURE VPrintf( REF format   IN D1 : STRING;
                         argarray IN D2 : ArgArrayPtr ): LONGINT;


GROUP
  DosIOGrp   = Open,Close,Read,FileHandlePtr,FileHandlePtr,Seek,Write,
               OpenMode,SeekMode,SelectInput,SelectOutput,FGetC,FPutC,
               UnGetC,FRead,FWrite,FGets,FPuts,VFWritef,VFPrintf,Flush,
               SetVBuf, WriteChars, PutStr, VPrintf, ArgArrayPtr;

|------------------------------------------------------------------------
|  2. Dateiverwaltung
|------------------------------------------------------------------------

LIBRARY DosBase BY -120
  PROCEDURE CreateDir( REF Name IN D1 : STRING ): FileLockPtr;

LIBRARY DosBase BY -126
  PROCEDURE CurrentDir( Lock IN D1 : FileLockPtr ): FileLockPtr;

LIBRARY DosBase BY -72
  PROCEDURE DeleteFile( REF Name IN D1 : STRING ): LONGBOOL;
```



```
LIBRARY DosBase BY -102
  PROCEDURE Examine(Lock      IN D1 : FileLockPtr;
                    InfoBlock IN D2 : FileInfoBlockPtr): LONGBOOL;

LIBRARY DosBase BY -108
  PROCEDURE ExNext( Lock      IN D1 : FileLockPtr;
                    InfoBlock IN D2 : FileInfoBlockPtr ): LONGBOOL;

LIBRARY DosBase BY -114
  PROCEDURE Info( Lock           IN D1 : FileLockPtr;
                  ParameterBlock IN D2 : InfoDataPtr ): LONGBOOL;

LIBRARY DosBase BY -210
  PROCEDURE ParentDir( Lock IN D1 : FileLockPtr ): FileLockPtr;

LIBRARY DosBase BY -78
  PROCEDURE Rename( REF OldName IN D1,
                        NewName IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -180
  PROCEDURE SetComment( REF Name    IN D1,
                            Comment IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -186
  PROCEDURE SetProtection( REF Name IN D1 : STRING;
                               Mask IN D2 : ProtectionFlagSet ): LONGBOOL;

LIBRARY DosBase BY -96
  PROCEDURE DupLock( lock IN D1 : FileLockPtr ): FileLockPtr;

LIBRARY DosBase BY -54
  PROCEDURE Input(): FileHandlePtr;

LIBRARY DosBase BY -132
  PROCEDURE IoErr(): IoErrors;

LIBRARY DosBase BY -216
  PROCEDURE IsInteractive( File IN D1 : FileHandlePtr ): LONGBOOL;

LIBRARY DosBase BY -84
  PROCEDURE Lock( REF name       IN D1 : STRING;
                      accessMode IN D2 : LockMode ): FileLockPtr;

LIBRARY DosBase BY -60
  PROCEDURE Output(): FileHandlePtr;

LIBRARY DosBase BY -90
  PROCEDURE Unlock( Lock IN D1 : FileLockPtr );

LIBRARY DosBase BY -270
  PROCEDURE LockRecord( fh      IN D1 : FileHandlePtr;
                        offset  IN D2 : LONGCARD;
                        length  IN D3 : LONGCARD;
                        mode    IN D4 : LONGCARD;
                        timeout IN D5 : LONGCARD ): LONGBOOL;
```

```
LIBRARY DosBase BY -276
  PROCEDURE LockRecords( recArray IN D1 : RecordLockPtr;
                         timeout  IN D2 : LONGCARD ): LONGBOOL;

LIBRARY DosBase BY -282
  PROCEDURE UnLockRecord( fh     IN D1 : FileHandlePtr;
                          offset IN D2 : LONGCARD;
                          length IN D3 : LONGCARD ): LONGBOOL;

LIBRARY DosBase BY -288
  PROCEDURE UnLockRecords(recArray IN D1 : RecordLockPtr):LONGBOOL;

LIBRARY DosBase BY -372
  PROCEDURE DupLockFromFH( fh IN D1: FileHandlePtr ): FileLockPtr;

LIBRARY DosBase BY -378
  PROCEDURE OpenFromLock( lock IN D1: FileLockPtr ): FileLockPtr;

LIBRARY DosBase BY -384
  PROCEDURE ParentOfFH( fh IN D1: FileHandlePtr ): FileHandlePtr;

LIBRARY DosBase BY -390
  PROCEDURE ExamineFH( fh  IN D1 : FileHandlePtr;
                       fib IN D2 : FileInfoBlockPtr ): LONGBOOL;

LIBRARY DosBase BY -396
  PROCEDURE SetFileDate( REF name IN D1 : STRING;
                             date IN D2 : DatePtr ): LONGBOOL;

PROCEDURE NameFromLock(     lock   : FileLockPtr;
                        VAR buffer : STRING;
                            len    : LONGINT:=0): LONGBOOL;


PROCEDURE NameFromFH(     fh     : FileHandlePtr;
                      VAR buffer : STRING;
                          len    : LONGINT:=0 ): LONGBOOL;

PROCEDURE SplitName( REF name      : STRING;
                         seperator : CHAR;
                     VAR buf       : STRING;
                         oldpos    : INTEGER;
                         size      : LONGINT:=0): INTEGER;

LIBRARY DosBase BY -420
  PROCEDURE SameLock( lock1 IN D1 : FileLockPtr;
                      lock2 IN D2 : FileLockPtr ): SameLockType;

LIBRARY DosBase BY -426
  PROCEDURE SetMode( fh   IN D1 : FileHandlePtr;
                     mode IN D2 : ScreenType ): LONGBOOL;
```



```
LIBRARY DosBase BY -432
  PROCEDURE ExAll( lock    IN D1 : FileLockPtr;
                   buffer  IN D2 : ANYPTR;
                   size    IN D3 : LONGINT;
                   type    IN D4 : ExAllType;
                   control IN D5 : ExAllControlPtr ): LONGBOOL;;

PROCEDURE ReadLink(     port   : MsgPortPtr;
                        lock   : FileLockPtr;
                    REF path   : STRING;
                    VAR buffer : STRING;
                        size   : LONGCARD:=0): LONGBOOL;

LIBRARY DosBase BY -444
  PROCEDURE MakeLink( REF name IN D1 : STRING;
                          dest IN D2 : LONGINT;
                          soft IN D3 : LinkType): LONGBOOL;

LIBRARY DosBase BY -450
  PROCEDURE ChangeMode( type    IN D1 : ChangeModeType;
                        fh      IN D2 : FileHandlePtr;
                        newmode IN D3 : LockMode): LONGBOOL;

LIBRARY DosBase BY -456
  PROCEDURE SetFileSize( fh   IN D1 : FileHandlePtr;
                         pos  IN D2 : LONGINT;
                         mode IN D3 : SeekMode): LONGINT;

LIBRARY DosBase BY -462
  PROCEDURE SetIoErr( result IN D1 : LONGINT ): LONGINT;

LIBRARY DosBase BY -708
  PROCEDURE IsFileSystem( REF name IN D1 : STRING ): LONGBOOL;

LIBRARY DosBase BY -714
  PROCEDURE Format( REF filesystem IN D1 : STRING;
                    REF volumename IN D2 : STRING;
                        dostype    IN D3 : LONGCARD ): LONGBOOL;

LIBRARY DosBase BY -720
  PROCEDURE Relabel( REF drive   IN D1 : STRING;
                     REF newname IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -726
  PROCEDURE Inhibit( REF name  IN D1 : STRING;
                         onoff IN D2 : LONGINT ): LONGBOOL;

LIBRARY DosBase BY -732
  PROCEDURE AddBuffers( REF name   IN D1 : STRING;
                            number IN D2 : LONGINT ): LONGBOOL;

LIBRARY DosBase BY -738
  PROCEDURE CompareDates( date1 IN D1 : DatePtr;
                          date2 IN D2 : DatePtr ): LONGINT;
```

```
LIBRARY DosBase BY -984
  PROCEDURE SameDevice( lock1 IN D1 : FileLockPtr;
                        lock2 IN D2 : FileLockPtr ): LONGBOOL;

GROUP
  DosFileGrp  =  CreateDir,FileLockPtr, CurrentDir,DeleteFile,
                 Examine,FileInfoBlockPtr,FileInfoBlock,Info,
                 InfoDataPtr,InfoData,ParentDir,Rename,SetComment,
                 SetComment,SetProtection,ProtectionFlagSet,
                 ProtectionFlags,DupLock,Input,IoErr,IsInteractive,
                 Lock,Lock,Output,Unlock,LockMode,LockRecord,DatePtr,
                 LockRecords, UnLockRecord, UnLockRecords, DupLockFromFH,
                 OpenFromLock, ParentOfFH, ExamineFH, SetFileDate,
                 NameFromLock,NameFromFH, SplitName, SameLock, SetMode,
                 ExAll, ReadLink,MakeLink, ChangeMode, SetFileSize,
                 SetIoErr, IsFileSystem,Format, Relabel, Inhibit,
                 AddBuffers, CompareDates, SameDevice,ExAllControl,
                 ExAllType,ScreenType,LinkType,LockMode;

|------------------------------------------------------------------------
|  4. Prozeßverwaltung
|------------------------------------------------------------------------

LIBRARY DosBase BY -138
  PROCEDURE CreateProc( REF Name       IN D1 : STRING;
                            Pri        IN D2 : LONGINT;
                            Segment    IN D3 : BPTR;
                            StackSize  IN D4 : LONGINT):ProcessId;

LIBRARY DosBase BY -192
  PROCEDURE DateStamp( VAR Date IN D1 : Date );

| Due to a bug in the kickstart ROM, we'll fix the problem ourselves
| |LIBRARY DosBase BY-198
PROCEDURE Delay( ticks IN D1  : LONGCARD );

LIBRARY DosBase BY -198
  PROCEDURE oldDelay( ticks IN D1 : LONGCARD );

LIBRARY DosBase BY -174
  PROCEDURE DeviceProc( REF Name IN D1 : STRING ): ProcessId;

LIBRARY DosBase BY -144
  PROCEDURE Exit( ReturnCode IN D1 : LONGINT );

LIBRARY DosBase BY -204
  PROCEDURE WaitForChar( File    IN D1 : FileHandlePtr;
                         Timeout IN D2 : LONGINT ): LONGBOOL;

PROCEDURE Fault(     code   : IoErrors;
                 REF header : STRING;
                 VAR buffer : STRING;
                     len    : LONGINT:=0 ): LONGBOOL;
```



```
LIBRARY DosBase BY -474
  PROCEDURE PrintFault(     code   IN D1 : IoErrors;
                        REF header IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -480
  PROCEDURE ErrorReport( code   IN D1 : LONGINT;
                         type   IN D2 : LONGINT;
                         arg1   IN D3 : LONGCARD;
                         device IN D4 : MsgPortPtr ): LONGBOOL;

LIBRARY DosBase BY -492
  PROCEDURE Cli(): CommandLineInterfacePtr;

LIBRARY DosBase BY -498
  PROCEDURE CreateNewProc( tags IN D1 : LIST OF NewProcTags): ProcessPtr;

LIBRARY DosBase BY -498
  PROCEDURE CreateNewProcList( tags IN D1: NewProcTagListPtr ): ProcessPtr;

LIBRARY DosBase BY -504
  PROCEDURE RunCommand(     seg     IN D1 : BPTR;
                            stack   IN D2 : LONGCARD;
                        REF param   IN D3 : STRING;
                            paramlen IN D4 : LONGCARD ): LONGINT;

LIBRARY DosBase BY -510
  PROCEDURE GetConsoleTask(): MsgPortPtr;

LIBRARY DosBase BY -516
  PROCEDURE SetConsoleTask( task IN D1 : MsgPortPtr ): MsgPortPtr;

LIBRARY DosBase BY -522
  PROCEDURE GetFileSysTask(): MsgPortPtr;

LIBRARY DosBase BY -528
  PROCEDURE SetFileSysTask( task IN D1 : MsgPortPtr ): MsgPortPtr;

LIBRARY DosBase BY -534
  PROCEDURE GetArgStr(): SysStringPtr;

LIBRARY DosBase BY -540
  PROCEDURE SetArgStr( REF string IN D1 : STRING ): SysStringPtr;

LIBRARY DosBase BY -546
  PROCEDURE FindCliProc( num IN D1 : LONGINT ): ProcessPtr;

LIBRARY DosBase BY -552
  PROCEDURE MaxCli(): LONGINT;

LIBRARY DosBase BY -558
  PROCEDURE SetCurrentDirName( REF name IN D1 : STRING ): LONGBOOL;

PROCEDURE GetCurrentDirName( VAR buf : STRING;
                                 len : LONGINT:=0): LONGBOOL;
```

```
LIBRARY DosBase BY -570
  PROCEDURE SetProgramName( REF name IN D1 : STRING ): LONGBOOL;

PROCEDURE GetProgramName( VAR buf : STRING;
                              len : LONGINT:=0): LONGBOOL;

LIBRARY DosBase BY -582
  PROCEDURE SetPrompt( REF name IN D1 : STRING ): LONGBOOL;

PROCEDURE GetPrompt( VAR buf : STRING;
                         len : LONGINT:=0): LONGBOOL;

LIBRARY DosBase BY -594
  PROCEDURE SetProgramDir( lock IN D1 : FileLockPtr ): FileLockPtr;

LIBRARY DosBase BY -600
  PROCEDURE GetProgramDir(): FileLockPtr;

LIBRARY DosBase BY -606
  PROCEDURE CallSystemTags( REF command IN D1 : STRING;
                                tags    IN D2 : LIST OF SystemTags):LONGINT;
LIBRARY DosBase BY -606
  PROCEDURE CallSystemTagList(REF command IN D1 : STRING;
                                  tags    IN D2 : SystemTagListPtr):LONGINT;

LIBRARY DosBase BY -612
  PROCEDURE AssignLock( REF name IN D1 : STRING;
                            lock IN D2 : FileLockPtr ): LONGBOOL;

LIBRARY DosBase BY -618
  PROCEDURE AssignLate( REF name IN D1 : STRING;
                        REF path IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -624
  PROCEDURE AssignPath( REF name IN D1 : STRING;
                        REF path IN D2 : STRING ): LONGBOOL;

LIBRARY DosBase BY -630
  PROCEDURE AssignAdd( REF name IN D1 : STRING;
                           lock IN D2 : FileLockPtr ): LONGBOOL;

LIBRARY DosBase BY -636
  PROCEDURE RemAssignList( REF name IN D1 : STRING;
                               lock IN D2 : FileLockPtr ): LONGBOOL;

LIBRARY DosBase BY -642
  PROCEDURE GetDeviceProc( REF name IN D1 : STRING;
                               dp   IN D2 : DevProcPtr ): DevProcPtr;

LIBRARY DosBase BY -648
  PROCEDURE FreeDeviceProc( dp IN D1 : DevProcPtr);

LIBRARY DosBase BY -744
  PROCEDURE DateToStr(VAR datetime IN D1 : DateTime ): LONGBOOL;
```



```
LIBRARY DosBase BY -750
  PROCEDURE StrToDate(VAR datetime IN D1 : DateTime ): LONGBOOL;


GROUP
  DosProcessGrp  = CreateProc,CreateProc,DateStamp,Delay,DeviceProc,Exit,
                   WaitForChar,ticksPerSecond,Date,Process,ProcessPtr,
                   ProcessId, Fault, PrintFault, ErrorReport, Cli,
                   CreateNewProc,RunCommand, GetConsoleTask,
                   SetConsoleTask, SetFileSysTask,GetArgStr, SetArgStr,
                   FindCliProc,MaxCli,SetCurrentDirName,GetCurrentDirName,
                   SetProgramName, GetProgramName, SetPrompt,GetPrompt,
                   SystemTagListPtr, AssignLock, AssignLate, AssignPath,
                   AssignAdd, RemAssignList, GetDeviceProc, FreeDeviceProc,
                   StrToDate,DateToStr,FileLockPtr,DevProcPtr,MsgPortPtr,
                   IoErrors,CommandLineInterfacePtr,SystemTags,
                   SystemTagList,NewProcTags,NewProcTags;


|------------------------------------------------------------------------
|  5. Programme laden
|------------------------------------------------------------------------

LIBRARY DosBase BY -222
  PROCEDURE Execute( REF CommandString IN D1 : STRING;
                         Input         IN D2 : FileHandlePtr;
                         Output        IN D3 : FileHandlePtr ): LONGBOOL;

LIBRARY DosBase BY -150
  PROCEDURE LoadSeg( REF Name IN D1 : STRING ): BPTR;

LIBRARY DosBase BY -156
  PROCEDURE UnLoadSeg( Segment IN D1 : BPTR );

LIBRARY DosBase BY -756
  PROCEDURE InternalLoadSeg( fh        IN D0 : FileHandlePtr;
                             table     IN A0 : BPTR;
                             funcarray IN A1 : ANYPTR;
                             stack     IN A2 : ANYPTR ): BPTR;

LIBRARY DosBase BY -762
  PROCEDURE InternalUnLoadSeg(seglist  IN D1 : BPTR;
                              freefunc IN A1 : ANYPTR):LONGBOOL;

LIBRARY DosBase BY -768
  PROCEDURE NewLoadSeg( REF file IN D1 : STRING;
                            tags IN D2 : TagListPtr ): BPTR;

LIBRARY DosBase BY -774
  PROCEDURE AddSegment( REF name   IN D1 : STRING;
                            seg    IN D2 : BPTR;
                            system IN D3 : LONGINT ): LONGBOOL;
```

```
LIBRARY DosBase BY -780
  PROCEDURE FindSegment( REF name   IN D1 : STRING;
                             seg    IN D2 : BPTR;
                             system IN D3 : LONGINT ): BPTR;

LIBRARY DosBase BY -786
  PROCEDURE RemSegment( seg IN D1 : BPTR ): LONGBOOL;

GROUP
  DosSegmentGrp  = Execute,LoadSeg, UnLoadSeg, InternalLoadSeg,
                   InternalUnLoadSeg,NewLoadSeg, AddSegment,
                   FindSegment, RemSegment, BPTR,FileHandlePtr;

|------------------------------------------------------------------------
|  6. DOS-Interne Befehle
|------------------------------------------------------------------------

LIBRARY DosBase BY -162
  PROCEDURE GetPacket( Wait IN D1 : LONGINT ): DosPacketPtr;

LIBRARY DosBase BY -168
  PROCEDURE QueuePacket( Packet IN D1 : DosPacketPtr ): LONGINT;

LIBRARY DosBase BY -240
  PROCEDURE DoPkt( port   IN D1 : MsgPortPtr;
                   action IN D2 : LONGINT;
                   arg1   IN D3 : LONGINT;
                   arg2   IN D4 : LONGINT;
                   arg3   IN D5 : LONGINT;
                   arg4   IN D6 : LONGINT;
                   arg5   IN D7 : LONGINT ): LONGINT;

LIBRARY DosBase BY -246
  PROCEDURE SendPkt( dp        IN D1 : DosPacketPtr;
                     port      IN D2 : MsgPortPtr;
                     replyport IN D3 : MsgPortPtr );

LIBRARY DosBase BY -252
  PROCEDURE WaitPkt(): DosPacketPtr;

LIBRARY DosBase BY -258
  PROCEDURE ReplyPkt( dp   IN D1 : DosPacketPtr;
                      res1 IN D2 : LONGINT;
                      res2 IN D3 : LONGINT );

LIBRARY DosBase BY -264
  PROCEDURE AbortPkt( port IN D1 : MsgPortPtr;
                      pkt  IN D2 : DosPacketPtr );

LIBRARY DosBase BY -924
  PROCEDURE CliInit( dp IN A0 : DosPacketPtr ): LONGINT;

LIBRARY DosBase BY -930
  PROCEDURE CliInitNewCli( dp IN A0 : DosPacketPtr ): LONGINT;
```



```
LIBRARY DosBase BY -936
  PROCEDURE CliInitRun( dp IN A0 : DosPacketPtr ): LONGINT;


GROUP
  PacketGrp     = DosPacket,DosPacketPtr,StandardPacket,StandardPacketPtr,
                  nil,getBlock,setMap,die,event,currentVolume,locateObject,
                  renameDisk,write,read,freelock,deleteObject,renameObject,
                  moreCache,copyDir,waitChar,setProtect,createDir,
                  examineObject,examineNext,diskInfo,info,flush,setComment,
                  parent,timer,inhibit,diskType,diskChange,setDate,
                  screenMode,readReturn,writeReturn,findUpDate,findInput,
                  findOutput,actionEnd,seek,truncate,writeLock,
                  GetPacket,QueuePacket, DoPkt, SendPkt, WaitPkt, ReplyPkt,
                  AbortPkt, CliInit, CliInitNewCli, CliInitRun,MsgPortPtr

|------------------------------------------------------------------------
|  7. Dos - Listen Befehle
|------------------------------------------------------------------------

LIBRARY DosBase BY -654
  PROCEDURE LockDosList( flags IN D1 : LockDosFlagSet ): DosListPtr;

LIBRARY DosBase BY -660
  PROCEDURE UnLockDosList( flags IN D1 : LockDosFlagSet );

LIBRARY DosBase BY -666
  PROCEDURE AttemptLockDosList( flags IN D1 : LockDosFlagSet ): DosListPtr;

LIBRARY DosBase BY -672
  PROCEDURE RemDosEntry( dlist IN D1 : DosListPtr ): LONGBOOL;

LIBRARY DosBase BY -678
  PROCEDURE AddDosEntry( dlist IN D1 : DosListPtr ): LONGBOOL;

LIBRARY DosBase BY -684
  PROCEDURE FindDosEntry(     dlist IN D1 : DosListPtr;
                          REF name  IN D2 : STRING;
                              flags IN D3 : LONGCARD ): DosListPtr;

LIBRARY DosBase BY -690
  PROCEDURE NextDosEntry( dlist IN D1 : DosListPtr;
                          flags IN D2 : LONGCARD ): DosListPtr;

LIBRARY DosBase BY -696
  PROCEDURE MakeDosEntry( REF name IN D1 : STRING;
                              type IN D2 : LONGINT ): DosListPtr;

LIBRARY DosBase BY -702
  PROCEDURE FreeDosEntry( dlist IN D1: DosListPtr );


GROUP
  DosListGrp    =  LockDosList, UnLockDosList, AttemptLockDosList,
                   RemDosEntry,AddDosEntry, FindDosEntry, NextDosEntry,
```

```
                   MakeDosEntry,FreeDosEntry, LockDosFlags,
                   LockDosFlagSet, DosListPtr;

|** Now the new 2.0 Functions **


|------------------------------------------------------------------------
|  8. Argument/PatternMatch & sonstige String Manipulationen
|------------------------------------------------------------------------

LIBRARY DosBase BY -798
  PROCEDURE ReadArgs( REF template IN D1 : STRING;
                          array    IN D2 : ANYPTR;
                          args     IN D3 : RDArgsPtr ): RDArgsPtr;

LIBRARY DosBase BY -804
  PROCEDURE FindArg( REF keyword  IN D1 : STRING;
                     REF template IN D2 : STRING ): LONGINT;

LIBRARY DosBase BY -810
  PROCEDURE ReadItem( REF name     IN D1 : STRING;
                          maxchars IN D2 : LONGINT;
                          cSource  IN D3 : CSourcePtr ): LONGINT;

LIBRARY DosBase BY -816
  PROCEDURE StrToLong( REF string IN D1 : STRING;
                           value  IN D2 : ANYPTR ): LONGINT;

LIBRARY DosBase BY -822
  PROCEDURE MatchFirst( REF pat    IN D1 : STRING;
                            anchor IN D2 : AnchorPathPtr ): LONGBOOL;

LIBRARY DosBase BY -828
  PROCEDURE MatchNext( anchor IN D1 : AnchorPathPtr ): LONGBOOL;

LIBRARY DosBase BY -834
  PROCEDURE MatchEnd( anchor IN D1 : AnchorPathPtr );

LIBRARY DosBase BY -840
  PROCEDURE ParsePattern( REF pat    IN D1 : STRING;
                          REF buf    IN D2 : STRING;
                              buflen IN D3 : LONGINT ): LONGINT;

LIBRARY DosBase BY -846
  PROCEDURE MatchPattern( REF pat IN D1 : STRING;
                          REF str IN D2 : STRING ): LONGBOOL;



LIBRARY DosBase BY -858
  PROCEDURE FreeArgs( args IN D1 : RDArgsPtr );

LIBRARY DosBase BY -870
  PROCEDURE FilePart( REF path IN D1 : STRING ): SysStringPtr;
```



```
LIBRARY DosBase BY -876
  PROCEDURE PathPart( REF path IN D1 : STRING ): SysStringPtr;

LIBRARY DosBase BY -882
  PROCEDURE AddPart( REF dirname  IN D1 : STRING;
                     REF filename IN D2 : STRING;
                         size     IN D3 : LONGCARD ): LONGBOOL;

LIBRARY DosBase BY -966
  PROCEDURE ParsePatternNoCase( REF buf    IN D1 : STRING;
                                    pat    IN D2 : ANYPTR;
                                    buflen IN D3 : LONGINT ): LONGINT;

LIBRARY DosBase BY -972
  PROCEDURE MatchPatternNoCase(     pat IN D1 : ANYPTR;
                                REF str IN D2 : STRING ): LONGBOOL;


GROUP
  DosArgGrp     =  ReadArgs, FindArg, ReadItem, StrToLong, MatchFirst,
                   MatchNext,MatchEnd, ParsePattern, MatchPattern,
                   FreeArgs, FilePart,PathPart, AddPart, RDArgs,
                   RDArgsPtr, CSource, CSourcePtr,AnchorPath,
                   AnchorPathPtr, ParsePatternNoCase, MatchPatternNoCase;


|------------------------------------------------------------------------
|  9. Var Group
|------------------------------------------------------------------------
LIBRARY DosBase BY -900
  PROCEDURE SetVar( REF name   IN D1 : STRING;
                    REF buffer IN D2 : STRING;
                        size   IN D3 : LONGINT;
                        flags  IN D4 : GetVarFlagSet): LONGBOOL;

PROCEDURE GetVar( REF name   : STRING;
                  VAR buffer : STRING;
                      size   : LONGINT;
                      flags  : GetVarFlagSet): LONGBOOL;

LIBRARY DosBase BY -912
  PROCEDURE DeleteVar( REF name  IN D1 : STRING;
                           flags IN D2 : GetVarFlagSet): LONGBOOL;

LIBRARY DosBase BY -918
  PROCEDURE FindVar( REF name IN D1: STRING;
                         type IN D2: GetVarFlagSet): LocalVarPtr;


GROUP
  DosVarGrp     =  SetVar, GetVar, DeleteVar, FindVar, LocalVar,
                   LocalVarPtr,GetVarFlags;
```

```
|-----------------------------------------------------------------------
|  10. Misc
|-----------------------------------------------------------------------


LIBRARY DosBase BY -228
  PROCEDURE AllocDosObject( type IN D1 : DosObject;
                            tags IN D2 : LIST OF DosObjectTags):ANYPTR;
LIBRARY DosBase BY -228
  PROCEDURE AllocDosObjectList( type IN D1 : DosObject;
                                tags IN D2 : DosObjectTagListPtr):ANYPTR;

LIBRARY DosBase BY -234
  PROCEDURE FreeDosObject( type IN D1 : DosObject;
                           ptr  IN D2 : ANYPTR );

LIBRARY DosBase BY -792
  PROCEDURE CheckSignal( mask IN D1 : LONGCARD ): LONGCARD;


LIBRARY DosBase BY -888
  PROCEDURE StartNotify( notify IN D1 : NotifyRequestPtr ): LONGBOOL;

LIBRARY DosBase BY -894
  PROCEDURE EndNotify( notify IN D1 : NotifyRequestPtr );


GROUP
  DosMiscGrp    =  AllocDosObject, FreeDosObject, CheckSignal, StartNotify,
                   EndNotify, NotifyRequest, NotifyRequestPtr;

  TypeGrp       = AChainFlags,        AnchorPathFlags,    BreakSignals,
                  ChangeModeType,     DateFormats,        DatFlags,
                  DeviceListType,     DevProcFlags,       DiskStatus,
                  DiskType,           DosCommands,        DosObject,
                  EntryType,          ErrorReportType,    ExAllType,
                  External,           GetVarFlags,
                  Hunks,              IoErrors,           LinkType,
                  GetVarFlags,        LockDosFlags,       LockMode,
                  NotifyRequestFlags, OpenMode,
                  ProcessFlags,       ProtectionFlags,    RDArgsFlags,
                  ReadItemType,       ReturnCode,         RootNodeFlags,
                  RunType,            SameLockType,       SameLockType,
                  SeekMode;


GROUP
  All           = AChain,
                  BSTR,                         CommandLineInterface,
                  CommandLineInterfacePtr,      ctrlC,
                  ctrlD,                        ctrlE,
                  ctrlF,                        Date,
                  DatePtr,                      DateTime,
                  DateTimePtr,
                  DeviceList,                   DeviceListPtr,
```



```
                  DeviceListType,               DeviceNode,
                  DeviceNodePtr,                DevProc,
                  DosArgGrp,                    DosBase,
                  DosEnvec,                     DosEnvecPtr,
                  DosErrorGrp,
                  DosFileGrp,                   DosInfo,
                  DosInfoPtr,                   DosIOGrp,
                  DosLibrary,                   DosLibraryPtr,
                  DosListGrp,                   DosMiscGrp,
                  DosProcessGrp,                DosSegmentGrp,
                  DosVarGrp,                    error,
                  fail,                         FileHandle,
                  FileLock,                     FileSysStartupMsg,
                  FileSysStartupMsgPtr,         ok,
                  PacketGrp,                    PathInfo,
                  PathInfoPtr,                  Process,
                  ResidentSegment,              ResidentSegmentPtr,
                  RootNode,                     RootNodePtr,
                  TaskArray,                    TaskArrayPtr,
                  TypeGrp,                      warn;


LIBRARY DosBase BY -402
  PROCEDURE ROMNameFromLock(lock   IN D1 : FileLockPtr;
                            buffer IN D2 : SysStringPtr;
                            len    IN D3 : LONGINT ): LONGBOOL;


LIBRARY DosBase BY -408
  PROCEDURE ROMNameFH(fh     IN D1 : FileHandlePtr;
                      buffer IN D2 : SysStringPtr;
                      len    IN D3 : LONGINT ): LONGBOOL;


LIBRARY DosBase BY -414
  PROCEDURE ROMSplitName( REF name      IN D1 : STRING;
                              seperator IN D2 : CHAR;
                              buf       IN D3 : SysStringPtr;
                              oldpos    IN D4 : INTEGER;
                              size      IN D5 : LONGINT ): INTEGER;

LIBRARY DosBase BY -438
  PROCEDURE ROMReadLink(     port   IN D1 : MsgPortPtr;
                             lock   IN D2 : FileLockPtr;
                         REF path   IN D3 : STRING;
                             buffer IN D4 : SysStringPtr;
                             size   IN D5 : LONGCARD ): LONGBOOL;

LIBRARY DosBase BY -468
  PROCEDURE ROMFault(     code   IN D1 : IoErrors;
                      REF header IN D2 : STRING;
                          buffer IN D3 : SysStringPtr;
                          len    IN D4 : LONGINT ): LONGBOOL;
```

```
 LIBRARY DosBase BY -564
  PROCEDURE ROMGetCurrentDirName(buf IN D1 : SysStringPtr;
                                 len IN D2 : LONGINT ): LONGBOOL;


LIBRARY DosBase BY -576
  PROCEDURE ROMGetProgramName(  buf IN D1 : SysStringPtr;
                                len IN D2 : LONGINT ): LONGBOOL;


LIBRARY DosBase BY -906
  PROCEDURE ROMGetVar( REF name   IN D1 : STRING;
                           buffer IN D2 : SysStringPtr;
                           size   IN D3 : LONGINT;
                           flags  IN D4 : GetVarFlagSet): LONGBOOL;

LIBRARY DosBase BY -198
  PROCEDURE ROMDelay( ticks IN D1 : LONGCARD );

LIBRARY DosBase BY -588
  PROCEDURE ROMGetPrompt(  buf IN D1 : SysStringPtr;
                           len IN D2 : LONGINT ): LONGBOOL;

END Dos.
```


8.14 Exec

```
DEFINITION MODULE Exec;
|   WARNING
|
| autodocs and fdfiles sometimes differ. search for "WARNING"
|


| Groups in this module ( in this order ):
|
| Const,      Node,       List,       Int,        Mem,        Task,
| MsgPort,    oldMsgPort, Lib,        DeviceIO,   Semaphore,  Resident,
| ExecBase,   Special,    Signal,     Trap,       Resource,   Kick,
| Private,    Cache,      Child,      Func,       Ptr,        Record
|

| No structures are based on ResHandles due to multiple inheritance. Look
| in T_Exec for ResHandles.
|
| MinNode, MinList, List, IntVector, MemChunk, MemEntry, Resident
|

(* $A- *)


FROM System    IMPORT
          (* T *) BITSET,             BPTR,               LONGSET,
                  PROC,               Regs,               SysStringPtr;

FROM Hardware  IMPORT
          (* T *) CustomPtr,          IntFlags,           IntFlagSet
                  (*,LIntFlags *);

|
| sometimes you need ...
|
TYPE

  LONGPTR = POINTER TO LONGINT;

|
| ExecBase forward for Library calls ( found in ExecBaseGrp )
|




TYPE ExecBasePtr  = POINTER TO ExecBaseType;

VAR  ExecBase : ExecBasePtr;
```

```
|------------------------------------------------------------------------
| types
|------------------------------------------------------------------------

| most of CBM types are superfluous, as Cluster has its own definitions
| This group is not included in All, as the members are not very important.

CONST

  includeVersion = 36;    | version of the includes used.
  byteMask       = 255;   | For the fumblers.
  libraryMinimum = 33;    | Lowest version supported by Commodore-Amiga.

GROUP

  ConstGrp = includeVersion, byteMask, libraryMinimum;


|------------------------------------------------------------------------
| nodes
|------------------------------------------------------------------------

TYPE

  | User node types work down from "user"
  NodeType    = ( unknown,            task,               interrupt,
                  device,             msgPort,            message,
                  freeMsg,            replyMsg,           resoucre,
                  library,            memory,             softInt,
                  font,               process,            semaphore,
                  signalSem,          bootNode,           (* v36 *)
                  kickMem,            graphics,           deathMessage,
                  user = 254,         extended );

  NodePri       = [-128..127]; | SHORTINT;

  MinNodePtr    = POINTER TO MinNode;
  MinNode       = RECORD                             | 8 Bytes
                    succ,
                    pred  : SAMEPTR;
                  END;

  NodePtr       = POINTER TO Node;
  Node          = RECORD OF MinNode                  | 14 Bytes
                    type  : NodeType;
                    pri   : NodePri;
                    name  : SysStringPtr;
                  END;
```



```
GROUP

  NodeGrp =
          (* I *) SysStringPtr,
          (* T *) MinNode,            MinNodePtr,         Node,
                  NodePtr,            NodeType;


|------------------------------------------------------------------------
| lists
|------------------------------------------------------------------------

TYPE

  MinListPtr    = POINTER TO MinList;
  MinList       = RECORD                                 | 12 Bytes
                    head,
                    tail,
                    tailPred : MinNodePtr
                  END;

  ListPtr       = POINTER TO List;
  List          = RECORD                                 | 14 Bytes
                    head,
                    tail,
                    tailPred : NodePtr;
                    type     : NodeType;
                    pad      : SHORTCARD
                  END;

|
| As List cannot be made a successor of MinList, we have to define seperate

| functions for listfunctions, which can be used both for Nodes/Lists and
| MinNodes/MinLists. The Min* variant will have an 'M' before the object.
|

LIBRARY ExecBase BY -240
  PROCEDURE AddHead( list IN A0 : ListPtr;
                     node IN A1 : NodePtr );

LIBRARY ExecBase BY -240
  PROCEDURE AddMHead( minList IN A0 : MinListPtr;
                      minNode IN A1 : MinNodePtr );

LIBRARY ExecBase BY -246
  PROCEDURE AddTail( list IN A0 : ListPtr;
                     node IN A1 : NodePtr );

LIBRARY ExecBase BY -246
  PROCEDURE AddMTail( minList IN A0 : MinListPtr;
                      minNode IN A1 : MinNodePtr );
```

```
| initialize a piece of memory pointed to by "list" to be an empty list.
PROCEDURE NewList( VAR list : List;
                       type : NodeType );

PROCEDURE NewMList( VAR minList : MinList );

| no MinNodes allowed.
LIBRARY ExecBase BY -270
  PROCEDURE Enqueue( list IN A0 : ListPtr;
                     node IN A1 : NodePtr );

LIBRARY ExecBase BY -276
  PROCEDURE FindName( list IN A0 : ListPtr;
                      name IN A1 : SysStringPtr ): NodePtr;

LIBRARY ExecBase BY -234
  PROCEDURE Insert( list     IN A0 : ListPtr;
                    node     IN A1,
                    listNode IN A2 : NodePtr );

LIBRARY ExecBase BY -234
  PROCEDURE MInsert( minList     IN A0 : MinListPtr;
                     minNode     IN A1,
                     minListNode IN A2 : MinNodePtr );

LIBRARY ExecBase BY -258
  PROCEDURE RemHead( list IN A0 : ListPtr ): NodePtr;

LIBRARY ExecBase BY -258
  PROCEDURE RemMHead( minList IN A0 : MinListPtr ): MinNodePtr;

| works for Nodes just as well
LIBRARY ExecBase BY -252
  PROCEDURE Remove( minNode IN A1 : MinNodePtr );

LIBRARY ExecBase BY -264
  PROCEDURE RemTail( list IN A0 : ListPtr ): NodePtr;

LIBRARY ExecBase BY -264
  PROCEDURE RemMTail( minList IN A0 : MinListPtr ): MinNodePtr;

GROUP

  ListGrp =
        (* I *) NodeGrp,
        (* T *) List,              ListPtr,            MinList,
                MinListPtr,
        (* P *) AddHead,           AddTail,            NewList,
                Enqueue,           FindName,           Insert,
                RemHead,           Remove,             RemTail;
```



```
|-------------------------------------------------------------------------
| interrupts
|-------------------------------------------------------------------------

TYPE

  InterruptPtr    = POINTER TO Interrupt;

  | Functions which can be used as interupt code.
  IntFuncPtr      = POINTER TO IntFunc;
  IntFunc         = PROCEDURE( data IN A1 : ANYPTR;       | Interrupt.data
                               custom IN A0 : CustomPtr;
                               int    IN D1 : IntFlagSet; | intenar AND
                                                          | intreqr
                               mycode IN A5 : IntFuncPtr;
                               exec   IN A6 : ExecBasePtr ) : LONGINT;
  Interrupt       = RECORD OF Node                       | 22 Bytes
                      data : ANYPTR;
                      code : PROC;                       | IntFunc
                    END;

  | For ExecBase use only
  IntVectorPtr    = POINTER TO IntVector;
  IntVector       = RECORD                               | 12 Bytes
                      data : ANYPTR;
                      code : PROC;
                      node : NodePtr
                    END;

  | For ExecBase use only
  SoftIntListPtr  = POINTER TO SoftIntList;
  SoftIntList     = RECORD OF List                 | 16 Bytes
                      pad  : CARDINAL              | keep longword alignment
                    END;

PROCEDURE AddIntServer( intNumber : IntFlags;      | Autodocs are wrong
                        interrupt : InterruptPtr );

|LIBRARY ExecBase BY -168
  |PROCEDURE AddIntServer( intNumber IN D0 : IntFlags; | Autodocs are wrong
  |                        interrupt IN A1 : InterruptPtr );
LIBRARY ExecBase BY -180
  PROCEDURE Cause( (* REF *) Interrupt IN A1 : InterruptPtr );

LIBRARY ExecBase BY -120
  PROCEDURE Disable;

LIBRARY ExecBase BY -126
  PROCEDURE Enable;

LIBRARY ExecBase BY -132
  PROCEDURE Forbid;
```

```
LIBRARY ExecBase BY -138
  PROCEDURE Permit;

  PROCEDURE RemIntServer( intNumber : IntFlags;        | Autodocs are wrong
                          interrupt : InterruptPtr );
|LIBRARY ExecBase BY -174
  |PROCEDURE RemIntServer( intNumber IN D0 : IntFlags; | Autodocs are wrong
  |                        interrupt IN A1 : InterruptPtr );

PROCEDURE SetIntVector( intNumber : IntFlags;          | Autodocs are wrong
                        interrupt : InterruptPtr ): InterruptPtr;
|LIBRARY ExecBase BY -162
  |PROCEDURE SetIntVector( intNumber IN D0 : IntFlags; | Autodocs are wrong
  |                        interrupt IN A1 : InterruptPtr ): InterruptPtr;
LIBRARY ExecBase BY -144
  PROCEDURE SetSR( newSR IN D0 : BITSET; mask IN D1 :  BITSET ): BITSET;

LIBRARY ExecBase BY -150
  PROCEDURE SuperState(): ANYPTR;

LIBRARY ExecBase BY -30
  PROCEDURE Supervisor( userFunction IN A5: PROC );

LIBRARY ExecBase BY -156
  PROCEDURE UserState( sysStack IN D0 : ANYPTR );


GROUP

  IntGrp  =
        (* I *) IntFlags,          (*LIntFlags,*)      ListGrp,
                PROC,
        (* T *) Interrupt,         InterruptPtr,       IntFunc,
                IntFuncPtr,        IntVector,          IntVectorPtr,
                SoftIntList,       SoftIntListPtr,
        (* P *) AddIntServer,      Cause,              Disable,
                Enable,            Forbid,             Permit,
                RemIntServer,      SetIntVector,       SetSR,
                SuperState,        Supervisor,         UserState;
```



```
|----------------------------------------------------------------------
| memory
|----------------------------------------------------------------------

TYPE

  | local, dma24, reverse and total are new for V36.
  | dma24 is memory within the 24 bit address space.
  MemReqs       = ( public,   chip,     fast,     mr3,      mr4,      mr5,
                    mr6,      mr7,      local,    dma24,    mr10,     mr11,
                    mr12,     mr13,     mr14,     mr15,     clear, largest,
                    reverse,  total );
  | clear used to be memClear when the compiler did not like it

  MemReqSet     = SET OF MemReqs;
  MemTypeSet    = SET OF [public..mr15];

  MemChunkPtr   = POINTER TO MemChunk;
  MemChunk      = RECORD                                    | 8 Bytes
                    next  : MemChunkPtr;
                    bytes : LONGCARD
                  END;

  MemHeaderPtr  = POINTER TO MemHeader;
  MemHeader     = RECORD OF Node                            | 32 Bytes
                    attributes : MemTypeSet;
                    first      : MemChunkPtr;
                    lower      : ANYPTR;
                    upper      : ANYPTR;
                    free       : LONGCARD
                  END;

  MemEntryPtr   = POINTER TO MemEntry;
  MemEntry      = RECORD                                    | 8 Bytes
                    IF KEY :INTEGER
                      OF 1 THEN reqs : MemReqSet
                      OF 2 THEN addr : ANYPTR
                    END;
                    length : LONGCARD;
                  END;

  MemListPtr    = POINTER TO MemList;
  MemList       = RECORD OF Node          | 16+me'RANGE*8 Bytes
                    numEntries : CARDINAL;
                    me         : ARRAY [1000] OF MemEntry;
                  END;


LIBRARY ExecBase BY -204
  PROCEDURE AllocAbs( byteSize IN D0 : LONGINT;
                      location IN A1 : ANYPTR ): ANYPTR;

LIBRARY ExecBase BY -186
  PROCEDURE Allocate( freeList IN A0 : MemHeaderPtr;
                      byteSize IN D0 : LONGCARD ): ANYPTR;
```

```
LIBRARY ExecBase BY -222
  PROCEDURE AllocEntry( memList IN A0 : MemListPtr ): MemListPtr;

LIBRARY ExecBase BY -198
  PROCEDURE AllocMem( byteSize     IN D0 : LONGCARD; | WARNING: used to
                                                     | be LONGINT
                      requirements IN D1 : MemReqSet ): ANYPTR;

| no doc available, prototype from fd file and thin air (V37).
LIBRARY ExecBase BY -708
  PROCEDURE AllocPooled( memSize    IN D0 : LONGCARD;
                         poolHeader IN A0 : ANYPTR ): ANYPTR;

| allocate memory and keep track of the size  (V36)
| like AllocMem, but remembers the size. FREE WITH FreeVec !!!
LIBRARY ExecBase BY -684
  PROCEDURE AllocVec( byteSize     IN D0 : LONGCARD;
                      requirements IN D1 : MemReqSet ): ANYPTR;

LIBRARY ExecBase BY -216
  PROCEDURE AvailMem( requirements IN D1 : MemReqSet ): LONGCARD;

| no doc available, prototype from fd file and thin air (V37).
LIBRARY ExecBase BY -696
  PROCEDURE CreatePrivatePool( requirements IN D0 : MemReqSet; | this is
                                                               | a guess
                               puddleSize   IN D1,
                               puddleThresh IN D2 : LONGCARD ): ANYPTR;

LIBRARY ExecBase BY -192
  PROCEDURE Deallocate( freeList    IN A0 : MemHeaderPtr;
                        memoryBlock IN A1 : ANYPTR;
                        byteSize    IN D0 : LONGCARD );

| no doc available, prototype from fd file and thin air (V37).
LIBRARY ExecBase BY -702
  PROCEDURE DeletePrivatePool( poolHeader IN A0 : ANYPTR );

LIBRARY ExecBase BY -228
  PROCEDURE FreeEntry( memList IN A0 : MemListPtr );

LIBRARY ExecBase BY -210
  PROCEDURE FreeMem( memoryBlock IN A1 : ANYPTR;
                     byteSize    IN D0 : LONGCARD );

| no doc available, prototype from fd file and thin air (V37).
LIBRARY ExecBase BY -714
  PROCEDURE FreePooled( memory IN A1, poolHeader IN A0 : ANYPTR ) : ANYPTR;

| return AllocVec() memory to the system  (V36)
LIBRARY ExecBase BY -690
  PROCEDURE FreeVec( memoryBlock IN A1: ANYPTR );

LIBRARY ExecBase BY -534
```



```
  PROCEDURE TypeOfMem( address IN A0 : ANYPTR ): MemReqSet;

GROUP

  | New for V36 are AllocPooled, AllocVec, CreatePrivatePool,
  | DeletePrivatePool, FreePooled, FreeVec.
  MemGrp  =
          (* I *)   NodeGrp,
          (* T *) MemChunk,          MemChunkPtr,       MemEntry,
                  MemEntryPtr,       MemHeader,         MemHeaderPtr,
                  MemList,           MemListPtr,        MemReqs,
                  MemReqSet,         MemTypeSet,
          (* P *) AllocAbs,          Allocate,          AllocEntry,
                  AllocMem,          AllocPooled,       AllocVec,
                  AvailMem,          CreatePrivatePool, Deallocate,
                  DeletePrivatePool, FreeEntry,         FreeMem,
                  FreePooled,        FreeVec,           TypeOfMem;


|--------------------------------------------------------------------------
| tasks
|--------------------------------------------------------------------------

TYPE

  | etask is new for V36
  TaskFlags     = ( procTime,   tf1,        tf2,        etask,
                    stackChk,   exception,  switch,     launch );
  TaskFlagSet   = SET OF TaskFlags;

  TaskState     = ( inval, added, run, ready, wait, except, removed );

  | Signals 0-15 are preallocated, but only some have official names. The
  | highest 16 signals (upto max) are there for the task's use. Please do
  | allocate them before using them
  |
  TaskSignals   = ( noSignal  = -1,
                    anySignal = -1,
                    abort,        child,        ts2,          ts3,
                    blit,         single=4,     intuition,    ts6,
                    ts7,          dos,          ts9,          ts10,
                    ts11,         ctrlC,        ctrlD,        ctrlE,
                    ctrlF,        max=31 );
  TaskSigSet    = SET OF TaskSignals;
```

```
  | Procedures registered as exceptCode get called like this. 'data' is
  | the exceptData field from the Task RECORD. 'signals' is the set of
  | exceptions, that occured. The result value of an ExceptPROC is put
  | into sigExcept, thus setting the signals which can cause a soft
  | exception from then on. The Cluster runtimesystem has an
  | exceptionhandling routine of its own, so be careful to use this
  | possibilty on your own.
  |
  ExceptPROC    = PROCEDURE( signals IN D0 : TaskSigSet;
                             data    IN A1 : ANYPTR ) : TaskSigSet;

  TaskPtr       = POINTER TO Task;
  Task          = RECORD OF Node                       | 92 Bytes
                    flags     : TaskFlagSet;
                    state     : TaskState;
                    idNestCnt,
                    tdNestCnt : SHORTINT;  | WARNING: used to be SHORTCARD
                    sigAlloc,
                    sigWait,
                    sigRecvd,
                    sigExcept : TaskSigSet;
                    trapAlloc,
                    trapAble  : BITSET;
                    exceptData: ANYPTR;
                    exceptCode: ExceptPROC;
                    trapData  : ANYPTR;
                    trapCode  : PROC;
                    spReg,
                    spLower,
                    spUpper   : ANYPTR;
                    switch,
                    launch    : PROC;
                    memEntry  : List;
                    userData  : ANYPTR;
                  END;

CONST

  | WARNING: Sig* are now real signals, not the Bit numbers.
  SigAbort    = TaskSigSet:{ abort };
  SigChild    = TaskSigSet:{ child };
  SigBlit     = TaskSigSet:{ blit };      | same as single
  SigSingle   = TaskSigSet:{ single };    | same as blit
  SigDos      = TaskSigSet:{ dos };       | used for Dos calls
  SigCtrlC    = TaskSigSet:{ ctrlC };
  SigCtrlD    = TaskSigSet:{ ctrlD };
  SigCtrlE    = TaskSigSet:{ ctrlE };
  SigCtrlF    = TaskSigSet:{ ctrlF };


LIBRARY ExecBase BY -282
  PROCEDURE AddTask( task      IN A1 : TaskPtr;
                     initialPC IN A2 : ANYPTR;
                     finalPC   IN A3 : ANYPTR ): TaskPtr;
```



```
LIBRARY ExecBase BY -288
  PROCEDURE RemTask( task IN A1 : TaskPtr );

LIBRARY ExecBase BY -294
  PROCEDURE FindTask( name IN A1 : SysStringPtr ): TaskPtr;

LIBRARY ExecBase BY -300
  PROCEDURE SetTaskPri( task IN A1 : TaskPtr;
                        pri  IN D0 : NodePri ): NodePri;

| CreateTask and DeleteTask are in T_Exec

GROUP

  TaskGrp =
          (* I *) ListGrp,          PROC,
          (* T *) ExceptPROC,       Task,             TaskFlags,
                  TaskFlagSet,      TaskPtr,          TaskSignals,
                  TaskSigSet,       TaskState,
          (* C *) SigAbort,         SigBlit,          SigChild,
                  SigDos,           SigSingle,        SigCtrlC,
                  SigCtrlD,         SigCtrlE,         SigCtrlF,
          (* P *) AddTask,          FindTask,         RemTask,
                  SetTaskPri;


|------------------------------------------------------------------------
| ports
|------------------------------------------------------------------------

TYPE

  MsgPortAction = ( signal, softInt, ignore );

  MsgPortPtr    = POINTER TO MsgPort;
  MsgPort       = RECORD OF Node
                    IF KEY flags : MsgPortAction
                      OF signal  THEN sigBit  : TaskSignals;
                                      sigTask : TaskPtr
                      OF softInt THEN softInt : InterruptPtr
                    END;
                    msgList : List;
                  END;

  MessagePtr    = POINTER TO Message;
  Message       = RECORD OF Node
                    replyPort : MsgPortPtr;
                    msgSize   : CARDINAL;    | include Message'SIZE !!!
                  END;


LIBRARY ExecBase BY -354
  PROCEDURE AddPort( port IN A1 : MsgPortPtr );
```

```
| Allocate and initialize a new message port  (V36)
| Alloc a signal, clear msgList and set port to signal the calling task.
| To make public, set name and pri, AddPort, use, RemPort and DeleteMsgPort.
| YOU *MUST* USE DeleteMsgPort TO DELETE PORTS CREATED WITH CreateMsgPort!
LIBRARY ExecBase BY -666
  PROCEDURE CreateMsgPort(): MsgPortPtr;

| Free a message port created by CreateMsgPort  (V36)
| port may be NIL. msgList MUST already be empty.
LIBRARY ExecBase BY -672
  PROCEDURE DeleteMsgPort( port IN A0 : MsgPortPtr );

LIBRARY ExecBase BY -390
  PROCEDURE FindPort( REF name IN A1 : STRING ): MsgPortPtr;

LIBRARY ExecBase BY -372
  PROCEDURE GetMsg( port IN A0 : MsgPortPtr ): MessagePtr;

LIBRARY ExecBase BY -366
  PROCEDURE PutMsg( port IN A0 : MsgPortPtr;
                    msg  IN A1 : MessagePtr );

LIBRARY ExecBase BY -360
  PROCEDURE RemPort( port IN A1 : MsgPortPtr );

LIBRARY ExecBase BY -378
  PROCEDURE ReplyMsg( msg IN A1 : MessagePtr );

LIBRARY ExecBase BY -384
  PROCEDURE WaitPort( port IN A0 : MsgPortPtr ): MessagePtr;

GROUP

  MsgPortGrp =
  (* I *)   TaskGrp,
  (* T *)   Message,        MessagePtr,     MsgPort,        MsgPortAction,
            MsgPortPtr,
  (* P *)   AddPort,        CreateMsgPort,  DeleteMsgPort,  FindPort,
            GetMsg,         PutMsg,         RemPort,
            ReplyMsg,       WaitPort;


|------------------------------------------------------------------------
| libraries
|------------------------------------------------------------------------

CONST

  vectSize      = 6;
  reserved      = 4;
  base          = -vectSize;
  userDef       = base-reserved*vectSize;
  nonStd        = userDef;
```



```
  extFunc       = -24;  | user functions begin here
  expunge       = -18;
  close         = -12;
  open          = -6;

TYPE

  LibFlags      = ( summing,  | currently being checksummed
                    changed,  | has just been changed
                    sumUsed,  | should be summed
                    delExp ); | delayed expunge
  LibFlagSet    = SET OF LibFlags;

  LibraryPtr    = POINTER TO Library;
  Library       = RECORD OF Node                | 34 Bytes
                    flags    : LibFlagSet;
                    libPad   : SHORTCARD;   | WARNING: was missing here
                    negSize,
                    posSize,
                    version,
                    revision : CARDINAL;
                    idString : SysStringPtr;
                    sum      : LONGCARD;
                    openCnt  : CARDINAL;
                  END;                  | Warning: not a longword multiple

  LibInitProc   = PROCEDURE ( lib     IN D0 : LibraryPtr;
                              segList IN A0 : BPTR;
                              exec    IN A6 : ExecBasePtr ): LibraryPtr;

  FuncArrayPtr  = POINTER TO ARRAY OF PROC;

LIBRARY ExecBase BY -396
  PROCEDURE AddLibrary( library IN A1 : LibraryPtr );

LIBRARY ExecBase BY -414
  PROCEDURE CloseLibrary( library IN A1 : LibraryPtr );

LIBRARY ExecBase BY -90
  PROCEDURE MakeFunctions( target       IN A0 : ANYPTR;
                           functArray    IN A1 : ANYPTR;
                           functDispBase IN A2 : ANYPTR );

LIBRARY ExecBase BY -84
  PROCEDURE MakeLibrary( vectors   IN A0 : FuncArrayPtr;
                         structure IN A1 : ANYPTR;
                         init      IN A2 : LibInitProc;
                         dataSize  IN D0 : LONGCARD;
                         segList   IN D1 : BPTR ): LibraryPtr;

LIBRARY ExecBase BY -408
  PROCEDURE OldOpenLibrary( REF name IN A1 : STRING ): LibraryPtr;

LIBRARY ExecBase BY -552
  PROCEDURE OpenLibrary( REF name    IN A1 : STRING;
```

```
                        version IN D0 : LONGINT ): LibraryPtr;

LIBRARY ExecBase BY -402
  PROCEDURE RemLibrary( library IN A1 : LibraryPtr );

LIBRARY ExecBase BY -420
  PROCEDURE SetFunction( library    IN A1 : LibraryPtr;
                         funcOffset IN A0 : INTEGER;
                         funcEntry  IN D0 : ANYPTR   ): ANYPTR;

| no doc available, prototype from fd file and thin air.
LIBRARY ExecBase BY -720
  PROCEDURE SetFunction8( funcOffset  IN D0 : LONGCARD;
                          newFunction IN D1 : ANYPTR;
                          array       IN A0 : ANYPTR;
                          library     IN A1 : LibraryPtr ): ANYPTR;

LIBRARY ExecBase BY -426
  PROCEDURE SumLibrary( library IN A1 : LibraryPtr );

GROUP

  | New for V36 is SetFunction8.
  LibGrp  =
  (* I *)   BPTR,           NodeGrp,
  (* T *)   FuncArrayPtr,   LibInitProc,    Library,        LibraryPtr,
  (* C *)   base,           close,          expunge,        extFunc,
            nonStd,         open,           reserved,       userDef,
            vectSize,
  (* P *)   AddLibrary,     CloseLibrary,   MakeFunctions,  MakeLibrary,
            OldOpenLibrary, OpenLibrary,    RemLibrary,     SetFunction,
            SetFunction8,   SumLibrary;


|------------------------------------------------------------------------
| devices
|------------------------------------------------------------------------

TYPE

  DevicePtr     = POINTER TO Device;
  Device        = RECORD OF Library END;

  UnitFlags     = ( active, inTask );
  UnitFlagSet   = SET OF UnitFlags;

  UnitPtr       = POINTER TO Unit;
  Unit          = RECORD OF MsgPort;
                    flags    : UnitFlagSet;
                    unit_Pad : SHORTCARD;     | WARNING: added
                    openCnt  : CARDINAL
                  END;

|------------------------------------------------------------------------
| + io
```



```
|------------------------------------------------------------------------

TYPE

  IOFlags       = ( IOquick, IO1, IO2, IO3, IO4, IO5, IO6, IO7 );
  IOFlagSet     = SET OF IOFlags;

  | CARDINAL values for I/O command
  IOCommand   = ( invalid,           reset,              read,
                  write,             update,             clear,
                  stop,              start,              flush,
                  nonstd,            makemecard = $7FFF );

  | SHORTCARD return values for device IO functions and the error field
  IOReturn    = ( badLength = $FC,
                  noCmd     = $FD,
                  aborted   = $FE,
                  openFail  = $FF,
                  ioOk      = 0);

CONST

  | used more easily to extend IOCommand
  nonstdVAL    = CARDINAL( nonstd );

  | offsets into device structures
  abortIO       = -36;
  beginIO       = -30;

  quick         = IOFlagSet:{ IOquick };


TYPE

  IORequestPtr  = POINTER TO IORequest;
  IORequest     = RECORD OF Message
                    device  : DevicePtr;
                    unit    : UnitPtr;
                    command : IOCommand;     | CARDINAL
                    flags   : IOFlagSet;
                    error   : IOReturn;      | SHORTCARD
                  END;

  IOStdReqPtr   = POINTER TO IOStdReq;
  IOStdReq      = RECORD OF IORequest
                    actual,
                    length  : LONGCARD;
                    data    : ANYPTR;
                    offset  : LONGCARD;
                  END;
```

```
| devices

LIBRARY ExecBase BY -432
  PROCEDURE AddDevice( device IN A1 : DevicePtr );

LIBRARY ExecBase BY -450
  PROCEDURE CloseDevice( ioRequest IN A1 : IORequestPtr );

| create an IORequest structure  (V36)
| Allocates memory for and initializes a new IO request block.
| size must be at least Message'SIZE
| port may be the non-NIL return value of CreateMsgPort.
| returns a pointer to the new IORequest block, or NIL.
LIBRARY ExecBase BY -654
  PROCEDURE CreateIORequest( port IN A0 : MsgPortPtr;
                             size IN D0 : LONGCARD ): IORequestPtr;

| DeleteIORequest() - Free a request made by CreateIORequest()  (V36)
LIBRARY ExecBase BY -660
  PROCEDURE DeleteIORequest( iorequest IN A0 : IORequestPtr );

|   OpenDevice
|
| returns a sign extended copy of ioRequest.error.
LIBRARY ExecBase BY -444
  PROCEDURE OpenDevice( REF devName   IN A0 : STRING;
                            unit      IN D0 : LONGCARD;
                            ioRequest IN A1 : IORequestPtr;
                            flags     IN D1 : LONGSET ): IOReturn;

LIBRARY ExecBase BY -438
  PROCEDURE RemDevice( device IN A1 : DevicePtr );

|
|  I/O Functions
|
| !! NOTE !! LONGINT return values are the sign extended version of
| ioRequest.error !
|

LIBRARY ExecBase BY -480
  PROCEDURE AbortIO( ioRequest IN A1 : IORequestPtr );

LIBRARY ExecBase BY -468
  PROCEDURE CheckIO( ioRequest IN A1 : IORequestPtr ): IORequestPtr;

LIBRARY ExecBase BY -456
  PROCEDURE DoIO( ioRequest IN A1 : IORequestPtr ): IOReturn;

LIBRARY ExecBase BY -462
  PROCEDURE SendIO( ioRequest IN A1 : IORequestPtr );

LIBRARY ExecBase BY -474
  PROCEDURE WaitIO( ioRequest IN A1 : IORequestPtr ): IOReturn;
```



```
GROUP

  | new for V36 are CreateIORequest and DeleteIORequest.
  DeviceIOGrp =
  (* I *)       LibGrp,          MsgPortGrp,
  (* T *)       Device,          DevicePtr,        IOFlagSet,
                IORequest,       IORequestPtr,     IOStdReq,
                IOStdReqPtr,     Unit,             UnitFlags,
                UnitFlagSet,     UnitPtr,
  (* C *)       abortIO,         beginIO,          nonStd,
                nonstdVAL,       quick,
  (* P *)       AbortIO,         AddDevice,        CheckIO,
                CloseDevice,     CreateIORequest,  DeleteIORequest,
                DoIO,            OpenDevice,       RemDevice,
                SendIO,          WaitIO;


|-----------------------------------------------------------------------
| semaphores
|-----------------------------------------------------------------------

TYPE

  SemaphoreRequestPtr = POINTER TO SemaphoreRequest;
  SemaphoreRequest    = RECORD OF MinNode
                          waiter : TaskPtr;
                        END;

  SignalSemaphorePtr  = POINTER TO SignalSemaphore;
  SignalSemaphore     = RECORD OF Node
                          nestCount    : INTEGER;
                          waitQueue    : MinList;
                          multipleLink : SemaphoreRequest;
                          owner        : TaskPtr;
                          queueCount   : INTEGER;
                        END;

  SemaphorePtr        = POINTER TO Semaphore;
  Semaphore           = RECORD OF MsgPort
                          bids : INTEGER
                        END;

LIBRARY ExecBase BY -600
  PROCEDURE AddSemaphore( sigSema IN A1 : SignalSemaphorePtr );

LIBRARY ExecBase BY -576
  PROCEDURE AttemptSemaphore(sigSema IN A0 : SignalSemaphorePtr):BOOLEAN;

LIBRARY ExecBase BY -594
  PROCEDURE FindSemaphore( name IN A1 : SysStringPtr ): SignalSemaphore;

LIBRARY ExecBase BY -558
  PROCEDURE InitSemaphore( sigSema IN A0 : SignalSemaphorePtr );
```

```
LIBRARY ExecBase BY -564
  PROCEDURE ObtainSemaphore( sigSema IN A0 : SignalSemaphorePtr );

LIBRARY ExecBase BY -582
  PROCEDURE ObtainSemaphoreList( list IN A0 : ListPtr );

| gain shared access to a semaphore (V36)
| works like you would expect even on older SignalSemaphores
LIBRARY ExecBase BY -678
  PROCEDURE ObtainSemaphoreShared( sigSema IN A0 : SignalSemaphorePtr );

LIBRARY ExecBase BY -540
  PROCEDURE Procure( sema   IN A0 : SemaphorePtr;
                     bidMsg IN A1 : MessagePtr ): BOOLEAN;

LIBRARY ExecBase BY -570
  PROCEDURE ReleaseSemaphore( sigSema IN A0 : SignalSemaphorePtr );

LIBRARY ExecBase BY -588
  PROCEDURE ReleaseSemaphoreList( list IN A0 : ListPtr );

LIBRARY ExecBase BY -606
  PROCEDURE RemSemaphore( sigSema IN A1 : SignalSemaphorePtr );

LIBRARY ExecBase BY -546
  PROCEDURE Vacate( sema IN A0 : SemaphorePtr );

GROUP

  | New for V36 is ObtainSemaphoreShared.
  SemaphoreGrp  =
  (* I *)         MsgPortGrp,
  (* T *)         Semaphore,                SemaphorePtr,
                  SemaphoreRequest,         SemaphoreRequestPtr,
                  SignalSemaphore,          SignalSemaphorePtr,
  (* P *)         AddSemaphore,             AttemptSemaphore,
                  FindSemaphore,            InitSemaphore,
                  ObtainSemaphore,          ObtainSemaphoreList,
                  ObtainSemaphoreShared,    Procure,
                  ReleaseSemaphore,         ReleaseSemaphoreList,
                  RemSemaphore,             Vacate;


|--------------------------------------------------------------------------
| resident
|--------------------------------------------------------------------------

TYPE

  | singleTask and afterDos are new for V36
  ResidentFlags   = ( coldStart,  singleTask, afterDos,   rf3,
                      rf4,        rf5,        rf6,        autoinit );
  ResidentFlagSet = SET OF ResidentFlags;
```



```
  ResidentPri     = NodePri;

  ResidentPtr     = POINTER TO Resident;
  Resident        = RECORD                                    | 26 Bytes
                      matchWord : CARDINAL;
                      matchTag  : ResidentPtr;
                      endSkip   : ANYPTR;
                      flags     : ResidentFlagSet;
                      version   : SHORTCARD;
                      type      : NodeType;
                      priority  : ResidentPri;
                      name      : SysStringPtr;
                      idString  : SysStringPtr;
                      init      : ANYPTR
                    END;

CONST
  matchWord       = $4AFC;  | = the 68000 "ILLEGAL" instruction

LIBRARY ExecBase BY -96
  PROCEDURE FindResident( name IN A1 : SysStringPtr ): ResidentPtr;

LIBRARY ExecBase BY -72
  PROCEDURE InitCode( startClass IN D0 : ResidentFlagSet;
                      version    IN D1 : LONGINT );

LIBRARY ExecBase BY -102
  PROCEDURE InitResident( resident IN A1 : ResidentPtr;
                          segList  IN D1 : BPTR );

GROUP

  ResidentGrp =
  (* I *)       BPTR,             NodeGrp,
  (* T *)       Resident,         ResidentFlags,
                ResidentFlagSet,  ResidentPtr,
  (* C *)       matchWord,
  (* P *)       FindResident,     InitCode,         InitResident;


|--------------------------------------------------------------------------
| execbase
|--------------------------------------------------------------------------

TYPE
  AttnFlags     = ( m68010, m68020, m68030, m68040, m68881,
                    m68882, af6,    af7,    af8,    af9 );
  AttnFlagSet   = SET OF AttnFlags; | 2 Bytes

TYPE
  ExecBaseType  = RECORD OF Library
                    softVer                : CARDINAL;
                    lowMemChkSum           : INTEGER;
                    chkBase                : LONGCARD;
```

```
        | A nice hidingplace for the classic virussians.
        coldCapture,
        coolCapture,
        warmCapture           : PROC;
        sysStkUpper,
        sysStkLower           : ANYPTR;
        maxLocMem             : LONGCARD;

        | Debug() address.
        debugEntry            : PROC;
        debugData,
        alertData             : ANYPTR;
        maxExtMem             : ANYPTR;
        chkSum                : CARDINAL;

        | Interface to these with the AddIntServer Function
        intVects              : ARRAY IntFlags OF IntVector;

        | "There can be only one" on single CPU machines ...
        thisTask              : TaskPtr;

        | All the statistics you get from Exec.
        idleCount,
        dispCount             : LONGCARD;
        quantum,
        elapsed               : CARDINAL;
        sysFlags              : CARDINAL;

        | Interrupt disable nest count
        idNestCnt,

        | Task disable nest count
        tdNestCnt             : SHORTCARD;

        | Attention flags
        attnFlags             : AttnFlagSet;

        | Rescheduling attention
        attnResched           : CARDINAL;

        | resident module array pointer
        resModules            : ANYPTR;

        | task administration
        taskTrapCode,
        taskExceptCode,
        taskExitCode          : PROC;
        taskSigAlloc          : LONGSET;
        taskTrapAlloc         : BITSET;

        | The most often read systemlists (read only in
        | Disable()ed code.
        memList,
        resourceList,
```



```
                    deviceList,
                    intrList,
                    libList,
                    portList,
                    taskReady,
                    taskWait              : List;

                    | the list of pending software interrupts. They have
                    | the priorities of -32, -16, 0, 16 and 32.
                    softInts              : ARRAY [0..4] OF SoftIntList;
                    lastAlert             : ARRAY [0..3] OF LONGINT;

                    | the next two should usualy contain 50 or 60
                    vBlankFrequency       : SHORTCARD;
                    powerSupplyFrequency  : SHORTCARD;
                    semaphoreList         : List;
                    kickMemPtr,
                    kickTagPtr            : ANYPTR;
                    kickCheckSum          : LONGCARD;
| >= V36
                    exPad0                : CARDINAL;
                    exReserved0           : LONGCARD;
                    RamLibPrivate         : ANYPTR;
                    eClockFrequency,                 | readable
                    cacheControl,                    | private
                    taskID,
                    puddleSize,
                    poolThreshold         : LONGCARD;
                    publicPool            : MinList;
                    mmuLock               : ANYPTR;
                    exReserved1           : ARRAY[ 0..11 ] OF SHORTCARD;

| <V36              execBaseReserved,
| <V36              execbaseNewReserved   : ARRAY [0..9] OF SHORTCARD;
                  END;

GROUP

  ExecBaseGrp =
  (* I *)       IntGrp,       LibGrp,       TaskGrp,       BITSET,
  (* T *)       AttnFlags,    AttnFlagSet,  ExecBase,      ExecBasePtr,
                ExecBaseType;


|-----------------------------------------------------------------------
| special
|-----------------------------------------------------------------------

TYPE

  PutChProc    = PROCEDURE( ch IN D0 : CHAR; data IN A3 :   ANYPTR );
```

```
  | new for V37
  StackSwapStructPtr = POINTER TO StackSwapStruct;
  StackSwapStruct    = RECORD
                         lower,             | new lower bound
                         upper,             | new upper bound
                         pointer : ANYPTR;  | stack pointer at switch point
                       END;


LIBRARY ExecBase BY -108
  PROCEDURE Alert( alertNum IN D7  : LONGINT );
| WARNING: used to be parameter 2 :'Params   IN A5 : ANYPTR'

| try to reboot the Amiga (V36)
LIBRARY ExecBase BY -726
  PROCEDURE ColdReboot();

LIBRARY ExecBase BY -114
  PROCEDURE Debug();

LIBRARY ExecBase BY -528
  PROCEDURE GetCC(): BITSET;

LIBRARY ExecBase BY -78
  PROCEDURE InitStruct( initTable IN A1  : ANYPTR;
                        memory    IN A2 : ANYPTR;
                        size      IN D0  : LONGCARD );

LIBRARY ExecBase BY -522
  PROCEDURE RawDoFmt( formatString IN A0 : SysStringPtr;
                      dataStream   IN A1 : ANYPTR;
                      putChProc    IN A2 : PutChProc;
                      putChData    IN A3 : ANYPTR );

| new for V37 and 1:10
LIBRARY ExecBase BY -732
  PROCEDURE StackSwap( newStack IN A0 : StackSwapStructPtr ): ANYPTR;

GROUP

  | New for V36/V37 are ColdReboot and StackSwap.
  SpecialGrp  =
  (* T *)       PutChProc,
  (* P *)       Alert,              ColdReboot,         Debug,
                GetCC,              InitStruct,         RawDoFmt,
                StackSwap,          StackSwapStruct,    StackSwapStructPtr;


|-------------------------------------------------------------------------
| signals
|-------------------------------------------------------------------------

LIBRARY ExecBase BY -330
  PROCEDURE AllocSignal( sigNum IN D0 : TaskSignals ): TaskSignals;
```



```
LIBRARY ExecBase BY -336
  PROCEDURE FreeSignal( sigNum IN D0 : TaskSignals );

LIBRARY ExecBase BY -312
  PROCEDURE SetExcept( newSigs IN D0,
                       sigMask IN D1 : TaskSigSet ): TaskSigSet;

LIBRARY ExecBase BY -306
  PROCEDURE SetSignal( newSigs IN D0,
                       sigMask IN D1 : TaskSigSet ): TaskSigSet;

LIBRARY ExecBase BY -324
  PROCEDURE Signal( task    IN A1 : TaskPtr;
                    signal  IN D0 : TaskSigSet );

LIBRARY ExecBase BY -318
  PROCEDURE Wait( signals IN D0 : TaskSigSet ): TaskSigSet;


GROUP

  SignalGrp     =
          (* I *) TaskGrp,
          (* P *) AllocSignal,        FreeSignal,         SetExcept,
                  SetSignal,          Signal,             Wait;


|-------------------------------------------------------------------------
|      traps
|-------------------------------------------------------------------------

|
| The Cluster runtime system uses several traps and has a traphandler.   Thus,
| TRAPs are off limits for Cluster programs. You can, of course, use TRAPs
| in your own exec tasks.
|

LIBRARY ExecBase BY -342
  PROCEDURE AllocTrap( trapNum IN D0 : LONGINT ): LONGINT;

LIBRARY ExecBase BY -348
  PROCEDURE FreeTrap( trapNum IN D0 : LONGINT );

GROUP

  TrapGrp       =
            (*P*) AllocTrap,          FreeTrap;
```

```
|------------------------------------------------------------------------
| resources
|------------------------------------------------------------------------

TYPE

  ResourcePtr = POINTER TO Resource;
  Resource    = RECORD OF Library END;

LIBRARY ExecBase BY -486
  PROCEDURE AddResource( resource IN A1 : ResourcePtr );

LIBRARY ExecBase BY -492
  PROCEDURE RemResource( resource IN A1 : ResourcePtr );

| WARNING: 2nd parameter "Version IN D0 : LONGCARD" removed, as it does not
| occur in the autodocs.
LIBRARY ExecBase BY -498
  PROCEDURE OpenResource( REF name IN A1 : STRING;
                              vers IN D0 : LONGCARD ): ResourcePtr;

GROUP

  ResourceGrp =
  (* I *)       LibGrp,
  (* T *)       Resource,             ResourcePtr,
  (* P *)       AddResource,          RemResource,       OpenResource;


|------------------------------------------------------------------------
| kickstart
|------------------------------------------------------------------------

LIBRARY ExecBase BY -612
  PROCEDURE SumKickData;

LIBRARY ExecBase BY -618
  PROCEDURE AddMemList(    size       IN D0 : LONGINT;
                           attributes IN D1 : MemReqSet;
                           priority   IN D2 : NodePri;
                           base       IN A0 : ANYPTR;
                       REF name       IN A1 : STRING ): LONGINT;

LIBRARY ExecBase BY -624
  PROCEDURE CopyMem( source IN A0,
                     dest   IN A1  : ANYPTR;
                     size   IN D0  : LONGCARD );

| highest offset for <V36
LIBRARY ExecBase BY -630
  PROCEDURE CopyMemQuick( source IN A0,
                          dest   IN A1  : ANYPTR;
                          size   IN D0  : LONGCARD );
```



```
GROUP

  KickGrp       =
          (*P*) SumKickData,        AddMemList,         CopyMem,
                CopyMemQuick;


|------------------------------------------------------------------------
| private Exec functions. Only in PrivateGrp, not in All.
|------------------------------------------------------------------------

|
| though ExecBase.negSize is 810, the newest functions have not been
| documented at all yet. Please write a thank-you letter to CBM in
| WestChester. You should only use other private functions if you really
| know what to do with them and are writing really low level code.
|

LIBRARY ExecBase BY -60
  PROCEDURE Dispatch();

LIBRARY ExecBase BY -66
  PROCEDURE Exception();

| no docs available
LIBRARY ExecBase BY -774
  PROCEDURE ExecReserved00( nothing IN D0 : ANYPTR );

| no docs available
LIBRARY ExecBase BY -780
  PROCEDURE ExecReserved01( nothing IN D0 : ANYPTR );

| no docs available
LIBRARY ExecBase BY -786
  PROCEDURE ExecReserved02( nothing IN D0 : ANYPTR );

| no docs available
LIBRARY ExecBase BY -792
  PROCEDURE ExecReserved03( nothing IN D0 : ANYPTR );

LIBRARY ExecBase BY -36
  PROCEDURE ExitIntr();

LIBRARY ExecBase BY -504
  PROCEDURE RawIOInit;

LIBRARY ExecBase BY -510
  PROCEDURE RawMayGetChar(): CHAR;

LIBRARY ExecBase BY -516
  PROCEDURE RawPutChar( ch IN D0 : CHAR );


LIBRARY ExecBase BY -48
```

```
  PROCEDURE Reschedule();

LIBRARY ExecBase BY -42
  PROCEDURE Schedule();

LIBRARY ExecBase BY -54
  PROCEDURE Switch();


GROUP

  | ExecReserved?? are new for V36
  PrivateGrp    =
            (*P*) Dispatch,           Exception,          ExecReserved00,
                  ExecReserved01,     ExecReserved02,     ExecReserved03,
                  ExitIntr,           RawIOInit,          RawMayGetChar,
                  RawPutChar,         Reschedule,         Schedule,
                  Switch;


|------------------------------------------------------------------------
| caches
|------------------------------------------------------------------------

TYPE

  Caches    = ( enableI,        | Enable instruction cache
                freezeI,        | Freeze instruction cache
                cacr2,
                clearI,         | Clear instruction cache
                ibe,            | Instruction burst enable
                cacr5, cacr6,
                cacr7,
                enableD,        | 68030 Enable data cache
                freezeD,        | 68030 Freeze data cache
                cacr10,
                clearD,         | 68030 Clear data cache
                dbe,            | 68030 Data burst enable
                writeAllocate,  | 68030 Write-Allocate mode
                                | (must always be set!)
                cacr14,
                cacr15, cacr16, cacr17, cacr18, cacr19,
                cacr20, cacr21, cacr22, cacr23, cacr24,
                cacr25, cacr26, cacr27, cacr28, cacr29,
                cacr30,
                copyBack        | Master enable for copyback caches
              );
  CacheSet  = SET OF Caches;

  CacheDMAFlags = ( continue=1,   | continue a broken up request (Pre)
                    noModify );   | dma didn't modify memory     (Post)
  CacheDMAFlagSet = SET OF CacheDMAFlags;
```



```
| User callable simple cache clearing (V37)
| Flush out the contents of any CPU instruction or data caches.
LIBRARY ExecBase BY -636
  PROCEDURE CacheClearU();

| Cache clearing with extended control (V37)
| WARNING: Cluster still uses the wrong LONGACARD'MAX=2^31-1. Should be
| 2^32-1. length = LONGCARD'MAX (= CAST(LONGCARD,-1)) means clear all
| addresses clearCaches must be IN {clearI,clearD}
LIBRARY ExecBase BY -642
  PROCEDURE CacheClearE( address     IN A0 : ANYPTR;
                         length      IN D0 : LONGCARD;
                         clearCaches IN D1 : CacheSet );

| Instruction & data cache control (V37?)
| Set global cachebits in mask to the values in bits and return old values.
LIBRARY ExecBase BY -648
  PROCEDURE CacheControl( bits IN D0, mask IN D1 : CacheSet ):CacheSet;

| note that length *must* remain a valid variable upto the CachePostDMA
| call.
| WARNING: Autodocs say D0 for flags, fd file says D1...
LIBRARY ExecBase BY -762
  PROCEDURE CachePreDMA(     vaddress IN A0 : ANYPTR;    | virtual address
                         VAR length   IN A1 : LONGCARD;  | to be updated
                             flags    IN D1 : CacheDMAFlagSet; ) : ANYPTR;
| WARNING: Autodocs say D0 for flags, fd file says D1...
LIBRARY ExecBase BY -768
  PROCEDURE CachePostDMA(     vaddress IN A0 : ANYPTR;   | same as for
                                                         | CachePreDMA
                          VAR length   IN A1 : LONGINT;  | same as for
                                                         | CachePreDMA
                              flags    IN D1 : CacheDMAFlagSet);
GROUP

  | All new for V36
  CacheGrp     =
            (*T*) Caches,             CacheSet,           CacheDMAFlags,
                  CacheDMAFlagSet,
            (*P*) CacheClearE,        CacheClearU,        CacheControl,
                  CachePreDMA,        CachePostDMA;


|------------------------------------------------------------------------
| kids. The docs are not ready yet.
| The return value might not be valid.
|------------------------------------------------------------------------

| no doc available, prototype from fd file and thin air.
LIBRARY ExecBase BY -738
  PROCEDURE ChildFree( tid IN D0 : ANYPTR ) : ANYPTR;
```

```
| no doc available, prototype from fd file and thin air.
LIBRARY ExecBase BY -744
  PROCEDURE ChildOrphan( tid IN D0 : ANYPTR ) : ANYPTR;

| no doc available, prototype from fd file and thin air.
LIBRARY ExecBase BY -750
  PROCEDURE ChildStatus( tid IN D0 : ANYPTR ) : ANYPTR;

| no doc available, prototype from fd file and thin air.
LIBRARY ExecBase BY -756
  PROCEDURE ChildWait( tid IN D0 : ANYPTR ) : ANYPTR;

GROUP
  | all new for V36
  ChildGrp      =
          (*P*) ChildFree,          ChildOrphan,        ChildStatus,
                ChildWait;

  | Just for compatibility at the moment.
  ExecIOGrp     = DeviceIOGrp;
  MsgGrp        = MsgPortGrp;

  | New for V36 are Cache, Child, DeviceIO, Resource and SpecialGrp.
  | Groups not included are Const, Private, Func, Ptr and Record.
  All           =
          (*I*) CacheGrp,           ChildGrp,           DeviceIOGrp,
                ExecBaseGrp,
  | compat
                ExecIOGrp,          MsgGrp,

                IntGrp,             KickGrp,            LibGrp,
                ListGrp,            MemGrp,             MsgPortGrp,
                NodeGrp,
  | old         oldMsgPortGrp,
                ResidentGrp,        ResourceGrp,        SemaphoreGrp,
                SignalGrp,          SpecialGrp,         TaskGrp,
                TrapGrp;

  | All Functions in this module (including Cluster coded functions)
  FuncGrp   = AbortIO,                AddDevice,              AddHead,
              AddIntServer,           AddLibrary,             AddMemList,
              AddPort,                AddResource,            AddSemaphore,
              AddTail,                AddTask,                Alert,
              AllocAbs,               Allocate,               AllocEntry,
              AllocMem,               AllocPooled,            AllocSignal,
              AllocTrap,              AllocVec,
              AttemptSemaphore,
              AvailMem,               CacheClearE,            CacheClearU,
              CacheControl,           Cause,                  CheckIO,
              ChildFree,              ChildOrphan,            ChildStatus,
              ChildWait,              CloseDevice,            CloseLibrary,
              ColdReboot,             CopyMem,                CopyMemQuick,
              CreateIORequest,        CreateMsgPort,
              CreatePrivatePool,      Deallocate,             Debug,
              DeleteIORequest,        DeleteMsgPort,
```



```
              DeletePrivatePool,
              Disable,                Dispatch,               DoIO,
              Enable,                 Enqueue,                Exception,
              ExitIntr,               FindName,
              FindPort,               FindResident,           FindSemaphore,
              FindTask,               Forbid,                 FreeEntry,
              FreeMem,                FreePooled,             FreeSignal,
              FreeTrap,               FreeVec,                GetCC,
              GetMsg,                 InitCode,               InitResident,
              InitSemaphore,          InitStruct,             Insert,
              MakeFunctions,          MakeLibrary,            NewList,
              ObtainSemaphore,        ObtainSemaphoreList,
              ObtainSemaphoreShared,
              OldOpenLibrary,         OpenDevice,             OpenLibrary,
              OpenResource,           Permit,                 Procure,
              PutMsg,                 RawDoFmt,               RawIOInit,
              RawMayGetChar,          RawPutChar,
              ReleaseSemaphore,
              ReleaseSemaphoreList,   RemDevice,              RemHead,
              RemIntServer,           RemLibrary,             Remove,
              RemPort,                RemResource,            RemSemaphore,
              RemTail,                RemTask,                ReplyMsg,
              Reschedule,             Schedule,               SendIO,
              SetExcept,              SetFunction,            SetFunction8,
              SetIntVector,           SetSignal,              SetSR,
              SetTaskPri,             Signal,                 StackSwap,
              SumKickData,            SumLibrary,             SuperState,
              Supervisor,             Switch,                 TypeOfMem,
              UserState,              Vacate,                 Wait,
              WaitIO,                 WaitPort;

  PtrGrp    = DevicePtr,              ExecBasePtr,            FuncArrayPtr,
              InterruptPtr,           IntFuncPtr,             IntVectorPtr,
              IORequestPtr,           IOStdReqPtr,            LibraryPtr,
              ListPtr,                MemChunkPtr,            MemEntryPtr,
              MemHeaderPtr,           MemListPtr,             MessagePtr,
              MinListPtr,             MinNodePtr,             MsgPortPtr,
              NodePtr,                ResidentPtr,            ResourcePtr,
              SemaphorePtr,           SemaphoreRequestPtr,
              SignalSemaphorePtr,
              SoftIntListPtr,         TaskPtr,                UnitPtr;

  RecordGrp = Device,                 ExecBaseType,           Interrupt,
              IntVector,              IORequest,              IOStdReq,
              Library,                List,                   MemChunk,
              MemEntry,               MemHeader,              MemList,
              Message,                MinList,                MinNode,
              MsgPort,                Node,                   Resident,
              Resource,               Semaphore,
              SemaphoreRequest,
              SignalSemaphore,        SoftIntList,            Task,
              Unit;

END Exec.
```

```
TABLE OF CONTENTS

FUNCTION                GRP                V36?

AbortIO                 DeviceIO
AddDevice               DeviceIO
AddHead                 List
AddIntServer            Int
AddLibrary              Lib
AddMemList              Kick
AddPort                 MsgPort
AddResource             Resource
AddSemaphore            Semaphore
AddTail                 List
AddTask                 Task
Alert                   Special
AllocAbs                Mem
Allocate                Mem
AllocEntry              Mem
AllocMem                Mem
AllocPooled             Mem                V36
AllocSignal             Signal
AllocTrap               Trap
AllocVec                Mem                V36
AttemptSemaphore        Semaphore
AvailMem                Mem
CacheClearE             Cache              V36
CacheClearU             Cache              V36
CacheControl            Cache              V36
Cause                   Int
CheckIO                 DeviceIO
ChildFree               Child              V36
ChildOrphan             Child              V36
ChildStatus             Child              V36
ChildWait               Child              V36
CloseDevice             DeviceIO
CloseLibrary            Lib
ColdReboot              Special            V36
CopyMem                 Kick
CopyMemQuick            Kick
CreateIORequest         DeviceIO           V36
|CreateMsgPort           MsgPort            V36
CreatePrivatePool       Mem                V36
Deallocate              Mem
Debug                   Special
DeleteIORequest         DeviceIO           V36
|DeleteMsgPort           MsgPort            V36
DeletePrivatePool       Mem                V36
Disable                 Int
Dispatch                Private
DoIO                    DeviceIO
Enable                  Int
Enqueue                 List
Exception               Private
```



```
ExecReserved00          Private            V36
ExecReserved01          Private            V36
ExecReserved02          Private            V36
ExecReserved03          Private            V36
ExitIntr                Private
FindName                List
FindPort                MsgPort
FindResident            Resident
FindSemaphore           Semaphore
FindTask                Task
Forbid                  Int
FreeEntry               Mem
FreeMem                 Mem
FreePooled              Mem                V36
FreeSignal              Signal
FreeTrap                Trap
FreeVec                 Mem                V36
GetCC                   Special
GetMsg                  MsgPort
InitCode                Resident
InitResident            Resident
InitSemaphore           Semaphore
InitStruct              Special
Insert                  List
MakeFunctions           Lib
MakeLibrary             Lib
ObtainSemaphore         Semaphore
ObtainSemaphoreList     Semaphore
ObtainSemaphoreShared   Semaphore          V36
OldOpenLibrary          Lib
OpenDevice              DeviceIO
OpenLibrary             Lib
OpenResource            Resource
Permit                  Int
Procure                 Semaphore
PutMsg                  MsgPort
RawDoFmt                Special
RawIOInit               Private
RawMayGetChar           Private
RawPutChar              Private
ReleaseSemaphore        Semaphore
ReleaseSemaphoreList    Semaphore
RemDevice               DeviceIO
RemHead                 List
RemIntServer            Int
RemLibrary              Lib
Remove                  List
RemPort                 MsgPort
RemResource             Resource
RemSemaphore            Semaphore
RemTail                 List
RemTask                 Task
ReplyMsg                MsgPort
Reschedule              Private
Schedule                Private
```

```
SendIO              DeviceIO
SetExcept           Signal
SetFunction         Lib
SetFunction8        Lib                V36
SetIntVector        Int
SetSignal           Signal
SetSR               Int
SetTaskPri          Task
Signal              Signal
StackSwap           Special            V36
SumKickData         Kick
SumLibrary          Lib
SuperState          Int
Supervisor          Int
Switch              Private
TypeOfMem           Mem
UserState           Int
Vacate              Semaphore
Wait                Signal
WaitIO              DeviceIO
WaitPort            MsgPort


NodeType
MemReqs
TaskFlags
TaskState
MsgPortAction
LibFlags
UnitFlags
ResidentFlags
AttnFlags
Caches
```


8.15 Expansion

```
(* $A- *)
DEFINITION MODULE Expansion;

FROM System IMPORT LONGSET, Register, SysStringPtr;
FROM Dos    IMPORT DeviceNodePtr, DosEnvec, DosEnvecPtr;
FROM Exec   IMPORT Interrupt, Library, LibraryPtr, List, Node,
                   SignalSemaphore;


TYPE
  ERomTypeFlags       = (er_mem0,er_mem1,er_mem2,chainedConfig,
                         diagValid,memList,er_type0,er_type1);
  ERomTypeFlagSet     = SET OF ERomTypeFlags;
  ExpansionRomFlags   = (erf0,erf1,erf2,erf3,zorro3,extended,noShutup,
                         memSpace);
  ExpansionRomFlagSet = SET OF ExpansionRomFlags;
  ExpansionRom        = RECORD
                          type           : ERomTypeFlagSet;
                          product        : SHORTCARD;
                          flags          : ExpansionRomFlagSet;
                          reserved03     : SHORTCARD;
                          manufacturer   : CARDINAL;
                          serialNumber   : LONGCARD;
                          initDiagVec    : CARDINAL;
                          reserved0c     : SHORTCARD;
                          reserved0d     : SHORTCARD;
                          reserved0e     : SHORTCARD;
                          reserved0f     : SHORTCARD
                        END;

  InterruptFlags      = (if0,intena,if2,reset,int2pend,int6pend,int7pend,
                         interrupting);
  InterruptFlagSet    = SET OF InterruptFlags;
  ExpansionControl    = RECORD
                          interrupt   : InterruptFlagSet;
                          z3HighBase  : SHORTCARD;
                          baseAddress : SHORTCARD;
                          shutup      : SHORTCARD;
                          reserved14  : SHORTCARD;
                          reserved15  : SHORTCARD;
                          reserved16  : SHORTCARD;
                          reserved17  : SHORTCARD;
                          reserved18  : SHORTCARD;
                          reserved19  : SHORTCARD;
                          reserved1a  : SHORTCARD;
                          reserved1b  : SHORTCARD;
                          reserved1c  : SHORTCARD;
                          reserved1d  : SHORTCARD;
                          reserved1e  : SHORTCARD;
                          reserved1f  : SHORTCARD;
                        END;
```

```
CONST
  slotSize             = $10000;
  slotMask             = $FFFF;
  slotShift            = 16;

  expansionBase        = $00E80000;
  z3ExpansionBase      = $FF000000;
  expansionSize        = $00080000;
  expansionSlots       = 8;
  memoryBase           = $00200000;
  memorySize           = $00800000;
  memorySlots          = 128;
  z3ConfigArea         = $40000000;
  z3ConfigAreaEnd      = $7FFFFFFF;
  z3SizeGranularity    = $00080000;

TYPE
  ConfigFlags          = (cf0,cf1,cf2,cf3,configTime,bindTime,byteWide,
                          wordWide);
  ConfigFlagSet        = SET OF ConfigFlags;
  DiagArea             = RECORD
                           config        : ConfigFlagSet;
                           flags         : SHORTCARD;
                           size          : CARDINAL;
                           diagPoint     : CARDINAL;
                           bootPoint     : CARDINAL;
                           name          : CARDINAL;
                           reserved01    : CARDINAL;
                           reserved02    : CARDINAL
                         END;

CONST
  busWidth             = ConfigFlagSet:{byteWide,wordWide};
  nibbleWide           = ConfigFlagSet:{};

  bootTime             = ConfigFlagSet:{configTime,bindTime};
  never                = ConfigFlagSet:{};

TYPE
  ConfigDevFlags       = (shutup,configMe,badMemory,cdf3,cdf4,cdf5,cdf6,
                          cdf7);
  ConfigDevFlagSet     = SET OF ConfigDevFlags;
  ConfigDevPtr         = POINTER TO ConfigDev;
  ConfigDev            = RECORD OF Node
                           flags       : ConfigDevFlagSet;
                           pad         : SHORTCARD;
                           rom         : ExpansionRom;
                           boardAddr   : ANYPTR;
                           boardSize   : LONGCARD;
                           slotAddr    : CARDINAL;
                           slotSize    : CARDINAL;
                           driver      : ANYPTR;
                           nextCD      : ConfigDevPtr;
                           unused      : ARRAY [0..3] OF LONGINT;
                         END;
```



```
TYPE
  CurrentBindingPtr    = POINTER TO CurrentBinding;
  CurrentBinding       = RECORD
                           configDev     : ConfigDevPtr;
                           fileName      : SysStringPtr;
                           productString : SysStringPtr;
                           toolTypes     : POINTER TO SysStringPtr;
                         END;

  BootNodeFlags        = (startProc,bnf1,bnf2,bnf3,bnf4,bnf5,bnf6,bnf7,
                          bnf8);
  BootNodeFlagSet      = SET OF BootNodeFlags;

  BootNodePtr          = POINTER TO BootNode;
  BootNode             = RECORD OF Node
                           flags       : BootNodeFlagSet;
                           deviceNode  : ANYPTR
                         END;

  ExpansionBaseFlags   = (clogged,shortmem,badmem,dosflag,kickback33,
                          kickback36,silentStart);
  ExpansionBaseFlagSet= SET OF ExpansionBaseFlags;

  ExpansionBasePtr     = POINTER TO ExpansionBaseType;
  ExpansionBaseType    = RECORD OF Library
                           flags       : ExpansionBaseFlagSet;
                           private01   : SHORTCARD;
                           private02   : LONGCARD;
                           private03   : LONGCARD;
                           private04   : CurrentBinding;
                           private05   : List;
                           mountList   : List;
                         END;

CONST
  errOk            = 0;
  errLastboard     = 40;
  errNoexpansion   = 41;
  errNomemory      = 42;
  errNoboard       = 43;
  errBadmem        = 44;

TYPE
  ParameterPktPtr = POINTER TO ParameterPkt;
  ParameterPkt    = RECORD
                      dosName   : SysStringPtr;
                      execName  : SysStringPtr;
                      unit      : LONGCARD;
                      flags     : LONGSET;
                      env       : DosEnvec;
                    END;

VAR ExpansionBase : ExpansionBasePtr;
```

```
LIBRARY ExpansionBase BY -30
  PROCEDURE AddConfigDev(configDev IN A0 : ConfigDevPtr);

LIBRARY ExpansionBase BY -36
  PROCEDURE AddBootNode(bootPri    IN D0 : SHORTINT;
                        flags      IN D1 : LONGSET;
                        deviceNode IN A0 : DeviceNodePtr;
                        configDev  IN A1 : ConfigDevPtr):BOOLEAN;

LIBRARY ExpansionBase BY -42
  PROCEDURE AllocBoardMem(slotSpec IN D0 : LONGINT);

LIBRARY ExpansionBase BY -78
  PROCEDURE FreeBoardMem(startSlot IN D0 : LONGCARD;
                         slotSpec  IN D1 : LONGCARD);


LIBRARY ExpansionBase BY -54
  PROCEDURE AllocExpansionMem(numSlots   IN D0 : LONGINT;
                              slotAlign  IN D1 : LONGINT;
                              slotOffset IN D2 : LONGINT):LONGINT;

LIBRARY ExpansionBase BY -90
  PROCEDURE FreeExpansionMem(startSlot IN D0 : LONGINT;
                             numSlots  IN D1 : LONGINT);

LIBRARY ExpansionBase BY -60
  PROCEDURE ConfigBoard(board     IN A0 : ANYPTR;
                        configDev IN A1 : ConfigDevPtr);

LIBRARY ExpansionBase BY -66
  PROCEDURE ConfigChain(baseAddr IN A0 : ANYPTR);


LIBRARY ExpansionBase BY -48
  PROCEDURE AllocConfigDev():ConfigDevPtr;

LIBRARY ExpansionBase BY -72
  PROCEDURE FindConfigDev(oldConfigDev IN A0 : ConfigDevPtr;
                          manufacturer IN D0 : LONGINT;
                          product      IN D1 : LONGINT):ConfigDevPtr;

LIBRARY ExpansionBase BY -84
  PROCEDURE FreeConfigDev(configDev IN A0 : ConfigDevPtr);

LIBRARY ExpansionBase BY -108
  PROCEDURE RemConfigDev(configDev IN A0 : ConfigDevPtr);

LIBRARY ExpansionBase BY -96
  PROCEDURE ReadExpansionByte(board  IN A0 : ANYPTR;
                              offset IN D0 : LONGINT):SHORTINT;
```



```
LIBRARY ExpansionBase BY -114
  PROCEDURE WriteExpansionByte(board  IN A0 : ANYPTR;
                               offset IN D0 : LONGCARD;
                               byte   IN D1 : SHORTCARD);


LIBRARY ExpansionBase BY -102
  PROCEDURE ReadExpansionRom(board     IN A0 : ANYPTR;
                             configDev IN A1 : ConfigDevPtr);


LIBRARY ExpansionBase BY -120 PROCEDURE ObtainConfigBinding;

LIBRARY ExpansionBase BY -126 PROCEDURE ReleaseConfigBinding;


LIBRARY ExpansionBase BY -132
  PROCEDURE SetCurrentBinding(currentBinding IN A0 : CurrentBindingPtr;
                              bindingSize    IN D0 : INTEGER);

LIBRARY ExpansionBase BY -138
  PROCEDURE GetCurrentBinding(currentBinding IN A0 : CurrentBindingPtr;
                              bindingSize    IN D0 : INTEGER):LONGINT;


LIBRARY ExpansionBase BY -144
  PROCEDURE MakeDosNode(parmPacket IN A0 : ParameterPktPtr):DeviceNodePtr;

LIBRARY ExpansionBase BY -150
  PROCEDURE AddDosNode(bootPri    IN D0 : SHORTINT;
                       flags      IN D1 : LONGSET;
                       deviceNode IN A0 : DeviceNodePtr):BOOLEAN;

END Expansion.
```

8.16 FileSystemResource

```
DEFINITION MODULE FileSystemResource;
(* $A- *)

|S. Herr, 01.10.1992

FROM Exec   IMPORT LibraryPtr,Node,List;
FROM System IMPORT Regs,SysStringPtr,LONGSET,BPTR;
FROM Dos    IMPORT FileLockPtr,BSTR,ProcessId,FileSysStartupMsgPtr;

TYPE
  FileSysResource = RECORD OF Node
                      creator       : SysStringPtr;
                      fileSysEntries : List;
                    END;

  FileSysEntry    = RECORD OF Node
                      dosType    : LONGCARD;
                      version    : LONGCARD;
                      patchFlags : LONGSET;
                      type       : LONGCARD;
                      task       : ProcessId;
                      lock       : FileLockPtr;
                      handler    : BSTR;
                      stackSize  : LONGCARD;
                      priority   : LONGINT;
                      startup    : BPTR;
                      segList    : BPTR;
                      globalVec  : FileSysStartupMsgPtr;
                    END;


VAR
  FileSysBase : LibraryPtr;


GROUP
  All = FileSysResource,FileSysEntry,FileSysBase;

END FileSystemResource.
```


8.17 GadTools

```
DEFINITION MODULE GadTools;

| WB 18 Aug 1992 Datei übernommen.

(* $A- *)

FROM Exec       IMPORT LibraryPtr, MsgPortPtr, ListPtr;
FROM Intuition  IMPORT IDCMPFlagSet, IDCMPFlags,  gaTB, layoTB, pgaTB,
                       strgTB,ScreenPtr, WindowPtr, RequesterPtr,
                       GadgetPtr,IntuiMessagePtr, ActivationFlagSet;
FROM Graphics   IMPORT TextAttrPtr, RastPortPtr;
FROM System     IMPORT LONGSET, Register, SysStringPtr;
FROM Utility    IMPORT StdTags, tagUser, HookPtr;

TYPE

  GadgetKind = (  generic,  button,   checkbox,   integer,
                  listview, mx,       number,     cycle,
                  palette,  scroller,             slider=11,
                  string,   text,
                  makeMeLong = $10000);

CONST
  arrowIDCMP    = IDCMPFlagSet:{gadgetUp,gadgetDown,intuiTicks,
                                mouseButtons};
  buttonIDCMP   = IDCMPFlagSet:{gadgetUp};
  checkBoxIDCMP = IDCMPFlagSet:{gadgetUp};
  integerIDCMP  = IDCMPFlagSet:{gadgetUp};
  listViewIDCMP = IDCMPFlagSet:{gadgetUp,gadgetDown,mouseMove}+arrowIDCMP;
  mxIDCMP       = IDCMPFlagSet:{gadgetDown};
  numberIDCMP   = IDCMPFlagSet:{};
  cycleIDCMP    = IDCMPFlagSet:{gadgetUp};
  paletteIDCMP  = IDCMPFlagSet:{gadgetUp};

  |  Use arrowIDCMP+scrollerIDCMP if your scrollers have arrows:
  scrollerIDCMP = IDCMPFlagSet:{gadgetUp,gadgetDown,mouseMove};
  sliderIDCMP   = IDCMPFlagSet:{gadgetUp,gadgetDown,mouseMove};
  stringIDCMP   = IDCMPFlagSet:{gadgetUp};

  textIDCMP     = IDCMPFlagSet:{};

|------------------------------------------------------------------------

CONST
  interWidth  = 8;
  interHeight = 4;

|------------------------------------------------------------------------
```

```
TYPE
  VisualInfo         = HIDDEN;

  NewGadgetFlags     = ( placeTextLeft,
                         placeTextRight,
                         placeTextAbove,
                         placeTextBelow,
                         placeTextIn,
                         highLabel,
                         rsvd31=31 );

  NewGadgetFlagSet   = SET OF NewGadgetFlags;

  NewGadget          = RECORD
                         leftEdge, topEdge : INTEGER;
                         width, height     : INTEGER;
                         gadgetText        : SysStringPtr;
                         textAttr          : TextAttrPtr;
                         gadgetID          : CARDINAL;
                         flags             : NewGadgetFlagSet;
                         visualInfo        : VisualInfo;
                         userData          : ANYPTR;
                       END;


TYPE
  NewMenuType        = ( end   =   0,
                         title =   1, item      =   2, sub      =   3,
                         image = 128, imageItem = 130, imageSub = 131);

TYPE
  NewMenuFlags       = (menuDisabled = 0,itemDisabled = 4,checkIt    = 0,
                        menuToggle   = 3,checked      = 8,makeMeWord = 15);
  NewMenuFlagSet     = SET OF NewMenuFlags;

  NewMenuPtr         = POINTER TO NewMenu;
  NewMenu            = RECORD
                         type          : NewMenuType;
                         label         : SysStringPtr;
                         commKey       : SysStringPtr;
                         flags         : NewMenuFlagSet;
                         mutualExclude : LONGSET;
                         userData      : ANYPTR;
                       END;

  NewMenuArray       = ARRAY OF NewMenu;

CONST
  barLabel = SysStringPtr(-1);

TYPE
  GTMenuPtr          = POINTER TO GTMenu;
  GTMenu             = RECORD OF Intuition.Menu
                         userData : ANYPTR;
                       END;
```



```
  GTMenuItemPtr      = POINTER TO GTMenuItem;
  GTMenuItem         = RECORD OF Intuition.MenuItem
                         userData : ANYPTR;
                       END;

  | Return codes through GTMN_ErrorCode tag

  CM2ndErr           = (menuTrimmed = 1,    | Too many menus, items, or
                                            | subitems,
                                            | menu is trimmed
                        menuInvalid = 2,    | Invalid NewMenu array
                        nomem       = 3,    | Out of memory
                        mmlong      = $10000 );

CONST
  gtTB = tagUser + $80000; | $80080000, base for GadTools Tagvalues


TYPE
  | Callback for number calculation  before display
  DispFunc           = PROCEDURE( gad IN A0: GadgetPtr;
                                  org IN D0: INTEGER ): LONGINT;

  GadgetTags =
    TAGS OF StdTags
      left             = gaTB+$01   : LONGINT;
      underScore       = gtTB+64    : LONGCHAR;  | the character that
                                                 | precedes the
                                    |
                                    | character to be underlined
                                    | (to indicate a shortcut).
                                    | Can be used for all classes.

      |  ButtonTags

      buDisabled       = gaTB+14    : LONGBOOL;       | =gaDisabled

      |  CheckBoxTags

      cbChecked        = gtTB+4     : LONGBOOL;       | condition of the
                                                      | checkbox
      cbDisabled       = gaTB+14    : LONGBOOL;       | =gaDisabled

      |  CycleTags

      cyLabels         = gtTB+14    : POINTER TO ARRAY OF SysStringPtr;
                                    |
                                    | NIL-terminated array of labels

      cyActive         = gtTB+15    : LONGCARD;  | active one in the cycle
      cyDisabled       = gaTB+14    : LONGBOOL;  | =gaDisabled

      |  IntegerTags

      inNumber         = gtTB+47    : LONGCARD; | initial number in gadget
```

```
inMaxChars                     : LONGCARD; | max number of digits (10)
inExitHelp      = strgTB+$13 : LONGBOOL; | as in Intution.StrgTags
inTabCycle      = gaTB+$24   : LONGBOOL; | =gaTabCycle
inDisabled      = gaTB+14    : LONGBOOL; | =gaDisabled

|  ListViewTags

lvTop           = gtTB+5   : LONGCARD; | Top visible one in listview
lvLabels        = gtTB+6   : ListPtr;  | List to display in listview
lvReadOnly      = gtTB+7   : LONGBOOL; | listview is to be read-only
lvScrollWidth   = gtTB+8   : LONGCARD; | UWORD; Width of scrollbar
lvShowSelected  = gtTB+$35 : GadgetPtr;| show selected entry
                           |
                           | beneath listview. Pass NULL for
                           | display-only,
                           | or a POINTER to a gadtools string gadget
                           | you've created

lvSelected      = gtTB+$36   : LONGCARD;       | CARDINAL
                             |
                             | SET ordinal number of selected entry
                             | in the list
lvSpacing       = layoTB+$2  : LONGCARD; | As in Intuition.LayoutTags

|  MxTags

mxLabels        = gtTB+9     : POINTER TO ARRAY OF SysStringPtr;
                             |
                             | NIL-terminated array of labels

mxActive        = gtTB+10    : LONGCARD;   | Active one in mx gadget
mxSpacing       = gtTB+61    : LONGCARD;   | Added to font height to
                             |
                             | figure out spacing between mx choices.
                             | Use instead of LayoutTags.spacing for
                             | mx gadgets.

|  NumberTags

nmNumber        = gtTB+13    : LONGINT;        | Number to display
nmBorder        = gtTB+58    : LONGINT;
                             |
                             | Put a border around Number-display
                             | gadgets

|  PaletteTags

paDisabled      = gaTB+$E    : LONGBOOL; | same as gaDisabled
paDepth         = gtTB+16    : LONGCARD; | bitplanes in palette
paColor         = gtTB+17    : LONGCARD; | Palette color
paColorOffset   = gtTB+18    : LONGCARD; | 1st color to use in
                                         | palette
paIndicatorW    = gtTB+19    : LONGCARD; | curr. color indicator
                                         | width
```



```
paIndicatorH    = gtTB+20    : LONGCARD; | cc. indicator height

|  ScrollerTags

scTop           = gtTB+21    : LONGINT;  | Top visible in scroller
scTotal         = gtTB+22    : LONGINT;  | Total in scroller area
scVisible       = gtTB+23    : LONGINT;  | Number visible in scroller
|scOverlap      = gtTB+24    ;           | Unused
scArrows        = gtTB+59    : LONGCARD; | size of arrows for scroller
scFreedom       = pgaTB+1    : LONGINT;  | =pgaFreedom
scImmediate     = pgaTB+$15  : LONGINT;  | =pgaImmediate
scRelVerify     = gaTB+$16   : LONGINT;  | =gaRelVerify
scDisabled      = gaTB+$E    : LONGBOOL; | =gaDisabled

|  SliderTags

slMin           = gtTB+38    : LONGINT;        | Slider min value
slMax           = gtTB+39    : LONGINT;        | Slider max value
slLevel         = gtTB+40    : LONGINT;        | Slider level
slMaxLevelLen   = gtTB+41    : LONGCARD;       | Max length of
                                               | printed level
slLevelFormat   = gtTB+42    : SysStringPtr;   | Format string
                                               | for level
slLevelPlace    = gtTB+43    : NewGadgetFlags; | place of level
slDispFunc      = gtTB+44    : DispFunc;
                             |
                             | Callback for number calc. before
                             | display

slFreedom       = pgaTB+$1   : LONGINT;        | =pgaFreedom
slImmediate     = gaTB+$15   : LONGINT;        | =gaImmediate
slRelVerify     = gaTB+$16   : LONGINT;        | =gaRelVerify
slDisabled      = gaTB+$E    : LONGBOOL;       | =gaDisabled

|  StringTags

stString        = gtTB+45    : SysStringPtr;
                             |
                             | String gadget's displayed string

stMaxChars      = gtTB+46    : LONGCARD;       | Max length of string
stDisabled      = gaTB+$E    : LONGBOOL;       | =gaDisabled
stExitHelp      = strgTB+$13 : LONGBOOL;       | =strgExitHelp
stTabCycle      = gaTB+$24   : LONGBOOL;       | =gaTabCycle
stEditHook      = gtTB+$37   : HookPtr;        | see StringExtend
stJustification = strgTB+$10 : ActivationFlagSet;
                             |
                             | choose one of stringLeft (def),
                             | stringRight, stringCenter.

stReplaceMode   = strgTB+$0D : LONGBOOL;       | =strgReplaceMode
|  TextTags
txText          = gtTB+11    : SysStringPtr;| Text to display
txCopyText      = gtTB+12    : LONGINT;     | Copy text label instead
                                            | of referencing it
```

```
    txBorder        = gtTB+57    : LONGINT;     | Put a border around

  END;  | of GadgetTags

GadgetTagListPtr  = POINTER TO GadgetTagList;
GadgetTagList     = ARRAY OF GadgetTags;


 |
 | Tags for calls to CreateMenus
 |
CreateMenuTags =
  TAGS OF StdTags
    frontPen        = gtTB+50    : LONGCARD;  | MenuItem text pen color
    fullMenu        = gtTB+62    : LONGBOOL;  | Asks CreateMenus() to
                                 |
                                 | validate the completeness of the menu
                                 | structure

    secondaryErr    = gtTB+63    : POINTER TO CM2ndErr;
                                 |
                                 | pointer to a CM2ndErr to receive error
                                 | reports from CreateMenus()
  END;

CrMenuTagListPtr  = POINTER TO CrMenuTagList;
CrMenuTagList     = ARRAY OF CreateMenuTags;

 |
 | Tags for calls to LayoutMenus
 |
LayoutMenuTags =
  TAGS OF StdTags
    textAttr        = gtTB+49  : TextAttrPtr; | GTMenuItem font TextAttr
    menu            = gtTB+60  : GTMenuPtr;   | Pointer to GTMenu for use
                                              | by  LayoutMenuItems()
  END;

LaMenuTagListPtr  = POINTER TO LaMenuTagList;
LaMenuTagList     = ARRAY OF LayoutMenuTags;

 |
 | Tags for calls to DrawBevelBox
 |
BevelBoxTags =
  TAGS OF StdTags
    recessed        = gtTB+51    : LONGINT;    | Make BevelBox recessed
    visualInfo                   : VisualInfo; | result of VisualInfo call
  END;

BevBoxTagListPtr  = POINTER TO BevBoxTagList;
BevBoxTagList     = ARRAY OF BevelBoxTags;


 | gtTB+25 through gtTB+37 are reserved
```



```
  | gtTB+65 on up reserved for future use


  GadToolsLibraryPtr = LibraryPtr;


VAR
  GadToolsBase : GadToolsLibraryPtr;


LIBRARY GadToolsBase BY -114
  PROCEDURE CreateContext( VAR GadgetPtr IN A0 : GadgetPtr ): GadgetPtr;

LIBRARY GadToolsBase BY -30
  PROCEDURE CreateGadgetTags(    kind     IN D0 : GadgetKind;
                                 previous IN A0 : GadgetPtr;
                             REF newgad   IN A1 : NewGadget;
                                 taglist  IN A2 : LIST OF GadgetTags):GadgetPtr;
LIBRARY GadToolsBase BY -30
  PROCEDURE CreateGadget(    kind     IN D0 : GadgetKind;
                             previous IN A0 : GadgetPtr;
                         REF newgad   IN A1 : NewGadget;
                             taglist  IN A2 : GadgetTagListPtr ): GadgetPtr;
LIBRARY GadToolsBase BY -48
  PROCEDURE CreateMenusTags(REF newmenu IN A0 : NewMenuArray;
                                taglist IN A1 : LIST OF CreateMenuTags):GTMenuPtr;
LIBRARY GadToolsBase BY -48
  PROCEDURE CreateMenus( REF newmenu IN A0 : NewMenuArray;
                             taglist IN A1 : CrMenuTagListPtr ): GTMenuPtr;
LIBRARY GadToolsBase BY -120
  PROCEDURE DrawBevelBox( rport   IN A0 : RastPortPtr;
                          left    IN D0,
                          top     IN D1,
                          width   IN D2,
                          height  IN D3 : INTEGER;
                          taglist IN A1 : LIST OF BevelBoxTags );

LIBRARY GadToolsBase BY -36
  PROCEDURE FreeGadgets( glist IN A0 : GadgetPtr );

LIBRARY GadToolsBase BY -54
  PROCEDURE FreeMenus( menu IN A0 : GTMenuPtr );

LIBRARY GadToolsBase BY -132
  PROCEDURE FreeVisualInfo( vi IN A0 : VisualInfo );

LIBRARY GadToolsBase BY -126
  PROCEDURE GetVisualInfo( screen  IN A0 : ScreenPtr;
                           taglist IN A1 : LIST OF StdTags ): VisualInfo;

LIBRARY GadToolsBase BY -90
  PROCEDURE GT_BeginRefresh( win IN A0 : WindowPtr );
```

```
LIBRARY GadToolsBase BY -96
  PROCEDURE GT_EndRefresh( win      IN A0 : WindowPtr;
                           complete IN D0 : BOOLEAN);

LIBRARY GadToolsBase BY -102
  PROCEDURE GT_FilterIMsg( imsg IN A1 : IntuiMessagePtr ): IntuiMessagePtr;

LIBRARY GadToolsBase BY -72
  PROCEDURE GT_GetIMsg( intuiport IN A0 : MsgPortPtr ): IntuiMessagePtr;

LIBRARY GadToolsBase BY -108
  PROCEDURE GT_PostFilterIMsg(modimsg IN A1 : IntuiMessagePtr):IntuiMessagePtr;

LIBRARY GadToolsBase BY -84
  PROCEDURE GT_RefreshWindow( win IN A0 : WindowPtr;
                              req IN A1 : RequesterPtr );

LIBRARY GadToolsBase BY -78
  PROCEDURE GT_ReplyIMsg( imsg IN A1 : IntuiMessagePtr );

LIBRARY GadToolsBase BY -42
  PROCEDURE GT_SetGadgetAttrs( gad     IN A0 : GadgetPtr;
                               win     IN A1 : WindowPtr;
                               req     IN A2 : RequesterPtr;
                               taglist IN A3 : LIST OF GadgetTags );

LIBRARY GadToolsBase BY -60
  PROCEDURE LayoutMenuItems(menuitem IN A0 : GTMenuItemPtr;
                            vi       IN A1 : VisualInfo;
                            taglist  IN A2 : LIST OF LayoutMenuTags):BOOLEAN;

LIBRARY GadToolsBase BY -66
  PROCEDURE LayoutMenus(menu    IN A0 : GTMenuPtr;
                        vi      IN A1 : VisualInfo;
                        taglist IN A2 : LIST OF LayoutMenuTags):BOOLEAN;


GROUP

  ArrayGrp =      NewMenuArray;

  ConstGrp =      arrowIDCMP,        barLabel,          buttonIDCMP,
                  checkBoxIDCMP,     cycleIDCMP,        CM2ndErr,
                  gtTB,
                  integerIDCMP,      interHeight,       interWidth,
                  gaTB,              pgaTB,             strgTB,
                  layoTB,            listViewIDCMP,     mxIDCMP,
                  numberIDCMP,       paletteIDCMP,
                  scrollerIDCMP,     sliderIDCMP,       stringIDCMP,
                  textIDCMP;

  EnumGrp =       GadgetKind,        NewGadgetFlags,    NewMenuType;

  PointerGrp =    GTMenuItemPtr,     GTMenuPtr,         GadToolsLibraryPtr,
                  NewMenuPtr,        VisualInfo;
```



```
  ProcGrp =       CreateContext,      CreateGadget,       CreateMenus,
                  DrawBevelBox,       FreeGadgets,        FreeMenus,
                  FreeVisualInfo,     GetVisualInfo,      GT_BeginRefresh,
                  GT_EndRefresh,      GT_FilterIMsg,      GT_GetIMsg,
                  GT_PostFilterIMsg,  GT_RefreshWindow,   GT_ReplyIMsg,
                  GT_SetGadgetAttrs,  LayoutMenuItems,    LayoutMenus;


  RecordGrp =     GTMenu,             GTMenuItem,
                  NewGadget,          NewMenu;

  SetGrp =        NewGadgetFlags,     NewMenuFlags;

  SubRangeGrp =   NewMenuFlags;

  TagsGrp =       BevelBoxTags,       CreateMenuTags,     GadgetTags,
                  LayoutMenuTags;


  TypeGrp =       ArrayGrp,           EnumGrp,            PointerGrp,
                  RecordGrp,          SetGrp,             SubRangeGrp;

  All =           ConstGrp,           ProcGrp,            TagsGrp,
                  TypeGrp;

END GadTools.
```

8.18 GamePort

```
DEFINITION MODULE GamePort;

$$LongAllign:= FALSE      | Absolutely essential

FROM T_Exec       IMPORT IOCommand, nonstdVAL, IOStdReq;
FROM Resources    IMPORT ContextPtr;

CONST
  readEvent    = IOCommand( nonstdVAL + 0 );
  askCType     = IOCommand( nonstdVAL + 1 );
  setCType     = IOCommand( nonstdVAL + 2 );
  askTrigger   = IOCommand( nonstdVAL + 3 );
  setTrigger   = IOCommand( nonstdVAL + 4 );

|Errors
  errSetCType  = 1;

  portOne      = 0;
  portTwo      = 1;

TYPE
  Keys            = ( downKeys, upKeys, makemeword = 15 );
  KeySet          = SET OF Keys;

  GamePortTrigger = RECORD
                      keys    : KeySet;
                      timeout : CARDINAL;
                      xDelta  : CARDINAL;
                      yDelta  : CARDINAL
                    END;

   Controller     = (allocated=-1,noController,mouse,relJoystick,
                     absJoystick);

   GameReqPtr     = POINTER TO GameReq;
   GameReq        = RECORD OF IOStdReq END;

PROCEDURE OpenGamePort( port : INTEGER;
                        context : ContextPtr:=NIL ): GameReqPtr;

PROCEDURE CloseGamePort( VAR request : GameReqPtr );

GROUP
  All   = T_Exec.ExecIOGrp,askCType,askTrigger,errSetCType,portOne,
          portTwo,readEvent,setCType,setTrigger,Keys,GamePortTrigger,
          GameReq,GameReqPtr,Controller,OpenGamePort,CloseGamePort;

END GamePort.
```


8.19 Graphics

```
DEFINITION MODULE Graphics;
(* $A- *)


FROM System     IMPORT BITSET,LONGSET,SHORTSET,Regs,PROC, SysStringPtr;
FROM Hardware   IMPORT BltNodePtr,BeamOFlags,BeamOFlagSet;
FROM Exec       IMPORT Interrupt,Library,List,Message,MinList,Node,NodePtr,
                       NodeType,SignalSemaphore,SignalSemaphorePtr,TaskPtr,
                       MsgPortPtr,LibraryPtr;
FROM Utility    IMPORT HookPtr,StdTags;

TYPE WindowPtr = DEFERRED POINTER Intuition.WindowPtr;

TYPE
  ViewPortExtraPtr = POINTER TO ViewPortExtra;
  ViewPortPtr      = POINTER TO ViewPort;
  VTagListPtr      = POINTER TO VTagList;

|
|  BitMaps & Raster
|
  BitMapPtr       = POINTER TO BitMap;
  BitMap          = RECORD
                      bytesPerRow  : CARDINAL;
                      rows         : CARDINAL;
                      flags        : SHORTSET;
                      depth        : SHORTCARD;
                      pad          : CARDINAL;
                      planes       : ARRAY [0..7] OF ANYPTR;
                    END;

  BitScaleArgsPtr = POINTER TO BitScaleArgs;
  BitScaleArgs    = RECORD
                      srcX,
                      srcY         : CARDINAL;
                      srcWidth,
                      srcHeight    : CARDINAL;
                      srcXFactor,
                      srcYFactor   : CARDINAL;
                      destX,
                      destY        : CARDINAL;
                      destWidth,
                      destHeight   : CARDINAL;
                      xDestFactor,
                      yDestFactor  : CARDINAL;
                      srcBitMap    : BitMapPtr;
                      destBitMap   : BitMapPtr;
                      flags        : LONGSET;
                      xDDA,
                      yDDA         : CARDINAL;
                      reserved     : ARRAY [2] OF LONGINT;
                    END;
```

```
|
| Layers & Regions
|
  LayerInfoPtr       = POINTER TO LayerInfo;
  LayerPtr           = POINTER TO Layer;

  RectanglePtr       = POINTER TO Rectangle;
  Rectangle          = RECORD
                         minX : INTEGER;
                         minY : INTEGER;
                         maxX : INTEGER;
                         maxY : INTEGER;
                       END;

  ClipRectPtr        = POINTER TO ClipRect;
  ClipRect           = RECORD
                         next     : ClipRectPtr;
                         prev     : ClipRectPtr;
                         lobs     : LayerPtr;
                         bitMap   : BitMapPtr;
                         bounds   : Rectangle;
                         p1       : ClipRectPtr;
                         p2       : ClipRectPtr;
                         reserved : LONGINT;
                         flags    : LONGINT;
                       END;

  RegionRectanglePtr = POINTER TO RegionRectangle;
  RegionRectangle    = RECORD
                         next   : RegionRectanglePtr;
                         prev   : RegionRectanglePtr;
                         bounds : Rectangle;
                       END;

  RegionPtr          = POINTER TO Region;
  Region             = RECORD
                         bounds          : Rectangle;
                         regionRectangle : RegionRectanglePtr;
                       END;

CONST
  needsNoConcealedRasters = 1;

  isLessX                 = 1;
  isLessY                 = 2;
  isGrtrX                 = 4;
  isGrtrY                 = 8;

TYPE
  LayerFlags       = (layerSimple,layerSmart,layerSuper,lf3,layerUpdating,
                      lf5,layerBackdrop,layerRefresh,layerClipRectsLost);
  LayerFlagSet     = SET OF LayerFlags;

  RastPortPtr      = POINTER TO RastPort;
```



```
  Layer            = RECORD
                       front               : LayerPtr;
                       back                : LayerPtr;
                       clipRect            : ClipRectPtr;
                       rp                  : RastPortPtr;
                       bounds              : Rectangle;
                       reserved            : ARRAY [4] OF SHORTCARD;
                       priority            : CARDINAL;
                       flags               : LayerFlagSet;
                       superBitMap         : BitMapPtr;
                       superClipRect       : ClipRectPtr;
                       window              : WindowPtr;
                       scrollX             : INTEGER;
                       scrollY             : INTEGER;
                       cr                  : ClipRectPtr;
                       cr2                 : ClipRectPtr;
                       crnew               : ClipRectPtr;
                       superSaveClipRects  : ClipRectPtr;
                       cliprects           : ClipRectPtr;
                       layerInfo           : LayerInfoPtr;
                       lock                : SignalSemaphore;
                       backFill            : HookPtr;
                       reserved1           : LONGCARD;
                       clipRegion          : RegionPtr;
                       saveClipRects       : RegionPtr;
                       width,
                       height              : INTEGER;
                       reserved2           : ARRAY [18] OF SHORTCARD;
                       damageList          : RegionPtr;
                     END;

  LayerInfo        = RECORD
                       layer               : LayerPtr;
                       lp                  : LayerPtr;
                       obs                 : LayerPtr;
                       freeClipRects       : MinList;
                       lock                : SignalSemaphore;
                       head                : List;
                       longreserved        : LONGINT;
                       flags               : LayerFlagSet;
                       count               : SHORTINT;
                       lockLayersCount     : SHORTINT;
                       layerInfoExtraSize  : CARDINAL;
                       blitbuff            : ANYPTR;
                       layerInfoExtra      : ANYPTR;
                     END;

CONST
  lmnRegion          = -1;
  newLayerInfoCalled = 1;
  alertLayersNoMem   = $83010000;
```

```
|
| Colors
|

TYPE
  DisplayInfoPtr  = POINTER TO DisplayInfo;

  ColorMapType    = (cmapV33,cmapV36);

  ColorMapFlags   = (colormapTransparency,colorPlaneTransparency,
                     borderBlanking,borderNoTransparency,
                     videoControlBatch,userCopperClip);
  ColorMapFlagSet = SET OF ColorMapFlags;

  ColorMapPtr     = POINTER TO ColorMap;
  ColorMap        = RECORD
                      flags             : ColorMapFlagSet;
                      type              : ColorMapType;
                      count             : CARDINAL;
                      colorTable        : ANYPTR;
                      vpe               : ViewPortExtraPtr;
                      transparencyBits  : ANYPTR;
                      transparencyPlane : SHORTCARD;
                      reserved1         : SHORTCARD;
                      reserved2         : CARDINAL;
                      vp                : ViewPortPtr;
                      normalDisplayInfo : ANYPTR;
                      coerceDisplayInfo : ANYPTR;
                      batchItems        : VTagListPtr;
                      vpModeID          : LONGCARD;
                    END;


|
| Copper & Views
|
CONST
  move = 0;
  wait = 1;
  next = 2;

  sys  = 13;
  sht  = 14;
  lof  = 15;

TYPE
  CopListPtr     = POINTER TO CopList;

  CopInsPtr      = POINTER TO CopIns;
  CopIns         = RECORD
                     IF KEY opCode:CARDINAL
                       OF move THEN destAddr:INTEGER;
                                    destData:INTEGER;
                       OF wait THEN vWaitPos:INTEGER;
                                    hWaitPos:INTEGER;
```



```
                       OF next THEN nxtlist:CopListPtr;
                     END;
                   END;

  CprListPtr     = POINTER TO CprList;
  CprList        = RECORD
                     next     : CprListPtr;
                     start    : ANYPTR;
                     maxCount : INTEGER;
                   END;

  CopList        = RECORD
                     next      : CopListPtr;
                     copList   : CopListPtr;
                     viewPort  : ViewPortPtr;
                     copIns    : CopInsPtr;
                     copPtr    : CopInsPtr;
                     copLStart : ANYPTR;
                     copSStart : ANYPTR;
                     count     : INTEGER;
                     maxCount  : INTEGER;
                     dyOffset  : INTEGER;
                     cop2Start : ANYPTR;
                     cop3Start : ANYPTR;
                     cop4Start : ANYPTR;
                     cop5Start : ANYPTR;
                   END;

  UCopListPtr    = POINTER TO UCopList;
  UCopList       = RECORD
                     next         : UCopListPtr;
                     firstCopList : CopListPtr;
                     copList      : CopListPtr;
                   END;

  CopInitPtr     = POINTER TO CopInit;
  CopInit        = RECORD
                     vsyncHBlank : ARRAY [2]       OF CARDINAL;
                     diwstart    : ARRAY [4]       OF CARDINAL;
                     diagstrt    : ARRAY [4]       OF CARDINAL;
                     sprstrtup   : ARRAY [2*8*2]   OF CARDINAL;
                     wait14      : ARRAY [2]       OF CARDINAL;
                     normHBlank  : ARRAY [2]       OF CARDINAL;
                     genloc      : ARRAY [4]       OF CARDINAL;
                     jump        : ARRAY [(2*2)]   OF CARDINAL;
                     sprstop     : ARRAY [4]       OF CARDINAL;
                   END;
|
| Text & Gfx
|

TYPE
  FontStyles     = (underlined,bold,italic,extended,fs4,fs5,colorFont,
                    tagged);
  FontStyleSet   = SET OF FontStyles;
```

```
  FontFlags      = (romFont,diskFont,revPath,tallDot,wideDot,proportional,
                    designed,removed);
  FontFlagSet    = SET OF FontFlags;

CONST
  normalFont     = FontStyleSet:{};

TYPE
  FontTags       = TAGS OF StdTags
                     DeviceDPI  = $80000001 : RECORD x,y : CARDINAL END;
                   END;
  FontTagList    = ARRAY OF FontTags;
  FontTagListPtr = POINTER TO FontTagList;

  TextAttrPtr    = POINTER TO TextAttr;
  TextAttr       = RECORD
                     name    : SysStringPtr;
                     ySize   : CARDINAL;
                     style   : FontStyleSet;
                     flags   : FontFlagSet;
                   END;
  TTextAttrPtr   = POINTER TO TTextAttr;
  TTextAttr      = RECORD OF TextAttr
                     tags    : FontTagListPtr;
                   END;

CONST
  maxFontMatchWeight = 32767;

TYPE

  CharLocPtr     = POINTER TO CharLoc;
  CharLoc        = RECORD
                     pos,width  : INTEGER;
                   END;

  TextFontPtr    = POINTER TO TextFont;
  TextFont       = RECORD OF Message
                     ySize     : CARDINAL;
                     style     : FontStyleSet;
                     flags     : FontFlagSet;
                     xSize     : CARDINAL;
                     baseline  : CARDINAL;
                     boldSmear : CARDINAL;
                     accessors : CARDINAL;
                     loChar    : CHAR;
                     hiChar    : CHAR;
                     charData  : ANYPTR;
                     modulo    : CARDINAL;
                     charLoc   : POINTER TO ARRAY OF CharLoc;
                     charSpace : POINTER TO ARRAY OF INTEGER;
                     charKern  : POINTER TO ARRAY OF INTEGER;
                   END;
```



```
CONST
  noRemFont      = 0;

TYPE
  TextFontExtensionPtr= POINTER TO TextFontExtension;
  TextFontExtension = RECORD
                        matchWord     : CARDINAL;
                        flags0        : SHORTSET;
                        flags1        : SHORTSET;
                        backPtr       : TextFontPtr;
                        origReplyPort : MsgPortPtr;
                        tags          : FontTagListPtr;
                        oFontPatchS   : ANYPTR;
                        oFontPatchK   : ANYPTR;
                      END;

CONST
  ctColormask=  $000F;
  ctColorfont=  $0001;
  ctGreyfont=   $0002;
  ctAntialias=  $0004;

  ctbMapcolor=  0;
  ctfMapcolor=  $0001;

TYPE
  ColorFontColorsPtr= POINTER TO ColorFontColors;
  ColorFontColors   = RECORD
                        reserved   : CARDINAL;
                        count      : CARDINAL;
                        colorTable : POINTER TO ARRAY OF CARDINAL;
                      END;

  ColorTextFontPtr  = POINTER TO ColorTextFont;
  ColorTextFont     = RECORD
                        tf              : TextFont;
                        flags           : CARDINAL;
                        depth           : SHORTCARD;
                        fgColor         : SHORTCARD;
                        low             : SHORTCARD;
                        high            : SHORTCARD;
                        planePick       : SHORTCARD;
                        planeOnOff      : SHORTCARD;
                        colorFontColors : ColorFontColorsPtr;
                        charData        : ARRAY [8] OF ANYPTR;
                      END;

  TextExtentPtr     = POINTER TO TextExtention;
  TextExtention     = RECORD
                        width    : CARDINAL;
                        height   : CARDINAL;
                        extend   : Rectangle;
                      END;
```

```
|
| Gels
|
CONST
  ringtrigger =  1;
  anfracsize  =  6;
  animhalf    =$20;

  b2Norm      = 0;
  b2Swap      = 1;
  b2Bobber    = 2;

TYPE
  BobPtr              = POINTER TO Bob;
  SimpleSpritePtr     = POINTER TO SimpleSprite;
  DBufPacketPtr       = POINTER TO DBufPacket;
  SimpleSprite        = RECORD
                          posctldata : ANYPTR;
                          height     : CARDINAL;
                          x          : CARDINAL;
                          y          : CARDINAL;
                          num        : INTEGER;
                        END;

  VSpriteFlags        = (vsprite,saveBack,overlay,mustDraw,vf4,vf5,vf6,vf7,
                         backSaved,bobUpdate,gelGone,vsOverflow,vf12,vf13,
                         vf14,vf15);
  VSpriteFlagSet      = SET OF VSpriteFlags;

  VSpritePtr          = POINTER TO VSprite;
  VSprite             = RECORD
                          nextVSprite : VSpritePtr;
                          prevVSprite : VSpritePtr;
                          drawPath    : VSpritePtr;
                          clearPath   : VSpritePtr;
                          oldY        : INTEGER;
                          oldX        : INTEGER;
                          flags       : VSpriteFlagSet;
                          y           : INTEGER;
                          x           : INTEGER;
                          height      : INTEGER;
                          width       : INTEGER;
                          depth       : INTEGER;
                          meMask      : BITSET;
                          hitMask     : BITSET;
                          imageData   : ANYPTR;
                          borderLine  : ANYPTR;
                          collMask    : ANYPTR;
                          sprColors   : ANYPTR;
                          vsBob       : BobPtr;
                          planePick   : SHORTSET;
                          planeOnOff  : SHORTSET;
                        END;
```



```
  BobFlags            = (saveBob,bobIsComp,bf2,bf3,bf4,bf5,bf6,bf7,bWaiting,
                         bDrawn,bobsAway,bobNix,savePreserve,outStep,
                         bf14,bf15);
  BobFlagSet          = SET OF BobFlags;

  AnimCompPtr         = POINTER TO AnimComp;
  Bob                 = RECORD
                          flags       : BobFlagSet;
                          saveBuffer  : ANYPTR;
                          imageShadow : ANYPTR;
                          before      : BobPtr;
                          after       : BobPtr;
                          bobVSprite  : VSpritePtr;
                          bobComp     : AnimCompPtr;
                          dBuffer     : DBufPacketPtr;
                        END;

  AnimObPtr           = POINTER TO AnimOb;
  AnimComp            = RECORD
                          flags         : INTEGER;
                          timer         : INTEGER;
                          timeSet       : INTEGER;
                          nextComp      : AnimCompPtr;
                          prevComp      : AnimCompPtr;
                          nextSeq       : AnimCompPtr;
                          prevSeq       : AnimCompPtr;
                          animCRoutine  : PROCEDURE;
                          yTrans        : INTEGER;
                          xTrans        : INTEGER;
                          headOb        : AnimObPtr;
                          animBob       : BobPtr;
                        END;

  AnimOb              = RECORD
                          nextOb       : AnimObPtr;
                          prevOb       : AnimObPtr;
                          clock        : LONGINT;
                          anOldY       : INTEGER;
                          anOldX       : INTEGER;
                          anY          : INTEGER;
                          anX          : INTEGER;
                          yVel         : INTEGER;
                          xVel         : INTEGER;
                          yAccel       : INTEGER;
                          xAccel       : INTEGER;
                          ringYTrans   : INTEGER;
                          ringXTrans   : INTEGER;
                          animORoutine : PROCEDURE;
                          headComp     : AnimCompPtr;
                        END;
```

```
  DBufPacket        = RECORD
                        bufY      : INTEGER;
                        bufX      : INTEGER;
                        bufPath   : VSpritePtr;
                        bufBuffer : ANYPTR;
                      END;
CONST
   borderHit        = 0;
   topHit           = 1;
   bottomHit        = 2;
   leftHit          = 4;
   rightHit         = 8;

TYPE
  CollTablePtr      = POINTER TO CollTable;
  CollTable         = RECORD
                        collPtrs : ARRAY [16] OF ANYPTR
                      END;
  GelsInfoPtr       = POINTER TO GelsInfo;
  GelsInfo          = RECORD
                        sprRsrvd      : SHORTSET;
                        flags         : SHORTCARD;
                        gelHead       : VSpritePtr;
                        gelTail       : VSpritePtr;
                        nextLine      : ANYPTR;
                        lastColor     : ANYPTR;
                        collHandler   : CollTablePtr;
                        leftmost      : INTEGER;
                        rightmost     : INTEGER;
                        topmost       : INTEGER;
                        bottommost    : INTEGER;
                        firstBlissObj : ANYPTR;
                        lastBlissObj  : ANYPTR;
                      END;

|
| GfxNodes
|
CONST
  viewExtraType      = NodeType(1);
  viewportExtraType  = NodeType(2);
  specialMonitorType = NodeType(3);
  monitorSpecType    = NodeType(4);

TYPE
  ExtendedNodePtr    = POINTER TO ExtendedNode;
  ExtendedNode       = RECORD OF Node
                         subsystem : SHORTCARD;
                         subtype   : SHORTCARD;
                         library   : LibraryPtr;
                         init      : PROCEDURE():LONGINT;
                       END;
```



```
|
| Monitor
|

TYPE
  SpecialMonitorPtr= POINTER TO SpecialMonitor;

CONST
  toMonitor          = 0;
  fromMonitor        = 1;
  standardXoffset    = 9;
  standardYoffset    = 0;
  requestNtsc        = 1;
  requestPal         = 2;
  requestSpecial     = 4;
  requestA2024       = 8;
  standardMonitorMask= requestNtsc + requestPal;

  defaultMonitorName = "default.monitor";
  ntscMonitorName    = "ntsc.monitor";
  palMonitorName     = "pal.monitor";
  vgaMonitorName     = "vga.monitor";
  vga70MonitorName   = "vga70.monitor";

  standardNtscRows   = 262;
  standardPalRows    = 312;
  standardColorclocks= 226;
  standardDeniseMax  = 455;
  standardDeniseMin  = 93;
  standardNtscBeamcon= Beam0FlagSet:{};
  standardPalBeamcon = displayPal; |???

  specialBeamcon     = Beam0FlagSet:{varVBlank,loLDis,varVSync,varBeam,
                                     csBlank};

  minNtscRow         = 21;
  minPalRow          = 29;
  standardViewX      = $81;
  standardViewY      = $2C;
  standardHbstrt     = $06;
  standardHsstrt     = $0B;
  standardHsstop     = $1C;
  standardHbstop     = $2C;
  standardVbstrt     = $122;
  standardVsstrt     = $2A6;
  standardVsstop     = $3AA;
  standardVbstop     = $1066;

  vgaColorclocks     = standardColorclocks DIV 2;
  vgaTotalRows       = standardNtscRows*2;
  vgaDeniseMin       = 59;
  minVgaRow          = 29;
  vgaHbstrt          = $08;
  vgaHsstrt          = $0E;
  vgaHsstop          = $1C;
```

```
  vgaHbstop          = $1E;
  vgaVbstrt          = $000;
  vgaVsstrt          = $153;
  vgaVsstop          = $235;
  vgaVbstop          = $CCD;

  vga70Colorclocks   = standardColorclocks DIV 2;
  vga70TotalRows     = 449;
  vga70DeniseMin     = 59;
  minVga70Row        = 35;
  vga70Hbstrt        = $08;
  vga70Hsstrt        = $0E;
  vga70Hsstop        = $1C;
  vga70Hbstop        = $1E;
  vga70Vbstrt        = $000;
  vga70Vsstrt        = $2A6;
  vga70Vsstop        = $388;
  vga70Vbstop        = $F73;

  vga70Beamcon       = specialBeamcon/Beam0FlagSet:{vSyncTrue};

  broadcastHbstrt    = $01;
  broadcastHsstrt    = $06;
  broadcastHsstop    = $17;
  broadcastHbstop    = $27;
  broadcastVbstrt    = $000;
  broadcastVsstrt    = $2A6;
  broadcastVsstop    = $54C;
  broadcastVbstop    = $1C40;
  broadcastBeamcon   = Beam0FlagSet:{loLDis,csBlank};
  ratioFixedpart     = 4;
  ratioUnity         = LONGCARD(1) SHL ratioFixedpart;

TYPE
  LongProc         = PROCEDURE():LONGINT;

  MonitorSpecPtr   = POINTER TO MonitorSpec;
  MonitorSpec      = RECORD OF ExtendedNode
                       flags                   : BITSET;
                       ratioh                  : LONGINT;
                       ratiov                  : LONGINT;
                       totalRows               : CARDINAL;
                       totalColorclocks        : CARDINAL;
                       deniseMaxDisplayColumn  : CARDINAL;
                       beamCon0                : CARDINAL;
                       minRow                  : CARDINAL;
                       special                 : SpecialMonitorPtr;
                       openCount               : CARDINAL;
                       transform,
                       translate,
                       scale                   : LongProc;
                       xoffset                 : CARDINAL;
                       yoffset                 : CARDINAL;
                       legalView               : Rectangle;
                       maxoscan                : LongProc;
```



```
                     videoscan               : LongProc;
                     deniseMinDisplayColumn : CARDINAL;
                     displayCompatible       : LONGCARD;
                     displayInfoDataBase     : List;
                     displayInfoDataBaseSemaphore : SignalSemaphore;
                     reserved                : ARRAY [2] OF LONGINT;
                   END;

  AnalogSignalIntervalPtr = POINTER TO AnalogSignalInterval;
  AnalogSignalInterval=RECORD
                     strt,
                     stop   : CARDINAL;
                   END;

  SpecialMonitor   = RECORD OF ExtendedNode
                     flags     : BITSET;
                     doMonitor : LongProc;
                     reserved1,
                     reserved2,
                     reserved3 : LongProc;
                     hblank    : AnalogSignalInterval;
                     vblank    : AnalogSignalInterval;
                     hsync     : AnalogSignalInterval;
                     vsync     : AnalogSignalInterval;
                   END;

|
| DisplayInfo etc.
|

TYPE
  DisplayInfoHandle=ANYPTR;

CONST
  dtagDisp      = $80000000;
  dtagDims      = $80001000;
  dtagMntr      = $80002000;
  dtagName      = $80003000;

TYPE
  QueryHeaderPtr= POINTER TO QueryHeader;
  QueryHeader   = RECORD
                    structID   : LONGCARD;
                    displayID  : LONGCARD;
                    skipID     : LONGCARD;
                    length     : LONGCARD;
                  END;

TYPE
  PropertyFlags   = (isLace,isDualpf,isPf2pri,isHam,isEcs,isPal,isSprites,
                     isGenlock,isWb,isDraggable,isPanelled,isBeamsync,
                     isExtrahalfbrite,is13,is14,is15,is16);
  PropertyFlagSet = SET OF PropertyFlags;
```

```
CONST
  diAvailNochips        = $0001;
  diAvailNomonitor      = $0002;
  diAvailNotwithgenlock = $0004;

TYPE
  PointPtr        = POINTER TO Point;
  Point           = RECORD
                      x,y:INTEGER;
                    END;

  DisplayInfo     = RECORD OF QueryHeader
                      notAvailable      : CARDINAL;
                      propertyFlags     : PropertyFlagSet;
                      resolution        : Point;
                      pixelSpeed        : CARDINAL;
                      numStdSprites     : CARDINAL;
                      paletteRange      : CARDINAL;
                      spriteResolution  : Point;
                      pad               : ARRAY [4] OF SHORTCARD;
                      reserved2         : ARRAY [2] OF LONGINT;
                    END;

  DimensionInfoPtr= POINTER TO DimensionInfo;
  DimensionInfo   = RECORD OF QueryHeader
                      maxDepth          : CARDINAL;
                      minRasterWidth    : CARDINAL;
                      minRasterHeight   : CARDINAL;
                      maxRasterWidth    : CARDINAL;
                      maxRasterHeight   : CARDINAL;
                      nominal           : Rectangle;
                      maxOScan          : Rectangle;
                      videoOScan        : Rectangle;
                      txtOScan          : Rectangle;
                      stdOScan          : Rectangle;
                      pad               : ARRAY [14] OF SHORTCARD;
                      reserved          : ARRAY [2] OF LONGINT;
                    END;


CONST
  mcompatMixed  = 0;
  mcompatSelf   = 1;
  mcompatNobody =-1;

TYPE
  MonitorInfoPtr  = POINTER TO MonitorInfo;
  MonitorInfo     = RECORD OF QueryHeader
                      mspc              : MonitorSpecPtr;
                      viewPosition      : Point;
                      viewResolution    : Point;
                      viewPositionRange : Rectangle;
                      totalRows         : CARDINAL;
                      totalColorClocks  : CARDINAL;
                      minRow            : CARDINAL;
```



```
                      compatibility     : INTEGER;
                      pad               : ARRAY [36] OF SHORTCARD;
                      reserved          : ARRAY [2] OF LONGINT;
                    END;

CONST
  displayNameLen=32;
TYPE
  NameInfoPtr     = POINTER TO NameInfo;
  NameInfo        = RECORD
                      header   : QueryHeader;
                      name     : ARRAY [displayNameLen] OF CHAR;
                      reserved : ARRAY [2] OF LONGINT;
                    END;

CONST
  invalidID              = -1;

  monitorIDmask          = $FFFF1000;

  defaultMonitorID       = $00000000;
  ntscMonitorID          = $00011000;
  palMonitorID           = $00021000;
  vgaMonitorID           = $00031000;
  a2024MonitorID         = $00041000;
  protoMonitorID         = $00051000;

  loresKey               = $00000000;
  hiresKey               = $00008000;
  superKey               = $00008020;
  hamKey                 = $00000800;
  loreslaceKey           = $00000004;
  hireslaceKey           = $00008004;
  superlaceKey           = $00008024;
  hamlaceKey             = $00000804;
  loresdpfKey            = $00000400;
  hiresdpfKey            = $00008400;
  superdpfKey            = $00008420;
  loreslacedpfKey        = $00000404;
  hireslacedpfKey        = $00008404;
  superlacedpfKey        = $00008424;
  loresdpf2Key           = $00000440;
  hiresdpf2Key           = $00008440;
  superdpf2Key           = $00008460;
  loreslacedpf2Key       = $00000444;
  hireslacedpf2Key       = $00008444;
  superlacedpf2Key       = $00008464;
  extrahalfbriteKey      = $00000080;
  extrahalfbritelaceKey  = $00000084;

  vgaextraloresKey       = $00031004;
  vgaloresKey            = $00039004;
  vgaproductKey          = $00039024;
  vgahamKey              = $00031804;
  vgaextraloreslaceKey   = $00031005;
```

```
  vgaloreslaceKey          = $00039005;
  vgaproductlaceKey        = $00039025;
  vgahamlaceKey            = $00031805;
  vgaextraloresdpfKey      = $00031404;
  vgaloresdpfKey           = $00039404;
  vgaproductdpfKey         = $00039424;
  vgaextraloreslacedpfKey  = $00031405;
  vgaloreslacedpfKey       = $00039405;
  vgaproductlacedpfKey     = $00039425;
  vgaextraloresdpf2Key     = $00031444;
  vgaloresdpf2Key          = $00039444;
  vgaproductdpf2Key        = $00039464;
  vgaextraloreslacedpf2Key = $00031445;
  vgaloreslacedpf2Key      = $00039445;
  vgaproductlacedpf2Key    = $00039465;
  vgaextrahalfbriteKey     = $00031084;
  vgaextrahalfbritelaceKey = $00031085;

  a2024tenhertzKey         = $00041000;
  a2024fifteenhertzKey     = $00049000;

|
| Ints
|
CONST
  blitMsgFault=4;

TYPE
  IsrvstrPtr       = POINTER TO Isrvstr;
  Isrvstr          = RECORD OF Node
                       iptr   : IsrvstrPtr;
                       code   : ANYPTR;
                       ccode  : ANYPTR;
                       carg   : INTEGER;
                     END;
|
| RastPort
|
TYPE
  AreaInfoPtr      = POINTER TO AreaInfo;
  AreaInfo         = RECORD
                       vctrTbl   : ANYPTR;
                       vctrPtr   : ANYPTR;
                       flagTbl   : ANYPTR;
                       flagPtr   : ANYPTR;
                       count     : INTEGER;
                       maxCount  : INTEGER;
                       firstX    : INTEGER;
                       firstY    : INTEGER;
                     END;
  TmpRasPtr        = POINTER TO TmpRas;
  TmpRas           = RECORD
                       rasPtr    : ANYPTR;
                       size      : LONGINT;
                     END;
```



```
  DrawModes        = (dm0,complement,inversvid);
  DrawModeSet      = SET OF DrawModes;

CONST
  jam1             = DrawModeSet:{};
  jam2             = DrawModeSet:{dm0};

TYPE

  Pen              = SHORTCARD;              | The actual pen variables
  PenArrayPtr      = POINTER TO PenArray;
  PenArray         = ARRAY OF CARDINAL;      | like in Intuition.DrawInfo

  RastPortFlags    = (firstDot,oneDot,dBuffer,areaOutline,noCrossFill=5,
                      rpf8=8);

  RastPortFlagSet  = SET OF RastPortFlags;
  RastPort         = RECORD
                       layer        : LayerPtr;
                       bitMap       : BitMapPtr;
                       areaPtrn     : ANYPTR;
                       tmpRas       : TmpRasPtr;
                       areaInfo     : AreaInfoPtr;
                       gelsInfo     : GelsInfoPtr;
                       mask         : SHORTSET;      | used to be SHORTCARD
                       fgPen        : SHORTCARD;
                       bgPen        : SHORTCARD;
                       aOlPen       : SHORTCARD;     | areafill outline pen
                       drawMode     : DrawModeSet;
                       areaPtSz     : SHORTCARD;
                       linPatCnt    : SHORTCARD;
                       dummy        : SHORTCARD;
                       flags        : RastPortFlagSet;
                       linePtrn     : CARDINAL;
                       x            : INTEGER;      | current position
                       y            : INTEGER;
                       minterms     : ARRAY [8] OF SHORTCARD;
                       penWidth     : INTEGER;
                       penHeight    : INTEGER;
                       font         : TextFontPtr;
                       algoStyle    : FontStyleSet;
                       txFlags      : FontFlagSet;
                       txHeight     : CARDINAL;
                       txWidth      : CARDINAL;
                       txBaseline   : CARDINAL;
                       txSpacing    : INTEGER;
                       user         : ANYPTR;
                       longreserved : ARRAY [0..1] OF LONGINT;
                       wordreserved : ARRAY [0..6] OF CARDINAL;
                       reserved     : ARRAY [0..7] OF SHORTCARD;
                     END;
```

```
|
| Viewport
|
TYPE
  VTags            = (VTAG_END     = 0,
                      ChromaKey_Clr = $8000000,
                      ChromaKey_Set,
                      BitPlaneKey_Clr,
                      BitPlaneKey_Set,
                      BorderBlank_Clr,
                      BorderBlank_Set,
                      BorderNotrans_Clr,
                      BorderNotrans_Set,
                      ChromaPen_Clr,
                      ChromaPen_Set,
                      ChromaPlane_Set,
                      AttachCm_Set,
                      NextBufCm,
                      BatchCm_Clr,
                      BatchCm_Set,
                      NormalDisp_Get,
                      NormalDisp_Set,
                      CoerceDisp_Get,
                      CoerceDisp_Set,
                      ViewPortExtra_Get,
                      ViewPortExtra_Set,
                      ChromaKey_Get,
                      BitPlaneKey_Get,
                      BorderNotTrans_Get,
                      ChromaPen_Get,
                      ChromaPlane_Get,
                      AttachCm_Get,
                      BatchCm_Get,
                      BatchItems_Get,
                      BatchItems_Set,
                      BatchItems_Add,
                      VpModeId_Get,
                      VpModeId_Set,
                      VpModeId_Clr,
                      UserClip_Get,
                      UserClip_Set,
                      UserClip_Clr);
  VTagList         = ARRAY OF VTags;

  ViewModes        = (vm0,genlocVideo,lace,vm3,vm4,superHires,pfba,
                      extraHalfbrite,genlocAudio,vm9,dualpf,ham,
                      extendedMode,vpHide,sprites,hires);
  ViewModeSet      = SET OF ViewModes;
  RasInfoPtr       = POINTER TO RasInfo;
  RasInfo          = RECORD
                       next              : RasInfoPtr;
                       bitMap            : BitMapPtr;
                       rxOffset          : INTEGER;
                       ryOffset          : INTEGER;
                     END;
```



```
  ViewPort         = RECORD
                       next               : ViewPortPtr;
                       colorMap           : ColorMapPtr;
                       dspIns             : CopListPtr;
                       sprIns             : CopListPtr;
                       clrIns             : CopListPtr;
                       uCopIns            : UCopListPtr;
                       dWidth             : INTEGER;
                       dHeight            : INTEGER;
                       dxOffset           : INTEGER;
                       dyOffset           : INTEGER;
                       modes              : ViewModeSet;
                       spritePriorities   : SHORTCARD;
                       extendedModes      : SHORTSET;
                       rasInfo            : RasInfoPtr;
                     END;

  ViewPtr          = POINTER TO View;
  View             = RECORD
                       viewPort           : ViewPortPtr;
                       lofCprList         : CprListPtr;
                       shfCprList         : CprListPtr;
                       dyOffset           : INTEGER;
                       dxOffset           : INTEGER;
                       modes              : ViewModeSet;
                     END;

  ViewExtraPtr     = POINTER TO ViewExtra;
  ViewExtra        = RECORD OF ExtendedNode
                       view     : ViewPtr;
                       monitor  : MonitorSpecPtr;
                     END;

  ViewPortExtra    = RECORD OF ExtendedNode
                       viewPort     : ViewPortPtr;
                       displayClip  : Rectangle;
                     END;


|
| GfxBase
|
  DisplayFlags     = (ntsc,genloc,pal,todaSafe,df4,df5,df6,df7,
                      df8,df9,df10,df11,df12,df13,df14,df15);
  DisplayFlagSet   = SET OF DisplayFlags;
  ChipRevs         = (hrAgnus,hrDenise,cr2,cr3,cr4,cr5,cr6,cr7);
  ChipRevSet       = SET OF ChipRevs;

TYPE
  GfxBasePtr       = POINTER TO GfxBaseType;
  GfxBaseType      = RECORD OF Library
                       actiView              : ViewPtr;
                       copinit               : CopInitPtr;
                       cia                   : ANYPTR;
```

```
blitter                : ANYPTR;
loFlist                : ANYPTR;
shFlist                : ANYPTR;
blthd                  : BltNodePtr;
blttl                  : BltNodePtr;
bsblthd                : BltNodePtr;
bsblttl                : BltNodePtr;
vbsrv                  : Interrupt;
timsrv                 : Interrupt;
bltsrv                 : Interrupt;
textFonts              : List;
defaultFont            : TextFontPtr;
modes                  : BITSET;
vBlank                 : SHORTCARD;
debug                  : SHORTCARD;
beamSync               : INTEGER;
bplcon0                : BITSET;
spriteReserved         : SHORTCARD;
bytereserved           : SHORTCARD;
flags                  : BITSET;
blitLock               : INTEGER;
blitNest               : INTEGER;
blitWaitQ              : List;
blitOwner              : TaskPtr;
waitQ                  : List;
displayFlags           : DisplayFlagSet;
simpleSprites          : ANYPTR;
maxDisplayRow          : CARDINAL;
maxDisplayColumn       : CARDINAL;
normalDisplayRows      : CARDINAL;
normalDisplayColumns   : CARDINAL;
normalDPMX             : CARDINAL;
normalDPMY             : CARDINAL;
lastChanceMemory       : SignalSemaphorePtr;
lcMptr                 : ANYPTR;
microsPerLine          : CARDINAL;
minDisplayColumn       : CARDINAL;
chipRevBits0           : ChipRevSet;
reservedPad            : SHORTCARD;
reserved               : ARRAY [4] OF SHORTCARD;
monitorId              : CARDINAL;
hedley                 : ARRAY [8] OF ANYPTR;
hedleySprites          : ARRAY [8] OF ANYPTR;
hedleySprites1         : ARRAY [8] OF ANYPTR;
hedleyCount            : INTEGER;
hedleyFlags            : BITSET;
hedleyTmp              : INTEGER;
hashTable              : ANYPTR;
currentTotRows         : CARDINAL;
currentTotCclks        : CARDINAL;
hedleyHint             : SHORTCARD;
hedleyHint2            : SHORTCARD;
nreserved              : ARRAY [4] OF LONGINT;
a2024SyncRaster        : ANYPTR;
controlDeltaPal        : INTEGER;
```



```
          controlDeltaNtsc       : INTEGER;
          currentMonitor         : MonitorSpecPtr;
          monitorList            : List;
          defaultMonitor         : MonitorSpecPtr;
          monitorListSemaphore   : SignalSemaphorePtr;
          displayInfoDataBase    : ANYPTR;
          actiViewCprSemaphore   : SignalSemaphorePtr;
          utilityBase            : LibraryPtr;
          execBase               : LibraryPtr;
        END;

VAR GfxBase : GfxBasePtr;

LIBRARY GfxBase BY -156
  PROCEDURE AddAnimOb(     anOb  IN A0 : AnimObPtr;
                       VAR anKey IN A1 : AnimObPtr;
                           rp    IN A2 : RastPortPtr);

LIBRARY GfxBase BY -96
  PROCEDURE AddBob( Bob IN A0 : BobPtr;
                    rp  IN A1 : RastPortPtr);

LIBRARY GfxBase BY -480
  PROCEDURE AddFont( textFont IN A1 : TextFontPtr );

LIBRARY GfxBase BY -102
  PROCEDURE AddVSprite( vs IN A0 : VSpritePtr;
                        rp IN A1 : RastPortPtr );

LIBRARY GfxBase BY -492
  PROCEDURE AllocRaster( width  IN D0 : CARDINAL;
                         height IN D1 : CARDINAL ): ANYPTR;

LIBRARY GfxBase BY -504
  PROCEDURE AndRectRegion( region    IN A0 : RegionPtr;
                           rectangle IN A1 : RectanglePtr );

LIBRARY GfxBase BY -624
  PROCEDURE AndRegionRegion( region1 IN A0 : RegionPtr;
                             region2 IN A1 : RegionPtr ): BOOLEAN;

LIBRARY GfxBase BY -162
  PROCEDURE Animate( VAR anKey IN A0 : AnimObPtr;
                         rp    IN A1 : RastPortPtr );

LIBRARY GfxBase BY -258
  PROCEDURE AreaDraw( rp IN A1 : RastPortPtr;
                      x  IN D0 : INTEGER;
                      y  IN D1 : INTEGER ): BOOLEAN;
```

```
LIBRARY GfxBase BY -186
  PROCEDURE AreaEllipse( rp IN A1 : RastPortPtr;
                         cX IN D0 : INTEGER;
                         cY IN D1 : INTEGER;
                         a  IN D2 : INTEGER;
                         b  IN D3 : INTEGER): BOOLEAN;

LIBRARY GfxBase BY -264
  PROCEDURE AreaEnd( rp IN A1 : RastPortPtr ): BOOLEAN;

LIBRARY GfxBase BY -252
  PROCEDURE AreaMove( rp IN A1 : RastPortPtr;
                      x  IN D0 : INTEGER;
                      y  IN D1 : INTEGER ): BOOLEAN;

LIBRARY GfxBase BY -474
  PROCEDURE AskFont( rp       IN A1 : RastPortPtr;
                     textAttr IN A0 : TextAttrPtr );

LIBRARY GfxBase BY -84
  PROCEDURE AskSoftStyle( rp IN A1 : RastPortPtr ): FontStyleSet;

LIBRARY GfxBase BY -654
  PROCEDURE AttemptLockLayerRom( layer IN A5 : LayerPtr ): BOOLEAN;

LIBRARY GfxBase BY -30
  PROCEDURE BltBitMap( srcBitMap IN A0 : BitMapPtr;
                       srcX      IN D0 : INTEGER;
                       srcY      IN D1 : INTEGER;
                       dstBitMap IN A1 : BitMapPtr;
                       dstX      IN D2 : INTEGER;
                       dstY      IN D3 : INTEGER;
                       sizeX     IN D4 : INTEGER;
                       sizeY     IN D5 : INTEGER;
                       minterm   IN D6 : SHORTCARD;
                       mask      IN D7 : SHORTCARD;
                       tempA     IN A2 : ANYPTR ): LONGCARD;

LIBRARY GfxBase BY -606
  PROCEDURE BltBitMapRastPort( srcbm   IN A0 : BitMapPtr;
                               srcX    IN D0 : INTEGER;
                               srcY    IN D1 : INTEGER;
                               destRp  IN A1 : RastPortPtr;
                               destX   IN D2 : INTEGER;
                               destY   IN D3 : INTEGER;
                               sizeX   IN D4 : INTEGER;
                               sizeY   IN D5 : INTEGER;
                               minterm IN D6 : SHORTCARD );

LIBRARY GfxBase BY -300
  PROCEDURE BltClear( memBlock  IN A1 : ANYPTR;
                      bytecount IN D0 : LONGCARD;
                      flags     IN D1 : LONGSET );
```



```
LIBRARY GfxBase BY -636
  PROCEDURE BltMaskBitMapRastPort( srcbm   IN A0 : BitMapPtr;
                                   srcX    IN D0 : INTEGER;
                                   srcY    IN D1 : INTEGER;
                                   destRp  IN A1 : RastPortPtr;
                                   destX   IN D2 : INTEGER;
                                   destY   IN D3 : INTEGER;
                                   sizeX   IN D4 : INTEGER;
                                   sizeY   IN D5 : INTEGER;
                                   minterm IN D6 : SHORTCARD;
                                   bltmask IN A2 : ANYPTR );

LIBRARY GfxBase BY -312
  PROCEDURE BltPattern( rp      IN A1 : RastPortPtr;
                        mask    IN A0 : ANYPTR;
                        xl      IN D0 : INTEGER;
                        yl      IN D1 : INTEGER;
                        maxX    IN D2 : INTEGER;
                        maxY    IN D3 : INTEGER;
                        bytecnt IN D4 : INTEGER );

LIBRARY GfxBase BY -36
  PROCEDURE BltTemplate( srcTemplate IN A0 : ANYPTR;
                         srcX        IN D0 : INTEGER;
                         srcMod      IN D1 : INTEGER;
                         rp          IN A1 : RastPortPtr;
                         dstX        IN D2 : INTEGER;
                         dstY        IN D3 : INTEGER;
                         sizeX       IN D4 : INTEGER;
                         sizeY       IN D5 : INTEGER );

LIBRARY GfxBase BY -366
  PROCEDURE CBump( c IN A1 : UCopListPtr );

LIBRARY GfxBase BY -420
  PROCEDURE ChangeSprite( vp      IN A0 : ViewPortPtr;
                          s       IN A1 : SimpleSpritePtr;
                          newdata IN A2 : ANYPTR );

LIBRARY GfxBase BY -42
  PROCEDURE ClearEOL( rp IN A1 : RastPortPtr );

LIBRARY GfxBase BY -522
  PROCEDURE ClearRectRegion( region    IN A0 : RegionPtr;
                             rectangle IN A1 : RectanglePtr ): BOOLEAN;

LIBRARY GfxBase BY -528
  PROCEDURE ClearRegion( region IN A0 : RegionPtr );

LIBRARY GfxBase BY -48
  PROCEDURE ClearScreen(rp IN A1 : RastPortPtr);
```

```
LIBRARY GfxBase BY -552
  PROCEDURE ClipBlit( src     IN A0 : RastPortPtr;
                      srcX    IN D0 : INTEGER;
                      srcY    IN D1 : INTEGER;
                      dest    IN A1 : RastPortPtr;
                      destX   IN D2 : INTEGER;
                      destY   IN D3 : INTEGER;
                      xSize   IN D4 : INTEGER;
                      ySize   IN D5 : INTEGER;
                      minterm IN D6 : SHORTCARD );

LIBRARY GfxBase BY -78
  PROCEDURE CloseFont( font IN A1 : TextFontPtr );

LIBRARY GfxBase BY -372
  PROCEDURE CMove( c IN A1 : UCopListPtr;
                   a IN D0 : ANYPTR;
                   v IN D1 : INTEGER );

LIBRARY GfxBase BY -450
  PROCEDURE CopySBitMap( layer IN A0 : LayerPtr );

LIBRARY GfxBase BY -378
  PROCEDURE CWait( c IN A1 : UCopListPtr;
                   v IN D0 : INTEGER;
                   h IN D1 : INTEGER );

LIBRARY GfxBase BY -462
  PROCEDURE DisownBlitter();

LIBRARY GfxBase BY -534
  PROCEDURE DisposeRegion( region IN A0 : RegionPtr );

LIBRARY GfxBase BY -108
  PROCEDURE DoCollision( rp IN A1 : RastPortPtr );

LIBRARY GfxBase BY -246
  PROCEDURE Draw( rp IN A1 : RastPortPtr;
                  x  IN D0 : INTEGER;
                  y  IN D1 : INTEGER );

LIBRARY GfxBase BY -180
  PROCEDURE DrawEllipse( rp IN A1 : RastPortPtr;
                         cX IN D0 : INTEGER;
                         cY IN D1 : INTEGER;
                         a  IN D2 : INTEGER;
                         b  IN D3 : INTEGER );

LIBRARY GfxBase BY -114
  PROCEDURE DrawGList( rp IN A1 : RastPortPtr;
                       vp IN A0 : ViewPortPtr );
```



```
LIBRARY GfxBase BY -330
  PROCEDURE Flood( rp   IN A1 : RastPortPtr;
                   mode IN D2 : LONGCARD;
                   x    IN D0 : INTEGER;
                   y    IN D1 : INTEGER ): BOOLEAN;

LIBRARY GfxBase BY -576
  PROCEDURE FreeColorMap( colorMap IN A0 : ColorMapPtr );

LIBRARY GfxBase BY -546
  PROCEDURE FreeCopList( coplist IN A0 : CopListPtr );

LIBRARY GfxBase BY -564
  PROCEDURE FreeCprList( cprlist IN A0 : CprListPtr );

LIBRARY GfxBase BY -600
  PROCEDURE FreeGBuffers( anOb IN A0 : AnimObPtr;
                          rp   IN A1 : RastPortPtr;
                          db   IN D0 : BOOLEAN );

LIBRARY GfxBase BY -498
  PROCEDURE FreeRaster( p      IN A0 : ANYPTR;
                        width  IN D0 : CARDINAL;
                        height IN D1 : CARDINAL );

LIBRARY GfxBase BY -414
  PROCEDURE FreeSprite( pick IN D0 : INTEGER );

LIBRARY GfxBase BY -540
  PROCEDURE FreeVPortCopLists( vp IN A0 : ViewPortPtr );

LIBRARY GfxBase BY -570
  PROCEDURE GetColorMap( entries IN D0 : LONGINT ): ColorMapPtr;

LIBRARY GfxBase BY -168
  PROCEDURE GetGBuffers( anOb IN A0 : AnimObPtr;
                         rp   IN A1 : RastPortPtr;
                         db   IN D0 : BOOLEAN ): BOOLEAN;

LIBRARY GfxBase BY -582
  PROCEDURE GetRGB4( colorMap IN A0 : ColorMapPtr;
                     entry    IN D0 : LONGINT ): LONGINT;

LIBRARY GfxBase BY -408
  PROCEDURE GetSprite( sprite IN A0 : SimpleSpritePtr;
                       pick   IN D0 : INTEGER ): INTEGER;

LIBRARY GfxBase BY -282
  PROCEDURE InitArea( VAR areainfo   IN A0 : AreaInfo;
                          buffer     IN A1 : ANYPTR;
                          maxvectors IN D0 : INTEGER );
```

```
LIBRARY GfxBase BY -390
  PROCEDURE InitBitMap( VAR bm     IN A0 : BitMap;
                            depth  IN D0 : INTEGER;
                            width  IN D1 : INTEGER;
                            height IN D2 : INTEGER );

LIBRARY GfxBase BY -120
  PROCEDURE InitGels( head  IN A0 : VSpritePtr;
                      tail  IN A1 : VSpritePtr;
                      gInfo IN A2 : GelsInfoPtr );

LIBRARY GfxBase BY -174
  PROCEDURE InitGMasks( anOb IN A0 : AnimObPtr );

LIBRARY GfxBase BY -126
  PROCEDURE InitMasks( vs IN A0 : VSpritePtr );

LIBRARY GfxBase BY -198
  PROCEDURE InitRastPort( VAR rp IN A1 : RastPort );

LIBRARY GfxBase BY -468
  PROCEDURE InitTmpRas( VAR tmpras IN A0 : TmpRas;
                            buffer IN A1 : ANYPTR;
                            size   IN D0 : LONGINT );

LIBRARY GfxBase BY -360
  PROCEDURE InitView( VAR view IN A1 : View );

LIBRARY GfxBase BY -204
  PROCEDURE InitVPort( VAR vp IN A0 : ViewPort );

LIBRARY GfxBase BY -192
  PROCEDURE LoadRGB4( vp     IN A0 : ViewPortPtr;
                      colors IN A1 : ANYPTR;
                      count  IN D0 : INTEGER );

LIBRARY GfxBase BY -222
  PROCEDURE LoadView( view IN A1 : ViewPtr );

LIBRARY GfxBase BY -432
  PROCEDURE LockLayerRom( layer IN A5 : LayerPtr );

LIBRARY GfxBase BY -216
  PROCEDURE MakeVPort( view     IN A0 : ViewPtr;
                       viewport IN A1 : ViewPortPtr );

LIBRARY GfxBase BY -240
  PROCEDURE Move( rp IN A1 : RastPortPtr;
                  x  IN D0 : INTEGER;
                  y  IN D1 : INTEGER );
```



```
LIBRARY GfxBase BY -426
  PROCEDURE MoveSprite( vp     IN A0 : ViewPortPtr;
                        sprite IN A1 : SimpleSpritePtr;
                        x      IN D0 : INTEGER;
                        y      IN D1 : INTEGER );

LIBRARY GfxBase BY -210
  PROCEDURE MrgCop( view IN A1 : ViewPtr );

LIBRARY GfxBase BY -516
  PROCEDURE NewRegion(): RegionPtr;

LIBRARY GfxBase BY -72
  PROCEDURE OpenFont( textAttr IN A0 : TextAttrPtr ): TextFontPtr;

LIBRARY GfxBase BY -510
  PROCEDURE OrRectRegion( region    IN A0 : RegionPtr;
                          rectangle IN A1 : RectanglePtr): BOOLEAN;

LIBRARY GfxBase BY -612
  PROCEDURE OrRegionRegion( region1 IN A0 : RegionPtr;
                            region2 IN A1 : RegionPtr ): BOOLEAN;

LIBRARY GfxBase BY -456
  PROCEDURE OwnBlitter();

LIBRARY GfxBase BY -336
  PROCEDURE PolyDraw( rp    IN A1 : RastPortPtr;
                      count IN D0 : INTEGER;
                      array IN A0 : ANYPTR );

LIBRARY GfxBase BY -276
  PROCEDURE QBlit( bp IN A1 : BltNodePtr );

LIBRARY GfxBase BY -294
  PROCEDURE QBSBlit( bsp IN A1 : BltNodePtr );

LIBRARY GfxBase BY -318
  PROCEDURE ReadPixel( rp IN A1 : RastPortPtr;
                       x  IN D0 : INTEGER;
                       y  IN D1 : INTEGER ): LONGINT;

LIBRARY GfxBase BY -306
  PROCEDURE RectFill( rp   IN A1 : RastPortPtr;
                      xMin IN D0 : INTEGER;
                      yMin IN D1 : INTEGER;
                      xMax IN D2 : INTEGER;
                      yMax IN D3 : INTEGER );

LIBRARY GfxBase BY -486
  PROCEDURE RemFont( textFont IN A1 : TextFontPtr );
```

```
LIBRARY GfxBase BY -132
  PROCEDURE RemIBob( bob IN A0 : BobPtr;
                     rp  IN A1 : RastPortPtr;
                     vp  IN A2 : ViewPortPtr );

LIBRARY GfxBase BY -138
  PROCEDURE RemVSprite( vs IN A0 : VSpritePtr );

LIBRARY GfxBase BY -396
  PROCEDURE ScrollRaster( rp   IN A1 : RastPortPtr;
                          dx   IN D0 : INTEGER;
                          dy   IN D1 : INTEGER;
                          xMin IN D2 : INTEGER;
                          yMin IN D3 : INTEGER;
                          xMax IN D4 : INTEGER;
                          yMax IN D5 : INTEGER );

LIBRARY GfxBase BY -588
  PROCEDURE ScrollVPort( vp IN A0 : ViewPortPtr );

LIBRARY GfxBase BY -342
  PROCEDURE SetAPen( rp  IN A1 : RastPortPtr;
                     pen IN D0 : CARDINAL );

LIBRARY GfxBase BY -348
  PROCEDURE SetBPen( rp  IN A1 : RastPortPtr;
                     pen IN D0 : CARDINAL );

LIBRARY GfxBase BY -144
  PROCEDURE SetCollision( num     IN D0 : LONGCARD;
                          routine IN A0 : PROC;
                          gInfo   IN A1 : GelsInfoPtr );

LIBRARY GfxBase BY -354
  PROCEDURE SetDrMd( rp   IN A1 : RastPortPtr;
                     mode IN D0 : DrawModeSet );

LIBRARY GfxBase BY -66
  PROCEDURE SetFont( rp   IN A1 : RastPortPtr;
                     font IN A0 : TextFontPtr );

LIBRARY GfxBase BY -234
  PROCEDURE SetRast( rp  IN A1 : RastPortPtr;
                     pen IN D0 : CARDINAL );

LIBRARY GfxBase BY -288
  PROCEDURE SetRGB4( vp IN A0 : ViewPortPtr;
                     n  IN D0 : CARDINAL;
                     r  IN D1 : CARDINAL;
                     g  IN D2 : CARDINAL;
                     b  IN D3 : CARDINAL );
```



```
LIBRARY GfxBase BY -630
  PROCEDURE SetRGB4CM( cm IN A0 : ColorMapPtr;
                       n  IN D0 : CARDINAL;
                       r  IN D1 : CARDINAL;
                       g  IN D2 : CARDINAL;
                       b  IN D3 : CARDINAL );

LIBRARY GfxBase BY -90
  PROCEDURE SetSoftStyle( rp     IN A1 : RastPortPtr;
                          style  IN D0 : FontStyleSet;
                          enable IN D1 : FontStyleSet ): FontStyleSet;

LIBRARY GfxBase BY -150
  PROCEDURE SortGList( rp IN A1 : RastPortPtr );

LIBRARY GfxBase BY -444
  PROCEDURE SyncSBitMap( layer IN A0 : LayerPtr );

LIBRARY GfxBase BY -60
  PROCEDURE Text( rp     IN A1 : RastPortPtr;
                  string IN A0 : ANYPTR;
                  count  IN D0 : LONGINT );

LIBRARY GfxBase BY -60
  PROCEDURE TextStr(     rp     IN A1 : RastPortPtr;
                     REF string IN A0 : STRING;
                         count  IN D0 : LONGINT );

LIBRARY GfxBase BY -54
  PROCEDURE TextLength( rp     IN A1 : RastPortPtr;
                        string IN A0 : ANYPTR;
                        count  IN D0 : INTEGER ): INTEGER;

LIBRARY GfxBase BY -54
  PROCEDURE TextLengthStr(     rp     IN A1 : RastPortPtr;
                           REF string IN A0 : STRING;
                               count  IN D0 : INTEGER ): INTEGER;

LIBRARY GfxBase BY -594
  PROCEDURE UCopperListInit( copperList IN A0 : UCopListPtr;
                             num        IN D0 : LONGINT ): UCopListPtr;


LIBRARY GfxBase BY -438
  PROCEDURE UnlockLayerRom( layer IN A5 : LayerPtr );

LIBRARY GfxBase BY -384
  PROCEDURE VBeamPos(): LONGINT;

LIBRARY GfxBase BY -228
  PROCEDURE WaitBlit();

LIBRARY GfxBase BY -402
  PROCEDURE WaitBOVP( vp IN A0 : ViewPortPtr );
```

```
LIBRARY GfxBase BY -270
  PROCEDURE WaitTOF();

LIBRARY GfxBase BY -324
  PROCEDURE WritePixel( rp IN A1 : RastPortPtr;
                        x  IN D0 : INTEGER;
                        y  IN D1 : INTEGER ): BOOLEAN;

LIBRARY GfxBase BY -558
  PROCEDURE XorRectRegion( region    IN A0 : RegionPtr;
                           rectangle IN A1 : RectanglePtr ): BOOLEAN;

LIBRARY GfxBase BY -618
  PROCEDURE XorRegionRegion( region1 IN A0 : RegionPtr;
                             region2 IN A1 : RegionPtr ): BOOLEAN;


LIBRARY GfxBase BY -660
  PROCEDURE GfxNew( gfxNodeType IN D0 : LONGINT ): ANYPTR;

LIBRARY GfxBase BY -666
  PROCEDURE GfxFree( gfxNodePtr IN A0 : ANYPTR );

LIBRARY GfxBase BY -672
  PROCEDURE GfxAssociate( associateNode IN A0 : ANYPTR;
                          gfxNodePtr    IN A1 : ANYPTR);

LIBRARY GfxBase BY -678
  PROCEDURE BitMapScale( VAR bitScaleArgs IN A0 : BitScaleArgs );

LIBRARY GfxBase BY -684
  PROCEDURE ScalerDiv( factor      IN D0 : CARDINAL;
                       numerator   IN D1 : CARDINAL;
                       denominator IN D2 : CARDINAL ): CARDINAL;

LIBRARY GfxBase BY -690
  PROCEDURE TextExtent(     rp         IN A1 : RastPortPtr;
                            string     IN A0 : ANYPTR;
                            count      IN D0 : INTEGER;
                        VAR textExtent IN A2 : TextExtention );

LIBRARY GfxBase BY -690
  PROCEDURE TextExtentStr(     rp         IN A1 : RastPortPtr;
                           REF string     IN A0 : STRING;
                               count      IN D0 : INTEGER;
                           VAR textExtent IN A2 : TextExtention );
```



```
LIBRARY GfxBase BY -696
  PROCEDURE TextFit(     rp                      IN A1 : RastPortPtr;
                         string                  IN A0 : ANYPTR;
                         strLen                  IN D0 : CARDINAL;
                     VAR textExtent              IN A2 : TextExtention;
                         constrainingExtent      IN A3 : TextExtentPtr;
                         strDirection            IN D1 : INTEGER;
                         constrainingBitWidth    IN D2 : CARDINAL;
                         constrainingBitHeight   IN D3 : CARDINAL):LONGCARD;
LIBRARY GfxBase BY -696
  PROCEDURE TextFitStr(     rp                      IN A1 : RastPortPtr;
                        REF string                  IN A0 : STRING;
                            strLen                  IN D0 : CARDINAL;
                        VAR textExtent              IN A2 : TextExtention;
                            constrainingExtent      IN A3 : TextExtentPtr;
                            strDirection            IN D1 : INTEGER;
                            constrainingBitWidth    IN D2 : CARDINAL;
                            constrainingBitHeight   IN D3 : CARDINAL):LONGCARD;
LIBRARY GfxBase BY -702
  PROCEDURE GfxLookUp( associateNode IN A0 : ANYPTR ): ANYPTR;

LIBRARY GfxBase BY -708
  PROCEDURE VideoControl( VAR colorMap IN A0 : ColorMap;
                              tagarray IN A1 : VTagListPtr ): BOOLEAN;

LIBRARY GfxBase BY -714
  PROCEDURE OpenMonitor( VAR monitorName IN A1 : STRING;
                             displayID   IN D0 : LONGCARD): MonitorSpecPtr;
LIBRARY GfxBase BY -720
  PROCEDURE CloseMonitor( monitorSpec IN A0 : MonitorSpecPtr ): BOOLEAN;

LIBRARY GfxBase BY -726
  PROCEDURE FindDisplayInfo(displayID IN D0 : LONGCARD):DisplayInfoHandle;
LIBRARY GfxBase BY -732
  PROCEDURE NextDisplayInfo( displayID IN D0 : LONGCARD ): LONGCARD;

LIBRARY GfxBase BY -756
  PROCEDURE GetDisplayInfoData( handle    IN A0 : DisplayInfoHandle;
                                buf       IN A1 : ANYPTR;
                                size      IN D0 : LONGCARD;
                                tagID     IN D1 : LONGCARD;
                                displayID IN D2 : LONGCARD ): LONGCARD;

LIBRARY GfxBase BY -762
  PROCEDURE FontExtent(     font       IN A0 : TextFontPtr;
                        VAR fontExtent IN A1 : TextExtention );
```

```
LIBRARY GfxBase BY -768
  PROCEDURE ReadPixelLine8( rp     IN A0 : RastPortPtr;
                            xstart IN D0 : CARDINAL;
                            ystart IN D1 : CARDINAL;
                            width  IN D2 : CARDINAL;
                            array  IN A2 : ANYPTR;
                            tempRP IN A1 : RastPortPtr ): LONGINT;

LIBRARY GfxBase BY -774
  PROCEDURE WritePixelLine8( rp     IN A0 : RastPortPtr;
                             xstart IN D0 : CARDINAL;
                             ystart IN D1 : CARDINAL;
                             width  IN D2 : CARDINAL;
                             array  IN A2 : ANYPTR;
                             tempRP IN A1 : RastPortPtr ): LONGINT;

LIBRARY GfxBase BY -780
  PROCEDURE ReadPixelArray8( rp     IN A0 : RastPortPtr;
                             xstart IN D0 : CARDINAL;
                             ystart IN D1 : CARDINAL;
                             xstop  IN D2 : CARDINAL;
                             ystop  IN D3 : CARDINAL;
                             array  IN A2 : ANYPTR;
                             temprp IN A1 : RastPortPtr ): LONGINT;

LIBRARY GfxBase BY -786
  PROCEDURE WritePixelArray8( rp     IN A0 : RastPortPtr;
                              xstart IN D0 : CARDINAL;
                              ystart IN D1 : CARDINAL;
                              xstop  IN D2 : CARDINAL;
                              ystop  IN D3 : CARDINAL;
                              array  IN A2 : ANYPTR;
                              temprp IN A1 : RastPortPtr ): LONGINT;

LIBRARY GfxBase BY -792
  PROCEDURE GetVPModeID( vp IN A0 : ViewPortPtr ): LONGCARD;

LIBRARY GfxBase BY -798
  PROCEDURE ModeNotAvailable( modeID IN D0 : LONGCARD ): LONGCARD;

LIBRARY GfxBase BY -804
  PROCEDURE WeighTAMatch( reqTextAttr    IN A0 : TextAttrPtr;
                          targetTextAttr IN A1 : TextAttrPtr;
                          targetTags     IN A2 : FontTagListPtr):CARDINAL;
LIBRARY GfxBase BY -810
  PROCEDURE EraseRect( rp   IN A1 : RastPortPtr;
                       xMin IN D0 : INTEGER;
                       yMin IN D1 : INTEGER;
                       xMax IN D2 : INTEGER;
                       yMax IN D3 : INTEGER );

LIBRARY GfxBase BY -816
  PROCEDURE ExtendFont( font     IN A0 : TextFontPtr;
                        fontTags IN A1 : FontTagListPtr ): LONGCARD;
```



```
LIBRARY GfxBase BY -822
  PROCEDURE StripFont( font IN A0 : TextFontPtr );
GROUP
  ViewGrp       = RasInfo,             RasInfoPtr,          View,
                  ViewModes,           ViewModeSet,         ViewPort,
                  ViewPtr;

  RasterGrp     = BitMap,              RastPort,            RastPortFlags,
                  RastPortFlagSet,     RastPortPtr,
                  AllocRaster,         FreeRaster,          InitBitMap,
                  InitRastPort;

  DrMdGrp       = RasterGrp,
                  jam1,                jam2,
                  DrawModes,           DrawModeSet,
                  SetAPen,             SetBPen,             SetDrMd,
                  SetRast;
                  | SetDrPt              SetWrMsk,

  DrawGrp       = RastPortPtr,
                  ClearEOL,            ClearScreen,         Draw,
                  DrawEllipse,         Move,                PolyDraw,
                  ReadPixel,           ScrollRaster,        Text,
                  TextLength,          TextLengthStr,       TextStr,
                  WritePixel;
                  |DrawCircle,         |TestPixel,

  AreaGrp       = AreaInfo,            AreaInfoPtr,         RastPortPtr,
                  TmpRas,              TmpRasPtr,
                  AreaDraw,            AreaEllipse,
                  AreaEnd,             AreaMove,            (*BndryOff,*)
                  Flood,               InitArea,            InitTmpRas,
                  RectFill;
                  |SetAfPt,             SetOPen,            AreaCircle;

  ColorGrp      = ColorMap,            ColorMapPtr,         Pen,
                  PenArray,            PenArrayPtr,         ViewPortPtr,
                  FreeColorMap,        GetColorMap,         GetRGB4,
                  LoadRGB4,            SetRGB4,             SetRGB4CM;

  BltGrp        = BitMapPtr,           LONGSET,             RastPortPtr,
                  SHORTSET,
                  BltBitMap,           BltBitMapRastPort,   BltClear,
                  BltClear,            BltTemplate,         ClipBlit,
                  DisownBlitter,       OwnBlitter,          QBlit,
                  QBSBlit,             WaitBlit;

  CopGrp        = ViewGrp,
                  lof,                 move,                next,
                  sht,                 wait,
                  CopInit,             CopInitPtr,          CopIns,
                  CopInsPtr,           CopList,             CopListPtr,
                  CprList,             CprListPtr,          UCopList,
                  UCopListPtr,         ViewPortPtr,         ViewPtr,
```

```
                   CBump,              CMove,              CWait,
                   FreeCopList,        FreeVPortCopLists,  InitView,
                   InitVPort,          LoadView,           MakeVPort,
                   MrgCop,             ScrollRaster,       UCopList,
                   UCopperListInit,    VBeamPos,           WaitBOVP,
                   WaitTOF;
                   |CEnd

  LayerGrp       = ClipRect,           Isrvstr,            IsrvstrPtr,
                   Layer,              LayerFlags,         LayerFlagSet,
                   LayerInfo,          LayerPtr,           Rectangle,
                   RectanglePtr,       Region,             RegionPtr,
                   RegionRectangle,
                   AndRectRegion,      AndRegionRegion,
                   AttemptLockLayerRom,
                   ClearRectRegion,    ClearRegion,        CopySBitMap,
                   DisposeRegion,      LockLayerRom,       NewRegion,
                   OrRectRegion,       OrRegionRegion,     SyncSBitMap,
                   UnlockLayerRom,     XorRectRegion,      XorRegionRegion;

  FontGrp        = AddFont,            AskFont,            AskSoftStyle,
                   CloseFont,          OpenFont,           RemFont,
                   SetFont,            SetSoftStyle,       TextFont,
                   FontStyles,         FontStyleSet,       FontFlags,
                   FontFlagSet,        normalFont,         TextAttrPtr,
                   TextAttr,           TextFontPtr;

  GelGrp         = AddAnimOb,          AnimOb,             RastPortPtr,
                   AddBob,             Bob,                AddVSprite,
                   Animate,            ChangeSprite,       SimpleSprite,
                   ViewPortPtr,        DoCollision,        DrawGList,
                   FreeGBuffers,       FreeSprite,         GetGBuffers,
                   GetSprite,          |InitAnimate,       InitGels,
                   InitGMasks,         InitMasks,          VSprite,
                   VSpritePtr,         MoveSprite,         |RemBob,
                   RemIBob,            RemVSprite,         SetCollision,
                   PROC,               GelsInfoPtr,        SortGList,
                   DBufPacketPtr,      DBufPacket,         BobPtr,
                   AnimObPtr,          AnimCompPtr,        AnimComp,
                   BobFlags,           BobFlagSe t,         VSpriteFlags,
                   VSpriteFlagSet,     SimpleSpritePtr,    CollTablePtr,
                   CollTable,          GelsInfo;

  GfxBasicGrp    = DrMdGrp,            DrawGrp,            AreaGrp,
                   ColorGrp;


  All            = ViewGrp,            RasterGrp,          DrMdGrp,
                   DrawGrp,            AreaGrp,            ColorGrp,
                   BltGrp,             CopGrp,             LayerGrp,
                   FontGrp,            GelGrp,             DisplayFlags,
                   DisplayFlagSet,     GfxBasePtr,         GfxBaseType,
                   GfxBase;

END Graphics.
```


8.20 HardBlock

```
DEFINITION MODULE HardBlocks;
(* $A- *)
FROM System IMPORT LONGSET;
FROM Dos    IMPORT DosEnvec;

TYPE
  IdField              = ARRAY [0..3] OF CHAR;

CONST
  locationLimit        = $10;
  nil                  = $FFFFFFFF;
  badBlock             = IdField:("B","A","D","B");
  fileSysHeader        = IdField:("F","S","H","D");
  loadSeg              = IdField:("L","S","E","G");
  partition            = IdField:("P","A","R","T");
  rigidDisk            = IdField:("R","D","S","K");

TYPE
  HardBlock            = RECORD
                             id           : IdField;
                             summedLongs : LONGCARD;
                             chkSum       : LONGINT;
                             hostId       : LONGCARD;
                           END;

  BadBlockEntry        = RECORD
                             badBlock,
                             goodBlock : LONGCARD
                           END;

  BadBlockBlock        = RECORD OF HardBlock;
                             next         : LONGCARD;
                             reserved     : LONGCARD;
                             blockPairs   : ARRAY [61] OF BadBlockEntry
                           END;

  FileSysHeaderBlock   = RECORD OF HardBlock;
                             next           : LONGCARD;
                             flags          : LONGSET;
                             reserved1      : ARRAY [2] OF LONGCARD;
                             dosType,
                             version        : LONGCARD;
                             patchFlags     : LONGSET;
                             type,
                             task,
                             lock,
                             handler,
                             stackSize      : LONGCARD;
                             priority,
                             startup,
                             segListBlocks,
                             globalVec      : LONGINT;
```

```
                   reserved2    : ARRAY [23] OF LONGCARD;
                   reserved3    : ARRAY [21] OF LONGCARD
                 END;

  LoadSegBlock   = RECORD OF HardBlock;
                   next         : LONGCARD;
                   loadData     : ARRAY [123] OF LONGCARD;
                 END;

  PartitionFlags     = (bootable, noMount, makeMeLong = 31);
  PartitionFlagSet   = SET OF PartitionFlags;

  PartitionBlock = RECORD OF HardBlock;
                   next         : LONGCARD;
                   flags        : PartitionFlagSet;
                   reserved1    : ARRAY [2] OF LONGCARD;
                   DevFlags     : LONGSET;
                   driveName    : ARRAY [32] OF CHAR;
                   reserved2    : ARRAY [15] OF LONGCARD;
                   enviroment   : DosEnvec;
                   ereserved    : ARRAY [15] OF LONGCARD;
                   reserved256  : ARRAY [256] OF SHORTCARD;
                 END;

  RigidDiskFlags     = (last,       lastLun,    lastTId,    noReselect,
                        diskId,     ctrlrId,
                        makeMeLong = 31);
  RigidDiskFlagSet   = SET OF RigidDiskFlags;

  RigidDiskBlock = RECORD OF HardBlock;
                   blockBytes         : LONGCARD;
                   flags              : RigidDiskFlagSet;
                   badBlockList,
                   partitionList,
                   fileSysHeaderList,
                   driveInit          : LONGCARD;
                   reserved1          : ARRAY [6] OF LONGCARD;
                   cylinders,
                   sectors,
                   heads,
                   interleave,
                   park               : LONGCARD;
                   reserved2          : ARRAY [3] OF LONGCARD;
                   writePreComp,
                   reducedWrite,
                   stepRate           : LONGCARD;
                   reserved3          : ARRAY [5] OF LONGCARD;
                   blocksLo,
                   blocksHi,
                   loCylinder,
                   hiCylinder,
                   cylBlocks,
                   autoParkSeconds    : LONGCARD;
                   reserved4          : ARRAY [2]   OF LONGCARD;
                   diskVendor         : ARRAY [8]   OF CHAR;
```



```
                   diskProduct        : ARRAY [16]  OF CHAR;
                   diskRevision       : ARRAY [4]   OF CHAR;
                   controllerVendor   : ARRAY [8]   OF CHAR;
                   controllerProduct  : ARRAY [16]  OF CHAR;
                   controllerRevision : ARRAY [4]   OF CHAR;
                   reserved5          : ARRAY [10]  OF LONGCARD;
                   reserved512        : ARRAY [256] OF SHORTCARD
                 END;


GROUP
  All   =
  (*I*)   LONGSET,
  (*T*)   BadBlockBlock,      BadBlockEntry,      FileSysHeaderBlock,
          IdField,            LoadSegBlock,       PartitionBlock,
          PartitionBlock,     PartitionFlags,     PartitionFlagSet,
          RigidDiskBlock,     RigidDiskBlock,     RigidDiskFlags,
          RigidDiskFlagSet,
  (*C*)   badBlock,           fileSysHeader,      loadSeg,
          locationLimit,      nil,                partition,
          rigidDisk;

END HardBlocks.
```

8.21 Hardware

```
DEFINITION MODULE Hardware;
(* $A- *)
FROM System IMPORT BITSET,PROC,SHORTSET;

TYPE
  AdkFlags      = (use0v1,use1v2,use2v3,use3vn,use0p1,use1p2,use2p3,use3pn,
                   fast,msbSync,wordSync,uartBrk,mfmPrec,preComp0,
                   preComp1,adkSet);
  AdkFlagSet    = SET OF AdkFlags;

CONST
  pre000ns      = AdkFlagSet:{};
  pre140ns      = AdkFlagSet:{preComp0};
  pre280ns      = AdkFlagSet:{preComp1};
  pre560ns      = AdkFlagSet:{preComp0,preComp1};

TYPE
  DmaFlags      = (aud0,aud1,aud2,aud3,disk,sprite,blitter,copper,
                   raster,master,blithog,df11,df12,bltnzero,bltdone,dmaSet);
  DmaFlagSet    = SET OF DmaFlags;

CONST
  DmaAll        = DmaFlagSet:{aud0..raster};

TYPE
  IntFlags      = (tbe,      dskblk,  softint, ports,   coper,   vertb,
                   blit,     aud0i,   aud1i,   aud2i,   aud3i,   rbf,
                   disksync, exter,   inten,   intSet );
  IntFlagSet    = SET OF IntFlags;

CONST
  hSizeBits     = 6;
  vSizeBits     = 16-hSizeBits;
  hSizeMask     = %0000000000111111;
  vSizeMask     = %0000001111111111;
  maxBytesPerRow= 128;

TYPE
  BC0Flags      = (nanbnc,nanbc,nabnc,nabc,anbnc,anbc,abnc,abc,
                   dest,srcC,srcB,srcA,ash1,ash2,ash4,ash8);
  BC0FlagSet    = SET OF BC0Flags;
  BC0FlagSetLow = SET OF [nanbnc..abc];

CONST
  aORb          = BC0FlagSet:{nabnc,nabc,anbnc,anbc,abnc,abc};
  aORc          = BC0FlagSet:{nanbc,nabc,anbnc,anbc,abnc,abc};
  aXORc         = BC0FlagSet:{nabc,abnc,nanbc,anbnc};
  aTOd          = BC0FlagSet:{abc,anbc,abnc,anbnc};
```



```
TYPE
  BC1Flags      = (lineMode,desc,fillCarryIn,fillOr,fillXor,ovFlag,signFlag,
                   bf7,bf8,bf9,bf10,bf11,bsh1,bsh2,bsh4,bsh8);
  BC1FlagSet    = SET OF BC1Flags;

CONST
  OneDot        = desc;
  blitReverse   = desc;
  aul           = BC1FlagSet:{fillCarryIn};
  sul           = BC1FlagSet:{fillOr};
  sud           = BC1FlagSet:{fillXor};
  octant1       = sud;
  octant2       = BC1FlagSet:{};
  octant3       = sul;
  octant4       = aul+sud;
  octant5       = aul+sul+sud;
  octant6       = aul+sul;
  octant7       = aul;
  octant8       = sul+sud;

TYPE
  BltNodePtr    = POINTER TO BltNode;
  BltNode       = RECORD
                    next      : BltNodePtr;
                    function  : PROC;
                    stat      : SHORTINT;
                    blitSize  : INTEGER;
                    beamSync  : INTEGER;
                    cleanUp   : PROC
                  END;

CONST
  cleanup       = $40;

TYPE
  BP0Flags          = (useBP3,ersy,bplace,lpen,bp04,bp05,bp06,bp07,gaud,
                       color,dbplf,homod,bpu0,bpu1,bpu2,bphires);
  BP0FlagSet        = SET OF BP0Flags;
  BP1Flags          = (p1h0,p1h1,p1h2,p1h3,p2h0,p2h1,p2h2,p2h3,bp18);
  BP1FlagSet        = SET OF BP1Flags;
  BP2Flags          = (pf1p0,pf1p1,pf1p2,pf2p0,pf2p1,pf2p2,pf2pri,bp27,
                       bp28,bp29,zdCten,zdBPen,zdBPSel0,zdBPSel1,zdBPSel2);
  BP2FlagSet        = SET OF BP2Flags;
  BP3Flags          = (extBlnkEn,extBlkZD,zdClkEn,bp33,brdNTran,brdNBlnk);
  BP3FlagSet        = SET OF BP3Flags;

  Beam0Flags        = (hSyncTrue,vSyncTrue,CSyncTrue,csBlank,
                       varCSync,displayPal,displayDual,varBeam,
                       varHSync,varVSync,cscBlankEn,loLDis,
                       varVBlank);
  Beam0FlagSet      = SET OF Beam0Flags;

  CXDFlags          = (p1p2,p1s01,p1s23,p1s45,p1s67,p2s01,p2s23,p2s45,p2s67,
                       s01s23,s01s45,s01s67,s23s45,s23s67,s45s67);
  CXDFlagSet        = SET OF CXDFlags;
```

```
CXCFlags          = (mvbp1,mvbp2,mvbp3,mvbp4,mvbp5,mvbp6,enbp1,enbp2,
                     enbp3,enbp4,enbp5,enbp6,ensp1,ensp3,ensp5,ensp7);
CXCFlagSet        = SET OF CXCFlags;

PotFlags          = (start,pf1,pf2,pf3,pf4,pf5,pf6,pf7,datalx,outlx,
                     dataly,outly,datarx,outrx,datary,outry);
PotFlagSet        = SET OF PotFlags;

SpriteControlFlags = (sho,ev8,sv8,sct3,sct4,sct5,sct6,att);
SpriteControlFlagSet = SET OF SpriteControlFlags;
SpriteControlInfo = RECORD
                      ev     : SHORTCARD;
                      flags  : SpriteControlFlagSet;
                    END;
SpriteInfo        = RECORD
                      pos   : CARDINAL;
                      ctl   : SpriteControlInfo;
                      data  : LONGCARD;
                    END;
Sprites           = ARRAY [0..7] OF SpriteInfo;

SerialFlags       = (d8,stop,sf2,rxd,tsre,tbes,rbfs,ovrun);
SerialFlagSet     = SET OF SerialFlags;
SerialInfo        = RECORD
                      flags : SerialFlagSet;
                      data  : CHAR
                    END;

DiskFlags         = (df0,df1,df2,df3,wordEqual,diskWrite,dmaOn,dskByte);
DiskFlagSet       = SET OF DiskFlags;
DiskInfo          = RECORD
                      flags : DiskFlagSet;
                      data  : SHORTCARD
                    END;
Coord             = RECORD
                      v,h : SHORTINT
                    END;

CustomPtr         = POINTER TO
                      RECORD
                        bltddat    : BITSET;
                        dmaconr    : DmaFlagSet;
                        vposr      : LONGCARD;
                        dskdatr    : CARDINAL;
                        joy0dat,
                        joy1dat    : Coord;
                        clxdat     : CXDFlagSet;
                        adkconr    : AdkFlagSet;
                        pot0dat,
                        pot1dat    : Coord;
                        potgor     : PotFlagSet;
                        serdatr    : SerialInfo;
                        dskbytr    : DiskInfo;
                        intenar,
                        intreqr    : IntFlagSet;
```



```
                        dskpt      : ANYPTR;
                        dsklen     : CARDINAL;
                        dskdat     : CARDINAL;
                        refptr     : CARDINAL;
                        vpos       : LONGCARD;
                        copcon     : BOOLEAN;
                        serdat     : SerialInfo;
                        serper     : CARDINAL;
                        potgo      : PotFlagSet;
                        joytest    : Coord;
                        strequ     : CARDINAL;
                        strvbl     : CARDINAL;
                        strhor     : CARDINAL;
                        strlong    : CARDINAL;
                        bltcon0    : BC0FlagSet;
                        bltcon1    : BC1FlagSet;
                        bltafwm,
                        bltalwm    : BITSET;
                        bltcpt,
                        bltbpt,
                        bltapt,
                        bltdpt     : ANYPTR;
                        bltsize    : CARDINAL;
                        pad2d      : SHORTCARD;
                        bltcon0l   : BC0FlagSetLow;
                        bltsizv,
                        bltsizh    : CARDINAL;
                        bltcmod,
                        bltbmod,
                        bltamod,
                        bltdmod    : CARDINAL;
                        unused2    : ARRAY [0..3] OF CARDINAL;
                        bltcdat,
                        bltbdat,
                        bltadat    : BITSET;
                        unused3    : ARRAY [0..2] OF CARDINAL;
                        deniseid   : CARDINAL;
                        dsksync    : BITSET;
                        cop1lc,
                        cop2lc     : ANYPTR;
                        copjmp1,
                        copjmp2    : INTEGER;
                        copins     : CARDINAL;
                        diwstrt,
                        diwstop,
                        ddfstrt,
                        ddfstop    : Coord;
                        dmacon     : DmaFlagSet;
                        clxcon     : CXCFlagSet;
                        intena,
                        intreq     : IntFlagSet;
                        adkcon     : AdkFlagSet;
                        aud        : ARRAY [0..3] OF
                                       RECORD
                                         audlc  : ANYPTR;
```

```
                  audlen : CARDINAL;
                  audper : CARDINAL;
                  audvol : CARDINAL;
                  auddat : INTEGER;
                  unused : LONGINT
                END;
  bplpt      : ARRAY [0..7] OF ANYPTR;
  bplcon0    : BP0FlagSet;
  bplcon1    : BP1FlagSet;
  bplcon2    : BP2FlagSet;
  bplcon3    : BP3FlagSet;
  bpl1mod,
  bpl2mod    : CARDINAL;
  unused5    : LONGINT;
  bpldat     : ARRAY [0..7] OF CARDINAL;
  sprpt      : ARRAY [0..7] OF ANYPTR;
  spr        : Sprites;
  colors     : ARRAY [0..31] OF CARDINAL;
  htotal     : CARDINAL;
  hsstop     : CARDINAL;
  hbstrt     : CARDINAL;
  hbstop     : CARDINAL;
  vtotal     : CARDINAL;
  vsstop     : CARDINAL;
  vbstrt     : CARDINAL;
  vbstop     : CARDINAL;
  sprhstrt   : CARDINAL;
  sprhstop   : CARDINAL;
  bplhstrt   : CARDINAL;
  bplhstop   : CARDINAL;
  hhposw     : CARDINAL;
  hhposr     : CARDINAL;
  beamcon0   : Beam0FlagSet;
  hsstrt     : CARDINAL;
  vsstrt     : CARDINAL;
  hcenter    : CARDINAL;
  diwhigh    : CARDINAL;
END;

CONST
  Custom        = CustomPtr($DFF000);

TYPE
  CiaIcrFlags   = (ta,tb,alrm,so,flg,if5,if6,setClr);
  CiaIcrFlagSet = SET OF CiaIcrFlags;

CONST
  ir            = setClr;

TYPE
  CiaCraFlags   = (craStart,craPbon,craOutmode,craRunmode,craLoad,
                   craInmode,craSpmode,craTodin);
  CiaCraFlagSet = SET OF CiaCraFlags;
```



```
CiaCrbFlags   = (crbStart,crbPbon,crbOutmode,crbRunmode,crbLoad,
                 crbInmode0,crbInmode1,crbAlarm);
CiaCrbFlagSet = SET OF CiaCrbFlags;

CiaaPraFlags  = (overlay,led,dskChange,dskPrt,dskTrack0,dskRdy,
                 gamePort0,gamePort1);
CiaaPraFlagSet= SET OF CiaaPraFlags;
CiaaPrbFlags  = [0..7];
CiaaPrbFlagSet= SET OF CiaaPrbFlags;

CiabPraFlags  = (prtrBusy,prtrPOut,prtrSel,comDSR,comCTS,comCD,comRTS,
                 comDTR);
CiabPraFlagSet= SET OF CiabPraFlags;
CiabPrbFlags  = (dskStep,dskDirec,dskSide,dskSel0,dskSel1,dskSel2,
                 dskSel3,dskMotor);
CiabPrbFlagSet= SET OF CiabPrbFlags;

Pad           = ARRAY [2..255] OF SHORTINT;

CIAAPtr       = POINTER TO
                  RECORD
                    pra    : CiaaPraFlagSet; pad0  : Pad;
                    prb    : CiaaPrbFlagSet; pad1  : Pad;
                    ddra   : CiaaPraFlagSet; pad2  : Pad;
                    ddrb   : CiaaPrbFlagSet; pad3  : Pad;
                    talo   : SHORTCARD;      pad4  : Pad;
                    tahi   : SHORTCARD;      pad5  : Pad;
                    tblo   : SHORTCARD;      pad6  : Pad;
                    tbhi   : SHORTCARD;      pad7  : Pad;
                    todlo  : SHORTCARD;      pad8  : Pad;
                    todmid : SHORTCARD;      pad9  : Pad;
                    todhi  : SHORTCARD;      pad10 : Pad;
                    unused : SHORTCARD;      pad11 : Pad;
                    sdr    : SHORTSET;       pad12 : Pad;
                    icr    : CiaIcrFlagSet;  pad13 : Pad;
                    cra    : CiaCraFlagSet;  pad14 : Pad;
                    crb    : CiaCrbFlagSet;  pad15 : Pad;
                  END;
```

```
  CIABPtr       = POINTER TO
                    RECORD
                      pra    : CiabPraFlagSet; pad0  : Pad;
                      prb    : CiabPrbFlagSet; pad1  : Pad;
                      ddra   : CiabPraFlagSet; pad2  : Pad;
                      ddrb   : CiabPrbFlagSet; pad3  : Pad;
                      talo   : SHORTCARD;      pad4  : Pad;
                      tahi   : SHORTCARD;      pad5  : Pad;
                      tblo   : SHORTCARD;      pad6  : Pad;
                      tbhi   : SHORTCARD;      pad7  : Pad;
                      todlo  : SHORTCARD;      pad8  : Pad;
                      todmid : SHORTCARD;      pad9  : Pad;
                      todhi  : SHORTCARD;      pad10 : Pad;
                      unused : SHORTCARD;      pad11 : Pad;
                      sdr    : SHORTSET;       pad12 : Pad;
                      icr    : CiaIcrFlagSet;  pad13 : Pad;
                      cra    : CiaCraFlagSet;  pad14 : Pad;
                      crb    : CiaCrbFlagSet;  pad15 : Pad;
                    END;

CONST
  CIAA          = CIAAPtr($BFE001);
  CIAB          = CIABPtr($BFD000);

END Hardware.
```


8.22 Icon

```
DEFINITION MODULE Icon;
(* $A- *)
FROM Exec      IMPORT LibraryPtr;
FROM Workbench IMPORT FreeListPtr,DiskObjectPtr,ToolTypeArrayPtr,
                      WBObjectType;
FROM System    IMPORT SysStringPtr,Regs;

VAR
  IconBase : LibraryPtr;

|---------------- Funktions for Icons ------------------------------------

LIBRARY IconBase BY -42
  PROCEDURE GetIcon(VAR Name  IN A0  : STRING;
                        icon  IN A1  : DiskObjectPtr;
                        f     IN A2  : FreeListPtr):LONGINT;

LIBRARY IconBase BY -48
  PROCEDURE PutIcon(VAR Name  IN A0  : STRING;
                        Obj   IN A1  : DiskObjectPtr):BOOLEAN;

GROUP
  IconGrp  = GetIcon,DiskObjectPtr,FreeListPtr,PutIcon;

|---------------- Funktions for Disks-Objects ----------------------------

LIBRARY IconBase BY -78
  PROCEDURE GetDiskObject(REF Name IN A0 : STRING):DiskObjectPtr;

LIBRARY IconBase BY -84
  PROCEDURE PutDiskObject(VAR Name IN A0 : STRING;
                              Obj  IN A1 : DiskObjectPtr):BOOLEAN;

LIBRARY IconBase BY -90
  PROCEDURE FreeDiskObject(Obj IN A0 : DiskObjectPtr);

LIBRARY IconBase BY -120
  PROCEDURE GetDefDiskObject(type IN D0 : WBObjectType):DiskObjectPtr;

LIBRARY IconBase BY -126
  PROCEDURE PutDefDiskObject(obj IN A0 : DiskObjectPtr);

LIBRARY IconBase BY -132
  PROCEDURE GetDiskObjectNew(REF name IN A0 : STRING):DiskObjectPtr;

LIBRARY IconBase BY -138
  PROCEDURE DeleteDiskObject(REF name IN A0 : STRING):BOOLEAN;
```

```
GROUP
  DiskObjectGrp  = GetDiskObject,DiskObjectPtr,PutDiskObject,
                   FreeDiskObject,GetDefDiskObject,PutDefDiskObject,
                   GetDiskObjectNew,DeleteDiskObject;

|--------------- Funktions for FreeLists -------------------------------

LIBRARY IconBase BY -72
  PROCEDURE AddFreeList(Free IN A0  : FreeListPtr;
                        mem  IN A1  : ANYPTR;
                        len  IN A2 : LONGINT):BOOLEAN;

LIBRARY IconBase BY -54
  PROCEDURE FreeFreeList(Free IN A0 : FreeListPtr);


GROUP
  FreeListGrp  = AddFreeList,FreeFreeList,FreeListPtr;

|--------------- Help-Funktions ---------------------------------------

LIBRARY IconBase BY -108
  PROCEDURE BumpRevision(New IN A0 : ANYPTR;
                         Old IN A1 : ANYPTR);

LIBRARY IconBase BY -96
  PROCEDURE FindToolType(ToolTypes  IN A0 : ToolTypeArrayPtr;
                         TypeName   IN A1 : SysStringPtr):SysStringPtr;

LIBRARY IconBase BY -102
  PROCEDURE MatchToolValue(TypeString IN A0 : SysStringPtr;
                           Val        IN A1 : SysStringPtr):BOOLEAN;


GROUP
  ToolTypeGrp  = BumpRevision,FindToolType,MatchToolValue,SysStringPtr;

  All          = IconGrp,DiskObjectGrp,FreeListGrp,ToolTypeGrp,IconBase;

END Icon.
```


8.23 IFFParse

```
(* $A- *)
DEFINITION MODULE IFFParse;

FROM Exec      IMPORT MinNode,LibraryPtr,MsgPort;
FROM System    IMPORT Regs,SysStringPtr;
FROM Utility   IMPORT HookPtr;
FROM Clipboard IMPORT IOClipboardPtr;

TYPE
|
| IFF return codes.  Most functions return either zero for success or
| one of these codes.   The exceptions are the read/write functions which
| return positive values for number of bytes or records read or written
| or a negative error code.  Some of these codes are not errors per sae,
| but valid conditions such as EOF or EOC (End of Chunk).

  IFFErr = (normalReturn = -12, noHook, notIFF, syntax, mangled, seek,
            write,read, noMem, noScope, eoc, eof, ok);

|       eof                           Reached logical end of file
|       eoc                           About to leave context
|       noScope                       No valid scope for property
|       noMem                         Internal memory alloc failed
|       read                          Stream read error
|       write                         Stream write error
|       seek                          Stream seek error
|       mangled                       Data in file is corrupt
|       syntax                        IFF syntax error
|       notIFF                        Not an IFF file
|       noHook                        No call-back hook provided
|       normalReturn                  Client handler normal return


|
| Universal IFF identifiers.
CONST
  IDFORM             = $464F524D;  | "FORM"
  IDLIST             = $4C495354;  | "LIST"
  IDCAT              = $43415420;  | "CAT "
  IDPROP             = $50524F50;  | "PROP"
  IDNULL             = $20202020;  | "NULL"

|
| Ident codes for universally recognized local context items.

  IFFLCIPROP         = $70726F70;  | "prop"
  IFFLCICOLLECTION   = $636F6C6C;  | "coll"
  IFFLCIENTRYHANDLER = $656E6864;  | "enhd"
  IFFLCIEXITHANDLER  = $65786864;  | "exhd"
```

```
TYPE
|
| Control modes for ParseIFF() function.

  ParseIFFMode  = (scan, step, rawStep, dummy = 31);

|
| Control modes for StoreLocalItem().

  StoreLocalItemMode    = (root = 1, top, prop, dummy = 31);

|
| "Flag" for writing functions.  If you pass this value in as a size
| to PushChunk() when writing a file, the parser will figure out the
| size of the chunk for you.  (Chunk sizes >= 2**31 are forbidden by the
| IFF specification, so this works.)

CONST   IFFSizeUnknown          = -1;

TYPE
|
| Possible call-back command values.  (Using 0 as the value for IFFCMD_INIT
| was, in retrospect, probably a bad idea.)

  IFFCmd      = (init, cleanup, read, write, seek, entry, exit, purgeLCI);

|       init          |  Prepare the stream for a session
|       cleanup       |  Terminate stream session
|       read          |  Read bytes from stream
|       write         |  Write bytes to stream
|       seek          |  Seek on stream
|       entry         |  You just entered a new context
|       exit          |  You're about to leave a context
|       purgeLCI      |  Purge a LocalContextItem

|
| Struct associated with an active IFF stream.
| "iff_Stream" is a value used by the client's read/write/seek functions
| it will not be accessed by the library itself and can have any value
| (could even be a pointer or a BPTR).
|

| Bit masks for "iff_Flags" field.

  IFFFlags      = (read, write, fseek, rseek,
                   reserved1 = 16,reserved16 = 31);
  IFFFlagSet    = SET OF IFFFlags;

  IFFHandlePtr  = POINTER TO IFFHandle;
  IFFHandle     = RECORD
                    stream      : ANYPTR;
                    flags       : IFFFlagSet;
                    depth       : LONGINT;    |  Depth of context stack.
                  END;
```



```
|
| When the library calls your stream handler, you'll be passed a pointer
| to this structure as the "message packet".

  IFFStreamCmdPtr = POINTER TO IFFStreamCmd;
  IFFStreamCmd    = RECORD
                      command : IFFCmd;  |  Operation to be performed
                      buf     : ANYPTR;  |  Pointer to data buffer
                      nBytes  : LONGINT; |  Number of bytes to be affected
                    END;

|
| A node associated with a context on the iff_Stack.  Each node
| represents a chunk, the stack representing the current nesting
| of chunks in the open IFF file.  Each context node has associated
| local context items in the (private) LocalItems list.  The ID, type,
| size and scan values describe the chunk associated with this node.

  ContextNodePtr= POINTER TO ContextNode;
  ContextNode   = RECORD
                    node        : MinNode;
                    iD          : LONGINT;
                    type        : LONGINT;
                    size        : LONGINT;  |  Size of this chunk
                    scan        : LONGINT;  |  # of bytes read/written so
                                            |  far
                  END;

|
| Local context items live in the ContextNode's.  Each class is identified
| by its lci_Ident code and has a (private) purge vector for when the
| parent context node is popped.

  LocalContextItemPtr = POINTER TO LocalContextItem;
  LocalContextItem    = RECORD
                          node        : MinNode;
                          iD,
                          type,
                          ident       : LONGCARD;
                        END;

|
| StoredProperty: a local context item containing the data stored
| from a previously encountered property chunk.

  StoredPropertyPtr = POINTER TO StoredProperty;
  StoredProperty    = RECORD
                        size      : LONGINT;
                        data      : ANYPTR;
                      END;
```

```
|
| Collection Item: the actual node in the collection list at which
| client will look.  The next pointers cross context boundaries so
| that the complete list is accessable.

  CollectionItemPtr = POINTER TO CollectionItem;
  CollectionItem    = RECORD
                        next       : CollectionItemPtr;
                        size       : LONGINT;
                        data       : ANYPTR;
                      END;

|
| Structure returned by OpenClipboard().  You may do CMD_POSTs and such
| using this structure.  However, once you call OpenIFF(), you may not
| do any more of your own I/O to the clipboard until you call CloseIFF().

ClipboardHandlePtr = POINTER TO ClipboardHandle;
ClipboardHandle    = RECORD
                       req          : IOClipboardPtr;
                       cBPort       : MsgPort;
                       satisfyPort  : MsgPort;
                     END;

VAR
  IFFParseBase  : LibraryPtr;


LIBRARY IFFParseBase BY -30
  PROCEDURE AllocIFF() : IFFHandlePtr;

LIBRARY IFFParseBase BY -36
  PROCEDURE OpenIFF(iff    IN A0: IFFHandlePtr;
                    rwMode IN D0: IFFFlagSet): IFFErr;

LIBRARY IFFParseBase BY -42
  PROCEDURE ParseIFF(iff     IN A0: IFFHandlePtr;
                     control IN D0: ParseIFFMode): IFFErr;

LIBRARY IFFParseBase BY -48
  PROCEDURE CloseIFF(iff IN A0: IFFHandlePtr);

LIBRARY IFFParseBase BY -54
  PROCEDURE FreeIFF(iff IN A0: IFFHandlePtr);

LIBRARY IFFParseBase BY -60
  PROCEDURE ReadChunkBytes(iff  IN A0: IFFHandlePtr;
                           buf  IN A1: ANYPTR;
                           size IN D0: LONGINT): LONGINT;

LIBRARY IFFParseBase BY -66
  PROCEDURE WriteChunkBytes(iff  IN A0: IFFHandlePtr;
                            buf  IN A1: ANYPTR;
                            size IN D0: LONGINT): IFFErr;
```



```
LIBRARY IFFParseBase BY -72
  PROCEDURE ReadChunkRecords(iff            IN A0: IFFHandlePtr;
                             buf            IN A1: ANYPTR;
                             bytesPerRecord IN D0: LONGINT;
                             nRecords       IN D1: LONGINT): LONGINT;

LIBRARY IFFParseBase BY -78
  PROCEDURE WriteChunkRecords(iff            IN A0: IFFHandlePtr;
                              buf            IN A1: ANYPTR;
                              bytesPerRecord IN D0: LONGINT;
                              nRecords       IN D1: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -84
  PROCEDURE PushChunk(iff  IN A0: IFFHandlePtr;
                      type IN D0: LONGINT;
                      id   IN D1: LONGINT;
                      size IN D2: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -90
  PROCEDURE PopChunk(iff IN A0: IFFHandlePtr): IFFErr;

LIBRARY IFFParseBase BY -102
  PROCEDURE EntryHandler(iff      IN A0: IFFHandlePtr;
                         type     IN D0: LONGINT;
                         id       IN D1: LONGINT;
                         position IN D2: LONGINT;
                         handler  IN A1: HookPtr;
                         object   IN A2: ANYPTR): IFFErr;

LIBRARY IFFParseBase BY -108
  PROCEDURE ExitHandler(iff      IN A0: IFFHandlePtr;
                         type     IN D0: LONGINT;
                         id       IN D1: LONGINT;
                         position IN D2: LONGINT;
                         handler  IN A1: HookPtr;
                         object   IN A2: ANYPTR): IFFErr;

LIBRARY IFFParseBase BY -114
  PROCEDURE PropChunk(iff  IN A0: IFFHandlePtr;
                      type IN D0: LONGINT;
                      id   IN D1: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -120
  PROCEDURE PropChunks(    iff       IN A0: IFFHandlePtr;
                       VAR propArray IN A1: ARRAY OF LONGINT;
                           nProps    IN D0: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -126
  PROCEDURE StopChunk(iff  IN A0: IFFHandlePtr;
                      type IN D0: LONGINT;
                      id   IN D1: LONGINT): IFFErr;
```

```
LIBRARY IFFParseBase BY -132
  PROCEDURE StopChunks(    iff       IN A0: IFFHandlePtr;
                       VAR propArray IN A1: ARRAY OF LONGINT;
                           nProps    IN D0: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -138
  PROCEDURE CollectionChunk(iff  IN A0: IFFHandlePtr;
                            type IN D0: LONGINT;
                            id   IN D1: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -144
  PROCEDURE CollectionChunks(    iff       IN A0: IFFHandlePtr;
                             VAR propArray IN A1: ARRAY OF LONGINT;
                                 nProps    IN D0: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -150
  PROCEDURE StopOnExit(iff  IN A0: IFFHandlePtr;
                       type IN D0: LONGINT;
                       id   IN D1: LONGINT): IFFErr;

LIBRARY IFFParseBase BY -156
  PROCEDURE FindProp(iff  IN A0: IFFHandlePtr;
                     type IN D0: LONGINT;
                     id   IN D1: LONGINT): StoredPropertyPtr;

LIBRARY IFFParseBase BY -162
  PROCEDURE FindCollection(iff  IN A0: IFFHandlePtr;
                           type IN D0: LONGINT;
                           id   IN D1: LONGINT): CollectionItemPtr;

LIBRARY IFFParseBase BY -168
  PROCEDURE FindPropContext(iff IN A0: IFFHandlePtr): ContextNodePtr;

LIBRARY IFFParseBase BY -174
  PROCEDURE CurrentChunk(iff IN A0: IFFHandlePtr): ContextNodePtr;

LIBRARY IFFParseBase BY -180
  PROCEDURE ParentChunk(contextNode IN A0: ContextNodePtr): ContextNodePtr;

LIBRARY IFFParseBase BY -186
  PROCEDURE AllocLocalItem(type     IN D0: LONGINT;
                           id       IN D1: LONGINT;
                           ident    IN D2: LONGINT;
                           dataSize IN D3: LONGINT): LocalContextItemPtr;

LIBRARY IFFParseBase BY -192
  PROCEDURE LocalItemData(localItem IN A0: LocalContextItemPtr): ANYPTR;

LIBRARY IFFParseBase BY -198
  PROCEDURE SetLocalItemPurge(localItem IN A0: LocalContextItemPtr;
                              purgeHook IN A1: HookPtr);

LIBRARY IFFParseBase BY -204
  PROCEDURE FreeLocalItem(localItem IN A0: LocalContextItemPtr);
```



```
LIBRARY IFFParseBase BY -210
  PROCEDURE FindLocalItem(iff   IN A0: IFFHandlePtr;
                          type  IN D0: LONGINT;
                          id    IN D1: LONGINT;
                          ident IN D2: LONGINT): LocalContextItemPtr;

LIBRARY IFFParseBase BY -216
  PROCEDURE StoreLocalItem(iff       IN A0: IFFHandlePtr;
                           localItem IN A1: LocalContextItemPtr;
                           position  IN D0: StoreLocalItemMode): IFFErr;

LIBRARY IFFParseBase BY -222
  PROCEDURE StoreItemInContext(iff         IN A0: IFFHandlePtr;
                               localItem   IN A1: LocalContextItemPtr;
                               contextNode IN A2: ContextNodePtr);

LIBRARY IFFParseBase BY -228
  PROCEDURE InitIFF(iff        IN A0: IFFHandlePtr;
                    flags      IN D0: LONGINT;
                    streamHook IN A1: HookPtr);

LIBRARY IFFParseBase BY -234
  PROCEDURE InitIFFasDOS(iff IN A0: IFFHandlePtr);

LIBRARY IFFParseBase BY -240
  PROCEDURE InitIFFasClip(iff IN A0: IFFHandlePtr);

LIBRARY IFFParseBase BY -246
  PROCEDURE OpenClipboard(unitNum IN D0: LONGINT): ClipboardHandlePtr;

LIBRARY IFFParseBase BY -252
  PROCEDURE CloseClipboard(clipboard IN A0: ClipboardHandlePtr);

LIBRARY IFFParseBase BY -258
  PROCEDURE GoodID(id IN D0: LONGINT): LONGINT;

LIBRARY IFFParseBase BY -264
  PROCEDURE GoodType(type IN D0: LONGINT): LONGINT;

LIBRARY IFFParseBase BY -270
  PROCEDURE IDtoStr(    id  IN D0: LONGINT;
                    VAR buf IN A0: STRING): SysStringPtr;

END IFFParse.
```

8.24 Input

```
DEFINITION MODULE Input;

(* $A- *)

FROM Resources IMPORT ContextPtr;
FROM Timer     IMPORT TimeVal;
FROM T_Exec    IMPORT nonstdVAL,IOCommand,IOStdReq;

TYPE
  IOInputPtr    = POINTER TO IOInput;
  IOInput       = RECORD OF IOStdReq END;

  ScreenPtr     = DEFERRED POINTER Intuition.ScreenPtr;

  Class         = (null,              rawkey,             rawmouse,
                   event,             pointerpos,         c15,
                   timer,             gadgetdown,         gadgetup,
                   requester,         menulist,           closewindow,
                   sizewindow,        refreshwindow,      newprefs,
                   diskremoved,       diskinserted,       activewindow,
                   inactivewindow,    newpointerpos,      menuhelp,
                   changewindow );


CONST
  classMax      = CARDINAL( Class'MAX );


TYPE
  SubClass      = (compatible,      pixel,              tablet );

  Coord         = RECORD
                    x, y : INTEGER;
                  END;

  | pointed to by InputEvent.eventAddress.
  | the key is in InputEvent.subClass.
  IEObjectPtr   = POINTER TO IEObject;
  IEObject      = RECORD
                    IF KEY : SubClass
                    OF pixel THEN
                      screen    : ScreenPtr;
                      position  : Coord;
                    OF tablet THEN
                      range,
                      value     : Coord;
                      pressure  : INTEGER;
                    END
                  END;
```



```
CONST
  | rawkey
  commCodeFirst = $78;
  commCodeLast  = $7F;
  keyCodeFirst  = $00;
  keyCodeLast   = $77;
  upPrefix      = $80;

  | ansi
  asciiDel      = $7F;
  asciiFirst    = $A0;
  asciiLast     = $7E;
  c0First       = $00;
  c0Last        = $1F;
  c1First       = $80;
  c1Last        = $9F;
  latin1First   = $A0;
  latin1Last    = $FF;

  | rawmouse
  lButton       = $68;
  mButton       = $6A;
  noButton      = $FF;
  rButton       = $69;

  | event
  newActive     = $01;
  newSize       = $02;
  refresh       = $03;

  | requester
  reqClear      = $00;
  reqSet        = $01;

TYPE
  Qualifiers    = (lShift,            rShift,             capsLock,
                   control,           lAlt,               rAlt,
                   lCommand,          rCommand,           numericPad,
                   repeat,            interrupt,          multiBroadcast,
                   midButton,         rightButton,        leftButton,
                   relativeMouse );
  QualifierSet  = SET OF Qualifiers;
  KeyQualSet    = SET OF [lShift..rCommand];
```

```
  InputEventPtr = POINTER TO InputEvent;
  InputEvent    = RECORD
                    nextEvent : InputEventPtr;
                    class     : Class;
                    subClass  : SubClass;
                    code      : CARDINAL;
                    qualifier : QualifierSet;
                    IF KEY : Class
                      OF newpointerpos THEN event         : IEObjectPtr;
                      OF rawkey        THEN prev1DownCode : SHORTCARD;
                                            prev1DownQual : KeyQualSet;
                                            prev2DownCode : SHORTCARD;
                                            prev2DownQual : KeyQualSet;
                      OF pointerpos    THEN x,y           : INTEGER
                    END;
                    timeStamp : TimeVal;
                  END;


CONST
  addHandler    = IOCommand( nonstdVAL + 0 );
  remHandler    = IOCommand( nonstdVAL + 1 );
  writeEvent    = IOCommand( nonstdVAL + 2 );
  setTresh      = IOCommand( nonstdVAL + 3 );
  setPeriod     = IOCommand( nonstdVAL + 4 );
  setMPort      = IOCommand( nonstdVAL + 5 );
  setMType      = IOCommand( nonstdVAL + 6 );
  setMTrig      = IOCommand( nonstdVAL + 7 );

PROCEDURE OpenInput( context : ContextPtr:=NIL ): IOInputPtr;

PROCEDURE CloseInput( VAR request : IOInputPtr );


GROUP
  EventGrp      = Class,            classMax,           SubClass,
                  Qualifiers,       QualifierSet,       IEObject,
                  IEObjectPtr,      InputEventPtr,      InputEvent;

  CommandGrp    = addHandler,        remHandler,        writeEvent,
                  setTresh,          setPeriod,         setMPort,
                  setMType,          setMTrig;

  FuncGrp       = IOInputPtr,        OpenInput,         CloseInput;

  All           = CommandGrp,        FuncGrp,           T_Exec.ExecIOGrp,
                  EventGrp;

END Input.
```


8.25 Intuition

```
DEFINITION MODULE Intuition;
(* $A- *)

FROM Exec       IMPORT Interrupt,           IOStdReq,            Library,
                       List,                ListPtr,             MemReqSet,
                       Message,             MinNode,             MsgPortPtr,
                       Node,                SignalSemaphore,     TaskPtr,
                       TaskSignals;

FROM Graphics   IMPORT BitMap,              BitMapPtr,           ClipRect,
                       DisplayInfoPtr,      PenArrayPtr,
                       DrawModeSet,         GfxBasePtr,          jam2,
                       LayerInfo,           LayerPtr,            RastPort,
                       RastPortPtr,         RectanglePtr,        RegionPtr,
                       RegionRectanglePtr,  SimpleSpritePtr,     TextAttr,
                       TextAttrPtr,         TextFontPtr,         TmpRas,
                       View,                ViewModeSet,         ViewModes,
                       ViewPort,            ViewPortPtr,         ViewPtr;

FROM Timer      IMPORT IOTimer,             TimeVal;

FROM System     IMPORT BITSET,              LONGSET,             PROC,
                       Regs,                SysStringPtr;

FROM Input      IMPORT InputEvent,          InputEventPtr,
                       lButton,             Qualifiers,          QualifierSet,
                       rButton,             upPrefix;

FROM Resources IMPORT ResHandles;

FROM Utility    IMPORT Hook,                HookPtr,             StdTags,
                       tagUser,             TagArrayPtr;

CONST
  wTB                 = tagUser + 99;  | $80000063    WindowTags base
  sTB                 = tagUser + 32;  | $80000020    ScreenTags base

TYPE
  BorderPtr           = POINTER TO Border;
  GadgetPtr           = POINTER TO Gadget;
  IBoxPtr             = POINTER TO IBox;
  ImagePtr            = POINTER TO Image;
  IntuiMessagePtr     = POINTER TO IntuiMessage;
  IntuiTextPtr        = POINTER TO IntuiText;
  MenuItemPtr         = POINTER TO MenuItem;
  PreferencesPtr      = POINTER TO Preferences;
  PropInfoPtr         = POINTER TO PropInfo;
  RememberPtr         = POINTER TO Remember;
  RequesterPtr        = POINTER TO Requester;
  ScreenPtr           = POINTER TO Screen;
  StringInfoPtr       = POINTER TO StringInfo;
  WindowPtr           = POINTER TO Window;
```

```
DEFINITION MODULE WinRes = Resources.ResHandles(WindowPtr);

DEFINITION MODULE ScreenRes = Resources.ResHandles(ScreenPtr);

TYPE
  MenuFlags    = ( menuEnabled,             | r/w this menu is enabled
                   miDrawn     = 8);        | r/o items are being drawn
  MenuFlagSet  = SET OF MenuFlags;

  MenuPtr      = POINTER TO Menu;
  Menu         = RECORD
                   nextMenu   : MenuPtr;
                   leftEdge   : INTEGER;
                   topEdge    : INTEGER;
                   width      : INTEGER;
                   height     : INTEGER;
                   flags      : MenuFlagSet;
                   menuName   : SysStringPtr;
                   firstItem  : MenuItemPtr;
                   jazzx      : INTEGER;
                   jazzy      : INTEGER;
                   beatx      : INTEGER;
                   beaty      : INTEGER;
                 END;

  MenuItemFlags   = (
                  checkIt,        | checkmarkable item
                  itemText,       | textual item, FALSE if graphical item
                  commSeq,        | command sequence
                  menuToggle,     | toggling checks (else mut. exclude)
                  itemEnabled,    | this item is enabled
                  mif5,           |
                  highComp,       | highlight by complementing the selectbox
                  highBox,        | highlight by "boxing" the selectbox
                  checked,        | state of the checkmark
                  mif9,           |
                  mif10,          |
                  mif11,          |
                  isDrawn,        | this item's subs are currently drawn
                  highItem,       | this item is currently highlighted
                  menuToggled );  | this item was already toggled

  MenuItemFlagSet = SET OF MenuItemFlags;

CONST
  highNone        = MenuItemFlagSet:{highBox,highComp};
  checkWidth      = 19;
  commWidth       = 27;
  lowCheckWidth   = 13;
  lowCommWidth    = 16;
```



```
TYPE

  MenuItem = RECORD
               nextItem      : MenuItemPtr;
               leftEdge      : INTEGER;
               topEdge       : INTEGER;
               width         : INTEGER;
               height        : INTEGER;
               flags         : MenuItemFlagSet;
               mutualExclude : LONGSET;
               itemFill      : ANYPTR;
               selectFill    : ANYPTR;
               command       : CHAR;
               subItem       : MenuItemPtr;
               nextSelect    : CARDINAL;
             END;

  RequesterFlags  = (
                  pointRel,        | TopLeft is relative to pointer for
                                   | DMRequester, relative to window
                                   | center for Request()
                  preDrawn,        | imageBMap -> predrawn Requester imagery
                  noisReg,         | requester does not filter input
                  rf3,             |
                  simpleReq,       | to use SIMPLEREFRESH layer (recommended)
                  userEqImage,     | render-order BackFill, image, gadgets,
                                   | text don't fill requester with
                                   | backFill pen
                  noReqBackFill,   |
                  rf7,             |
                  rf8,             |
                  rf9,             |
                  rf10,            |
                  rf11,            |
                  reqOffWindow,    | r/o part of the Gadgets was offwindow
                  reqActive,       | r/o this requester is active
                  sysRequest,      | (unused) this requester caused by system
                  deferRefresh );  | this Requester stops a Refresh broadcast

  RequesterFlagSet= SET OF RequesterFlags;

  Requester= RECORD
               olderRequest  : RequesterPtr;
               leftEdge      : INTEGER;
               topEdge       : INTEGER;
               width         : INTEGER;
               height        : INTEGER;
               relLeft       : INTEGER;
               relTop        : INTEGER;
               gadgets       : GadgetPtr;
               border        : BorderPtr;
               text          : IntuiTextPtr;
               flags         : RequesterFlagSet;
               backFill      : SHORTCARD;                | background pen
               layer         : LayerPtr;
```

```
              pad1          : ARRAY [0..31] OF SHORTINT;
              imageBMap     : BitMapPtr;                | if predrawn
              rWindow       : WindowPtr;                | our owner
              image         : ImagePtr;                 | if userEqImage
              reqPad2       : ARRAY [0..31] OF SHORTINT;
            END;

  GadgetFlags       = (gadgHBox,gadgHImage,gadgImage,gRelBottom,gRelRight,
                       gRelWidth,gRelHeight,selected,gadgDisabled,
                       tabCycle,stringExtend,ggf11,labelString,labelImage);
  GadgetFlagSet     = SET OF GadgetFlags;

  ActivationFlags   = (relVerify,gadgImmediate,endGadget,followMouse,
                       rightBorder,leftBorger,topBorder,bottomBorder,
                       toggleSelect,stringCenter,stringRight,longint,
                       altKeyMap,extended,activeGadget,bordersniff);
  ActivationFlagSet = SET OF ActivationFlags;

CONST
  gadgHighbits = GadgetFlagSet:{gadgHBox,gadgHImage};
  gadgHNone    = GadgetFlagSet:{gadgHBox,gadgHImage};
  gadgHComp    = GadgetFlagSet:{};
  labelIText   = GadgetFlagSet:{};
  labelMask    = GadgetFlagSet:{labelString,labelImage};
  stringLeft   = ActivationFlagSet:{};

TYPE
  GadgetTypeBits = (gtb0,gtb1,gtb2,gtb3,
                    gtb4,gtb5,gtb6,gtb7,
                    gtb8,gtb9,gbtA,gtbB,
                    reqGadget,gzzGadget,scrGadget,sysGadget);
  GadgetType     = SET OF GadgetTypeBits;

CONST
  boolGadget     = GadgetType:{gtb0};
  propGadget     = GadgetType:{gtb0,gtb1};
  strGadget      = GadgetType:{gtb2};
  customGadget   = GadgetType:{gtb0,gtb2};

  wSizing        = GadgetType:{gtb4};
  wDragging      = GadgetType:{gtb5};
  sDragging      = GadgetType:{gtb4,gtb5};
  wUpFront       = GadgetType:{gtb6};
  sUpFront       = GadgetType:{gtb4,gtb6};
  wDownBack      = GadgetType:{gtb5,gtb6};
  sDownBack      = GadgetType:{gtb4,gtb5,gtb6};
  wClose         = GadgetType:{gtb7};

TYPE
  GadgInfoPtr = POINTER TO GadgInfo;
  GadgInfo    = RECORD END;
```



```
  Gadget = RECORD
             nextGadget   : GadgetPtr;
             leftEdge     : INTEGER;
             topEdge      : INTEGER;
             width        : INTEGER;
             height       : INTEGER;
             flags        : GadgetFlagSet;
             activation   : ActivationFlagSet;
             gadgetType   : GadgetType;
             gadgetRender : ANYPTR;
             selectRender : ANYPTR;
             gadgetText   : IntuiTextPtr;
             mutualExclude: LONGSET;
             specialInfo  : GadgInfoPtr;
             gadgetID     : INTEGER;
             userData     : ANYPTR;
           END;

CONST
  boolMask = 1;

TYPE
  BoolInfo = RECORD OF GadgInfo
               flags      : BITSET;
               mask       : ANYPTR;
               reserved   : LONGCARD;
             END;

  PropInfoFlags     = (autoKnob,freeHoriz,freeVert,propBorderless,
                         propNewLook,pf5,pf6,pf7,knobHit);
  PropInfoFlagSet   = SET OF PropInfoFlags;

CONST
  knobVmin = 4;
  knobHmin = 6;
  maxBody  = $FFFF;
  maxPot   = $FFFF;

TYPE
  PropInfo = RECORD OF GadgInfo
               flags      : PropInfoFlagSet;
               horizPot   : CARDINAL;
               vertPot    : CARDINAL;
               horizBody  : CARDINAL;
               vertBody   : CARDINAL;
               cWidth     : CARDINAL;
               cHeight    : CARDINAL;
               hPotRes    : CARDINAL;
               vPotRes    : CARDINAL;
               leftBorder : CARDINAL;
               topBorder  : CARDINAL;
             END;
```

```
  StringExtFlags   = (replace,fixedField,noFilter,noChange,
                      noWorkBuffer,control,longint,exitHelp,
                      makeMeLong = 31);
  StringExtFlagSet = SET OF StringExtFlags;

  StringExtendPtr = POINTER TO StringExtend;
  StringExtend    = RECORD
                      font        : TextFontPtr;
                      pens        : ARRAY [2] OF SHORTCARD;
                      activePens  : ARRAY [2] OF SHORTCARD;
                      initModes   : StringExtFlagSet;
                      editHook    : HookPtr;
                      buffer      : SysStringPtr;
                      reserved    : ARRAY [4] OF LONGINT;
                    END;

  StringInfo = RECORD OF GadgInfo
                 buffer       : ANYPTR;
                 undoBuffer   : ANYPTR;
                 bufferPos    : INTEGER;
                 maxChars     : INTEGER;
                 dispPos      : INTEGER;
                 undoPos      : INTEGER;
                 numChars     : INTEGER;
                 dispcount    : INTEGER;
                 cLeft        : INTEGER;
                 cTop         : INTEGER;
                 extension    : StringExtendPtr;
                 longint      : LONGINT;
                 altKeyMap    : ANYPTR;
               END;


CONST
  autoFrontPen   = 0;
  autoBackPen    = 1;
  autoDrawMode   = jam2;
  autoLeftEdge   = 6;
  autoTopEdge    = 3;
  autoITextFont  = NIL;
  autoNextText   = NIL;

TYPE
  IntuiText   = RECORD
                  frontPen    : SHORTCARD;
                  backPen     : SHORTCARD;
                  drawMode    : DrawModeSet;
                  leftEdge    : INTEGER;
                  topEdge     : INTEGER;
                  iTextFont   : TextAttrPtr;
                  iText       : SysStringPtr;
                  nextText    : IntuiTextPtr;
                END;
```



```
GROUP
  IntuiTextGrp = IntuiText,IntuiTextPtr,DrawModeSet,
                 Graphics.DrawModes,autoFrontPen,autoBackPen,autoDrawMode,
                 autoLeftEdge,autoTopEdge,autoITextFont,autoNextText;

TYPE
  BorderList   = ARRAY OF RECORD x,y : INTEGER END;
  Border       = RECORD
                   leftEdge,
                   topEdge     : INTEGER;
                   frontPen,
                   backPen     : SHORTCARD;
                   drawMode    : DrawModeSet;
                   count       : SHORTCARD;
                   xy          : POINTER TO BorderList;
                   nextBorder  : BorderPtr;
                 END;

GROUP
  BorderGrp   = BorderPtr,Border,DrawModeSet,
                Graphics.DrawModes;


TYPE
  Image       = RECORD
                  leftEdge,
                  topEdge,
                  width,
                  height,
                  depth        : INTEGER;
                  imageData    : ANYPTR;
                  planePick,
                  planeOnOff   : SHORTCARD;
                  nextImage    : ImagePtr;
                END;


TYPE
  IDCMPFlags  = ( sizeVerify,        newSize,           refreshWindow,
                  mouseButtons,      mouseMove,         gadgetDown,
                  gadgetUp,          reqSet,            menuPick,
                  closeWindow,       rawKey,            reqVerify,
                  reqClear,          menuVerify,        newPrefs,
                  diskInserted,      diskRemoved,       wbenchMessage,
                  activeWindow,      inactiveWindow,    deltaMove,
                  vanillaKey,        intuiTicks,        idcmpUpdate,
                  menuHelp,          changeWindow,      c26,
                  c27,               c28,               c29,
                  c30,               lonelyMessage );
  IDCMPFlagSet  = SET OF IDCMPFlags;

  IDCMPFlagSetPtr  = POINTER TO IDCMPFlagSet;
```

```
CONST
  selectUp     = lButton+upPrefix;
  selectDown   = lButton;
  menuUp       = rButton+upPrefix;
  menuDown     = rButton;
  menuNull     = $FFFF;
  noMenu       = $1F;
  noItem       = $3F;
  noSub        = $1F;
  keyCodeQ     = $10;
  keyCodeX     = $32;
  keyCodeV     = $34;
  keyCodeB     = $35;
  keyCodeN     = $36;
  keyCodeM     = $37;
  cursorUp     = $4C;
  cursorDown   = $4D;
  cursorRight  = $4E;
  cursorLeft   = $4F;
  menuHot      = 1;
  menuCancel   = 2;
  menuWaiting  = 3;
  okOk         = menuHot;
  okAbort      = 4;
  okCancel     = menuCancel;
  wbenchOpen   = 1;
  wbenchClose  = 2;
  altLeft      = QualifierSet:{lAlt};
  altRight     = QualifierSet:{rAlt};
  amigaLeft    = QualifierSet:{lCommand};
  amigaRight   = QualifierSet:{rCommand};
  amigaKeys    = amigaLeft+amigaRight;

TYPE
  IntuiMessage = RECORD OF Message
                   class        : IDCMPFlagSet;
                   code         : CARDINAL;
                   qualifier    : QualifierSet;
                   iAddress     : ANYPTR;
                   mouseX,
                   mouseY       : INTEGER;
                   seconds,
                   micros       : LONGCARD;
                   idcmpWindow  : WindowPtr;
                   specialLink  : IntuiMessagePtr;
                 END;


GROUP
  IDCMPGrp   =  IDCMPFlags,          IDCMPFlagSet,
                selectUp,            selectDown,          menuUp,
                menuDown,            menuNull,            noMenu,
                noItem,              noSub,               keyCodeQ,
                keyCodeX,            keyCodeV,            keyCodeB,
```



```
                keyCodeN,            keyCodeM,            cursorUp,
                cursorDown,          cursorRight,         cursorLeft,
                menuHot,             menuCancel,          menuWaiting,
                okOk,                okAbort,             okCancel,
                wbenchOpen,          wbenchClose,         altLeft,
                altRight,            amigaLeft,           amigaRight,
                amigaKeys,           IntuiMessage,        IntuiMessagePtr;

TYPE
  WindowFlags   = (windowSizing,windowDrag,windowDepth,windowClose,
                   sizeBRight,sizeBBottom,simpleRefresh,superBitMap,
                   backDrop,reportMouse,gimmeZeroZero,borderless,
                   activate,windowActive,inRequest,menuState,rmbTrap,
                   noCareRefresh,nw_Extended,wf19,wf20,wf21,wf22,wf23,
                   windowRefresh,wbenchWindow,windowTicked,visitor,zoomed,
                   hasZoom,wf30,wf31);
  WindowFlagSet = SET OF WindowFlags;

  ScreenFlags   = (wbenchScreen,publicScreen,sf2,sf3,showTitle,beeping,
                   customBitMap,screenBehind,screenQuiet,screenHiRes,
                   sf10,sf11,nsExtended,sf13,autoScroll);
  ScreenFlagSet = SET OF ScreenFlags;


CONST
  defaultMouseQueue = 5;    | the default for WindowTags.mouseQueue
  otherRefresh      = WindowFlagSet:{simpleRefresh,superBitMap};
  refreshBits       = otherRefresh;
  stdScreenHeight   = -1;
  stdScreenWidth    = -1;
  customScreen      = ScreenFlagSet:{wbenchScreen..sf3};

TYPE
  NewWindowPtr = POINTER TO NewWindow;
  NewWindow    = RECORD
                   leftEdge,
                   topEdge,
                   width,
                   height      : INTEGER;
                   detailPen,
                   blockPen    : SHORTINT;
                   idcmpFlags  : IDCMPFlagSet;
                   flags       : WindowFlagSet;
                   firstGadget : GadgetPtr;
                   checkMark   : ImagePtr;
                   title       : SysStringPtr;
                   screen      : ScreenPtr;
                   bitMap      : BitMapPtr;
                   minWidth,
                   minHeight,
                   maxWidth,
                   maxHeight   : INTEGER;
                   type        : ScreenFlagSet;
                 END;
```

```
ColorSpecPtr = POINTER TO ColorSpec;
ColorSpec    = RECORD
                 colorIndex  : INTEGER;
                 red,
                 green,
                 blue        : CARDINAL;
               END;
ColorSpecs   = ARRAY OF ColorSpec;

ColorPtr     =  POINTER TO ColorSpecs;

EasyStructPtr = POINTER TO EasyStruct;
EasyStruct    = RECORD
                  structSize  : LONGCARD;
                  flags       : LONGCARD;
                  title,
                  textFormat,
                  gadgetFormat: SysStringPtr;
                END;

WindowTags      =
  TAGS OF StdTags
    left              = wTB+$01 : LONGINT;
    top               = wTB+$02 : LONGINT;
    width             = wTB+$03 : LONGINT;
    height            = wTB+$04 : LONGINT;
    detailPen         = wTB+$05 : LONGCARD;       | SHORTCARD
    blockPen          = wTB+$06 : LONGCARD;       | SHORTCARD
    IDCMP             = wTB+$07 : IDCMPFlagSet;
    flags             = wTB+$08 : WindowFlagSet;
    gadgets           = wTB+$09 : GadgetPtr;
    checkmark         = wTB+$0A : ImagePtr;
    title             = wTB+$0B : SysStringPtr;
    screenTitle       = wTB+$0C : SysStringPtr;
    customScreen      = wTB+$0D : ScreenPtr;
    superBitMap       = wTB+$0E : BitMapPtr;
    minWidth          = wTB+$0F : LONGINT;        | INTEGER
    minHeight         = wTB+$10 : LONGINT;        | INTEGER
    maxWidth          = wTB+$11 : LONGCARD;       | CARDINAL
    maxHeight         = wTB+$12 : LONGCARD;       | CARDINAL
    innerWidth        = wTB+$13 : LONGINT;
    innerHeight       = wTB+$14 : LONGINT;

    pubScreenName     = wTB+$15 : SysStringPtr;
    pubScreen         = wTB+$16 : ScreenPtr;
    pubScreenFallBack = wTB+$17 : LONGBOOL;
    windowName        = wTB+$18 : SysStringPtr;
    colors            = wTB+$19 : ColorPtr;  | may not ever be implemented
    zoom              = wTB+$1A : IBoxPtr;
    mouseQueue        = wTB+$1B : LONGCARD;  | def. defaultMouseQueue
    backFill          = wTB+$1C : HookPtr;   | see Layers.InstallLayerHook
    RPTQueue          = wTB+$1D : LONGINT;   | queue for events (see RKM)

    sizeGadget        = wTB+$1E : LONGBOOL;
    dragBar           = wTB+$1F : LONGBOOL;
```



```
    depthGadget       = wTB+$20 : LONGBOOL;
    closeGadget       = wTB+$21 : LONGBOOL;
    backDrop          = wTB+$22 : LONGBOOL;
    reportMouse       = wTB+$23 : LONGBOOL;
    noCareRefresh     = wTB+$24 : LONGBOOL;
    borderless        = wTB+$25 : LONGBOOL;
    activate          = wTB+$26 : LONGBOOL;
    RMBTrap           = wTB+$27 : LONGBOOL;
    wBenchWindow      = wTB+$28 : LONGBOOL;
    simpleRefresh     = wTB+$29 : LONGBOOL;
    smartRefresh      = wTB+$2A : LONGBOOL;
    sizeBRight        = wTB+$2B : LONGBOOL;  | SizeGadget in right border
    sizeBBottom       = wTB+$2C : LONGBOOL;  | SizeGadget in bottom Border
    autoAdjust        = wTB+$2D : LONGBOOL;
    gimmeZeroZero     = wTB+$2E : LONGBOOL;
    menuHelp          = wTB+$2F : LONGBOOL;  | see IDCMP.menuHelp
  END;

WindowTagListPtr = POINTER TO WindowTagList;
WindowTagList    = ARRAY OF WindowTags;

ExtNewWindowPtr  = POINTER TO ExtNewWindow;
ExtNewWindow     = RECORD OF NewWindow;
                     extension   : WindowTagListPtr;
                   END;

NewScreenPtr     = POINTER TO NewScreen;
NewScreen        = RECORD
                     leftEdge    : INTEGER;
                     topEdge     : INTEGER;
                     width       : INTEGER;
                     height      : INTEGER;
                     depth       : INTEGER;
                     detailPen   : SHORTINT;
                     blockPen    : SHORTINT;
                     viewModes   : ViewModeSet;
                     type        : ScreenFlagSet;
                     font        : TextAttrPtr;
                     defaultTitle: ANYPTR;
                     gadgets     : GadgetPtr;
                     customBitMap: BitMapPtr;
                   END;

| Types for ScreenTags

ErrorTypePtr     = POINTER TO ErrorType;
ErrorType        = ( noMonitor,     noChips,       noMem,
                     noChipMem,     pubNotUnique,  unknownMode,
                     mmlong = $10000 );

FontPrefs        = ( oldDefaultFont,    | The old fixed-width default
                     wbScreenFont,      | be font sensitive !!!
                     mmlong = $10000 );

OScanType        = ( text,standard,max,video );
```

```
ScreenTags       =
  TAGS OF StdTags
    left         = sTB + $01 : LONGINT;
    top          = sTB + $02 : LONGINT;
    width        = sTB + $03 : LONGINT;
    height       = sTB + $04 : LONGINT;
    depth        = sTB + $05 : LONGINT;
    detailPen    = sTB + $06 : LONGCARD;     | SHORTCARD
    blockPen     = sTB + $07 : LONGCARD;     | SHORTCARD
    title        = sTB + $08 : SysStringPtr;
    colors       = sTB + $09 : ColorPtr;     | initial Palette. Finish
                                             | with -1.
    errorCode    = sTB + $0A : ErrorTypePtr;
    font         = sTB + $0B : TextAttrPtr;
    sysFont      = sTB + $0C : FontPrefs;
    type         = sTB + $0D : ScreenFlagSet;
    bitMap       = sTB + $0E : BitMapPtr;      | custom bitmap
    pubName      = sTB + $0F : SysStringPtr;   | state before the two
                                               | below
    pubSig       = sTB + $10 : LONGINT;        | signal for pubTask
    pubTask      = sTB + $11 : TaskPtr;        | the pubscreen task
    displayID    = sTB + $12 : LONGINT;        | a custom Display
    dClip        = sTB + $13 : RectanglePtr;   | better use overScan
    overScan     = sTB + $14 : OScanType;
    obsolete1    = sTB + $15 : SysStringPtr;   | used to be Monitorname
    showTitle    = sTB + $16 : LONGBOOL;
    behind       = sTB + $17 : LONGBOOL;
    quiet        = sTB + $18 : LONGBOOL;
    autoScroll   = sTB + $19 : LONGBOOL;
    pens         = sTB + $1A : PenArrayPtr; | init screen.drawInfo.pens
    fullPalette  = sTB + $1B : LONGBOOL;    | init all preferences colors
  END;

ScreenTagListPtr = POINTER TO ScreenTagList;
ScreenTagList    = ARRAY OF ScreenTags;

ExtNewScreenPtr  = POINTER TO ExtNewScreen;
ExtNewScreen     = RECORD OF NewScreen;
                     extension    : ScreenTagListPtr;
                   END;

Window           = RECORD  OF WinRes.ResHandle
                     nextWindow   : WindowPtr;
                     leftEdge     : INTEGER;
                     topEdge      : INTEGER;
                     width        : INTEGER;
                     height       : INTEGER;
                     mouseY       : INTEGER;
                     mouseX       : INTEGER;
                     minWidth     : INTEGER;
                     minHeight    : INTEGER;
                     maxWidth     : INTEGER;
                     maxHeight    : INTEGER;
                     flags        : WindowFlagSet;
```



```
                     menuStrip    : MenuPtr;
                     title        : SysStringPtr;
                     firstRequest: RequesterPtr;
                     dmRequest    : RequesterPtr;
                     reqCount     : INTEGER;
                     wScreen      : ScreenPtr;
                     rPort        : RastPortPtr;
                     borderLeft   : SHORTINT;
                     borderTop    : SHORTINT;
                     borderRight  : SHORTINT;
                     borderBottom: SHORTINT;
                     borderRPort  : RastPortPtr;
                     firstGadget  : GadgetPtr;
                     parent       : WindowPtr;
                     descendant   : WindowPtr;
                     pointer      : ANYPTR;
                     ptrHeight    : SHORTINT;
                     ptrWidth     : [0..16];
                     xOffset      : SHORTINT;
                     yOffset      : SHORTINT;
                     idcmpFlags   : IDCMPFlagSet;
                     userPort     : MsgPortPtr;
                     windowPort   : MsgPortPtr;
                     messageKey   : IntuiMessagePtr;
                     detailPen    : SHORTCARD;
                     blockPen     : SHORTCARD;
                     checkMark    : ImagePtr;
                     screenTitle  : SysStringPtr;
                     gzzMouseX    : INTEGER;
                     gzzMouseY    : INTEGER;
                     gzzWidth     : INTEGER;
                     gzzHeight    : INTEGER;
                     extData      : ANYPTR;
                     userData     : ANYPTR;
                     wLayer       : LayerPtr;
                     iFont        : TextFontPtr;
                   END;

Screen           = RECORD OF ScreenRes.ResHandle
                     nextScreen   : ScreenPtr;
                     firstWindow  : WindowPtr;
                     leftEdge     : INTEGER;
                     topEdge      : INTEGER;
                     width        : INTEGER;
                     height       : INTEGER;
                     mouseY       : INTEGER;
                     mouseX       : INTEGER;
                     flags        : ScreenFlagSet;
                     title        : SysStringPtr;
                     defaultTitle: SysStringPtr;
                     barHeight    : SHORTINT;
                     barVBorder   : SHORTINT;
                     barHBorder   : SHORTINT;
                     menuVBorder  : SHORTINT;
                     menuHBorder  : SHORTINT;
```

```
                     wBorTop      : SHORTINT;
                     wBorLeft     : SHORTINT;
                     wBorRight    : SHORTINT;
                     wBorBottom   : SHORTINT;
                     font         : TextAttrPtr;
                     viewPort     : ViewPort;
                     rastPort     : RastPort;
                     bitMap       : BitMap;
                     layerInfo    : LayerInfo;
                     firstGadget  : GadgetPtr;
                     detailPen    : SHORTCARD;
                     blockPen     : SHORTCARD;
                     saveColor0   : CARDINAL;
                     barLayer     : LayerPtr;
                     extData      : ANYPTR;
                     userData     : ANYPTR;
                   END;

CONST
  private         =  1;
  maxPubScreenName =  139;          (* names no longer *)

TYPE
  PBNFlags         = ( shanghai, popPubScreen, makeMeWord = 15 );
  PBNFlagSet       = SET OF PBNFlags;

  PubScreenNode    = RECORD OF Node;
                       screen        : ScreenPtr;
                       flags         : PBNFlagSet;
                       size          : INTEGER;
                       visitorCount  : INTEGER;
                       sigTask       : TaskPtr;
                       sigBit        : TaskSignals;
                     END;


CONST
  filenameSize  = 30;
  pointerSize   = (1+16+1)*2;
  topazEighty   = 8;
  topazSixty    = 9;

TYPE
  PrinterPort   = (parallelPrinter,serialPrinter);


(* PrinterTypes *)
CONST
  CustomName     = 0;
  AlphaP101      = 1;
  Brother15XL    = 2;
  CbmMps1000     = 3;
  Diab630        = 4;
  DiabAdvD25     = 5;
```



```
  DiabC150       = 6;
  Epson          = 7;
  EpsonJX80      = 8;
  Okimate20      = 9;
  QumeLP20       = 10;
  HpLaserjet     = 11;
  HpLaserjetPlus = 12;

TYPE
  SerParShk      =(shakeXon,shakeRts,shakeNone,sps3,parityNone,parityEven,
                   parityOdd);

  SerParShkSet   = SET OF SerParShk;

  Print          =(ignoreDimensions,correctRed,correctGreen,correctBlue,
                   centerImage,boundedDimensions,absoluteDimensions,
                   pixelDimensions,mulitplyDimensions,integerScaling,
                   halftoneDithering,floydDithering,antiAlias,greyScale2);

  PrintFlags     = SET OF Print;


CONST
  orderedDithering = ignoreDimensions;
  correctRgbMask   = PrintFlags :{correctRed..correctBlue};
  dimensionsMask   = PrintFlags :{boundedDimensions..pixelDimensions};
  ditheringMask    = PrintFlags :{halftoneDithering,floydDithering};

TYPE

  PaperType = ( fanfold       = $0,
                single        = $080 );

  Preferences    = RECORD
                     fontHeight         : SHORTCARD;
                     printerPort        : PrinterPort;
                     baudRate           : CARDINAL;
                     keyRptSpeed        : TimeVal;
                     keyRptDelay        : TimeVal;
                     doubleClick        : TimeVal;
                     pointerMatrix      : ARRAY [0..pointerSize-1] OF CARDINAL;
                     xOffset            : SHORTINT;
                     yOffset            : SHORTINT;
                     color17            : CARDINAL;
                     color18            : CARDINAL;
                     color19            : CARDINAL;
                     pointerTicks       : CARDINAL;
                     color0             : CARDINAL;
                     color1             : CARDINAL;
                     color2             : CARDINAL;
                     color3             : CARDINAL;
                     viewXOffset        : SHORTINT;
                     viewYOffset        : SHORTINT;
                     viewInitX          : INTEGER;
                     viewInitY          : INTEGER;
```

```
                  enableCLI          : BOOLEAN;
                  printerType        : CARDINAL;
                  printerFilename    : ARRAY [0..filenameSize-1] OF CHAR;
                  printPitch         : CARDINAL;
                  printQuality       : CARDINAL;
                  printSpacing       : CARDINAL;
                  printLeftMargin    : CARDINAL;
                  printRightMargin   : CARDINAL;
                  printImage         : CARDINAL;
                  printAspect        : CARDINAL;
                  printShade         : CARDINAL;
                  printTreshhold     : INTEGER;
                  paperSize          : CARDINAL;
                  paperLength        : CARDINAL;
                  paperType          : PaperType;
                  serRWBits          : SHORTCARD;
                  serStopBuf         : SHORTCARD;
                  serParShk          : SerParShkSet;
                  laceWB             : BOOLEAN;
                  workName           : ARRAY [0..filenameSize-1] OF CHAR;
                  padding            : ARRAY [0..15] OF SHORTINT;
                END;


CONST
  baud110       = 0;
  baud300       = 1;
  baud1200      = 2;
  baud2400      = 3;
  baud4800      = 4;
  baud9600      = 5;
  baud19200     = 6;
  baudMidi      = 7;
  pica          = $0;
  elite         = $0400;
  fine          = $0800;
  draft         = $0;
  letter        = $0100;
  sixLPI        = $0;
  eightLPI      = $0200;
  imagePositive = 0;
  imageNegative = 1;
  aspectHoriz   = 0;
  aspectVert    = 1;
  shadeBW       = 0;
  shadeGreyscale= 1;
  shadeColor    = 2;
  usLetter      = $0;
  usLegal       = $010;
  nTractor      = $020;
  wTractor      = $030;
  custom        = $040;

  readBits      = $0F0;
  writeBits     = $0F;
```



```
  stopBits      = $0F0;
  bufSizeBits   = $0F;
  buf512        = 0;
  buf1024       = 1;
  buf2048       = 2;
  buf4096       = 3;
  buf8000       = 4;
  buf16000      = 5;

TYPE
  Remember      = RECORD
                    nextRemember : RememberPtr;
                    rememberSize : LONGCARD;
                    memory       : ANYPTR;
                  END;

CONST
  deadendAlert  = $80000000;
  recoveryAlert = 0;

TYPE
  DisplayMode   = (hiresPick,lowresPick);

CONST
  dMountCode    = LONGINT(DisplayMode'MAX)+1;
  eventMax      = 10;

TYPE
  Res           = (hiresGadget,lowresGadget);

CONST
  resCount      = LONGINT(Res'MAX)+1;

TYPE
  Gadgets       = (upFrontGadget,downBackGadget,sizeGadget,
                   closeGadget,dragGadget,
                   sUpFrontGadget,sDownBackGadget,sDragGadget);

CONST
  gadgetCount   = LONGINT(Gadgets'MAX)+1;

TYPE
  ILocks        = (iStateLock,layerInfoLock,gadgetsLock,layerRomLock,
                   viewLock,iBaseLock,rpLock);

CONST
  numILocks     = LONGINT(ILocks'MAX)+1;

TYPE
  FatIntuiMessagePtr = POINTER TO FatIntuiMessage;
  FatIntuiMessage    = RECORD OF IntuiMessage
                         prevKeys     : LONGCARD;
                       END;
```

```
IBox            = RECORD
                    left          : INTEGER;
                    top           : INTEGER;
                    width         : INTEGER;
                    height        : INTEGER;
                  END;

PointPtr        = POINTER TO Point;
Point           = RECORD
                    x             : INTEGER;
                    y             : INTEGER;
                  END;

PenPair         = RECORD
                    detailPen     : SHORTCARD;
                    blockPen      : SHORTCARD;
                  END;


CoordinatesPtr  = POINTER TO Coordinates;
Coordinates     = RECORD
                    x,
                    y   : CARDINAL;
                  END;



DrawInfoFlags   = (newLook,makeMeLong=31);
DrawInfoFlagSet = SET OF DrawInfoFlags;

DrawInfoPenss   = (detailPen,
                   blockPen,
                   textPen,
                   shinePen,
                   shadowPen,
                   fillPen,
                   fillTextPen,
                   backgroundPen,
                   highlightTextPen);

DrawInfoPtr     = POINTER TO DrawInfo;
DrawInfo        = RECORD
                    version       : CARDINAL;
                    numPens       : CARDINAL;
                    pens          : ANYPTR;  | Pointer to Pen-Array ??
                    font          : TextFontPtr;
                    depth         : CARDINAL;
                    resolution    : Coordinates;
                    flags         : DrawInfoFlagSet;
                    reserved      : ARRAY [0..6] OF LONGCARD;
                  END;
```



```
(*
 * Package of information passed to custom and 'boopsi'
 * gadget "hook" functions.  This structure is READ ONLY.
 *)

  GadgetInfoPtr = POINTER TO GadgetInfo;
  GadgetInfo    = RECORD
                    screen    : ScreenPtr;
                    window    : WindowPtr;
                    requester : RequesterPtr;
                    |
                    | rendering information:
                    | don't use these without cloning/locking.
                    | Official way is to call ObtainRPort()
                    |
                    rastPort  : RastPortPtr;
                    layer     : LayerPtr;
                    |
                    | copy of dimensions of screen/window/g00/req(/group)
                    | that gadget resides in.  Left/Top of this box is
                    | offset from window mouse coordinates to gadget
                    | coordinates
                    |   screen gadgets           : 0,0 (from screen coords)
                    |   window gadgets (no g00)  : 0,0
                    |   gzzGadget (borderlayer)  : 0,0
                    |   gzz innerlayer gadget    : borderleft, bordertop
                    |   Requester gadgets        : reqleft, reqtop
                    |
                    domain    : IBox;
                    pens      : PenPair;
                    |
                    | the Detail and Block pens in drInfo.pens are
                    | for the screen.  Use the above for window-sensitive
                    | colors.
                    |
                    drInfo    : DrawInfo;
                    |
                    | reserved space: this structure is extensible anyway,
                    | but using these saves some recompilation
                    | (orig. CBM comment).
                    |
                    reserved  : ARRAY [0..5] OF LONGCARD;
                  END;


CONST
  numIEvents = 4;

TYPE
  PGX           = RECORD
                    container   : IBox;
                    newKnob     : IBox;
                  END;
```

```
SGActionFlags   = (use,end,beep,reuese,redisplay,
                   nextActive,prevActive,makeMeLong=31);
SGActionFlagSet = SET OF SGActionFlags;

EditOperation   = (null,noOp,delBackward,delForward,
                   moveCursor,enter,reset,replaceChar,
                   insertChar,badFormat,bigChange,undo,
                   clear,special,
                   makeMeWord = $1000);

SGWorkPtr       = POINTER TO SGWork;
SGWork          = RECORD
                    gadget       : GadgetPtr;
                    strInfo      : StringInfoPtr;
                    workBuffer,
                    prevBuffer   : SysStringPtr;
                    modes        : StringExtFlagSet;
                    iEvent       : InputEventPtr;
                    code         : CARDINAL;
                    bufferPos,
                    numChars     : INTEGER;
                    actions      : SGActionFlagSet;
                    longInt      : LONGINT;
                    info         : GadgetInfoPtr;
                    editOp       : EditOperation;
                  END;


IClassPtr  = POINTER TO IClass;
ObjectPtr  = POINTER TO Object;
Object     = RECORD OF MinNode;
               class  :  IClassPtr;
             END;

| Dispatched method ID's
| NOTE: Applications should use Intuition entry points, not direct
| DoMethod() calls, for NewObject, DisposeObject, SetAttrs,
| SetGadgetAttrs, and GetAttr.
|
| the following enum is for clean access for magic string values in
| variant RECORD ClassId just below.

MethodID = (dummy   = $100,  | to start us off
            new,             | 'object' parameter is "true class"
            dispose,         | delete self (no parameters)
            set,             | set attributes (in tag list)
            get,             | return single attribute value
            addTail,         | add self to a List (let root do it)
            remove,          | remove self from list
            notify,          | send to self: notify dependents
            update,          | notification message from somebody
            addMember,       | used by various classes with lists
            remMember,       | used by various classes with lists
            ldummy = $10000);| dummy to make the enum a long value
```



```
  ClassId   = SysStringPtr;

  |
  | you can use this type to point to a "generic" message,
  | in the object-oriented programming parlance.  Based on
  | the value of 'MethodID', you dispatch to processing
  | for the various message types.  The meaningful parameter
  | packet structure definitions are defined below.
  |
  Msg      = POINTER TO MsgRoot;
  MsgRoot  = RECORD
               methodId  : MethodID;
               | method-specific data follows, some examples below
             END;

CONST
  RootClass      = "rootclass";
  ImageClass     = "imageclass";
  FrameIClass    = "frameiclass";
  SysIClass      = "sysiclass";
  FillRectClass  = "fillrectclass";
  GadgetClass    = "gadgetclass";
  PropGClass     = "propgclass";
  StrGClass      = "strgclass";
  ButtonGClass   = "buttongclass";
  FRButtonGClass = "frbuttonclass";
  GroupGClass    = "groupgclass";
  ICClass        = "icclass";
  ModelClass     = "modelclass";

TYPE
  |
  | new and set:
  |
  OpSetPtr  = POINTER TO OpSet;
  OpSet     = RECORD OF MsgRoot
                attrList  : ANYPTR;        | Taglist of any tags
                gInfo     : GadgetInfoPtr; | always there for gadgets,
              END;                         | when SetGadgetAttrs() is used,
                                           | but will be NULL for new

  |
  | notify and update:
  |
  | this flag (interim) means that the update message is being issued
  | from something like an active gadget, a la followMouse.
  | When the gadget goes inactive,
  | it will issue a final update message with this bit cleared
  | Examples of use are for followMouse equivalents for propgadclass, and
  | repeat strobes for buttons.
  |
  OpuFlags   =  (   interim,  opufmax = 31 );
```

```
OpuFlagSet =  SET OF OpuFlags;
|
| Message for notify or update Methods
|
OpUpdatePtr = POINTER TO OpUpdate;
OpUpdate    = RECORD OF MsgRoot
                attrList  : ANYPTR;    | TagList of new attributes
                gInfo     : GadgetPtr; | non-NULL when SetGadgetAttrs or
                                       | notification resultingfrom gadget
                                       | input occurs.
                flags     : OpuFlagSet;
              END;

|
| Message for Method 'get'
|
OpGetPtr    = POINTER TO OpGet;
OpGet       = RECORD OF MsgRoot
                attrId    : LONGCARD;
                storage   : ANYPTR; | may be other types, but
                                    | "int" , types are all LONGCARD
              END;

|
| 'addTail' Message
|
OpAddTailPtr = POINTER TO OpAddTail;
OpAddTail    = RECORD OF MsgRoot
                 list     : ListPtr;
               END;

|
| addMember and remMember Message
|
OpMemberPtr = POINTER TO OpMember;
OpMember    = RECORD OF MsgRoot
                object   : ObjectPtr;
              END;

IClassFlags     = ( inPublicList, icfmax=31 );
IClassFlagSet   = SET OF IClassFlags;

IClass    =  RECORD
               dispatcher   :  Hook;
               reserved     :  LONGCARD;
               super        :  IClassPtr;
               id           :  ClassId;

               instOffset   :  CARDINAL;
               instSize     :  CARDINAL;
               userData     :  LONGCARD;
               subClassCount:  LONGCARD;
               objectCount  :  LONGCARD;
               flags        :  LONGCARD;
             END;
```



```
  Class = IClass;



|
|   GadgetClass
|


CONST
  |
  | Tag offsets
  |
  gaTB          = tagUser + $30000;  | OffSet GadgetClass attributeTags
  pgaTB         = gaTB    + $01000;  | Offset PropGClass attributeTags
  strgTB        = gaTB    + $02000;  | Offset STRGClass attributeTags
  layoTB        = gaTB    + $08000;  | Offset Gadget layout attributeTags

  defaultMaxChars   = 128;


TYPE
  GadgetTags        =
    TAGS OF StdTags
      dummy = tagUser + $30000;
      left            : LONGINT;
      relRight        : LONGINT;
      top             : LONGINT;
      relBottom       : LONGINT;
      width           : LONGINT;
      relWidth        : LONGINT;
      height          : LONGINT;
      relHeight       : LONGINT;
      text            : SysStringPtr;
      image           : ImagePtr;
      border          : BorderPtr;
      selectRender    : ANYPTR;
      highLight       : LONGINT; | ????
      disabled        : LONGBOOL;
      gzzGadget       : LONGBOOL;
      id              : LONGINT;
      userData        : ANYPTR;
      specialInfo     : GadgInfoPtr;
      selected        : LONGBOOL;
      endGadget       : LONGBOOL;
      immediate       : LONGBOOL;
      relVerify       : LONGBOOL;
      followMouse     : LONGBOOL;
      rightBorder     : LONGBOOL;
      leftBorder      : LONGBOOL;
      topBorder       : LONGBOOL;
      bottomBorder    : LONGBOOL;
      toggleSelect    : LONGBOOL;
```

```
    sysGadget      : LONGBOOL;
                   |
                   | bool, sets GTYP_SYSGADGET field in type
                   |
    sysGType       : GadgetType;     | internal use only
                   |
                   | e.g., wUpFront , ...
                   |
    previous       : GadgetPtr;
                   |
                   | previous gadget (or GadgetPtrPtr) in linked list
                   | This attribute CANNOT be used to link new gadgets
                   | to an open window or requester. Use AddGList().
                   |
    next           : GadgetPtr;      | not implemented
    drawInfo       : DrawInfoPtr;
                   |
                   | some fancy gadgets need to see a DrawInfo
                   | when created or for layout
                   |
    intuiText      : IntuiTextPtr;
                   |
                   | use at most ONE of text, intuiText, or labelImage
                   |
    labelImage     : ImagePtr;
                   |
                   | an Image object used in place of GadgetText
                   |
    tabCycle       : LONGBOOL;
                   |
                   | indicates that this gadget is to participate in
                   | cycling activation with Tab or Shift-Tab.
  END;

GadgetTagListPtr  = POINTER TO GadgetTagList;
GadgetTagList     = ARRAY OF GadgetTags;


OrientationFlags   = (horizontal,vertical,makeMeLong = 31);
OrientationFlagSet = SET OF OrientationFlags;

PropGTags         = TAGS OF GadgetTags
                      dummy = tagUser + $31000;

                      freedom      : OrientationFlagSet;
                      borderless   : LONGCARD;
                      horizPot     : LONGCARD;
                      horizBody    : LONGCARD;
                      vertPot      : LONGCARD;
                      vertBody     : LONGCARD;
                      total        : LONGCARD;
                      visible      : LONGCARD;
                      top          : LONGCARD;
                      newLook      : LONGBOOL;
                    END;
```



```
  TagStrgPtr        = POINTER TO StringTags;
  StringTags        = TAGS OF GadgetTags
                        dummy = tagUser + $32000;

                        maxChars         : LONGINT;
                        buffer           : SysStringPtr;
                        undoBuffer       : SysStringPtr;
                        workBuffer       : SysStringPtr;
                        bufferPos        : SysStringPtr;
                        dispPos          : LONGINT;
                        altKeyMap        : ANYPTR;
                        font             : LONGINT;
                        pens             : LONGINT;
                        activePens       : LONGINT;
                        editHook         : LONGINT;
                        editModes        : LONGINT;

                        replaceMode      : BOOLEAN;
                        fixedFieldMode   : BOOLEAN;
                        noFilterMode     : BOOLEAN;

                        justification    : GadgetFlagSet;

                        longVal          : LONGINT;
                        textVal          : SysStringPtr;

                        exitHelp         : LONGBOOL;
                      END;

  StringTagListPtr  = POINTER TO StringTagList;
  StringTagList     = ARRAY OF StringTags;

  TagLayoutPtr      = POINTER TO LayoutTags;
  LayoutTags        = TAGS OF GadgetTags
                        dummy = tagUser + $38000;
                        layoutObj       : LONGINT;
                        spacing         : LONGINT;
                        orientation     : OrientationFlagSet;
                      END;

  LayoutTagListPtr  = POINTER TO LayoutTagList;
  LayoutTagList     = ARRAY OF LayoutTags;

CONST
    hitTest     = MethodID(0); | return 'gadgetHit' if you are clicked on
                               | (whether or not you are disabled).
    render      = MethodID(1); | draw yourself, in the appropriate state
    goActive    = MethodID(2); | you are now going to be fed input
    handleInput = MethodID(3); | handle that input
    goInactive  = MethodID(4); | whether or not by choice, you are done

  | HitTest-Return Value :
  gadgetNotHit      =  0;
  gadgetHit         =  4;
```

```
TYPE
  |
  | Parameter "Messages" passed to gadget class methods
  |
  | 'hitTest' Message:
  |
  HitTest           = RECORD OF MsgRoot;
                        gInfo       : GadgInfo;
                        mouse       : Point;
                      END;

  | values for Render.redraw
  |
  RedrawType    = (toggle,reDraw,update,makemelong = 1000000 );

  |
  | 'render' message:
  |
  Render            = RECORD OF MsgRoot;
                        gInfo       : GadgInfo;
                        rPort       : RastPort;
                        redraw      : RedrawType;
                      END;

TYPE
  |
  | 'goActive' and 'handleInput' messages:
  |
  GPInput           = RECORD OF MsgRoot;
                        gInfo       : GadgInfo;
                        iEvent      : InputEvent;
                        termination : POINTER TO LONGINT;
                        mouse       : Point;
                      END;


  InputGoActiveCodeFlags = (iga0,noReUse,reUse,verify,nextActive,
                            prevActive);
  IGoActFlagSet          = SET OF InputGoActiveCodeFlags;


  |
  | 'goInactive' message:
  |
  GoInactive        = RECORD OF MsgRoot;
                        gInfo       : GadgInfo;
                        abort       : LONGBOOL;
                      END;
```



```
(* IcClass starts here : *)

CONST
  ioCdummy      = MethodID($0401); | used for nothing
  setLoop       = MethodID($0402); | set/increment loop counter
  clearLoop     = MethodID($0403); | clear/decrement loop counter
  checkLoop     = MethodID($0404); | set/increment loop

  icaTB          = tagUser + $40000;
  icTargetIDCMP  = -1;   (* ? *)

TYPE

  |
  | InterConnection Attribute Tags
  |
  ICATags     = TAGS OF StdTags
                  dummy = tagUser + $40000;
                  target : LONGINT;
                  map    : LONGINT;
                  code   : LONGINT;
                END;

  ICATagListPtr = POINTER TO ICATagList;
  ICATagList    = ARRAY OF ICATags;


CONST
  customImageDepth    = -1;


TYPE
  SYSIAWichValues   = (depthImage=0,zoomImage,sizeImage,closeImage,
                       sDepthImage=5,leftImage=$A,upImage,rightImage,
                       downImage,checkImage,mxImage);

  SYSIASizeValues   = (medRes=0,lowRes,hiRes);


  ImageMessageId    = (draw=$202,
                       hitTest,    | return TRUE if click hits image
                       erase,      | erase yourself
                       move,       | draw new and erase old, smoothly
                       drawFrame,  | draw with specified dimensions
                       frameBox,   | get recommended frame around some box
                       hitFrame,   | hittest with dimensions
                       eraseFrame  | hittest with dimensions
                      );
```

```
(* image draw states or styles, for IM_DRAW *)
 ImageDrawStates    = (normal=0,
                      selected,        | for selected gadgets
                      disabled,        | for disabled gadgets
                      busy,            | for future functionality
                      indeterminant,   | for future functionality
                      inactiveNormal,  | normal, in inactive window border
                      inactiveSelected,| selected, in inactive border
                      inactiveDisabled | disabled, in inactive border
                      );

  TagImagePtr       = POINTER TO ImageTags;
  ImageTags         = TAGS OF StdTags
                       dummy = tagUser + $20000;
                       left         : LONGINT;
                       top          : LONGINT;
                       width        : LONGINT;
                       height       : LONGINT;
                       fgPen        : LONGINT;
                       bgPen        : LONGINT;
                       data         : LONGINT;
                       lineWidth    : LONGINT;
                       sysIAShadowPen     : LONGINT;
                       sysIAHighLightPen  : LONGINT;
                       size         : SYSIASizeValues;
                       sysIADepth   : LONGINT;
                       sysIAWich    : SYSIAWichValues;
                       pens         : ANYPTR; (* ? Pointer to Pens[] *)
                       resolution   : LONGCARD;
                       aPattern     : LONGINT;
                       aPatSize     : LONGINT;
                       mode         : LONGINT;
                       font         : LONGINT;
                       outLine      : LONGINT;
                       recessed     : LONGINT;
                       doubleEmboss : LONGINT;
                       edgesOnly    : LONGINT;
                       sysIADrawInfo  : LONGINT;
                     END;

  ImageTagListPtr   = POINTER TO ImageTagList;
  ImageTagList      = ARRAY OF ImageTags;

  ImpFrameBox   = RECORD OF MsgRoot;
                    contentsBox   : IBoxPtr; | input: relative box
                                             | of contents
                    frameBox      : IBoxPtr; | output: rel. box of
                                             | encl frame
                    drInfo        : DrawInfo;
                    frameFlags    : LONGCARD;
                  END;
```



```
  (* es folgen seltsame Macros.... ? *)

  DimensionRec  = RECORD
                    width   : INTEGER;
                    height  : INTEGER;
                  END;

  ImpDraw       = RECORD
                    methodId    : LONGCARD;
                    rPort       : RastPort;
                    offset      : Point;
                    state       : LONGCARD;
                    drInfo      : DrawInfo;
                    (* this parameters only valid in DRAWFRAME *)
                    dimensions  : DimensionRec;
                  END;

(* IM_ERASE, IM_ERASEFRAME      *)
(* NOTE: This is a subset of impDraw    *)
  ImpErase      = RECORD
                    methodId    : LONGCARD;
                    rPort       : RastPort;
                    offset      : Point;
                    (* this parameters only valid in ERASEFRAME *)
                    dimensions  : DimensionRec;
                  END;

  ImpHitTest    = RECORD
                    methodId    : LONGCARD;
                    point       : Point;
                    (* this parameters only valid in HITFRAME *)
                    dimensions  : DimensionRec;
                  END;




(* ------------------ Procedures ----------------------- *)


TYPE
  IntuitionBasePtr  = POINTER TO IntuitionBaseType;
  IntuitionBaseType = RECORD OF Library
                        viewLord      : View;
                        activeWindow  : WindowPtr;
                        activeScreen  : ScreenPtr;
                        firstScreen   : ScreenPtr;
                        flags         : LONGSET;
                        mousey        : INTEGER;
                        mousex        : INTEGER;
                        seconds       : LONGCARD;
                        micros        : LONGCARD;
                      END;
```

```
VAR
  IntuitionBase : IntuitionBasePtr;

LIBRARY IntuitionBase BY -30
  PROCEDURE OpenIntuition();

LIBRARY IntuitionBase BY -36
  PROCEDURE Intuition( iEvent IN A0: InputEventPtr );

LIBRARY IntuitionBase BY -42
  PROCEDURE AddGadget( window   IN A0: WindowPtr;
                       gadget   IN A1: GadgetPtr;
                       position IN D0: LONGCARD ): LONGCARD;

LIBRARY IntuitionBase BY -48
  PROCEDURE ClearDMRequest( window IN A0: WindowPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -54
  PROCEDURE ClearMenuStrip( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -60
  PROCEDURE ClearPointer( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -66
  PROCEDURE CloseScreen( screen IN A0: ScreenPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -72
  PROCEDURE CloseWindow( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -78
  PROCEDURE CloseWorkbench(): LONGBOOL;

LIBRARY IntuitionBase BY -78
  PROCEDURE CloseWorkBench(): LONGBOOL;

LIBRARY IntuitionBase BY -84
  PROCEDURE CurrentTime( VAR seconds IN A0: LONGCARD;
                         VAR micros  IN A1: LONGCARD );

LIBRARY IntuitionBase BY -90
  PROCEDURE DisplayAlert(     alertNumber IN D0: LONGCARD;
                          REF string      IN A0: STRING;
                              height      IN D1: LONGINT ): LONGBOOL;

LIBRARY IntuitionBase BY -96
  PROCEDURE DisplayBeep( screen IN A0: ScreenPtr );

LIBRARY IntuitionBase BY -102
  PROCEDURE DoubleClick( sSeconds IN D0 : LONGCARD;
                         sMicros  IN D1 : LONGCARD;
                         cSeconds IN D2 : LONGCARD;
                         cMicros  IN D3 : LONGCARD ): LONGBOOL;
```



```
LIBRARY IntuitionBase BY -108
  PROCEDURE DrawBorder( rp         IN A0: RastPortPtr;
                        border     IN A1: BorderPtr;
                        leftOffset IN D0: LONGINT;
                        topOffset  IN D1: LONGINT );

LIBRARY IntuitionBase BY -114
  PROCEDURE DrawImage( rp         IN A0: RastPortPtr;
                       image      IN A1: ImagePtr;
                       leftOffset IN D0: LONGINT;
                       topOffset  IN D1: LONGINT );

LIBRARY IntuitionBase BY -120
  PROCEDURE EndRequest( requester IN A0: RequesterPtr;
                        window    IN A1: WindowPtr );

LIBRARY IntuitionBase BY -126
  PROCEDURE GetDefPrefs( preferences IN A0: PreferencesPtr;
                         size        IN D0: LONGINT ): PreferencesPtr;

LIBRARY IntuitionBase BY -132
  PROCEDURE GetPrefs( preferences IN A0: PreferencesPtr;
                      size        IN D0: LONGINT ): PreferencesPtr;

LIBRARY IntuitionBase BY -138
  PROCEDURE InitRequester( requester IN A0: RequesterPtr );

LIBRARY IntuitionBase BY -144
  PROCEDURE ItemAddress( menuStrip  IN A0: MenuPtr;
                         menuNumber IN D0: LONGCARD ): MenuItemPtr;

LIBRARY IntuitionBase BY -150
  PROCEDURE ModifyIDCMP( window IN A0: WindowPtr;
                         flags  IN D0: IDCMPFlagSet ): LONGBOOL;

LIBRARY IntuitionBase BY -156
  PROCEDURE ModifyProp( gadget    IN A0: GadgetPtr;
                        window    IN A1: WindowPtr;
                        requester IN A2: RequesterPtr;
                        flags     IN D0: LONGCARD;
                        horizPot  IN D1: LONGCARD;
                        vertPot   IN D2: LONGCARD;
                        horizBody IN D3: LONGCARD;
                        vertBody  IN D4: LONGCARD );

LIBRARY IntuitionBase BY -162
  PROCEDURE MoveScreen( screen IN A0: ScreenPtr;
                        dx     IN D0: LONGINT;
                        dy     IN D1: LONGINT );

LIBRARY IntuitionBase BY -168
  PROCEDURE MoveWindow( window IN A0: WindowPtr;
                        dx     IN D0: LONGINT;
                        dy     IN D1: LONGINT );
```

```
LIBRARY IntuitionBase BY -174
  PROCEDURE OffGadget( gadget    IN A0: GadgetPtr;
                       window    IN A1: WindowPtr;
                       requester IN A2: RequesterPtr );

LIBRARY IntuitionBase BY -180
  PROCEDURE OffMenu( window     IN A0: WindowPtr;
                     menuNumber IN D0: LONGCARD );

LIBRARY IntuitionBase BY -186
  PROCEDURE OnGadget( gadget    IN A0: GadgetPtr;
                      window    IN A1: WindowPtr;
                      requester IN A2: RequesterPtr );

LIBRARY IntuitionBase BY -192
  PROCEDURE OnMenu( window     IN A0: WindowPtr;
                    menuNumber IN D0: LONGCARD );

LIBRARY IntuitionBase BY -198
  PROCEDURE OpenScreen( REF newScreen IN A0: NewScreen ): ScreenPtr;

LIBRARY IntuitionBase BY -204
  PROCEDURE OpenWindow( REF newWindow IN A0: NewWindow ): WindowPtr;

LIBRARY IntuitionBase BY -210
  PROCEDURE OpenWorkbench(): LONGCARD;

LIBRARY IntuitionBase BY -210
  PROCEDURE OpenWorkBench(): LONGCARD;

LIBRARY IntuitionBase BY -216
  PROCEDURE PrintIText( rp    IN A0: RastPortPtr;
                        iText IN A1: IntuiTextPtr;
                        left  IN D0: LONGINT;
                        top   IN D1: LONGINT );

LIBRARY IntuitionBase BY -222
  PROCEDURE RefreshGadgets( gadgets   IN A0: GadgetPtr;
                            window    IN A1: WindowPtr;
                            requester IN A2: RequesterPtr );

LIBRARY IntuitionBase BY -228
  PROCEDURE RemoveGadget( window IN A0: WindowPtr;
                          gadget IN A1: GadgetPtr ): LONGCARD;

LIBRARY IntuitionBase BY -234
  PROCEDURE ReportMouse( flag   IN D0: LONGBOOL;
                         window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -240
  PROCEDURE Request( requester IN A0: RequesterPtr;
                     window    IN A1: WindowPtr ): LONGBOOL;
```



```
LIBRARY IntuitionBase BY -246
  PROCEDURE ScreenToBack( screen IN A0: ScreenPtr );

LIBRARY IntuitionBase BY -252
  PROCEDURE ScreenToFront( screen IN A0: ScreenPtr );

LIBRARY IntuitionBase BY -258
  PROCEDURE SetDMRequest( window    IN A0: WindowPtr;
                          requester IN A1: RequesterPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -264
  PROCEDURE SetMenuStrip( window IN A0: WindowPtr;
                          menu   IN A1: MenuPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -270
  PROCEDURE SetPointer( window  IN A0: WindowPtr;
                        pointer IN A1: ANYPTR;
                        height  IN D0: LONGINT;
                        width   IN D1: LONGINT;
                        xOffset IN D2: LONGINT;
                        yOffset IN D3: LONGINT );

LIBRARY IntuitionBase BY -276
  PROCEDURE SetWindowTitles(     window      IN A0: WindowPtr;
                             VAR windowTitle IN A1: SHORTCARD;
                             VAR screenTitle IN A2: SHORTCARD );

LIBRARY IntuitionBase BY -282
  PROCEDURE ShowTitle( screen IN A0: ScreenPtr;
                       showIt IN D0: LONGBOOL );

LIBRARY IntuitionBase BY -288
  PROCEDURE SizeWindow( window IN A0: WindowPtr;
                        dx     IN D0: LONGINT;
                        dy     IN D1: LONGINT );

LIBRARY IntuitionBase BY -294
  PROCEDURE ViewAddress(): ViewPtr;

LIBRARY IntuitionBase BY -300
  PROCEDURE ViewPortAddress( window IN A0: WindowPtr ): ViewPortPtr;

LIBRARY IntuitionBase BY -306
  PROCEDURE WindowToBack( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -312
  PROCEDURE WindowToFront( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -318
  PROCEDURE WindowLimits( window    IN A0: WindowPtr;
                          widthMin  IN D0: LONGINT;
                          heightMin IN D1: LONGINT;
                          widthMax  IN D2: LONGCARD;
                          heightMax IN D3: LONGCARD ): LONGBOOL;
```

```
LIBRARY IntuitionBase BY -324
  PROCEDURE SetPrefs( preferences IN A0: PreferencesPtr;
                      size        IN D0: LONGINT;
                      inform      IN D1: LONGBOOL ): PreferencesPtr;

LIBRARY IntuitionBase BY -330
  PROCEDURE IntuiTextLength( iText IN A0: IntuiTextPtr ): LONGINT;

LIBRARY IntuitionBase BY -336
  PROCEDURE WBenchToBack(): LONGBOOL;

LIBRARY IntuitionBase BY -342
  PROCEDURE WBenchToFront(): LONGBOOL;

LIBRARY IntuitionBase BY -348
  PROCEDURE AutoRequest( window  IN A0: WindowPtr;
                         body    IN A1: IntuiTextPtr;
                         posText IN A2: IntuiTextPtr;
                         negText IN A3: IntuiTextPtr;
                         pFlag   IN D0: LONGCARD;
                         nFlag   IN D1: LONGCARD;
                         width   IN D2: LONGINT;
                         height  IN D3: LONGINT ): LONGBOOL;

LIBRARY IntuitionBase BY -354
  PROCEDURE BeginRefresh( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -360
  PROCEDURE BuildSysRequest( window  IN A0: WindowPtr;
                             body    IN A1: IntuiTextPtr;
                             posText IN A2: IntuiTextPtr;
                             negText IN A3: IntuiTextPtr;
                             flags   IN D0: LONGCARD;
                             width   IN D1: LONGINT;
                             height  IN D2: LONGINT ): WindowPtr;

LIBRARY IntuitionBase BY -366
  PROCEDURE EndRefresh( window   IN A0: WindowPtr;
                        complete IN D0: LONGBOOL );

LIBRARY IntuitionBase BY -372
  PROCEDURE FreeSysRequest( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -378
  PROCEDURE MakeScreen( screen IN A0: ScreenPtr );

LIBRARY IntuitionBase BY -384
  PROCEDURE RemakeDisplay();

LIBRARY IntuitionBase BY -390
  PROCEDURE RethinkDisplay();
```



```
LIBRARY IntuitionBase BY -396
  PROCEDURE AllocRemember( VAR rememberKey IN A0: RememberPtr;
                               size        IN D0: LONGCARD;
                               flags       IN D1: LONGCARD ): ANYPTR;

LIBRARY IntuitionBase BY -408
  PROCEDURE FreeRemember( VAR rememberKey  IN A0: RememberPtr;
                              reallyForget IN D0: LONGBOOL );

LIBRARY IntuitionBase BY -414
  PROCEDURE LockIBase( dontknow IN D0: LONGCARD ): LONGCARD;

LIBRARY IntuitionBase BY -420
  PROCEDURE UnlockIBase( ibLock IN A0: LONGCARD );

LIBRARY IntuitionBase BY -426
  PROCEDURE GetScreenData( buffer IN A0: ANYPTR;
                           size   IN D0: LONGCARD;
                           type   IN D1: LONGCARD;
                           screen IN A1: ScreenPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -432
  PROCEDURE RefreshGList( gadgets   IN A0: GadgetPtr;
                          window    IN A1: WindowPtr;
                          requester IN A2: RequesterPtr;
                          numGad    IN D0: LONGINT );

LIBRARY IntuitionBase BY -438
  PROCEDURE AddGList( window    IN A0: WindowPtr;
                      gadget    IN A1: GadgetPtr;
                      position  IN D0: LONGCARD;
                      numGad    IN D1: LONGINT;
                      requester IN A2: RequesterPtr ): LONGCARD;

LIBRARY IntuitionBase BY -444
  PROCEDURE RemoveGList( window IN A0: WindowPtr;
                         gadget IN A1: GadgetPtr;
                         numGad IN D0: LONGINT ): LONGCARD;

LIBRARY IntuitionBase BY -450
  PROCEDURE ActivateWindow( window IN A0: WindowPtr ): LONGINT;

LIBRARY IntuitionBase BY -456
  PROCEDURE RefreshWindowFrame( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -462
  PROCEDURE ActivateGadget( gadgets   IN A0: GadgetPtr;
                            window    IN A1: WindowPtr;
                            requester IN A2: RequesterPtr ): LONGBOOL;
```

```
LIBRARY IntuitionBase BY -468
  PROCEDURE NewModifyProp( gadget    IN A0: GadgetPtr;
                           window    IN A1: WindowPtr;
                           requester IN A2: RequesterPtr;
                           flags     IN D0: LONGCARD;
                           horizPot  IN D1: LONGCARD;
                           vertPot   IN D2: LONGCARD;
                           horizBody IN D3: LONGCARD;
                           vertBody  IN D4: LONGCARD;
                           numGad    IN D5: LONGINT );

LIBRARY IntuitionBase BY -474
  PROCEDURE QueryOverscan( displayID IN A0: LONGCARD;
                           rect      IN A1: RectanglePtr;
                           oScanType IN D0: LONGINT ): LONGINT;

LIBRARY IntuitionBase BY -480
  PROCEDURE MoveWindowInFrontOf( window       IN A0: WindowPtr;
                                 behindWindow IN A1: WindowPtr );

LIBRARY IntuitionBase BY -486
  PROCEDURE ChangeWindowBox( window IN A0: WindowPtr;
                             left   IN D0: LONGINT;
                             top    IN D1: LONGINT;
                             width  IN D2: LONGINT;
                             height IN D3: LONGINT );

LIBRARY IntuitionBase BY -492
  PROCEDURE SetEditHook( hook IN A0: HookPtr ): HookPtr;

LIBRARY IntuitionBase BY -498
  PROCEDURE SetMouseQueue( window      IN A0: WindowPtr;
                           queueLength IN D0: LONGCARD ): LONGINT;

LIBRARY IntuitionBase BY -504
  PROCEDURE ZipWindow( window IN A0: WindowPtr );

LIBRARY IntuitionBase BY -510
  PROCEDURE LockPubScreen( name IN A0: SysStringPtr ): ScreenPtr;

LIBRARY IntuitionBase BY -516
  PROCEDURE UnlockPubScreen( name   IN A0: SysStringPtr;
                             screen IN A1: ScreenPtr );

LIBRARY IntuitionBase BY -522
  PROCEDURE LockPubScreenList(): ListPtr;

LIBRARY IntuitionBase BY -528
  PROCEDURE UnlockPubScreenList();

LIBRARY IntuitionBase BY -534
  PROCEDURE NextPubScreen( screen  IN A0: ScreenPtr;
                           namebuf IN A1: SysStringPtr ): SysStringPtr;
```



```
LIBRARY IntuitionBase BY -540
  PROCEDURE SetDefaultPubScreen( name IN A0: SysStringPtr );

LIBRARY IntuitionBase BY -546
  PROCEDURE SetPubScreenModes( modes IN D0: LONGCARD ): LONGCARD;

LIBRARY IntuitionBase BY -552
  PROCEDURE PubScreenStatus( screen      IN A0: ScreenPtr;
                             statusFlags IN D0: LONGCARD ): LONGCARD;

LIBRARY IntuitionBase BY -558
  PROCEDURE ObtainGIRPort( gInfo IN A0: GadgetInfoPtr ): RastPortPtr;

LIBRARY IntuitionBase BY -564
  PROCEDURE ReleaseGIRPort( rp IN A0: RastPortPtr );

LIBRARY IntuitionBase BY -570
  PROCEDURE GadgetMouse( gadget     IN A0: GadgetPtr;
                         gInfo      IN A1: GadgetInfoPtr;
                         mousePoint IN A2: Point );

(* private
LIBRARY IntuitionBase BY -576
  PROCEDURE SetIPrefs( ptr  IN A0: PreferencesPtr;
                       size IN D0: LONGINT;
                       type IN D1: LONGBOOL ): PreferencesPtr;
****)

(* public *)

LIBRARY IntuitionBase BY -582
  PROCEDURE GetDefaultPubScreen( nameBuffer IN A0: SysStringPtr );

LIBRARY IntuitionBase BY -588
  PROCEDURE EasyRequest( window     IN A0: WindowPtr;
                         easyStruct IN A1: EasyStructPtr;
                         idcmpPtr   IN A2: IDCMPFlagSetPtr;
                         args       IN A3: LIST OF LONGINT ): LONGINT;

LIBRARY IntuitionBase BY -594
  PROCEDURE BuildEasyRequestArgs( window     IN A0: WindowPtr;
                                  easyStruct IN A1: EasyStructPtr;
                                  idcmp      IN D0: IDCMPFlagSet;
                                  args       IN A3: ANYPTR ): WindowPtr;
LIBRARY IntuitionBase BY -594
  PROCEDURE BuildEasyRequest( window     IN A0: WindowPtr;
                              easyStruct IN A1: EasyStructPtr;
                              idcmp      IN D0: IDCMPFlagSet;
                              args       IN A3: LIST OF LONGINT):WindowPtr;
LIBRARY IntuitionBase BY -600
  PROCEDURE SysReqHandler( window    IN A0: WindowPtr;
                           idcmpPtr  IN A1: IDCMPFlagSetPtr;
                           waitInput IN D0: LONGBOOL ): LONGINT;
```

```
LIBRARY IntuitionBase BY -606
  PROCEDURE OpenWindowTags( newWindow IN A0: NewWindowPtr;
                            tagList   IN A1: LIST OF WindowTags):WindowPtr;
LIBRARY IntuitionBase BY -606
  PROCEDURE OpenWindowTagList(newWindow IN A0: NewWindowPtr;
                              tagList   IN A1: WindowTagListPtr):WindowPtr;
LIBRARY IntuitionBase BY -612
  PROCEDURE OpenScreenTags( newScreen IN A0: NewScreenPtr;
                            tagList   IN A1: LIST OF ScreenTags):ScreenPtr;
LIBRARY IntuitionBase BY -612
  PROCEDURE OpenScreenTagList(newScreen IN A0: NewScreenPtr;
                              tagList   IN A1: ScreenTagListPtr):ScreenPtr;
LIBRARY IntuitionBase BY -618
  PROCEDURE DrawImageState( rp         IN A0: RastPortPtr;
                            image      IN A1: ImagePtr;
                            leftOffset IN D0: LONGINT;
                            topOffset  IN D1: LONGINT;
                            state      IN D2: LONGCARD;
                            drawInfo   IN A2: DrawInfoPtr );

LIBRARY IntuitionBase BY -624
  PROCEDURE PointInImage( point IN D0: Point;
                          image IN A0: ImagePtr ): LONGBOOL;

LIBRARY IntuitionBase BY -630
  PROCEDURE EraseImage( rp         IN A0: RastPortPtr;
                        image      IN A1: ImagePtr;
                        leftOffset IN D0: LONGINT;
                        topOffset  IN D1: LONGINT );

LIBRARY IntuitionBase BY -636
  PROCEDURE NewObjectA( class   IN A0: IClassPtr;
                        classID IN A1: ClassId;  (* SysStringPtr *)
                        tagList IN A2: TagArrayPtr ): ANYPTR;

LIBRARY IntuitionBase BY -642
  PROCEDURE DisposeObject( object IN A0: ANYPTR );

LIBRARY IntuitionBase BY -648
  PROCEDURE SetAttrs( object  IN A0: ANYPTR;
                      tagList IN A1: LIST OF StdTags ): LONGCARD;

LIBRARY IntuitionBase BY -648
  PROCEDURE SetAttrsA( object  IN A0: ANYPTR;
                       tagList IN A1: TagArrayPtr ): LONGCARD;

LIBRARY IntuitionBase BY -654
  PROCEDURE GetAttr( attrID     IN D0: LONGCARD;
                     object     IN A0: ANYPTR;
                     storagePtr IN A1: ANYPTR ): LONGCARD;      (* ? *)
```



```
LIBRARY IntuitionBase BY -660
  PROCEDURE SetGadgetAttrs( gadget    IN A0: GadgetPtr;
                            window    IN A1: WindowPtr;
                            requester IN A2: RequesterPtr;
                            tagList   IN A3: LIST OF GadgetTags):LONGCARD;
LIBRARY IntuitionBase BY -660
  PROCEDURE SetGadgetAttrsA( gadget    IN A0: GadgetPtr;
                             window    IN A1: WindowPtr;
                             requester IN A2: RequesterPtr;
                             tagList   IN A3: GadgetTagListPtr): LONGCARD;
LIBRARY IntuitionBase BY -666
  PROCEDURE NextObject( objectPtrPtr IN A0: ANYPTR ): ANYPTR;

|*****
| private
|
|LIBRARY IntuitionBase BY -672 PROCEDURE FindClass( classID IN A0: #);
|******

LIBRARY IntuitionBase BY -678
  PROCEDURE MakeClass( classID       IN A0: SysStringPtr;
                       superClassID  IN A1: SysStringPtr;
                       superClassPtr IN A2: IClassPtr;
                       instanceSize  IN D0: LONGCARD;
                       flags         IN D1: IClassFlagSet ): IClassPtr;

LIBRARY IntuitionBase BY -684
  PROCEDURE AddClass( class IN A0: IClassPtr );

LIBRARY IntuitionBase BY -690
  PROCEDURE GetScreenDrawInfo( screen IN A0: ScreenPtr ): DrawInfoPtr;

LIBRARY IntuitionBase BY -696
  PROCEDURE FreeScreenDrawInfo( screen   IN A0: ScreenPtr;
                                drawInfo IN A1: DrawInfoPtr );

LIBRARY IntuitionBase BY -702
  PROCEDURE ResetMenuStrip( window IN A0: WindowPtr;
                            menu   IN A1: MenuPtr ): LONGBOOL;

LIBRARY IntuitionBase BY -708
  PROCEDURE RemoveClass( classPtr IN A0: IClassPtr );

LIBRARY IntuitionBase BY -714
  PROCEDURE FreeClass( classPtr IN A0: IClassPtr ): LONGBOOL;


(* private *)

LIBRARY IntuitionBase BY -720
  PROCEDURE LockPubClass();

LIBRARY IntuitionBase BY -726
  PROCEDURE UnlockPubClass();
```

```
GROUP
  WindowGrp  =
       (* I *)    IDCMPGrp,
       (* T *)
                  ExtNewWindow,       ExtNewWindowPtr,    NewWindow,
                  NewWindowPtr,       RastPort ,          RastPortPtr,
                  Window,             WindowFlags,        WindowFlagSet,
                  WindowPtr,
       (* C *)    otherRefresh ,      refreshBits,
       (* P *)    ActivateWindow,     CloseWindow,        ClearMenuStrip,
                  ModifyIDCMP,        MoveWindow,         OpenWindow,
                  OpenWindowTagList,  OpenWindowTags,
                  RefreshWindowFrame,
                  ResetMenuStrip,     SetMenuStrip,       SetWindowTitles,
                  SizeWindow,         WindowLimits,       WindowToBack,
                  WindowToFront;

  GadgetProcGrp =
       (* P *)    ActivateGadget,     AddGadget,          AddGList,
                  ModifyProp,         NewModifyProp,      OffGadget,
                  OnGadget,           RefreshGadgets,     RefreshGList,
                  RemoveGadget,       RemoveGList;

  ImageGrp    =   ImagePtr, Image;

  GadgetGrp   =
          (* I *) GadgetProcGrp,
                  GadgetPtr,          PropInfoPtr,       StringInfoPtr,
                  GadgetFlags,        GadgetFlagSet,     ActivationFlags,
                  ActivationFlagSet,  gadgHighbits,      gadgHNone,
                  gadgHComp,          GadgetType,        GadgInfoPtr,
                  GadgInfo,           Gadget,            boolMask,
                  BoolInfo,           PropInfoFlags,     PropInfoFlagSet,
                  knobVmin,           knobHmin,          maxBody,
                  maxPot,             PropInfo,          BITSET,
                  StringInfo,         BorderGrp,         ImageGrp,
                  IntuiTextGrp;

  MenuGrp  =
          (* I *) BITSET,             LONGSET,
          (* T *) Menu,               MenuItem,           MenuItemFlags,
                  MenuItemFlagSet,    MenuItemPtr,        MenuPtr,
          (* C *) checkWidth,         commWidth,          highNone,
                  lowCheckWidth,      lowCommWidth,       menuEnabled,
                  miDrawn,
          (* P *) ClearMenuStrip,     ItemAddress,        OffMenu,
                  OnMenu;

  ReqGrp         =
          (* P *) AutoRequest,        BuildSysRequest,    ClearDMRequest,
                  DisplayAlert,       EndRequest,         FreeSysRequest,
                  InitRequester,      Request,            SetDMRequest,
                  Requester,          RequesterFlags,
```



```
                  RequesterFlagSet,   RequesterPtr,
          (* C *) deadendAlert,       recoveryAlert;

  ScreenGrp      =
          (* I *) ViewModeSet,        ViewModes,
          (* T *) BitMap,             BitMapPtr,          ExtNewScreen,
                  ExtNewScreenPtr,    NewScreen,          NewScreenPtr,
                  RastPortPtr,        Screen,             ScreenFlags,
                  ScreenFlagSet,      ScreenPtr,          ViewPortPtr,
          (* C *) stdScreenHeight,    customScreen,
          (* P *) CloseScreen,        CloseWorkbench,     CloseWorkBench,
                  DisplayBeep,        GetScreenData,      LockPubScreen,
                  LockPubScreenList,
                  MakeScreen,         NextPubScreen,
                  MoveScreen,         OpenScreen,
                  OpenScreenTagList,
                  OpenScreenTags,     OpenWorkbench,
                  OpenWorkBench,      ScreenToBack,       ScreenToFront,
                  ShowTitle,          UnlockPubScreen,
                  UnlockPubScreenList,
                  WBenchToBack,
                  WBenchToFront;

  GfxGrp         =
          (* I *) IntuiTextGrp,       BorderGrp,          ImageGrp,
          (* P *) ClearPointer,       DrawBorder,         DrawImage,
                  IntuiTextLength,    PrintIText,         SetPointer;

  MemGrp         =
          (* I *) Exec.MemReqSet,     Exec.MemReqs,
          (* T *) Remember,           RememberPtr,
          (* P *) AllocRemember,      FreeRemember;

  RefreshGrp     =
          (* P *) BeginRefresh,       EndRefresh,         RemakeDisplay,
                  RethinkDisplay;

  TimeGrp        = CurrentTime,       DoubleClick;

  PrefGrp     = GetDefPrefs,        GetPrefs,           SetPrefs,
                filenameSize,       pointerSize,        topazEighty,
                topazSixty,         PrinterPort,        CustomName,
                AlphaP101,          Brother15XL,        CbmMps1000,
                Diab630,            DiabAdvD25,         DiabC150,
                Epson,              EpsonJX80,          Okimate20,
                QumeLP20,           HpLaserjet,         HpLaserjetPlus,
                SerParShk,          SerParShkSet,       Preferences,
                baud110,            baud300,            baud1200,
                baud2400,           baud4800,           baud9600,
                baud19200,          baudMidi,           pica,
                elite,              fine,               draft,
                letter,             sixLPI,             eightLPI,
                imagePositive,      imageNegative,      aspectHoriz,
                aspectVert,         shadeBW,            shadeGreyscale,
                shadeColor,         usLetter,           usLegal,
```

```
               nTractor,           wTractor,           custom,
               PaperType,          readBits,
               writeBits,          stopBits,           bufSizeBits,
               buf512,             buf1024,            buf2048,
               buf4096,            buf8000,            buf16000;

  ViewGrp     = ViewAddress, ViewPortAddress;

  LockGrp     = LockIBase, UnlockIBase;

  IntuiIOGrp  = ReportMouse,  IDCMPGrp, Exec.MsgGrp,  Input.EventGrp;

  RecordGrp   =   BoolInfo,            Border,              BorderList,
                  ColorSpec,           Coordinates,         DimensionRec,
                  DrawInfo,            EasyStruct,          ExtNewScreen,
                  ExtNewWindow,        FatIntuiMessage,     Gadget,
                  GadgetInfo,          GadgetInfo,          GadgInfo,
                  GoInactive,          GPInput,
                  HitTest,             IBox,                IClass,
                  Image,               ImpDraw,             ImpErase,
                  ImpFrameBox,         ImpHitTest,          IntuiMessage,
                  IntuiText,           IntuitionBaseType,   Menu,
                  MenuItem,            MsgRoot,             NewScreen,
                  NewWindow,           Object,              OpAddTail,
                  OpGet,               OpMember,            OpSet,
                  OpUpdate,            PenPair,             PGX,
                  Point,               Preferences,         PropInfo,
                  PubScreenNode,       Remember,            Render,
                  Requester,           Screen,              StringInfo,
                  Window;

  All           = IntuiTextGrp,        BorderGrp,           ImageGrp,
                  GadgetGrp,           IDCMPGrp,            WindowGrp,
                  GadgetProcGrp,       MemGrp,              GfxGrp,
                  ReqGrp,              MenuGrp,             ReqGrp,
                  GadgetProcGrp,       ScreenGrp,
                  RefreshGrp,          TimeGrp,             PrefGrp,
                  ViewGrp,             LockGrp,             IntuiIOGrp,
                  DisplayMode,         dMountCode,          eventMax,
                  Res,                 resCount,            Gadgets,
                  gadgetCount,         ILocks,              numILocks,
                  FatIntuiMessage,     IBox,                Point,
                  PenPair,
                  GadgetInfo,          numIEvents,          IntuitionBase,
                  IntuitionBasePtr,    IntuitionBaseType;

END Intuition.
```


8.26 Keyboard

```
DEFINITION MODULE Keyboard;
(* $A- *)
FROM T_Exec     IMPORT IOCommand, nonstdVAL, IOStdReqPtr;
FROM Resources  IMPORT ContextPtr;

CONST
  readEvent        = IOCommand( nonstdVAL + 0 );
  readMatrix       = IOCommand( nonstdVAL + 1 );
  addResetHandler  = IOCommand( nonstdVAL + 2 );
  remResetHandler  = IOCommand( nonstdVAL + 3 );
  resetHandlerDone = IOCommand( nonstdVAL + 4 );

PROCEDURE OpenKeyboard(context : ContextPtr:=NIL):IOStdReqPtr;

PROCEDURE CloseKeyboard(VAR request : IOStdReqPtr);

GROUP
  All  = readEvent,readMatrix,addResetHandler,remResetHandler,
         resetHandlerDone,OpenKeyboard,CloseKeyboard,T_Exec.ExecIOGrp;

END Keyboard.
```

8.27 KeyMap

```
DEFINITION MODULE KeyMap;
(* $A- *)
FROM Exec    IMPORT Node,List,LibraryPtr;
FROM System  IMPORT LONGSET,SysStringPtr,Regs;
FROM Input   IMPORT Qualifiers,InputEventPtr;

TYPE
  KeyMapTypes        = (shift,alt,control,downup,kmp4,dead,string,nop);
  KeyMapTypeSet      = SET OF KeyMapTypes;
  DeadPrefixBytes    = (dpbMod,dpb1,dpb2,dpbDead);
  DeadPrefixByteSet  = SET OF DeadPrefixBytes;

CONST
  dp2dIndexMax  = $0F;
  dp2dFacShift  = 4;
  maxKeys       = 64;
  noQual        = KeyMapTypeSet:{};
  vanilla       = KeyMapTypeSet:{shift,alt,control}

TYPE
  BitTable         = ARRAY [maxKeys DIV (8*LONGSET'SIZE)] OF LONGSET;
  BitTablePtr      = POINTER TO BitTable;

  KeyInfo          = RECORD
                       IF KEY : INTEGER
                         OF 0 THEN ch : ARRAY [0..3] OF CHAR;
                         OF 1 THEN st : SysStringPtr;
                        END
                     END;

  Types            = ARRAY [0..maxKeys-1] OF KeyMapTypeSet;
  TypesPtr         = POINTER TO Types;

  Info             = ARRAY [0..maxKeys-1] OF KeyInfo;
  InfoPtr          = POINTER TO Info;

  KeyMap           = RECORD
                       loKeyMapTypes   : TypesPtr;
                       loKeyMap        : InfoPtr;
                       loCapsable      : BitTablePtr;
                       loRepeatable    : BitTablePtr;
                       hiKeyMapTypes   : TypesPtr;
                       hiKeyMap        : InfoPtr;
                       hiCapsable      : BitTablePtr;
                       hiRepeatable    : BitTablePtr;
                     END;
  KeyMapPtr        = POINTER TO KeyMap;

  KeyMapNode       = RECORD OF Node
                       keyMap : KeyMap
                     END;
```



```
  KeyMapResource    = RECORD OF Node
                        keyMaps : List
                      END;
  CodeQualPair      = RECORD
                        code : SHORTCARD;
                        qual : SET OF [lShift..rCommand];
                      END;
  CodeQualArray     = ARRAY OF CodeQualPair;

VAR
  KeymapBase  : LibraryPtr;

LIBRARY KeymapBase BY -30
  PROCEDURE SetKeyMapDefault(keyMap IN A0 : KeyMapPtr);

LIBRARY KeymapBase BY -36
  PROCEDURE AskKeyMapDefault():KeyMapPtr;

LIBRARY KeymapBase BY -42
  PROCEDURE MapRawKey(event  IN A0 : InputEventPtr;
                      buffer IN A1 : ANYPTR;
                      length IN D1 : INTEGER;
                      keyMap IN A2 : KeyMapPtr):INTEGER;

LIBRARY KeymapBase BY -48
  PROCEDURE MapANSI(REF string IN A0 : STRING;
                        count  IN D0 : LONGINT;
                    VAR buffer IN A1 : CodeQualArray;
                        length IN D1 : LONGINT;
                        keyMap IN A2 : KeyMapPtr):LONGINT;

GROUP
  All    = KeyMapTypes,KeyMapTypeSet,DeadPrefixBytes,DeadPrefixByteSet,
           dp2dIndexMax,dp2dFacShift,maxKeys,noQual,vanilla,BitTable,
           BitTablePtr,KeyInfo,Types,TypesPtr,Info,InfoPtr,KeyMap,
           KeyMapPtr,KeyMapNode,KeyMapResource;

END KeyMap.
```

8.28 Layers

```
DEFINITION MODULE Layers;


FROM Graphics   IMPORT ClipRectPtr,BitMapPtr,LayerFlagSet,LayerInfoPtr,
                       LayerPtr,RastPortPtr,RegionPtr,Rectangle;
FROM Utility    IMPORT HookPtr;
FROM Exec       IMPORT LibraryPtr;
FROM System     IMPORT Regs,LONGSET;

TYPE
  LayerMessagePtr = POINTER TO LayerMessage;
  LayerMessage    = RECORD
                      layer    : LayerPtr;
                      bounds   : Rectangle;
                      xoffset,
                      yoffset  : INTEGER;
                    END;


VAR LayersBase : LibraryPtr;

LIBRARY LayersBase BY  -78
  PROCEDURE BeginUpdate(l IN A0 : LayerPtr);

LIBRARY LayersBase BY  -54
  PROCEDURE BehindLayer(l IN A1 : LayerPtr);

LIBRARY LayersBase BY  -42
  PROCEDURE CreateBehindLayer(li    IN A0  : LayerInfoPtr;
                              bm    IN A1  : BitMapPtr;
                              x0    IN D0,
                              y0    IN D1,
                              x1    IN D2,
                              y1    IN D3  : LONGINT;
                              flags IN D4  : LayerFlagSet;
                              bm2   IN A2 : BitMapPtr):LayerPtr;

LIBRARY LayersBase BY -186
  PROCEDURE CreateUpfrontHookLayer(li      IN A0 : LayerInfoPtr;
                                   bm      IN A1 : BitMapPtr;
                                   x0      IN D0 : LONGINT;
                                   y0      IN D1 : LONGINT;
                                   x1      IN D2 : LONGINT;
                                   y1      IN D3 : LONGINT;
                                   flags   IN D4 : LayerFlagSet;
                                   hook    IN A3 : HookPtr;
                                   superbm IN A2 : BitMapPtr):LayerPtr;
```



```
LIBRARY LayersBase BY  -36
  PROCEDURE CreateUpfrontLayer(li    IN A0  : LayerInfoPtr;
                               bm    IN A1  : BitMapPtr;
                               x0    IN D0,
                               y0    IN D1,
                               x1    IN D2,
                               y1    IN D3  : LONGINT;
                               flags IN D4  : LONGSET;
                               bm2   IN A2  : BitMapPtr):LayerPtr;

LIBRARY LayersBase BY -192
  PROCEDURE CreateBehindHookLayer(li      IN A0 : LayerInfoPtr;
                                  bm      IN A1 : BitMapPtr;
                                  x0      IN D0 : LONGINT;
                                  y0      IN D1 : LONGINT;
                                  x1      IN D2 : LONGINT;
                                  y1      IN D3 : LONGINT;
                                  flags   IN D4 : LayerFlagSet;
                                  hook    IN A3 : HookPtr;
                                  superbm IN A2 : BitMapPtr):LayerPtr;

LIBRARY LayersBase BY  -90
  PROCEDURE DeleteLayer(l IN A1 : LayerPtr);

LIBRARY LayersBase BY -150
  PROCEDURE DisposeLayerInfo(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY  -84
  PROCEDURE EndUpdate(l    IN A0 : LayerPtr;
                      flag IN D0 : BOOLEAN);

LIBRARY LayersBase BY -156
  PROCEDURE FattenLayerInfo(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY  -30
  PROCEDURE InitLayers(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY -174
  PROCEDURE InstallClipRegion(l      IN A0 : LayerPtr;
                              region IN A1 : RegionPtr):RegionPtr;

LIBRARY LayersBase BY  -96
  PROCEDURE LockLayer(l IN A1 : LayerPtr);

LIBRARY LayersBase BY -120
  PROCEDURE LockLayerInfo(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY -108
  PROCEDURE LockLayers(li IN A0 : LayerInfoPtr),
```

```
LIBRARY LayersBase BY  -60
  PROCEDURE MoveLayer(l  IN A1 : LayerPtr;
                      dx IN D0,
                      dy IN D1 : LONGINT);

LIBRARY LayersBase BY -168
  PROCEDURE MoveLayerInFrontOf(l      IN A0,
                               target IN A1 : LayerPtr);

LIBRARY LayersBase BY -144
  PROCEDURE NewLayerInfo():LayerInfoPtr;

LIBRARY LayersBase BY  -72
  PROCEDURE ScrollLayer(l  IN A1 : LayerPtr;
                        dx IN D0,
                        dy IN D1 : LONGINT);

LIBRARY LayersBase BY  -66
  PROCEDURE SizeLayer(l  IN A1 : LayerPtr;
                      dx IN D0,
                      dy IN D1 : LONGINT);

LIBRARY LayersBase BY -180
  PROCEDURE MoveSizeLayer(layer IN A0 : LayerPtr;
                          dx    IN D0 : LONGINT;
                          dy    IN D1 : LONGINT;
                          dw    IN D2 : LONGINT;
                          dh    IN D3 : LONGINT):BOOLEAN;

LIBRARY LayersBase BY -126
  PROCEDURE SwapBitsRastPortClipRect(rp IN A0 : RastPortPtr;
                                     cr IN A1 : ClipRectPtr);

LIBRARY LayersBase BY -162
  PROCEDURE ThinLayerInfo(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY -102
  PROCEDURE UnlockLayer(l IN A0 : LayerPtr);

LIBRARY LayersBase BY -138
  PROCEDURE UnlockLayerInfo(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY -114
  PROCEDURE UnlockLayers(li IN A0 : LayerInfoPtr);

LIBRARY LayersBase BY  -48
  PROCEDURE UpfrontLayer(l IN A1 : LayerPtr);

LIBRARY LayersBase BY -132
  PROCEDURE WhichLayer(li IN A0 : LayerInfoPtr;
                       x  IN D0,
                       y  IN D1 : LONGINT):LayerPtr;
```



```
LIBRARY LayersBase BY -198
  PROCEDURE InstallLayerHook(layer IN A0 : LayerPtr;
                             hook  IN A1 : HookPtr):HookPtr;

END Layers.
```

8.29 MathIEEEResource

```
DEFINITION MODULE MathIEEEResource;
(* $A- *)
FROM System IMPORT PROC;
FROM Exec   IMPORT Node;

TYPE
  MathIEEEResourceFlags    = (dblbas,dbltrans,sglbas,sgltrans,extbas,
                              exttrans,makeMeWord = 15);
  MathIEEEResourceFlagSet  = SET OF MathIEEEResourceFlags;

  MathIEEEResourcePtr      = POINTER TO MathIEEEResource;
  MathIEEEResource         = RECORD OF Node;
                               flags        : MathIEEEResourceFlagSet;
                               baseAddr     : ANYPTR;
                               dblBasInit,
                               dblTransInit,
                               sglBasInit,
                               sglTransInit,
                               extBasInit,
                               extTransInit : PROC
                             END;

VAR
  MathBase  : MathIEEEResourcePtr;

GROUP
  All  = MathIEEEResourceFlags,MathIEEEResourceFlagSet,MathIEEEResource,
         MathIEEEResourcePtr,MathBase;

END MathIEEEResource.
```


8.30 MiscResource

```
DEFINITION MODULE MiscResource;
(* $A- *)
FROM Exec    IMPORT Resource;
FROM System  IMPORT Regs;

TYPE
  ResourceTypes      = (serialPort,serialBits,parallelPort,
                        parallelBits);
  MiscResource       = RECORD OF Resource;
                         allocArray : ARRAY ResourceTypes OF ANYPTR
                       END;
  MiscResourcePtr    = POINTER TO MiscResource;

VAR
  MiscBase  : MiscResourcePtr;

LIBRARY MiscBase BY -6
  PROCEDURE AllocMiscResource(    unitNum IN D0  : LONGINT;
                              REF name    IN A1  : STRING):ANYPTR;

LIBRARY MiscBase BY -12
  PROCEDURE FreeMiscResource(unitNum IN D0  : LONGINT);

GROUP
  All   = ResourceTypes,MiscResource,MiscResourcePtr,MiscBase,
          AllocMiscResource,FreeMiscResource;

END MiscResource.
```

8.31 Narrator

```
DEFINITION MODULE Narrator;
(* $A- *)
FROM T_Exec     IMPORT IOStdReq,IOReturn;
FROM Resources  IMPORT ContextPtr;
FROM Utility    IMPORT StdTags;
FROM System     IMPORT SysStringPtr;

CONST
  noMem       = IOReturn(256-2);
  noAudLib    = IOReturn(256-3);
  makeBad     = IOReturn(256-4);
  unitErr     = IOReturn(256-5);
  cantAlloc   = IOReturn(256-6);
  unimpl      = IOReturn(256-7);
  noWrite     = IOReturn(256-8);
  expunged    = IOReturn(256-9);
  phonErr     = IOReturn(256-20);
  rateErr     = IOReturn(256-21);
  pitchErr    = IOReturn(256-22);
  sexErr      = IOReturn(256-23);
  modeErr     = IOReturn(256-24);
  freqErr     = IOReturn(256-25);
  volErr      = IOReturn(256-26);
  dCentErr    = IOReturn(256-27);
  centPhonErr = IOReturn(256-28);

TYPE
  Sex              = (male,female,dummy=$1000);
  PitchMode        = (natural,robotic,manual,dummy=$1000);
  SpeakRate        = [40..400];
  Pitch            = [65..320];
  SampleFreq       = [5000..28000];
  Volume           = [0..64];
  Central          = [0..100];

  NarratorFlags    = (newIO,wordSync,sylableSync);
  NarratorFlagSet  = SET OF NarratorFlags;
```



```
IONarratorPtr     = POINTER TO IONarrator;
IONarrator        = RECORD OF IOStdReq
                      rate         : SpeakRate;
                      pitch        : Pitch;
                      mode         : PitchMode;
                      sex          : Sex;
                      chMask       : ANYPTR;
                      nmMask       : CARDINAL;
                      pad1         : SHORTCARD;
                      volume       : Volume;
                      sampFreq     : SampleFreq;
                      mouths       : BOOLEAN;
                      chanMask     : SHORTCARD;
                      numChan      : SHORTCARD;
                      flags        : NarratorFlagSet;
                      enthusiasm   : SHORTCARD;
                      perturbation : SHORTCARD;
                      f1adj,
                      f2adj,
                      f3adj        : SHORTINT;
                      a1adj,
                      a2adj,
                      a3adj        : SHORTINT;
                      articulate   : SHORTCARD;
                      centralize   : SHORTCARD;
                      centPhon     : SysStringPtr;
                      aVBias       : SHORTINT;
                      aFBias       : SHORTINT;
                      priority     : SHORTINT;
                      pad2         : SHORTINT;

                      width        : SHORTCARD;
                      heigth       : SHORTCARD;
                      shape        : SHORTCARD;
                      sync         : NarratorFlagSet;
                    END;
```

```
  NarratorTags      = TAGS OF StdTags;
                        rate         : SpeakRate;
                        pitch        : Pitch;
                        mode         : PitchMode;
                        sex          : Sex;
                        volume       : Volume;
                        sampFreq     : SampleFreq;
                        mouths       : BOOLEAN;
                        enthusiasm   : SHORTCARD;
                        perturbation : SHORTCARD;
                        f1adj        : SHORTINT;
                        f2adj        : SHORTINT;
                        f3adj        : SHORTINT;
                        a1adj        : SHORTINT;
                        a2adj        : SHORTINT;
                        a3adj        : SHORTINT;
                        articulate   : SHORTCARD;
                        centralize   : SHORTCARD;
                        centPhon     : SysStringPtr;
                        aVBias       : SHORTINT;
                        aFBias       : SHORTINT;
                        priority     : SHORTINT;
                      END;

PROCEDURE OpenNarrator(context  : ContextPtr:=NIL;
                       tags     : LIST OF NarratorTags):IONarratorPtr;

PROCEDURE CloseNarrator(VAR  request : IONarratorPtr);

GROUP
  ErrorGrp  = noMem,noAudLib,makeBad,unitErr,cantAlloc,
              unimpl,noWrite,expunged,phonErr,rateErr,
              pitchErr,sexErr,modeErr,freqErr,volErr,
              dCentErr,centPhonErr;

  DeviceGrp = NarratorFlags,NarratorFlagSet,IONarratorPtr,
              IONarrator,NarratorTags,OpenNarrator,CloseNarrator;

  All       = ErrorGrp,DeviceGrp;

END Narrator.
```


8.32 Parallel

```
DEFINITION MODULE Parallel;
(* $A- *)

|
| WB  4 Jun 1992 IOCommand
|

FROM T_Exec     IMPORT IOCommand, nonstdVAL, IOStdReq;
FROM Resources  IMPORT ContextPtr;

CONST
  query      = IOCommand( nonstdVAL + 0 );
  setParams  = IOCommand( nonstdVAL + 1 );

TYPE
  IOPArray      = ARRAY [0..7] OF CHAR;

  ParErr        = (pe0,devBusy,bufTooBig,invParam,lineErr,notOpen,
                   portReset,initErr);

  ParFlags      = (pf0,eofMode,ackMode,radBoogie,fastMode=3,slowMode,
                   shared);
  ParFlagSet    = SET OF ParFlags;

  Status        = (parBusy,paperOut,parSel,rwDir,active,abort,queued);
  StatusSet     = SET OF Status;

  IOParallel    = RECORD OF IOStdReq
                    pExtFlags    : LONGCARD;
                    status       : StatusSet;
                    parFlags     : ParFlagSet;
                    pTermArray   : IOPArray
                  END;
  IOParallelPtr = POINTER TO IOParallel

PROCEDURE OpenParallel(context : ContextPtr:=NIL):IOParallelPtr;

PROCEDURE CloseParallel(VAR request : IOParallelPtr)

GROUP
  All   =  query,setParams,IOPArray,ParErr,ParFlags,ParFlagSet,Status,
           StatusSet,IOParallel,IOParallelPtr,OpenParallel,CloseParallel;

END Parallel.
```

8.33 PotgoResource

```
DEFINITION MODULE PotgoResource;
(* $A- *)
FROM Exec     IMPORT LibraryPtr;
FROM Hardware IMPORT PotFlags,PotFlagSet;
FROM System   IMPORT Regs;

VAR
  PotgoBase : LibraryPtr;

LIBRARY  PotgoBase  BY -6
  PROCEDURE AllocPotBits(bits     IN D0 : PotFlagSet):PotFlagSet;

LIBRARY  PotgoBase  BY -12
  PROCEDURE FreePotBits(allocated IN D0 : PotFlagSet);

LIBRARY  PotgoBase  BY -18
  PROCEDURE WritePotgo(word       IN D0  : PotFlagSet;
                       mask       IN D1  : PotFlagSet);

GROUP
  All  = PotgoBase,AllocPotBits,FreePotBits,WritePotgo,PotFlags,PotFlagSet;

END PotgoResource.
```


8.34 Printer

```
DEFINITION MODULE Printer;
(* $A- *)

|
| WB  4 Jun 1992 IOCommand
|

FROM T_Exec    IMPORT IOCommand, nonstdVAL, DevicePtr, IOFlagSet, Message,
                      UnitPtr, IORequest;
FROM Graphics  IMPORT ColorMapPtr,RastPortPtr,ViewModeSet;
FROM Resources IMPORT ContextPtr;

CONST
  rawWrite        = IOCommand( nonstdVAL + 0 );
  prtCommand      = IOCommand( nonstdVAL + 1 );
  dumpRPort       = IOCommand( nonstdVAL + 2 );
  query           = IOCommand( nonstdVAL + 3 );

TYPE
  PrtCommands = (ris,rin,ind,nel,ri,
                 sgr0,sgr3,sgr23,sgr4,sgr24,sgr1,sgr22,sfc,sbc,
                 shorp0,shorp2,shorp1,shorp4,shorp3,shorp6,shorp5,
                 den6,den5,den4,den3,den2,den1,
                 sus2,sus1,sus4,sus3,sus0,plu,pld,
                 fnt0,fnt1,fnt2,fnt3,fnt4,fnt5,fnt6,fnt7,fnt8,fnt9,fnt10,
                 prop2,prop1,prop0,tss,jfy5,jfy7,jfy6,jfy0,jfy3,jfy1,
                 verp0,verp1,slpp,perf,perf0,
                 lms,rms,tms,bms,stbm,slrm,cam,
                 hts,vts,tbc0,tbc3,tbc1,tbc4,tbcall,tbsall,extend,raw,
                 makeMeWord = $1000);

TYPE
  Error          = (nonErr,cancel,notGraphics,invertHam,badDimension,
                    dimensionOvflow,internalMemory,buffMemory,tookControl);

  Special        = (milCols,milRows,fullCols,fullRows,fracCols,fracRows,
                    center,aspect,densBit0,densBit1,densBit2,noFormFeeds,
                    trustMe,noPrint);

  SpecialSet     = SET OF Special;


CONST
  density1  = SpecialSet:{densBit0};
  density2  = SpecialSet:{densBit1};
  density3  = SpecialSet:{densBit0,densBit1};
  density4  = SpecialSet:{densBit2};
  density5  = SpecialSet:{densBit0,densBit2};
  density6  = SpecialSet:{densBit1,densBit2};
  density7  = SpecialSet:{densBit0,densBit1,densBit2};
```

```
TYPE
  IOPrinter    = RECORD OF IORequest
                   IF KEY : INTEGER
                     OF 0 THEN
                       actual,
                       length  : LONGINT;
                       data    : ANYPTR;
                       offset  : LONGINT;
                     OF 1 THEN
                       prtCommands : PrtCommands;
                       parm0       : SHORTCARD;
                       parm1       : SHORTCARD;
                       parm2       : SHORTCARD;
                       parm3       : SHORTCARD;
                     OF 2 THEN
                       rastPort   : RastPortPtr;
                       colorMap   : ColorMapPtr;
                       modesHi    : CARDINAL;
                       modes      : ViewModeSet;
                       srcX       : CARDINAL;
                       srcY       : CARDINAL;
                       srcWidth   : CARDINAL;
                       srcHeight  : CARDINAL;
                       destCols   : LONGINT;
                       destRows   : LONGINT;
                       special    : SpecialSet
                   END
                 END;
  IOPrinterPtr = POINTER TO IOPrinter


PROCEDURE OpenPrinter(context : ContextPtr:=NIL):IOPrinterPtr;

PROCEDURE ClosePrinter(VAR request : IOPrinterPtr)


GROUP
  All  = rawWrite,query,prtCommand,dumpRPort,PrtCommands,Error,Special,
         SpecialSet,density1,density2,density3,density4,density5,
         density6,density7,IOPrinter,IOPrinterPtr,OpenPrinter,ClosePrinter,
         T_Exec.ExecIOGrp;

END Printer.
```


8.35 PrtBase

```
DEFINITION MODULE PrtBase;
(* $A- *)
FROM Exec      IMPORT Library,MsgPort,Task,Device,ExecBasePtr;
FROM Intuition IMPORT Preferences;
FROM Parallel  IMPORT IOParallel;
FROM Serial    IMPORT IOSerial;
FROM Timer     IMPORT IOTimer;
FROM System    IMPORT SysStringPtr,PROC,BPTR;

TYPE
  DeviceData    = RECORD OF Library;
                    segment     : ANYPTR;
                    execBase    : ExecBasePtr;
                    cmdVectors  : ANYPTR;
                    cmdBytes    : ANYPTR;
                    numCommands : CARDINAL;
                  END;
  DeviceDataPtr = POINTER TO DeviceData;

CONST
  oldStkSize = $800;
  stkSize    = $1000;
  bufSize    = 256;
  safeSize   = 128;


TYPE
  NormTask             = RECORD OF Task END;
  PrinterSegmentPtr    = POINTER TO PrinterSegment;

  PrinterData          = RECORD OF DeviceData;
                             unit             : MsgPort;
                             printerSegment   : BPTR;
                             printerType      : CARDINAL;
                             segmentData      : PrinterSegmentPtr;
                             printBuf         : ANYPTR;
                             pWrite           : PROCEDURE():INTEGER;
                             pBothReady       : PROCEDURE():INTEGER;
                             IF KEY : INTEGER
                               OF 1 THEN p0   : IOParallel;
                                         p1   : IOParallel
                               OF 2 THEN s0   : IOSerial;
                                         s1   : IOSerial
                             END;
                             tior             : IOTimer;
                             ioRPort          : MsgPort;
                             tc               : NormTask;
                             oldStk           : ARRAY [oldStkSize] OF SHORTCARD;
                             flags            : SHORTCARD;
                             pad              : SHORTCARD;
                             preferences      : Preferences;
                             pWaitEnabled     : SHORTCARD;
```

```
                         flags1          : SHORTCARD;
                         stk             : ARRAY [stkSize] OF SHORTCARD;
                       END;
  PrinterDataPtr       = POINTER TO PrinterData;
  PrinterClass         = (gfx,color);
  PrinterClassSet      = SET OF PrinterClass;

CONST
  bwAlpha    = PrinterClassSet:{};
  bwGfx      = PrinterClassSet:{gfx};
  colorAlpha = PrinterClassSet:{color};
  colorGfx   = PrinterClassSet:{gfx,color};

TYPE
  ColorClass    = (blackAndWhite,colors,fourColor,additive,multipass);
  ColorClassSet = SET OF ColorClass;

CONST
  bw         = ColorClassSet:{blackAndWhite};
  ymc        = ColorClassSet:{colors};
  ymcBw      = ColorClassSet:{blackAndWhite,colors};
  ymbc       = ColorClassSet:{fourColor};
  wb         = ColorClassSet:{blackAndWhite,additive};
  bgr        = ColorClassSet:{colors,additive};
  bgrWb      = ColorClassSet:{blackAndWhite,colors,additive};
  bgrw       = ColorClassSet:{fourColor,additive};

TYPE
  DoSpecial              = PROCEDURE(Command      : CARDINAL;
                                     OutPutBuffer : SysStringPtr;
                                     Line,
                                     LineSpace,
                                     CRLF,
                                     Params       : SHORTINT):INTEGER;

  Render                 = PROCEDURE(ct     : SHORTCARD;
                                     x,
                                     y      : CARDINAL;
                                     status : SHORTCARD):INTEGER;

  ConvFunc               = PROCEDURE():LONGINT;
```



```
  PrinterExtendedData    = RECORD
                             printerName  : SysStringPtr;
                             init         : PROC;
                             expunge      : PROC;
                             open         : PROCEDURE():INTEGER;
                             close        : PROC;
                             printerClass : PrinterClassSet;
                             colorClass   : ColorClassSet;
                             maxColumns   : SHORTCARD;
                             numCharSets  : SHORTCARD;
                             numRows      : CARDINAL;
                             maxXDots     : LONGCARD;
                             maxYDots     : LONGCARD;
                             xDotsInch    : CARDINAL;
                             yDotsInch    : CARDINAL;
                             commands     : ANYPTR;
                             doSpecial    : DoSpecial;
                             render       : Render;
                             timeOutSecs  : LONGINT;
                             eightBitChars: ANYPTR;
                             printMode    : LONGINT;
                             convFunc     : ConvFunc;
                           END;
  PrinterExtendetDataPtr = POINTER TO PrinterExtendedData;

  PrinterSegment         = RECORD
                             nextSegment  : BPTR;
                             runAlert     : LONGCARD;
                             version      : CARDINAL;
                             revision     : CARDINAL;
                             ped          : PrinterExtendedData
                           END

GROUP
  All   = DeviceData,DeviceDataPtr,bufSize,safeSize,stkSize,oldStkSize,
          NormTask,PrinterSegmentPtr,PrinterData,PrinterDataPtr,
          PrinterClass,PrinterClassSet,bwAlpha,bwGfx,colorAlpha,
          colorGfx,ColorClass,ColorClassSet,bw,ymc,ymcBw,ymbc,wb,bgr,
          bgrWb,bgrw,DoSpecial,Render,ConvFunc,PrinterExtendedData,
          PrinterExtendetDataPtr,PrinterSegment;


END PrtBase.
```

8.36 Rexx

```
DEFINITION MODULE Rexx;
(* $A- *)
FROM System    IMPORT BITSET,BPTR,Regs;
FROM Dos       IMPORT DeviceListPtr,DosLibraryPtr,FileHandlePtr,
                      FileLockPtr,StandardPacketPtr;
FROM Exec      IMPORT ExecBasePtr,Library,List,ListPtr,MemReqs,Message,
                      MessagePtr,MsgPort,MsgPortPtr,Node,NodePtr;

TYPE
  RexxErrors            = (ok,
                           programNotFound,
                           executionNotHalted,
                           noMemoryAvailable,
                           invalidCharacterInProgram,
                           unmatchedQuote,
                           unterminatedComment,
                           clauseTooLong,
                           unrecognizedToken,
                           symbolOrStringTooLong,

                           invalidMessagePacket,
                           commandStringError,
                           errorReturnFromFunction,
                           hostEnvironmentNotFound,
                           requiredLibraryNotFound,
                           functionNotFound,
                           noReturnValue,
                           wrongNumbersOfArguments,
                           invalidArgumentToFunction,
                           invalidProcedure,

                           unexpectedThenElse,
                           unexpectedWhenOtherwise,
                           unexpectedLeaveIterate,
                           invalidStatementInSelect,
                           missingThenClauses,
                           missingOtherwise,
                           missingOrUnexpectedEnd,
                           symbolMismatchOnEnd,
                           invalidDoSyntax,
                           incompleteDoIfSelec,

                           labelNotFound,
                           symbolExpected,
                           stringOrSymbolExpected,
                           invalidSubKeyword,
                           requiredKeywordMissing,
                           extraneousCharacters,
                           subKeywordConflict,
                           invalidTemplate,
                           invalidTraceRequest,
                           uninitializedVariable,
```



```
                           invalidVariableName,
                           invalidExpression,
                           unbalancedParentheses,
                           nestingLevelExceeded,
                           invalidExpressionResult,
                           expressionRequired,
                           booleanValueNot0or1,
                           arithmeticConversionError,
                           invalidOperand);

CONST
  ReturnOk              = 0;
  ReturnWarn            = 5;
  ReturnError           = 10;
  ReturnFatal           = 20;

TYPE
  AttributeFlags        = (keep,string,notNum,number,binary,float,ext,
                           source);
  AttributeFlagSet      = SET OF AttributeFlags;
  NexxStr               = RECORD
                            iValue   : LONGINT;
                            length   : CARDINAL;
                            flags    : AttributeFlagSet;
                            hash     : SHORTCARD;
                            (*buff    : ARRAY OF CHAR;*)
                          END;
  NexxStrPtr            = POINTER TO NexxStr;

CONST
  intNum    = AttributeFlagSet:{number,binary,string};
  dpNum     = AttributeFlagSet:{number,float};
  alpha     = AttributeFlagSet:{notNum,string};
  owned     = AttributeFlagSet:{source,ext,keep};
  keepStr   = AttributeFlagSet:{string,source,notNum};
  keepNum   = AttributeFlagSet:{string,source,number,binary};

TYPE
  RexxArg     = RECORD
                  size    : LONGINT;
                  length  : CARDINAL;
                  flags   : AttributeFlagSet;
                  hash    : SHORTCARD;
                  (* buff    : ARRAY OF CHAR *)
                END;
   RexxArgPtr = POINTER TO RexxArg;

CONST
  maxRMArg = 15;

TYPE
  Command          = (co0,comn,func,close,query,co5,co6,addFH,addLib,remLib,
                      addCon,remCon,tcOpn,tcCls);
```

```
  ModifierFlags   = (noIO,result,moString,token,nonRet,mo5,mo6,mo7);
  ModifierFlagSet = SET OF ModifierFlags;

  ActionRec       = RECORD
                      command   : Command;
                      modifier  : ModifierFlagSet;
                      add       : CARDINAL
                    END;

  RexxMsg         = RECORD OF Message
                      taskBlock,
                      libBase     : ANYPTR;
                      action      : ActionRec;
                      result1,
                      result2     : LONGINT;
                      args        : ARRAY [0..maxRMArg] OF ANYPTR;
                      passPort    : MsgPortPtr;
                      commAddr,
                      fileExt     : ANYPTR;
                      stdin,
                      stdout      : FileHandlePtr;
                      avail       : LONGINT
                    END;
  RexxMsgPtr      = POINTER TO RexxMsg;

  RexxRsrc        = RECORD OF Node
                      func  : INTEGER;
                      base  : ANYPTR;
                      size,
                      arg1,
                      arg2  : LONGINT
                    END;
  RexxRsrcPtr     = POINTER TO RexxRsrc;
  RsrcNodeType    = (any,lib,port,file,host,clip);

CONST
  globalsz   = 200;

TYPE
  RexxTaskFlags      = (trace,halt,susp,tcUse,rtf4,rtf5,wait,rtfClose);
  RexxTaskFlagSet    = SET OF RexxTaskFlags;
  RexxTask           = RECORD
                         global    : ARRAY [0..globalsz-1] OF SHORTINT;
                         msgPort   : MsgPort;
                         flags     : RexxTaskFlagSet;
                         sigBit    : SHORTCARD;
                         clientID,
                         msgPkt,
                         taskID,
                         rexxPort,
                         errTrap,
                         stackPtr  : ANYPTR;
                         envList,
                         freeList,
```



```
                         memList,
                         fileList,
                         portList  : List
                       END;
  RexxTaskPtr        = POINTER TO RexxTask;

CONST
  memQuant   = $10;
  memMask    = $FFFFFFF0;
  memQuick   = public;

TYPE
  SrcNodePtr = POINTER TO SrcNode;
  SrcNode    = RECORD
                 succ,
                 pred  : SrcNodePtr;
                 ptr   : ANYPTR;
                 size  : LONGINT
               END;

CONST
  rxsdir   = "REXX";
  rxstname = "ARexx";

TYPE
  RxsLibPtr  = POINTER TO RxsLib;
  RxsLib     = RECORD OF Library
                 flags       : RexxTaskFlagSet;
                 sysBase     : ExecBasePtr;
                 dosBase     : DosLibraryPtr;
                 ieeeDPBase  : ANYPTR;
                 segList     : BPTR;
                 nil         : FileHandlePtr;
                 chunk,
                 maxNest     : LONGINT;
                 null,
                 false,
                 true,
                 rexx,
                 command,
                 stdin,
                 stdout,
                 stderr      : NexxStrPtr;
                 version,
                 taskName    : ANYPTR;
                 taskPri     : LONGINT;
                 taskSeg     : BPTR;
                 stackSize   : LONGINT;
                 rexxDir,
                 cTable      : ANYPTR;
                 notice      : NexxStrPtr;
                 rexxPort    : MsgPort;
                 readLock    : CARDINAL;
                 traceFH     : FileHandlePtr;
                 taskList    : List;
```

```
               numTask    : INTEGER;
               libList    : List;
               numLib     : INTEGER;
               clipList   : List;
               numClip    : INTEGER;
               msgList    : List;
               numMsg     : INTEGER;
               pgmList    : List;
               numPgm     : INTEGER;
               traceCnt   : CARDINAL;
               avail      : INTEGER
             END;


CONST
  tcOpen       = rtf4;
  stop         = wait;
  vers         = 34;
  rev          = 9;
  rxsalloc     = $80000000;
  rxschunk     = $00000400;
  rxsnest      = $00000020;
  rxstpri      = $00000000;
  rxsstack     = $00001000;
  rxslisth     = $00000005;

TYPE
  CharAttrFlags     = (space,digit,caAlpha,rexxSym,rexxOpr,rexxSpc,upper,
                        lower);
  CharAttrFlagSet   = SET OF CharAttrFlags;

CONST
   buffsz  = 204;

TYPE
  IoBuff    = RECORD  OF RexxRsrc;
               rpt   : ANYPTR;
               rct   : LONGINT;
               dfh,
               lock  : FileLockPtr;
               bct   : LONGINT;
               area  : ARRAY [0..buffsz-1] OF SHORTINT
              END;
  IoBuffPtr = POINTER TO IoBuff;

CONST
  exist   = -1;
  strf    =  0;
  read    =  1;
  write   =  2;
  append  =  3;
```



```
TYPE
  RexxMsgPort = RECORD OF RexxRsrc;
                  port      : MsgPort;
                  replyList : List
                END;

CONST
  stack  = 2002;
  queue  = 2003;

  fail   = -1;

VAR
  RexxBase   : RxsLibPtr;

LIBRARY RexxBase BY -126
  PROCEDURE CreateArgstring(String IN A0 : ANYPTR;
                            Length IN D0 : LONGINT):ANYPTR;

LIBRARY RexxBase BY -132
  PROCEDURE DeleteArgstring(ArgString IN A0 : ANYPTR);

LIBRARY RexxBase BY -138
  PROCEDURE LengthArgstring(Arg IN A0 :RexxArgPtr):LONGINT;

LIBRARY RexxBase BY -144
  PROCEDURE CreateRexxMsg(ReplyPort IN A0 : MsgPortPtr;
                          Extension IN A1 : ANYPTR;
                          Host      IN D0 : ANYPTR):RexxMsgPtr;

LIBRARY RexxBase BY -150
  PROCEDURE DeleteRexxMsg(Message IN A0 : RexxMsgPtr);

LIBRARY RexxBase BY -156
  PROCEDURE ClearRexxMsg(MsgPtr  IN A0 : MessagePtr;
                         Count   IN D0 : LONGINT);

LIBRARY RexxBase BY -162
  PROCEDURE FillRexxMsg(MsgPtr   IN A0 : RexxMsgPtr;
                        Count    IN D0 : LONGINT;
                        Mask     IN D1 : BITSET):BOOLEAN;

LIBRARY RexxBase BY -168
  PROCEDURE IsRexxMsg(MsgPtr IN A0 : MessagePtr):BOOLEAN;


LIBRARY RexxBase BY -450
  PROCEDURE LockRexxBase(Resource IN D0 : RsrcNodeType);

LIBRARY RexxBase BY -456
  PROCEDURE UnlockRexxBase(Resource IN D0 : RsrcNodeType);
```

```
GROUP
  TypeGrp  =  AttributeFlags,AttributeFlagSet,NexxStr,NexxStrPtr,intNum,
              dpNum,alpha,owned,keepStr,keepNum,RexxArg,RexxArgPtr,
              maxRMArg,Command,ModifierFlags,ModifierFlagSet,ActionRec,
              RexxMsg,RexxMsgPtr,RexxRsrc,RexxRsrcPtr,RsrcNodeType,
              globalsz,RexxTaskFlags,RexxTaskFlagSet,RexxTask,RexxTaskPtr,
              memQuant,memMask,memQuick,SrcNodePtr,SrcNode,rxsdir,
              rxstname,RxsLib,tcOpen,stop,vers,rev,rxsalloc,rxschunk,
              rxsnest,rxstpri,rxsstack,rxslisth,CharAttrFlags,
              CharAttrFlagSet,buffsz,IoBuff,IoBuffPtr,exist,strf,read,
              write,append,RexxMsgPort,stack,queue,fail,RexxBase;

  ProcGrp  =  CreateArgstring,DeleteArgstring,LengthArgstring,
              CreateRexxMsg,DeleteRexxMsg,
              ClearRexxMsg,FillRexxMsg,IsRexxMsg,
              LockRexxBase,UnlockRexxBase;

  All      =  TypeGrp,ProcGrp;

END Rexx.
```


8.37 SCSIDisk

```
DEFINITION MODULE SCSIDisk;
(* $A- *)
CONST
  scsiCmd     = 28;

|Error values
  selfUnit    = 40;
  dma         = 41;
  phase       = 42;
  parity      = 43;
  selTimeout  = 44;
  badStatus   = 45;
  noBoard     = 50;

TYPE
  SCSIFlags      = (readNotWrite,autoSense,oldAutoSense);
  SCSIFlagSet    = SET OF SCSIFlags;

  SCSICmd        = RECORD
                     data        : ANYPTR;
                     length      : LONGCARD;
                     actual      : LONGCARD;
                     command     : ANYPTR;
                     cmdLength,
                     cmdActual   : CARDINAL;
                     flags       : SCSIFlagSet;
                     status      : SHORTCARD;
                     senseLength : CARDINAL;
                     senseActual : CARDINAL;
                   END;
  SCSICmdPtr     = POINTER TO SCSICmd;


GROUP
  All   = scsiCmd,selfUnit,dma,phase,parity,selTimeout,badStatus,noBoard,
          SCSIFlags,SCSIFlagSet,SCSICmd,SCSICmdPtr;

END SCSIDisk.
```

8.38 Serial

```
DEFINITION MODULE Serial;
(* $A- *)
FROM System     IMPORT LONGSET;
FROM T_Exec     IMPORT IOCommand, nonstdVAL, IOStdReq, DevicePtr,
                       IOFlagSet, IOFlags;
FROM Resources  IMPORT ContextPtr;
FROM Utility    IMPORT StdTags;

CONST
  query       = IOCommand( nonstdVAL );
  break       = IOCommand( nonstdVAL + 1 );
  setParams   = IOCommand( nonstdVAL + 2 );
  active      = IOFlagSet:{ IO4 };
  abort       = IOFlagSet:{ IO5 };
  queued      = IOFlagSet:{ IO6 };
  bufrRead    = IOFlagSet:{ IO7 };

TYPE
  SerFlags       = (parityOn,parityOdd,sevenWire,queuedBrk,radBoogie,
                    shared,eofMode,xDisabled);
  SerFlagSet     = SET OF SerFlags;

  ExtSerFlags    = (mark,mSpOn,esf2,esf3,esf4,esf5,esf6,esf7,esf8,esf9,
                    esf10,esf11,esf12,esf13,esf14,esf15,esf16);
  ExtSerFlagSet  = SET OF ExtSerFlags;
  Status         = (busy,paperOut,select,dataSetReady,clearToSend,
                    carrierDetect,readyToSend,dataTerminalReady,overrun,
                    wroteBreak,readBreak,xOffWrite,xOffRead);
  StatusSet      = SET OF Status;

  Error          = (e0,devBusy,baudMismatch,invBaud,bufErr,invParam,
                    lineErr,notOpen,portReset,parityErr,initErr,
                    timerErr,bufOverflow,nodsr,nocts,detectedBreak);

  IOSerial       = RECORD OF IOStdReq
                     ctlChar   : LONGCARD;
                     rBufLen   : LONGCARD;
                     extFlags  : ExtSerFlagSet;
                     baud      : LONGCARD;
                     brkTime   : LONGCARD;
                     termArray : ARRAY [0..1] OF LONGCARD;
                     readLen   : SHORTCARD;
                     writeLen  : SHORTCARD;
                     stopBits  : SHORTCARD;
                     serFlags  : SerFlagSet;
                     status    : StatusSet;
                   END;
  IOSerialPtr    = POINTER TO IOSerial;
```



```
  IOSerialTags   = TAGS OF StdTags
                     ctlChar   : LONGCARD;
                     rBufLen   : LONGCARD;
                     extFlags  : ExtSerFlagSet;
                     baud      : LONGCARD;
                     brkTime   : LONGCARD;
                     termArray0: LONGCARD;
                     termArray1: LONGCARD;
                     readLen   : SHORTCARD;
                     writeLen  : SHORTCARD;
                     stopBits  : SHORTCARD;
                     serFlags  : SerFlagSet;
                   END;


CONST
  ringIndicator  = select;

EXCEPTION SetParamsFailed : "SetParams failed";


PROCEDURE OpenSerial(context : ContextPtr:=NIL;
                     tags    : LIST OF IOSerialTags):IOSerialPtr;

PROCEDURE CloseSerial(VAR request : IOSerialPtr);


GROUP
  All  = query,break,setParams,active,abort,queued,bufrRead,SerFlags,
         SerFlagSet,ExtSerFlags,ExtSerFlagSet,Status,StatusSet,Error,
         IOSerial,IOSerialPtr,ringIndicator,OpenSerial,CloseSerial,
         T_Exec.ExecIOGrp;


END Serial.
```

8.39 Timer

```
DEFINITION MODULE Timer;
(* $A- *)
FROM T_Exec    IMPORT nonstdVAL, IOCommand, IORequest, DevicePtr;
FROM System    IMPORT Regs;
FROM Resources IMPORT ContextPtr;

CONST
  addRequest    = IOCommand( nonstdVAL + 0 );
  getSysTime    = IOCommand( nonstdVAL + 1 );
  setSysTime    = IOCommand( nonstdVAL + 2 );

TYPE
  TimerUnits   = (microHz, vBlank, eClock, waitUntil, waitEClock);

TYPE
  TimeVal      = RECORD
                   secs,
                   micro : LONGCARD;
                 END;
  TimeValPtr   = POINTER TO TimeVal;


  EClockVal    = RECORD
                   evHi  : LONGCARD;
                   evLo  : LONGCARD;
                 END;
  EClockValPtr = POINTER TO EClockVal;


  IOTimer      = RECORD OF IORequest
                   IF KEY : BOOLEAN
                     OF TRUE  THEN time   : TimeVal;
                     OF FALSE THEN eClock : EClockVal;
                   END;
                 END;
  IOTimerPtr   = POINTER TO IOTimer;


VAR
  TimerBase    : DevicePtr;

LIBRARY TimerBase BY -42
  PROCEDURE AddTime(VAR dest   IN A0,
                        source IN A1  : TimeVal);

LIBRARY TimerBase BY -48
  PROCEDURE SubTime(VAR dest   IN A0,
                        source IN A1  : TimeVal);
```



```
LIBRARY TimerBase BY -54
  PROCEDURE CmpTime(VAR time1 IN A0,
                        time2 IN A1  : TimeVal):INTEGER;

LIBRARY TimerBase BY -60
  PROCEDURE ReadEClock(VAR dest IN A0 : EClockVal):LONGCARD;

LIBRARY TimerBase BY -66
  PROCEDURE GetSysTime(VAR dest IN A0 : TimeVal);

PROCEDURE OpenTimer(unit    : TimerUnits;
                    context : ContextPtr := NIL ):IOTimerPtr;

PROCEDURE CloseTimer(VAR request : IOTimerPtr);

GROUP
  All   = addRequest,getSysTime,setSysTime,TimerUnits,TimeVal,TimeValPtr,
          IOTimer,IOTimerPtr,TimerBase,AddTime,SubTime,CmpTime,OpenTimer,
          CloseTimer,T_Exec.ExecIOGrp;

END Timer.
```

8.40 TrackDisk

```
DEFINITION MODULE TrackDisk;
(* $A- *)

|S. Herr, 30.09.92

FROM T_Exec      IMPORT IOCommand, nonstdVAL, IOStdReq, Unit, IOFlags,
                        IOReturn;
FROM System      IMPORT SHORTSET;
FROM Resources   IMPORT ContextPtr;

CONST
  motor            = IOCommand( nonstdVAL + 0 );
  seek             = IOCommand( nonstdVAL + 1 );
  format           = IOCommand( nonstdVAL + 2 );
  remove           = IOCommand( nonstdVAL + 3 );
  changeNum        = IOCommand( nonstdVAL + 4 );
  changeState      = IOCommand( nonstdVAL + 5 );
  protStatus       = IOCommand( nonstdVAL + 6 );
  rawRead          = IOCommand( nonstdVAL + 7 );
  rawWrite         = IOCommand( nonstdVAL + 8 );
  getDriveType     = IOCommand( nonstdVAL + 9 );
  getNumTracks     = IOCommand( nonstdVAL + 10 );
  addChangeInt     = IOCommand( nonstdVAL + 11 );
  remChangeInt     = IOCommand( nonstdVAL + 12 );
  getGeometry      = IOCommand( nonstdVAL + 13 );
  eject            = IOCommand( nonstdVAL + 14 );
  lastComm         = IOCommand( nonstdVAL + 15 );

  extCom           = CARDINAL($8000);
  extWrite         = IOCommand( extCom + CARDINAL(write) );
  extRead          = IOCommand( extCom + CARDINAL(read) );
  extMotor         = IOCommand( extCom + CARDINAL(motor) );
  extSeek          = IOCommand( extCom + CARDINAL(seek) );
  extFormat        = IOCommand( extCom + CARDINAL(format) );
  extUpdate        = IOCommand( extCom + CARDINAL(update) );
  extClear         = IOCommand( extCom + CARDINAL(IOCommand.clear) );
  extRawRead       = IOCommand( extCom + CARDINAL(rawRead) );
  extRawWrite      = IOCommand( extCom + CARDINAL(rawWrite) );

  numSecs          = 11;
  numUnits         = 4;

  sector           = 512;
  secShift         = 9;

  labelSize        = 16;

  | Flags für die IORequest Struktur
  |
  indexSync        = IO4;
  wordSync         = IO5;
```



```
TYPE
  | Flags für OpenDevice()
  |
  TDFlags          = (allowNon3_5,dummy=31);
  TDFlagSet        = SET OF TDFlags;

  getDriveTypeFlags   = (drive3_5, drive5_25, drive3_5_150RPM);
  getDriveTypeFlagSet = SET OF getDriveTypeFlags;

CONST
  | Driver Errors
  |
  notSpecified     = IOReturn(20);
  noSecHdr         = IOReturn(21);
  badSecPreamble   = IOReturn(22);
  badSecID         = IOReturn(23);
  badHdrSum        = IOReturn(24);
  badSecSum        = IOReturn(25);
  tooFewSecs       = IOReturn(26);
  badSecHdr        = IOReturn(27);
  writeProt        = IOReturn(28);
  diskChanged      = IOReturn(29);
  seekError        = IOReturn(30);
  noMem            = IOReturn(31);
  badUnitNum       = IOReturn(32);
  badDriveType     = IOReturn(33);
  driveInUse       = IOReturn(34);
  postReset        = IOReturn(35);


TYPE
  | Typen für das driveGeometry-Kommando
  |
  DeviceTypes       = (directAccess,sequentialAccess,printer,processor,worm,
                       cdRom,scanner,opticalDisk,mediumChanger,
                       communication,unknown=31);

  GeoFlags          = (removable);
  GeoFlagSet        = SET OF GeoFlags;

  DriveGeometry     = RECORD
                        sectorSize   : LONGCARD;
                        totalSectors : LONGCARD;
                        cylinders    : LONGCARD;
                        cylSectors   : LONGCARD;
                        heads        : LONGCARD;
                        trackSectors : LONGCARD;
                        bufMemType   : LONGCARD;
                        deviceType   : DeviceTypes;
                        flags        : GeoFlagSet;
                        reserved     : CARDINAL;
                      END;
  DriveGeometryPtr = POINTER TO DriveGeometry;
```

```
TYPE
  IOTrackDiskPtr = POINTER TO IOTrackDisk;
  IOTrackDisk    = RECORD OF IOStdReq;
                     count    : LONGCARD;
                     secLabel : LONGCARD
                   END;


  DriveNum       = ( DF0, DF1, DF2, DF3);



  PubFlags         = (noClick);
  PubFlagSet       = SET OF PubFlags;

  TDUPublicUnitPtr = POINTER TO TDUPublicUnit;
  TDUPublicUnit    = RECORD OF Unit;
                       comp01Track    : CARDINAL;
                       comp10Track    : CARDINAL;
                       comp11Track    : CARDINAL;
                       stepDelay      : LONGCARD;
                       settleDelay    : LONGCARD;
                       retryCnt       : SHORTCARD;
                       pubFlags       : PubFlagSet;
                       currTrk        : CARDINAL;
                       calibrateDelay : LONGCARD;
                       counter        : LONGCARD;
                     END;


PROCEDURE OpenTrackDisk( drive   : DriveNum:=DF0;
                         flags   :=TDFlagSet:{};
                         context : ContextPtr:=NIL ): IOTrackDiskPtr;

PROCEDURE CloseTrackDisk( VAR request : IOTrackDiskPtr );


GROUP

  CommandGrp = motor,seek,format,remove,changeNum,changeState,protStatus,
               rawRead,rawWrite,getDriveType,getNumTracks,addChangeInt,
               remChangeInt,getGeometry,eject,lastComm,extCom,extWrite,
               extRead,extMotor,extSeek,extFormat,extUpdate,extClear,
               extRawRead,extRawWrite;

  ConstGrp   = numSecs,numUnits,sector,secShift,labelSize,indexSync,
               wordSync;

  MiscGrp    = TDFlags,TDFlagSet,getDriveTypeFlags,getDriveTypeFlagSet;

  ErrorGrp   = notSpecified,noSecHdr,badSecPreamble,badSecID,badHdrSum,
               badSecSum,tooFewSecs,badSecHdr,writeProt,diskChanged,
               seekError,noMem,badUnitNum,badDriveType,driveInUse,
               postReset;
```



```
  GeometryGrp= DeviceTypes,GeoFlags,GeoFlagSet,DriveGeometry,
               DriveGeometryPtr;

  DeviceGrp  = IOTrackDiskPtr,IOTrackDisk,OpenTrackDisk,CloseTrackDisk,
               DriveNum;

  UnitGrp    = PubFlags,PubFlagSet,TDUPublicUnitPtr,TDUPublicUnit;


  All        = CommandGrp,ConstGrp,MiscGrp,ErrorGrp,GeometryGrp,DeviceGrp,
               UnitGrp;



END TrackDisk.
```

8.41 Translator

```
DEFINITION MODULE Translator
(* $A- *)
FROM Exec   IMPORT LibraryPtr;
FROM System IMPORT SysStringPtr,Regs;

VAR
  TranslatorBase : LibraryPtr;

CONST
  notUsed  = -1;
  noMem    = -2;
  makeBad  = -4;

$$OwnHeap:=TRUE
PROCEDURE NewTrans(REF in : STRING):STRING;

LIBRARY TranslatorBase BY -30
  PROCEDURE Translate(in     IN A0 : SysStringPtr;
                      inLen  IN D0 : LONGINT;
                      out    IN A1 : SysStringPtr;
                      outLen IN D1 : LONGINT):BOOLEAN;
GROUP
  All  = TranslatorBase,notUsed,noMem,makeBad,NewTrans,
         TranslatorBase;

END Translator.
```


8.42 Utility

```
DEFINITION MODULE Utility;


FROM Exec    IMPORT MinNode,LibraryPtr;
FROM System  IMPORT Regs,LONGSET;

CONST
  tagUser         = $80000000;

TYPE
  ClockData       = RECORD
                      sec,
                      min,
                      hour,
                      mday,
                      month,
                      year,
                      wday       :  CARDINAL
                    END;

  HookPtr         = POINTER TO Hook;

  HookCall        = PROCEDURE(hook    : HookPtr;
                              object  : ANYPTR;
                              message : ANYPTR);

  Hook            = RECORD OF MinNode
                      entry    : PROCEDURE(hook    IN A0 : HookPtr;
                                           object  IN A2 : ANYPTR;
                                           message IN A1 : ANYPTR);
                      subEntry : HookCall;
                      data     : ANYPTR
                    END;

  Tag             = LONGCARD;

  TagArrayPtr     = POINTER TO TagArray;  | an array of Tag values
  TagArray        = ARRAY OF Tag;         | without data association

  TagItemPtr      = POINTER TO TagItem;
  TagItem         = RECORD
                      tag  : Tag;
                      data : LONGCARD;
                    END;

  TagAPtr         = POINTER TO TagA;
  TagA            = ARRAY OF TagItem;
```

```
  StdTags         = TAGS
                      DONE   = 0;
                      IGNORE      : ANYPTR;
                      MORE        : TagAPtr;
                      MOREA  = 2  : ANYPTR;  | for easy programming
                      SKIP        : INTEGER;
                      USER   = tagUser - 1;
                    END;

  StdTagAPtr      = POINTER TO StdTagA;
  StdTagA         = ARRAY OF StdTags;

  TagListPtr      = POINTER TO TagList;
  TagList         = StdTagA;

VAR
  UtilityBase     : LibraryPtr;


PROCEDURE Dispatcher(hook    IN A0     :  HookPtr;
                     object  IN A2     :  ANYPTR;
                     message IN A1     :  ANYPTR);

PROCEDURE InitHook(VAR hook : Hook;
                       call : HookCall;
                       data : ANYPTR):HookPtr;

LIBRARY UtilityBase BY -30
  PROCEDURE FindTagItem(tagVal  IN D0 : LONGCARD;
                        tagList IN A0 : TagListPtr):TagItemPtr;

LIBRARY UtilityBase BY -36
  PROCEDURE GetTagData(tagVal  IN D0 : LONGCARD;
                       default IN D1 : LONGINT;
                       tagList IN A0 : TagListPtr):LONGINT;

LIBRARY UtilityBase BY -42
  PROCEDURE PackBoolTags(initFlags IN D0 : LONGSET;
                         tagList   IN A0 : TagListPtr;
                         boolMap   IN A1 : TagListPtr):LONGSET;

LIBRARY UtilityBase BY -48
  PROCEDURE NextTagItem(VAR tagItem IN A0 : TagItemPtr):TagItemPtr;

LIBRARY UtilityBase BY -54
  PROCEDURE FilterTagChanges(changeList IN A0,
                             oldValues  IN A1 : TagListPtr;
                             apply      IN D0 : LONGBOOL);

LIBRARY UtilityBase BY -60
  PROCEDURE MapTags(tagList     IN A0 : TagListPtr;
                    mapList     IN A1 : TagListPtr;
                    includeMiss IN D0 : LONGBOOL);
```



```
LIBRARY UtilityBase BY -66
  PROCEDURE AllocateTagItems(numItems IN D0 : INTEGER):TagListPtr;

LIBRARY UtilityBase BY -72
  PROCEDURE CloneTagItems(tagList IN A0 : TagListPtr):TagListPtr;

LIBRARY UtilityBase BY -78
  PROCEDURE FreeTagItems(tagList IN A0 : TagListPtr);

LIBRARY UtilityBase BY -84
  PROCEDURE RefreshTagItemClones(cloneList IN A0,
                                 origList  IN A1 : TagListPtr);

LIBRARY UtilityBase BY -90
  PROCEDURE TagInArray(tagVal   IN D0 : LONGCARD;
                       tagArray IN A0 : TagListPtr):LONGBOOL;

LIBRARY UtilityBase BY -96
  PROCEDURE FilterTagItems(tagList     IN A0 : TagListPtr;
                           filterArray IN A1 : TagArrayPtr;
                           logic       IN D0 : LONGINT);

LIBRARY UtilityBase BY -102
  PROCEDURE CallHookPkt(hook   IN A0 : HookPtr;
                        object IN A2 : ANYPTR;
                        params IN A1 : ANYPTR);


LIBRARY UtilityBase BY -120
  PROCEDURE Amiga2Date(    amigaTime IN D0 : LONGCARD;
                       VAR date      IN A0 : ClockData);

LIBRARY UtilityBase BY -126
  PROCEDURE Date2Amiga(VAR date IN A0 : ClockData):LONGCARD;

LIBRARY UtilityBase BY -132
  PROCEDURE CheckDate(VAR date IN A0 : ClockData):LONGCARD;

END Utility.
```

8.43 Workbench

```
DEFINITION MODULE Workbench;
(* $A- *)
FROM Dos       IMPORT FileLockPtr;
FROM Intuition IMPORT Gadget,GadgetFlags,GadgetFlagSet,NewWindow,WindowPtr;
FROM Exec      IMPORT Message,MsgPortPtr,List,LibraryPtr;
FROM System    IMPORT BPTR,SysStringPtr,Regs;

CONST
  diskMagic       = $E310;
  diskVersion     = 1;
  gadgetBackFill  = GadgetFlagSet:{gadgHBox};
  noIconPosition  = -1;

TYPE
  WBObjectType        = (wb0,disk,drawer,tool,project,garbage,device,kick);

  DiskObjectPtr       = POINTER TO DiskObject;
  DrawerDataPtr       = POINTER TO DrawerData;
  FreeListPtr         = POINTER TO FreeList;
  WBArgPtr            = POINTER TO WBArg;
  WBStartupPtr        = POINTER TO WBStartup;
  ToolTypeArrayPtr    = POINTER TO ARRAY OF SysStringPtr;

  WBArg          = RECORD
                     lock : FileLockPtr;
                     name : SysStringPtr;
                   END;

  WBArgumentsPtr = POINTER TO ARRAY OF WBArg;

  WBStartup      = RECORD OF Message
                     process   : MsgPortPtr;
                     segment   : BPTR;
                     numArgs   : LONGINT;
                     toolWindow: ANYPTR;
                     argList   : WBArgumentsPtr;
                   END;

  FreeList       = RECORD
                     numFree : INTEGER;
                     memList : List;
                   END;

  DiskObject     = RECORD
                     magic      : CARDINAL;
                     version    : CARDINAL;
                     gadget     : Gadget;
                     type       : WBObjectType;
                     defaultTool: SysStringPtr;
                     toolTypes  : ToolTypeArrayPtr;
                     currentX   : LONGINT;
```



```
                     currentY   : LONGINT;
                     drawerData : DrawerDataPtr;
                     toolWindow : ANYPTR;
                     stackSize  : LONGINT;
                   END;

  DrawerData     = RECORD
                     newWindow : NewWindow;
                     currentX  : LONGINT;
                     currentY  : LONGINT;
                     flags     : LONGINT;
                     modes     : CARDINAL;
                   END;

CONST
  drawerDataFileSize = DrawerData'SIZE;

TYPE
  AppMsgTypes    = (pstd,toolexit,diskchange,timer,closedown,ioproc,
                    appwindow,appicon,appmenuitem,copyexit,iconput,
                    makeword=$1000);

   | pstd        : standard message
   | toolexit    : exit message from tools
   | diskchange  : dos telling of a disk change
   | timer       : we got a time tick
   | appwindow   : msg from an app window
   | appicon     : msg from an app icon
   | appmenuitem : msg from an app menuitem
   | copyexit    : exit msg from copy process
   | iconput     : msg from PutDiskObject in icon.library

TYPE
  AppMessagePtr  = POINTER TO AppMessage;
  AppMessage     = RECORD OF Message
                     type     : AppMsgTypes;
                     userData : ANYPTR;
                     id       : LONGINT;
                     numArgs  : LONGINT;
                     wbArgs   : WBArgumentsPtr;
                     version  : CARDINAL;
                     class    : CARDINAL;
                     mouseX,
                     mouseY   : INTEGER;
                     seconds,
                     micros   : LONGINT;
                     reserved : ARRAY [8] OF LONGINT;
                   END;

TYPE
  AppWindowPtr   = HIDDEN;
  AppIconPtr     = HIDDEN;
  AppMenuItemPtr = HIDDEN;

VAR
```

```
  StartupMsg     : WBStartupPtr;
  WorkbenchBase  : LibraryPtr;

LIBRARY WorkbenchBase BY -48
  PROCEDURE AddAppWindow(    id       IN D0 : LONGINT;
                             userData IN D1 : ANYPTR;
                             window   IN A0 : WindowPtr;
                             msgPort  IN A1 : MsgPortPtr;
                             taglist  IN A2 : ANYPTR):AppWindowPtr;

LIBRARY WorkbenchBase BY -54
  PROCEDURE RemoveAppWindow(window IN A0 : AppWindowPtr);

|LIBRARY WorkbenchBase BY -60  | geht so nicht wegen A4

  PROCEDURE AddAppIcon(    id       IN D0 : LONGINT;
                           userData IN D1 : ANYPTR;
                       REF text     IN A0 : STRING;
                           msgport  IN A1 : MsgPortPtr;
                           lock     IN A2 : FileLockPtr;
                           diskobj  IN A3 : DiskObjectPtr):AppIconPtr;

LIBRARY WorkbenchBase BY -66
  PROCEDURE RemoveAppIcon(appIcon IN A0 : AppIconPtr);

LIBRARY WorkbenchBase BY -72
  PROCEDURE AddAppMenuItem(    id       IN D0 : LONGINT;
                               userData IN D1 : ANYPTR;
                           REF text     IN A0 : STRING;
                               msgport  IN A1 : MsgPortPtr;
                               tags     IN A2 : ANYPTR):AppMenuItemPtr;
                               | ^= NIL for now


LIBRARY WorkbenchBase BY -78
  PROCEDURE RemoveAppMenuItem(appMenu IN A0 : AppMenuItemPtr);

GROUP
  All    = diskMagic,diskVersion,gadgetBackFill,noIconPosition,
           WBObjectType,DiskObjectPtr,DrawerDataPtr,
           FreeListPtr,WBArgPtr,WBStartupPtr,WBArg,WBArgumentsPtr,
           FreeList,DiskObject,DrawerData,WBStartup,drawerDataFileSize,
           StartupMsg;

END Workbench.
```


Anhang

Anhang A

Lösungen

Achtung! Lesen Sie nur weiter, wenn Sie dies mit Ihrem Gewissen vereinbaren können.

Anschließend sind die Lösungen zu allen Übungsaufgaben des Einführungskapitels angeführt.

Sie sollten sich wirklich genau überlegen, ob Sie die Lösung nicht selbst erarbeiten wollen. Oft ist es ein Zeichen von Schwäche, wenn man in einer vorgegebenen Lösung nachschaut. Benutzen Sie diese Auflistung also nur, wenn Sie kontrollieren wollen, ob das von Ihnen erarbeitete Ergebnis richtig ist. Bedenken Sie dabei, daß es bei der Programmierung oft verschiedene Lösungsmöglichkeiten gibt. Aus diesem Grunde können die hier aufgeführten Ergebnisse von Ihren abweichen, jedoch sollten die Resultate natürlich gleich sein.

A.1 Lösungen 1

1. b, c, d, f
2. `CARDINAL`, `LONGCARD`, `SHORTCARD`, `INTEGER`, `SHORTINT`, `LONGINT`, `REAL`, `LONGREAL`, `BOOLEAN`, `CHAR`
3. Wenn REAL-Zahlen geteilt werden, wird das „`/`"-Zeichen benutzt, mit `DIV` werden INTEGER- und CARDINAL-Zahlen dividiert.
4. Die Zuweisung ist ungültig, da `ch` und `I` nicht den gleichen Typ haben.

A.2 Lösungen 2

1. Sie gibt die Meldung „`Ich liebe \Cluster{}!`" aus.
2.
```
MODULE SummeIntegers;

FROM InOut IMPORT ReadInt, WriteString, WriteLn;

VAR
  I,J, Summe: INTEGER;

BEGIN
  WriteString("Erste Zahl eingeben ");
  ReadInt(I);
  WriteLn; | Eine Leerzeile erzeugen
  WriteString("Zweite Zahl eingeben ");
  ReadInt(J);
  WriteLn; | Noch eine Leerzeile
  Summe := I + J;
  WriteString("Die Summe ist ");
  WriteInt(Summe)
END SummeIntegers.
```
3. Der CLOSE-Teil wird dann ausgeführt, wenn das eigentliche Programm beendet wird. Dies geschieht bei einem Programmabbruch durch den Benutzer, durch einen Fehler oder auch bei normalem Programmverlauf. Hier befinden sich Anweisungen, die reservierten Speicher freigeben oder offene Fenster wieder schließen.
4. CARDINAL-Werte können nur positive ganze Zahlen sein, wohingegen INTEGER-Werte positive und auch negative ganzzahlige Werte annehmen können.
5. (a) `BOOLEAN`
   (b) `LONGINT` (Zahlenbereich beachten!)
   (c) `REAL` oder `LONGREAL`
   (d) `SHORTCARD`, `CARDINAL`, `LONGCARD`, `SHORTINT`, `INTEGER`, `LONGINT`

   (e) `CHAR`
   (f) `ARRAY OF CHAR` oder `STRING`

6. ```
   CONST
     TestWert = 70000;
   ```
7. c und d.
8. ```
   MODULE Wandel;

   VAR
     R: REAL;
     I: INTEGER;

   BEGIN
     R := 1005.97;
     I := INTEGER (R);
     R := REAL (I)
   END Wandel.
   ```


A.3 Lösungen 3

1. ```
   MODULE SumSum;

   FROM InOut IMPORT ReadInt;

   VAR
     i,j,Summe: INTEGER;

   BEGIN
     ReadInt(i);
     Summe := 0;
     FOR j := i TO 0 BY -1 DO
       Summe := Summe + j;
       | oder INC(Summe,j);
     END
   END SumSum.
   ```
2. REPEAT..UNTIL wird mindestens einmal ausgeführt, wobei WHILE..DO je nach Schleifenbedingung möglicherweise überhaupt nicht durchlaufen wird.

   Weiterhin wird WHILE..DO ausgeführt, wenn eine Bedingung TRUE ist, REPEAT..UNTIL, wenn eine Bedingung FALSE ist.
3. WHILE..DO, REPEAT..UNTIL, LOOP..EXIT, FOR.
4. 50
5. Der ELSE-Teil (Anweisung5)! `a` ist negativ, deshalb ist die erste Bedingung `FALSE`, `b` ist zwar `TRUE` wird durch `NOT` allerdings negiert, so daß die zweite Bedingung `FALSE` wird (`c OF ...` braucht nicht mehr ausgewertet zu werden, da eine mit **`AND`** verknüpfte Bedingung `FALSE` wird, wenn nur eine der beiden Werte `FALSE` wird).

   Anweisung4 wird scheinbar ausgeführt, beachten Sie aber die Groß/Kleinschreibung, so werden Sie sehen, daß „`d`“ nicht in **`OF`** `"B".."Z"` enthalten ist.
   ```

A.4 Lösungen 4

1. Eine Aufzählung ist ein benutzerdefinierter Typ, bei dem alle möglichen Werte namentlich aufgeführt sind. Ein Unterbereich ist ein benutzerdefinierter Typ, der einem begrenzten Bereich eines vordefinierten Typs oder eines Standardtyps entspricht.

2. Eine Aufzählung ist eine Auflistung der möglichen Werte eines Typs, eine Menge eine Zusammenfassung von Objekten.

3. Der Wertebereich für `Y` (1-10) wird überschritten.

4. `A + B = {a, b, c, d, e, f, g}`

   `A * B = {d}`

   `A - B = {a, b, c}`

   `A / B = {a, b, c, e, f, g}`

5. Nein, da sie nicht außerhalb eines MODULEs bestehen kann.

6. Global: `B` und `C`. Lokal: `X` und `ZAEHLER`.

7. `Willi: ARRAY [0..9], [0..99], [0..8] OF INTEGER;`

8. Die Felder sind nicht vom selben Typ und können nicht direkt zugewiesen werden.

9. a: 202, b: 242, c: 2

10. Ein Feld muß aus gleichen Elementen bestehen, wogegen ein RECORD aus vielen unterschiedlichen Variablentypen zusammengesetzt sein kann.

11.
```
TYPE
  Flugzeug = RECORD
               Motoren: SHORTCARD;
               Reichweite: LONGCARD;
               PassagierZahl: CARDINAL;
               Name: STRING(30)
             END;
```


12.
```
PROCEDURE Anzeigen (A: Flugzeug);
BEGIN
  WriteCard(Motoren,5);
  WriteCard(PassagierZahl,5);
  WriteCard(Reichweite,5);
  WriteString(Name);
END Anzeigen;
```

A.5 Lösungen 5

1. Die Routine ruft sich selbst unendlich auf und stürzt schließlich ab, egal von welchem Programm sie aufgerufen wurde.

2. Zuerst verringert die Arbeit mit kleineren Modulen die Compilierungszeit während der Entwicklung. Weiterhin kann jedes Modul unabhängig von einem eigenen Team entwickelt werden.

3. Ein Definitionsmodul gibt einfach an, was eine Bibliothekseinheit leistet; das entsprechende Implementationsmodul enthält tatsächlich den Code für die Ausführung der Funktionen.

4. (a) Da `InOut` unqualifiziert importiert wurde, müssen alle Aufrufe der Bezeichner ausdrücklich qualifiziert werden.

   (b) Das Modulendekennzeichen lautet fälschlicherweise `A`.

   Das richtige Programm lautet:

```
MODULE BSP1; |Das ist richtig!
IMPORT InOut;
BEGIN
  InOut.WriteString("Cluster la vista!");
  InOut.WriteLn;
END BSP1.
```


Anhang B

Fehlermeldungen

B.1 Fehlermeldungen des Editors

- Fehlerhaftes Cluster-Format

? Sie haben versucht, einen Text zu laden, dessen Format weder dem ASCII-Format noch dem Cluster-Format entspricht. Oder einen Text, der im Cluster-Format gespeichert wurde hat einen Format-Fehler.

! Laden Sie nur ASCII- oder Cluster-Texte ein.

- Gewünschter Text konnte nicht geladen werden

? Entweder Name oder Pfad des Textes waren falsch, wodurch der Text nicht geladen werden konnte.

! Kontrollieren Sie den Pfad und Namen auf ihre Richtigkeit.

- Gewünschter Text konnte nicht gesichert werden

? Wahrscheinlich ist Ihre Diskette oder Festplatte voll.

! Schaffen Sie Platz auf Ihrem Speichermedium.

- Kein Block in diesem Text markiert

? Sie haben COPY, CUT oder DELETE angewählt, ohne daß zuvor ein Block markiert wurde.

! Markieren Sie einen Block.

- Kein Block vorhanden

? Sie haben PASTE angewählt, ohne das ein Block im Puffer war.

! Es muß erst ein Block im Puffer stehen, wenn Sie ihn kopieren wollen.

- Letzte Möglichkeit. Sichern???

? Diese Abfrage befindet sich im CLOSE-Teil des Editors. Sie erscheint normalerweise nur dann, wenn Sie den Editor verlassen wollen und einen Text noch nicht gesichert haben. Erscheint sie zu einem anderen Zeitpunkt, ist mit Sicherheit mit dem Absturz des Editors zu rechnen. Sollten Sie also nicht ganz sicher sein, daß Sie den Text nicht mehr brauchen, sollten Sie ihn auf alle Fälle sichern.

! Text abspeichern oder nicht.

- Nicht genug Systemspeicher für die gewünschte Funktion

? Sie haben versucht mehr Textspeicher einzustellen, als von Ihrer Speicherkonfiguration her möglich.

! Beenden Sie andere Tasks, löschen Sie unnötige Files aus der Ram-Disk oder kaufen Sie sich eine Speichererweiterung.

- Nicht genug Textspeicher für gewünschte Funktion

? Sie haben entweder einen Text zu laden versucht, der nicht in den aktuellen Textspeicher paßt oder durch eine Blockoperation einen bestehenden Text so vergrößert, daß er nicht mehr in den Speicher passen würde.

! Stellen Sie in den Voreinstellungen eine größere Textspeichergröße ein.

- Rettungs-Save. Text nicht gesichert. Sichern???

? Erscheint diese Meldung, sollten Sie „JA“ anwählen, wenn Sie sich nicht sicher sind, ob Sie den Text schon gespeichert haben. Dieser Requester kann nämlich mehrere Ursachen haben: Die harmloseste ist, daß Sie in den Preferences CHICON angewählt haben und ICONIZ aufgerufen haben. Eine andere Möglichkeit ist, daß Sie versuchten einen veränderten Text zu löschen oder einen anderen Text versuchen über den geänderten zu laden. Die schlimmste Möglichkeit ist jedoch, daß der Editor gerade am abstürzen ist und ihnen noch die Gelegenheit geben will ihre Texte zu speichern. Daher sollten Sie vor allem, wenn diese Meldung sehr plötzlich auftaucht (z. B. bei einem RUN oder MAKE oder bei sonst einer Gelegenheit, für die Sie nicht verantwortlich sind), „JA“ anwählen.

! Text abspeichern oder nicht.

B.2 Feldermeldungen des Compilers

Bei einigen Fehlermeldungen, ist als Ursache „Compilerfehler“ angegeben. Diese Fehler sollten eigentlich nicht mehr vorkommen. Sollte dies dennoch geschehen, ist dies meist kein Fehler des Compilers, sondern Folge einer seltenen Fehlerkombination, die den Compiler aus dem „Takt“ gebracht hat. Falls ein solcher Fehler einmal bei Ihnen auftritt, wären wir Ihnen dankbar, wenn Sie und dies melden und uns das Programm, bei dem der Fehler auftrat, zusenden würden.

- „;“ erwartet

? Sie haben hinter der letzten Anweisung das Semikolon vergessen.

! Fügen Sie ein Semikolon an der richtigen Stelle ein. Achtung: Bei diesem Fehler kann der Compiler nicht die exakte Position angeben, er steht meist etwas unterhalb.

- „ “ erwartet

? Sie haben ein Leerzeichen vergessen, daß an dieser Stelle laut Sprachdefinition stehen sollte.

! Fügen Sie ein Leerzeichen an der richtigen Stelle ein.

- „." erwartet

? Sie haben bei einer Qualifizierung oder am Modulende den Punkt vergessen.

- „.." erwartet.

? Sie haben bei einer Array- oder Unterbereichsdefinition zwischen den Indexgrenzen oder bei einem **OF** zwischen den Bereichsgrenzen, zwei Punkte vergessen.

- „)" erwartet.

? Sie haben bei einem Ausdruck mehr öffnende als schließende Klammern verwendet, oder bei einer Prozedurdeklaration wurde die schließende Klammer am Prozedurkopf vergessen.

! Zählen Sie die Klammern nach, und fügen Sie eventuell eine schließende Klammer ein. Bei einem Ausdruck liegt der Fehler oft nicht an einer falschen Anzahl sondern einfach an der falschen Position der Klammer.

- „/" nur für REAL und SET's

? Sie haben den „/"-Operator auf einen anderen Typen, als auf REAL oder einen Set angewandt.

! Kontrolieren Sie die Typen der verwendeten Variablen/Konstanten.

- „:=" erwartet

? Wahrscheinlich haben Sie bei einer Zuweisung statt „:=" nur „=" geschrieben.

- „&" oder „"" erwartet

? Wahrscheinlich haben Sie versucht einer Zeichen- oder Stringvariablen eine Zahl oder einen String ohne Anführungszeichen zuzuweisen.

! Kontrollieren Sie, ob es sich bei der Variablen wirklich um eine Zeichenvariable handeln soll, oder ob Sie nur ein „&" oder „"" vergessen haben.

- $W Option nicht möglich

? Sie haben die $W-Option gesetzt und dennoch einen Prozeduraufruf innerhalb der WITH-Struktur.

! Löschen Sie die Option oder rufen Sie die Prozedur außerhalb der WITH-Struktur auf.

- „'W' oder 'L' erwartet"

? Sie haben bei einer Assembleranweisung die Länge Byte verwendet, obwohl hier nur Wort oder Langwort möglich ist.

! Korrigieren Sie bitte die Längenangabe.

- „*" nicht für diesen Typen definiert
- „+" nicht für diesen Typen definiert
- „-" nicht für diesen Typen definiert

? Sie haben einen der oberen Operatoren auf einen Typen angewandt, für den dieser nicht definiert ist (z. B. BOOLEAN).

! Kontrollieren Sie den Typen der verwendeten Variable/Konstante.

- Adreßausdruck zu komplex

? Komplexe Adressierung bei akutem Registermangel, in Folge vieler Adreßregistervariablen.

! Werfen Sie einige der Variablen aus den Registern.

- Adreßregister erwartet als Parameter bei UNLK
- Adreßregister erwartet

? Sie haben eine indirekte Adressierung ohne Adreßregister als Basis versucht.

! Verwenden Sie ein Adreßregister.

- Anderer Typ als in Forward-Deklaration

? Bei der Deklaration einer FORWARD-deklarierten Prozedur unterscheiden sich sie Parameter in den Typen.

! Vergleichen Sie FORWARD- und Hauptdeklaration.

- Array erwartet

? Sie haben auf eine normale Variable, wie auf ein Array zugegriffen (`a[x]`)

! Kontrollieren Sie den Typen der verwendeten Variablen.

- Array bzw. String erwartet

? An einen SysStringPtr in einer Librarydefinition kann entweder ein SysStringPtr, ein String oder ein **ARRAY OF** `CHAR` übergeben, Sie haben versucht etwas anderes zu übergeben, was der Compiler selbstverständlich reklamiert.

- Array mit zu großem Index

? Sie haben ein Array mit mehr als 32000 Elementen definiert.

! Sollten Sie wirklich ein so großes Feld benötigen, realisieren Sie dies bitte über eine dynamische Listenstruktur.

- AS in WITH erwartet

? Sie haben in einer WITH-Struktur nur einen Typen angegeben, jedoch kein **AS** dahinter geschrieben.

! Prüfen Sie, ob der angegebene Typ auch wirklich ein Typ und nicht nur eine falsch geschriebene Variable ist. Ergänzen Sie gegebenenfalls ein **AS**.

- Attributsbezeichner erwartet

? Sie haben hinter einen Typen/Variable ein „`'`“ geschrieben, jedoch kein Attribut wie Z. B. `SIZE` oder`PTR`.

- Aufgeschobener Zeiger nicht definiert

? Der Zeiger war nicht in dem Modul definiert, aus dem er importiert wurde.

- Aufzählungs-/Unterbereichstyp erwartet:

? Sie haben bei einer Array-Definition als Indextypen keinen Aufzählungs-/Unterbereichstypen verwendet, sondern z. B. `REAL` oder `LONGRAL`.

- Aufzählungs- oder Numerischen Typen erwartet

? Sie haben bei einer Array-Definition als Indextypen keinen zählbaren Typen verwendet, sondern wahrscheinlich einen komplexen Typen (Record, Array) oder eine Menge oder einen Zeiger.

- Ausdruck nicht Adressierbar

? Sie haben versucht, die Adresse eines nicht adressierbaren Ausdrucks, z. B. einer Summe zu bilden. Dieser Fehler tritt auch auf, wenn Sie versuchen einen Ausdruck an einen VAR-Parameter zuübergeben.

! Weisen Sie den Ausdruck erst einer Variablen zu, oder verwenden Sie Werteparameter.

- Ausdruck zu komplex

? Ihr Audruck hat zu viele Schachtelungen (Klammern), so daß dem Compiler bei der Auswertung die Register ausgehen.

! Zerlegen Sie den Ausdruck in mehrere Teilausdrücke oder nehmen Sie einige Variablen aus Registern, sofern sie Registervariablen verwenden.

- Ausgabefile konnt nicht geöffnet werden

? Der Compiler konnte kein Symbol-/Objektfile erstellen, weil die Diskette/Festplatte voll war oder kein Verzeichnis `OBJ` gefunden wurde.

- BEGIN erwartet in Prozedur:

? Sie haben in einer Prozedur das **BEGIN** vergessen. Im Gegensatz zu Modulen ist dies bei Prozeduren Pflicht.

! Fügen Sie ein **BEGIN** ein.

- Bei CAST Typ als erster Parameter

? Sie haben bei CAST die Variable als ersten Parameter angegeben.

- Bereich erwartet

? Sie haben bei einer Arrayvariablen-Definition innerhalb eines Records den Bereich vergessen. Offene Arrays sind nur als Typen oder Konstanten erlaubt.

- Bereich des Objekts kann nicht ermittelt werden

? `'RANGE 'MAX 'MIN` ist für diesen Typen nicht definiert.

- Bezeichner erwartet

? Nach der Sprachdefinition erwartet der Compiler hier einen Bezeichner. Hierfür kann es ziemlich viele Gründe geben z. B. nach **VAR**, **TYPE**, **RECORD** etc..

- Bezeichner existiert bereits

? Sie haben einen Bezeichner definiert, der an anderer Stelle dieses Sichtbarkeitsbereiches schon verwendet wird.

- Bezeichner ist kein Typ

? Sie haben bei einer Deklaration als Typ einen Bezeichner angegeben, der kein Typ ist.

! Kontrollieren Sie die Definition des Bezeichners.

- Bezeichner nicht bekannt
- Bezeichner nicht sichtbar

? Sie haben einen Bezeichner verwendet, der dem Compiler noch nicht bekannt ist. Dies kann daran liegen, daß er noch nicht definiert wurde, daß ein Schreibfehler vorliegt, oder daß er erst später definiert wird.

! Kontrollieren Sie die Schreibweise und Deklaration des Bezeichners, deklarieren Sie ihn gegebenenfalls mit FORWARD.

- Bezeichner überladen nur für Aufzählungen

? Element eines Aufzählungstypen existiert schon als anderes Objekt.

! Kontrollieren Sie die Schreibweise, verwenden Sie gegebenenfalls einen anderen Bezeichner für das Element.

- BOOLEAN erwartet

? In **IF**, **WHILE**, **REPEAT**, etc. oder einer Boolzuweisung, steht als Bedingung kein bollscher Ausdruck. Dies passiert leicht, wenn man bei einem Funktionsaufruf, der einen Boolwert zurückliefert, die Klammern vergißt, wodurch der Compiler den Funktionsaufruf als Zeiger auf die Funktion interpretiert.

- BY erwartet

? Sie haben eine FOR-Schleife programmiert, bei der der Startwert größer als der Endwert ist. Bei runterzählenden Schleifen ist eine Schrittwertangabe jedoch zwingend. Die zweite Möglichkeit: Sie haben bei einer Libraryprozedur ein **BY** vergessen.

- CAST kann nur zwischen gleichlangen Typen konvertieren

? Sie haben versucht zwei verschieden lange Typen mit CAST zu konvertiern. Dies ist jedoch nicht möglich.

! Versuchen Sie zu konvertieren, indem sie `t(v)` schreiben wobei `t` der neue Type und`v` den zu konvertierenden Ausdruck/Variable darstellt.

- CHK nicht in REG - Compiler Fehler

? Compilerfehler

- CLASS ?

? Sie haben eine normale Variable wie einen offenen Typen benutzt.

- CLONE vorerst nur bei Objekten

? Sie haben CLONE auf ein Nicht-Objekt, also Variable, Konstante, Prozedur etc. angewendet. Dies ist nicht möglich.

- CLOSE erwartet

? Nach dem BEGIN eines Moduls folgte weder CLOSE noch END, sondern ein Element, daß innerhalb eines BEGIN-Teils nicht erlaubt ist.

- ComChkNil ohne InDirFlag

? Compilerfehler

- Datenregister als zweiten Typen erwartet.

? Ein Assemblerbefehl erwartet als zweiten Parameter ein Datenregister.

- Definitionsmodul nicht gefunden

? Modul konnte beim Import nicht gefunden werden.

! Kontrollieren Sie die Importpfade, überprüfen Sie, ob das Modul wirklich existiert.

- Der Typ der unteren Grenze muß mit dem der oberen übereinstimmen

? Bei einer Unterbereichsdefinition stimmt die untere Grenze nicht mit der oberen Grenze vom Typ her überein.

- Die Variable gibt's schon

? Sie haben zwei Variablen mit dem gleiche Namen definiert.

- Direktoperand erwartet

? Der von Ihnen verwendete Assemblerbefehl erwartet einen Direktoperanden.

- Displacement erwartet

? Sie haben bei der Definition einer Library-Prozedur das Displacement **BY** `-x` vergessen.

- Displacement ist zu groß.

? Bei einer indirekten Adressierung mit Displacement, ist das Displacement außerhalb des erlaubten Bereichs.

- DISPOSE bisher nur für Objekte

? Sie haben die Standardprozedur `DISPOSE` auf ein Nicht-Objekt angewandt.

- DIV nur für ganze Zahlen

? Sie haben den Operator DIV auf Real-Zahlen angewandt, obwohl dieser nur für Ganzzahlen definiert ist.

! Verwenden Sie „`/`“ für Real-Zahlen.

- Division durch 0

? Sie haben durch Null oder durch eine Konstante mit Wert Null geteilt.

- DO nach FOR erwartet
- DO nach WITH erwartet

? Sie haben nach **FOR** oder **WITH** „**DO**“ vergessen.

- DO, AND_WHILE erwartet

? Sie haben nach der Bedingung bei einer While-Schleife etwas anderes als „**DO**“ oder „**AND_WHILE**“ geschrieben.

- Doppelte Defintion

- Doppelte Defintion im gleichen Sichtbarkeitsbereich

? Sie haben einen Bezeichner definiert, den es in dieser Prozedurebene schon gibt.

- Doppelte Libary-Definition

? Sie haben in einer Libary-Definition zwei Prozeduren mit gleichem Namen definiert.

- Doppeltes Forward

? Eine Prozedur wurde zweimal mit FORWARD-definiert.

- Ein Operand erwartet

? Auf einen Assemblerbefehl, der ein einziges Argument erwartet, folgte entweder kein oder zwei Argumente.

- Eigene Symboldatei konnte nicht geöffnet werden

? Sie haben ein Implementationsmodul compiliert, und der Compiler konnte dessen Symboldatei nicht finden.

! Compilieren Sie erst das Definitionsmodul.

- Einfache Konstante erwartet

? Sie haben statt einer einfachen Zahlen- oder Zeichenkonstante eine typisierte Konstante verwendet.

- END in WITH erwartet

- END nach LOOP erwartet

? Der Compiler ist auf ein Prozedur-/Modulende gestoßen, bevor eine WITH-/LOOP-Struktur beendet worden ist.

- END erwartet

? Der Compiler merkt am Ende einer Prozedur/Modul, daß die Anzahl der begonnenen Strukturen nicht mit der der beendeten übereinstimmt.

! Ein Rezept zu geben ist schwierig, da der Fehler meist erst weit von dessen Ursprung entdeckt wird. In schweren Fällen empfiehlt es sich Teile auszukommentieren, um die Fehlerstelle zu lokalisieren.

- END oder CLOSE erwartet

? Nach dem BEGIN eines Moduls folgte weder CLOSE noch END, sondern ein Element, daß innerhalb eines BEGIN-Teils nicht erlaubt ist.

- Erster Operand bei SHL muß Ganzzahl oder SET sein

- Erster Operand bei SHR muß Ganzzahl oder SET sein

? Bei den Operatoren SHL/SHR muß der erste Operand eine Ganzzahl oder eine Menge sein, was in diesem Fall nicht der Fall war.

! Kontrollieren Sie die Typen der verwendeten Variablen.

- Erben nur von Klassen möglich.

? Sie können nur von Klassen erben.

- EXCEPT erwartet

? Nach einem **TRY** folgte kein **EXCEPT** sondern ein **END** oder ein **OF**.

! Kontrollieren Sie, ob ein überzähliges **END** existiert, oder ob Sie tatsächlich das **EXCEPT** vergessen haben. Vielleicht haben Sie auch einfach **TRY** mit **TRACK** verwechselt.

- Exception erwartet

? Nach einem **EXCEPT OF** folgte keine Exception/Exceptiongruppe. Möglicherweise ein Schreibfehler.

- EXG nur für Register

? Der Assemblerbefehl EXG ist nur auf Register anwendbar.

- Falsches Registerin NeedD0Ptr

- Falsches Registerin NeedD1Ptr
- Fehler bei WITH-Freigabe, Stackdifferenzen
- Fehler beim Import von Enumeration - Compilerfehler

? Compilerfehler

- Fehler in 'ActSwitches'

? Compilerfehler

- Fehlerhafter Operand

? Der verwendete Operator ist nicht für den Typ dieses Operanden definiert.

- Fehlerhafte Addressierungsart
- Fehlerhafte Adressierung
- Fehlerhafte Längenangabe

? Fehler, die bei der Assemblerprogrammierung auftreten können. Der Grund ist in der Fehlermeldung deutlich ersichtlich.

- Fehlerhafte Standardkonstante

? Compilerfehler

- Fehlerhafter Operand für USE

? Nach dem Assemblerbefehl USE folgte etwas anderes als eine Registerliste.

- Fehlerhaftes Symbol in Ausdruck

? An dieser Stelle des Ausdrucks darf dieses Symbol nicht verwendet werden.

- File ist kein Symbolfile

? Ein Symbolfile eines Moduls wurde zwar gefunden, hat aber einen Formatfehler.

! Compilieren Sie das entsprechende Definitionsmodul noch einmal.

- FOR Verregisterung

? Compilerfehler

- FPU mismatch

? Sie haben FPU-Register verwendet, ohne die FPU angeschaltet zu haben.

! Schalten Sie den Schalter `MC68881` ein.

- Gefüllte Liste ist leer ??

? Compilerfehler

- Generative Module nur für Pointer möglich

? Sie haben versucht ein generisches Modul mit einem anderen Typen als einem Pointer auszuprägen.

- Generatives Modul erwartet Typen

? Sie haben versucht ein generisches Modul ohne Typangabe zu importieren.

- Generativer Parameter paßt nicht

? Der aktuelle generische Parameter paßt nicht zur Formaldefinition im generischen Modul.

- Generatives Modul erwartet.

? Sie haben versucht ein einfaches Modul wie eine generisches zu importieren.

- Gruppen nur in Def.Modulen

? Sie haben versucht in einem Implementations- oder Hauptmodul eine Importgruppe zu definieren, welches nur in Definitionsmodulen erlaubt ist.

- Größe des Objects kann nicht ermittelt werden

? Das Attribut `'SIZE` ist für dieses Object nicht definiert.

- IF erwartet END

? Sie haben nach einer IF-Struktur das „**END**“ vergessen.

- Illegaler Funktionsaufrufmodus

? Sie haben versucht eine Prozedur als Funktion aufzurufen, oder umgekehrt.

- Illegaler Modus in NeedD0Ptr
- Illegaler Modus in NeedD1Ptr

? Compilerfehler

- Illegales Zeichen in Switch

? Bei der Definition von Compilerswitches gelten die gleichen Regeln für Schreibweise wie für alle anderen Bezeichner von Cluster. Siehe Sprachdefinition.

- Immediate oder Datenregister als Quelloperand

? Fehlermeldung des Assemblers.

- IMPORT erwartet

? Sie haben nach einem **FROM** *Modul* „**IMPORT**“ vergessen.

- IN erwartet

? Sie haben bei Registerparametern oder oder bei einem SET-Vergleich „**IN**“ vergessen.

- INC/DEC erwarten nummerischen Typ als zweiten Parameter

? Sie haben bei INC/DEC als zweiten Parameter einen komplexen Typen, einen Zeiger oder eine Realzahl übergeben.

- Indirekte Adressierung erwartet
- Indirekte Adressierung nach Displacement erwartet

? Fehlermeldung des Assemblers.

- Inkompatible Operandentypen
- Inkompatible Typen
- Inkompatible Zuweisung

? Sie haben bei einer Operation oder Zuweisung nicht kompatible Typen verwendet. Beachten Sie, daß es Typen gibt, die zwar zuweisungs- jedoch nicht rechnungskompatibel sind.

- Inkompatibler Indextyp

? Unterschiedliche Typen bei unterer und oberer Grenze eines Indexes einer Array-/Unterbereichsdefinition.

- Inkompatibler Tag typ

? Der Typ des Tagwertes stimmt nicht mit dem der Tagdefinition überein.

- INTEGER erwartet.

? Sie haben ein Object mit einem anderen Typen als Integer an einer Stelle verwendet, an der die Sprachdefinition Integer vorraussetzt.

- INTEGER-Konstante erwartet.

? Sie haben eine andere oder keine Konstante an einer Stelle verwendet, an der die Sprachdefinition eine Integer-Konstante vorraussetzt, z. B.: Sie haben bei einer FOR-Schleife als Schrittwert eine Realzahl angegeben.

- IS erwartet Objekttypen als zweiten Parameter

? Nur Objekte lassen sich Ihren Typ mit **IS** abfragen.

- Kein bekannter Recordbezeichner

? Sie haben ein Recordelement (`r.elem`) qualifiziert, welches nicht in diesem Record enthalten ist. Meist nur ein Tippfehler.

! Untersuchen Sie die Recorddefinition.

- Kein eigener Heap eingestellt

? Sie wollten an einen VAR-/REF-Parameter eine Funktion zuweisen, die einen offenen Typen z. B. STRING zurückgibt, ohne daß diese für einen eigenen Zwischenspeicher für den Rückgabewert sorgt.

! Entweder haben Sie nur vergessen `$$OwnHeap:=TRUE` zu setzen, oder sich gar nicht um einen Heap gekümmert. Im letzteren Fall sollten Sie unter refALLOC-RESULT nachlesen.

- Kein Element des Records

? Sie haben versucht auf ein Element eines Records zuzugreifen, das in der Definition nicht existiert.

- Kein generatives Modul

? Sie haben versucht ein einfaches (nicht generisches Modul) mit einer Typangabe zu importieren.

- Kein MAX dieses Typs
- Kein MIN dieses Typs

? Sie haben versucht, das größte bzw. kleinste Element eines nicht abzählbaren Typen zu ermitteln, was nicht möglich ist. Dies ist z. B. bei Record und Pointern der Fall.

- Kein Typ/Typkonstruktor

? In Ihrem Programm folgt hinter dem Doppelpunkt einer Variablen- oder dem Gleichheitszeichen einer Typdeklaration kein Typ oder Typkonstruktor wie **ARRAY**, **RECORD**, etc..

- Keine offenen Arrays forwarddeklarierbar

? Sie haben versucht ein offenes Array vor der eigentlichen Definition zu deklarieren, was nicht möglich ist.

- Kein offener Typ für RESULT

? Sie haben versucht einen Heap mit ALLOC_RESULT zu erzeugen, obwohl es sich bei dem Rückgabetypen um keinen offenen handelte.

- Kein Schreibzugriff auf Object erlaubt

? Sie können nicht auf Konstanten oder REF-Parameter schreibend zugreifen.

- Kein Stapel

? Sie haben versucht einen Switch zu stapeln, der dafür nicht geeignet ist.

- Keine DEFERRED Variablen

? Sie haben innerhalb eines Objekts versucht eine Variable Deffered zu erklären, die ist nur mit Methoden möglich.

- Keine direkte Nachfolgerbeziehung vorhanden

? Sie haben versucht zwei Objekte einander zuzuweisen, die zwar von einer gemeinsamen Klasse abstammen, jedoch nicht in einer Erblinie stehen, also eher Brüder, als Söhne und Väter sind. Eine derartige Zuweisung ist nicht möglich.

- Keine Methoden bei lokalen Prozeduren

? Innerhalb von lokalen Prozeduren kann man keine Methoden definieren.

- Keine Objekte als Variablen erlaubt

? Man kann keine Objekte als Variablen deklarieren, nur Zeiger auf Objekte.

- Keine Procedure

? Wahrscheinlich haben Sie einen Bezeichner, der keine Prozedur ist, als Prozedur oder Funktion aufzurufen versucht.

- Keine Parameter für Constructor
- Keine Parameter für Destructor

? Sie haben versucht die Methode `Construct/Destruct` eines Objectes mit Parametern aufzurufen. Dies ist jedoch nicht möglich.

- Keine Rückwärtsrecorddefinition

? Sie können nicht während der Definition eines konstanten Records, bei der Definition eines einzelnen Elements auf vorhergehende Elemente zurückgreifen.

- Keine Single.IEEE library, (REAL # FFP)!

? Sie haben REAL-Zahlen verwendet ohne die singleIEEE.library oder eine FPU zu besitzen.

! Verwenden Sie FFP-Zahlen.

- Keine Symboldatei ???

? Die Symboldatei eines Moduls hat einen Formatfehler.

! Übersetzen Sie das entsprechende Definitionsmodul noch einmal.

- Keine typisierte Standardkonstanten

? Man kann keine einfachen Konstanten mit Typangaben deklarieren.

! Lassen Sie die Typangabe weg, der Compiler erkennt den richtigen Typ schon selbst.

- Keine VAR Taglisten für typen unter 4 Bytes

? Elemente, auf die Zeiger in einer Tagliste zeigen, müssen mindestens 4 Bytes groß sein.

- Keine Variable

? Wahrscheinlich haben Sie versucht eine Konstante zu verändern, oder einem Typen oder einer Prozedur etwas zuzuweisen.

- KilledLabels nicht leer

? Compilerfehler.

- Konstante außerhalb des Bereichs

? Sie haben eine für diese Stelle zu große Konstante verwendet.

- Konstante erwartet
- Konstanten Ausdruck erwartet

? Sie haben an dieser Stelle eine Variable oder etwas anderes angegeben, an der der Compiler eine Konstante erwartet.

- Konstante `'TYPE` erwartet

? An einer Stelle, an der ein Typ erwartet wird, hat der Compiler eine Konstante gefunden. Er interpretiert dies so, das als Typ der Typ der Konstante verwendet werden sollte. In diesem Fall muß allerdings ein `'TYPE` an die Konstante gehängt werden.

! Fügen Sie ein `'TYPE` an.

- Konstanter Index auserhalb des Berreichs

? Die Indexgrenzen eines Arrays müssen im Bereich von Integer liegen.

! Verwenden Sie eine dynamische Struktur z. B. eine Liste.

- Konstantes ARRAY erwartet

? Sie haben eine typisierte Konstante vom Typ **ARRAY** definiert, nach der Typ-Angabe aber keine Elemente angegeben.

- Konstantes SET erwartet

? PUSH/POP erwartet ein konstantes Set als Parameter, keine Variable.

- Kontext als zweiter Parameter

? Sie haben bei New den Kontext als ersten Parameter angegeben.

- Label -1 kann nicht gesetzt werden

? Compilerfehler

- Label in falscher Liste

? Compilerfehler

- LDIV nur für LONGINT, LONGCARD
- LMUL nur für INTEGER, CARDINAL

? Sie haben für die obengenannten Funktionen einen anderen Typen als den in der Meldung genannten verwendet.

- LEA erwartet Adressregister als zweiten Parameter

? Fehlermeldung des Assemblers

- LIST Parameter nur als letzten Parameter

? Eine **LIST OF** Parameterliste ist nur als letzter Parameter einer Prozedurdeinition möglich.

- LONGREALs nur in FPU-Registern

? Will man eine LONGREAL-Variable in ein Register legen, kann man dafür nur FPU-Register verwenden. Vorausgesetzt man besitzt eine FPU.

- Mathematische Funktionen nur für REAL-Zahlen

? Sie haben eine mathematische Funktion wie z. B. `SIN`, `EXP` auf eine Variable angewendet, die nicht vom Typ `REAL` oder `LONGREAL` war.

- Methoden nur für eigene Typen

? Sie können nur Methoden zu Typen definieren, die auch in Ihrem Modul definiert sind, nicht zu importierten Typen .

- Methodenredefinition nicht angegeben

? Sie haben versucht die Methode eines Objektes zu redefinieren, ohne das diese in der Objektdefinition angegeben wurde.

- Mnemonic oder Label erwartet

? Der Assembler erwartet entweder einen Befehl oder eine Sprungmarke, traf jedoch auf etwas anderes.

- MOD nur für ganze Zahlen

? Sie haben den Operatobf MOD auf eine Realzahl angewandt.

! Führen Sie, wenn es wirklich eine Real-Variable sein soll, vorher eine Typumwandlung durch.

- modStackPtr#0 bei DecModEnd, Compiler Fehler
- modStackPtr#0 nach Import, Compiler Fehler
- modStackPtr=0 bei DecLEnd, Compiler Fehler
- modStackPtr=0 bei DecModEnd, Compiler Fehler

? Compilerfehler

- MODUL-Bezeichner am Anfang und Ende verschieden

? Der Name am Modulende unterscheidet sich von dem im Modulkopf

- ModulaII : nur Records in WITH

? Sie haben versucht eine andere Variable als einen Record zu WITHen, obwohl sie den Modula 2 Modus eingeschaltet haben. In Modula 2 ist WITH nur für Records definiert.

- MODULE erwartet

? Sie haben am Modulanfang das Schlüsselwort **MODUL** vergessen.

- NEW vorerst nur bei Objekten

? NEW als Standardprozedur ist zur Zeit nur für Objekte definiert.

- Nicht genau definierte Konstante

? Sie verwenden ein Element eines überladenen Aufzählungstypen und der Compiler kann nicht genau ermitteln, zu welchem Aufzählungstyp das Element gehört.

! Qualifizieren Sie das Element über den Namen des Aufzählungstypen, zu dem es gehört.

- Nicht genug Speicher

? Der Compiler, Linker oder das Make benötigen für den Import von Modulen eine große Menge Speicher. So kann es passieren, daß einem bei der Übersetzung von Modulen, die viele andere Module importieren, der Speicher ausgeht, vor allem wenn man nur 2 MB Speicher besitzt.

! Benutzen Sie den ARexx-Compiler von der Shell aus, beenden Sie alle anderen Programme, die Sie nicht unbedingt benötigen, oder kaufen Sie sich eine Speichererweiterung.

- Nicht genug UNLK für LINKs

? Assembler: Die Zahl der UNLKs muß mit der der LINKs übereinstimmen.

- Nicht implementierte Konversion

? Sie haben versucht einen Typen in einen anderen zu konvertieren, für den eine solche Konversion nicht definiert ist.

! Kontrollieren Sie Deklaration und Schreibweise.

- Noch nicht implementiert

? Fehlermeldung des Assemblers. Derzeit sind nocht keine Supervisorbefehle implementiert.

- Nur 16 Register vorhanden

? Sie haben eine höhere Registernummer als 15 angegeben (0-15).

- Nur einfache Typen für Regvars

? Sie haben versucht eine komplexe Variable (Record etc.) bei der Variablendefinition in ein Register zu legen, dies ist jedoch im Gegensatz zur WITH-Anweisung nicht möglich.

- Nur feste Register erlaubt

? Sie haben als Registernummer eine Variable statt einer Konstanten verwendet. Dies ist nicht möglich, da Cluster eine Compilersprache ist und kein Interpreter.

- Nur globale Prozeduren möglich

? Sie können nur globale Prozeduren an Variablen zuweisen.

- Nur IF KEY in Recorddefinition möglich

? Sie haben wahrscheinlich bei der Definition eines varianten Records „**IF**“ vergessen oder als ehemaliger Modulaprogrammierer „`case`“ geschrieben.

- Nur komplexe Typen Forward-deklarierbar

? Einfache Typen können nicht forward deklariert werden.

- Nur Konstante als Schrittwert erlaubt

? Sie haben bei einer FOR-Schleife eine Variable als Schrittwert verwendet, was nicht möglich ist.

! Falls ein variabler Schrittwert notwendig ist, verwendet Sie eine WHILE-Schleife.

- Nur POINTER als HIDDEN-Typen erlaubt !

? Sie haben versucht einen anderen Typen als einen Pointertypen als Hiddden-Typen zu definieren.

- Nur Records oder Pointer auf Records für Methoden

? Der erste Parameter einer Methode muß Record oder ein Zeiger auf einen solchen sein.

- Nur Strings an STRING übergebbar.

? Sie haben versucht eine Variable an einen Parameter vom Typ STRING zu übergeben, welche kein String ist.

- Nur Wortbreite möglich

? Der Assembler erlaubt hier nur Wortbreite. Sie wollten Byte- oder Langwortbreite.

- Nur Zeiger an ANYPTR übergeben

? Sie können nur Zeiger an ANYPTR übergeben.

- Nur Zeigertypen für generische Module

? Der generische Parameter eines generischen Moduls muß ein Zeiger sein.

- ODD nur für numerische Typen

? ODD kann nur bei zählbaren Typen angewandt werden

- OF erwartet

? Sie haben bei einem **IF KEY** das erste **OF** vergessen oder bei einer ARRAY-Definition das **OF** vergessen.

- OF oder „,“ erwartet

? Sie haben in einer ARRAY-Deklaration ein folgendes OF oder „,“ vergessen.

- OF erwartet in offenem ARRAY

? Sie müssen bei der Definition eines offenen Array ein **OF** vor dem Typen schreiben.

- Offenes ARRAY erwartet

? Der Compiler erwartet an dieser Stelle ein offenes Array.

- Operand nur für BOOLEAN definiert

? Sie haben einen boolschen Operator auf einen anderen Typen als auf BOOLEAN angewandt.

! Meist hat man nur die Klammern vergessen, wenn man zwei Boolausdrücke mit AND oder OR verbinden wollte.

- OpenLabels nicht leer

? Compilerfehler

! Meist hat man nur die Klammern vergessen, wenn man zwei Boolausdrücke mit AND oder OR verbinden wollte.

- Parametername nicht bekannt oder an falscher Stelle

? Bei dem Versuch einer Prozedur einen Parameter durch Angabe des Parameternamens zu übergeben haben Sie diesen entweder falsch geschrieben, an der falschen Stelle angegeben, oder er existiert gar nicht.

! Kontrollieren Sie die Prozedurdefinition.

- Parent nicht gefunden

? Beim qualifiziertem Erben wurde der Vorfahre nicht gefunden. Meist ein Schreibfehler, kontrollieren Sie die Objektdefinition.

- POINTER erwartet

? An dieser Stelle erwartet der Compiler einen Zeiger und keine einfache Variable.

- Positionaler nach benanntem Parameter

? Nachdem Sie den ersten Parameter eines Prozeduraufrufs über seinen Namen angesprochen haben, müssen Sie auch alle anderen dieser Prozedur, die Sie noch übergeben wollen, über den Namen ansprechen. Es ist nicht möglich, wieder in eine positionale Übergabe überzugehen.

• PosJumps nicht leer

? Compilerfehler

• Potenzierung nur für REAL

? Der Operator „^“ ist nur für REAL-Zahlen definiert.

• PROCEDURE erwartet

? Laut Sprachdefinition erwartet der Compiler an dieser Stelle das Schlüsselwort **PROCEDURE**.

• Prozedur zu komplex

? Sie haben die Maximalgröße einer Prozedur überschritten. Eigentlich fast unmöglich. Man schreibt keine so großen Prozeduren.

! Zerlegen Sie die Prozedur in mehrere kleinere.

• Quelle oder Ziel muß Register sein

? Fehlermeldung des Assemblers.

• Record erwartet

? Wahrscheinlich haben Sie versucht eine normale Variable zu qualifizieren.

• Record zu groß, maximal 32KBytes

? Sie haben die maximale Größe eines Records überschritten.

• Register bei MUL/... als zweiten Parameter

? Fehlermeldung des Assemblers

• Register bereits vergeben

? Sie haben zweimal das selbe Register innerhalb einer Prozedur belegt.

• Register beschmutzt, das nicht existiert

• Register einmal zuoft unused

• Register nicht ordentlich freigegeben

• Register nicht wieder ordentlich ge-unused

? Compilerfehler

• Register erwartet

• Registernummer erwartet

? Sie haben nach einem Variablennamen zwar ein **IN** geschrieben, jedoch folgte keine Registernummer

• Schalter existiert bereits

? Sie haben einen Schalter definiert, dessen Namen schon von einem anderen Schalter belegt ist.

! Verwenden Sie einen anderen Namen. Prüfen Sie, ob der schon existierende nicht schon die Aufgabe erfüllt.

• SELF nicht gefunden

? Sie haben in einer Nicht-Objektmethode versucht auf SELF zuzugreifen, was natürlich nicht geht.

• SET erwartet

? Sie haben einen anderen Variablentypen wie einen SET behandelt.

• SetLabels nicht leer

? Compilerfehler

- Settyp hat mehr als 32 Elemente

? Um das Sethandling möglichst schnell zu machen, werden Setelemente als einzelne Bits in einem Langwort dargestellt. Da diese genau 32 Bit lang sind, kann ein Set auch höchstens diese Anzahl an Elementen haben.

! Verwenden Sie ein BOOLEAN-Array.

- SHL/SHR erwarten numerischen Parameter als zweiten Parameter

? Sie haben keinen numerischen Parameter als zweiten Parameter an SHL/SHR übergeben.

- Skalare Konstante erwartet
- Skalarer Typ erwartet

? Sie haben einen nichtskalaren (nicht zählbaren) Typen verwendet.

- Sonstwas erwartet in Ausdruck

? Das verwendete Symbol ist an dieser Stelle nicht erlaubt.

- Sorry Register A0 und A1 schon vergeben

? TRY..EXCEPT belegt die Register A0 und A1.

- Sorry akuter Registermangel
- Sorry Register schon belegt

? Sie haben versucht mehr Variablen in Register zu packen als Register vorhanden sind.

! Werfen Sie alle Variablen, die Sie nicht unbedingt in Registern brauchen aus diesen raus.

- Sorry, D0 unwiederbringlich vergeben
- Sorry, D1 unwiederbringlich vergeben

? Sie haben eine Variable in D0/D1 gelegt und darauf eine Funktion oder eine Prozedur aufgerufen, die diese Register zur Parameterübergabe benötigt (Funktionen geben ihr Ergebnis immer in D0 zurück).

! Legen Sie die Variable in ein anderes Register oder verzichten Sie an dieser Stelle auf Registervariablen.

- Sorry, nur einfache Variablen als Werteparameter an Register

? Sie haben versucht eine komplexe Variable an einen Register-Werte-Parameter zu übergeben.

! Übergeben Sie komplexe Typen als VAR/REF-Parameter oder mittels eines Zeigers.

- Sorry, VAR-Parameter nur in Adressregistern.

? Da ein VAR-Parameter im Grunde nichts anderes als eine Übergabe durch einen Zeiger ist, muß man ihn, will man ihn in ein Register legen, in ein Adreßregister legen.

- Stackdifferenzen

? Am Ende eines Assmblerteils stimmte der Stack nicht mit dem zu Begin überein.

! Bitte mit dem STACK-Befehl korrigieren.

- Stapelübelauf

? Sie haben die maximale Schachtelungstiefe für Compilerswitches von 16 überschritten.

- Statische Variablen nur in Prozeduren

? Sie haben versucht eine globale Variable als **STATIC** zu definieren, was unsinnig ist, und deshalb vom Compiler unterbunden wird.

- String erwartet

? Die Meldung dürfte alles aussagen.

- String oder Nummer für Exception erwartet

? Bei einer Exceptiondefinition folgte nach dem Doppelpunkt weder ein String, noch eine Nummer wie es die Sprachdefinition verlangt.

- SUPER nicht gefunden (Compiler Fehler)

? Compilerfehler

- SUPER.method erwartet

? Sie haben innerhalb einer Methode den Bezeichner SUPER aufgerufen, ohne eine Methode anzugeben, was natürlich keinen Sinn macht.

- Symbol nicht in Def erlaubt

? In Ihrem Defintionsmodul haben Sie ein Symbol benutzt, daß laut Sprachdefintion nicht erlaubt ist, wie z. B. **CLOSE**, **BEGIN**, etc..

- Symbolfile ist aus der Zukunft

? Aus irgendeinem Grund hat ein Symbolfile ein Datum, das später ist, als das ihrer System-Uhr.

! Kontrollieren Sie das Datum Ihrer internen Uhr, ob es dem heutigen Datum entspricht.

- Symfile konnte nicht geöffnet werden

? Der Compiler konnte ein Symbolfile eines importierten Moduls nicht finden.

! Kontrollieren Sie, ob das Modul in Ihrem aktuellen Projektverzeichnis liegt. Wenn nicht, tragen Sie bitte das entsprechende Verzeichnis in den Modulsuchpfad ein. Findet der Compiler es dann immer noch nicht, kontrollieren Sie bitte, ob Sie das dazugehörige Definitionsmodul schon übersetzt haben.

- TAG-Typ erwartet
- TAGListe erwartet

? An dieser Stelle wird eine Tagliste erwartet.

- THEN, AND_IF erwartet

? Nach der Bedingung in einer IF-Anweisung fehlt ein **THEN** oder **AND_IF**.

- TO erwartet

? Sie haben nach einer FOR-Anweisung das **TO** vergessen.

- TRUE,FALSE oder OLD erwartet

? Nach einem `$$Switch:=` folgte etwas anderes als `TRUE`, `FALSE` oder `OLD`.

- Typ erwartet
- Typen erwartet

? Laut Sprachdefinition müßte an dieser Stelle ein Typ stehen. Eine genauere Angabe ist leider nicht möglich, da dieser Fehler sehr viele verschiedene Ursachen haben kann.

- Typ nicht adressierbar

? Sie haben versucht die Adresse eines Typs zu ermitteln, was logischerweise nicht geht.

- Typecheck nur für Objekte

? Sie haben versucht einen Typvergleich mit **IS** mit einem einfachen Typen durchzuführen, dies ist jedoch nur mit Objekten möglich.

- Unbekannte Varadrtypes
- Unbekannter decType
- Unbekannter Konstantaddresstyp

- Unbekannter offener Typ als Werteparameter (Compiler-Fehler)

? Compilerfehler

- Unbekanntes Attribut

? Sie haben ein anderes Attribut als `'PTR`, `'ADR`, `'SIZE`, `'RANGE` `'MAX`, `'MIN` verwendet.

- Unbekannte TAG-Id
- Unbekannter TAG Wert

? Sie haben einen Tag verwendet, der nicht definiert ist.

- Unbekanntes Klassenelement

? Sie haben auf ein Element eines Objekt-Records zugeriffen, das nicht in der Klasse definiert ist.

! Kontrollieren Sie die Klassendefinition.

- Unbekanntes Mnemonic

? Sie haben einen Assemblerbefehl verwendet, der nicht definiert ist.

- Unbekanntes Symbol

? Sie haben ein Sonderzeichen verwendet, das nicht in der Sprachbeschreibung enthalten ist.

- Unbekanntes Symbol in Switch

? Sie haben einem Switch etwas anderes als `TRUE, FALSE, OLD` oder einen anderen Switch zugewiesen. Dies ist nicht erlaubt.

- Undefinierter Modul-Namen

? Der Compiler traf auf ein lokales Implementationsmodul, zu dem kein Definitionsmodul existiert.

! Wahrscheinlich ein Schreibfehler, überprüfen Sie den Modulnamen.

- Undefinierter Pointer freigegeben
- Undefiniertes Pointerziel

? Compilerfehler

- Undefinierter Vergleich

? Sie haben versucht Typen zu vergleichen, für die kein Vergleich definiert ist z. B. char und Longreal.

- Undefiniertes label

? Sie haben versucht ein Assemblerlabel anzuspringen, das nicht definiert ist.

- Unerlaubte Benutzung einfacher Pointer für Klassen

? Sie haben über einen einfachen Zeiger auf spezielle Eigenschaften einer Klasse zugegriffen, was nicht möglich ist.

! Verwenden Sie einen **CLASSPTR**.

- Unerwartetes Dateiende

? Der Compiler traf beim Übersetzen unerwartet auf ein Dateiende, da ein Modulende fehlte.

- Ungültiger IF-KEY-Index
- Ungültiger CaseIndex

? Der Compiler erwartet nach einem IF KEY in einer varianten Recorddefinition entweder einen Bezeichner oder einen Doppelpunkt.

- Unklare Mengenkonstante

? Der Compiler konnte den Typ einer Mengenkonstanten nicht eindeutig feststellen, dies passiert z. B. bei Vergleichen von Mengen.

! Bitte geben Sie den Typ der Menge mit an.

- Unmögliche Adressierung

? Sie haben wahrscheinlich versucht die Adresse einer Registervariablen zu ermitteln, was nicht möglich ist.

- Unmögliche Adressierung für EOR
- Unmögliche Adressieung für ABCD
- Unmögliche Adressieung für ADDX
- Unmögliche Adressieung für ADDX
- Unmögliche Adressieung für SBCD
- Unmögliche Operanden bei LINK
- Unmögliche Parameter bei EXT

? Fehlermeldungen des Assemblers, der Grund geht aus der Meldung hervor.

- Unmögliche Konstante

? Sie haben versucht eine Konstante eines Typs zu definieren, der für diesen Zweck ungeeignet ist, z. B. ein offener Record.

- Unmögliche Konstante in FOR

? Sie haben in einer FOR-Schleifendefinition eine Konstante verwendet, die keine Ganzzahl ist. Dies ist nicht möglich.

- Unmögliche Registervariable ???

? Sie haben eine Variable mit nicht zulässigem Typ in ein Register gelegt.

! Lesen Sie in Kapitel 6 nach, welche Typen dazu geeignet sind.

- Unmögliche Standardlänge

? Compilerfehler

- Unmöglicher Assembler String

? Eine Assemblerstringkonstante darf maximal 4 Zeichen lang sein.

- Unmöglicher Pointer für `+^`
- Unmöglicher Pointer für `-^`

? Den Postincrement/Predecrement-Mode kann man nur bei Registervariablen anwenden.

! Legen Sie den Pointer in ein Register.

- Unmöglicher Typ für EVEN
- Unmöglicher Typ für INC/DEC
- Unmöglicher Typ für SHL/SHR

? Für diese Standardprozeduren/Operatoren dürfen nur ganzahlige Typen verwendet werden, INC/DEC auch für Aufzählungstypen und Pointer, SHL/SHR auch für Sets.

- Unmöglicher Typ in ComAddAcc
- Unmöglicher Typ in ComIndex

? Compilerfehler

- Unterschiedliche Bezeichner an Ende und Anfang

? Eine Prozedur/Modul hat am Ende einen anderen Bezeichner als am Anfang.

! Kontrollieren Sie die Schreibweise der Namen.

- UNTIL bei REPEAT erwartet

? Der Compiler trifft nach einem **REPEAT** auf ein anderes Symbol als das erwartete **UNTIL**; z. B. ein überzähliges **END**.

- Ursprüngliche Definition hatte anderen Rückgabetyp
- Ursprüngliche Definition hatte anderen Typ
- Ursprüngliche Definition hatte anderes Register
- Ursprüngliche Definition hatte keine gleichen Parameter hier

- Ursprüngliche Definition hatte keine Registerparameter
- Ursprüngliche Definition hatte keinen Rückgabewert
- Ursprüngliche Definition hatte mehr Parameter
- Ursprüngliche Definition hatte Registerparameter
- Ursprüngliche Definition hatte weniger Parameter
- Ursprüngliche Definition hatte Rückgabewert
- Ursprüngliche Definition hatte anderen Namen
- Ursprüngliche Definition war kein VAR-Parameter
- Ursprüngliche Definition war VAR-Parameter

? Alle oben aufgeführten Fehlermeldungen sind darauf zurückzuführen, daß sich die endgültige Prozedurdefinition von der Forward-Deklaration bzw. der Deklaration im Definitionsmodul unterscheidet. Worin, entnehmen Sie bitte der Fehlermeldung

- Variable `'TYPE` erwartet

? An einer Stelle, an der ein Typ erwartet wird, hat der Compiler eine Variable gefunden. Er interpretiert dies so, daß als Typ der Typ der Variablen verwendet werden sollte. In diesem Fall muß allerdings ein `'TYPE` an die Variable gehängt werden.

! Fügen Sie ein `'TYPE` an.

- Variable überschreitet Limit, benutze Zeiger !!!

? Sie haben eine Variable größer 30KB definiert. Dies ist nicht möglich.

! Bitte verwenden Sie einen Zeiger und allozieren den Speicher während der Laufzeit.

- Variablen Parameter erwartet

? Sie haben eine Konstante an einen VAR-Parameter übergeben.

! Verwenden Sie einen REF-Parameter.

- Vergleich einer statischen Prozedur mit NIL

? Sie haben eine statische Prozedur mit NIL verglichen. Da dieser Vergleich immer FALSE ergibt ist er unsinnig und deutet auf einen Fehler hin, daher reklamiert der Compiler bei einem solchen Vergleich, auch wenn er nach der Sprachdefinition möglich wäre.

- Verlorener DecType (Compilerfehler)

? Comilerfehler

- Verschiedene Typen als VAR-Parameter

? Bei der Übergabe an VAR-Parameter müssen die Übergabewerte von selben Typ des VAR-Parameters sein. Betrifft im Normalfall zuweisungskompatible Typen.

! Führen Sie eine Typkonversion durch.

- Verschiedene Typen in LDIV/LMUL

? Diesen Funktionen können jeweils nur die gleiche Typen übergeben werden, also INTEGER & LONGINT oder CARDINAL & LONGCARD.

- Versionskonflikt

? Es wurde ein Modul entdeckt, dessen Defintionsteil nach einem Modul compiliert wurde, von dem es importiert wird.

! Führen Sie am besten ein Make durch.

- Wer wird denn den SP poppen

? Compilerfehler

- WHILE erwartet END

? Sie haben nach einer WHILE-Schleife das **END** vergessen.

- WITH Killer aufdreckige Register angewendet

? Compilerfehler

- WITH verlangt Variable oder Typen

? Nach einem **WITH** muß ein e Variable oder ein Typ folgen.

- WITH-Wert verloren, weis der Teufel warum

? Compilerfehler

- Zahlenbasis zu groß

? Bei der Zahleneingabe zu einer beliebigen Basis in der Form `Basis:Wert`, darf die Basis nicht größer als 16 sein.

- Zeiger erwartet
- Zeigervariable erwartet

? Sie haben versucht eine Variable zu dereferenzieren, die kein Zeiger ist.

- Ziffer ist größer als Basis

? Wenn Sie eine Zahl in er Form `Basis:Wert` angeben, darf die eine Ziffer des Wertes nicht größer sein als die Basis-1. Z. B. `4:32` falsch wäre `4:46`.

- Zuviele Elemente in Konstante

? Bei der Definition eines konstanten Records/Arrays wurden mehr Elemente angegeben, als in der Typdefinition definiert sind.

- Zuviele Module importiert

? Sie haben es fertiggebracht, mehr als 100 Module in einem Modul zu importieren. Dies ist durch den Aufbau des Compilers nicht möglich, sollte normalerweise aber auch nicht vorkommen.

! Wenn es denn unbedingt nötig ist, dann zerlegen Sie das Modul in mehrere kleinere. Die Gesammtzahl der an einem Programm beteiligten Module ist nicht limitiert.

- Zuviele Schalter

? Compilerfehler

- Zuviele UNLK für LINKs

? Die Zahl der verwendeten UNLks muß mit der der LINKs übereinstimmen.

- Zwei Operanden erwartet

? Assembler: Sie haben einen Befehl, der zwei Argumente erwartet mit nur einem verwendet.

- Zweiter Operand bei CMP muß Register sein.

? Assembler: Die Meldung erklärt sich selbst.

- Zweiter Operand bei SHL muß Ganzzahl sein
- Zweiter Operand bei SHR muß Ganzzahl sein

? Bei SHL/SHR muß der Betrag, um den geschoben werden soll eine Ganzzahl sein, also INTEGER oder CARDINAL.

B.3 Fehlermeldungen des Linkers

- obj ist kein Objectfile

? Die Objektdatei eines Moduls hat einen Formatfehler.

! Übersetzen Sie das entsprechende Modul noch einmal.

- obj nicht gefunden

? Objektdatei zu einer existierenden Symboldatei konnte nicht eingebunden werden, da es nicht gefunden wurde.

! Übersetzen Sie das entsprechende Modul.

- Nicht genug Speicher

? Der Linker benötigt bei großen Programmen viel Speicher.

! Benützen Sie den Rexx-Comiler von der Shell aus.

- Nicht implementiert im eigenen Modul
- Nicht implementiertes Objekt

? Ein Objekt, das in einem Modul definiert wurde, ist nicht implementiert worden. Dabei gibt im eigenen Modul an, das ein Objekt im Hauptmodul nicht implementiert wurde, ansonsten ist es ein Objekt, das importiert wurde.

! Überprüfen Sie das entsprechende Modul.

- Welches Programm ???

? Sie befinden sich nicht in einem Hauptmodul. Daher will der Linker wissen, welches Programm er linken soll.

B.4 Fehlermeldungen des Loaders

Zum einen kann der Loader alle Fehlermeldungen des Linkers auslösen. Zudem gehören zu seinen Meldungen auch alle Laufzeitfehler, die beim Start eines Programmes auftreten. Da zu denn Laufzeitfehlern auch alle nicht abgefangenen Exceptions gehören, wäre es vergeblich hier eine vollständige Auflistung aller möglichen Laufzeitfehler geben zu wollen. Da bei jedem Run-Time-Error nicht nur der Fehler sondern auch das Modul in dem er ausgelöst wurde angegeben ist, können Sie in der Beschreibung der einzelnen Module die Bedeutung der einzelnen Exceptions nachlesen. Ist eine Exception nicht in diesem Modul definiert, dann finden Sie sie warscheinlich im Modul Exceptions.

Die Laufzeitfehler, die von Cluster ausgelöst werden können, aber nicht in Exceptions erklärt sind sind hier aufgelistet:

- `Function_No_Return`

? Innerhalb einer Funktion fehlt die Rückgabe eines Wertes mittels **RETURN**.

- `Local_Proc_Var`

? Die Adresse einer lokalen Prozedur wurde in eine Variable gespeichert, dies ist nicht zulässig.

- `NIL_Dereferenced`

? Ein Zeiger mit nicht validem Inhalt wurde dereferenziert.

- UserBreak

? Der Benutzer hat Ctrl+C gedrückt und so das Programm abgebrochen.

B.5 Fehlermeldungen des Make

- DEF ohne MOD

? Das Make konnte das Implementationsmodul zu einem Definitionsmodul nicht finden oder laden. Letzteres kann an einem zu kleinem Textspeicher liegen.

- Error in Make

? Während des Makes trat in einem Modul ein Fehler auf. Das betroffene Modul wird automatisch geladen.

! Korrigieren Sie das Modul und starten Sie das Make neu.

- Make File not found

? Keine Makedatei vorhanden.

! Wählen Sie zuerst Create-Make aus.

- No Cluster format

? Einer der zu übersetzenden Texte liegt nicht im Clusterformat vor bzw. es fehlt der Info-Header vor dem ASCII-Text.

! Erzeugen Sie einen Info-Header oder speichern Sie den Text im Clusterformat.

- Source not found

? Quell-Text eines Moduls wurde nicht gefunden.

- Welches Programm ???

? Das Modul, von dem Sie das Make starten ist kein Hauptmodul, Sie haben noch kein Standardprojekt gesetzt und der Text hat auch keinen Projekteintrag. Daher will das Make wissen, welches Modul als Hauptmodul bearbeitet werden soll.

- Zyklische Imports

? An diesem Programm sind Module beteiligt, die sich im Kreis importieren, was nicht möglich ist.

! Vermeiden Sie Ringimporte und verwenden Sie, wenn es notwendig ist, **DEFFERED POINTER**.

Anhang C

Einbinden fremder Module:

Eine externe Prozedur wird folgendermaßen definiert:

```
PROCEDURE Foo(x IN D2 : INTEGER):INTEGER EXTERN _foo;
```

Die Prozedur ist dann im Clustersymbolraum unter dem Namen Foo erreichbar und kann wie jede andere Prozedur aufgerufen werdem. In der externen Referenz erscheint er unter dem namen „_foo”.“

Aufgrund der unterschiedlichen Aufrufssyntax zwischen „C“ und „Cluster“ ist es nicht möglich ohne Interfacecode externe Routinen mit Stackvariablen zu benutzen. Ein mögliches Interface für die „C“ Funktion „foo’”:

```
int foo(Window * win,int x,int y,char c)
{...
}
```

könnte so aussehen:

```
PROCEDURE CFoo EXTERN _foo;   | Stummelrest

PROCEDURE Foo(win : WindowPtr;x,y : LONGINT;c : CHAR):LONGINT;
```

```
BEGIN
  ASSEMBLE(MOVE.L   #0,D0      | Argumente in C-Konvention auf
           MOVE.B   c,D0       | den Stack packen
           MOVE.L   D0,-(A7)
           MOVE.L   y,-(A7)
           MOVE.L   x,-(A7)
           MOVE.L   win,-(A7)

           JSR      CFoo       | C-Funktion aufrufen

           ADD.L    #4*4,A7     | Stack wieder korrigieren
          );
END Foo;
```

Wird ein Programm gelinkt, in dem sich eine Referenz auf ein externes Modul befindet, erzeugt der Linker kein Programmodul, sondern ein Amiga Standard Objektmodul. Dieses muß dann noch mit einem Standardlinker „ALink“ oder „BLink“ mit den externen Objektmodulen gebunden werden.

Anhang D

Erzeugen von Libraries:

Eine Library kann nur aus Implementationsmodulen (und natürlich einem Hauptmodul) zusammengesetzt werden, die als Library übersetzt werden. Dies kann entweder durch den Schalter in der **EU** oder durch die Compilerschalter `$$Library:=TRUE` oder `$$LibraryAlso:=TRUE` erreicht werden[2].

Eine Library besteht aus drei Teilen: Den Librarydaten in der Librarybase, einer Sprungtabelle auf die Funktionen der Library und die Routinen der Library selbst. Alle globalen Variablen, die in einer Clusterlibrary benutzt werden, liegen, nach den öffentlichen Feldern, in der Librarybase.

Wird eine Library in den Hauptspeicher eingebracht, so muß sie durch das Betriebssystem initialisiert werden. Dies geschieht, indem eine Init-Vector im Residentteil der Library angesprungen wird. In Cluster führt dies zur Ausführung der BEGIN-Teile aller beteiligten Implementationsmodule. Somit bleibt die Initialisierung analog zu der normaler Module.

Soll eine Library, die nicht mehr benötigt wird, aus dem Speicher entfernt werden, so ruft das System den Expunge-Vector der Library auf. Dies wird in Cluster auf die Kette der CLOSE-Teile aller beteiligten Module umgebogen, so daß auch die Deinitialisierung analog zu bestehenden normalen Modulen bleibt.

[2] letzerer Schalter erzeugt sowohl ein Library als auch ein normales Objektmodul

Dies macht es möglich, das viele Module ohne Änderung sowohl in Programmen als auch in Libraries verwendet werden können.

Das Hauptmodul einer Library muß einem vorgegebenen Rahmen folgen:

```
$$Library:=TRUE
MODULE DummyLibrary;

FROM System      IMPORT BITSET,SHORTSET,Regs,LONGSET,EndBEGIN,
                        OwnLibBase,SysStringPtr,PROC,CloseProc;

TYPE
  |
  | Dies ist der Typ der Library Sprungtabelle. Die Einträge 0 bis 3
  | sind durch Systemfunktionen belegt. Der letzte Eintrag enthält -1
  | um das Ende der Tabelle anzuzeigen. Werden neue Funktionen in
  | die Library aufgenommen, muß
  | dieser Tabellentyp entsprechend erweitert werden.
  |
  TableType = ARRAY [0..4] OF ANYPTR;

CONST
  |
  | Name der Library, under dem sie auch im Libs: erscheinen muß.
  | Dies muß die erste Konstante im Hauptmodul sein. Es dürfen
  | vorher keine Konstanten deklariert werden. Auch die nächsten
  | beiden Konstanten "IdString" und "Table"
  | müssen genau diesen Platz einnehmen
  |
  Name        = "dummy.library";
  |
  | Identifikationsstring der Library, er enthält die
  | Versionsinformation in lesbarer Form, so daß die Library auch
  | über "Version dummy.library file" untersucht werden kann,
  | falls sich noch eine andere Version im System befinden sollte.
  |
  IdString    = "$VER: dummy.library $$.$$$ (© 1993 StoneWare)";
  |
```



```
  | Forward deklaration der Sprungtabelle, die völlig ausgefüllte
  | Version kann erst am Ende des Modules stehen.
  |
  Table       = TableType;


|
| Systemeinsprung zum öffnen einer Library. Diese Routine wird
| angesprungen, wenn ein Programm "OpenLibrary" aufruft.
|
PROCEDURE Open;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6});  | Sichern aller Register
  SETREG(REG(A6),A4);            | Librarybase als Lager der
                                 | globalen Variablen

  INC(OwnLibBase.openCnt);       | Anzahl der Öffnungen erhöhen
  EXCL(OwnLibBase.flags,3);      | Expunge flag löschen

  |
  | Platz für eigene initialisierungen
  |

  SETREG(REG(A4),D0);            | Librarybase als Rückgabewert
  POP(RegSet:{D2..D7,A2..A6});   | Rückholen der Register
END Open;

|
| Systemeinsprung zum Schliessen der Library. Diese Funktion wird
| aufgerufen, wenn ein Programm "CloseLibrary" aufruft.
|
PROCEDURE Close;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6});  | Sichern der Register
  SETREG(REG(A6),A4);            | Librarybase als Lager der
                                 | globalen Variablen

  |
```

```
  | Hier kann etwas Abbaucode stehen
  |

  DEC(OwnLibBase.openCnt);       | Anzahl der Benutzer vermindern
  IF = AND (3 IN OwnLibBase.flags) THEN | Soll expunged werden ??
    ASSEMBLE(MOVE.L CloseProc,A0    | Wenn ja, dann den Expunge
             JMP    (A0));          | Einsprung aus dem
                                    | Laufzeitsystem anspringen.
  END;
  SETREG(0,D0);                  | Sonst NULL zurückgeben,
                                 | um zu zeigen, daß
                                 | die Library im System verbleibt.

  POP(RegSet:{D2..D7,A2..A6});   | Rückholen der Register
END Close;

|
| Systemvektor um eine Library dazu aufzufordern, sich aus dem System
| zu entfernen. Diese Routine wird entweder bei Speichermangel oder
| durch die Funktion "RemLibrary" aufgerufen.
|
PROCEDURE Expunge;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6});  | Register sichern
  SETREG(REG(A6),A4);            | Librarybase enthält globale Variablen

  IF OwnLibBase.openCnt=0 THEN   | gibt es noch Benutzer der Library
    ASSEMBLE(MOVE.L CloseProc,A0 | nein, dann den Expunge Einsprung
                                 | aus dem
             JMP    (A0));       | Laufzeitsystem anspringen
  ELSE
    INCL(OwnLibBase.flags,3);    | Sonst das Flag für deferred expunge
  END;                           | setzen

  SETREG(0,D0);                  | und NULL zurückgeben, um zu zeigen,
                                 | daß die Library im System verbleibt
  POP(RegSet:{D2..D7,A2..A6});   | Rückholen der Register
END Expunge;
```



```
|
| Reservierte Funktion, bisher noch nicht genutzt
|
PROCEDURE Reserved;
BEGIN
  SETREG(0,D0);                  | muß hier sein für
                                 | zukünftige Erweiterungen
END Reserved;

|
| Dieser Bereich ist für eigene Funktionen der Library
|



|
| Diese Tabelle enthält Zeiger auf alle Funktionen der Library.
| Ihre Größe wird bereits oben festgelegt, da sie für die
| aufgeschobene Definition benötigt wird.
|
| Die ersten vier Einträge sind die Systemfunktionen, danach
| kommen private bzw. öffentliche Funktionen der Library.
| Den Schluss bildet eine -1 als Terminationssymbol.
|
CONST
  Table  = TableType:(Open,Close,Expunge,Reserved,

                      PROC(-1));

BEGIN
  |
  | Initcode der Library.
  |
  ASSEMBLE(JMP  EndBEGIN) | Zeichen, daß die Library
                          | resident verbleibt.
CLOSE
  |
  | Exitcode der Library.
```

```
  |
END DummyLibrary.
```

Prozeduren, die als Libraryfunktionen für andere Programme verfügbar sein sollen, müssen ebenfalls einem gewissen Format folgen. Dies soll hier an einer Prozedur „Foo“ erläutert werden. Die Schnittstelle der Funktion soll nach außen folgendes Aussehen besitzen:

```
LIBRARY DummyBase BY -30 PROCEDURE Foo(rp  IN A0 : RastPortPtr;
                                       x   IN D0 : INTEGER;
                                       y   IN D1 : INTEGER):LONGINT;
```

Die Implementierung könnte nun folgendermaßen aussehen:

```
PROCEDURE Foo(rp  IN A0 : RastPortPtr;
              x   IN D0 : INTEGER;
              y   IN D1 : INTEGER):LONGINT;
VAR bla : LONGINT;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6}); | Register sichern
  SETREG(REG(A6),A4);           | globale Daten sind in der
                                | Librarybase

  |
  | hier ist die Implementierung
  |

  POP(RegSet:{D2..D7,A2..A6});  | Register zurückholen
  RETURN bla;                   | Die Rückgabe muß nach dem POP erscheinen,
                                | und darf keine Berrechnungen mehr
                                | enthalten.
END Foo;
```

Allerdings ist bei diesem Verfahren sehr störend, daß der Compiler versucht, die übergebenen Argumente in Registern zu halten. Dies kann sich bei komplexen Funktionen, die andere Prozeduren aufrufen, zu Problemen und Laufzeiteinbußen führen. Eine andere Technik, die dies vermeidet, ist diese:


```
PROCEDURE Foo;
VAR rp   : RastPortPtr;
    x,y  : INTEGER;
    bla  : LONGINT;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6}); | Register sichern
  SETREG(REG(A6),A4);           | globale Daten sind in der Librarybase

  rp:=RastPortPtr(REG(A0));     | hier werden die Argumente aus den
  x:=REG(D0);                   | Prozessorregistern extrahiert.
  y:=REG(D1);

  |
  | hier ist die Implementierung
  |

  SETREG(bla,D0);               | Rückgabe erfolgt in D0
  POP(RegSet:{D2..D7,A2..A6});  | Register zurückholen
END Foo;
```

Soll die Schnittstelle der Library von den eigentlichen Routinen getrennt werden, so daß in der Schnittstellenroutinen nur ein Aufruf auf die eigentliche Routine vorhanden ist, empfiehlt sich folgendes Verfahren:

```
PROCEDURE Foo;
BEGIN
  PUSH(RegSet:{D2..D7,A2..A6}); | Register sichern
  SETREG(REG(A6),A4);           | globale Daten sind in der Librarybase

  SETREG(I_Foo(RastPortPtr(REG(A0)),REG(D0),REG(D1)),D0);

  POP(RegSet:{D2..D7,A2..A6});  | Register zurückholen
END Foo;
```

Hier wird die eigentliche Routine mit den Argumenten direkt aus den Registern aufgerufen, so daß keine Zwischenvariablen benötigt werden.

Spezielle Clustertypen können nicht immer problemlos an Libraries übergeben werden. Hier zu nennen sind vor allem Clusterstrings und **LIST OF**

Parameter. Strings erscheinen in der Libraryimplementierung als SysStringPtr, **LIST OF** Parameter als **POINTER TO ARRAY OF**...

Beispiel:

```
LIBRARY DummyBase BY -36
  PROCEDURE bar(REF str  IN A0 : STRING;
                    tags IN A1 : LIST OF DummyTags);
```

erscheint in der Implementierung als:

```
TYPE
  DummyTagList    = ARRAY OF DummyTags;
  DummyTagListPtr = POINTER TO DummyTagList;

PROCEDURE bar(str  IN A0 : SysStringPtr;
              tags IN A1 : DummyTagListPtr);
```

Exceptions können auch in Libraries wie gewohnt verwendet werden. Allerdings gibt es keinen äußersten Exceptionhandler, der das Program einfach beenden würde. Wird eine Exception ausgelöst, die keinen Handler findet, so versenkt sich der Rechner in tiefe Meditation über den Sinn seiner Existenz.

In Libraries gelten gewisse Einschränkungen des Clustersprachumfangs. Die hauptsächliche Einschränkung ist das Verbot dynamisch allokierter offener Rückgabetypen. Dies führt dazu, daß in Libraries die Funktion `ALLOC_RESULT` nicht implementiert ist. Auch die Rückgabe von komplexen Typen an einen VAR- oder REF-Parameter ist aus diesem Grund nicht gestattet. Einige Module sind ebenfalls nicht für Libraries verfügbar, ein typisches Beispiel hierfür wäre z. B. InOut.

Anhang E

Sprachdefinition

E.1 Grundlegende Sprachelemente

Die Sprache Cluster basiert auf dem ASCII Zeichenstandard. Verwendet werden die großen und kleinen Buchstaben des Standardalphabets, sowie die üblichen Sonderzeichen. Cluster ist case-sensitive, das heißt es wird zwischen großen und kleinen Buchstaben unterschieden. In der Tradition von Algol, Pascal, C und Modula ist Cluster formatfrei, d. h. zwischen zwei Symbolen der Sprache dürfen beliebig viele Leerzeichen und Zeilenvorschübe enthalten sein.

Cluster ist eine Sprache, die für einen Singlepasscompiler zugeschnitten ist. Dies hat zur Folge, daß mit Ausnahme von Pointerzielen alle Bezeichner vor ihrer Verwendung definiert werden müssen.

Die Sprachdefinition wird in EBNF angegeben.

```
$$ letter      ::=    a|b|c|...X|Y|Z
$$ digit       ::=    0|1|2|3|4|5|6|7|8|9
$$ hexdigit    ::=    digit|A|B|C|D|E|F
```

E.1.1 Bezeichner

Bezeichner bestehen in Cluster aus großen und kleinen Buchstaben, Zahlen und dem Unterstrich „_“. Sie müssen immer mit einem Buchstaben beginnen.

```
$$ ident       ::=    letter{letter|digit|"_"}
```

Beispiele für richtige Bezeichner:

```
Cluster, Text2, Text_2, aus, Ein_Text
```

Falsch wären:

```
2Pi, _LVO, Ein Text
```

Großschreibungen im Wort und der Unterstrich können dazu dienen, die Bezeichner lesbarer zu gestalten.

Manche Bezeichner müssen noch qualifiziert werden:

```
$$ qualident    ::=     {ident.}ident
```

Folgende Bezeichner sind vorbelegt:

```
SHORTCARD       CARDINAL        LONGCARD        SHORTINT
INTEGER         LONGINT         FFP             REAL
LONGREAL        BOOLEAN         CHAR            ANYTYPE
ANYPTR          STRING          SAMEPTR

TRUE            FALSE           NIL

INC     DEC     INCL    EXCL    FLIP    CAST    ABS     CAP
ODD     ACOS    ASIN    ATAN    COS     COSH    EXP     LN
LOG     SIN     SINH    SQRT    TAN     TANH    CEIL    FLOOR
SETREG  REG     SHL     SHR     LMUL    LDIV    EVEN    ROL
ROR     UNI     SEC     HALT    HALT2   PUSH    POP     SUCC
PRED    ASSERT  ASSERT2 RAISE   RAISE2  ALLOC_RESULT
```

E.1.2 Einfache Konstanten

Ganzzahlkonstanten können binär, dezimal, hexadezimal oder zu einer beliebigen Basis angegeben werden. Beginnt eine Konstante mit einer Zahl, wird sie als dezimal interpretiert. Ein Dollarzeichen „$“ leitet eine Hexzahl, ein Prozentzeichen „%“ eine Binärzahl ein. Zahlen zu einer beliebigen Basis haben die Form „Basis:Wert“

```
$$ intconst     ::=     (digit{digit})|("$"{hexdigit})|("%"{bindigit})|
                        (digit{digit}:hexdigit{hexdigit})
```

Folgende Konstanten haben zum Beispiel den selben Wert:

```
107, $6B, %1101011, 16:6B 4:1223
```

Realkonstanten enthalten einen Punkt und bei Bedarf eine Exponentenangabe. Diese wird durch ein „E“ in der Zahl begonnen. Danach kann bei Bedarf ein „-“ folgen.


```
$$ realconst    ::=     digit{digit}"."{digit}["E"["-"]digit{digit}]
```

Folgende Konstanten haben z.B. den selben Wert:

```
127.64, 1.2764E2, 12764.E-2
```

Zeichenkonstanten werden durch ein kaufmännisches Und „&“, mit nachfolgendem ASCII-Wert in Dezimalschreibweise angegeben. Alternativ kann auch das Zeichen in Anführungstrichen gegeben sein.

```
$$ charconst    ::=     ("&"digit{digit})|("char")
```

Beispiele:

```
"A", "O", &13, &10
```

E.1.3 Zeichenkettenkonstanten

Zeichenketten werden als Folge von ASCII-Zeichen in Anführungsstrichen angegeben. Dabei ist eine Konkatenation erlaubt, um im Text Sonderzeichen verwenden zu können, oder eine Konstante über mehrere Zeilen erstrecken zu lassen.

```
$$ strconst     ::=     ("{char}")|charconst
$$ stringconst  ::=     strconst{"+"strconst}
```

Beispiele:

```
"Haus", "Dies ist ein Text", "X", "Return"+&10+"neue Zeile"+&10
```

Der Typ einer Zeichenkettenkonstante ist der Typ `STRING`. Zur Kompatibilität mit Sprachen ohne expliziten Stringtyp, wird einer Zeichenkettenkonstante ein unsichtbares ASCII-NULL (`&0`) angehängt.

E.1.4 Geschützte Bezeichner und Symbole

In Cluster sind viele Sprachsymbole durch Bezeichner gegeben. Diese Bezeichner bestehen nur aus großen Buchstaben, und werden durch den Editor zusätzlich Fett hervorgehoben.

Folgende Symbole werden in Cluster verwendet:

```
"       #       $       %       &       (       )       (*      *)      +
+^      ,       -       -^      .       ..      /       :       ;       <
<=      =       >       >=      [       ]       ^       {       }
```

```
AND     AND_IF  AND_WHILE       ARRAY   AS      BCPLPTR BEGIN   BY
CLASSPTR        CLOSE   CONST   DEFINITION      DIV     DO
ELSE    ELSIF   END     EXCEPT  EXIT    FOR     FORGET  FORWARD
FROM    GROUP   HIDDEN  IF      IMPLEMENTATION  IMPORT  IN      KEY
LIBRARY LOOP    MOD     MODULE  NOT     OF      OR      OR_IF
OR_WHILE        POINTER PROCEDURE       RECORD  REF     REPEAT  RETURN
SET     SHL     SHR     STATIC  TAGS    THEN    TO      TRACK   TRY
TYPE    UNTIL   VAR     WHILE   WITH
```

E.1.5 Kommentare

Kommentare können als `(* <Kommentartext> *)` angegeben werden. Sie können geschachtelt werden, und zwischen zwei beliebigen Symbolen liegen. Durch einen senkrechten Strich "|" kann eine Zeile bzw. der Rest der Zeile nach dem Strich zum Kommentar erklärt werden.

Beispiele:

```
(* Dies ist ein (* geschachtelter! *) Kommentar *)
WriteLn; | Die ist ein Zeilenende-Kommentar
| .. der aber auch am Zeilenanfang stehen kann !
```

E.2 Typen

Cluster ist eine streng und statisch getypte Sprache, das heißt, bei allen Operationen werden die Typen beachtet. Die Typüberprüfung findet soweit möglich während der Compilierung statt.
Eine Typausdruck hat folgendes Format:

```
$$ simpletype   ::=     qualident|("["expression[".."expression]"]")|
                        ("("ident[:=expression]{,ident[:=expression]}")")
$$ varlist      ::=     ident{","ident}":"typeexpress
$$ elements     ::=     expression[".."expression]
$$ elemlist     ::=     elements{","elements}
$$ recordtype   ::=     varlist|
                        (IF KEY [ident]":"simpletype
                        {OF elemlist THEN {varlist}} END)
```



```
$$ tagtype      ::=     ident [:= expression] : typeexpress
$$ paradeclar   ::=     [VAR|REF] ident [IN expression]
                        {,ident [IN expression]}:qualident
$$ formalparams ::=     ["("[paradeclar{;paradeclar}]")"[":"qualident]
$$ typeexpress  ::=     qualident|simpletype|
                        (ARRAY [simpletype{,simpletype}] OF typeexpress)|
                        (RECORD [OF qualident] recordtype{;recordtype})|
                        (SET OF simpletype)|
                        (POINTER|CLASSPTR|BCPLPTR TO typeexpress)|
                        (ident"("expression")")|
                        (PROCEDURE formalparams)|
                        (TAG [OF qualident] tagtype{;tagtype})
                        (OBJECT [OF qualident [AS ident]
                                {"," qualident [AS ident] } ";"]
                          {
                            (ident{","ident} : typeexpress ";")|
                            ([DEFERRED]
                              (METHOD|CONSTRUCTOR|DESTRUCTOR)
                              ident [ formalparams ] ";")
                          }
                         END)
```

Eine Typdefiniton hat folgendes Format:

```
$$ typedef      ::=     TYPE {ident"="typeexpress}
```

Dabei wird einem Bezeichner ein Typ zugeordnet, der dann über diesen Bezeichner weiter verwendet werden kann.

E.2.1 Einfache Typen

E.2.1.1 Zählbare Typen

```
SHORTINT        -128..127
INTEGER         -32768..32767
LONGINT         -2147483648..2147483647

SHORTCARD       0..255
CARDINAL        0..65535
LONGCARD        0..4294967296

CHAR            &0..&255 Zeichen

BOOLEAN         TRUE , FALSE
```

Aufzählungstypen sind Typen, die eine feste Anzahl von Werten repräsentieren. Diese sind durch Konstanten gegeben, die von Null an aufwärts numeriert sind. Ein Aufzählungstyp wird durch eine Aufzählung der Elemente in runden Klammern definiert.
Beispiel:

```
Obst=(Apfel,Banane,Kirsche) oder Farben=(rot,gruen,blau)
```

Die Numerierung der Elemente eines Aufzählungstyps kann unterbrochen und mit einem neuen Wert fortgesetzt werden:

```
FileAccess=(readWrite=1004,readOnly=1005,newFile=1006);
```

oder auch:

```
FileAccess=(readWrite=1004,readOnly,newFile);
```

Verschiedene Aufzählungstypen dürfen die selben Bezeichner für ihre Elemente benutzen z.B.:
Hardware:

```
TYPE IntFlags = (tbe,dskblk,softint,ports,copper ... );
```

Exec:

```
TYPE
  NodeType = (unknown,task,interrupt,device,msgport,message,
              freeMsg,replyMsg,resource,library,memory,
              softint,font ... );
```

```
TYPE
  MsgPortAction = (signal,softint,ignore);
```

In allen drei Typen ist ein Element namens „`softint`“ enthalten. Der Compiler versucht so weit ihm dies möglich ist, das richtige Element zu finden, ist ihm dies nicht eindeutig möglich, meldet er einen Fehler. Das Element muß dann über seinen Typen qualifiziert werden:

```
  NodeType.softint
```

Unterbereichstypen repräsentieren eine Untermenge der Elemente eines einfachen Typs. Die Grenzen werden in eckigen Klammern getrennt durch „`..`“ gegeben. Wird nur ein Wert angegeben, bedeutet dies einen Unterbereich von NULL an mit sovielen Elementen.
Beispiel:

```
[1..10], [-200..200], ["A".."Z"], [10], [gruen..blau]
```


E.2.1.2 Überabzählbare Typen

Cluster bietet drei Typen, die die Menge der reellen Zahlen repräsentieren. Diese Typen sind unvollständig, da der Speicherplatz eines Rechners beschränkt ist.

```
FFP, REAL        7 geltende Stellen, Exponent bis +/- 19
LONGREAL        16 geltende Stellen, Exponent bis +/- 304
```

REAL-Zahlen sind IEEE-single-precision Zahlen. Da diese erst ab OS 2.0 bzw. mit einem mathematischen Coprozessor verfügbar sind, können sie auf einem normalen Rechner unter 1.3 nicht genutzt werden. Auf einem Amiga ohne FPU sind FFP-Zahlen schneller als REAL-Zahlen; besitzt man jedoch eine FPU, sind REAL-Zahlen schneller.

E.2.1.3 Mengentypen

Mengentypen sind Typen von Mengen über einer vorgegebenen Grundmenge. Diese muß sich aus Elementen eines zählbaren Typs zusammensetzen. Die maximale Anzahl der Elemente der Grundmenge ist 32. Ein Mengentyp wird definiert durch „**SET OF**“.
Beispiel:

```
BITSET=SET OF [0..15], Farbset=SET OF Farben
```

E.2.2 Komplexe Typen

Komplexe Typen sind in Cluster Arrays, Records, Objekte, Tags und Strings.

E.2.2.1 Arrays

Ein Array ist eine Zusammenfassung mehrerer Elemente eines Typs zu einem neuen, wobei jedes Element eindeutig über einen Index (oder mehrere Indices) bezeichnet wird. Ein Array besitzt einen Unterbereichstypen als Indextyp und einen Basistyp.
Beispiel:

```
Farbwerte = ARRAY Farben OF [0..15];
```

Mehrdimensionale Arrays sind Arrays von Arrays:

```
ARRAY T1 OF ARRAY T2 OF ARRAY T3 .. OF T
```

ist gleichbedeutend mit

```
ARRAY T1,T2,T3... OF T

Matrix = ARRAY [1..3],[1..3] OF REAL;
```

Es ist auch möglich Arrays über einen Typen zu definieren, dessen Obergrenze nicht bekannt ist. Diese offenen Arrays haben immer INTEGER als Indextyp.

```
TYPE
  Vector = ARRAY OF REAL;
```

Sie können nur auf vier Arten verwendet werden, als Ziel eines Pointers, als Parameter einer Prozedur, als Typ einer Konstanten oder bei der Definition eines geschlossenen Typen. Wird ein offenes Array als Pointerziel verwendet, sollte dafür ein **CLASSPTR** verwendet werden, da nur so eine Bereichsüberprüfung möglich ist. Die Grenzen eines existierenden offenen Arrays können über Attribute ermittelt werden.

```
TYPE
  Vector3 = Vector(3);
  VecPtr  = CLASSPTR TO Vector;

CONST
  IntArray = ARRAY OF INTEGER:(1,2,3,4);
```

Will man ein solches offenes Array allozieren, trägt man zuerst den `RANGE` des Arrays ein, und ruft dann `New` auf:

```
VAR
  UserVec  : VecPtr;

BEGIN
  UserVec'RANGE:=6;
  New(UserVec);
```

E.2.2.2 Records

Ein Record ist eine Zusammenfassung mehrerer Elemente verschiedener Typen zu einem neuen, wobei jedes Element über einen zusätzlichen Namen angesprochen wird.

```
TYPE
  Adresse = RECORD
```



```
              namen,
              vornamen : STRING(20);
              alter    : INTEGER;
            END;
```

Werden in einem Record nicht alle Felder gleichzeitig benötigt, können diese übereinandergelegt werden. Dies spart Speicherplatz. Man spricht in diesem Fall von einem varianten Record.

```
TYPE
  Geschlecht = (maennlich,weiblich);
  Person     = RECORD
                 namen,
                 vornamen : STRING(20);
                 IF KEY geschl : Geschlecht
                   OF maennlich THEN dienstgrad   : STRING(20);
                   OF weiblich  THEN maedchenname : STRING(20);
                 END;
               END;
```

Records lassen sich um weitere Elemente erweitern, dabei bleiben alle vorherigen Felder erhalten. Ein Record, der sich auf einen bereits bestehenden gründet, wird durch **RECORD OF** eingeleitet.
Ein Beispiel aus Exec:

```
MinNodePtr = POINTER TO MinNode;
MinNode    = RECORD
               succ,pred : MinNodePtr;
             END;

Node       = RECORD OF MinNode
               type : NodeTypes;
               pri  : SHORTINT;
               name : SysSTringPtr;
             END;

Message    = RECORD OF Node
               replyPort : MsgPortPtr;
               length    : CARDINAL;
```

```
              END;

IORequest  = RECORD OF Message
               device  : DevicePtr;
               unit    : UnitPtr;
               command : CARDINAL;
               flags   : IOFlagSet;
               error   : SHORTCARD;
             END;
```

So lassen sich Hierarchien von Typen aufbauen, die nach unten zuweisungskompatibel sind.

E.2.2.3 Strings

Ein String in Cluster hat folgendes Format:

```
STRING = RECORD
           len  : INTEGER;
           data : ARRAY OF CHAR;
         END;
```

Es gibt wie bei Arrays offene und feste Strings. Für Variablen dürfen nur Stringtypen mit fester Maximallänge verwendet werden.

```
String30 = STRING(30);
```

Offene Strings unterliegen den selben Beschränkungen wie offene Arrays.

Jeder String hat zwei Endkennzeichen: die in `len` gegebene Länge und ein Nullbyte nach dem letzten Zeichen. Dieses Nullbyte geht nicht in `len` ein, wohl aber in die maximale Länge des Strings.

E.2.2.4 Tag-Typen

Ein Tag Element besteht immer aus zwei Elementen. Das erste Element enthält einen TAG-Wert, der aussagt, welchen Typ das zweite Element hat. Dieses zweite Feld ist maximal vier Bytes groß. Hierbei ist ein Erben von bereits bestehenden Tag-Typen möglich.
Beispiel:


```
ScreenTags = TAGS OF StdTags
               width  = $80000020 : INTEGER;
               height             : INTEGER;
               depth              : INTEGER;
               name               : POINTER TO STRING;
               flags              : ScreenFlagSet
             END;
```

Tags werden durch das Amiga-OS 2.0 häufig verwendet und zwar in der Form von TAG-Listen, das sind (nach Bedarf auch verkettete) Arrays von Tag- Typen.
Beispiel:

```
ScreenTagList = ARRAY OF ScreenTags;
```

Um nun einen Screen zu öffnen, muß eine Tag-Liste angegeben werden, in der die Elemente aufgeführt sind, die sich von den Werten eines Standardscreens unterscheiden.

```
OpenScreenTagList(ScreenTagList:(width  : 320,
                                 height : 256,
                                 name   : "Name"'PTR,
                                 DONE));
```

Alternativ können statt Arrays von Tags auch Listen von Tags verwendet werden, dann können als Argumente der Tags auch nicht konstante Ausdrücke verwendet werden.

```
OpenScreenTags(ns : NewScreenPtr;tags : LIST OF ScreenTags);
...
OpenScreenTags(NIL, width  : Max(320,StandardWidth),
                    height : Max(256,StandardHeight),
                    name   : "Name"'PTR,
                    DONE);
```

Für Listen von Tags sind zwei spezielle Standardfunktionen definiert, **TGET** und **TPUT**. **TGET** extrahiert einen Tagwert aus einer Liste, **TPUT** ändert einen Wert in einer Tagliste. **TGET** erhält drei Argumente, die Liste, den Namen des Tags, dessen Wert ermittelt werden soll, und einen Defaultwert, der zurückgeliefert werden soll, falls der Tag nicht in der Liste enthalten ist. **TPUT** hat eine ähnliche Argumentliste, lediglich der letzte Wert ist kein Defaultwert, sondern der neue Inhalt des Taglistenelements.

```
VAR tags := ScreenTagList:(width : 320, height : 256,
                            name : "Name"'PTR,DONE);
    w    : INTEGER;
...
   w:=TGET(tags,width,0);
   TPUT(tags,name,"Neuer name");
...
```

E.2.3 Zeigertypen

Es gibt in Cluster drei Zeigertypen: normale Zeiger (POINTER), Zeiger für offene Typen (CLASSPTR) und Zeiger, die zu BCPL kompatibel sind (BCPLPTR).
Beispiel:

```
TYPE
  NodePtr = POINTER TO Node;

  Node    = RECORD
              prev,
              next  : NodePtr;
              key   : INTEGER;
            END;
```

Es existiert ein Pointertyp, der zu allen Pointertypen und zu `LONGINTs` bzw. `LONGCARDs` kompatibel ist: `ANYPTR`. Dieser Pointer hat kein eigentliches Ziel. Zu diesem Typ gibt es die Konstante `NIL`, die eine nicht vorhandene Adresse bedeutet.

Neben diesen allgemeineren Pointertypen exisitiert noch ein Typ, der speziell zum Gebrauch bei der Vererbung von Records eingeführt wurde, der `SAMEPTR`. Ein `SAMEPTR` kann nur als Element eines Records verwendet werden, er stellt einen Zeiger auf den Typen dar, den der aktuelle Record bildet.
Beispiel:

```
TYPE
  Node    = RECORD
              prev,
              next  : SAMEPTR;
            END;
  Node2   = RECORD OF Node
              data  : INTEGER
            END;
```


Die Felder `prev` und `next` haben in Records des Typs `Node` den Typ **POINTER TO** `Node`, in Records des Typs `Node2` aber den Typen **POINTER TO** `Node2`. Der Record bleibt trotzdem nach unten kompatibel, da ja ein **POINTER TO** `Node2` auch ein Nachfahre von **POINTER TO** `Node` darstellt. Der Vorteil ist aber, daß Ausdrücke der Form `Node2^.next^.data` möglich sind, was bei der Verwendung des Typs **POINTER TO** `Node` anstatt eines `SAMPETRs` nicht möglich wäre.

E.2.4 Opake-Typen

Opake Typen sind Typen, die in einem Definitions-Modul angegeben, aber erst im Implementations-Modul definiert werden. Dies ermöglicht das Geheimnisprinzip. Ein opaker Typ darf nur stellvertretend für einen Pointer stehen.

```
DEFINITION MODULE <name>

TYPE
  List  = HIDDEN;

END <name>.

IMPLEMENTATION MODULE <name>

TYPE
  List  = POINTER TO
            RECORD
              first,
              prev   : NodePtr;
            END;

END <name>.
```

E.2.5 Objekttypen

Objekte sind in Cluster mit Records vergleichbar, besitzen allerdings zwei zusätzliche Eigenschaften, einen dynamischen Typen, und eine Menge von Methoden, die dynamisch zur Laufzeit ermittelt werden.

Objekte können nicht direkt als Variablen verwendet werden, sondern lediglich durch Zeiger auf Objekte. Spezielle Methoden, die durch das Schlüsselwort `CONSTRUCT` (bzw. `DESTRUCT`) eingeleitet werden, dienen der Erzeugung und Vernichtung der Objekte.

Objekte können wie Records durch Erben auseinander hervorgehen, hierbei ist auch ein Mehrfacherben von mehreren Elternobjekten möglich.

In der Objektdefinition müssen auch alle für dieses Objekt definierten Methoden mitangegeben werden. Soll eine Methode eines Elternobjektes durch eine neue Methode überdefiniert werden, so muß dies ebenfalls in der Definition angegeben werden.

E.2.6 Prozedurtypen

Prozeduren haben in Cluster ebenfalls einen Typ. Dieser ist durch die Übergabeparameter gegeben (siehe Prozedurdeklaration).

E.3 Variablendeklaration

Alle Variablen, die in Cluster benutzt werden, haben einen festen Typ, und müssen vor ihrer Benutzung deklariert werden.

```
$$ vardeclar      ::= ident [(IN expression)|STATIC]
                        {,ident [(IN expression)|STATIC]}
                        (([:typeexpress] := expression)|(:typeexpress))
$$ vardeclaration ::=VAR [vardeclar{;vardeclar}]
```

Das Schlüsselwort IN gibt an, daß die Variable in ein Register zu legen ist. Dies gilt nur für den unmittelbaren Sichtbarkeitsbereich der Variablen, nicht jedoch für weiter innen liegende.
Beispiel:

```
VAR i,j      : INTEGER;
    c        : CHAR;
    k IN D2  : INTEGER;
    l        : INTEGER := 2;
    n        := NewScreen:(width=...);
```

Die Registernamen `D0` bis `A7` sind im Modul `SYSTEM` als Aufzählungstyp „`Regs`" definiert.

Durch die Initialisierung „`:=`" kann der Variable bei ihrer Deklaration ein Startwert zugewiesen werden. Lokale Variablen werden am Prozeduranfang, globale Variablen dagegen werden am Programmanfang initialisiert. Alle globalen Variablen, die nicht mit einer Konstante vorbelegt werden, werden am Programmanfang mit Nullen initialisiert.

Das Schlüsselwort **STATIC** kann nur bei Variablendeklarationen innerhalb von Prozeduren verwendet werden. Diese Variablen überleben ihre Prozedur, und haben beim nächsten Aufruf noch den selben Wert wie beim Verlassen. Werden sie vorinitialisiert, so geschieht dies nur einmal am Programmanfang.

E.4 Konstanten

E.4.1 Einfache Konstanten

Einfache Konstanten sind Konstanten der einfachen Typen.
Beispiel:

```
123.43, "C", 12, $1234
```

Der Typ einer derartigen Konstante ergibt sich aus ihrer Darstellung, so ist „C" vom Typ `CHAR`.

E.4.2 Komplexe Konstanten

In Cluster ist es auch möglich, Konstanten komplexer Typen zu bilden. Stringkonstanten werden einfach in Hochkomma eingeschlossen:

```
"Dies ist ein Text"
```

Andere komplexe Konstanten werden durch einen Typbezeichner mit nachfolgendem Doppelpunkt eingeleitet:

```
$$ comconst     ::=    expression|
                       ("("[comconst{,comconst}]")")|
                       ("("[ident=comconst{,ident=comconst}]")")|
                       ("{"elemlist"}")|
                       (ident : comconst)

$$ complexconst ::=    ((ident|(ARRAY OF))":"comconst)|
                       ("{"elemlist"}")
```

Beispiel:

```
ARRAY OF Vector3:((1,0,0),(0,1,0),(0,0,1));

FarbSet:{gruen,blau};

ARRAY OF Adresse:((name   ="Sigmund",
                   vorname="Ulrich",
                   alter  =23),
                  (name   ="Pfrengle",
                   vorname="Thomas",
                   alter  =21));
```

Ein Zeiger auf eine derartige komplexe Konstante ist ebenfalls eine Konstante. Dieser Zeiger kann durch das Attribut PTR gewonnen werden.

Bei Mengen, deren Typ dem Compiler sicher bekannt ist (bei Zuweisungen, innerhalb komplexer Konstanten etc.) kann auf den Typbezeichner verzichtet werden.

Bei Recordkonstanten kann auf die Bezeichner verzichtet werden (sollte nur in Ausnahmefällen geschehen, da die Lesbarkeit des Programms meist vermindert wird), und die Elemente können in ihrer Reihenfolge durch Kommata getrennt aufgezählt werden. Diese Technik ist zu empfehlen bei großen Arrays aus kleinen Records.

Bei Konstanten, die aus offenen Arrays gebildet werden, wird die wirkliche Größe aus der Anzahl der angegebenen Elemente gebildet.

E.4.3 Konstantendefinition

Konstanten können mit einem Namen belegt werden, und so mehrfach verwendet werden. Dabei ist für komplexe Konstanten auch eine Vorwärtsdeklaration möglich, indem nur der Typ der Konstanten angegeben wird. Dies ist besonders in Defintionsmodulen oder für konstante verkette Listen sinnvoll.

```
$$ constdef        ::= ident "=" (ident|expression)
$$ constdefinition ::= CONST [constdef{;constdef}]
```

```
CONST
  pi    = 3.1415;
  Basis = ARRAY OF Vector3:((1,0,0),
                            (0,1,0),
                            (0,0,1));
```

E.4.4 Ausnahmedeklaration

Ausnahmen sind Zustände, die eintreten, wenn nicht planmäßige Ereignisse auftreten, wie zu wenig Speicher, eine Division durch Null oder der Versuch, aus einer leeren Datenstruktur ein Element zu lesen.

Es gibt drei Arten von Ausnahmen: solche, die durch den Prozessor hervorgerufen werden (wie Division durch Null, Bereichsfehler etc.); solche, die durch die Standard/Schnittstellenmodule ausgelöst werden (wie zu wenig Speicher) und benutzerdefinierte Ausnahmen.

```
$$ exceptdefs   ::= ident : (stringconst|intconst)
$$ exceptdef    ::= EXCEPTION exceptdef{;exceptdef}
```

Beispiele:

```
EXCEPTION
  NotEnoughMemory : "Nicht genug Systemspeicher verfügbar";
  DivisionByZero  : 8;
```


E.4.5 Gruppendeklaration

Objekte und Deklarationen in einem Definitionsmodul können zu einer Gruppe zusammengefaßt werden, durch deren Importierung alle darin enthaltenen Objekte importiert werden.

```
$$ groupdef     ::=     GROUP {ident = ident{,ident};}
```

In einer solchen Gruppe können auch andere Gruppen enthalten sein. Stammen diese aus einem anderen Modul, müssen sie qualifiziert angegeben werden.

E.5 Ausdrücke

Ein Ausdruck besteht aus Variablen, Konstanten, Attributen, Funktionen und Operatoren.

```
$$ relop        ::=     "="|"#"|"<="|">="|"<"|">"
$$ operator     ::=     "+"|"-"|"*"|"/"|"^"|DIV|MOD|SHL|SHR|AND|OR
$$ paralist     ::=     [expression{,expression}]
$$ term         ::=     {-}|{NOT}(qualident|complexconst|intconst|realconst|
                        charconst|stringconst|("("expression")"))|
                        relop|"+"
$$ expression   ::=     term{(operator term)|("^"|"+^"|"-^")|
                        ("." ident)|("'" ident)|(IN expression)|
                        (OF elemlist)|("("paralist")")|
                        ("["expression{,expression}"]")
```

Die Operatoren haben bei verschiedenen Typen verschiedene Bedeutungen:

Operator	Zahlen:	Mengen:	Zeiger:	Boolean:
+	Summe	Vereinigung		
-	Differenz	Differenz		
*	Produkt	Schnitt		
/	Division	Sym.Differenz		
^	Potenz		Dereferenz	
DIV	Ganzzahldiv.			
MOD	Rest der Div.			
SHL	Linksshiften			
SHR	Rechtsshiften			
+^			Postinc.Deref.	
-^			Predec.Deref.	
=	gleich	gleich	gleich	Äquivalenz
#	ungleich	ungleich	ungleich	Exklusiv-Oder
<	kleiner		kleiner	
<=	kleiner/gleich	Teilmenge von	kleiner/gleich	
>	größer		größer	
>=	größer/kleich	Obermenge von	größer/gleich	
AND				Und
OR				Oder
IN		Element aus		
OF	Element aus			

Wird ein relationaler Operator als parameterlose Funktion benutzt, liefert er den Zustand des korrespondierenden Prozessorflags.
Folgende Regeln gelten für die Rechnungskompatibilität der Typen t1 und t2:

```
t1 ist gleich t2
t1 und t2 sind SHORTINT, INTEGER, LONGINT oder ANYPTR
t1 und t2 sind SHORTCARD, CARDINAL oder LONGCARD
t1 und t2 sind FFP, REAL oder LONGREAL
t1 und t2 sind Unterbereiche zweier rechnungskompatibler Typen
```

Das Zeichen „.“ dient dazu, ein Element eines Records zu qualifizieren. Mit den Klammern [] wird ein Element eines Arrays bestimmt. Dies kann bei mehrdimensionalen Arrays auf zwei verschiedene Arten geschehen:


```
A[i1][i2].. oder A[i1,i2,..]
```

Das Zeichen „^“ dient zur Dereferenzierung eines Zeigers, d. h. aus dem Zeiger wird das Objekt gebildet, auf das der Zeiger zeigt. Bei Zeigern auf Records und Arrays kann auf die explizite Dereferenzierung verzichtet werden, wenn diese unmittelbar darauf qualifiziert oder indiziert werden.
Bei der Auswertung eines Ausdrucks gelten folgende Prioritäten:

1. Funktionsauswertung, Dereferenzierung, Indizierung, Qualifizierung, Attributbildung
2. Unäres Minus, Negation
3. Potenzierung
4. Multiplikation, Division, Shifts, logisches Und
5. Addition, Subtraktion, logisches Oder
6. relationale Operatoren

Attribute liefern Informationen über Objekte und deren Typen. Sie werden durch ein Hochkomma „'“ und einen Attributsbezeichner ermittelt. Folgende Attribute sind vorgegeben:

'PTR	: Zeiger auf das Objekt
'ADR	: Adresse des Objekts
'MIN	: kleinster Wert, oder untere Indexgrenze
'MAX	: größter Wert, oder obere Indexgrenze
'RANGE	: Anzahl der Elemente eines Arrays/Strings
'SIZE	: Größe in Bytes

Neben selbstdefinierbaren Funktionen existiert auch eine Anzahl vordefinierter Funktionen. Diese werden meist nicht aufgerufen, sondern direkt in den erzeugten Code eingebaut.

```
ODD(x:INTEGER):BOOLEAN  liefert TRUE, falls x ungerade ist.
CAST(Typ,x):Typ         Wandelt den Typ des Objekts x in Typ
LMUL(x,y):LONG..        Multiplikation von 16x16Bit auf 32Bit
LDIV(x,y):...           Division von 32x16Bit auf 16Bit
REG(x):LONGINT          Inhalt des Prozessorregisters x
PRED(x):...             Liefert den Wert von x um eins vermindert zurück,
                        auch für Aufzählungstypen
SUCC(x):...             Liefert den Wert von x, um eins erhöht zurück.
CEIL(real):...          Liefert die nächste ganze Zahl, die größer oder
```

```
                         gleich real ist.
FLOOR(real):...          Liefert die nächste ganze Zahl, die kleiner oder
                         gleich real ist.
```

Die Standardfunktionen `SUCC` und `PRED` liefern bei Integers etc. den Wert +/− 1, bei Zeigern auf Strukturen einen Zeiger des selben Typs, dessen Zieladresse um die Größe eines Zielelements erhöht ist.
Beispiel:

```
  SUCC(3)   = 4
  PRED("B") = "A"

TYPE
  Elem  = RECORD
            x,y,z : INTEGER;
            c,h   : CHAR;
          END;

VAR
  Array : ARRAY [100] OF Elem;
  p     : POINTER TO Elem;
  i     : INTEGER;

BEGIN
  p:=Array[0]'PTR;
  FOR i:=Array'MAX TO 0 BY -1 DO
    p.x:=10;...;p.h:="C";
    p:=SUCC(p)
  END;
END...
```

Oder auch extrem:

```
  SUCC(SUCC(p))^.x:=4;
```

Neben diesen allgemeinen Funktionen verfügt Cluster noch über eine große Anzahl mathematischer Funktionen für `REAL`s und `LONGREAL`s:

`SIN, ASIN, SINH, COS, ACOS, COSH, TAN, ATAN, TANH, LN, LOG, EXP, SQRT, ABS`

Der Typ eines Ausdrucks kann sicher in einen anderen überführt werden, indem der Typbezeichner als Funktion verwendet wird.

E.6 Anweisungen und Strukturen

Eine Anweisungsfolge ist eine Folge von Anweisungen, die durch Semikola getrennt sind.

```
$$ statementsequence ::= statement{;statement}
$$ statement         ::= |assignment|repeatstatement|whilestatement|
                          ifstatement|loopstatement|forstatement|
                          withstatement|exitstatement|returnstatement|
                          trystatement|forgetstatement|procedurecall
```

Auch die leere Anweisung ist eine Anweisung.

E.6.1 Zuweisungen

Die wichtigste Anweisung ist die Zuweisung. Dabei wird einem Ausdruck auf der linken Seite der Wert des Ausdrucks der rechten Seite zugewiesen. Der Ausdruck der linken Seite muß eine Variable bzw. eine durch einen Pointer bezeichnete Speicherstelle sein.

```
$$ assignment   ::= expression ":=" expression
```

Folgende Regeln gelten für Zuweisungskompatibilität von t1 und t2:

```
t1 ist gleich t2
t1 und t2 sind Pointer auf den selben Typen
t1 und t2 sind SHORTINT, INTEGER, LONGINT, SHORTCARD, CARDINAL, LONGCARD
               oder ANYPTR
t1 und t2 sind REAL oder LONGREAL
t1 und t2 sind Unterbereiche eines zuweisungskompatiblen Typen
t1 und t2 sind Arrays gleichen Basistyps mit gleichviel Elementen
t1 und t2 sind Strings beliebiger Länge
t1 und t2 sind direkte Nachfolger bei offenen Records
```

E.6.2 REPEAT..UNTIL..

```
$$ repeatstatement     ::= REPEAT statementsequence UNTIL expression
```

Die Anweisungsfolge zwischen **REPEAT** und **UNTIL** wird solange ausgeführt, bis der Ausdruck an ihrem Ende wahr wird.

```
i:=0;
REPEAT WriteInt(i,0);INC(i) UNTIL i=10
```

liefert:

```
0 1 2 3 4 5 6 7 8 9
```

E.6.3 IF-Struktur

Die IF-Struktur ist extrem mächtig, sie enthält auch die von Modula bekannte CASE-Anweisung.

```
$$ ifcase       ::= (expression (THEN statementsequence)|
                    (AND_IF ifcase))|(KEY expression
                    {OF elemlist (THEN statementsequence END)|
                    (AND_IF ifcase)})
                    [OR_IF|ELSIF ifcase]|([ELSE statementsequence] END)
$$ ifstatement  ::= IF ifcase
```

Die Grundstruktur ist:

```
IF b1 THEN
  S1
OR_IF b2 THEN
  S2
...
ELSE
  SN
END;
```

Der zu dem ersten Boolausdruck `bi`, der `TRUE` ist, gehörende Programmteil `Si` wird ausgeführt, und das Programm hinter `END` fortgesetzt. Ist kein Ausdruck wahr, wird `SN` ausgeführt.

Jeder **IF**-Fall kann durch Einführen einer **AND_IF**-Klausel eingeschränkt werden, die selbst wieder **OR_IF** und/oder **ELSE**-Klauseln besitzt. Ist keiner dieser Fälle wahr, und kein **ELSE**-Teil vorhanden, wird der Vergleich mit dem nächsten tieferliegenden **OR_IF** oder **ELSE** weitergeführt.

```
...
OR_IF bx
  AND_IF bx1 THEN
    ...
  OR_IF bx2 THEN
    ...
  ELSE
  ...
  END
OR_IF by
...
```


Sollen anhand eines Schlüsselwertes verschiedene Möglichkeiten geprüft werden, kann „**KEY**“ benutzt werden.

```
OR_IF KEY a1
  OF ... THEN ... END
  OF ... AND_IF bx THEN
           ...
         OR_IF KEY a11
           OF ... THEN
           ...
         END
  OF ... THEN ... END
OR_IF ...
```

E.6.4 WHILE-Struktur

Die WHILE-Struktur ist völlig symmetrisch zur IF-Struktur. Der Unterschied ist, daß nachdem ein Programmteil zu einem DO ausgeführt wurde, wieder zurück zum Anfang der WHILE-Struktur gesprungen wird.

```
$$ whilecase      ::= (expression (DO statementsequence)|
                      (AND_WHILE  whilecase))|(KEY expression
                      {OF elemlist (DO statementsequence END)|
                      (AND_WHILE whilecase)})
                      [OR_WHILE whilecase]|([ELSE statementsequence] END)
$$ whilestatement ::= WHILE whilecase
```

E.6.5 LOOP..END, EXIT

Die LOOP-Struktur ist eine Schleife ohne dedizierte Terminationsbedingung. Die Schleife wird durch das Schlüsselwort EXIT terminiert, das an beliebiger Stelle innerhalb der Schleife stehen darf.

```
$$ loopstatement       ::= LOOP statementsequence END
$$ exitstatement       ::= EXIT
```

E.6.6 FOR..DO..END

Die FOR-Struktur ist eine Schleifenstruktur, deren Termination durch einen Zähler gegeben ist. Dieser läuft von einem angegebenen Startwert zu einem Endwert, wobei der Schleifenindex jedesmal um eine vorgegebene konstante Schrittweite erhöht wird. Als Schleifenzähler sind alle zählbaren Typen zugelassen.

```
$$ forstatement ::= FOR assignment TO expression [BY expression] DO
                    statementsequence END
```

```
FOR i:=start TO end BY step DO statement END;
```

entspricht für positives step:

```
i:=start;WHILE i<=end DO statement;INC(i,step) END;
```

bei negativem step:

```
i:=start;WHILE i>=end DO statement;DEC(i,step) END;
```

Wird kein Schrittwert angegeben, wird eins verwendet.

E.6.7 WITH..DO..END

Die WITH-Struktur erzeugt temporäre Variablen, oder gibt bestehenden einen neuen Namen.

```
$$ withstatement    ::= WITH expression [AS ident]{,expression AS ident} DO
                        statementsequence END
```

Wird als Ausdruck ein Typ angegeben, wird eine neue Variable erzeugt.

```
VAR source,dest : POINTER TO
                    ARRAY [0..3] OF
                      POINTER TO
                        ARRAY [0..9] OF INTEGER;
    i           : INTEGER;

FOR i:=0 TO 9 DO
  dest^[2]^[i]:=source^[3]^[9-i]
END;
```

Kann durch WITH vereinfacht werden:

```
WITH dest^[2]^ AS d,source^[3]^ AS s DO
  FOR i:=0 TO 9 DO
    d^[i]:=s^[9-i]
  END;
END;
```


Dies ermöglicht dem Compiler auch Optimierungen, da er die Adressausdrücke nur einmal errechnen muß.

Wird kein neuer Namen durch **AS** angegeben, wird der Name der Basisvariablen beibehalten.

E.6.8 RETURN

RETURN beendet die Ausführung einer Prozedur. Handelt es sich dabei um eine Funktion mit einem einfachen Rückgabetypen, muß dahinter ein Ausdruck angegeben werden, der den Rückgabewert darstellt. Bei Funktionen mit komplexem Rückgabewert wird dieser durch die Variable RESULT zurückgegeben.

```
$$ returnstatement      ::= RETURN [expression]
```

E.6.9 Prozeduraufruf

Eine Prozedur wird aufgerufen, wenn ein Ausdruck einen Prozedurtypen liefert (eine normale Prozedur ist eine Konstante ihres eigenen Prozedurtyps).

```
$$ procedurecall        ::= expression "("paralist")"
```

Cluster verfügt über eine Reihe von Compilerprozeduren, die nicht aufgerufen werden, sondern direkt in den erzeugten Code eingebaut werden.

`INC(x)`	erhöht `x` um 1, nur zählbare Typen und Zeiger
`INC(x,m)`	erhöht `x` um `m`
`DEC(x)`	erniedrigt `x` um 1
`DEC(x,m)`	erniedrigt `x` um `m`
`EVEN(x)`	rundet `x` auf die nächste gerade Zahl ab
`SHL(x,n)`	shiftet `x` um `n` Bits links
`SHR(x,n)`	shiftet `x` um `n` Bits rechts
`ROL(x)`	rotiert `x` um ein Bit nach links
`ROR(x)`	rotiert `x` um ein Bit nach rechts
`INCL(s,e)`	fügt das Element `e` in die Menge `s` ein
`EXCL(s,e)`	entfernt das Element `e` aus der Menge `s`
`FLIP(s,e)`	wechselt die Anwesenheit von `e` in der Menge `s`
`UNI(d,s)`	verinigt die Menge `d` mit der Menge `s`
`SEC(d,s)`	schneidet die Menge `d` mit der Menge `s`
`SETREG(r,v)`	schreibt den Wert von `v` in Register `r`
`INLINE(x,..)`	schreibt eine Anzahl von Datenwörter in das erzeugte Programm

`HALT(x)`	beendet das Programm mit Fehlernummer `x`
`HALT2(x)`	beendet das Programm mit Fehlernummer `x` an der Stelle, an der die aktuelle Prozedur aufgerufen wurde
`RAISE(x)`	löst die Ausnahme `x` aus
`RAISE2(x)`	löst die Ausnahme `x` aus, wobei als Ereignispunkt der Punkt des Prozeduraufrufs angegeben wird
`ASSERT(b,x)`	stellt sicher, daß die Bedingung `b` erfüllt ist, andernfalls wird die Ausnamhe `x` ausgelöst
`ASSERT2(b,x)`	wie ASSERT, nur wird als Ereignispunkt der Punkt des Prozeduraufrufs angegeben.
`PUSH(set)`	Sichert die im set angegebenen Register auf den Stapel, z.B. für Interrupts etc.
`POP(set)`	Nimmt mit PUSH gesicherte Register wieder vom Stapel herunter.

E.6.10 Der Inline-Assembler

Der Assembler ist als Standardfunktion implementiert: `ASSEMBLE(...)`. Er kann an einer beliebigen Stelle im Programm benutzt werden, der erzeugte Code wird an genau dieser Stelle ins Programm eingefügt.

Bsp.:

```
  BEGIN
  ...
    ASSEMBLE(
             MOVE.L D0,A0
             ...
             BNE    loop
            );
  ...
  END ...;
```

Es ist möglich auf alle lokalen und globalen Variablen über ihren Namen zuzugreifen, der Assembler wählt automatisch die richtige Adressierungsart. Auch auf Variablen innerhalb einer Prozedur kann zugegriffen werden, der Assembler übersetzt den Zugriff in eine `SP`-relative Adressierung. Dies wirft allerdings ein Problem auf, wenn der Stackpointer verändert wird, da dies den Compiler schwer durcheinander bringt.

Bsp.:

```
VAR i : ARRAY [0..5] OF INTEGER;

PROCEDURE IncI(at,um : INTEGER);
BEGIN
  ASSEMBLE(
           MOVE um,D0
           MOVE at,D1
           ASL  #1,D1
           LEA  i,A0
           ADD  D0,(A0,D1)
          )
END IncI;
```

oder auch:

```
PROCEDURE IncI(at,um : INTEGER);
BEGIN
  ASSEMBLE(USE  D0
           MOVE um,D0
           ADD  um,i[at]
          )
END IncI;
```

Der Assembler überführt diesen Code automatisch in die entsprechenden Anweisungen. Da der Compiler dazu freie Register benötigt, diese dem Programm aber nicht einfach stehlen kann, ist es sinnvoll, die Register, die man innerhalb des Programmes benutzen möchte, mit `USE` anzugeben. Der Assembler betrachtet die nicht belegten Register als sein Eigentum und nutzt diese für erweiterte Adressierungen.
Es können auch Registervariablen verwendet werden:

```
PROCEDURE IncI(at IN D0,um IN D1 : INTEGER);
BEGIN
  ASSEMBLE(USE  A0
           LEA  i,A0
           ASL  #1,at
           ADD  um,(A0,at)
          )
END IncI;
```

Der Assembler versucht so gut wie möglich, dem Stack-Pointer zu folgen; so werden Veränderungen des `SP` durch `ADD #,A7`, `SUB #,A7`, `LINK` sowie `-(A7)` und `(A7)+` erkannt, und richtig in die Berechnung der Prozedur-lokalen Variablen einbezogen. Was der Assembler nicht berücksichtigen kann sind Veränderungen des `SP` durch `MOVE ...,A7` o.ä, Veränderungen durch variable Werte wie in `ADD D0,A7`, und Unterprogramme.
Bsp.:

```
PROCEDURE ReturnSame(i : INTEGER):INTEGER;
VAR j : INTEGER;
BEGIN
  ASSEMBLE(SUB.W  #100,A7
           MOVE   i,j
           ADD.W  #100,A7
  RETURN j;
END ReturnSame;
```

Dies ist völlig legal und wird auch richtig übersetzt.

Der Compiler verlangt, daß am Ende eines Assemblerteiles der Stackpointer wieder den selben Wert hat wie beim Eintritt, da er sonst beim Mitzählen aus dem Ruder geraten würde.

Labels können an einer beliebigen Stelle im Assemblerteil verwendet oder definiert werden (dies natürlich nicht mitten in einem Befehl). Bei der Definition eines Labels muß dieses von einem Doppelpunkt gefolgt werden. Es gibt keine feste Form, wie Assembleranweisungen im Text formatiert werden müssen, es muß lediglich zwischen Label, Mnemonic und Operanden mindestens ein Space stehen, auch dürfen beliebig viele Anweisungen in einer Zeile stehen.
Bsp.:

```
PROCEDURE Copy(from IN A0,to IN A1,size IN D2 : LONGINT);
BEGIN
  ASSEMBLE(USE     D1
           MOVE    size,D1  | .L wird erkannt, da "size" ein LONGINT
           SHR     #2,size
           SUB     #1,size
  loop:    MOVE.L  (from)+,(to)+ DBRA size,loop
           BTST    #1,D1 BEQ not2 MOVE.W (from)+,(to)+
  not2:    BTST    #0,D1 BEQ not1 MOVE.B (from)+,(to)+
  not1:    )
END Copy;
```


Dies steigert zwar nicht die Lesbarkeit, ist aber möglich. Nicht möglich ist es mit Labels zu rechnen, da dies den Shortbranch-Optimierer aus dem Ruder bringen würde (vorerst sind noch fast alle Optimierungen, die der Compiler normalerweise ausführt, auch im Assemblerteil vorhanden, also keinen Code patchen !!).

Es gibt keine Anweisungen wie ADDA oder ADDI, diese werden durch den Assembler bei Bedarf erzeugt.

E.6.11 FORGET

Viele Funktionen (gerade solche des Betriebssystems) sind eigentlich keine Funktionen, sondern Prozeduren, da ihre eigentliche Aufgabe nicht in der Rückgabe eines Wertes, sondern in der Ausführung einer Tätigkeit besteht (z.B. WritePixel). Diese Funktionen liefern mehr als Nebeneffekt auch noch einen Wert zurück, der in den meisten Fällen problemlos ignoriert werden kann. Da es aber sehr gefährlich ist, eine Verwendung von Funktionen als Prozeduren zuzulassen, es andererseits aber sehr störend ist, das Funktionsergebnis jedesmal einer Dummy-Variablen zuzuweisen, kann mit FORGET ein Ausdruck ausgewertet werden, dessen Ergebnis danach einfach vergessen wird.

```
$$ forgetstatement      ::= FORGET
```

Beispiel:

```
FORGET WritePixel(rast,x,y);
```

E.6.12 TRY..EXCEPT

Ausnahmesituationen (exceptions) können durch Programm- bzw. Datenfehler, systembedingte Probleme wie Speichermangel oder durch beutzerdefinierte Fehlersituationen entstehen.

Ist kein exception handler installiert, kann das Programm nur durch einen Abbruch reagieren.

```
$$ trystatement ::= TRY statementsequence EXCEPT
                    {OF ident{","ident} THEN statementsequence END}
                    [ELSE statementsequence] END
```

Die Anweisungssequenz zwischen **TRY** und **EXCEPT** wird ausgeführt; tritt dabei eine Ausnahme auf, wird das Programm mit dem passenden exception handler des **EXCEPT** Teils fortgesetzt. Ist kein passender exception handler vorhanden, wird der eventuell vorhandene **ELSE**-Teil ausgeführt, und die Ausnahme an den nächsten übergeordneten exception handler weitergereicht.

Somit werden alle Ausnahmen vom nächsten passenden exception handler bearbeitet. Ist überhaupt kein exception handler vorhanden, wird das Programm mit einem Laufzeitfehler abgebrochen. Ein für die exception angegebener Text wird als Fehlermeldung verwendet.
Beispiel: Auswertung einer math. Funktion

```
PROCEDURE F(x : INTEGER):INTEGER;
BEGIN
  RETURN 100 DIV x
END F;
```

Diese Prozedur löst für `x=0` eine „DivisionByZero" exception aus, die durch den Funktionsplotter abgefangen werden muß:

```
PROCEDURE WriteF(l,r : INTEGER);
VAR i : INTEGER;
BEGIN
  FOR i:=l TO r DO
    TRY
      WriteInt(F(i),10);
    EXCEPT
      OF DivisionByZero THEN WriteString("Division durch Null") END
    END
    WriteLn
  END;
END WriteF;
```

Durch dieses **TRY**-Statement wird nur eine Division durch Null, nicht aber andere Ausnahmen, wie `Ctrl-C` o.ä. abgefangen. Diese Ausnahmen werden ohne Änderung weitergereicht.

Ausnahmen sollten so weit zurückgereicht werden, bis eine sichere Bearbeitung möglich ist.

Der **ELSE**-Teil der **TRY** Anweisung dient als Close-Teil, so daß bereits allozierte Strukturen noch freigegeben bzw. geschlossen werden können.

E.6.13 TRACK..END

Im Zuge des Resourcetrackings (Verfolgen der Systemresourcen durch das Laufzeitsystem, siehe auch „`Resources.def`") wurde **TRACK**..**END** eingeführt.

Durch **TRACK** wird eine neuer Kontext erzeugt und zum ActContext erklärt. Dieser Kontext wird bei Verlassen der Struktur (normal, exception oder **RETURN**) wieder entfernt.

E.7 Prozedurdeklaration

Eine Prozedurdeklaration besteht aus einem Prozedurkopf, in dem ihre Schnittstelle definiert wird, und einem Prozedurkörper, in dem der eigentliche Inhalt der Prozedur definiert wird. Eine Funktionsprozedur hat zusätzlich zu den normalen Parametern noch einen Rückgabetypen, der hinter einem Doppelpunkt definiert wird.

Innerhalb einer Prozedur können Typen, Variablen, Konstanten und weitere Prozeduren deklariert werden. Objekte, die innerhalb einer Prozedur deklariert sind, existieren nur, solange die Prozedur läuft. Sie sind auch nur innerhalb dieser sichtbar. Alle außen bekannten Bezeichner sind auch innerhalb bekannt, solange sie nicht durch neue Deklarationen überdeckt werden.

```
$$ paradeclar      ::= [VAR|REF] ident [IN expression]
                       {,ident [IN expression]}:
                       (qualident[:= expression])|(:=expression)
$$ formalparams    ::= ["("[paradeclar{;paradeclar}]")"[":"qualident]
$$ procedureheader ::= PROCEDURE|METHOD qualident formalparams ";"
$$ proceduredeclar ::= ([FORWARD] procedureheader)|(procedureheader
                       {import|typedef|vardeclaration|constdef|proceduredeclar}
                       BEGIN statementsequence END ident) ";"
```

Soll eine Prozedur verwendet werden, bevor ihr Prozedurrumpf implementiert werden kann, wenn sich also z.B. zwei Prozeduren gegenseitig benötigen, muß sie mit dem Schlüsselwort FORWARD vorwärts deklariert werden.

Parameter können mit einem Vorgabewert vorbelegt werden. Parameter mit Vorgabewert brauchen beim Prozeduraufruf nicht mit angegeben werden, statt dessen wird der Defaultwert vorgegeben:
Beisp:

```
PROCEDURE WriteInt(val : LONGINT;width : INTEGER := 0);
```

kann aufgerufen werden mit:

```
WriteInt(10,4) oder aber auch WriteInt(41)
```

Hat eine Prozedur viele Parameter, kann nicht immer davon ausgegangen werden, daß die zu überspringenden Parameter am Ende liegen. In diesem Fall muß die Zuweisung durch Schlüsselwörter (die Namen der Parameter) vorgenommen werden:

```
PROCEDURE New(VAR p       : ANYPTR;
                  chip,
                  clear   : BOOLEAN    := FALSE;
```

```
                 context : ContextPtr := NIL);

New(p);
New(p,TRUE,FALSE,MyContext);
New(p,clear:=TRUE);
New(p,clear:=TRUE,context:=MyContext);
...
```

Die Reihenfolge der Parameter muß eingehalten werden, auch dürfen nach Parametern mit Bezeichnern keine Positionalen mehr folgen.

Parameter können auf drei Arten übergeben werden, die bestimmen, wie innerhalb der Prozedur auf diese zugegriffen werden kann, welche Objekte übergeben werden können und ob der Parameter nach der Prozedur verändert sein kann.

E.7.1 Übergabe durch Wert (Call by value)

Als aktueller Parameter kann ein beliebiger Ausdruck angegeben werden. Das Ergebnis der Auswertung wird als Kopie an die Prozedur übergeben. Änderungen dieses Parameters innerhalb der Prozedur bleiben außen ohne Wirkung, da die Prozedur ja lediglich über eine Kopie verfügt.

Nachteilig bei diesem Verfahren ist, daß das Kopieren bei komplexen Typen relativ viel Zeit und Speicherplatz benötigt.

```
PROCEDURE WriteInt(i : INTEGER);
```

E.7.2 Übergabe durch Referenz (Call by reference)

Als aktueller Parameter kann ein adressierbares Objekt angegeben werden (Variablen, Konstanten, Parameter oder Zeigerziele). Die Prozedur bekommt die Adresse des Parameters übergeben, es wird im Gegensatz zu oben nicht kopiert. Der Parameter kann innerhalb der Prozedur nicht verändert werden, da ja sonst das Original verändert würde. Der aktuelle Parameter wird also mit Sicherheit nicht verändert.

Diese Übergabe lohnt sich für größere Typen (Arrays oder Records), auf keinen Fall aber für einfache Typen wie Zeiger oder Zahlen.

```
PROCEDURE WriteString(REF s : STRING);
```

E.7.3 Übergabe durch 'wirkliche' Referenz (mit Schreibzugriff)

Als aktueller Parameter kann eine Variable, ein Parameter oder ein Pointerziel angegeben werden. Die Prozedur erhält die Adresse dieses Objekts, und darf dieses auch verändern. Die Änderungen bleiben nach dem Prozeduraufruf erhalten, sind also 'richtige' Änderungen am Objekt.


```
PROCEDURE ReadString(VAR s : STRING);
```

E.7.4 Methoden

Hat man mehre Prozeduren, die die gleiche Funktion für verschiedene Typen erfüllen, ist es unangenehm, jeder Prozedur einen anderen Namen geben zu müssen. Daher besteht die Möglichkeit, beliebig viele Methoden mit dem gleichen Namen zu definieren, wenn Sie einen RECORD oder einen Zeiger auf einen RECORD als ersten Parameter erwarten.
Hier einige Beispiele aus dem Modul DosSupport:

```
METHOD Get(VAR data       : FileData;
           REF path       : STRING)

METHOD Get(VAR list       : DirList;
           REF path       : STRING;
           REF pattern    : STRING:="#?";
               type       := DirSelectType:{selectDirs,selectFiles};
               context    : ContextPtr := NIL);
```

Diese Methoden werden nicht direkt importiert und aufgerufen, sondern sie werden qualifiziert über eine Variable des Typs des ersten Parameters aufgerufen z.B.:

```
VAR
  Dir      : DirList;
  FileInfo : FileData;

BEGIN
  FileInfo.Get("s:startup-sequence");
  Dir.Get("DH0:");
```

Die Methoden von Objekten werden leicht anders definiert, hier wird der Name der Methode durch den Objekttypnamen genauer qualifiziert.

```
TYPE Counter = OBJECT
                    value : INTEGER;
                    METHOD Inc(by : INTEGER := 0);
                  END

METHOD Counter.Inc(by k : INTEGER);
```

Innerhalb einer Methode kann auf alle Instanzvariablen des eigenen Objektes direkt zugegriffen werden, ohne daß eine zusätzliche Qualifizierung nötig wäre.

```
METHOD Counter.Inc(by : INTEGER);
BEGIN
  INC(value,k)
END Inc;
```

E.7.5 Funktions-Prozeduren

Hat eine Prozedur oder Methode einen Rückgabewert, bezeichnet man sie als Funktion. Am Ende der Prozedur wird dieser mit **RETURN** zurückgegeben (siehe E.6.8). Handelt es sich bei dem Wert, der zurückgegeben werden soll, um einen komplexen Typen (Array/Record), so weist man diesen der Variablen RESULT zu, und verläßt die Funktion durch **RETURN**, ohne danach den Wert anzugeben.

Handelt es sich bei dem Rückgabetypen um einen offenen Typen (String/offenes Array), so muß man vor der Prozedurdeklaration den Schalter `$$OwnHeap:=TRUE` setzen (bei Prozeduren, die exportiert werden, genügt es, dies im Definitionsteil zu machen.). Außerdem muß man dafür sorgen, daß im Falle, daß das Ergebnis an einen VAR/REF-Parameter übergeben wird, ausreichend Platz auf dem Stack alloziert wird, um das Ergebnis dort zwischenzulagern. Hierzu dient die Standardprozedur

```
ALLOC_RESULT(range : LONGINT);
```

Der Parameter `range` gibt dabei den Bereich des Arrays/Strings an, für das Platz alloziert werden soll. Denken Sie dabei daran, daß Sie bei Strings `range` = Maximallänge+1 wählen, um Platz für das 0-Byte zu haben. Die zwei Bytes für die Länge bei einem String alloziert `ALLOC_RESULT` selbstständig, wenn es sich bei dem Rückgabetypen um einen String handelt.

E.8 Moduldeklaration

Es gibt drei Arten von Modulen: Programmodule, Definitionsmodule und Implementationsmodule. Immer ein Definitionsmodul gehört zu einem Implementationsmodul, in ihm werden die Schnittstellen zu anderen Modulen definiert.

```
$$ import       ::= (FROM qualident[AS ident] IMPORT ident{","ident}";")|
                    (IMPORT qualident[AS ident]{","qualident AS ident}";")
$$ modulebody   ::= {import|typedef|vardeclaration|constdef|proceduredeclar|
                    defmodule|implementmod}
                    [BEGIN statementsequence]
                    [CLOSE statementsequence]
```



```
$$ implementmod ::= IMPLEMENTATION MODULE ident ";"
                    modulebody END ident
$$ mainmodule   ::= MODULE ident ";" modulebody END ident
$$ defmodule    ::= DEFINITION MODULE ident
                    [("("ident : typedef")")|
                    ("=" qualident "(" typedef ")")]
                    {import|typedef|vardeclaration|constdef|procedureheader|
                    defmodule} END ident
$$ module       ::= mainmodule|implementmod|defmodule "."
```

Durch die Importklausel können Elemente anderer Module im eigenen Modul verwendet werden. Es ist sowohl ein qualifizierender als auch unqualifizerender Import möglich.
Beispiel:
Durch den Import

```
FROM InOut IMPORT WriteInt,Read;
```

werden dem Programm die Bezeichner „`WriteInt`“ und „`Read`“ direkt zur Verfügung gestellt, aber auch der Bezeichner „`InOut`“, so daß auch über „`InOut.xxx`“ auf andere Bezeichner des Objektes zugegriffen werden kann.

E.8.1 Generische Module

Viele Datenstrukturen und dazugehörige Algorithmen werden häufig in verschiedenen Kontexten und in Verbindung mit anderen Datenstrukturen benötigt (z.B. Listen, AVLBäume, Stacks etc.).

In einer streng und vor allem statisch getypten Sprache tritt das Problem auf, daß diese Strukturen jedesmal neu definiert und implementiert werden müssen. Diesem Problem wird durch Generizität abgeholfen.

Ein generisches Modul besitzt einen generischen Parameter. Dieser Parameter ist ein Zeigertyp, über den das Modul auch bereits Annahmen machen kann.

Ein derartiges Modul muß vor seiner Verwendung durch ein anderes Modul durch einen aktuellen Parameter ausgeprägt werden. Dieser Typ muß zum formalen Parametertyp passen.
Beispiel:

```
DEFINITION MODULE BiList(NodePtr : POINTER TO Node);

  TYPE
    Node  = RECORD
              pred,
```

```
          succ   : NodePtr
        END;
    List  = RECORD
          first,
          last   : NodePtr
        END;

    ApplyProc = PROCEDURE(n : NodePtr);     | Prozeduren von diesem Typ
                                            | können in der Liste arbeiten

  PROCEDURE Init(VAR l : List);
  PROCEDURE InsertTop(VAR l : List;n : NodePtr)
  PROCEDURE Apply(VAR l : List;apply : ApplyProc); | wendet eine Benutzer-
                                                   | prozedur an

END BiList;


IMPLEMENTATION MODULE BiList;

  PROCEDURE Init(VAR l : List);
  BEGIN
    l.first:=NIL;
    l.last :=NIL
  END Init;

  PROCEDURE InsertTop(VAR l : List;n : NodePtr);
  BEGIN
    n.pred:=NIL;
    n.succ:=l.first;
    IF l.first=NIL THEN
      l.last:=n
    ELSE
      l.first.pred:=n
    END;
    l.first:=n
  END InsertTop
```



```
  PROCEDURE Apply(VAR l : List;apply : ApplyProc);
  VAR n : NodePtr;
  BEGIN
    n:=l.first;
    WHILE n#NIL DO
      apply(n);
      n:=n.next;
    END;
  END Apply;

END BiList;
```

Das Implementations-Modul kann über die bekannten Elemente des Knotentyps (`pred`, `succ`) verfügen, da durch die Definition des generischen Parameters sichergestellt wird, daß nur ein Zeiger auf einen Nachfolger von Node in Frage kommt.

```
TYPE
  NamePtr  = POINTER TO NameNode;

  DEFINITION MODULE NameList = BiList(NamePtr);

TYPE
  NameNode = RECORD OF NameList.Node
               name,
               vorname : STRING(100);
               alter   : INTEGER;
             END;
```

Das Modul `NameList` verwaltet nun eine Liste derartiger `NameNodes`. Die Typsicherheit ist voll gegeben, da keine anderen Knotentypen zugelassen sind.

```
VAR
  MyNames  : NameList.List;
  name     : NameNode;
```

Die Prozeduren des Modules NameList können wie üblich importiert und benutzt werden. Werden jedoch mehrere Ausprägungen des selben generischen Moduls verwendet, treten Namenskonflikte auf. Diese könnten durch ständiges qualifizieren umgangen werden, was jedoch sehr aufwendig und fehleranfällig ist.

```
NameList.Init(MyNames);
New(name);
NameList.InsertTop(MyNames,name);
```

Um dies zu umgehen, kann eine Prozedur aus einem generischen Modul auch auf eine andere Art aufgerufen werden:

```
MyNames.Init;
New(name);
MyNames.InsertTop(name);
```

Für alle Ausprägungen eines generischen Moduls wird in einem Programm der selbe Code benutzt, es wird also kein Code vervielfältigt. Aus diesem Grund können auch nur Zeiger als generische Parameter dienen.

Werden für eine Implementation einer Datenstruktur gewisse Fähigkeiten des generischen Parameters (wie eine Ordnungsrelation) benötigt, so kann dies über Prozedurvariablen erreicht werden.

Es ist möglich, lokale Prozeduren als Parameter zu verwenden; es ist allerdings nicht möglich, einer Prozedurvariablen eine lokale Prozedur zuzuweisen.

```
PROCEDURE LasseAltern(VAR l : NameList.List;um : INTEGER);

  PROCEDURE altere(n : NodePtr);
  BEGIN
    INC(n.alter,um)
  END altere;

BEGIN
  l.Apply(altere);
END LasseAltern;
```

E.9 Compilerswitches

Compilerswitches (Schalter) können verwendet werden, um Teile eines Moduls bedingt zu compilieren. Ein Compilerswitch wird durch `$$` eingeleitet: `$$xxx:=yyy` setzt den Switch `xxx` auf den Ausdruck `yyy`. Einige Switches können abgekürzt werden.
Beispiel:

```
$$RangeChk:=FALSE
```


oder die Abkürzung (wie in einigen Pascal-Dialekten):

```
(* $R- *)
```

Die Zustände der stapelbaren Schalter können durch `$$xxx:=OLD` wieder in ihren vorherigen Zustand zurückgesetzt werden, die maximale Schachtelungstiefe eines jeden Schalters ist 16.

Zu jedem Projekt können bis zu 24 Schalter selbstdefiniert werden, dies geschieht im Project-Requester. Die Schalter können global, im Info-Requester oder im Quelltext gesetzt/gelöscht werden. Sie können zur bedingten Compilierung eingesetzt werden.

Näheres zu Compilerswitches entnehmen Sie bitte der Beschreibung des Compilers.

Anhang F

Glossar

Algorithmus

- Unter einem Algorithmus versteht man eine Verarbeitungsvorschrift, die so präzise ist, daß sie von einem Computer durchgeführt werden kann.
- Ein durch Regeln festgelegter Rechenvorgang, der häufig zyklisch wiederkehrende Gesetzmäßigkeiten aufweist. Algorithmen sind von besonderer Bedeutung für die Lösung mathematischer Aufgaben mit Hilfe von Rechenautomaten, da die Anzahl anzugebender Anweisungen weitaus geringer sein kann, als die Anzahl auszuführender Operationen.

  Ein Algorithmus gibt an, wie Eingabedaten schrittweise in Ausgabedaten umgewandelt werden.

  Determiniertheit: wird ein Algorithmus mit den gleichen Eingabewerten und Startbedingungen wiederholt, so liefert er stets das gleiche Ergebnis.

Anweisung (engl.: statement)

Anweisungen sind in der Regel die Teile eines Programms, welche den Zustand des Programms (Werte der Variablen, Inhalte der Ein-/Ausgabedateien) verändern, d. h. in denen bei Ausführung des Programms Daten verarbeitet werden (im Gegensatz zu Deklarationen). Weiterhin gibt es Kontrollstrukturen, die keine Daten ändern.

Elementare Anweisungen/ Sequenz/ Sprung/ Prozeduraufruf/ bedingte Anweisung/ Schleife/ Block.

Argument

Bei einer Funktionsprozedur wird häufig ein Wert mitgegeben, welcher in der Prozedur verwendet werden soll, z. B. berechnet `SQRT(5.0)` die Quadratwurzel von `5.0`. Hierbei ist `5.0` das Argument der Funktionsprozedur.

Ausdruck

- in der formalen Logik ist Ausdruck ein Oberbegriff für Aussage und Aussageform.
- Bei Programmiersprachen verwendet man Ausdruck als Synonym für Term. Die Regeln, nach denen Ausdrücke einer Programmiersprache gebildet werden, entnimmt man der jeweiligen Syntax.

Befehlsliste

Ist eine Folge von Programmbefehlen, oft auch Listing (Programmlisting) genannt.

`BEGIN` Schlüsselwort

Zeigt den eigentlichen Anfang eines Programms, Moduls oder einer Prozedur an.

Betrag (Absolutwert)

Mathematischer Begriff: `ABS(x)` wandelt eine Zahl `x` stets in eine positive Zahl um.

Bezeichner (engl.: identifier)

Bezeichner sind z. B. Namen für Variablen, Konstanten und Prozeduren. Ein Bezeichner ist also eine Zeichenfolge, welche in einem Programm zur eindeutigen Identifizierung eines Objektes dient. Ein Bezeichner darf innerhalb einer Programmeinheit, z. B. innerhalb einer Prozedur nur einmal deklariert werden. (Ausnahme: Methoden).

Block

Programmeinheit, die aus einer Folge von Anweisungen besteht. Blöcke dienen vorwiegend der klaren Strukturierung eines Programms. Ein Block wird mit den Schlüsselwörtern BEGIN und END geklammert:

```
BEGIN
   <Anweisungen>;
END
```

Bottom-up-Methode

Ist die Umkehrung der →Top-Down-Methode. Sie beginnt bei einem vorhandenen Computersystem. Man beginnt nun zur Lösung eines Problems bei den grundliegenden Funktionen. Komplexere Probleme werden dadurch gelöst, daß die grundlegenden Funktionen zu komfortableren Funktionen zusammengesetzt werden:

Beispiel:

- Computersystem: Addition zweier Zahlen.
- Entwurfsschritt: Realisierung der Multiplikation durch Addition

Grundsätzlich wird die Top-Down-Methode der Bottom-up-Methode vorgezogen.

case-sensitiv

Einige Programmiersprachen, wie auch CLUSTER, unterscheiden bei der Programmierung Groß- und Kleinschreibweise. In anderen Worten, die Bezeichner `test`, `tEsT` und `TEST` sind in CLUSTER verschieden. In PASCAL beispielsweise würden die drei o. a. Schreibweisen ein und denselben Bezeichner angeben.

Compiler (Übersetzer)

Der Compiler übersetzt (compiliert) den Quelltext, der bei uns in der Sprache CLUSTER verfaßt ist, in Maschinencode, der vom

Computer direkt ausgeführt werden kann. Ein Compiler macht also nichts anderes, als Programme aus der einen Programmiersprache in Programme einer anderen Programmiersprache zu übertragen.

Computerprogramm →Programm.

Deklaration (Vereinbarung)

Festlegung, welche Bedeutung und evtl. Wertebereich ein Bezeichner in einem nachfolgenden Programmtext besitzt:

- Deklaration von Konstanten
- Deklaration von Datentypen bzw. Datenstrukturen
- Deklaration von Variablen
- Deklaration von Prozeduren

Editor

Ein Editor dient zur Bearbeitung von Texten oder Graphiken (CLUSTER: Editor zur Bearbeitung des Programmtextes).

Interaktiver Editor: arbeitet im Dialog; ausschnittsweise Darstellung der im Speicher vorhandenen Daten.

Endlosschleife

Schleife, deren Ausführung nie abbricht. Die Ursache liegt häufig in einer fehlerhaften Abbruchbedingung. Man sagt auch: Die Schleife terminiert nicht.

`END` Schlüsselwort

Beendet einen Block (Modul, Prozedur, Schleife, ...)

Exponent mathematischer Begriff

Bei der im Computerbereich üblichen Gleitpunktdarstellung wird eine Zahl in Exponentialdarstellung angegeben. Z. B. $5.0 * 10^3$, hierbei ist der Exponent die 3.

Externspeicher

Ist Speicher, welcher nicht der Zentraleinheit (Computer) intern zugeordnet ist. Gehört zur Peripherie und kann z. B. eine externe Festplatte sein.

Formatfreiheit

Von Formatfreiheit spricht man, wenn es dem Compiler egal ist, wie weit der Abstand des einen Befehls von dem ihm folgenden ist. Es gibt Sprachen, bei denen es essentiell ist, in welcher Spalte welches Wort steht, da die Befehle abhängig von ihrer Position im Editor interpretiert werden. CLUSTER ist eine formatfreie Sprache.

Funktion (Funktionsprozedur)

Dies sind Prozeduren, in deren Prozedurkopf der Ergebnistyp anzugeben ist und in denen dem Bezeichner der Prozedur im Prozedurrumpf das Ergebnis zugewiesen wird. Funktionen berechnen einen Wert (Funktionswert) und geben ihn zurück, sobald das Schlüsselwort `RETURN` erreicht ist, z. B. `RETURN 15.8`.

Gleichung

Schreibt man zwischen zwei Terme ein Gleichheitszeichen, so entsteht eine Gleichung.

Grammatik

Damit sich zwei Menschen miteinander verständigen können, müssen sie eine gemeinsame Sprache sprechen. Sie müssen wissen, nach welchen Regeln die Sprache aufgebaut ist und was die einzelnen Zeichen bzw. Laute bedeuten. Die zulässige Form der Sätze einer Sprache nennt man Syntax. Zur Festlegung der Syntax einer Sprache verwendet man Grammatiken. Eine Grammatik ist eine Menge von Regeln, die bestimmen, welche Sätze zu einer Sprache gehören oder nicht. Natürliche Sprachen sind durch ihre Komplexität für die Verwendung auf Computersystemen ungeeignet. Aus diesem Grunde hat man einfachere Sprachen, die sog. Programmiersprachen entwickelt. Ihre Struktur kann präzise durch Grammatiken definiert werden.

Internspeicher

Ist Speicher, welcher zur Zentraleinheit (Computer) intern zugeordnet ist. In der Regel also normaler Hauptspeicher.

Konkatenation (Verkettung)

Die Konkatenation zweier Wörter ist das Wort, das sich durch Hintereinanderschreiben der beiden einzelnen Wörter ergibt. Z. B. „`BAUM`“ und „`HAUS`“ ergibt „`BAUMHAUS`“.

Konstante

Bezeichner mit einem festen Wert. Eine Konstante besitzt einen eindeutig festgelegten Bezeichner, einen Datentyp und einen festen Wert aus der Wertemenge des Datentyps. Konstanten müssen in CLUSTER vor Programmablauf definiert werden (Deklaration), wobei nur der Wert angegeben werden muß. Der Datentyp wird automatisch aus dem Wert erkannt. Wenn im Programm dann der Bezeichner der Konstanten benutzt wird, so wird an dieser Stelle immer der Konstantenwert eingesetzt. Nach der Deklaration darf einer Konstanten kein Wert mehr zugewiesen werden.

Mantisse mathematischer Begriff, →Exponent.

Bei der im Computerbereich üblichen Gleitpunktdarstellung wird eine Zahl in Exponentialdarstellung angegeben. Z. B. $3.1415 * 10^{-3}$, hierbei ist die Mantisse die 3.1415.

Modul

Ein Modul ist eine Zusammenfassung von Konstanten, Datentypen, Variablen und Prozeduren zu einer Einheit. Soll ein Modul von einem anderen Modul benutzt werden, so muß man angeben, welche Elemente des Moduls nach außen sichtbar sein sollen und welche nicht. Aus diesem Grunde wird ein Modul in zwei Teile aufgeteilt: Einen Definitionsteil und einen Implementationsteil. Im Definitionsmodul wird angegeben, welche Modulteile nach außen exportiert werden, d. h. welche Bezeichner von Konstanten, Datentypen, Variablen und Prozeduren für andere Module sichtbar sind. Die Liste dieser Bezeichner nennt man *Exportschnittstelle* eines Modules. Im Implementierungsmodul steht dann die eigentliche Programmierung der Prozeduren.

`MODULE` Schlüsselwort

Zeigt den Anfang eines CLUSTER-Programms an.

Modulodivision

Bei der Modulodivision wird der ganzzahlige Rest einer Division augegeben. Z. B. ergibt `10 MOD 3 = 1`, da die `3` dreimal in die `10` reingeht und sich damit ein Rest von `1` ergibt.

Objektcode (Objektprogramm)

Programm in Maschinensprache, das vom Compiler erzeugt wurde. Es handelt sich also um ein spezielles Zielprogramm.

Operationen Arbeitsvorgänge im Computer

arithmetische O.

logische O. (UND...)

Transporto. (Speicherbereiche kopieren...)

Eingabe-/Ausgabeo.

O. zur Steuerung des Kontrollflusses (Sprünge...)

sonstige O. (Unterbrechungen...)

Operatoren

Funktionszeichen für arithmetische Operationen, z. B. `+`. `-`, `*`, `\`, ...

Parameter

Platzhalter in einer Programmeinheit, der erst bei der konkreten Verwendung (Aufruf) der Programmeinheit, z. B. einer Prozedur festgelegt wird.

Portabilität (Übertragbarkeit)

Eigenschaft von Programmen, ohne große Änderungen auf unterschiedlichen Rechnersystemen ausgeführt werden zu können. Eine geläufige Methode ist die Zweiteilung von Programmen in einen maschinenunabhängigen und einen maschinenabhängigen Teil, der bei einer Übertragung des Programms auf ein anderes Rechnersystem neu geschrieben werden muß.

Potenz mathematischer Begriff

Potenzieren heißt, eine Zahl wiederholt als Faktor setzen. Man schreibt z. B.

$$3 \cdot 3 \cdot 3 \cdot 3 = 3^4 = 81$$

Oder allgemein ausgedrückt: Die Potenz a^n ist hiernach als Abkürzung für ein Produkt von n gleichen Faktoren aufzufassen.

`PROCEDURE` Schlüsselwort

Zeigt den Anfang einer Prozedur oder einer Funktionsprozedur an. Z. B.:

```
PROCEDURE Test(REF wert1:INTEGER;wert2:BOOLEAN);
```

Programm

Ein Programm ist die Formulierung eines Algorithmus oder mehrerer Algorithmen in einer Programmiersprache. Im Gegensatz zu Algorithmen sind Programme wesentlich konkreter: Sie sind im Formalismus einer Programmiersprache verfaßt. Sie nehmen Bezug auf eine bestimmte Darstellung der verwendeten Daten. Sie sind auf einem Computer ausführbar.

Programmbibliothek

Eine Sammlung von Programmodulen, die in anderen Programmen verwendet werden können, ähnlich wie man ein Buch in einer Bibliothek ausleihen kann.

Programmiersprache →Sprache

Prozedur (Unterprogramm, engl. procedure / subroutine):

Hierdurch kann jede als Programm formulierte Vorschrift zu einer elementaren Anweisung in einem anderen Programm werden. Darüberhinaus dienen Prozeduren der Zerlegung und Strukturierung umfangreicher Programme, und sie erlauben die Verwendung rekursiver Techniken. Weitere Ausführungen finden Sie in dem Kapitel unter 3.13.1 ab Seite 75.

Quelltext (source code) / Quellprogramm

Programm, das von einem anderen Programm verarbeitet werden soll. Speziell bezeichnet man das Programm, das von einem Übersetzer in ein Zielprogramm übersetzt werden soll, als Quellprogramm. Ferner versteht man unter dem Quellprogramm oder dem Quelltext die Originalversion eines übersetzten Programms. Korrekturen und Veränderungen nimmt man am Quellprogramm vor, wenn man Fehler oder unerwünschte Verhaltensweisen während der Ausführung eines Programms festgestellt hat.

Rekursion

Definition einer Funktion durch sich selbst. Aufruf einer Funktion oder Prozedur durch sich selbst.

reserviertes Wort →Schlüsselwort

Schleife

Folge von Anweisungen, die mehrfach durchlaufen werden kann (Iteration). Ihrem Aufbau nach unterteilt man die Schleife in den Schleifenkopf und den Schleifenrumpf. Der Schleifenkopf enthält die Kontrollinformation für die Anzahl der Schleifendurchläufe, die mehrfach zu durchlaufende Anweisungsfolge heißt Schleifenrumpf. Zählschleife / Bedingte Schleife

Schlüsselwort

In fast allen Programmiersprachen sind Zeichenfolgen definiert, die eine in der Sprache genau festgelegte Bedeutung haben, wie z. B. in CLUSTER die Zeichenfolgen „`MODULE`“, „`VAR`“, „`WHILE`“ oder „`END`“. Solche Zeichenfolgen bezeichnet man als Schlüsselwörter der Programmiersprache. Sie dürfen in der Regel – und so auch bei CLUSTER – nicht als Bezeichner in Programmen verwendet werden. In diesem Fall bezeichnet man die Schlüsselwörter als reservierte Wörter der Programmiersprache.

Schnittstelle

- In der Elektrotechnik versteht man unter einer Schnittstelle einen Verbindungspunkt zwischen verschiedenen elektronischen Bauteilen oder Geräten.
- In der Computertechnik ist damit in der Regel ein Anschlußstecker z. B. für eine Peripheriegerät gemeint.
- Bei bestimmten Computersprachen versteht man darunter eine Art Konvention, die ein Programmierer beachten muß, wenn er ein Modul und die darin definierten Prozeduren verwenden möchte. Bei Modula 2 und CLUSTER ist die Programmierschnittstelle im Definitionsmodul definiert.

Software-Engineering

Anwendung von Prinzipien, Methoden und Techniken auf den Entwurf und die Implementierung von Programmen und Programmsystemen.

Sprache

- Natürliche Sprachen:

  werden von Menschen zum Informationsaustausch und zu allen Zwecken der Kommunikation verwendet. Die Beherrschung einer Sprache und ihrer Ausdrucksmöglichkeiten beeinflußt stark die Vorstellungswelt und die Denkweise von Menschen. Zu jeder natürlichen Sprache existiert in der Regel eine Schriftsprache.
- Künstliche Sprachen:

  wurden Ende des 19. Jhdts. entwickelt, um Fakten, Denkabläufe, Schlußfolgerungen beschreiben und analysieren zu können. Neben diesen logischen Kalkülen entstanden mit der Entwicklung von Rechenanlagen seit 1940 Programmiersprachen, die ebenfalls präzise definiert werden mußten. Programmiersprachen regeln den Umgang mit Datenverarbeitungsanlagen. Künstliche Sprachen werden nach Regeln aufgebaut (→Syntax, →Grammatik), und ihre Wörter und Sätze besitzen eine wohldefinierte Bedeutung. Während sich bei natürlichen Sprachen die Wörter, Regeln und Bedeutungen im Laufe der Jahre ändern, besitzen künstliche Sprachen ein festes endliches Grundvokabular und eine feste Syntax und Semantik.

Struktur

Wenn man logisch zusammengehörende Daten unter einem Bezeichner zusammenfaßt, wird dies als Struktur bezeichnet. In Modula 2 und CLUSTER geschieht dies mithilfe eines `RECORD`s.

Strukturierte Programmierung (systematisches/methodisches Programmieren)

Programmiermethode, bei der das vorgegebene Problem in Teilprobleme und die Beziehungen zwischen diesen Teilproblemen (→Schnittstellen) zerlegt wird. In weiteren Verfeinerungsschritten werden die Teilprobleme in der gleichen Weise behandelt, bis die Aufgaben schließlich so klein geworden sind, daß man sie ohne weitere Verfeinerung lösen kann.

Durch Zusammensetzen („Integration") der Einzellösungen erhält man dann eine Lösung für das Ausgangsproblem (→Software-Engineering). Wichtig ist hierbei:

- Man geht von dem umfassenderen, abstrakter beschriebenen Problem zu einfacheren und konkreteren Teilproblemen über (→Top-down-Methode).
- Jedes Teilproblem wird durch die Zerlegung vollständig beschrieben und kann unabhängig von anderen Teilproblemen weiterbearbeitet werden (alle Abhängigkeiten werden vollständig durch die Schnittstellen dargestellt).
- Die Zerlegung erfolgt problemorientiert. Ein Teilproblem wird insbesondere nicht danach untersucht und zergliedert, wie es später bestmöglich von einem Rechnersystem bearbeitet werden kann. Hierdurch werden maschinenunabhängige Lösungen begünstigt (→Portabilität).

- Die Struktur des Ausgangsproblems, die sich in der fortschreitenden Zerlegung widerspiegelt, findet sich in der Lösungsstruktur und somit im späteren Programm wieder! Hierdurch werden Programme lesbar und verständlich, und man kann Programme leichter korrigieren, wenn Fehler in der Programmentwicklung aufgetreten sind oder wenn sich Rahmenbedingungen verändert haben.
- Teillösungen existieren unabhängig von dem speziellen Problem und können anderweitig mitverwendet (→Programmbibliothek), einzeln getestet und gegebenfalls als korrekt bewiesen werden (→Verifikation).

Ein systematisches Strukturieren ist bei umfangreichen Problemen unerläßlich. Ebensowenig, wie man ein Haus ohne sorgfältig entwickelten Bauplan erstellen kann, kann man Anwendungs- und Systemprogramme, die aus vielen tausend Anweisungen bestehen, nicht ohne methodisches Vorgehen schreiben.

Syntax Regelsystem einer Sprache

Die Syntax ist praktisch die Grammatik einer Programmiersprache. Eine Sprache wird durch eine Folge von Zeichen, die nach bestimmten Regeln aneinandergereiht werden dürfen, definiert. Den hierdurch beschriebenen formalen Aufbau der Sätze oder Wörter, die zur Sprache gehören, bezeichnet man als ihre Syntax. Die Syntax einer Programmiersprache legt fest, welche Zeichenreihen korrekt formulierte Programme der Sprache sind und welche nicht. Um präzise feststellen zu können, ob ein Programm syntaktisch korrekt ist, muß man zuvor die Syntax der Sprache formal beschreiben.

Terme sind gewisse sinnvolle mathematische Zeichenreihen. Alle Zahlen (Konstanten) und Variablen sind Terme. Enthält ein Term eine Variable, so geht er bei Ersetzung der Variablen durch Elemente der Grundmenge (Definitionsmenge) in eine Zahl über.

Top-down-Methode

Durch schrittweise Verfeinerung wird das Ausgangsproblems in einfacher zu lösende Teilprobleme zerlegt. Hierbei wird bei jedem Entwurfsschritt festgelegt, was die Untermodule (Teilprogramme) leisten sollen. Die Funktionen der Module einer Ebene sollen nur durch die Funktionen jener Module der direkt darunterliegenden Ebene verwirklicht werden.

Variable (Platzhalter, Leerstelle) nennt man ein Symbol, für welches Elemente einer Grundmenge eingesetzt werden dürfen. Meistens benutzt man für Variablen Buchstaben. Beim Einsetzen von Elementen der Grundmenge für die Variablen muß man darauf achten, daß gleiche Variable auch durch gleiche Elemente ersetzt werden. Man beachte, daß zu verschiedenen Variablen auch verschiedene Grundmengen gehören können.

In einer Gleichung ist nach den Elementen aus der Grundmenge gefragt, die bei Einsetzen für die Variablen eine wahre Aussage liefern. Die Gleichungsvariablen nennt man manchmal auch Unbestimmte oder Unbekannte. Bei Funktionsgleichungen, z. B. $y = x + 2x + 1$, unterscheidet man abhängige und unabhängige Variable. In obigem Beispiel ist x die unabhängige und y die abhängige Variable.

Verifikation

Formaler Nachweis von Eigenschaften von Programmen oder Programmteilen. Hierbei wird versucht zu beweisen, ob ein Programm in allen Belangen korrekt arbeitet.

Index