KCPASCAL

PASCAL-STECKMODUL

DOKUMENTATION

als Manuskript gedruckt

Auf der Grundlage des TURBO - PASCAL Handbuches von Borland Inc. (USA)

Betreuer:	Entwickler und Bearbeiter
PROF. H. K. ROTH	DIPL. ING. M. STURM DIPL. ING. R. BERGMANN DIPL. ING. W. TISCHER

in elektronisch lesbare Form gebracht von Ulrich Zander / 2006
Achtung!
Das Original hat viele Grammatik- und Rechtschreibfehler, die teilweise von mir geändert wurden. Volker Pohlers hat die Sachfehler korrigiert.

Inhaltsverzeichnis

Einführung

Vor Ihnen liegt das Handbuch für das Programm KCPASCAL in der Form, wie es auf dem KC 85/1 oder KC 87 läuft. Obwohl darin viele Beispiele beschrieben werden, ist es nicht als Lehr- oder Textbuch gedacht, und deshalb sollten Sie wenigstens Grundkenntnisse von Pascal besitzen.

Pascal ist eine allgemein anwendbare 'High-Level'-Programmiersprache, die von Professor Nikolaus Wirth der Techn. Universität Zürich entwickelt und nach Blaise Pascal, dem berühmten Philosophen und Mathematiker aus dem 17. Jahrhundert benannt wurde.

Diese 1971 veröffentlichte Programmiersprache sollte mittels des strukturierten Programmierens eine systematische Annäherung an die Arbeit mit Computern erlauben. Seitdem wird Pascal auf fast allen Computern in fast allen Anwendungsbereichen genutzt. Heute ist Pascal eine der meist-benutzten, 'High-Level'-Programmiersprachen, sowohl im Lehr- als auch im professionellen Programmbereich.

KCPASCAL
ist eng an das Standard-PASCAL von N. Wirth angelehnt, wie es in dem "Pascal User Manual an Report" beschrieben wird.

Hinzu kommen einige Erweiterungen:
- Absolute Adressierung der Variablen
- Bit Byte Manipulierung
- Direkter Zugriff auf die CPU und die Datenports
- Dynamische Strings
- Freie Anordnung der Sektionen innerhalb des Deklarationsteils
- Volle Unterstützung des Betriebssystems
- Erzeugung von In-line Maschinencode
- Logische Operationen bei ganzen Zahlen
- Strukturierte Konstanten
- Typenwandlungs-Funktionen

Dazu kommen viele neue Standardprozeduren und -funktionen, die die Handhabbarkeit von KCPASCAL erhöhen.
In der vorliegenden Version ist Dateiarbeit mit Kassette nicht möglich.

1. Die Benutzung von KCPASCAL

Dieses Kapitel behandelt den Gebrauch von KCPASCAL, insbesondere den integrierten Editor.

1.1. Start von KCPASCAL

Geben Sie das Kommando:

KCPASCAL

auf Ihrem KC 87 ein. Das System antwortet mit der folgenden Meldung:

```
KC-Pascal     V NNXX
(c) W.Tischer D.Poenigk 1987
```

Bild 1-1: Einschaltmeldung

Daraufhin erscheint das Hauptmenü von KCPASCAL.

```
Work file:

Edit    Compile    Run    Save
Quit    compiler Options

Text:       0 bytes (0500-0500)
Frei:   12615 bytes (0502-3648)
>
```

Bild 1-2: Haupt-Menü

Das Menü zeigt Ihnen die verfügbaren Kommandos, wenn Sie mit KCPASCAL arbeiten. Das Kommando wird ausgeführt, wenn Sie den entsprechenden Großbuchstaben eingeben. Drücken Sie nicht <ENTER>, da das Kommando sofort nach Eingabe des Großbuchstabens ausgeführt wird. Das Menü verschwindet zwar, wenn Sie mit dem System arbeiten, aber es läßt sich leicht wieder auf den Bildschirm holen: Geben Sie einen unerlaubten Befehl ein, z.B. <ENTER> oder <SPACE>.

Die folgenden Abschnitte beschreiben jedes Kommando im Detail.

1.2 Das Hauptmenü

1.2.1 Die Wahl der Arbeitsdatei

Das W-Kommando wird zur Auswahl der zu bearbeitenden Datei benutzt. Nach seiner Eingabe erscheint auf dem Bildschirm:

Workfilename:

Sie antworten mit einem legalen Dateinamen, d.h. mit einem Namen, der aus 1 bis 8 Buchstaben, optional einem Punkt und möglicherweise dem Dateityp (max. 3 Buchstaben) besteht:

FILENAME.TYP

Wenn Sie einen Namen ohne Punkt und Dateityp eingeben, wird automatisch der Typ .PAS generiert. Es ist möglich, einen Dateinamen ohne Typ anzugeben, indem hinter den Namen nur der Punkt gesetzt wird.

Beispiele:

```
PROGRAM      wird zu PROGRAM.PAS
PROGRAM.     wird nicht verändert
PROGRAM.FIL  wird nicht verändert
```

Die Dateitypen .BAK und .COM sollten vermieden werden, da KCPASCAL diesen

Namen zu einem bestimmten Zweck verwendet.

Nachdem die Arbeitsdatei angegeben wurde, kann sie von der Kassette eingelesen werden. Wenn nicht, erscheint als Meldung "Neu". Wenn Sie eine soeben editierte Datei noch nicht abgespeichert haben, erscheint die Meldung:

Workfile FILENAME.TYP sichern? (J/N)?

Das soll Sie davor warnen, eine neue Datei in den Speicher zu laden und damit diejenige, an der Sie gerade arbeiten, zu überschreiben und zu zerstören. Antworten Sie J zur Speicherung und N zur Löschung.

Der neue Name der Arbeitsdatei erscheint im Menü nach der nächsten Aktualisierung, d.h., wenn Sie <SPACE> eingeben.

1.2.2 Das Edit-Kommando

Das E-Kommando ruft den integrierten Editor auf, um die Datei, die als Arbeitsdatei bezeichnet wurde, editieren zu können. Wenn keine Arbeitsdatei angegeben ist, werden Sie aufgefordert, dies zu tun. Das Menü verschwindet und der Editor ist aktiviert.

1.2.3 Das Compile-Kommando

Das C-Kommando aktiviert den Compiler. Die Compilierung kann jederzeit durch Drücken einer Taste unterbrochen werden.

Die Compilierung endet entweder mit einem Programm, das im Arbeitsspeicher verbleibt oder mit einer .COM-Datei. Diese Wahl können Sie im Optionen-Menü des Compilers treffen. Die Voreinstellung beläßt das Programm im Arbeitsspeicher.

1.2.4 Das Run-Kommando

Falls schon ein compiliertes Programm im Speicher ist, wird dieses aktiviert. Wenn nicht, findet die Compilierung automatisch statt.

1.2.5 Das Save-Kommando

Das S-Kommando sichert die aktuelle Arbeitsdatei auf Kassette.

1.2.6 Das Quit-Kommando

Das Q-Kommando wird benutzt, um das KCPASCAL-System zu verlassen. Wenn die Arbeitsdatei nach dem Laden bearbeitet wurde, werden Sie gefragt, ob Sie sie abspeichern wollen, bevor Sie beenden.

1.2.7 Compiler-Optionen

Das O-Kommando wählt ein Menü an, in dem Sie einen Überblick über einige voreingestellte Werte des Compilers erhalten und diese ändern können.

Außerdem bietet es eine hilfreiche Funktion zur Auffindung von Laufzeitfehlern in Programmen.

1.3 Der PASCAL-Editor

Der integrierte Editor ist ein Bildschirmeditor, der speziell zur Programmtexterstellung geschaffen wurde. Falls Sie mit dem Textprogramm TP vertraut sind, benötigen Sie keine weitere Einführung in die Handhabung des Editors, da die Standardfunktionen exakt denen von TP entsprechen. Es gibt einige kleinere Unterschiede. Der KCPASCAL-Editor besitzt darüberhinaus einige Erweiterungen. Diese werden im Kapitel 1.4 besprochen.

Die Benutzung des Editors ist sehr einfach: Wenn Sie eine Arbeitsdatei definiert haben und E eingeben, verschwindet das Menü und der Editor ist aktiviert. Wenn die Arbeitsdatei auf Kassette ist, kann sie geladen werden und die erste Seite des Textes erscheint. Wenn es eine neue Datei ist, ist der Bildschirm ab der Statuszeile leer.

Sie verlassen den Editor und kehren durch Drücken von CONTR-K-D zum Menü zurück. Mehr darüber erfahren Sie später.

Der Text wird, wie auf einer Schreibmaschine, auf der Tastatur eingegeben. Um eine Zeile zu beenden, drücken Sie <ENTER>. Wenn Sie Ihren Bildschirm vollgeschrieben haben, wird die oberste Zeile nach oben weggeschoben.

1.3.1 Die Statuszeile

Die oberste Zeile auf dem Bildschirm ist die Statuszeile. Sie enthält folgende Informationen:

```
Z n      S n     Einf.    Tab  FILENAME.TYP
```

Bild 1-5: Statuszeile des Editors

Z n
Zeigt die Nummer der Zeile an, in der sich der Cursor befindet, vom oberen Bildschirmrand her gezählt.

S n
Zeigt die Nummer der Spalte an, in der der Cursor steht, von der linken Seite her gezählt.

Einf.
Hier wird angezeigt, daß die Zeichen, die über die Tastatur eingegeben werden, an der Cursorposition eingefügt werden. Der bereits existierende Text rechts vom Cursor wird entsprechend der Länge des neuen Textes nach rechts verschoben. Wenn Sie das Kommando Einf. aus (CONTR-V nach Voreinstellung) eingeben, erscheint stattdessen Ueb. (Überschreiben). Nun wird der Text an der Cursorposition überschrieben und nicht nach rechts verschoben.

Tab
Gibt an, daß die automatische Tabulierfunktion eingeschaltet ist. Mit dem Kommando auto-Tab ein/aus (CONTR-Q CONTR-I nach Voreinstellung) kann ein-

oder ausgeschaltet werden.

FILENAME.TYP
Name und Typ der Datei, die editiert werden soll.

1.3.2 Editier-Kommandos

Wie schon erwähnt, wird der Text wie auf einer Schreibmaschine geschrieben. Da Sie aber auf einem Computer arbeiten, werden Ihnen einige Editiermöglichkeiten geboten, die die Textbearbeitung, und in diesem Fall das Schreiben von Programmen, sehr vereinfachen.

Der KCPASCAL-Editor erlaubt bis zu 45 Editier-Kommandos, die den Cursor bewegen, durch den Text blättern, Textstrings finden und ersetzen usw. Die Kommandos können in vier Gruppen unterteilt werden:

Kommandos zur Cursorsteuerung
Kommandos zum Einfügen und Löschen
Blockkommandos
Weitere Kommandos

Jede dieser Gruppen enthält in sich zusammenhängende Kommandos, die in den folgenden Abschnitten einzeln besprochen werden. Die folgende Tabelle gibt einen Überblick über die vorhandenen Kommandos:

Cursorsteuerungs-Kommandos

^S	Zeichen links	^QE	Oberer Bildschirmrand
^D	Zeichen rechts	^QR	Dateianfang
^A	Wort links	^QC	Dateiende
^F	Wort rechts	^QS	Zeile links
^E	Zeile nach oben	^QD	Zeile rechts
^W	Rollen nach unten	^QB	Blockanfang
^Z	Rollen nach unten	^QK	Blockende
^R	Seite nach oben	^QP	Letzte Cursorposition
^C	Seite nach unten		

Einfüge- und Löschkommandos

^V	Einfügen-Modus an/aus	^T	Wort rechts löschen
^N	Zeile einfügen	^G	Zeichen unter dem Cursor löschen
Löschen bis Zeilenende		^Y	Zeile löschen
<DEL>	Zeichen links löschen		

Block-Kommandos		**Verschiedene Kommandos**	
^KB	Blockanfang markieren	^KD	Ende des Editierens
^KK	Blockende markieren	^I	Tabulator
^KT	einzelnes Wort markieren	^QI	Auto Tab an/aus
^KC	Block kopieren	^QL	Zeile sichern
^KV	Block bewegen	^QF	Suchen
^KY	Block löschen	^QA	Suchen/Tauschen
^KR	Block von Kassette lesen	^L	Letztes Suchen wiederholen
^KW	Block auf Kassette speichern		
^KH	Block verdecken/zeigen		

Tabelle 1-2 Editier-Kommandos

In so einem Fall lernt man am besten, indem man arbeitet. Starten Sie

KCPASCAL, bestimmen Sie eines der Programmbeispiele als Arbeitsdatei und geben Sie "E" ein, um editieren zu können. Dann probieren Sie die Kommandos aus, so wie Sie sie lesen.

Jede der folgenden Beschreibungen besteht aus einer Kopfzeile, die das Kommando bezeichnet, gefolgt von der Angabe der Tasten, die das Kommando ausführen.

1.3.3 Eine Bemerkung zu den Kontrollzeichen

Alle Kommandos sind so angelegt, daß die Kontrollzeichen verwenden. Ein Kontrollzeichen ist ein spezielles Zeichen, das von Ihrer Tastatur erzeugt wird, indem Sie die <CONTR>-Taste gleichzeitig mit einem Buchstaben von A bis Z drücken.

Die CONTR-Taste arbeitet wie die <SHIFT>-Taste. Wenn Sie <SHIFT> und ein A gleichzeitig drücken, erscheint auf dem Bildschirm ein A. Wenn Sie die <CONTR>-Taste und A drücken, erhalten Sie ein Kontroll-A (CONTR-A).

1.3.4 Bevor Sie anfangen: Wie Sie wieder aufhören können

Das Kommando, mit dem Sie den Editor verlassen und wieder in das Hauptmenü zurückkehren können, ist CONTR-K-D. Dieses Kommando sichert aber nicht automatisch die Datei. Dies muß vom Menü aus mit dem Save-Kommando geschehen.

1.3.5 Cursorsteuerungs-Kommandos

1.3.5.1 Kommandos der Grundbewegungen

Das Erste, was man über einen Editor wissen muß, ist, wie man den Cursor auf dem Bildschirm bewegt. Der KCPASCAL-Editor benutzt dazu eine Gruppe spezieller Kontrollzeichen, namentlich die Kontrollzeichen A, S, D, F, E, R, X und C.

```
   E
S     D
   X
```

Die Lage der vier Buchstaben zeigt schon optisch an, daß CONTR-E den Cursor nach oben, CONTR-X nach unten, CONTR-S nach links und CONTR-D nach rechts bewegt. Versuchen Sie nun den Cursor mit diesen vier Steuerzeichen auf dem Bildschirm hin- und herzubewegen. Da die Tastatur Wiederhol-Tasten hat, können Sie durch anhaltendes Drücken der CONTR- und einer der vier anderen Tasten den Cursor sehr schnell über den Bildschirm bewegen.

Nun schauen wir uns einige Erweiterungen zu diesen Bewegungen an:

```
     E   R
A   S   D   F
     X   C
```

Die Lage von CONTR-R neben CONTR-E impliziert, daß damit der Cursor nach oben bewegt wird, nur nicht um eine Zeile, sondern um eine ganze Seite. Entsprechend bewegt CONTR-C den Cursor um eine ganze Seite nach unten.

Entsprechendes gilt für CONTR-A und CONTR-F: CONTR-A bewegt den Cursor um ein ganzes Wort nach links, CONTR-F um ein ganzes Wort nach rechts.

Die beiden letzten Grundbewegungen steuern den Cursor nicht nur auf dem Bildschirm, sondern lassen den ganzen Bildschirm in der Datei nach oben oder unten rollen:

```
  W   E   R
A   S   D   F
  Z   X   C
```

CONTR-W rollt in der Datei nach oben (die Zeilen des Bildschirms bewegen sich nach unten) und CONTR-Z rollt nach oben (die Zeilen des Bildschirmes bewegen sich nach oben).

Zeichen nach links **CONTR-S**

Bewegt den Cursor ein Zeichen nach links, ohne dieses Zeichen zu verändern. <BACKSPACE> kann die gleiche Funktion übernehmen. Das Kommando führt den Cursor nicht über das Zeilenende hinaus, d.h., wenn der Cursor den linken Rand des Bildschirmes erreicht hat, stoppt er.

Zeichen nach rechts **CONTR-D**

Bewegt den Cursor ein Zeichen nach rechts, ohne dieses Zeichen zu verändern Das Kommando führt den Cursor nicht in die nächste Zeile, d.h., wenn der Cursor den rechten Rand des Bildschirms erreicht, beginnt der Text spaltenweise nach links auszuwandern, bis der Cursor die Spalte 128, den äußersten rechten Rand, erreicht, wo er stoppt.

Wort nach links **CONTR-A**

Bewegt den Cursor zum Wortbeginn nach links. Ein Wort ist als eine Sequenz von Zeichen definiert, die von den Zeichen: <space>.<>.: 0 S begrenzt wird. Dieses Kommando gilt über das Zeichenende hinaus.

Wort nach rechts **CONTR-F**

Bewegt den Cursor zum Wortbeginn nach rechts. Zur Definition von Wort: siehe oben. Führt den Cursor auch in die nächste Zeile.

Zeile nach oben **CONTR-E**

Bewegt den Cursor um eine Zeile nach oben. Wenn er die oberste Zeile erreicht hat, rollt der Bildschirm um eine Zeile nach unten.

Zeile nach unten **CONTR-X**

Bewegt den Cursor eine Zeile nach unten. Wenn der Cursor die unterste Zeile erreicht hat, rollt der Bildschirm um eine Zeile nach oben.

Aufwärts rollen **CONTR-W**

Rollt den Cursor gegen den Anfang der Datei, jeweils um eine Zeile (d.h., der ganze Bildschirminhalt rollt nach unten). Der Cursor bleibt auf der Zeile, bis diese das untere Ende des Bildschirms erreicht.

Abwärts rollen **CONTR-Z**

Rollt den Cursor gegen das Ende der Datei, jeweils um eine Zeile (d.h., der ganze Bildschirminhalt rollt nach oben). Der Cursor bleibt auf der Zeile, bis diese den oberen Rand des Bildschirms erreicht.

Seite nach oben **CONTR-R**

Bewegt den Cursor um eine Seite nach oben, mit einer Überlappung von einer Zeile, d.h., er bewegt sich um eine Bildschirmseite, abzüglich einer Zeile, im Text zurück.

Seite nach unten **CONTR-C**

Bewegt den Cursor um eine Seite nach unten, mit einer Überlappung von einer Zeile, d.h., er bewegt sich um eine Bildschirmseite, abzüglich einer Zeile, im Text nach vorn.

1.3.5.2 Kommandos für erweiterte Bewegungen

Die eben besprochenen Kommandos erlauben es Ihnen, sich frei im Text zu bewegen, sie sind leicht zu erlernen und zu verstehen. Wenn Sie einige Zeit damit arbeiten, merken Sie, wie einfach es ist.

Wenn Sie sie beherrschen, werden Sie den Cursor auch einmal schneller bewegen wollen. Der Editor von KCPASCAL stellt 5 Kommandos zur Verfügung, die es erlauben, sehr schnell zu den äußeren Enden der Zeilen, des Textes und zur letzten Cursorposition zu gelangen.

Diese Kommandos erfordern die Eingabe zweier Zeichen, CONTR-Q und dann eines der folgenden Kontrollzeichen (CONTR-)S, D, E, X, R und C. Die ersten vier wurden schon vorher besprochen:

```
  E   R
S   D
  X   C
```

d.h., CONTR-Q-R bewegt den Cursor zum Beginn, CONTR-Q-C zum Ende des Textes, CONTR-Q-S zum äußersten linken Ende und CONTR-Q-D zum äußersten rechten Ende der Zeile. CONTR-Q-E bewegt den Cursor an den oberen Rand des Bildschirms, CONTR-Q-X an den unteren Rand des Bildschirms.

CONTR-Q zusammen mit einem B, K oder P ermöglicht es Ihnen, innerhalb der Datei weit zu springen.

Blockanfang **CONTR-Q-B**

Bewegt den Cursor an die Stelle der Blockanfangs-Markierung, die mit CONTR-K-K gesetzt wurde (deshalb das Q-B).

Blockende **CONTR-Q-K**

Bewegt den Cursor an die Stelle der Blockende-Markierung, die mit CONTR-K-K gesetzt wurde (deshalb das Q-K).

Letzte Cursorposition **CONTR-Q-P**

Bewegt den Cursor an seine vorherige Position (P soll an Position erinnern). Beispielsweise, um den Cursor nach einem 'Sichern' oder nach einem 'Suchen/Tauschen' auf seine letzte Position zurück zu bewegen.

1.3.6. Einfüge- und Lösch-Kommandos

Einfüge-Modus an/aus **CONTR-V**

Wenn Sie Text eingeben, können Sie zwischen zwei Eingabemodi wählen: Einfügen und Überschreiben. Der bei Aufruf des Editors voreingestellte Einfügemodus erlaubt es Ihnen, in bestehenden Text neuen Text einzufügen. Der bestehende Text rechts vom Cursor wird dabei weiter nach rechts verschoben.

Der Modus 'Überschreiben' kann gewählt werden, wenn Sie den alten Text durch einen neuen ersetzen wollen. Die neu eingegebenen Zeichen ersetzen dabei die, die sich gerade unter dem Cursor befinden.

Zwischen den Modi schalten Sie mit der Eingabe <CONTR>-V hin und her. Der aktuelle Modus wird in der Statuszeile am oberen Bildschirmrand angezeigt.

Linkes Zeichen löschen **<DEL>**

Bewegt den Cursor um eine Stelle nach links und löscht das dort befindliche Zeichen. Jedes Zeichen rechts vom Cursor rutscht gleichzeitig um eine Stelle nach links.

Zeichen unter Cursor löschen **CONTR-G**

Löscht das Zeichen unter dem Cursor und bewegt alle Zeichen rechts davon um eine Stelle nach links. Es können nur Zeichen innerhalb der Zeile gelöscht werden.

Rechtes Wort löschen **CONTR-T**

Löscht das Wort rechts vom Cursor. befindet sich rechts vom Cursor kein Zeichen mehr in der Zeile, so werden die Weiterschaltung (<ENTER>) und anschließend die Wörter der nächsten Zeile gelöscht.

Zeile einfügen **CONTR-N**

Fügt an der Cursorposition eine neue Zeile ein, ohne den Cursor zu bewegen.

Zeile löschen **CONTR-Y**

Löscht die Zeile, in der sich der Cursor befindet und bewegt alle Zeilen darunter um eine Zeile nach oben. Eine gelöschte Zeile ist nicht mehr rekonstruierbar, Sie sollten bei der Anwendung dieses Kommandos also vorsichtig sein.

Löschen bis zum Zeilenende **CONTR-Q-Y**

Löscht die Zeile von der Cursorposition bis zum Zeilenende.

1.3.7 Block-Kommandos

Ein Textblock ist einfach eine Menge Text, von einem Zeichen bis zu mehreren Seiten. Ein Block wird durch eine Anfangsmarkierung "Begin Block" vor dem ersten Zeichen und einer Endemarkierung "End Block" nach dem letzten Zeichen des gewünschten Textblocks gekennzeichnet. So markiert, kann er nun kopiert, bewegt, gelöscht oder abgespeichert werden. Es existiert auch ein Kommando, das eine, auf einer Kassette befindliche Datei als Block in den Text lädt. Außerdem gibt es ein spezielles Kommando, das ein einzelnes Wort als Block kennzeichnet.

Blockanfangs-Markierung **CONTR-K-B**

Diese Kommando markiert den Beginn des Blocks. Die Markierung selbst wird auf dem Bildschirm nicht dargestellt.

Blockende-Markierung **CONTR-K-K**

Dieses Kommando markiert das Ende des Blocks. Entsprechend der Markierung für den Blockanfang ist sie nicht sichtbar.

Markierung eines einzelnen Wortes **CONTR-K-T**

Dieses Kommando markiert ein einzelnes Wort als Block und ersetzt die Blockanfang/-endemarkierung, die für ein einzelnes Wort zu umständlich wäre.

Kopieren eines Blockes **CONTR-K-C**

Dieses Kommando kopiert den vorher markierten Block und plaziert ihn beginnend an der Position des Cursors. Der Block, der kopiert wurde, bleibt unverändert, der neue Block besitzt ebenfalls die Markierungen.

Verschieben eines Blockes **CONTR-K-V**

Dieses Kommando verschiebt den markierten Block an die Stelle des Cursors. Die ursprüngliche Stelle ist daraufhin leer. Der Block besitzt an seiner neuen Position noch die Markierungen.

Löschen eines Blocks **CONTR-K-Y**

Dieses Kommando löscht einen markierten Block. Dieser Vorgang ist nicht rückgängig zu machen, seien Sie also vorsichtig.

Lesen eines Blockes aus einer Datei **CONTR-K-R**

Dieses Kommando liest eine Datei von Kassette und fügt sie an der aktuellen Cursorposition in den Text ein. Der eingelesene Block ist markiert. Wenn dieses Kommando gegeben wird, werden Sie aufgefordert, den Namen der einzulesenden Datei anzugeben. Jeder legale Dateiname ist möglich. .PAS wird automatisch dazugeschrieben. Eine Datei ohne Typenbezeichnung hat nach dem Namen einen Punkt.

Schreiben eines Blockes in eine Datei **CONTR-K-W**

Dieses Kommando speichert einen markierten Block als Datei ab. Der Block wird unverändert gelassen, die Markierungen verbleiben an ihrer Stelle. Wenn dieses Kommando eingegeben wird, werden Sie nach dem Namen der Datei gefragt.

1.3.8 Weitere Editier-Kommandos

Beenden des Editierens **CONTR-K-D**

Dieses Kommando beendet das Editieren und führt zum Hauptmenü zurück. Das Editieren fand ausschließlich im Arbeitsspeicher statt. Um nun die editierte Datei abzuspeichern, muß entweder das Kommando "Save" aus dem Hauptmenü verwendet werden, oder es geschieht automatisch in Verbindung mit einer Compilierung oder der Definition einer neuen Arbeitsdatei.

Tabulierung **TAB CONTR-I**

Der Editor von KCPASCAL hat keine festgesetzten Tabulatorpositionen. Stattdessen werden die Tabulierungen automatisch zu Beginn jedes Wortes auf der Zeile über dem Cursor gesetzt. Dies ist besonders nützlich beim Editieren von Programmen, wo Sie oft Spalten zueinander in Beziehung stehender Punkte aufbauen wollen, z.B. die Deklarierung von Variablen und ähnliches. Denken Sie daran, daß Pascal Ihnen erlaubt, besonders schöne Quelltexte zu schreiben. Tun Sie es, nicht aus Purismus, sondern um Ihre Programme leichtverständlich und übersichtlich zu gestalten. Das hilft Ihnen, falls Sie diese nach einiger Zeit verändern müssen.

Automatische Tabulierung an/aus **CONTR-Q-I**

Die Einrichtung für automatische Tabulierung, falls sie eingeschaltet ist, wiederholt die Spalten der vorhergehenden Zeile, d.h., wenn Sie ein <ENTER> eingeben, geht der Cursor nicht zum Zeilenanfang, sondern zur ersten benutzten Spalte der darüberstehenden Zeile. Wenn Sie eine andere Spalte wünschen, benutzen Sie lediglich eines der Kommandos zur Cursorsteuerung links/rechts. Wenn die automatische Tabulierung eingeschaltet ist, erscheint in der Statuszeile des Editors "Tab", ansonsten wird nichts angezeigt. In der Voreinstellung ist die Tabulierungsfunktion aktiv.

Zeilensicherung **CONTR-Q-L**

Dieses Kommando erlaubt, alle Änderungen in einer Zeile rückgängig zu machen. Unabhängig von der Art der Änderung wird diese Zeile in ihrem

ursprünglichen Zustand erscheinen, falls sie in der Zeile geblieben sind. Aus diesem Grund ist bei Verwendung des Lösch-Kommandos (CONTR-Y) "delete line" die Zeile unwiderruflich verloren. Wenn Sie eines Tages auf der CONTR-Y Taste einschlafen, hilft nur eine lange Arbeitspause.

Finden **CONTR-Q-F**

Dieses Kommando gibt Ihnen die Möglichkeit, einen String von max. 30 Zeichen zu suchen. Wenn Sie es eingeben, wird die Statuszeile gelöscht und Sie werden aufgefordert, einen Suchstring einzugeben. Machen Sie dies und drücken Sie anschließend <ENTER>. Der Suchstring kann alle Zeichen, auch Kontrollzeichen enthalten. Kontrollzeichen werden in den Suchstring mittels des CONTR-P Präfix eingegeben: Wenn Sie z.B. ein CONTR-A eingeben wollen, drücken Sie die <CONTR>-Taste und gleichzeitig erst P dann A. Um in den String das Kommando <ENTER> einzugeben, tippen Sie CONTR-M, CONTR-J. Beachten Sie, daß CONTR-A eine besondere Bedeutung hat, es steht für jedes Zeichen und dient damit quasi als Joker in Suchstrings.

Suchstrings können mit den Kommandos "Zeichen links", "Zeichen rechts", "Wort links" und "Wort rechts" editiert werden. "Wort rechts" bringt den vorherigen Suchstring zurück, der dann editiert werden kann. Die Suche kann mit dem Abbruch-Kommando (CONTR-U) abgebrochen werden.

Wenn Sie einen Suchstring angegeben haben, werden Sie nach den Suchoptionen gefragt. Die folgenden Optionen sind verfügbar:

B Suche von der Cursorposition rückwärts bis zum Beginn des Textes
G Suche im gesamten Text, unabhängig von der Grundposition
n n steht für eine beliebige Zahl, die Sie wählen können. Suche nach dem n-ten Vorkommen des Suchstrings, gezählt von der aktuellen Cursorposition.
U Ignorieren von Groß- und Kleinschreibung, d.h., Groß- und Kleinbuchstaben werden gleich behandelt.
W Suche nach ganzen Wörtern, Zeichenfolgen, die in Wörtern eingebettet sind, werden ignoriert.

Beispiele:
W Wenn der Suchstring "Zeit" ist, wird nur das Wort "Zeit" gesucht. Das Wort "Zeitgeist" würde ignoriert werden.
BU Suche rückwärts, unabhängig von Groß- und Kleinschreibung. Bei dem Suchstring "Block" werden auch "blocken" und "BLOCKADE" gefunden.
125 Suche das 125. Vorkommen des Suchstrings.

Nach der Eingabe der Optionen tippen Sie <ENTER> und die Suche beginnt. Wenn im Text eine Textfolge existiert, die dem Suchstring entspricht, stellt sich der Cursor an das Ende der Kombination. Mit dem Kommando für "Wiederhole letzte Suchfunktion" (CONTR-L) kann die Suche wiederholt werden.

Suchen und Ersetzen **CONTR-Q-A**

Das Suchen- und Ersetzen-Kommando erlaubt Ihnen, eine Zeichenkombination von max. 30 Zeichen zu suchen und durch eine Kombination mit ebenfalls bis zu 30 Zeichen zu ersetzen. Wenn Sie das Kommando eingeben, wird die

Statuszeile gelöscht und Sie werden aufgefordert, den Suchstring einzugeben. Machen Sie dies und drücken Sie <ENTER>. Zur Eingabe von Kontrollzeichen müssen Sie den CONTR-P Präfix voranstellen, z.B. Sie geben CONTR-A ein, indem Sie die <CONTR>-Taste und gleichzeitig erst P und dann A drücken. Das Editieren des Strings und der Abbruch der Operation entsprechen dem beim "Finden-Kommando" beschriebenen Vorgehen.

Suchstrings können mit den Kommandos "Zeichen links", "Zeichen rechts", "Wort links" und "Wort rechts" editiert werden. "Wort rechts" bringt den vorherigen Suchstring zurück, der dann editiert werden kann. Die Suche wird mit dem Abbruch-Kommando (CONTR-U) abgebrochen.

Wenn der Suchstring bestimmt ist, werden Sie aufgefordert, den String einzugeben, der diesen ersetzen soll. Bis zu 30 Zeichen sind möglich. Die Eingabe von Kontrollzeichen und das Editieren geht wie oben beschrieben. Beachten Sie aber, daß CONTR-A keine besondere Bedeutung in einem Suchstring hat. Wenn Sie nur <ENTER> eingeben, wird der gefundene String durch nichts ersetzt, d.h., gelöscht.

Schließlich können Sie noch unter folgenden Optionen wählen:

B Suchen und Ersetzen rückwärts, d.h., von der aktuellen Cursorposition bis zum Beginn des Textes
G Suche im gesamten Text, unabhängig von der Cursorposition
n n steht für eine beliebige Integerzahl. Suche und ersetze n-mal die nächsten vorkommenden, im Suchstring definierten Zeichenkombinationen von der aktuellen Cursorposition an.
N Ersetze ohne Nachfrage, d.h., die Frage "Replace Y/N" (Ersetzen Ja/Nein) entfällt.
U Ignoriere Groß- und Kleinschreibung.
W Suche und ersetze nur ganze Wörter. Zeichenfolgen, die in Wörtern eingebettet sind, werden ignoriert.

Beispiele:
N10 Suche und ersetze die nächsten 10 Kombinationen, ohne nachzufragen.
GWU Finde und ersetze nur ganze Wörter im gesamten Text. Ignoriert Groß- und Kleinschreibung.

Beenden Sie die Wahl der Optionen mit <ENTER>, dann wird die Suche eingeleitet. Wenn die Zeichenfolge gefunden ist, befindet sich der Cursor an deren Ende und Sie werden gefragt, ob eine Ersetzung gewünscht wird: "Replace Y/N?" die Frage erscheint in der Statuszeile, aber nur wenn die Option N nicht gewählt wurde. Sie können an dieser Stelle die gesamte Operation abbrechen, indem Sie das Abbruch-Kommando CONTR-U eingeben. Mit CONTR-L können Sie die Operation wiederholen.

Wiederholen der letzten Suche **CONTR-L**

Diese Kommando wiederholt das letzte Suchen und Ersetzen-Kommando so, als ob es ganz neu eingegeben worden wäre.

Kontrollzeichen Präfix **CONTR-P**

Der Editor von KCPASCAL erlaubt Ihnen, Kontrollzeichen in eine Datei einzugeben, indem Sie bei der Eingabe das Präfix CONTR-P voranstellen.

Wenn Sie in einen Text ein Kontrollzeichen eingeben wollen, müssen Sie zuerst CONTR-P und dann das gewünschte Kontrollzeichen eingeben.

Abbruch **CONTR-U**

Das Kommando CONTR-U erlaubt Ihnen den Abbruch jeder Operation, wenn eine Eingabe möglich ist. Z.B. bei der Abfrage "Austauschen (J/N)" des Suchen- und Ersetzen-Kommandos, ebenso bei der Eingabe eines Suchstrings oder Dateinamens (Block lesen/schreiben).

2. Grundlegende Sprachelemente

2.1 Grundlegende Symbole

Das wesentliche Vokabular von KCPASCAL besteht aus Symbolen, die in Buchstaben, Zahlen und Spezialsymbole eingeteilt werden können:

```
Buchstaben     A bis Z, a bis z und (Unterstreichung)
Zahlen         1 2 3 4 5 6 7 8 9
Spezialsymbole + - , = < > ( ) . . : : = $
```

Zwischen Groß- und Kleinschreibung wird keine Unterscheidung getroffen. Bestimmte Operatoren und Begrenzer werden aus zwei Spezialsymbolen gebildet:

```
Zuweisungs-Operator:    :=
Relationaler Operator: <>  <=  >=
Teilbereichs-Begrenzer:  ..:
Klammern:              (. und .) können anstatt Ä und Ü verwendet werden
Kommentare:            (* und *) können anstatt ä und ü verwendet werden
```

2.2 Reservierte Wörter

Reservierte Wörter sind ein integraler Bestandteil von KCPASCAL und können nicht neu definiert werden. Sie können also nicht als vom Nutzer definierte Bezeichner verwendet werden. Die reservierten Wörter sind:

*absolute	*external	nil	*shl
and		not	*shr
array	forward		*string
begin	for	of	then
case	function	or	type
const	goto	packed	to
div	*inline	procedure	until
do	if	program	var
downto	in	record	while
else	label	repeat	with
end	mod	set	*xor

Im gesamten Handbuch sind die reservierten Wörter fett gedruckt. Die Sterne zeigen reservierte Wörter an, die in Standard-Pascal nicht definiert sind.

2.3 Standardbezeichner

KCPASCAL hat folgende Standard-Bezeichner von vordefinierten Typen, Konstanten, Variablen, Prozeduren und Funktionen. Jeder dieser Bezeichner kann neu definiert werden, was aber bedeutet, daß ihre Möglichkeiten eingeschränkt werden. Solche Veränderungen können auch leicht zur Verwirrung führen. Die folgenden Standard-Bezeichner sind deshalb am besten in ihrer ursprünglichen Definition zu belassen:

Addr	Delay	Length	Release
ArcTan	Delete	Ln	
	EOF	Lo	
Aux	EOLN		
AuxInPtr	Erase	Lst	Round
AuxOutPtr	Execute	LstOutPtr	
	Exit	Mark	Sin
	Exp	MaxInt	SizeOf
Boolean	False	Mem	
Buflen		MemAvail	
Byte		Move	Sqr
Chain	FillChar	New	Sqrt
Char	Flush		Str
Chr	Frac	Odd	Succ
	GetMem	Ord	Swap
ClrEOL	GotoXY	Output	Text
ClrScr	Halt	Pi	Trm
Con	HeapPtr	Port	True
ConInPtr	Hi	Pos	Trunc
ConOutPtr	IOresult	Pred	UpCase
Concat	Input	Ptr	Usr
ConstPtr	Insline	Random	UsrInPtr
Copy	Insert	Randomize	UsrOutPtr
Cos	Int	Read	Val
	Integer	ReadLn	Write
	Kbd	Real	WriteLn
DelLine	KeyPressed		

Im gesamten Handbuch werden die Standard-Bezeichner, wie auch alle anderen Bezeichner mit Klein- und Großbuchstaben geschrieben. Im Text werden sie kursiv gedruckt.

2.4 Begrenzer

Sprachelemente müssen wenigstens von einem der folgenden Begrenzer unterbrochen werden: Leerzeichen, neue Zeile oder Kommentar.

2.5 Programmzeilen

Programmzeilen können maximal 127 Zeichen lang sein, darüber hinausgehende Zeichen werden vom Compiler ignoriert. Aus diesem Grund erlaubt der KCPASCAL-Editor auch nur 127 Zeichen in einer Zeile.

3. Skalare Standardtypen

Ein Datentyp definiert die Art der Werte, die eine Variable annehmen kann. Jede Variable in einem Programm darf einem und nur einem Datentyp zugeordnet sein. Obwohl Datentypen in KCPASCAL sehr komplex sein können, werden sie alle aus einfachen unstrukturierten Typen aufgebaut.

Ein einfacher Typ kann entweder vom Programmierer definiert sein (er wird dann deklarierter skalarer Typ genannt), oder er ist einer der skalaren Standardtypen integer, real, boolean, char oder byte. Es folgt eine Beschreibung dieser fünf skalaren Standardtypen.

3.1 Integer (ganze Zahlen)

Integer sind ganze Zahlen, die bei KCPASCAL auf einen Bereich von -32768 bis 32767 begrenzt sind. Ganze Zahlen belegen zwei Bytes im Speicher.

3.2 Byte

Der Typ Byte ist ein Teilbereich des Typs Integer mit den Grenzen 0 und 255. Bytes sind deshalb mit dem Typ Integer kompatibel, d.h., wann immer ein Wert vom Typ Byte erwartet wird, kann ein integer Wert angegeben werden und umgekehrt, außer bei der Übergabe von Parametern. Weiterhin können Byte und Integer in Ausdrücken gemischt werden und Byte-Variable können integer-Werte zugewiesen bekommen. Eine Variable vom Typ Byte belegt ein Byte im Speicher.

3.3 Real (reelle Zahlen)

Der Bereich reeller Zahlen (Datentyp real) ist 1E-38 bis 1E+38 mit einer Mantisse mit bis zu 11 signifikanten Stellen. Reelle Zahlen belegen 6 Bytes im Speicher.

Bei einer arithmetischen Operation mit reellen Zahlen verursacht ein Überlauf einen Programmstop und die Anzeige eines Ausführungsfehlers. Eine Unterschreitung der Bereichsgrenze führt zu einem Ergebnis von Null.

Obwohl der Typ real zu den skalaren Standardtypen gehört, sollte folgender Unterschied zwischen dem Typ real und anderen skalaren Typen beachtet werden:

1.) Die Funktionen Pred und Succ dürfen keine reellzahligen Argumente enthalten.
2.) Der Typ real darf nicht bei der Indexierung von Arrays verwendet werden.
3.) Der Typ real kann nicht verwendet werden, um den Grundtyp einer Menge zu definieren.
4.) Der Typ real kann nicht in kontrollierenden for- und case-Anweisungen verwendet werden.
5.) Teilbereiche des Typs real sind nicht erlaubt.

3.4 Boolean (Bool'sche Wahrheitswerte)

Ein Bool'scher Wahrheitswert kann einen der beiden logischen Werte wahr oder falsch, die durch die Standardbezeichner True bzw. False bezeichnet sind, annehmen. Diese sind so definiert, daß True < False ist. Eine Boolean-Variable belegt ein Byte im Speicher.

3.5 Char (alphanumerische Zeichen)

Ein Char (alphanumerischer) Wert entspricht einem Zeichen aus der ASCII-Zeichenmenge. Die Zeichen sind entsprechend ihrem ASCII-Wert geordnet, z.B. 'A' > 'B'. Die ordinalen (ASCII) Werte der Zeichen reichen von 0 bis 255. Eine Char-Variable belegt ein Byte im Speicher.

4. Benutzerdefinierte Sprachelemente

4.1 Bezeichner

Bezeichner (engl. Identifier) werden verwendet, um Labels, Konstanten, Typen, Variablen, Prozeduren und Funktionen zu bezeichnen. Ein Bezeichner besteht aus einem Buchstaben oder einer Unterstreichung gefolgt von beliebigen Kombinationen von Buchstaben, Zahlen oder Unterstreichungen. Ein Bezeichner wird in der Länge nur von der Länge der Zeile, die maximal 127 Zeichen betragen kann, begrenzt und alle Zeichen sind signifikant.

```
Beispiele:
KCPASCAL
square
persons-counted
BirthDate
3rdRoot     illegal, da eine Zahl am Anfang steht
Two Words   illegal, da kein Leerzeichen enthalten sein darf.
```

Da KCPASCAL nicht zwischen Groß- und Kleinschreibung unterscheidet, hat die Verwendung von Groß- und Kleinbuchstaben, wie z.B. bei BirthDate, keine Auswirkung. Diese wird jedoch empfohlen, da sie die Lesbarkeit erhöht. SehrLangerBezeichner ist leichter für das menschliche Auge zu lesen, als SEHRLANGERBEZEICHNER. Die vermischte Verwendung von Groß- und Kleinschreibung wird im gesamten Handbuch für Bezeichner gebraucht.

4.2 Zahlen

Zahlen sind Konstanten vom Typ integer oder real. Integer-Konstanten sind ganze Zahlen, die in dezimaler oder hexadezimaler Notation ausgedrückt sind. Hexadezimale Konstanten werden dadurch identifiziert, daß ihnen ein Dollarzeichen voransteht: $ABC ist eine hexadezimale Konstante. Der dezimale Integer-Bereich ist -32768 bis 32767 und der hexadezimale Integer-Bereich ist $0000 bis $FFFF.

```
Beispiele:
1
12345
-1
$123
$ABC
$123G       illegal, da G keine legale hexadezimale Zahl ist
4.2345      illegal, da eine ganze Zahl keine Stelle hinter dem Komma
            haben kann
```

Der Bereich reeller Zahlen hat eine Mantisse von 1E-38 bis 1E+38, die 11 signifikante Stellen aufweist. Sie können Exponentenschreibweise verwenden, wobei der Buchstabe E dem Exponentialfaktor vorausgeht und bedeutet 10mal die Potenz von *). Eine Integer-Konstante ist überall erlaubt, wo eine real Konstante stehen kann. Trennungszeichen sind innerhalb von Zahlen nicht erlaubt.

*) besser: 10 hoch Exponentialfaktor

```
Beispiele:
1.0
1234.5678
-0.012
1E0
2E-5
-1.2345678901E+12
1           zulässig, aber es ist keine reelle, sondern eine ganze Zahl
```

4.3 Strings

Eine Stringkonstante ist eine Sequenz von Zeichen, die mit einfachen Anführungszeichen eingefaßt sind.

```
'Dies ist eine String-Konstante'
```

Ein einzelnes Anführungszeichen kann in einem String stehen, indem es doppelt geschrieben wird. Strings, die nur ein einziges Zeichen enthalten, gelten als Standardtyp Char. Ein String ist mit einem "array of Char" derselben Länge kompatibel. Alle Stringkonstanten sind mit allen String-Typen kompatibel.

```
Beispiele:
'KCPASCAL'
'You''ll see'
''''
```

Wie man an Beispiel 2 und 3 sieht, wird ein einzelnes Anführungszeichen in einem String als zwei aufeinanderfolgende Anführungszeichen geschrieben. Die vier aufeinanderfolgenden Anführungszeichen im Beispiel 3 stellen einen String dar, der ein einzelnes Anführungszeichen enthält.

4.3.1 Kontrollzeichen

Mit KCPASCAL können Sie auch Kontrollzeichen n Strings einbetten. Es werden zwei Notationen für Kontrollzeichen unterstützt:

1.) Das Symbol # gefolgt von einer Integer-Konstanten im Bereich 0..255 bezeichnet ein Zeichen des entsprechenden ASCII-Werts.
2.) Das Symbol ^ gefolgt von einem Zeichen, bezeichnet das entsprechende Kontrollzeichen.

```
Beispiele:
# 10  ASCII 10 dezimal (Zeilen-Vorschub)
#$1B  ASCII 1B hex (Escape)
^G    Control-G (Bell = Klingel)
^L    Control-L (Formular-Vorschub)
^     Control   (Escape).
```

Sequenzen von Kontrollzeichen können zu Strings verkettet werden, indem sie ohne Trennungszeichen zwischen den einzelnen Zeichen geschrieben werden:

```
#3#10
#27^U20
^G^G^G^G
```

Die obigen Strings enthalten zwei, drei und vier Zeichen.

Kontrollzeichen können auch mit Textstrings gemischt sein:

```
'Waiting for input!'^G^G^G'Please wake up'
#27'U'
'This is another line of text'^M^J
```

Diese Strings enthalten jeweils 37, 3 und 31 Zeichen.

4.4 Kommentare

Kommentare können überall im Programm eingefügt werden, wo ein Begrenzer erlaubt ist. Ein Kommentar ist durch die Symbole (* und *) bzw. von geschweiften Klammern begrenzt.

Beispiele:

```
(* Dies ist ein Kommentar *)
{ Dies ist ebenfalls ein Kommentar }
```

Geschweifte Klammern dürfen nicht geschachtelt werden, (*...*) ebenfalls nicht. Es dürfen aber geschweifte Klammern in (*...*) geschachtelt sein und umgekehrt.

4.5 Compilerbefehle

Eine Reihe der Eigenschaften des KCPASCAL-Compilers wird durch Compiler-befehle gesteuert. Ein Compilerbefehl wird als Kommentar mit einer speziellen Syntax eingeführt. Das bedeutet, daß überall da, wo ein Kommentar erlaubt ist, auch ein Compilerbefehl stehen kann.

Ein Compilerbefehl besteht aus einer öffnenden, geschweiften Klammer oder (*, gefolgt von einem Dollarzeichen, unmittelbar darauf folgt ein Compilerbefehlsbuchstabe oder eine Liste von Compilerbefehlsbuchstaben, die durch Kommas getrennt sind. Der Compilerbefehl wird durch eine schließende, geschweifte Klammer bzw. *) abgeschlossen. Die Syntax des Befehls oder der Liste von Befehlen ist unterschiedlich. Sie ist in den entsprechenden Kapiteln beschrieben; eine Zusammenfassung der Compilerbefehle befindet sich in Anhang 3.

Beispiele:

```
(*$R-,B-,V-*)
(*$X-*)
```

Beachten Sie, daß vor und nach dem Dollarzeichen keine Leerzeichen erlaubt sind.

5. Programmkopf und Programmblock

Ein Pascal-Programm besteht aus einem Programmkopf, gefolgt von einem Programmblock. Der Programmblock ist weiter unterteilt in einen Deklarierungsteil, in dem alle im Programm vorkommenden Objekte definiert werden und einen Ausführungsteil, in dem die Aktionen spezifiziert werden, die mit diesen Objekten ausgeführt werden sollen. Beide werden im folgenden genau beschrieben.

5.1 Programmkopf

Bei KCPASCAL ist der Programmkopf nur optional und hat keine Bedeutung für das Programm. Wenn vorhanden, gibt er dem Programm einen Namen und listet wahlweise die Parameter auf, durch die das Programm mit der Umgebung kommuniziert. Die Liste besteht aus einer Reihe von Bezeichnern, die in Klammern stehen und mit Kommas getrennt sind.

Beispiele:

```
program Circles;
program Accountant(Input,Output);
program Writer(Input,Printer);
```

5.2 Deklarierungsteil

Der Deklarierungsteil eines Blocks deklariert alle Bezeichner, die im Anweisungsteil dieses Blocks (und möglicherweise anderer Blöcke innerhalb von diesem) benutzt werden. Der Deklarierungsteil ist in fünf unterschiedliche Abschnitte eingeteilt:

1) Label-Deklarierungsteil
2) Konstanten-Definitionsteil
3) Typen-Definitionsteil
4) Variablen-Deklarationsteil
5) Prozeduren-/Funktionen-Deklarationsteil

Während Standard-Pascal vorschreibt, daß jeder Abschnitt entweder garnicht oder einmal vorkommen darf und nur in der obigen Reihenfolge, erlaubt KCPASCAL, daß jeder dieser Abschnitte beliebig oft und in jeder Reihenfolge im Deklarierungsteil auftreten darf.

5.2.1 Label-Deklarationsteil

Jede Anweisung kann mit einem vorangestellten Label versehen werden, was es ermöglicht, mittels einer goto-Anweisung direkt zu dieser Anweisung zu verzweigen. Ein Label besteht aus einem Labelnamen, dem ein Komma folgt. Vor Gebrauch muß es in einem Label-Deklarationsteil deklariert werden. Das reservierte Wort "label" steht am Anfang dieses Teils. Es folgt eine Liste der Labelbezeichner, die mit Kommas untereinander getrennt sind und von einem Semikolon abgeschlossen werden.

Beispiel:

```
label, 10, Fehler, 999, Abbruch;
```

Während Standard-Pascal die Label auf Zahlen mit höchstens vier Stellen einschränkt, erlaubt KCPASCAL, sowohl Zahlen als auch Bezeichner als Label zu verwenden.

5.2.2 Konstanten-Definitionsteil

Der Konstanten-Definitionsteil führt Bezeichner als Synonyme für die Konstantenwerte ein. Das reservierte Wort "const" steht am Anfang des Konstanten-Definitionsteils, es folgt eine Liste der Konstantenzuweisungen, die durch Semikolons getrennt sind. Jede Konstantenzuweisung besteht aus einem Bezeichner, auf den ein Gleichheitszeichen und eine Konstante folgen. Konstanten sind entweder Strings oder Zahlen.

Beispiel:

```
const
Limit = 255;
Max = 1024;
Passwort = 'SESAM';
CursHome = ^'V';
```

Die folgenden Konstanten sind in KCPASCAL vordefiniert, d.h., auf sie kann ohne vorherige Definition Bezug genommen werden:

```
Name:       Typ und Wert:
Pi          Real (3.1415926536E-00)
False       Boolean (der Wahrheitswert falsch)
True        Boolean (der Wahrheitswert wahr)
Maxint      Integer (32767)
```

Wie in Kapitel 13 beschrieben, kann ein Konstanten-Definitionsteil auch typisierte Konstanten definieren.

5.2.3 Typen-Definitionsteil

Ein Datentyp kann in Pascal entweder direkt in dem Variablen-Deklarationsteil beschrieben sein, oder es kann durch einen Typenbezeichner auf ihn Bezug genommen werden. Es stehen mehrere Standardtypen zur Verfügung: weiterhin kann ein Programmierer durch die Verwendung der Typendefinition eigene Datentypen erzeugen. Das reservierte Wort "type" steht am Anfang des Typendefinitionsteils. Es folgen eine oder mehrere Zuweisungen, die durch Semikolons getrennt werden. Jede Typzuweisung besteht aus einem Typbezeichner, auf den ein Gleichheitszeichen und ein Typ folgen.

Beispiel:

```
type
Number = Integer;
Day = (mon, tue, wed, thu, fri, sat, sun);
List = array[1..10] of real;
```

Weitere Beispiele für Typendefinitionen finden sich in den folgenden Abschnitten.

5.2.4 Variablen-Deklarationsteil

Jede Variable, die in einem Programm auftaucht, muß vor ihrer Verwendung deklariert werden. Die Deklarierung muß textlich einer Verwendung der Variablen vorausgehen, d.h., die Variable muß dem Compiler bekannt sein, bevor sie benutzt werden kann.

Eine Variablendeklarierung besteht aus dem reservierten Wort "var", darauf folgen ein oder mehrere Bezeichner, die durch Kommas getrennt sind und dann jeweils ein Doppelpunkt und eine type-Angabe.

Der Geltungsbereich dieser Bezeichner ist der Block, in dem sie definiert sind und jeder weitere Block innerhalb dieses Blocks. Beachten Sie, daß jeder Block innerhalb eines anderen Blocks, eine andere Variable definieren kann, die denselben Bezeichner verwendet. Diese Variable wird als lokal zu dem Block bezeichnet, in dem sie definiert ist (und in jedem weiteren Block innerhalb dieses Blocks). Die Variable, die auf dem äußeren Level deklariert wurde (die globale Variable), wird unzugänglich.

Beispiel:

```
var
    Result, Intermediate, SubTotal:Real;
    I, J, X, Y:Integer;
    Accepted, Valid:Boolean;
    Period: Day;
    Buffer: array[0..127] of Byte;
```

5.2.5 Prozedur- und Funktions-Deklarationsteil

Eine Prozedurdeklarierung dient dazu, eine Prozedur innerhalb einer gegenwärtigen Prozedur oder eines Programms zu definieren (siehe Seite 131). Eine Prozedur wird von einer Prozedur-Anweisung aktiviert (siehe Seite 56). Nach Abschluß der Prozedur geht die Programmausführung mit der Anweisung weiter, die unmittelbar auf die aufrufende Anweisung folgt.

Eine Funktionsdeklarierung dient dazu, einen Programmteil zu definieren, der einen Wert berechnet und ausgibt (siehe Seite 137). Eine Funktion wird aktiviert, wenn ihr Bezeichner (engl.:designator) als ein Teil eines Ausdrucks angetroffen wird (siehe Seite 54).

5.3 Anweisungsteil

Der Anweisungsteil ist der letzte Teil eines Blocks. Er spezifiziert die vom Programm auszuführenden Aktionen. Der Anweisungsteil hat die Form einer zusammengesetzten Anweisung, der ein Absatz oder ein Semikolon folgt. Eine zusammengesetzte Anweisung besteht aus dem reservierten Wort "begin", es folgt eine Liste von Anweisungen, getrennt durch Semikolons und wird durch das reservierte Wort "end" abgeschlossen.

6. Ausdrücke

Ausdrücke (engl.: expressions) sind algorithmische Konstrukte, die Regeln für die Berechnung von Werten angeben. Sie bestehen aus Operanden, d.h. Variablen, Konstanten und Funktionsbezeichnern, die mittels Operatoren kombiniert werden.

Dieser Abschnitt beschreibt, wie Ausdrücke aus den skalaren Standardtypen Integer, Real, Boolean und Char gebildet werden. Ausdrücke, die die deklarierten, skalaren Typen, String-Typen und Set-Typen enthalten, werden auf den Seiten 63, 67 und 86 in dieser Reihenfolge beschrieben.

6.1 Operatoren

Operatoren fallen in fünf Kategorien, die hier nach ihrer Priorität geordnet sind:

1) Monadisches Minus (Minus mit nur einem Operanden).
2) Not-Operator.
3) Multiplikations-Operator: ',/,div,mod,and,shl.shr.
4) Additions-Operatoren: +,-,or und xor.
5) Relationale Operatoren: =,<>,<,>,<=,>= und in.

Folgen von Operatoren derselben Priorität werden von links nach rechts berechnet. Ausdrücke in Klammern werden zuerst berechnet, unabhängig von vorausgehenden oder nachfolgenden Operatoren.

Wenn beide Operanden eines Multiplikations- oder Additionsoperators vom Typ Integer sind, dann ist das Ergebnis ebenfalls Integer. Wenn einer (oder beide) der Operanden vom Typ Real ist(sind), ist auch das Ergebnis vom Typ Real.

6.1.1 Monadisches Minus

Das monadische Minus bezeichnet eine Negation seines Operanden, dieser kann vom Typ Real oder Integer sein.

6.1.2 Not-Operator

Der not-Operator negiert (kehrt den logischen Wert seines Boolschen Operanden um).

```
not True    = False
not False   = True
```

KCPASCAL erlaubt auch die Anwendung des not-Operators auch auf einen integer-Operanden, in diesem Fall findet eine bitweise Negation statt:

Beispiele:

```
not0        = -1
not-15      = 14
notS2345    = SDCBA
```

6.1.3 Multiplikations-Operatoren

Operator	Wirkung	Typ des Operanden	Ergebnistyp
*	Multiplikation	Real	Real
*	Multiplikation	Integer	Integer
*	Multiplikation	Real, Integer	Real
/	Division	Real, Integer	Real
/	Division	Integer	Real
/	Division	Real	Real
div	Integer Division	Integer	Integer
mod	Modulus	Integer	Integer
and	arithmet. und	Integer	Integer
and	logisches und	Boolean	Boolean
shl	verschieben links	Integer	Integer
shr	verschieben rechts	Integer	Integer

Beispiele:

```
12 * 34                 = 408
123/4                   = 30.75
123 div 4               = 30
12 mod 5                = 2
True und False          = False
12 and 22               = 4
2 shl 7                 = 256
256 shr 7               = 2
```

6.1.4 Additions-Operatoren

Operator	Wirkung	Typ des Operanden	Ergebnistyp
+	Addition	Real	Real
+	Addition	Integer	Integer
+	Addition	Real, Integer	Real
-	Subtraktion	Real	Real
-	Subtraktion	Real, Integer	Real
-	Subtraktion	Integer	Integer
or	arithmet. oder	Integer	Integer
or	logisches oder	Boolean	Boolean
xor	arithmet. exclusiv- oder	Integer	Integer
xor	log. exclusiv-oder	Boolean	Boolean

Beispiele:

```
123 + 456               = 579
456 - 123.0             = 333.0
True or False           = True
12 or 22                = 30
True xor False          = True
12 xor 22               = 26
```

6.1.5 Relationale Operatoren

Relationale Operatoren gelten für alle skalaren Standardtypen: Integer, Real, Boolean, Char und Byte. Operanden der Typen Integer, Real und Byte können gemischt werden. Der Ergebnistyp ist immer Boolean, d.h. True oder

False (wahr oder unwahr).

```
=     ist gleich
<>    ungleich
>     größer als
<     kleiner als
>=    größer gleich
<=    kleiner gleich
```

Beispiele:

```
a = b       ist wahr, falls a gleich b
a <> b      ist wahr, falls a ungleich b
a > b       ist wahr, falls a größer b
a < b       ist wahr, falls a kleiner b
a >= b      ist wahr, falls a größer gleich b
a <= b      ist wahr, falls a kleiner gleich b
```

6.2 Funktionsbezeichnung

Die Funktionsbezeichnung (engl.: function designator) ist ein Funktionsbezeichner, dem wahlweise eine Parameterliste folgt, die eine oder mehrere Variable oder Ausdrücke, die durch Kommas getrennt sind und von Klammern umschlossen werden, enthält. Das Auftreten einer Funktionsbezeichnung aktiviert die Funktion mit diesem Namen. Wenn eine Funktion keine vordefinierte Standardfunktion ist, muß sie vor der Aktivierung erst deklariert werden.

Beispiele:

```
Round (PlotPos)
Writeln (pi * (Sqr(R))
(Max (X/Y) < (25) and (Z) > Sqrt (X*Y))
Volume (Radius, Heigth)
```

7. Anweisungen

Der Ausführungsteil definiert die Aktion, die vom Programm (oder Unterprogramm) ausgeführt werden soll, als Folge von Anweisungen (engl.: statements). Jede Anweisung spezifiziert einen Teil der Aktion. In diesem Sinne ist Pascal eine sequentielle Programmiersprache: Anweisungen werden zeitlich sequentiell abgearbeitet, nie zugleich. Der Anweisungsteil ist durch Semikolons getrennt. Anweisungen können einfach oder strukturiert sein.

7.1 Einfache Anweisungen

Einfache Anweisungen enthalten keine anderen Anweisungen. Einfache Anweisungen sind die Zuweisungs-, Prozedur-, goto- und leere Anweisung.

7.1.1 Zuweisungsanweisung

Die grundlegendste aller Anweisungen ist die Zuweisungsanweisung. Sie wird verwendet, um anzugeben, daß ein bestimmter Wert einer bestimmten Variablen zugewiesen werden soll. Eine Zuweisung besteht aus einem Variablenbezeichner, dem der Zuweisungsoperator := und ein Ausdruck folgen.

Zuweisungen sind zu Variablen beliebigen Typs (außer Dateien) möglich, solange die Variable und der Ausdruck vom selben Typ sind. Als Ausnahme davon kann bei einer Real-Variablen der Ausdruck Integer sein.

Beispiele:

```
Angle := Angle  Pi;
AccessOK := False;
Entry := Answer = Password;
SpherVol := 4 * Pi * R * R;
```

7.1.2 Prozedur-Anweisung

Die Prozedur-Anweisung dient dazu, eine zuvor vom Benutzer definierte oder eine vordefinierte Standardprozedur zu aktivieren. Die Anweisung besteht aus einem Prozedurbezeichner, wahlweise gefolgt von einer Parameterliste. Diese Parameterliste ist eine Liste von Variablen oder Ausdrücken, die durch Kommas getrennt und in Klammern eingeschlossen sind. Wenn bei der Ausführung des Programms die Prozedur-Anweisung erreicht wird, wird die Kontrolle auf die Prozedur übertragen, die Werte möglicher Parameter werden ebenfalls auf die Prozedur übertragen. Wenn die Prozedur beendet ist, geht die Programmausführung mit der Anweisung weiter, die auf die Prozeduranweisung folgt.

Beispiele:

```
Find (Name,Adresse);
Sort (Adresse);
UpperCase (Text);
UpdateCustFile (CustRecord);
```

7.1.3 Goto-Anweisung

Eine goto-Anweisung besteht aus dem reservierten Wort "goto", auf das ein Labelbezeichner folgt. Sie dient dazu, die weitere Verarbeitung an die Stelle im Programmtext zu übergeben, die durch das Label markiert ist.

7.1.4 Leere Anweisung

Eine leere Anweisung besteht aus keinen Symbolen und hat keine Wirkung. Sie darf vorkommen, wo immer die Syntax von Pascal eine Anweisung verlangt, aber keine Aktion stattfinden soll.

7.2 Strukturierte Anweisungen

7.2.1 Zusammengesetzte Anweisung

Eine zusammengesetzte Anweisung (engl.: compound statment) wird benutzt, wenn in einer Situation mehr als eine Anweisung ausgeführt werden soll, in der die Pascal-Syntax nur die Spezifikation einer Anweisung erlaubt. Sie besteht aus einer beliebigen Zahl von Anweisungen, die mit Semikolons getrennt sind und von den reservierten Wörtern "begin" und "end" eingeschlossen werden. Die einzelnen Anweisungen der zusammengesetzten Anweisung werden in der Reihenfolge, in der sie geschrieben sind, ausgeführt.

Beispiel:

```
if (Small) Big then
begin
  Tmp := Small;
  Small := Big;
  Big := Tmp;
end;
```

7.2.2 Bedingte Anweisung

7.2.2.1 If-Anweisung

Die if-Anweisung (Entscheidung) spezifiziert, daß eine Anweisung nur dann ausgeführt wird, wenn eine bestimmte Bedingung (Boolean Ausdruck) erfüllt (wahr) ist. Wenn sie nicht erfüllt (falsch) ist, dann wird entweder keine Anweisung oder die Anweisung, die auf das reservierte Wort "else" folgt, ausgeführt. Beachten Sie, daß "else" kein Semikolon vorangehen darf.

Die syntaktische Zweideutigkeit, die aus folgendem Konstrukt besteht:

```
if expr1 then
if expr2 then
  stmt1
  else
    stmt2
```

wird beseitigt, indem das Konstrukt folgendermaßen interpretiert wird:

```
if expr1 then
begin
  if expr2 then
    stmt1
  else
    stmt2
end;
```

d.h., der else-Klauselteil gehört generell zur letzten if-Anweisung, die keinen else-Teil hat.

```
Beispiele:
if Interest > 25 then
  Usury := true
else
  TakeLoan := OK;

if (Entry <= 0 ) or (Entry > 100 ) then
begin
  Write ('Range is 1 to 1000, please, re-enter:');
  Read (Entry);
end;
```

7.2.2.2 Case-Anweisung

Die case-Anweisung (Auswahl) besteht aus einem Ausdruck (dem Sortierer) und einer Liste von Anweisungen, denen jeweils case-Label vom Typ des Sortierers vorausgehen. Sie gibt an, daß die Anweisung, deren Label dem aktuellen Wert des Sortierers entspricht, ausgeführt werden soll. Wenn kein case-Label den Wert des Sortierers enthält, dann werden entweder keine oder wahlweise die Anweisungen, die dem reservierten Wort "else" folgen, ausgeführt. Die else-Klausel ist eine Erweiterung von Standard-Pascal.

Ein case-Label besteht aus einer beliebigen Zahl von Konstanten oder Teilbereichen, die durch Kommas getrennt sind und denen ein Semikolon folgt. Ein Teilbereich wird als zwei Konstanten geschrieben, die von dem Teilbereichs-Begenzer '..' getrennt werden. Der Typ der Konstanten muß gleich dem Typ des Sortierers sein. Die Anweisung, die dem case-Label folgt, wird ausgeführt, wenn der Wert des Sortierers gleich einer der Konstanten ist oder in einem der Teilbereiche liegt.

Gültige Sortierer-Typen sind alle einfachen Typen, alle skalaren Typen außer reellen Zahlen.

```
Beispiele:
case Operator of
'+' : Result := Answer + Result;
'-' : Result := Answer - Result;
'*' : Result := Answer * Result;
'/' : Result := Answer / Result;
end
```

```
case Year of
  Min..1939: begin
    Time := PreWorldWar2;
    Writeln ('The world at peace..');
    end;
1946 .. Max: begin
    Time := PostWorldWar2;
    Writeln ('Building a new world');
    end;
else
  Time := WorldWar2;
  Writeln ('We are at war');
end;
```

7.2.3 Wiederholte Anweisungen

7.2.3.1 For-Anweisung

Die for-Anweisung (Laufanweisung oder Zählschleife) zeigt an, daß die Teilanweisung wiederholt ausgeführt werden soll. Die ansteigenden Werte werden einer Variablen zugewiesen, die Kontrollvariable genannt wird. Die Werte können aufsteigend "to" oder absteigend "downto" bis zu dem endgültigen Wert sein.

Die Kontrollvariable, der anfängliche Wert und der endgültige Wert müssen alle vom selben Typ sein. Gültige Typen sind alle einfachen Typen, d.h. alle skalaren Typen außer Real. Wenn bei Verwendung der to-Klausel der anfängliche Wert größer als der endgültige Wert ist, oder bei Verwendung der downto-Klausel der anfängliche Wert kleiner als der endgültige Wert ist, wird der Anweisungsteil nicht ausgeführt.

```
Beispiele:
for I:= 2 to 100 do if A[I]> Max then Max = A[I]
for I:= 1 to NoOfLines do
begin
  Readln (Line);
  if Length (Line) < Limit then ShortLines := ShortLines + 1
  else
    Longlines := Longlines - 1
end;
```

7.2.3.2 While-Anweisung

Der Ausdruck, der die Wiederholung kontrolliert, muß vom Typ boolean sein. Die Anweisung wird so lange wiederholt, solange der Ausdruck true (wahr) ist. Ist der Wert schon zu Beginn false (falsch), wird die Anweisung überhaupt nicht ausgeführt.

```
Beispiele:
while Size > 1 do Size := Sqrt(Size):

while ThisMonth do
begin
  ThisMonth := CurMonth = SampleMonth:
  Process;
  bearbeite dieses Beispiel mit der Process-Prozedur
end;
```

7.2.3.3 Repeat-Anweisung

Der Ausdruck, der die Wiederholung kontrolliert, muß vom Typ boolean sein. Die Sequenz der Anweisung zwischen den reservierten Wörtern "repeat" und "until" wird so oft wiederholt, bis der Ausdruck wahr wird. Im Unterschied zur while-Anweisung wird die repeat-Anweisung immer mindestens einmal ausgeführt, da erst am Ende der Schleife die Abbruch-bedingung abgefragt wird.

Beispiel:

```
repeat
  Write (^M.'Delete this item?(Y/N)');
  Read (Answer);
until UpCase (Answer) in 'Y','N';
```

8. Skalare Datentypen und deren Teilbereiche

Der skalare Datentyp ist bei Pascal der grundlegende Datentyp. Er bildet eine endliche, linear angeordnete Reihe von Werten. Obwohl der Standard-datentyp Real auch als skalarer Datentyp betrachtet wird, fällt er nicht unter diese Definition. Deshalb können real Zahlen nicht immer im gleichen Zusammenhang wie andere skalare Datentypen verwendet werden.

8.1 Skalare Datentypen

Neben den skalaren Standardtypen (Integer, Real, Boolean, Char und Byte) unterstützt Pascal auch vom Benutzer definierte skalare Datentypen). Die Definition eines skalaren Datentyps werden durch Bezeichner repräsentiert, die deren Konstanten sind.

Beispiele:

```
Type
  Operator        =(Plus, Minus, Multi, Divide);
  Day             =(Mon, Tue, Wed, Thu, Fri, Sat, Sun);
  Month           =(Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
                    Nov, Dec);
  Card            =(Club, Diamond, Harrt, Spade);
```

Variablen des Datentyps Card können eine der vier oben angegebenen Werte annehmen. Mit dem skalaren Standardtyp Boolean sind Sie schon vertraut. Er wird folgendermaßen definiert:

```
type
  Boolean         =(False, True);
```

Die Operatoren =.<>.<.>.<= und >= können allen skalaren Datentypen beigeordnet werden, unter der Bedingung, daß beide Operanden vom gleichen Datentyp sind (als Ausnahme können Real- und Integer-Zahlen gemischt werden). Die Operanden werden in der Reihenfolge ihres Auftretens verglichen, so gilt für den Typen Card aus obigem Beispiel:

```
Club < Diamond < Heart < Spade
```

Drei Standardfunktionen sind vorhanden, die mit den skalaren Datentypen arbeiten:

```
Succ(Diamond)     der Nachfolge (successor) von Diamond(=Heart)
Pred(Diamond)     der Vorgänger (predecessor) von Diamond(=Club)
Ord(Diamond)      die Platznummer (ordinal number) von Diamond(=1)
```

Das erste Element des Datentyps hat immer die Ordnungszahl bzw. Platz-nummer 0. Der Datentyp des Ergebnisses von Succ und Pred entspricht dem Datentyp des jeweiligen Arguments, von Ord ist es immer eine Integer-Zahl.

8.2 Teilbereiche skalarer Datentypen

Ein Datentyp kann auch als Teilbereich eines bereits definierten skalaren Typs definiert werden. Diese werden als Teilbereiche skalarer Datentypen bezeichnet. Die Definition bestimmt lediglich den niedrigsten und den höchsten Wert dieses Teilbereichs.

Die erste Konstante bestimmt die untere Grenze und darf nicht größer als die zweite sein, die die obere Grenze bildet. Ein Teilbereich des Typs Real ist nicht erlaubt.

```
Bespiele:
type
  HemiSphere      = (North,South,West,East;
  World           = (East,West);
  CompassRange    = 0..360;
  Upper           = 'A'..'Z';
  Lower           = 'a'..'z';
  Degree          = (Celc,Fahr,Ream,Kelv);
  Wine            = (Red,White,Rose,Sparkling);
```

Der Datentyp World ist ein Teilbereich des skalaren Datentyps Hemisphere. Der Teilbereich von CompassRange ist integer und der dazugehörige skalare Typ von Upper und Lower ist Char.

Sie kennen bereits den standardisierten Teilbereich des Datentyps Byte, der wie folgt definiert ist:

```
type
  Byte = 0..255;
```

Ein Teilbereich besitzt alle Attribute seines zugeordneten skalaren Datentyps und ist lediglich durch die Menge seiner möglichen Werte begrenzt.

8.3 Umwandlung von Datentypen

Mit der Funktion Ord können skalare Datentypen dem Wert einer integer-Zahl zugeordnet werden. Standard-Pascal unterstützt diese Umwandlung nicht in die umgekehrte Richtung, d.h., eine integer-Zahl läßt sich nicht in einen skalaren Wert umwandeln.

```
Integer(Heart)    = 2
Month(10)         = Nov
Hemisphere(2)     = East
Upper(14)         = 'O'
Degree(3)         = Kelv
Char(78)          = 'N'
Integer('7')      = 55
```

8.4 Überprüfung der Variablengröße

Die Erzeugung von Code, der Überprüfungen der Wertebereiche von Variablen zuläßt, wird mit dem Compilerbefehl R kontrolliert. Die Voreinstellung ist {$r}, d.h., daß keine Überprüfungen stattfinden. Wenn einer skalaren Variablen oder einem ihrer Teilbereiche ein Wert zugewiesen wird, wird dieser auf seine Größe geprüft, solange der Compilerbefehl aktiv ist ($R+). Es wird empfohlen, diese Option solange aktiv zu lassen, solange noch Fehler im Programm sind.

Beispiel:

```
program RangeCheck;
type
  Digit = 0..9;
Var
  Dig1,Dig2,Dig3:digit;
begin
  Dig1:= 5;             gültig
  Dig2:= Dig1+3;        gültig, da Dig1 +3 <= 9.
  Dig3:= 47;            ungültig, aber ohne Fehlermeldung
  $R+Dig3:= 55;         ungültig, ergibt einen Laufzeitfehler
  $R-Dig3:= 167;        ungültig, aber ohne Fehlermeldung
end.
```

9. Strings

KCPASCAL bietet Ihnen den Datentyp String, um Zeichenketten zu verarbeiten. Zeichenketten sind eine Aneinanderreihung von Zeichen. Der Datentyp String ist strukturiert und in vielen dem Array (Abschnitt 10) sehr ähnlich. Es gibt allerdings einen großen Unterschied: die Anzahl der Zeichen in einem String, bzw. dessen Länge, kann dynamisch zwischen 0 und einer oberen Grenze variieren, während die Anzahl der Elemente in einem Array feststeht.

9.1 Definition des Strings

Die Definition des Datentyps String muß die obere Grenze der Anzahl der enthaltenen Zeichen, d.h. die Maximalmenge angeben. Die Definition besteht aus dem reservierten wort "string", dem in eckigen Klammern die Maximallänge folgt. Diese muß eine integer-Konstante zwischen 1 und 255 sein. Strings haben keine voreingestellte Länge, d.h., sie muß immer genau bestimmt werden.

```
Beispiel:
type
  FileName = string[14];
  ScreenLine = string[80];
```

Stringvariablen besetzen den Speicher in der definierten Maximallänge und zusätzlich ein Byte für die aktuelle Länge der Variablen. Die einzelnen Zeichen eines Strings sind, mit 1 beginnend, über die gesamte Länge durchnumeriert.

9.2 Stringausdruck

Strings werden mittels Stringausdrücken bearbeitet. Diese besteht aus Stringkonstanten, Stringvariablen, Funktionsbezeichnern und Operatoren.

Das Pluszeichen kann Strings verbinden. Die Funktion Concat (siehe Seite 71) macht das Gleiche, aber der Operator '+' ist oft einfacher zu handhaben. Sollte die Länge des entstehenden Strings größer als 255 sein, wird eine Laufzeitfehlermeldung ausgegeben.

```
Beispiel:
'KCPASCAL' + 'Pascal'   = 'KCPASCAL'
'123'+'.'+'456'         = '123.456'
'A'+'B'+'C'+'D'         = 'ABCD'
```

Die relationalen Operatoren (=,<>,>,<,>=,<=) haben eine geringere Prereferenz als der Verbindungsoperator. Wenn die relationalen Operatoren bei Stringoperanden angewendet werden, ist das Ergebnis vom Typ Boolean (True oder False). Beim Vergleich zweier Strings werden die einzelnen Buchstaben von links nach rechts miteinander verglichen. Wenn die Strings verschiedene Längen haben, und der kürzere bis hin zum letzten Buchstaben den am Anfang stehenden Zeichen des längeren Strings entspricht, dann wird der kürzere als der kleinere erkannt. Strings sind nur dann gleich, wenn sie sich sowohl im Inhalt als auch in der Länge entsprechen.

```
Beispiel:
'A'<'B'                                  wahr
'A'>'B'                                  falsch
'2'<'12'                                 falsch
'KCPASCAL' = 'KCPASCAL'                  wahr
'KCPASCAL ' = 'KCPASCAL'                 wahr
'Pascal Compiler'<'Pascal compiler'      wahr
```

9.3 Stringzuordnung

Der Zuordnungsoperator weist den Wert eines Stringausdruckes einer Stringvariablen zu.

```
Beispiel:
Age := 'fiftieth';
Line:= 'Many happy ENTERs on your' -Age- 'birthday';
```

Wenn die angegebene Maximallänge der Stringvariablen überschritten wird, werden die überzähligen Buchstaben verschluckt. Das heißt, wenn die obige Variable mit string 5 deklariert wurde, wird die Variable nach der Zuweisung nur die fünf Buchstaben, die links stehen, enthalten: 'fifti'.

9.4 Stringprozeduren

Die folgenden Standardstringprozeduren sind in KCPASCAL verfügbar:

9.4.1 Löschen

```
Syntax: Delete (St,Pos,Num)
```

Delete löscht aus einer Stringvariablen (St) eine bestimmte Anzahl (Num) von Buchstaben, beginnend bei der Position (Pos). Pos und Num sind Integer-Ausdrücke. Wenn Pos größer als die Länge von St ist, wird kein Buchstabe gelöscht. Wenn versucht wird, Buchstaben zu löschen, die sich die sich jenseits des rechten Endes des Strings befinden, d.h., Pos und Num sind größer als die Länge des Strings, werden nur Buchstaben innerhalb des Strings gelöscht. Wenn Pos außerhalb des Bereiches 0..255 ist, wird eine Laufzeitfehlermeldung ausgegeben.

Wenn St den Wert 'ABCDEFG' hat, dann nimmt St unter den folgenden Bedingungen nachstehende Werte an:

```
Delete(ST,2,4)    ergibt den Wert 'AFG'
Delete(St,2,10)   ergibt den Wert 'A'
```

9.4.2 Einfügen

```
Syntax: Insert(Obj,Target.Pos)
```

Insert fügt den String Obj in den String Target an der Position Pos ein. Obj ist eine Stringvariable und Pos ist eine integer Zahl. Wenn Pos größer als die Länge der Stringvariablen ist, dann wird der String-ausdruck an Target angefügt. Wenn das Ergebnis größer als die angegebene Maximallänge von Target ist, verschwinden die überzähligen Buchstaben und Target enthält lediglich die am weitesten links stehenden Buchstaben. Ist Pos außerhalb des Bereiches 0..255, wird eine Laufzeitfehlermeldung ausgegeben.

Wenn St den Wert 'ABCDEFG' hat, dann gibt Insert ('XX',St,3) St den Wert 'ABXXCDEFG'.

9.4.3 Str

Syntax: Str(Value,St)

Die Prozedur Str wandelt den numerischen Wert Value in einen String um und speichert das Ergebnis als St ab. Value ist ein Schreibparameter des Typs integer oder real, St ist eine Stringvariable. Schreibparameter sind Ausdrücke mit speziellen Formatierbefehlen (siehe Seite **111**).

```
Wenn I den Wert   1234 hat, gilt:
Str(I:5,St)       St erhält den Wert '1234'

Wenn X den Wert   2.5E4 hat, gilt:
Str(X:10:0,St)    St erhält den Wert '    2500'
```

Achtung: Eine Funktion, die die Str-Prozedur benutzt, darf nie durch einen Ausdruck in einer Write- oder Writeln-Anweisung aufgerufen werden.

9.4.4 Val

Syntax: Val(St,Var,Code)

Val wandelt den Stringausdruck St in einen integer- oder real-Wert in Var. St muß ein String sein, der einen numerischen Wert ausdrückt, entsprechend den Regeln bei numerischen Konstanten (siehe Seite 43). Weder davor, noch danach sind Leerzeichen erlaubt. Var muß eine integer- oder Real-Variable sein und Code eine integer-Variable sein. Wenn keine Fehler gefunden werden, wird die Variable Code auf 0 gesetzt. Ansonsten wird Code auf des erste fehlerhafte Zeichen gesetzt, und der Wert von Var ist undefiniert.

```
Wenn St den Wert '234' hat, gilt:
Val(St,I,Result)
  I erhält den Wert '234' und Result '0'

Wenn St den Wert '12x' hat, gilt:
Val(St,I,Result)
  I ist undefiniert und Result hat den Wert '3'

Wenn St den Wert '2.5E4' hat und X eine real-Variable ist, gilt:
Val(St,X,Result)
  X hat den Wert '2500' und Result '0'
```

CP/M 80-Benutzer: eine Funktion, die die Var-Prozedur benutzt, darf nie durch einen Ausdruck in einer Write- oder Writeln-Anweisung aufgerufen werden.

9.5 Stringfunktionen

Folgende Standardstringfunktionen sind in KCPASCAL anwendbar:

9.5.1 Copy

Syntax: Copy(St,Pos,Num)

Copy gibt einen Teilstring eines Strings (St) aus, der eine bestimmte Anzahl (Num) von Zeichen enthält, gezählt von der Position Pos. St ist ein Stringausdruck, Pos und Num sind integer-Ausdrücke. Wenn der Wert von Pos die Länge des Strings übersteigt, wird ein leerer Teilstring ausgegeben. Wenn versucht wird, Zeichen jenseits des Endes des Strings zu erhalten, d.h., der Wert von Pos + Num übersteigt die Länge des Strings, werden nur noch die innerhalb eines Strings befindlichen Zeichen ausgegeben. Wenn Pos außerhalb des Bereiches 0..255 ist, erfolgt eine Laufzeitfehlermeldung.

Wenn St den Wert 'ABCDEFG' hat, gilt:
Copy(St,3,2) gibt den Wert 'CD' aus
Copy(St,4,10) gibt den Wert 'DEFG' aus
Copy(St,4,2) gibt den Wert 'DE' aus

9.5.2 Concat

Syntax: Concat(St1,St2,StN)

Die Funktion Concat gibt einen Gesamtstring aus, der aus beliebig vielen Einzelstrings in der angegebenen Ordnung (St1..StN) zusammengesetzt wird. Ist das Ergebnis größer als 255, wird eine Laufzeitfehlermeldung ausgegeben. Wie auf Seite 68 schon besprochen wurde, kann der Operator '+' das Gleiche und unter Umständen sogar auf einfachere Art und Weise. Concat wurde in KCPASCAL aufgenommen, um die Kompatibilität zu erhalten.

Wenn St1 den Wert 'KC' und St2 den Wert 'ist am schnellsten' hat, ergibt:

Concat(St1,'PASCAL ',St2)

den Wert 'KCPASCAL ist am schnellsten'

9.5.3 Length

Syntax: Length(St)

Gibt die Länge des Stringausdruckes Str aus, d.h. die Anzahl der darin enthaltenen Zeichen. Das Ergebnis ist integer.

Wenn St den Wert '123456789' hat, ergibt:
Length(St) den Wert 9

9.5.4 Pos

Syntax: Pos(Obj,Target)

Diese Funktion durchsucht den String Target nach dem ersten Vorkommen des Stringausdruckes Obj. Das Ergebnis ist integer und bezeichnet die

Position im Stringausdruck Target, die das erste Zeichen von Obj innehat. Die Position des ersten Zeichens im String ist '1'. Wird die Zeichenkombination nicht gefunden, liefert Pos den Wert '0'.

Wenn St den Wert 'ABCDEFG' hat, ergibt:

```
Pos('DE',St)      den Wert '4'
Pos('H',St)       den Wert '0'
```

10. Arrays

Ein Array ist ein strukturierter Datentyp mit einer festgesetzten Anzahl von Komponenten, die alle vom gleichen Typ sind, dem Grundtyp. Auf jede Komponente kann mit Indizes zugegriffen werden. Indizes sind integer-Ausdrücke, die in eckige Klammern hinter den Arraybezeichnern stehen. Ihr Datentyp wird Indextyp genannt.

10.1 Arraydefinition

Die Definition eines Arrays besteht aus dem ersten reservierten Wort "array", dem der Indextyp in eckigen Klammern folgt. Danach steht das reservierte Wort "of" gefolgt vom Grundtyp.

Beispiele:

```
type
  Day = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
Var
  Workhour  :[array]1..8 of Integer;
  Week      :[array]1..7 of Day;

type
  Players   =(Player1, Player2, Player3, Player4);
  Hand      =(One, Two, Pair,, TwoPair, Three, Straight,
              Flush, FullHouse, Four, StraigthFlush, RSF);
  LegalBild =1..200;
  Bid       = array[Players] of LegalBid;
Var
  Player    : array[Players] of Hand;
  Pot       : Bid;
```

Auf eine Arraykonstante greift man zu, indem ein Index in eckigen Klammern an den Variablenbezeichner des Arrays gehängt wird:

```
Player Player3 := FullHouse;
Pot Player3 := 100;
Player Player4 := Flush;
Pot Player4 := 50;
```

Da eine Zuweisung zwischen zwei beliebigen Variablen erlaubt ist, solange sie vom gleichen Datentyp sind, können ganze Arrays mit einer einzigen Zuweisungsanordnung kopiert werden.

Der Compilerbefehl R kontrolliert bei der Codegenerierung, ob die Arrays im zulässigen Bereich liegen. Nach Voreinstellung ist er inaktiv. R+ verursacht eine Überprüfung aller Indexausdrücke auf die Einhaltung der Grenzen ihres Indextyps.

10.2 Multidimensionale Arrays

Die Komponenten eines Arrays können beliebigen Datentyps sein, d.h., daß die Komponenten auch Arrays sein können. Eine solche Struktur nennt man multidimensionales Array.

```
Beispiel:
type
  Card      = (Two, Three, FourFive, Six, Seven, Eigth, Nine, Ten,
               Knigth, Queen, King, Ace);
  Suit      = (Hearts, Spade, Clubs, Diamomds);
  AllCards  = array Suit of array 1..13]] of Card;
Var
  Deck:  AllCards;
```

Ein multidimensionales Array kann auch einfacher definiert werden:

```
type
  AllCards = array Suit[1..13] of Card;
```

Eine ähnliche Kurzform kann bei der Wahl der Arraykomponenten gewählt werden:

DecköHearts,10 entspricht DeckHearts ö10

Es ist natürlich auch möglich, multidimensionale Arrays in der Form von vordefinierten Arraytypen zu benutzen.

```
Beispiel:
type
  Pupils    = string 10;
  Class     = array[1..30] of Pupils;
  School    = array[1..100] of class;
Var
  J.P.Vacant      :Integer;
  ClassA,
  ClassB          :Class;
  NewTownSchool   :School;
```

Nach diesen Definitionen sind alle folgenden Zuweisungen möglich:

10.3 Zeichenarrays

Zeichenarrays sind Arrays mit einem Index und Komponenten des skalaren Standardtyps Char. Zeichenarrays können als String mit konstanter Länge gedacht werden.

Bei KCPASCAL können Zeichenarrays an Stringausdrücken teilnehmen. In diesem Fall wird das Array in einen String der gleichen Länge umgewandelt: So können Arrays auf die gleiche Art und Weise wie Strings verglichen und behandelt, und Stringkonstanten können Zeichenarrays zugewiesen werden, solange sie die gleiche Länge haben. Stringvariable und Werte aus Stringausdrücken können nicht den Zeichenarrays zugewiesen werden.

10.4 Vordefinierte Arrays

KCPASCAL bietet zwei vordefinierte Arrays vom Type Byte, Mem und Port, die als direkter Zugang zum CPU-Speicher und zu den Daten-Ports benutzt werden können. Diese werden im Kapitel 11 besprochen.

11. Recordtyp (Satzart)

Ein Record ist eine Datenstruktur, die aus einer festgelegten Zahl von Komponenten besteht. Der Zusammenschluß mehrerer Felder als Record wird auch Verbund genannt. Die Felder können aus verschiedenen Datentypen bestehen, und jedes wird mit einem Feldbezeichner (field identifier) benannt. Dieser dient der Feldselektion in einem Record.

11.1 Definition des Records

Die Definition des Datentyps Record besteht aus dem reservierten Wort "record", dem eine Auflistung der einzelnen Felder (field list) folgt. Danach steht das reservierte Wort "end".
Die Felderauflistung ist eine Folge von Sätzen (record sections), die durch Strichpunkte getrennt werden. Jeder Satz besteht aus einem oder mehreren Bezeichnern, gefolgt von einem Doppelpunkt und einem Datentyp-bezeichner. So bestimmt jeder Satz den Typ und den Bezeichner für ein oder mehrere Felder.

```
Beispiele:
type
  DaysOfMonth: record
                 Day: 1..31;
                 Month: (Jan, Feb, Mar, Apr, May, Jun,
                         Jul, Aug, Sep, Oct, Nov, Dec);
                 Year: 1900..1999;
               end;
Var
  Birth: Date;
  WorkDay: array[1..5] of date;
```

Day, Month und Year sind Feldbezeichner. Ein Feldbezeichner ist nur für den Record spezifisch, in dem er definiert wurde. Ein Feld wird durch den Variablenbezeichner und den Feldbezeichner, beide durch einen Punkt getrennt, angesprochen.

```
Beispiele:
Birth.Month:=Jun;
Birth.Year:=1950;
WorkDay[Current]:=WorkDay[Current-1];
```

Beachten Sie, daß, ähnlich wie bei Arrays, eine Zuweisung zwischen ganzen Records des gleichen Typs möglich ist. Da jede einzelne Komponente beliebigen Typs sein kann, sind deshalb folgende Konstruktionen der Ineinanderschachtelung möglich.

```
type
  Name      =     record
                    FamilyName: string[32];
                    ChristianName: array[1..3] of string[16];
                  end;
  Rate      =     record
                    NormalRate, OverTime;
                    NigthTime, Weekend:Integer;
                  end;
  Date      =     record
                    Day: 1..31;
```

```
                    Month: (Jan, Feb, Mar, Apr, May, Jun,
                          Jul, Aug, Sep, Oct, Nov, Dec);
                    Year: 1900..1999;
                  end;
Person = record
           ID:Name;
           Time:Date;
         end;
Wages  = record
           Individual Person;
           Cost:Rate;
         end;

VarSalary, Fee:Wages;
```

Nach diesem Beispiel wären folgende Zuweisungen möglich:

```
Salary := Fee;
Salary.Cost.Overtime := 950;
Salary.Individual.Time := Fee.Individual.Time;
Salary.Individual.ID.FamilyName := Smith;
```

11.2 With-Anweisung

Der oben beschriebene Gebrauch von Records führt manchmal zu relativ langen Anweisungen. Es wäre einfacher, wenn man auf die einzelnen Felder in einem Record wie auf einfache Variablen zugreifen könnte. Dies ist die Funktion der with-Anweisung. Sie eröffnet einen Record, so daß die Feldbezeichner wie Variablenbezeichner benutzt werden können.

Eine with-Anweisung besteht aus dem reservierten Wort "with", gefolgt von einer Auflistung der Recordvariablen, die mit Kommas getrennt sind. Darauf folgt das reservierte Wort "do" und eine Anweisung.

Innerhalb einer with-Anweisung ist ein Feld lediglich durch den Feldbezeichner bestimmt und nicht durch den Variablenbezeichner des Records.

```
with Salary do
begin
  Individual := NewEmployer;
  Cost := StandardRates;
end;
```

Records können innerhalb einer with-Anweisung geschachtelt werden, d.h., daß Records wie folgt eröffnet werden können:

```
with Salary, Individual, ID do
begin
  FamilyName := 'Smith';
  ChristianNames1 := 'James';
end;
```

Dies entspricht:

```
with Salary do with Individual do with ID do
```

11.3 Varianten (variant records)

Die Syntax eines Records erlaubt auch die Verwendung von Varianten, d.h. von alternativen Strukturen, bei denen sich die Recordfelder aus einer unterschiedlichen Anzahl und unterschiedlichen Typen von Komponenten zusammensetzen. Das hängt gewöhnlich vom Wert eines Variantenmarkierfeldes (engl.: tag-field) ab.

Ein Variantenvorteil besteht aus dem Variantenmarkierfeld eines zuvor definierten Typs, dessen Wert die jeweilige Variante bestimmt. Ihm folgen Labels, die jedem möglichen Wert des Variantenmarkierfeldes entsprechen. Jedes Label steht einer Auflistung der Felder voran, die entsprechenden Variantentyp definiert.

Angenommen es sei folgender Typ gegeben:

```
Origin = (Citizen, Alien);
```

Hinzu kommen die Datentypen Name und Date. Nachstehender Record erlaubt nun dem Feld CitizenShip verschiedene Strukturen anzunehmen, je nachdem ob der Wert des Feldes Citizen oder Alien ist:

```
type
  Person =  record
            PersonName : Name;
            BirthDate: Date;
            case CitizenShip:Origin of Citizen: (BirthPlace:Name);
            Alien: (CountryOfOriginal:Name;
                    DateOfEnty:Date;
                    PermittedUntil:Date;
                    PortOfEntry:Name;
            end;
```

In dieser Variantendefinition ist das Variantenmarkierfeld ein eigenes Feld, das wie jedes andere Feld behandelt werden kann. Ist Passenger eine Variable des Typs Person, sind folgende Anweisungen durchaus möglich:

```
Passenger, CitizenShip := Citizen;

with Passenger, PersonName do
  if CitizenShip = Alien then writeln (FamilyName);
```

Der feststehende Teil eines Records, d.h. der Teil, der die normalen Felder beinhaltet, muß immer dem Variantenteil voranstehen. Im obigen Teil sind PersonName und BirthDate die feststehenden Felder. Ein Record kann nur einen einzigen Variantenteil haben. Auf jeden Fall muß eine Variante Klemmern haben, auch wenn nichts darin steht.

Der Programmierer muß darauf achten, daß die Variantenmarkierfelder tatsächlich vorhanden sind. In KCPASCAL kann nämlich auch auf das Feld Date Of Entry zugegriffen werden, wenn der Wert des Variantenmarkierungsfeldes nicht Alien ist. Es ist sogar möglich, alle Feldbezeichner wegzulassen und nur die Bezeichner der Datentypen anzugeben.

Derartige Rekordvarianten werden freie Verbindungen (free unions) genannt, im Unterschied zu solchen mit Variantenmarkierfeld (discriminated unions). Die freien Verbindungen werden nicht mehr häufig angewandt, und unerfahrene Programmierer sollten sie meiden.

12. Mengen

Eine Menge (set) ist eine Zusammenfassung mehrerer Objekte des selben Typs, die als Ganzes gedacht werden. Jedes einzelne Objekt einer solchen Menge wird Element (member, element) genannt. Einige Beispiele:

1) Alle integer Zahlen zwischen 1 und 100

2) Alle Buchstaben des Alphabets

3) Alle Konsonanten des Alphabets

Zwei Mengen sind nur dann gleich, wenn auch ihre Elemente die gleichen sind.
Es gibt in ihnen keine Rangordnung, do daß die Mengen [1,3,5],[1,5,3] und [5,3,1] gleich sind. Wenn die Elemente einer Menge auch die Menge einer anderen sind, gilt die erste Menge als in der zweiten enthalten. Bei obigem Beispiel 3) in 2) enthalten.

Mit Mengen kann man auf drei verschiedene Arten rechnen (ähnlich der Addition, der Subtraktion und der Multiplikation mit Zahlen).

Die Vereinigung (union, sum) zweier Mengen A und B (geschrieben (A+B) ist die Menge, deren Elemente entweder in A oder in B enthalten sind. Die Vereinigung von [1,3,4,5,7] und [2,3,4] ist [1,2,3,4,5,7].

Der Durchschnitt (intersection, product) zweier Mengen A und B (geschrieben A^B) ist die Menge, die in den beiden Mengen gemeinsam ist. Der Durchschnitt von [1,3,4,5,7] und [2,3,4] ist [3,4].

Die Differenz oder das Komplement zweier Mengen, das relative Komplement (geschrieben A-B) ist die Menge der Elemente der zuerst angegebenen Menge, die nicht auch in der zweiten Menge enthalten ist. Die Differenz von [1,3,5,7] und [2,3,4] ist [1,5,7].

12.1 Mengendefinition

Obwohl es in der Mathematik keine Beschränkungen gibt, welche Elemente in einer Menge sein können, hat Pascal doch einige Restriktionen. Die Elemente einer Menge müssen alle vom gleichen Typ, dem Grundtyp sein, und dieser muß ein einfacher Datentyp sein, d.h., jeder skalare Datentyp, außer dem reellen. Einer Menge steht das reservierte Wort "set of" voran, gefolgt von einem einfachen Datentyp.

Beispiele:
```
type
  DaysOfMonth = set of 0..31;
  WorkWeek = set of Mon..Fri;
  Letter = set of 'A'..'Z';
  Additive Colors = set of (Red,Green,Blue);
  Charakters = set of Char;
```

Bei KCPASCAL können maximal 256 Elemente in einer Menge enthalten sein, und die Ordinalwerte des Grundtyps müssen zwischen 0 und 255 liegen.

12.2 Mengenausdrücke

Die Werte einer Menge können mit den Werten einer anderen Menge über Mengenausdrücke einer Rechenoperation unterliegen. Mengenausdrücke bestehen aus Mengenkonstanten, Mengenvariablen, den Angaben der Menge und den Mengenoperationen.

12.2.1 Angabe der Menge

Die Mengenangabe besteht aus einem oder mehreren, durch Kommas getrennten und in eckigen Klammern stehenden Elementbestimmungen. Eine Elementbestimmung ist ein Ausdruck vom gleichen Datentyp wie der Grundtyp der Menge. Oder es ist ein Bereich, der durch zwei solche Ausdrücke, zwischen denen zwei aufeinanderfolgende Punkte stehen, bestimmt wird.

```
Beispiele:
['T','U','R','B','O']
['X,Y']
[X..Y]
[1..5]
['A'..'Z','a'..'z','0'..'9']
[1,3..10,12]
[]
```

Das letzte Beispiel zeigt eine leere Menge, die, da sie keine Ausdrücke enthält, die ihren Grundtyp angeben würde, zu allen Mengentypen kompatibel ist. Die Menge 1..5 ist äquivalent zu der Menge 1,2,3,4,5. Wenn X > Y, dann steht X..Z für eine leere Menge.

12.2.2 Mengenoperatoren

Die Regeln der Mengenbildung geben den Vorrang der Mengenoperatoren nach den drei verschiedenen Klassen von Operatoren an:

1) ^ Durchschnitt der Mengen

2) + Vereinigung der Mengen
   - Differenz der Mengen

3) < > Test auf Gleichheit der Mengen
   < > Test auf Ungleichheit der Mengen
   >= Inklusion ('enthält') ist wahr, wenn der zweite Operand im ersten enthalten ist.
   <= Inklusion ('ist enthalten') ist wahr, wenn der erste Operand im zweiten enthalten ist.
   IN Test auf Mitgliedschaft in einer Menge. Der zweite Operand ist eine Menge und der erste Operand ein Ausdruck des gleichen Typs, wie der Grundtyp der Menge. Das Ergebnis ist wahr (true, wenn der erste Operand ein Element des zweiten ist.

Es gibt keinen Operator für die Exklusion, aber man kann ihn wie folgt programmieren:

```
A ^ B = []
```

Mengenausdrücke können zur Vereinfachung komplizierter Tests sehr hilfreich sein:

```
if (Ch='T')or(CH='U')or(Ch='R')or(Ch='B')or(Ch='O')
```

kann auch klarer ausgedrückt werden:

```
Ch in ['T','U','R','B','O']
```

Und der Test

```
if(Ch >='0' and CH <='9') then ...
```

sähe folgendermaßen besser aus:

```
if Ch in ['0'..'9'] then ...
```

12.3 Mengenzuweisungen

Werte, die von Mengenausdrücken kommen, werden den Mengenvariablen mittels des Zuweisungsoperators := zugewiesen.

```
Beispiele:
type
  ASCII = set of 0..127;
var
  NoPrint,Print,AllChars:ASCII;
begin
  AllChars:= 0..127;
  NoPrint:= 0..31;
  Print:= AllChars - NoPrint;
end;
```

13. Typisierte Konstanten

Typisierte Konstanten sind eine Besonderheit von KCPASCAL.
Sie können genauso benutzt werden, wie eine Variable des gleichen Typs. Typisierte Konstanten können also als initialisierte Variablen benutzt werden, da der Wert einer typisierten Konstanten definiert ist, während der Wert einer Variablen undefiniert ist, solange sie keinen Wert zuweisen, wenn sie tatsächlich als Konstante verwendet werden sollen.

Die Benutzung von typisierten Konstanten verkleinert den Code, wenn sie oft im Programm gebraucht werden, da sie nur einmal im Programmcode auftauchen, während eine untypisierte Konstante, jedesmal, wenn sie benutzt wird, im Programm angegeben werden muß.

Typisierte Konstanten sind wie nicht-typisierte (siehe Seite 48) definiert, mit dem Unterschied, daß die Definition nicht nur den Wert, sondern auch den Typ angibt. In der Definition folgen dem Bezeichner der typisierten Konstanten ein Doppelpunkt und der Bezeichner des Datentyps. Danach steht ein Gleichheitszeichen und die aktuelle Konstante.

13.1 Unstrukturierte typisierte Konstanten

Eine unstrukturierte typisierte Konstante ist wie ein skalarer Datentyp definiert:

```
const
  NumberOfCars:Integer = 1267;
  Interest:Real = 12.67;
  Heading:string[7]= 'SECTION';
  Xon:Char = ^Q;
```

Im Unterschied zu nicht-typisierten, können typisierte Konstanten anstelle von Variablen als Variablenparameter einer Prozedur oder Funktion stehen. Da eine typisierte Konstante in Wirklichkeit eine Variable mit konstantem Wert ist, kann sie nicht in der Definition anderer Konstanten oder Typen verwendet werden. Sind Min und Max typisierte Konstanten, ist folgender Konstrukt nicht zulässig:

```
const
  Min:Integer = 0;
  Max:Integer = 50;
type
  Range:array[Min..Max] of integer;
```

13.2 Strukturierte typisierte Konstanten

Strukturierte Konstanten umfassen Array-Konstanten, Record-Konstanten und Mengenkonstanten. Sie werden oft für initialisierte Tafeln und Mengen im Testbereich, für Umwandlungen und für Mappingfunktionen benutzt. Die folgenden Abschnitte beschreiben jeden einzelnen Typ im Detail.

13.2.1 Array-Konstanten

Die Definition einer Array-Konstanten besteht aus dem Konstantenbezeichner, einem Doppelpunkt und dem Typenbezeichner eines zuvor definierten

Arraytyps, dem ein mit Kommas getrennte und in Klammern stehende menge von Konstanten ausgedrückt.

```
Beispiel:
type
  Status = (Active,Passive,Waiting);
  StrinRep = array [Status] of string [7];
const
  Stat:StringRep = ('active','passive','waiting');
```

Dieses Beispiel definiert die Array-Konstante Stat, die z.B. Werte des skalaren Datentyps Status in die entsprechenden Strings umwandelt. Die Komponenten von Stat sind:

```
Stat[Active]     = 'active'
Stat[Passive]    = 'passive'
Stat[Waiting]    = 'waiting'
```

Der Typ einer Komponente ist beliebig. Ausgeschlossen sind lediglich File und Pointer. Array-Konstanten aus Zeichen können entweder als einfache Zeichen oder als Strings bestimmt werden. Also kann folgende Definition

```
const
  Digits:array[0..9] of Char = ('0','1','2','3','4','5','6','7','8','9');
```

auch einfacher ausgedrückt werden:

```
const
  Digits:array[0..9] of Char = '1234567890';
```

13.2.2 Multidimensionale Array-Konstanten

Multidimensionale Array-Konstanten werden definiert, indem die Konstanten jeder einzelnen Definition als eigenständige Mengen in Klammern ausgegeben werden. Die am weitesten innen stehenden Konstanten korrespondieren dabei zu den am weitesten rechts stehenden Dimensionen.

```
Beispiel:
type
  Cube = array [0..1,0..1,0..1] of integer;
  const
  Maze:Cube = (((0,1),(2,3)),((4,5),(6,7)));
begin
  Writeln(Maze[0,0,0],' = 0');
  Writeln(Maze[0,0,1],' = 1');
  Writeln(Maze[0,1,0],' = 2');
  Writeln(Maze[0,1,1],' = 3');
  Writeln(Maze[1,0,0],' = 4');
  Writeln(Maze[1,0,1],' = 5');
  Writeln(Maze[1,1,0],' = 6');
  Writeln(Maze[1,1,1],' = 7');
end.
```

13.2.3 Record Konstanten

Die Definition einer Record-Konstanten besteht aus einem Konstantenbezeichner, einem Doppelpunkt und dem Typenbezeichner eines zuvor definierten Recordtypen. Darauf folgt ein Gleichheitszeichen und der Wert

der Konstanten. Der Wert wird durch eine Liste von Feldkonstanten ausgedrückt, die in runden Klammern stehen. Die einzelnen Komponenten sind durch Kommas getrennt.

Beispiel:
```
type
  Point     = record
              X,Y,Z:integer;
              end;
  OS        = (CPM80,CPM86,MSDOS,UNIX);
  UI        = (CCP,SomethingElse,MenueMaster);
  Computer  = record
                OperatingSystem:array[1..4] of OS;
                UserInterface:UI;
              end;

const
  Origo:Point = (X:0 ;Y:0 ;Z:0);
  SuperComp:Computer =
                    (OperatingSystems:(CPM80,CPM86,MSDOS,UNIX);
                    UserInterface:MenueMaster);
  Planel:array[1..3] of Point =
                    ((X:1;Y:4;Z:5),(X:10;Y:-78;Z:45),(X:100;Y:10;Z:-78));
```

Die Feldkonstanten müssen in der gleichen Ordnung angegeben werden, die mit Kommas voneinander getrennt sind und in eckigen Klammern stehen. Ein Element muß eine Konstante oder ein Ausdruck eines Bereiches sein, der aus zwei Konstanten, die mit zwei aufeinanderfolgenden Punkten getrennt sind, besteht.

Beispiel:
```
type
  Up  = set of 'A'..'Z';
  Low = set of 'a'..'z';
const
  UpperCase:Up = ['A'..'Z'];
  Vocals   :Low = ['a','e','i','o','u','y'];
  Delimiter:set of Char = ['!'..'/',':'..'?'];
```

14. Ein- und Ausgabe

14.1 Logische Geräteeinheiten

Folgende Geräteeinheiten sind möglich:

CON:
Konsole (Console). Der Output wird über das Betriebssystem an das Ausgabegerät gegeben, gewöhnlich an den Bildschirm, der Input erfolgt über das Eingabegerät, gewöhnlich die Tastatur.

TRM:
Terminal. Der Output wird normalerweise an den Bildschirm gesendet. Input wird üblicherweise von der Tastatur aufgenommen. Es findet ein Echo der eingegebenen Zeichen statt, solange es keine Kontrollzeichen sind. Das einzige Kontrollzeichen mit Echo ist ein Carriage ENTER (CR). Dessen Echo ist CR/LF (Carriage Return, Line Feed).

KBD:
Tastatur (Keyboard). Üblicherweise steht KBD für Tastatur. Der Input hat kein Echo.

LST:
Lister. Üblicherweise wird damit der Drucker bezeichnet.

AUX:
Alternative Auxiliary. (RDR: und PUN:)

USR:
Benutzer (User). Output wird an die Output-Routine des Benutzers gesendet. Input kommt von seiner Input-Routine.

14.2 Standarddateien

KCPASCAL bietet eine Anzahl vordeklarierter Textdateien, die für die Bearbeitung vorbereitet sind.

Input	Die Inputdatei erster Ordnung. Diese Datei ist entweder dem CON:Gerät zugeordnet oder dem TRM:Gerät (weitere Erläuterungen siehe unten).
Output	Die Outputdatei erster Ordnung. Diese Datei ist entweder dem CON:Gerät oder dem TRM:Gerät zugeordnet (weitere Erläuterungen siehe unten).
Con	Der Konsole zugeordnet (CON:).
Trm	Dem Terminal zugeordnet (TRM:).
Kbd	Der Tastatur zugeordnet (KBD:).
Lst	Dem Ausgabegerät (Drucker) zugeordnet (LST:).
Aux	Alternativ verwendbar (AUX:).
Usr	Dem Benutzer zugeordnet (USR:).

Wenn Eingaben nicht automatisch auf dem Bildschirm angezeigt werden sollen, sollten sie von der Standarddatei KBD aus gemacht werden:

```
Read (kbd, var)
```

Da die Standarddateien Input und Output sehr häufig benutzt werden, werden sie durch die Voreinstellung automatisch gewählt, falls kein Datentyp definiert wird. Die folgende Tabelle zeigt die Textdateioperationen und ihre Entsprechungen:

Write (Ch)	Write (Output, Ch)
Read (Ch)	Read (Input, Ch)
Writeln	Writeln (Output)
Readln	Readln (Input)

14.3 Read-Prozedur

Die Read-Prozedur ermöglicht die Eingabe von Buchstaben, Strings und Zahlen. Die Syntax der Raed-Anweisung ist:

```
Read (Var1, Var2, ... , VarN)
```

oder

```
Read (FilVar, Var1, Var2, ... , VarN)
```

wobei Var1, Var2, ..., VarN Variable vom Typ Char, Strings, Integer oder Real sind. Im ersten Fall sind die Variablen Eingaben von der Standard-datei Input, gewöhnlich der Tastatur, im zweiten Fall sind die Variablen Eingaben von einer Textdatei, die vorher zum FilVar erklärt und zum Lesen aufbereitet worden sind.

14.4 Readln-Prozedur

```
Readln (Var1, Var2, ... , VarN)
```

Nach einem Readln wird das folgende Read oder Readln am Anfang der nächsten Zeile zu lesen beginnen.

14.5 Write-Prozedur

Mit der Write-Prozedur können Zeichen, Strings, Bool'sche Werte und Zahlen ausgegeben werden.
Die Syntax der Write-Anweisung ist wie folgt:

```
Write (Var1, Var2, ... , VarN)
```

Dabei sind Var1, Var2, ... , VarN Variable vom Typ Char, String, Boolean, Integer oder Real, denen wahlweise in Komma oder ein ganzzahliger Ausdruck folgen soll, mit denen die Größe des Ausgabefeldes bestimmt wird. Es werden die Variablen zur Standarddatei Output ausgegeben, gewöhnlich dem Bildschirm.

Das Format der Write-Parameter hängt vom Variablentyp ab. es folgt eine Beschreibung der unterschiedlichen Formate, ihrer Bedeutung und ihrer Symbole:

l,m,n	bezeichnet Ausdrücke vom Typ Integer.
R	bezeichnet Ausdrücke vom Typ Real.
Ch	bezeichnet Ausdrücke vom Typ Char.
S	bezeichnet Ausdrücke vom Typ String und
B	bezeichnet Ausdrücke vom Typ Boolean.
Ch	Das Zeichen Ch wird ausgegeben.

Ch:n Das Zeichen Ch wird in einem n Zeichen langen Feld rechtsbündig plaziert. Das restliche Feld wird mit Leerzeichen aufgefüllt.

S Der String wird ausgegeben. Arrays von Zeichen können ebenfalls ausgegeben werden, wenn sie mit Strings kompatibel sind.

S:n Der String S wird in einem n Stellen langen Feld rechtsbündig plaziert. Das restliche Feld wird mit Leerzeichen aufgefüllt.

B Abhängig vom Wert von B wird entweder das Wort TRUE oder das Wort FALSE ausgegeben.

B:n Abhängig vom Wert von B wird entweder das Wort TRUE oder das Wort FALSE in einem Feld rechtsbündig plaziert, das n Zeichen umfaßt.

I:n Die Dezimaldarstellung des Wertes von i wird in einem Feld rechtsbündig plaziert, das n Zeichen umfaßt.

R:n:m Die Dezimaldarstellung des Wertes von R wird ausgegeben und in einem n Stellen langen Feld rechtsbündig plaziert, wobei das Festkommaformat, mit m Stellen nach dem Dezimalpunkt verwandt wird, m muß im Wertebereich von 0 bis 24 liegen; andernfalls wird das Gleitkommaformat verwandt. Um die Feldgröße n zu füllen, gehen der Zahl entsprechend viele Leerzeichen voraus.

14.6 Writeln-Prozedur

Die Writeln-Prozedur entspricht der Write-Prozedur, mit der Ausnahme, daß nach dem letzten Wert eine CR/LF-Sequenz ausgegeben wird.

```
Writeln
```

14.7 I/O-Fehlerroutinen

Der I Compilerbefehl wird benutzt, um die Art der I/O Überwachung einzustellen. Voreingestellt ist in (*$I+*), d.h., daß jede I/O-Operation sofort nach der Ausführung überprüft wird. I/O-Fehler verursachen dann einen Programmabbruch, und es erscheint Fehlermeldung die die Art des Fehlers anzeigt.

Wenn die automatische I/O-Überwachung nicht aktiviert ist, d.h., bei (*$I-*), wird keine Laufzeitprüfung vorgenommen. Ein I/O-Fehler ruft dann keinen Programmabbruch hervor, jedoch werden alle weiteren I/O-Operationen unterbunden, bis die Standardfunktion IOresult aufgerufen wird. Diese Funktion stellt den Zustand vor Auftreten des Fehlers wieder her und Eingabe/Ausgabe kann wieder stattfinden. Es obliegt dem Programmierer, den I/O-Fehler zu beheben. Antwortet IOresult mit einer Null, zeigt dies den fehlerfreien Ablauf einer Operation an. Jede andere Antwort bedeutet, daß die letzte I/O-Operation fehlerhaft war. Im Anhang I sind alle Fehlermeldungen und ihre Codes aufgelistet. Beachten Sie, daß

bei Aufruf von IOresult der Zustand vor Auftreten des Fehlers hergestellt wird. Erneute Aufrufe von IOresult werden solange die Antwort Null erzeugen, bis der nächste I/O-Fehler auftritt.

15. Zeiger-Typen (Pointer)

Die bisher diskutieren Variablen waren statischer Natur, d.h., ihre Form und Größe ist vorbestimmt und wird während der gesamten Bearbeitung des Abschnitts, für den sie definiert wurden, aufrechterhalten. Oft erfordern Programme jedoch eine Datenstruktur, die während der Bearbeitung in Form und Größe veränderlich sein sollte. Diesem Zweck dienen dynamische Variablen. Sie können bei Bedarf aufgerufen werden und entfallen, wenn sie nicht mehr benötigt werden.

Diese dynamischen Variablen werden nicht wie die statischen mittels einer Variablendeklarierung aufgerufen, und sie lassen sich nicht über einen Bezeichner direkt zitieren. Stattdessen wird eine besondere Variable, die die Speicheradresse der Variablen enthält, benutzt, um auf die Variable zu zeigen. Diese besondere Variable heißt Zeigervariable.

15.1 Definition der Zeigervariablen

Ein Zeigertyp wird durch das Zeigersymbol ^ definiert, dem der Typenbezeichner der dynamischen Variablen folgt, der durch eine Zeiger-variable dieses Typs zitiert werden kann.

Im Folgenden wird gezeigt, wie Records mit verwandten Zeigern angelegt werden können. Der Typ PersonPointer ist definiert als Zeiger von Variablen des Typs PersonRecord:

```
type
  PersonPointer = ^PersonRecord;

  PersonRecord = record
                       Name: string[50];
                       Job: string[50];
                       Next: PersonPointer;
                     end;

Var
  FirstPerson,LastPerson,NewPerson: PersonPointer;
```

Die Variablen NextPerson, LastPerson und NewPerson sind jene Zeigervariablen, die auf Records vom Typ PersonRecord zeigen können. Wie man sieht, kann sich die Typenbezeichnung in einer Definition vom Typ Zeiger auf eine Bezeichnung beziehen, die noch nicht definiert wurde.

15.2 Zuordnung von Variablen (NEW)

Bevor irgendwelche von diesen Zeigervariablen benutzt werden, muß man natürlich einige Variablen haben, auf die man zeigen kann. Neue Variablen, egal von welchem Typ, werden mit der Standardprozedur New be-zeichnet. Diese Prozedur hat einen Parameter, der den Variablen, die definiert werden sollen, den Typ zuweist.

Eine neue Variable vom Typ PersonRecord wird also folgendermaßen definiert:

```
New (FirstPerson);
```

mit dem Ergebnis, daß FirstPerson auf einen dynamisch zugeordneten Record vom Typ PersonRecord weist.

Zuweisungen zwischen Zeigervariablen können vorgenommen werden, solange die Zeiger vom gleichen Typ sind. Zeiger vom gleichen Typ können auch mit den logischen Operatoren = und <> untereinander verglichen werden, wobei sich als Ergebnis ein Bool'scher Wahrheitswert ergibt (true bzw. false).

Die Funktion "nil" ist mit allen Typen von Zeigern kompatibel, nil zeigt auf keine dynamische Variable und kann Zeigervariablen zugewiesen werden, um die Abwesenheit eines brauchbaren Zeigers anzuzeigen, nil kann auch in Vergleichen benutzt werden.

Variable, die durch die Standardprozedur New definiert wurden, werden in einer stapelartigen Struktur, "heap" genannt, abgelegt. Das KCPASCAL-System kontrolliert den Heap, indem es einen Heapzeiger erhält, der bei Beginn eines Programms auf die Adresse des ersten freien Bytes im Speicher initialisiert wird. Bei jedem Aufruf von New wird der Heapzeiger an die Spitze des freien Speichers gesetzt, entsprechend der Anzahl der Bytes , die der Größe der neuen dynamischen Variablen entspricht.

15.3 Mark und Release

Wenn eine dynamische Variable nicht länger benötigt wird, benutzt man die Standardprozeduren "Mark" und "Release", um den diesen Variablen zugewiesenen Speicherplatz wieder freizumachen. Die Mark-Prozedur weist den Wert des Heapzeigers einer Variablen zu. Die Syntax eines Aufrufs von Mark ist:

```
Mark (Var);
```

Dabei ist Var eine Zeigervariable. Die Release-Prozedur setzt den Heap-zeiger an die in ihren Argumenten enthaltene Adresse. Die Syntax lautet:

```
Release (Var);
```

wobei Var eine Zeigervariable ist, die zuvor durch Mark gesetzt wird. Release entfernt dann alle dynamischen Variablen oberhalb dieser Adresse, kann aber nicht den durch Variablen benutzten Platz in der Mitte des Heap freimachen. Wenn Sie das tun möchten, sollten Sie anstatt von Mark/Release "Dispose" verwenden (Seite 124).
Die Standardfunktion "MemAvail" kann jederzeit benutzt werden, um den auf dem Heap verfügbaren Platz zu bestimmen. Für weitere Hinweise siehe Kapitel 20, 21 und 22.

15.4 Die Benutzung von Zeigern

Angenommen, wir haben die Prozedur New benutzt, um eine Serie von Records des Typs PersonRecord zu schaffen (wie im Beispiel auf der folgenden Seite), und daß das Feld Next in jedem Record auf das nächste PersonRecord deutet, dann gehen die folgenden Anweisungen die Liste durch und geben den Inhalt jedes Records aus (FirstPerson zeigt auf die erste Person in der Liste):

```
while FirstPerson <> nil do
  with FirstPerson^ do
```

```
    begin
      Writeln (Name, 'is a ',Job);
      FirstPerson := Next;
    end;
```

FirstPerson^.Name kann als FirstPerson's.Name gelesen werden, d.h. als das Feld Name, auf das im Record mit FirstPerson gezeigt wird.

Folgende Beispiele demonstrieren den Gebrauch von Zeigern, um eine Liste von Namen und gewünschten Berufen zu erstellen. Die Namen und gewünschten Berufe werden solange gelesen, bis ein Leerzeichen eingegeben wird. Danach wird die Liste ausgedruckt. Anschließend ist der benutzte Speicherplatz wieder frei. Die Zeigervariable HeapTop wird nur zur Aufnahme und zum Speichern des Anfangswertes gebraucht. Ihre Definition als ^Integer (Zeiger auf Integer) ist deshalb rein willkürlich.

```
procedure Jobs;
type
  PersonZeiger = ^PersonRecord;

  PersonRecord = record
                       Name: string[50];
                       Job: string[50];
                       Next: PersonZeiger;
                 end;
Var
  HeapTop: ^Integer;
  FirstPerson, LastPerson, NewPerson:PersonZeiger;
  Name: string[50];
begin
  FirstPerson := nil;
  Mark (HeapTop);
  repeat
    Write ('Enter name:    ');
    Readln (Name);
    if Name <> '' then
    begin
      New (NewPerson);
      NewPerson^.Name := Name;
      Write ('Enter profession:');
      Readln (NewPerson^.Job);
      Writeln;
      if FirstPerson = nil then
         FirstPerson = NewPerson
      else
         LastPerson^.Next := NewPerson;
         LastPerson := NewPerson;
         LastPerson^.Next := nil;
    end;
  until Name =';
  Writeln;
  while FirstPerson <> nil do
  with FirstPerson^ do
  begin
    Writeln (Name,' is a ',Job;
    FirstPerson := Next;
  end;
  Release (HeapTop);
end.
```

15.5 Dispose

Anstelle von Mark/Release kann die Dispose-Prozedur von Standard-Pascal benutzt werden, um Speicherplatz im Heap zurückzugewinnen.

Beachten Sie, daß Dispose und Release verschiedene Arten der Heapverwaltung verwenden und diese nie zugleich benutzt werden dürfen. Ein Programm kann entweder Dispose oder Mark/Release verwenden, um den Heap zu verwalten. Diese zu mischen, verursacht unvorhersagbare Ergebnisse.

Die Syntax ist: Dispose(Var), wobei Var eine Zeigervariable ist.

Dispose erlaubt es, einen dynamischen Speicher, der von einer spezifischen Zeigervariablen genutzt wird, für eine neuerliche Verwendung zurückzugewinnen. Im Gegensatz dazu setzen Mark und Release den ganzen Heap von der spezifischen Zeigervariablen an abwärts frei.

Nehmen wir an, Sie haben eine Reihe von Variablen, die dem Heap zugewiesen wurden. Das folgende Bild zeigt den Inhalt des Heap und die Wirkung von Dispose (Var) und Mark (Var3) Release (Var3):

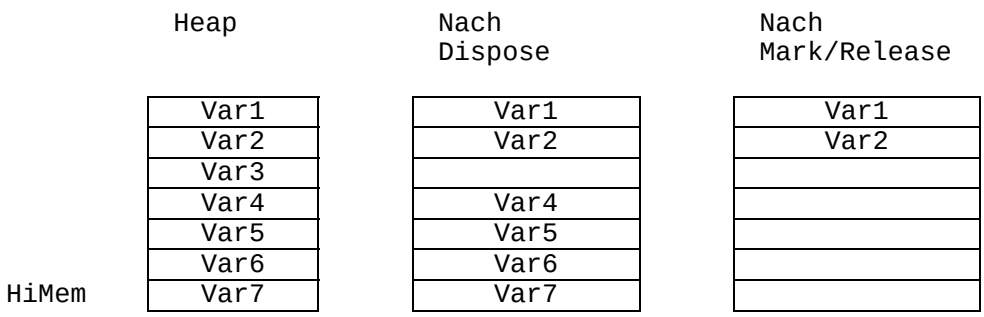

Bild 15-1: Gebrauch von Dispose

Nach der Anwendung von Dispose auf eine Zeigervariable kann der Heap aus einer Reihe von benutzten Speicherelementen und dazwischenliegenden freien Speicherbereichen bestehen. Darauffolgende Aufrufe von New verwenden diese, wenn die neue Zeigervariable an die Stelle paßt.

15.6 GetMem

Die Standardprozedur "GetMem" wird benutzt, um auf dem Heap einen bestimmten Platzbedarf zu reservieren. Im Gegensatz zu New, wo soviel Platz zugewiesen wird, wie es der Typ benötigt, auf dessen Argument gezeigt wird, erlaubt GetMem dem Programmierer, die Größe des zugewiesenen Platzes zu kontrollieren. GetMem wird mit zwei Parametern aufgerufen:

```
GetMem (PVar,I)
```

PVar ist eine beliebige Zeigervariable, und I ist ein Integer-Ausdruck, der die Anzahl der Bytes angibt, für die Platz benötigt wird.

15.7 FreeMem

Syntax: FreeMem;

Die FreeMem-Standardprozedur wird gebraucht, um einen ganzen Block auf dem Heap wieder freizumachen. Es ist also das Gegenstück zu GetMem. FreeMem wird mit zwei Parametern aufgerufen:

```
FreeMem (PVar,I);
```

wobei PVar eine beliebige Zeigervariable ist und I ein Integer-Ausdruck, der die Zahl der Bytes angibt, die wieder freizumachen sind. Diese Zahl muß exakt der Zahl von Bytes entsprechen, die vorher durch GetMem dieser Variablen zugewiesen worden sind.

15.8 MaxAvail

Die MaxAvail-Standardfunktion gibt die Größe des größten zusammenhängenden freien Platzes an, die auf dem Heap besteht. Bei 16-Bit-Systemen steht dieser Platz in Paragraphen (pro 16 Bytes): bei 8-Bit-Systemen in Bytes. Das Ergebnis ist eine ganze Zahl, und wenn mehr als 32767 Paragraphen Bytes verfügbar sind, gibt MaxAvail eine negative Zahl aus. Die korrekte Zahl freier Paragraphen/Bytes wird dann durch 65536.0 + MaxAvail berechnet. Beachten Sie, daß reelle Konstanten verwendet werden müssen, um ein reelles Ergebnis zu erhalten, falls das Ergebnis größer als MaxInt ist.

16. Prozeduren und Funktionen

Ein Pascalprogramm besteht aus einem oder mehreren Blöcken, die wieder in Blöcke unterteilt sein können usw. Ein solcher Block ist eine Prozedur, ein anderer eine Funktion (gemeinhin Unterprogramm genannt). Eine Prozedur ist also ein eigenständiger Teil im Programm und kann mittels einer Prozeduranweisung aufgerufen werden (siehe Seite 56). Eine Funktion ist dem ziemlich ähnlich, aber sie berechnet einen Wert, wenn ihr Bezeichner während der Ausführung erreicht wird (siehe Seite 54) und gibt diesen dann aus.

16.1 Parameter

Werte können in Prozeduren und Funktionen durch Parameter übergeben werden. Dies erlaubt, ein Unterprogramm mit verschiedenen Werten zu fahren und damit auch unterschiedliche Ergebnisse zu bekommen.

Die Prozeduranweisung oder die Funktionsbezeichnung, die das Unterprogramm aufruft, kann eine Liste von Parametern enthalten, die aktuellen Parameter. Diese werden an die formalen Parameter, die im Kopf des Unterprogramms bestimmt sind, übergeben. Die Reihenfolge der Übergabe entspricht der Reihenfolge der Parameterliste. Pascal unterstützt zwei verschiedene Methoden der Parameterübergabe: über den Wert und über die Referenz, einer Veränderung der formalen Parameter. Dabei ist die Wirkung auf die aktuellen Parameter jeweils unterschiedlich.

Wenn Parameter über den Wert übergeben werden, entspricht der formale Parameter einer logischen Variablen im Unterprogramm, und Veränderungen der formalen Parameter haben keine Auswirkung auf aktuelle Parameter. Der aktuelle Parameter kann jeder beliebige Ausdruck sein, der vom selben Typ ist wie der entsprechende formale Parameter, einschließlich einer Variablen. Solche Parameter heißen Wertparameter und werden wie im folgenden Beispiel im Unterprogramm deklariert. Dieses und das nächste Beispiel zeigen Prozedurüberschriften: Funktionsüberschriften unterscheiden sich etwas davon und sind auf Seite 137 beschrieben.

```
procedure Example (Num1, Num2: Number; Str1, Str2:Txt);
```

Number und Txt sind vorher definierte Typen (z.B. integer oder string[255]), und Num1, Num2, Str1 und Str2 sind formale Parameter, an die der Wert der aktuellen Parameter übergeben wird. Die Typen von formalen und aktuellen Parametern müssen übereinstimmen.

Beachten Sie, daß der Typ des Parameters im Parameterteil so wie ein vorher definierter Typenbezeichner angegeben werden muß. Deshalb ist die Angabe

```
procedure Select(Model: array[1..500] of integer);
```

nicht erlaubt. Stattdessen sollte der gewünschte Typ in der type-Definition des Blocks definiert werden, und der Typenbezeichner sollte dann in der Parametererklärung benutzt werden.

```
type
  Range = array[1..500] of integer;

procedure Select(Model: Range);
```

Wenn ein Parameter durch Bezugnahme übergeben wird, entspricht der formale Parameter tatsächlich während der Ausführung des Unterprogramms dem aktuellen Parameter. Jede Veränderung des formalen Parameters gilt folglich auch für den aktuellen Parameter, der deshalb eine Variable sein muß. Parameter, die durch Bezugnahme übergeben werden, werden Variablenparameter genannt und wie folgt deklariert:

```
procedure Example(Var Num1,Num2: Number);
```

Wertparameter und Variablenparameter können in derselben Prozedur gemischt werden, entsprechend folgende Beispiel:

```
procedure Example(Var Num1,Num2: Number; Str1,Str2: Txt);
```

in dem Num1 und Num2 Variablenparameter sind und Str1 und Str2 Wertparameter.

Alle Adreßberechnungen werden zum Zeitpunkt des Prozeduraufrufs durchgeführt. Wenn eine Variable eine Komponente eines Arrays ist, werden deshalb ihre Indices überprüft, wenn das Unterprogramm aufgerufen ist.

Beachten Sie, daß Dateiparameter immer als Variablenparameter deklariert werden müssen.

Wenn eine große Datenstruktur, wie etwa ein Array an ein Unterprogramm als ein Parameter übergeben werden soll, spart die Benutzung eines Variablenparameters Zeit und Speicherplatz, da dann nur die Adresse des aktuellen Parameters an das Unterprogramm übergeben wird. Ein Wertparameter würde Speicherplatz und Zeit für eine zusätzliche Kopie der ganzen Datenstruktur benötigen.

16.1.1 Lockerung der Parametertyp-Überprüfung

Im Normalfall müssen bei der Benutzung von Variablenparametern die formalen und aktuellen Parameter exakt übereinstimmen. Unterprogramme, die Variablenparameter des Typs String verwenden, laufen nur mit Strings von genau der Länge, wie sie im Unterprogramm definiert ist. Diese Einschränkung kann durch den V-Compilerbefehl aufgehoben werden. Der voreingestellte aktive Status {$V-} die Typenprüfung lockert und es erlaubt, aktuelle Parameter jeglicher Stringlänge zu übergeben, ungeachtet der Länge der formalen Parameter.

```
Beispiel:
program Encoder;
{$V-}
type
  WorkString = string[255];
Var
  Line1: string[80];
  Line2: string[100];
procedure Encode(Var LineToEncode: WorkString);
Var I: integer;
```

```
begin
  for I:=1 to Length(LineToEncode) do
    LineToEncode[I]:=Chr(Ord(LineToEncode[I])-30);
end;
begin
  Line1:='This is a secret message';
  Encode(Line1);
  Line2:='Here is another (longer) secret message';
  Encode(Line2);
end.
```

16.1.2 Nicht-typisierte Variablenparameter

Wenn der Typ des formalen Parameters nicht definiert ist, d.h., die Typendefinition in dem Parameterteil des Unterprogrammkopfes nicht aufgelistet ist, dann wird dieser Parameter nichttypisiert genannt. Deshalb kann der entsprechende aktuelle Parameter beliebigen Typs sein.

Der untypisierte, formale Parameter selbst ist nicht kompatibel mit den anderen Typen, und er kann deshalb nur dann benutzt werden, wenn der Datentyp keine Rolle spielt, z.B. als Parameter zu Addr, BlockRead/Write, FillChar oder Move, oder als die Adressenspezifikation einer absoluten Variablen.

Die SwitchVar-Prozedur im folgenden Beispiel demonstriert den Gebrauch von nichttypisierten Parametern. Es wird der Inhalt von A1 nach A2 und der Inhalt von A2 nach A1 bewegt.

```
procedure SwitchVar(Var A1p,A2p,Size:Integer);

type
  A = array 1..MaxInt of Byte;
Var
  A1:A absolute A1p;
  A2:A absolute A2p;
  Tmp:Byte;
  Count:Integer;
begin
  for Count = 1 to Size do
  begin
    Tmp:= A1[Count];
    A1[Count]:= A2[Count];
    A2[Count]:= Tmp;
  end;
end.
```

Angenommen die Angaben lauten:

```
type
  Matrix = array[1..50,1..25] of Real;
Var
  TestMatrix,BestMatrix:Matrix;
```

dann kann man SwitchVar verwenden, um die Werte zwischen den beiden Matrizen zu vertauschen:

```
SwitchVar(TestMatrix,BestMatrix,SizeOf(Matrix));
```

16.2 Prozeduren

Eine Prozedur kann entweder vordeklariert ('oder standardisiert') oder vom Programmierer deklariert sein. Vordeklarierte Prozeduren sind Teile des KCPASCAL-Systems und können weitere Angaben aufgerufen werden. Eine vom Benutzer festgelegte Prozedur kann den Namen einer Standardprozedur tragen; aber dann wird diese Standardprozedur unbrauchbar innerhalb des Bereichs der vom Benutzer festgelegten Prozedur.

16.2.1 Prozedurdeklarierung

Die Prozedurdeklarierung besteht aus einem Prozedurkopf, gefolgt von einem Block, der aus einem Deklarierungsteil und einem Anweisungsteil besteht.

Der Prozedurkopf besteht aus dem reservierten Wort "procedure" gefolgt von einem Bezeichner, für den Namen der Prozedur. Wahlweise gefolgt von einer formalen Parameterliste, wie auf Seite 127 beschrieben.

```
Beispiele:
procedure LogOn;
procedure Position(X,Y:Integer);
procedure Compute(Var Data:Matrix,Scale:Real);
```

Der Deklarierungsteil einer Prozedur hat die gleiche Form wie der eines Programms. Alle Bezeichner, die in der formalen Parameterliste und dem Deklarierungsteil deklariert sind, beziehen auf die jeweilige Prozedur und die darin eingebundenen Prozeduren. Außerhalb dieses Bezugsrahmens ist der Bezeichner nicht bekannt. Eine Prozedur kann sich auf jede Konstante, Variable, Prozedur oder Funktion beziehen, die in einem anderen Block steht.

Der Anweisungsteil spezifiziert die auszuführende Aktion, wenn die Prozedur aufgerufen wird und hat die Form einer gesamten Befehlszeile (siehe Seite 57). Wenn der Prozedurbezeichner innerhalb des Anwendungsteils selbst benutzt wird, wird die Prozedur rekursiv ausgeführt (nur CP/M-80-Benutzer). Beachten Sie, daß der A-Compilerbefehl [$A]- passiv sein muß. Rekursion siehe auch Anhang C.

Das nächste Beispiel ist ein Programm, daß eine Prozedur benutzt und einen Parameter an die Prozedur übergibt. Wenn der aktuelle Parameter, der an die Prozedur übergeben wird, in manchen Fällen eine Konstante ist (ein einfacher Ausdruck), muß der formale Parameter ein Wertparameter sein.

```
program Box;
Var
  I:Integer;
procedure DrawBox (X1,Y1,X2,Y2:Integer);
  Var I:Integer;
  begin
    GotoXY(X1,Y1);
    for I:= X1 to X2 do write('-');
    GotoXY(X1,Y1±1);
    for I:= Y1+1 to Y2 do
    begin
      GotoXY(X1,I);Write('!');
      GotoXY(X2,I);Write('!');
    end;
```

```
    GotoXY(X1,Y2);
    for I:= X1 to X2 do Write('-');
  end; {of procedure DrawBox}
begin
  ClrScr;
  for I:= 1 to 5 do DrawBox(I^4,I^2,10^I,4^I);
  DrawBox(1,1,80,25);
end.
```

Oft sollen die Veränderungen bei den formalen Parametern in einer Prozedur auch die aktuellen Parameter betreffen. In solchen Fällen werden Variablenparamater verwendet, wie im folgenden Beispiel:

```
procedure Switch(Var A,B:Integer);
Var Tmp:Integer;
begin
  Tmp:= A;A:=B;B:=Tmp;
end;
```

Wenn diese Prozedur durch die Anweisung

```
Switch(I,J);
```

aufgerufen wird, werden die Werte von I und J vertauscht. Wenn der Prozedurkopf in Switch wie folgt deklariert war:

```
procedure Switch(A,B:Integer);
```

d.h. mit einem Wertparameter, dann würde die Anweisung Switch (I,J) I und J vertauschen.

16.2.2 Standardprozeduren

KCPASCAL kennt eine Reihe von Standardprozeduren. Diese sind:

1) String-Handhabungsprozeduren
2) Datei-Handhabungsprozeduren
3) Prozeduren für die Zuweisung von dynamischen Variablen
4) Eingabe- und Ausgabeprozeduren

16.2.2.1 ClrScr

Syntax: ClrScr

Löscht den Bildschirm und plaziert den Cursor in die linke obere Ecke. Beachten Sie, daß manche Bildschirme auch die Videoeigenschaften zurücksetzen, wenn der Schirm gelöscht wird, was möglicherweise vom Benutzer gesetzte Eigenschaften verändern kann.

16.2.2.2 Delay (Verzögerung)

Syntax: Delay (Time)

Die Prozedur "Delay" erzeugt eine Schleife, die ungefähr soviele Millisekunden läuft, wie im Argument Time, das eine ganze Zahl sein muß, angegeben ist. Die exakte Zeit kann in unterschiedlichen Betriebs-umgebungen verschieden sein.

16.2.2.3 GotoXY

Syntax: GotoXY(Xpos,Ypos)

Bewegt den Cursor auf die Position, die durch die ganzzahligen Ausdrücke Xpos (vertikaler Wert oder Spalteneinteilung) und Ypos (horizontaler Wert oder Zeileneinteilung) angegeben werden. Die obere linke Ecke ist die Position (1,1) (home position).

16.2.2.4 Exit

Syntax: Exit

Verläßt den gegenwärtigen Block. Wenn Exit in einem Unterprogramm ausgeführt wird, bewirkt dies das Verlassen des Unterprogramms. Wenn Exit in dem Anweisungsteil ausgeführt wird, verursacht es den Abbruch des Programms. Ein Aufruf von Exit kann mit einer goto-Anweisung verglichen werden, die auf eine Adresse, kurz vor dem Ende (end) des Blocks zeigt.

16.2.2.5 Halt

Syntax: Halt

Beendet die Programmausführung und führt zum Betriebssystem zurück.

16.2.2.6 Randomize

Syntax: Randomize

Startet den Zufallsgenerator mit einer Zufallszahl.

16.2.2.7 Move

Syntax: Move(Var1,Var2,Num)

Führt direkt im Speicher eine Kopie einer Anzahl Bytes aus. Var1 und Var2 sind zwei Variablen beliebigen Typs, Num ist en ganzzahliger Ausdruck. Die Prozedur kopiert einen Block von Num Bytes, beginnend beim ersten Byte, von Var1 auf das erste Byte von Var2. Es gibt keine 'moveright'- und 'moveleft'-Prozeduren, da Move automatisch mögliche Überlappungen während des Move-Prozesses handhabt.

16.2.2.8 FillChar

Syntax: FillChar(Var,Num,Value)

Füllt einen Speicherbereich mit einem gegebenen Wert. Var ist eine Variable beliebigen Typs, Num ist ein ganzzahliger Ausdruck und Value ein Ausdruck vom Typ Byte oder Char. Num Bytes, beginnend beim ersten durch Var belegten Byte, werden mit dem Wert Value aufgefüllt.

16.3 Funktionen

Wie Prozeduren sind Funktionen entweder standardisiert (vordeklariert) oder vom Programmierer deklariert.

16.3.1 Funktionsdeklarierung

Eine Funktionsdeklarierung besteht aus einem Kopf und einem Block, der aus einem Deklarationsteil, gefolgt von einem Anweisungsteil, besteht.

Der Funktionskopf entspricht dem Prozedurkopf, außer daß im Kopf der Typ des Ergebnisses der Funktion definiert sein muß. Dies geschieht, indem ein Doppelpunkt und ein Typ hinzugefügt wird:

```
function KeyHit: Boolean;
function Compute(Var Value: Sample) : Real;
function Power(X,Y): Real) : Real;
```

Der Ergebnistyp einer Funktion muß skalar (z.B. Integer, Real, Boolean, Char, deklarierter, skalarer Typ oder Teilbereich sein), vom Typ String oder ein Zeigertyp sein.

Der Deklarierungsteil einer Funktion ist identisch mit dem einer Prozedur.

Der Anwendungsteil einer Funktion ist eine zusammengesetzte Anweisung, wie auf Seite 57 beschrieben. Innerhalb des Anweisungsteiles muß mindestens eine Anweisung enthalten sein, die dem Funktionsbezeichner einen Wert zuweist. Die zuletzt ausgeführte Zuweisung bestimmt das Ergebnis der Funktion. Wenn die Funktionsbezeichnung innerhalb des Anweisungsteils selbst steht, wird die Funktion rekursiv aufgerufen (Nur für CP/M-80-Benutzer: Beachten Sie, daß der A-Compilerbefehl passiv sein muß {$A-}. Zur Rekursion siehe Anhang C).

Das folgende Beispiel zeigt den Gebrauch einer Funktion zur Berechnung der Summe einer Zeile aus ganzen Zahlen von I bis J.

```
function RowSum(I:J:Integer):Integer;
   function SimpleRowSum(S:Integer):Integer;
   begin
     SimpleRowSum:= S*(S+1)div 2;
   end;
begin
  RowSum:= SimpleRowSum(J)-SimpleRowSum(I-1);
end;
```

Die Funktion SimpleRowSum ist in die Funktion RowSum eingebettet. SimpleRowSum ist daher nur innerhalb des Bereichs von RowSum verfügbar.

Das folgende Programm ist das klassische Demonstrationsbeispiel für den Gebrauch einer rekursiven Funktion zur Berechnung des Faktors einer ganzen Zahl:

```
{$A-}
program Factorial;
Var Number:Integer;
function Factorial(Value:Integer):Real;
begin
  if Value = 0 then Factorial := 1
  else Factorial := Value*Factorial(Value-1);
end;
begin
  Read(Number);
  Writeln(Number,'!=',Factorial(Number));
end;
```

Beachten Sie, daß der Typ, der in der Definition des Funktionstyps benutzt wird, zuvor als Typbezeichner spezifiziert sein muß. Deshalb ist

```
function LowCase(Line:UserLine):string[80];
```

nicht zulässig. Stattdessen sollte ein Typenbezeichner mit dem Typ string80 verbunden sein, und dann dazu benutzt werden, den Ergebnistyp der Funktion zu definieren, z.B.:

```
type
  Str80 = string[80];

function LowCase(Line:UserLine):Str80;
```

Wegen der Implementation der Standardprozeduren Write und Writeln darf eine Funktion, die die Standardprozeduren Read, Readln, Write und Writeln benutzt, nie durch einen Ausdruck innerhalb einer Write- oder Writeln-Anweisung aufgerufen werden. Dies gilt auch für die Standardprozeduren Str und Val.

16.3.2 Standardfunktionen

Die folgenden Standardfunktionen sind in KCPASCAL implementiert:

1) String-Behandlungsfunktionen (beschrieben auf Seite 71 ff)
2) Datei-Handhabungsfunktionen (beschrieben auf Seite 94 und 101)
3) Zeigerfunktionen (beschrieben auf Seite 120 und 125)

16.3.2.1 Arithmetische Funktionen

16.3.2.1.1 Abs

Syntax: Abs(Num)

Gibt den absoluten Wert von Num aus. Das Argument muß entweder reell oder integer sein und das Ergebnis ist vom selben Typ wie das Argument.

16.3.2.1.2 ArcTan

Syntax: ArcTan(Num)

Gibt den Winkel, dessen Tangente Num ist, im Bogenmaß an. Das Argument X muß entweder real oder integer sein, das Ergebnis ist real.

16.3.2.1.3 Cos

Syntax: Cos(Num)

Gibt den Cosinus von Num aus. Das Ergebnis wird im Bogenmaß ausgedrückt, und es muß entweder integer oder real sein. Das Ergebnis ist real.

16.3.2.1.4 Exp

Syntax: Exp(Num)

Gibt den Exponenten von Num, d.h. e25num26 aus. Das Argument Num muß entweder integer oder real sein, das Ergebnis ist real.

16.3.2.1.5 Frac

Syntax: Frac(Num)

Gibt den Bruchteil von Num aus, d.h. Frac(Num) = Num-Int(Num). Das Argument muß entweder integer oder real sein, das Ergebnis ist real.

16.3.2.1.6 Int

Syntax: Int(Num)

Gibt den ganzzahligen Teil von Num an und zwar die größte ganze Zahl kleiner gleich Null, falls Num>=0 oder die kleinste ganze Zahl größer gleich Num, wenn Num<0. Das Argument Num muß entweder integer oder real sein, das Ergebnis ist real.

16.3.2.1.7 Ln

Syntax: Ln(Num)

Gibt den natürlichen Logarithmus von Num aus. Das Argument Num muß entweder integer oder real sein, das Ergebnis ist real.

16.3.2.1.8 Sin

Syntax. Sin(Num)

Gibt den Sinus von Num aus Das Argument wird im Bogenmaß ausgedrückt, es muß entweder integer oder real sein, das Ergebnis ist real.

16.3.2.1.9 Sqr

Syntax: Sqr(Num)

Gibt das Quadrat von Num (Num^Num) aus. Das Argument Num muß entweder integer oder real sein, das Ergebnis ist vom selben Typ wie das Argument.

16.3.2.1.10 Sqrt

Syntax: Sqrt(Num)

Gibt die Wurzel von Num aus. Das Argument Num muß entweder integer oder real sein.

16.3.2.2 Skalar-Funktionen

16.3.2.2.1 Pred

Syntax: Pred(Num)

Gibt den Vorgänger von Num aus (falls dieser existiert). Num ist ein beliebiger skalarer Typ.

16.3.2.2.2 Succ

Syntax: Succ(Num)

Gibt den Nachfolger von Num aus (falls dieser existiert). Num ist ein beliebiger skalarer Typ.

16.3.2.2.3 Odd

Syntax: Odd(Num)

Gibt den Bool'schen Wahrheitswert True an, wenn Num eine ungerade Zahl ist und False, wenn Num eine gerade Zahl ist. Num muß integer sein.

16.3.2.3 Transfer-Funktionen

Die Transfer-Funktionen werden gebraucht, um die Werte eines skalaren Typen in die eines anderen umzurechnen. Zusätzlich zu den folgenden Funktionen dient auch die auf Seite 65 beschriebene retype Möglichkeit zu diesem Zweck.

16.3.2.3.1 Chr

Syntax: Chr(Num)

Gibt das Zeichen mit dem ordinalen Wert, der durch den ganzzahligen Ausdruck Num gegeben ist, an. Beispiel: Chr(65) gibt das Zeichen "A" aus.

16.3.2.3.2 Ord

Syntax: Ord(Var)

Gibt die ordinal Zahl des Wertes von Var in der durch den Typ Var definierten Menge an. Ord(Var) ist gleichbedeutend mit Integer(Var) (siehe Seite 56). Var kann beliebigen skalaren Typs sein, außer real, das Ergebnis ist integer.

16.3.2.3.3 Round

Syntax: Round(Num)

Rundet den Wert von Num wie folgt:

Wenn Num >=0, dann ist Round(Num) = Trunc(Num+0.5) und
wenn Num <=0, dann ist Round(Num) = Trunc(Num-0.5)

Num muß real sein, das Ergebnis ist integer.

16.3.2.3.4 Trunc

Syntax: Trunc(Num)

Gibt die größte ganze Zahl kleiner gleich Num an, falls Num >=0 oder die kleinste ganze Zahl größer gleich Num, falls Num < 0. Num muß real sein und das Ergebnis ist integer.

16.3.2.4 Verschiedenartige Standardfunktionen

16.3.2.4.1 Hi

Syntax: Hi(I)

Das niederwertigste Byte des Ergebnisses enthält das höherwertigere Byte des Wertes des ganzzahligen Ausdrucks I. Das höherwertige Byte des Ergebnisses ist Null. Das Ergebnis ist integer.

16.3.2.4.2 KeyPressed

Syntax: KeyPressed

Gibt den Bool'schen Wahrheitswert True aus, falls eine Taste gedrückt wurde. Das Ergebnis erhält man, indem man über das Betriebssystem den Status der Konsole abfragt.

16.3.2.4.3 Lo

Syntax: Lo(I)

Gibt das niederwertigere Byte des Wertes des ganzzahligen Ausdrucks I, mit dem auf Null gesetzten höherwertigen Byte aus. Das Ergebnis ist integer.

16.3.2.4.4 Random

Gibt eine Zufallszahl größer oder gleich Null und kleiner Eins aus. Der Typ ist real.

16.3.2.4.5 Random(Num)

Syntax: Random(Num)

Gibt eine Zufallszahl größer oder gleich Null und kleiner als Num aus. Num und die Zufallszahl sind beide integer.

16.3.2.4.6 SizeOf

Syntax: SizeOf(Name)

Gibt die Zahl der durch die Variable oder den Typ Name im Speicher belegten Bytes aus. Das Ergebnis ist integer.

16.3.2.4.7 Swap

Syntax Swap(Num)

Die Swap-Funktion (Austauschfunktion) tauscht die nieder- und höherwertigen Bytes des ganzzahligen Arguments Num aus und gibt den Ergebniswert als ganze Zahl aus.

Beispiel:
Swap($1234) ergibt $3412 (Werte im Hex-Code)

16.3.2.4.8 UpCase

Syntax: UpCase(ch)

Gibt das großgeschriebene Äquivalent des Arguments ch an, das vom Typ Char sein muß. Falls kein großgeschriebenes, äquivalentes Zeichen existiert, wird das Argument unverändert ausgegeben.

16.4 Forward-Referenzen

Ein Unterprogramm ist forward deklariert, indem sein Kopf getrennt vom Block spezifiziert ist. Dieser separate Unterprogrammkopf ist exakt wie der normale Kopf, er wird nur mit dem reservierten Wort "forward" abgeschlossen. Der Block folgt später innerhalb desselben Deklarierungsteils. Beachten Sie, daß der Block von einer Kopie des Kopfes eingeleitet wird, die nur den Namen und keine Parameter, Typen usw. spezifiziert.

Beispiel:
```
program catch22;
Var
  X:Integer;
function Up(Var I:Integer):Integer;forward;
function Down(Var I:Integer):Integer;
begin
  I:=I div 2;Writeln(I);
  if I<>1 then I:=Up(I);
end;
function Up;
begin
  while I mod 2 <> 0 do
  begin
    I:=I*3+1;Writeln(I);
  end;
  I:=Down(I);
end;
begin
  Write('Enter any integer:)'
  Readln(X);
  X:=Up(X);
  Write('Ok. Program stopped again');
end.
```

Wenn dieses Programm ausgeführt wird, und man z.B. 6 eingibt, erhält man als Ergebnis:

```
3
10
5
16
8
4
2
1
Ok. Program stopped again.
```

```
program Catch222;
Var
  X:Integer;
begin
  Write('Enter any integer:');
  Readln(X);
  while X <> 1 do
  begin
    if X mod 2 = then X := X div 2 else X:= X*3-1;
    Writeln(X);
  end;
  Write('Ok. Program stopped again');
end.
```

Vielleicht interessiert es Sie, daß dieses kleine und sehr einfache Programm nicht darauf geprüft werden kann, ob es wirklich für jede ganze Zahl stoppt!

17. Weitere Besonderheiten

Dieser Anhang beschreibt zusätzliche Eigenschaften von KCPASCAL.

1) Für den effizienten Gebrauch von KCPASCAL unbedingt nötige Informationen.

2) Die restlichen Abschnitte beschreiben Einzelheiten, die nur für erfahrene Programmierer interessant sind, z.B. den Aufruf von Assemblerroutinen, technische Aspekte des Compilers usw.

17.1 Compiler-Optionen

Das O-Kommando wählt das folgende Menü an, in dem Sie einige voreingestellte Werte des Compilers sehen und verändern können. Es ist auch beim Finden von Laufzeitfehlern in Programmen die in Objketcode-Dateien compiliert sind, hilfreich.

```
compile     >     Memory
                  Com-File

command line Parameter

Find run-time error     Quit
```

Bild 17-1: Optionen - Menü

17.1.1 Memory/Com-Datei

Die drei Befehle M, C und H steuern die Verarbeitungsart der Quelle und die Abfrage des erzeugten Objekt-Codes durch den Compiler.

Memory (Arbeitsspeicher) ist der voreingestellte Modus. Der Code wird im Speicher erzeugt und behalten. Das Programm kann dann direkt vom Speicher aus durch den Run-Befehl ausgeführt werden.

Com-File wird durch Eingabe C gewählt und durch den Pfeil angezeigt. Der Code wird im Falle der Aktivierung auf eine Datei mit demselben Namen wie die Arbeitsdatei (oder Hauptdatei, falls angegeben) als .COM-File geschrieben. Diese Datei enthält den Objekt-Code und die Pascal 'runtime library'. Programme, die auf diese Weise compiliert werden, können größer sein als im Speicher compilierte Programme, da der Objekt-Code selbst keinen Speicherplatz während der Compilierung braucht und bei einer niedrigen Adresse beginnt.

Wenn der Com-Modus gewählt wird, erweitert sich das Menü um folgende zwei Zeilen:

```
Start adress:    XXXX  (min YYYY)
End address:     XXXX  (max YYYY)
```

Bild 17-2: Start- und Endadressen

17.1.2 Start-Adresse

"Start address" gibt die Adresse (hexadezimal) des ersten Bytes des Codes an. Das ist normalerweise die Endadresse der Pascal-Library plus eins, kann aber auch auf eine höhere Adresse geändert werden, falls man Platz reservieren will, z.B. für absolute Variablen, die eine Reihe von verketteten Programmen gemeinsam sind.

Wenn Sie ein "S" eingeben, werden Sie veranlaßt, eine neue Startadresse einzugeben. Falls Sie nur <ENTER> drücken, wird der kleinste Wert angenommen.

Setzen Sie die Startadresse nicht niedriger als den minimalen Wert, da der Code dann Teile der Pascal-Library überschreibt.

17.1.3 End-Adresse

"End address" gibt die höchste für das Programm verfügbare Adresse an (hexadezimal). Der Wert in Klammern zeigt die Spitze der TPA auf ihrem Computer.
Wenn Sie "E" eingeben, werden Sie aufgefordert, eine Endadresse einzugeben. Wenn Sie <ENTER> drücken, wird der voreingestellte Wert übernommen.

17.1.4 Finden von Laufzeitfehlern

Wenn Sie ein im Speicher compiliertes Programm laufen lassen, und es tritt ein Laufzeitfehler auf, wird der Editor aufgerufen und der Fehler automatisch angezeigt. Dies ist natürlich nicht möglich, wenn das Programm in einer .COM- oder .CHN-Datei steht. Laufzeitfehlermeldungen zeigen den Fehlercode und den Wert des Programmzählers zur Zeit des Fehlers an, z.B.:

```
Run-time error  0.1PC = 1B56
Program aborted
```

Bild 17-3: Laufzeit-Fehlermeldung

Um die Stelle in der Quelle zu finden, an der der Fehler auftrat, müssen Sie den F-Befehl im Optionsmenü eingeben. Wenn die Bereitschaftsmeldung für die Adresse da ist, geben Sie die von der Fehlermeldung angegebene Adresse ein.

```
Enter  PC:  1B56
```

Bild 17-4: Finden eines Laufzeitfehlers

Die Stelle in der Quelle wird jetzt gefunden und genauso ausgegeben, als ob der Fehler während eines Programmablaufs im Speicher aufgetreten wäre.

17.2 Standardbezeichner

Es gibt folgende Standardbezeichner:

Bios	Bdos	RecurPtr
BiosH1	BdosH1	StackPtr

17.3 Absolute Variablen

Variablen können deklariert werden, so daß sie an bestimmten Speicheradressen stehen. Sie heißen dann absolute Variablen. Dies geschieht, indem man bei der Variablendeklarierung das reservierte Wort "absolute" hinzufügt und eine Adresse als ganzzahlige Konstante angibt. "Absolute" kann auch genutzt werden, um eine Variable an die Spitze einer anderen Variablen zu deklarieren, d.h., daß die Variable an derselben Adresse wie die andere Variable starten soll. Wenn "absolute" vor dem Variablen- (oder Parameter-) Bezeichner steht, startet die neue Variable an der Adresse dieser Variablen (oder dieses Parameters).

Beispiel:
```
Var
  Str: string[32];
  StrLen: Byte absolute Str;
```

Die obige Deklarierung gibt an, daß die Variable StrLen an derselben Adresse starten soll wie die Variable Str, und da das erste Byte einer Stringvariablen die Länge des Strings festlegt, enthält StrLen die Länge von Str. Beachten Sie, daß nur ein Bezeichner in einer absoluten Deklarierung angegeben werden kann, d.h. das Konstrukt

```
Ident1, Ident2: Integer absolute $8000
```

ist unzulässig. Weitere Details über die Platzzuweisung für Variablen finden Sie auf den Seiten 278 und 288.

17.4 Addr-Funktion

Syntax: Addr(Name)

Gibt die Speicheradresse des ersten Bytes des Typen, der Variablen, der Prozedur oder Funktion mit dem Bezeichner "name" aus. Wenn "name" ein Feld (Array) ist, kann er vorgemerkt werden, wenn "name" ein Record ist, können bestimmte Felder ausgewählt werden. Der ausgegebene Wert ist integer.

17.5 Vordefinierte Arrays

KCPASCAL bietet zwei vordefinierte Arrays vom Typ Byte, Mem und Port, die als direkter Zugang zum CPU-Speicher und zu den Datenports benutzt werden können.

17.5.1 Mem-Array

Das vordefinierte Array "Mem" wird benutzt, um auf Speicher zuzugreifen. Jede Komponente des Arrays ist ein Byte. Die Indizes entsprechen den Adressen im Speicher. Der Index ist integer. Wenn ein Wert einer Komponente von Mem zugewiesen wird, wird er an der durch den Indexausdruck gegebenen Adresse gespeichert. Wenn das Feld Mem in einem Ausdruck benutzt wird, wird das Byte der im Index angegebenen Adresse verwendet.

```
Beispiel:
Mem [WsCursor] := 2;
Mem [WsCursor+1] := $1B;
Mem [WsCursor+2] := Ord(m);
IOByte := Mem [3];
Mem [Addr+Offset] := Mem[Addr];
```

17.5.2 Port Array

Das Port-Array wird benutzt, um die Datenports der CPU anzusprechen. Jedes Element des Arrays repräsentiert einen Daten-Port, deren Indizes den Port-Nummern entsprechen. Falls Daten-Ports durch 8-Bit-Adresse angewählt werden, ist der Index vom Typ Byte. Wenn ein Wert einer Komponente von Port zugewiesen ist, wird er an den spezifischen Port ausgegeben. Wenn auf eine Komponente von Port in einem Ausdruck Bezug genommen wird, wird ihr Wert von dem spezifischen Port eingegeben.

Der Gebrauch des Port-Arrays ist beschränkt auf Zuweisung und Bezugnahme in Ausdrücken, d.h., Komponenten von Port können nicht als Variablenparameter für Prozeduren und Funktionen dienen. Weiterhin sind Operationen, die auf das ganze Port-Array Bezug nehmen, nicht erlaubt (Bezugnahme ohne Index).

17.6 Array-Subscript Optimierung

Der X-Compilerbefehl erlaubt es dem Programmierer zu wählen, ob die Array-Subscription eher hinsichtlich der Ausführungszeit oder der Codegröße optimiert wird. Der voreingestellte Modus ist aktiv, d.h. (*$X+*), was Optimierung der Ausführungszeit zur Folge hat. Wenn der passive Modus gewählt wird, d.h. (*$X-*), wird die Codegröße minimiert.

17.7 With-Anweisungen

Die voreingestellte Tiefe der Schachtelung von With-Anweisungen ist 2. Der W-Befehl kann verwendet werden, diesen Wert zwischen 1 und 9 zu verändern. Für jeden Block benötigen With-Anweisungen zwei Bytes Speicher pro Schachtelungsniveau. Der möglichst sparsame Gebrauch der Schachtelung beeinflußt stark die Größe des Datenbereichs, in Programmen mit vielen Unterprogrammen.

17.8 Hinweise zu Zeigern

17.8.1 MemAvail

Die Standardfunktion MemAvail kann immer benutzt werden, um den Platz auf dem Heap zu ermitteln. Das Ergebnis ist eine ganze Zahl. Falls mehr als 32767 Bytes zur Verfügung stehen, gibt MemAvail eine negative Zahl aus. Die korrekte Zahl der freien Bytes wird dann berechnet, indem man zu 65536.0 MemAail addiert. Achten Sie auf die Verwendung reeller Konstanten, um ein reelles Ergebnis zu erzeugen, falls das Ergebnis größer als Maxint ist. Das Speichermanagement wird ausführlicher auf Seite 288 beschrieben.

17.8.2 Zeiger auf ganze Zahlen

Die Standardfunktionen Ord und Ptr erlauben diese Kontrolle über die in einem Zeiger enthaltene Adresse. Ord gibt die in einem Zeigerargument enthaltene Adresse als ganze Zahl aus. Ptr wandelt sein ganzzahliges Argument in einen Zeiger um, der mit allen Zeigertypen kompatibel ist.

17.9 Betriebssystem-Funktionsaufrufe

Um BDOS- und BIOS-Routinen aufzurufen, hat KCPASCAL die zwei Standard-prozeduren Bdos, Bios und die vier Standardfunktionen Bdos, BdosHL, Bios und BiosHL.

Details über diese Routinen finden Sie im Benutzerhandbuch des Betriebs-systems.

17.9.1 Prozedur und Funktion Bdos

Syntax: Bdos (Func,Param)

Die Prozedur Bdos wird verwendet, um BDOS-Routinen aufzurufen. Func und Param sind Integer-Ausdrücke. Func bezeichnet die Nummer der aufgerufenen Routine und wird ins C-Register geladen. Param ist optional und bezeichnet einen Parameter, der in das Registerpaar DE geladen wird. BDOS wird auf Adresse 5 aufgerufen.

Die Funktion Bdos wird wie die Prozedur aufgerufen und gibt einen Integer-Wert aus, der dem Wert entspricht, der durch das BDOS in das A-Register ausgegeben wird.

17.9.2 Die Funktion BdosHL

Syntax: BdosHL (Func,Param)

Diese Funktion entspricht Bdos, die gerade besprochen wurde, jedoch wird der Wert über das Registerpaar HL ausgegeben.

17.9.3 Prozedur und Funktion Bios

Syntax: Bios (Func,Param)

Die Prozedur Bios wird benutzt, um die BIOS-Routinen aufzurufen. Func und Param sind Integer-Ausdrücke. Func bezeichnet die Nummer der aufgerufenen Routine. 0 steht für die Routine WBOOT, 1 für CONST usw. Die Adresse der aufgerufenen Routinen ist Func*3 plus die WBOOT-Adresse, die in den Adressen 1 und 2 enthalten ist. Param ist optional und bezeichnet einen Parameter, der vor dem Aufruf in das Registerpaar BC geladen wird.

Die Funktion Bios wird wie die Prozedur aufgerufen und gibt einen Integer-Wert aus, der dem Wert entspricht, der vom BIOS in das A-Register ausgegeben wird.

17.9.4 Die Funktion BiosHL

Syntax: BiosHL (Func,Param)

Diese Funktion entspricht der Funktion Bios, jedoch wird das Ergebnis über das Registerpaar HL ausgegeben.

17.10 Benutzergeschriebene I/O-Treiber

I/O-Treiber sind zu definieren, um mit externen Geräten kommunizieren zu können. Die folgenden Parameter sind Teil von KCPASCAL und werden durch die Standard-I/O-Treiber benutzt (obwohl sie nicht als Standardprozeduren oder -funktionen zur Verfügung stehen):

```
function    ConSt  : boolean;
function    ConIn  : Char;
procedure   ConOut (Ch : Char);
procedure   LstOut (Ch : Char);
procedure   AuxOut (Ch : Char);
function    AuxIn  : Char;
procedure   UsrOut (ch : Char);
function    UsrIn  : Char;
```

Die Routine ConSt wird durch die Funktion KeyPressed aufgerufen. Die ConIn- und ConOut-Routinen können von den CON:-, TRM:- und KBD:-Geräten benutzt werden. Die LstOut-Routine wird von dem LST:-Gerät benutzt. Die AuxOut- und AuxIn-Routinen werden von dem AUX:-Gerät, UsrIn und UsrOut von dem USR:-Gerät benutzt.

Laut Voreinstellung benutzen diese Treiber die entsprechenden BIOS-Eingangspunkte des Betriebssystems, d.h., ConSt benutzt CONST. ConIn benutzt CONIN, ConOut benutzt CONOUT, LstOut benutzt LIST, AuxOut benutzt PUNCH, AuxIn benutzt READER, UsrOut benutzt CONOUT und UsrIn benutzt CONIN. Dies kann jedoch vom Programmierer verändert werden, indem er eine der folgenden Standardvariablen der Adresse in einer selbst definierten Treiberprozedur oder Treiberfunkktion zuweist:

Variable	enthält Adresse der
ConStPtr	ConSt-Funktion
ConInPtr	ConIn-Funktion
ConOutPtr	ConOut-Prozedur
LstOutPtr	LstOut-Prozedur
AuxOutPtr	AuxOut-Prozedur
AuxInPtr	AuxIn-Funktion
UsrOutPtr	UsrOut-Prozedur
UsrInPtr	UsrIn-Funktion

Eine vom Benutzer definierte Treiberprozedur oder Treiberfunktion muß den oben gegebenen Definitionen entsprechen, d.h., ein ConSt-Treiber muß eine Bool'sche Funktion sein, ein ConIn-Treiber muß eine Char-Funktion sein usw.

17.11 Externe Unterprogramme

Das reservierte Wort "external" wird benutzt, um externe Prozeduren und Funktionen zu deklarieren, typischerweise in Assembler geschriebene Prozeduren und Funktionen.

Ein externes Unterprogramm hat keinen Block, d.h. keinen Deklarierungs- und Anweisungsteil. Es wird nur der Unterprogrammkopf spezifiziert, unmittelbar gefolgt von dem reservierten Wort external und einer

ganzzahligen Konstanten, die die Speicheradresse des Unterprogramms definiert.
Parameter können an externe Unterprogramme übergeben werden. Die Syntax ist genau dieselbe wie bei normalen Prozedur- und Funktionsaufrufen:

```
procedure Plot(X,Y: Integer);external $F003;
procedure QuickSort(VarList:PartNo); external $D000;
```

17.12 Inline Maschinencode (Assembler)

KCPASCAL enthält inline-Anweisungen, die einen sehr bequemen Weg bieten, Maschinencode (Assembler) direkt in den Programmtext einzufügen. Eine inline-Anweisung besteht aus dem reservierten Wort "inline", gefolgt von einer oder mehreren Konstanten, Variablenbezeichnern oder Kommandozähler-referenzen, die durch Schrägstriche getrennt und in Klammern einge-schlossen sind.

Ein Codeelement ist aus einem oder mehreren Datenelementen aufgebaut, die durch "±" oder "-" Zeichen getrennt sind. Ein Datenelement ist entweder eine integer-Konstante, ein Funktionsbezeichner oder ein Kommandozähler-wert. Ein Kommandozählerwert wird als * (Stern) geschrieben.'

```
Beispiel:
inline (10$23-5/count - 1/sort-* -2);
```

Diese inline-Anweisung erzeugt drei Bytes Code: $34, $44 und $00.

Jedes Codeelement erzeugt ein Byte oder ein Wort (zwei Bytes) Code. Der Wert eines Bytes oder Wortes wird durch Addition oder Subtraktion der Werte des Datenelementes, je nach Trennungszeichen, errechnet. Der Wert eines Variablenbezeichners ist die Adresse (oder der Offset) der Variablen. Der Wert eines Prozedur- oder Funktionsbezeichners ist die Adresse (oder der Offset) des Kommandozählers, d.h. die Adresse, an der das nächste Byte Code erzeugt wird.

Ein Codeelement erzeugt ein Byte Code, wenn es nur aus einer integer-Konstanten besteht und wenn der Wert innerhalb des 8-Bit-Bereiches (0...255) liegt. Wenn der Wert außerhalb des 8-Bit-Bereiches liegt, oder das Codeelement sich auf einen Variablen-, Prozedur- oder Funktions-bezeichner bezieht oder das Codeelement einen Kommandozählerwert enthält, wird ein Wort Code (das niedrigste Byte steht zuerst) erzeugt.

Die Zeichen < und > können verwendet werden, um die oben beschriebene automatische Größenwahl zu überschreiben. Wenn das Codeelement mit dem Zeichen < beginnt, wird nur das wenigst signifikante Byte des Werts codiert, auch wenn es sich um einen 16-Bit-Wert handelt. Wenn das Codeelement mit dem Zeichen > beginnt, wird immer eine Wort codiert, auch wenn das niedrigste Byte 0 ist.

Das folgende Beispiel einer inline-Anweisung generiert den Maschinencode, der alle Zeichen in ihrem Stringargument in Großbuchstaben umwandelt.

```
Type
  Str = String[255];
procedure UpperCase (Var Strg:str);
{$A-}
begin
  inline
    ($2A/strg/    {LD   HL,(Strg)}
```

```
    $04/          {     INC   B     }
    $05/          {L1:  DEC   B     }
    $CA/*+20/     {     JP    Z,L2  }
    $23/          {     INC   HL    }
    $7E/          {     LD    A,(HL)}
    $FE/$61/      {     CP    'a'   }
    $DA/*-9/      {     JP    C,L1  }
    $FE/$7B/      {     CP    'z'±1 }
    $D2/*-14/     {     JP    NC,L1 }
    $D6/$20/      {     SUB   20H   }
    $77           {     LD    HL,A  }
    $C3/*-20);    {     JP    L1    }
                  {L2:  EQU   $     }
end;
```

Inline-Anweisungen können innerhalb des Anweisungsteils eines Blockes jederzeit mit anderen Anweisungen gemischt werden, und sie können alle Register der CPU benutzen. Beachten Sie, daß der Inhalt eines Stackzeiger-Registers (SP) am Ein- und Ausgang einer inline-Routine der gleiche sein muß.

Anhang A. Zusammenfassung der Standardprozeduren und Standardfunktionen

Dieser Anhang listet alle in KCPASCAL verfügbaren Standardprozeduren und -funktionen auf und beschreibt ihre Syntax, ihre Parameter und ihre Typen. Die folgenden Symbole bezeichnen Elemente verschiedenen Typs:

```
type        beliebiger Typ
string      beliebiger Stringtyp

file        beliebiger Dateityp
scalar      beliebiger Skalartyp
pointer     beliebiger Zeigertyp
```

Wo keine Parametertypspezifikation vorhanden ist, bedeutet das, daß die Prozedur oder die Funktion Variablenparameter beliebigen Typs akzeptiert.

A.1 Ein-/Ausgabeprozeduren und -funktionen

Die folgenden Prozeduren benutzen in ihrer Parameterliste nicht die Standardsyntax:

```
procedure
  Read (var F:file of type; var v: type;
  Read (var F:text;var I: Integer);
  Read (var F:text;var R: Real);
  Read (var F:text;var C: Char);
  Read (var F:text;var S: string);
  Readln (var F:text);
  Write (var F:file of type; var v:type);
  Write (var F:text;var I: Integer);
  Write (var F:text;var R: Real);
  Write (var F:text;var B: Boolean);
  Write (var F:text;var C: Chat);
  Write (var F:text;var S: string);
  Writeln (var F:text);
```

A.2 Arithmetische Funktionen

```
funktion
  Abs(I:Integer):Integer;
  Abs(R:Real):Real;
  ArcTan(r:Real):Real;
  Cos(R:Real):Real;
  Exp(R:Real):Real;
  Frac(R:Real):Real;
  Int(R:Real):Real;
  Ln (R:Real):Real;
  Sin (R:Real):Real;
  Sqr (I:Integer):Integer;
  Sqr (R:Real):Real;
  Sqrt (R:Real):Real;
```

A.3 Skalarfunktionen

```
funktion
  Odd(I:Integer):Boolean;
  Pred(X:scalar):scalar;
  Succ(X:scalar):scalar;
```

A.4 Transferfunktionen

```
funktion
  Chr(I:Integer):Char;
  Ord(X:scalar):scalar;
  Round(R:real):Integer;
  Trunc(R:Real):Integer;
```

A.5 Stringprozeduren und -funktionen

Die Str-Prozedur benutzt nichtstandardisierte Syntax für ihre numerischen Parameter.

```
procedure
  Delete(var S:string;Pos,Len:Integer);
  Insert(S:string;var D:string;Pos:Integer);
  Str(I:Integer;var S:string);
  Str(R:Real;var S:string);
  Val(S:string;var R:Real;var P:Integer);
  Val(S:string;var I,P:Integer);

function
  Concat(S1,S2,..,Sn:string):string;
  Copy(S:string;Pos,Len:Integer):string;
  Length(S:string):Integer;
  Pos(Pattern,Source:string):Integer;
```

A.6 Heap Kontrollprozeduren und -funktionen

```
procedure
  Dispose(var P:pointer);
  FreeMem(var P:pointer,I:Integer);
  GetMem(var P:pointer,I:Integer);
  Mark(var P:pointer);
  New(var P:pointer);
  Release(var P:pointer);

function
  MaxAvail:Integer;
  MemAvail:Integer;
  Ord(P:pointer):Integer;
  Ptr(I:Integer):pointer;
```

A.7 Bildschirmbezogene Prozeduren

```
procedure
  CtrScr;
  GotoXY(X,Y:Integer);
```

A.8 Verschiedenartige weitere Prozeduren und Funktionen

```
procedure
  Bdos(func,param:Integer);  (nur CP/M-80)
  Bios(func,param:Integer);  (nur CP/M-80)
  Delay(mS:Integer);
  FillChar(var dest;length:Integer;data:Char);
  FillChar(var dest;length:Integer;data:byte);
  Halt;
  Move(var source,dest,length:Integer);
  Randomize;

function
  Addr(var Variable):Integer;
  Addr(<function identifier>):Integer;
  Addr(<procedure identifier>):Integer;
  Bdos(Func,Param:Integer):Byte;
  BdosHL(Func,Param:Integer):Integer;
  Bios(Func,Param:Integer):Byte;
  BiosHL(Func,Param:Integer):Integer;
  Hi(I:Integer):Integer;
  IOresult*Boolean;
  KeyPressed:Boolean;
  Lo(I:Integer):Integer;
  ParamCount:Integer;
  ParamStr(N:Integer):String;
  Random(Range:Integer):Integer;
  Random:Real;
  SizeOf(var Variable):Integer;
  SizeOf(<type identifier>):Integer;
  Swap(I:Integer):Integer;
  UpCase(Ch:Char):Char;
```

B. Zusammenfassung der Operatoren

Die folgende Tabelle gibt eine Übersicht über alle Operatoren von KCPASCAL. Die Operatoren sind nach abnehmender Wichtigkeit gruppiert. Wo der Typ des Operanden als Integer.Real angegeben ist, ist das Ergebnis wie folgt:

Operand	Ergebnis
Integer,Integer	integer
Real,Real	Real
Real,Integer	Real

Operator	Wirkung	Type des Operanden(S)	Ergebnistyp
.monadisch	Zeichenidentität	Integer,Real	wie Operand
.monadisch	Zeichenumkehrung	Integer,Real	wie Operand
not	Negation	Integer,Boolean	wie Operand
*	Multiplikation	Integer,Real	Integer,Real
	Schnittmenge	jeder Mengentyp	wie Operand
div	Division	Integer,Real	Real
	IntegerDivision	Integer	Integer
mod	Modulus	Integer	Integer
and	arithmet. und	Integer	Integer
	logisches und	Boolean	Boolean
shl	Shift nach links	Integer	Integer
shr	Shift nach rechts	Integer	Integer

+	Addition	Integer,Real	Integer,Real
	Verkettung	string	string
	Mengenvereinigung	jeder Mengentyp	wie Operand
-	Subtraktion	Integer,Real	Integer,Real
	Mengenschnitt	jeder Mengentyp	wie Operand
or	arithmet. oder	Integer	Integer
	logisches oder	Boolean	Boolean
xor	arithmet. exoder	Integer	Integer
	logische exoder	Boolean	Boolean

Operator	Wirkung	Type des Operanden	Ergebnistyp
=	Gleichheit	jeder Skalartyp	Boolean
	Gleichheit	String	Boolean
	Gleichheit	jeder Mengentyp	Boolean
	Gleichheit	jeder Zeigertyp	Boolean
<>	Ungleichheit	jeder Skalartyp	Boolean
	Ungleichheit	String	Boolean
	Ungleichheit	jeder Mengentyp	Boolean
	Ungleichheit	jeder Zeigertyp	Boolean
>=	größer oder gleich	jeder Skalartyp	Boolean
	größer oder gleich	String	Boolean
	Mengeneinschluß	jeder Mengentyp	Boolean
<=	kleiner oder gleich	jeder Skalartyp	Boolean
	kleiner oder gleich	String	Boolean
	Mengeneinschluß	jeder Mengentyp	Boolean
>	größer als	jeder Skalartyp	Boolean
	größer als	String	Boolean
<	kleiner als	jeder Skalartyp	Boolean
	kleiner als	String	Boolean
in	Mitglied einer Menge	siehe unten	Boolean

Der erste Operand des "in"-Operators kann von beliebigem Skalartyp sein, der zweite Operand muß eine Menge dieses Type sein.

C. Zusammenfassung der Compilerbefehle

Einige der Eigenschaften des KCPASCAL-Compilers werden durch Compilerbefehle kontrolliert. Ein Compilerbefehl wird als Kommentar mit spezieller Syntax eingeführt, was bedeutet, daß überall wo ein Kommentar erlaubt ist, auch ein Compilerbefehl erlaubt ist.

Ein Compilerbefehl besteht aus einer geschweiften Klammer auf, unmittelbar gefolgt von einem Compilerbefehlsbuchstaben oder einer Liste von Compilerbefehlsbuchstaben, die durch Kommas getrennt sind. Ein Compilerbefehl wird schließlich mit einer geschweiften Klammer abgeschlossen.

Beispiele:

```
{$I-}
{$I INCLUDE.FIL}
{$B-,R+,V-}
{$U+}
```

Beachten Sie, daß keine Zwischenräume vor und nach dem Dollarzeichen erlaubt sind. Ein "+" Zeichen nach einem Befehl zeigt an, daß die damit verbundene Eigenschaft in Kraft gesetzt wird (aktiv) und ein "-" Zeichen zeigt, daß sie außer Kraft gesetzt ist (passiv).

C.1 ACHTUNG !!!

Alle Compilerbefehle haben voreingestellte Werte. Diese wurden so gewählt, daß die Ausführungsgeschwindigkeit und die Codegröße optimiert wird. Das bringt mit sich, daß z.B. die Erzeugung von Code für rekursive Prozeduren und Indexprüfung außer Kraft gesetzt wurden. Prüfen Sie im Zweifelsfall also, ob ihre Programme die benötigten Compilerbefehls-einstellungen enthalten!

C.2 Allgemeine Compilerbefehle

C.2.1 B - I/O-Modusauswahl

Der B-Befehl kontrolliert die Auswahl des Ein-/Ausgabemodus. Wenn der Modus aktiv ist (SB+), ist das CON:Gerät (Konsole) den Standarddateien Input und Output zugewiesen, d.h. dem voreingestellten Eingabe-/Ausgabe-kanal. Im Passivmodus (SB-) wird das TRM:Gerät (Terminal) benutzt. Dieser Befehl ist für das ganze Programm gültig und kann nicht innerhalb des Programms umdefiniert werden.

C.2.2 C - Control-S und Control-C

Der C-Befehl steuert die Interpretation der Kontrollzeichen während der Ein-/Ausgabe durch die Konsole. Im Aktivmodus (SC+) unterbricht ein CTRL-C als Antwort auf eine Read- oder Readln-Anweisung die Programmausführung und CTRL-S schaltet die Bildschirmausgabe an und aus. Im Passivmodus (SC-) werden Kontrollzeichen nicht interpretiert. Der Aktivmodus verlangsamt die Bildschirmausgabe etwas, falls Ihnen die Ausgabegeschwindigkeit wichtig ist, müßten Sie diesen Befehl ausschalten. Dieser Befehl ist für das ganze Programm gültig und kann nicht während des Programms umdefiniert werden.

C.2.3 I - Eingabe/Ausgabefehler-Handhabung

Voreinstellung: I-

Der I-Befehl kontrolliert die Ein-/Ausgabefehler-Handhabung. Im Aktiv-modus {$I+} werden alle Ein-/Ausgabe-Operationen auf Fehler geprüft. Im Passivmodus {$I-} liegt es in der Verantwortung des Programmmierers. I/O-Fehler sind durch die Standardfunktion IOresult zu prüfen.

C.2.4 I - Includedateien

Der I-Befehl, gefolgt von einem Dateinamen, weist den Compiler an, die Datei mit dem angegebenen Namen in die Compilierung aufzunehmen.

C.2.5 R - Index-Bereichsprüfung

Voreinstellung: R-

Der R-Befehl kontrolliert die Indexprüfung zur Laufzeit. Im Aktivmodus {$R+} werden alle Feldindizierungsoperationen darauf geprüft, ob sie innerhalb der definierten Grenzen sind, ebenfalls alle Zuweisungen zu Skalaren und Teilbereichsvariablen. Im Passivmodus {$SR-} werden keine Prüfungen unternommen und Indexfehler können ein Programm dann durcheinanderbringen. Bei der Programmentwicklung empfiehlt es sich, diesen Befehl zu benutzen. Wenn es dann fehlerfrei ist, wird die Ausführung beschleunigt, indem der Passivmodus (Voreinstellung) eingestellt wird.

C.2.6 V - Parametertypprüfung

Voreinstellung: V-

Der V-Compilerbefehl kontrolliert die Typenprüfung von Strings, die als var-Parameter übergeben werden. Im Aktivmodus {$SV+} wird strenge Typenprüfung durchgeführt, d.h., die Länge der aktuellen und formalen Parameter muß übereinstimmen. Im Passivmodus {$V-} erlaubt der Compiler die Übergabe von aktuellen Parametern auch dann, wenn sie nicht zu der Länge der formalen Parameter passen.

C.2.7 U - Benutzerunterbrechung

Voreinstellung: U-

Der U-Compilerbefehl kontrolliert Programmunterbrechungen durch den Benutzer. Im Aktivmodus {$U+} kann der Benutzer das Programm jederzeit bei der Ausführung unterbrechen, indem er CTRL-C eingibt. Im Passivmodus hat diese Eingabe keine Wirkung. Die Aktivierung dieses Befehls vermindert die Ausführungsgeschwindigkeit beträchtlich.

C.2.8 A - Absoluter Code

Voreinstellung: A+

Der A-Befehl kontrolliert die Erzeugung von absoluten, d.h. nichtrekursivem Code. Im Aktivmodus {$A+} wird absoluter Code erzeugt. Im Passivmodus {$A-} erzeugt der Compiler einen Code, der rekursive Aufrufe erlaubt. Dieser Code benötigt mehr Speicher und ist langsamer in der Ausführung.

C.2.9 W - Schachtelung von With-Anweisungen

Voreinstellung: W2

Der W-Befehl kontrolliert das Niveau der Schachtelung von With-Anweisungen, d.h. der Zahl von Records, die innerhalb eines Blocks geöffnet werden können. Das W muß unmittelbar von einer Ziffer zwischen 1 und 9 gefolgt sein. Für weitere Hinweise siehe Seite 81.

C.2.10 X - Arrayoptimierung

Voreinstellung: X+

Der X-Befehl kontrolliert die Arrayoptimierung. Im Aktivmodus {$X+} ist die Codererzeugung auf maximale Geschwindigkeit hin optimiert. Im Passivmodus {$X-} minimiert der Compiler stattdessen die Codegröße. Dies wird auf Seite 75 weiter erläutert.

D. KCPASCAL VS. Standard-Pascal

Die KCPASCAL-Sprache folgt sehr weitgehend Standard-Pascal, wie es von Jensen und Wirth in ihrem "User Manual and Report" definiert ist. Die vorhandenen kleineren Abweichungen sind aus Gründen der Effizienz eingeführt. Diese Unterschiede werden im folgenden beschrieben. Beachten Sie, daß die Erweiterungen, die KCPASCAL anbietet, hier nicht erläutert sind.

D.1 Dynamische Variablen

Dynamische Variablen und Zeiger benutzen die Standardprozeduren New Mark und Release, statt der New- und Dispose-Prozeduren, die von Standard-Pascal vorgeschlagen werden. Diese Abweichung vom Standard ist vor allem weit effizienter bezüglich der Ausführungsgeschwindigkeit und des benötigten Codes, außerdem bietet sie Kompatibilität mit anderen weitverbreiteten Pascalcompilern (z.B. UCSD-Pascal).

D.2 Rekursion

Wegen der Art, wie lokale Variablen während der Rekursion behandelt werden, darf eine zu einem Unterprogramm lokale Variable nicht als var-Parameter in rekursive Aufrufe übergeben werden.

D.3 Get und Put

Die Standardprozeduren Get und Put sind nicht implementiert. Dateiarbeit ist nicht möglich.

D.4 Goto-Anweisungen

Eine Goto-Anweisung darf den aktuellen Block nicht verlassen.

D.5 Page-Prozedur

Die Standardprozedur Page ist nicht implementiert, da das CP/M-Betriebssystem kein Seitenvorschubzeichen definiert.

D.6 Gepackte Variablen

Das reservierte Wort "packed" hat in KCPASCAL keine Wirkung, aber es ist dennoch erlaubt, weil Packung automatisch vorgenommen wird, wo immer es möglich ist. Aus demselben Grund sind die Standardprozeduren von Pack und Unpack nicht implementiert.

D.7 Prozedurale Parameter

Prozeduren und Funktionen können nicht als Parameter übergeben werden.

E. Compiler-Fehlermeldungen

Es folgt eine Liste von Fehlermeldungen, die Sie vom Compiler bekommen können. Wenn ein Fehler auftritt, gibt der Compiler mindestens immer die Fehlernummer aus. Erklärende Texte werden nur ausgegeben, wenn Sie die Fehlermeldungsdatei auf der KCPASCAL-Kassette geladen haben (Antwort Y auf die erste Frage beim Start von KCPASCAL).

Viele Fehlermeldungen erklären sich selbst, aber einige benötigen weitere Erläuterungen, wie sie im folgenden gegeben werden.

01 ';' erwartet
02 ':' erwartet
03 ' ' erwartet
04 '(' erwartet
05 ')' erwartet
06 '=' erwartet
07 ':=' erwartet
08 ' ' erwartet
09 ' ' erwartet
10 '.' erwartet
11 '..' erwartet
12 BEGIN erwartet
13 DO erwartet
14 END erwartet
15 OF erwartet
16 PROCEDURE oder FUNCTION erwartet
17 THEN erwartet
18 TO oder DOWNTO erwartet
20 Bool'scher Begriff erwartet
21 Datei-Variable erwartet
22 Integer-Konstante erwartet
23 Integer-Ausdruck erwartet
24 Integer-Variable erwartet
25 Integer- oder reelle Konstante erwartet
26 Integer- oder reeller Ausdruck erwartet
27 Integer- oder reelle Variable erwartet
28 Zeiger-Variable erwartet
29 Record-Variable erwartet
30 Einfacher Typ erwartet
Einfache Typen sind alle skalaren Typen außer Real.
31 Einfacher Ausdruck erwartet
32 Stringkonstante erwartet
33 Stringausdruck erwartet
34 Stringvariable erwartet
35 Textdatei erwartet
36 Typenbezeichner erwartet
37 Untypisierte Datei erwartet
40 Undefiniertes Label
Eine Anweisung weist auf ein undefiniertes Label hin.
41 Unbekannter Bezeichner oder Syntaxfehler
Unbekannt: Label, Konstante, Type, Variable, Feldbezeichner oder Syntaxfehler in der Anweisung
42 Undefinierter Zeigertyp in vorhergehenden Typdefinitionen
Eine vorhergehende Zeigertypdefinition enthält einen Verweis auf einen unbekannten Typenbezeichner.

43 Doppelter Bezeichner oder doppeltes Label
Dieser Bezeichner oder dieses Label wurde schon in dem laufenden Block verwendet.

44 Unpassende Typen
1) Inkompatibler Typ einer Variablen oder eines Ausdrucks in einem Zuweisungsstatement
2) Inkompatibler Typ von aktuellem und formalem Parameter in einem Unterprogrammaufruf
3) Typ des Ausdrucks ist inkompatibel mit dem Indextyp in der Arrayzuweisung
4) Die Typen von Operanden in einem Ausdruck sind nicht kompatibel

45 Konstante außerhalb der Grenze

46 Konstanten und CASE-Selektortyp passen nicht zusammen

47 Typ des Operanden paßt nicht zum Typ, z.B. 'A'div'2'

48 Ungültiger Ergebnistyp
Gültige Typen sind alle Skalar-, String- und Zeigertypen.

49 Ungültige Stringlänge
Die Länge eines Strings muß im Bereich ...255 liegen.

50 Stringkonstantenlänge paßt nicht zum Typ

51 Ungültiger Teilbereichsgrundtyp
Gültige Grundtypen sind alle Skalartypen außer real.

52 Untere Grenze, obere Grenze
Der ordinale Wert der oberen Grenze muß größer oder gleich dem ordinalen Wert der unteren Grenze sein.

53 Reserviertes Wort
Diese dürfen nicht als Bezeichner verwendet werden.

54 Unerlaubte Zuweisung

55 Stringkonstante geht über die Zeile hinaus
Stringkonstanten dürfen sich nicht über die Zeile hinaus erstrecken.

56 Fehler bei einer Integer-Konstanten
Eine Integer-Konstante stimmt nicht mit der in Abschnitt 4.2 beschriebenen Syntax überein oder ist nicht innerhalb des Integer-Bereichs -32768..32767. Ganze reelle Zahlen sollten von einem Dezimalpunkt und einer Null abgeschlossen werden, z.B. 123456789.0

57 Fehler bei einer Real-Konstanten
Die Syntax der Real-Konstanten ist auf Seite 43 definiert.

58 Unerlaubtes Zeichen in einem Bezeichner

60 Konstanten sind hier nicht erlaubt

61 Dateien und Zeiger sind hier nicht erlaubt

62 Strukturierte Variablen sind hier nicht erlaubt

63 Textdateien sind hier nicht erlaubt

64 Textdateien und untypisierte Dateien sind hier nicht erlaubt

65 Untypisierte Dateien sind hier nicht erlaubt
Variablen dieses Typs können nicht ein- oder ausgegeben werden.

66 Eingabe/Ausgabe ist hier nicht erlaubt
Variablen dieses Typs können nicht ein- oder ausgegeben werden.

67 Dateien müssen var-Parameter sein

68 Dateikomponenten dürfen keine Dateien sein

69 Ungültige Ordnung von Feldern

70 Mengengrundtyp außerhalb des zulässigen Bereichs
Der Grundtyp einer Menge muß ein Skalar mit nicht mehr als 256 möglichen Werten sein oder ein Teilbereich mit den Grenzen 0..255.

71 Unerlaubtes GOTO
Ein goto kann nicht auf ein Label innerhalb einer FOR-Schleife von außerhalb dieser FOR-Schleife hinweisen.

72 Label nicht innerhalb des gegenwärtigen Blocks
Ein goto kann nicht auf ein Label außerhalb des gegenwärtigen Blocks hinweisen.

73 Undefinierte FORWARD-Prozedur(en)
Ein Unterprogramm wird foreward deklariert, aber es ist kein Block vorgekommen.
74 INLINE-Fehler
75 Unerlaubter Gebrauch von ABSOLUTE
1) Nur Bezeichner können vor dem Doppelpunkt in einer absolute Variablendeklaration auftreten.
2) Absolute darf in einem Record nicht verwendet werden.
76 Overlays können nicht FORWARD deklariert werden
FORWARD kann nicht in Verbindung mit Overlays verwendet werden.
77 Im Direktmodus sind Overlays nicht erlaubt
Overlays können nur von Programmen verwendet werden, die auf eine Datei compiliert sind.
90 Datei nicht gefunden
Die angegebene Include-Datei existiert nicht.
91 Unerwartetes Ende der Quelle
Ihr Programm kann nicht richtig enden. Das Programm mehr begin- als end-Angaben.
92 Es kann keine Overlaydatei gebildet werden
97 Zuviele geschachtelte WITHs
Benutzen Sie den W-Compilerbefehl, um die maximale Zahl von WITH-Anweisungen zu erhöhen.
Voreinstellung ist 2.
98 Speicherüberlauf
Sie versuchen, mehr Speicherplatz für Variablen zur Verfügung zu stellen, als vorhanden ist.
99 Compilerüberlauf
Es ist nicht genügend Speicherplatz vorhanden, um das Programm zu compilieren. Dieser Fehler kann auch auftreten, wenn freier Speicherplatz dazusein scheint; dieser ist jedoch vom Stack und der Symboltafel bei der Compilierung belegt. Teilen Sie Ihre Quelle in kleinere Segmente auf und benutzen Sie Include-Dateien.

F. Laufzeit-Fehlermeldungen

Schwere Laufzeitfehler bewirken einen Programmabbruch und eine Anzeige folgender Fehlermeldung:

```
Run-time error NN.PC=addr
Program aborted
```

wobei NN die Nummer des Laufzeitfehlers ist und addr die Adresse im Programmcode, bei der der Fehler aufgetreten ist. Die Bedeutung der Nummern wird im folgenden erklärt. Beachten Sie, daß alle Zahle hexadezimal sind!

01 Gleitkommaüberlauf
02 Sie haben versucht, durch Null zu dividieren
03 Sqrt Argumentfehler
Sie haben versucht, die Wurzel aus einer negativen Zahl zu ziehen, d.h., Sie haben die Sqrt-Funktion aufgerufen und als Argument eine negative Zahl eingegeben.
04 Ln-Argumentfehler
Sie haben die Ln-Funktion aufgerufen und als Argument Null oder eine negative Zahl eingegeben, d.h., versucht, den Logarithmus von Null oder einer negativen Zahl zu ermitteln.

10 String-Längenfehler
1) Eine Verkettung von Strings ergab einen String von mehr als 255 Zeichen.
2) Nur Strings von der Länge 1 können in Buchstabenzeichen umgewandelt werden.

11 Ungültiger Stringindex
Der Stringindex liegt bei Copy-, Delete- oder Insert-Prozeduren nicht zwischen 1 und 255.

90 Index außerhalb des zulässigen Bereichs
Der Index eines Arrays liegt nicht im zulässigen Bereich.

91 Skalar oder Teilbereich außerhalb des zulässigen Bereichs
Sie haben einem Skalar oder Teilbereich einen Wert zugeordnet, der nicht im zulässigen Bereich liegt.

92 Außerhalb des Integer-Bereiches
Sie haben bei Trunc oder Round einen ganzzahligen Wert eingegeben, der außerhalb des integeren Bereiches von -32767...32767 liegt.

FF Heap/Stackkollision
Sie haben die Standardprozedur New oder ein rekursives Unterprogramm aufgerufen, aber es ist nicht genügend freier Speicherplatz vorhanden

G. I/O-Fehlermeldungen

Ein Fehler während einer Ein-/Ausgabeoperation ist ein I/O-Fehler. Wenn die I/O-Fehlerroutine in Betrieb ist, (der I-Compilerbefehl aktiviert ist), verursacht ein I/O-Fehler eine Programmunterbrechung und die Anzeige der folgenden Fehlermeldung:

```
I/O error NN.PC=addr
Program aborted
```

wobei NN die I/O-Fehlernummer ist und addr die Programmadresse, bei der der Fehler aufgetreten ist.

Wenn die automatische I/O-Fehlerroutine nicht in Betrieb ist (SI-), wird das Programm bei Auftreten eines I/O-Fehlers nicht unterbrochen. Stattdessen werden alle weiteren I/O-Anweisungen erst dann ausgeführt, wenn das Ergebnis der I/O-Operation mit Hilfe der Standardfunktion IOresult überprüft worden ist. Wenn versucht wird, eine I/O-Operation auszuführen, ohne daß nach einem Fehler IOresult aufgerufen wurde, kann dies einen Programmstillstand zur Folge haben.

Es folgt eine Erläuterung aller Laufzeit-Fehlernummern. Beachten Sie, daß es sich dabei um hexadezimale Zahlen handelt!

01 Datei ist nicht vorhanden
Sie haben bei Reset, Erase, Rename, Execute oder Chain eine Datei spezifiziert, die nicht vorhanden ist.

02 Lesen der Datei nicht möglich
1) Sie versuchen, von einer Datei zu lesen (mit Read oder Readln), ohne vorheriges Reset oder Rewrite.
2) Sie versuchen, von einer Textdatei zu lesen (die mit Rewrite vorbereitet, aber noch leer ist).
3) Sie versuchen vom logischen Gerät LST: (Drucker) einzulesen. LST: ist jedoch ein reines Ausgabegerät.

03 Ausgabe in die Datei nicht möglich
1) Sie versuchen, in eine Datei zu schreiben, (mit Write oder Writeln), ohne vorheriges Rest oder Rewrite.
2) Sie versuchen, in eine Textdatei zu schreiben, die mit Reset vorbereitet wurde.
3) Sie versuchen, auf das logische Gerät KBD: (Tastatur) zu schreiben. KBD: ist jedoch ein reines Eingabegerät.

04 Datei nicht offen
Sie versuchen (mit BlockRead oder BlockWrite) eine Datei zu bearbeiten, ohne vorheriges Reset oder Rewrite.

10 Fehler im numerischen Format
Der String, der von einer Textdatei in eine numerische Variable eingelesen wurde, entspricht nicht dem richtigen numerischen Format (siehe Abschnitt 4.2).

20 Operation auf einem logischen Gerät nicht zugelassen
Sie versuchen, eine Datei, die einem logischen Gerät zugeordnet ist, mit einer der nachfolgenden Operationen zu bearbeiten: Erase, Rename, Execute oder Chain.

21 Im Direktmodus nicht zugelassen
Programme können nicht mit Execute oder Chain von einem Programm aufgerufen werden, das im Direktmodus läuft (d.h. ein Programm, das mit der Compileroption Memory mit dem Run-Befehl aufgerufen wurde).

22 Zuordnung als Standarddatei nicht zulässig

90 Unpassende Recordlänge
Die Recordlänge einer Dateivariablen entspricht nicht der Datei, der Sie sie zuzuordnen versuchen.

91 Unerwartetes end-of-file
1) Beim Lesen von einer Textdatei wurde das physikalische Dateiende vor dem EOF-Zeichen erreicht.
2) Es wurde versucht, bei einer fest definierten Datei über das end-of-file hinauszulesen.

F0 Kassettenschreibfehler
Beim Versuch, eine Datei zu vergrößern, ist die Kassette voll geworden. Dies kann bei den Ausgabeoperationen Write, Writeln, BlockWrite und Flush auftreten, aber auch Read, Readln und Close können diesen Fehler verursachen, wenn Sie den Schreibpuffer zum Überlaufen bringen.

F1 Verzeichnis ist voll
Sie versuchen, eine neue Datei zu erstellen, aber es ist ein Platz mehr im Verzeichnis.

F2 Dateigrößenüberschreitung
Sie versuchen, auf einer definierten Datei auf einen Record über 65535 hinaus zu schreiben.

FF Datei verschwunden
Es wurde ein Versuch unternommen, eine Datei zu schließen, die nicht mehr im Verzeichnis vorhanden ist, z.B. weil die Kassette gewechselt wurde.