

CPC464

HiSoft PASCAL 4T

herausgegeben von der
Schneider Computer Division
Silvastraße 1
D-8939 Türkheim 1

Vervielfältigung und Weitergabe
– auch auszugsweise –
dieses Handbuches und der
dazugehörigen Programmcassette
bedürfen der vorherigen
schriftlichen Zustimmung
der Schneider Computer Division.

SW 155
Erste Ausgabe 1984
Originalausgabe in Englisch
Originalcopyright © David Link, David Nutkins, 1983, 1984
Deutsche Bearbeitung: ESCON GmbH, Freising

ABSCHNITT 0 EINLEITUNG

0.0 Einführung

HiSoft Pascal für den Schneider CPC464 (HP464) ist eine schnelle, leistungsfähige und leicht anwendbare Implementierung der Programmiersprache Pascal. Sie entspricht den Spezifikationen des Pascal User Manual and Report (Jensen/Wirth Second Edition) mit folgenden Ausnahmen:

- Der Datentyp „Datei“ ist nicht implementiert, Variablen können jedoch auf Cassette gespeichert werden.
- Der Datentyp „Record“ darf keinen Variantenteil enthalten.
- Prozeduren und Funktionen sind nicht als Parameter gültig.

In der Praxis wirken sich diese Ausnahmen jedoch nicht einschränkend aus.

Viele zusätzliche Funktionen und Prozeduren sind einbezogen worden, um den sich ständig ändernden Umgebungsbedingungen von Compilern gerecht zu werden (z.B. POKE, PEEK, TIN, TOUT und ADDR). Eine ganze Reihe weiterer Funktionen steht zur Verfügung, um den vollen Leistungsumfang des Schneider CPC464 zu nutzen. So wird beispielsweise die Ereignis-Bearbeitung (AFTER, EVERY etc.) ebenfalls unterstützt. Weiterhin wird auf der zweiten Seite der Programm-Cassette ein Routinen-Paket für die Schildkröten-Graphik angeboten. Die Beschreibung dieser Graphik-Routinen befindet sich in Anhang 5 dieses Handbuchs.

Zum Lieferumfang gehört ebenfalls ein Satz selbstdokumentierender Routinen für den Anschluß von HiSoft Pascal an die Firmware des CPC464. Ein Listing dieser Routinen ist in Anhang 6 bereitgestellt.

Der Compiler nimmt ungefähr 12 KByte in Anspruch. Der Editor begnügt sich mit 2 KByte. Zur Laufzeit werden nochmals ca. 5 KByte belegt, wodurch für Programme und Daten gute 20 KByte verbleiben. HiSoft Pascal arbeitet als Vordergrund-RAM-Programm und wird vom normalen Vordergrund-Programm (BASIC) des CPC464 aufgerufen. Dadurch können BASIC und HiSoft Pascal nie gleichzeitig arbeiten, was auch nicht erforderlich ist, da alle Funktionen unter BASIC auch in Pascal verfügbar sind.

Um Ihnen Appetit zu machen und Ihnen gleich ein Gefühl für dieses Paket zu geben, zeigen wir Ihnen ein kurzes Beispiel zur Erstellung, Übersetzung und zum Ablauf eines Pascal-Programms. Wir empfehlen jedoch, alle Abschnitte dieses Handbuchs vor der Benutzung dieses Pakets sorgfältig zu studieren. Insbesondere sollten dieser Abschnitt, die Editor-Beschreibung (Abschnitt 4) und die Beispiel-Programme in Anhang 4 gelesen werden.

0.0.1 Ladehinweise

Zu Beginn muß der Compiler geladen werden. Legen Sie die Cassette so in den Datacorder, daß die Aufschrift „HiSoft Pascal“ lesbar ist, und drücken Sie die **PLAY**-Taste. Geben Sie daraufhin **CTRL [ENTER]** ein, indem Sie erst **CTRL** und dann die kleine **[ENTER]**-Taste gemeinsam niederdrücken. Ein kurzes BASIC-Ladeprogramm wird nun eingelesen und selbständig gestartet. Auf dem Bildschirm erscheint daraufhin die Meldung:

```
RAM top ([ENTER] to default)?
```

Sie sollten nun entweder eine Dezimalzahl und **[ENTER]** oder nur **[ENTER]** eingeben. Eine Dezimalzahl wird als höchste Speicheradresse interpretiert, die gerade nicht mehr durch Pascal verwendet wird.

Alle darunter liegenden Speicherplätze sind dann belegt und demzufolge überschrieben. Betätigen Sie auf diese Meldung nur die **[ENTER]**-Taste, so wird der Standardwert 45312 angenommen, der den größtmöglichen Speicherplatz für Ihre Pascal-Programme bereitstellt. In den meisten Fällen ist die Betätigung der **[ENTER]**-Taste ausreichend. Nur wenn der obere Speicherbereich durch Maschinencode-Programme belegt ist, die Sie von Pascal aus ansprechen wollen, sollte die Eingabe einer Dezimalzahl erfolgen.

Nach Festlegung der RAM-Obergrenze wird der Editor und ein Menü bereitgestellt.

0.0.2 Beispielprogramm

Wir nehmen an, daß Sie die RAM-Obergrenze durch Betätigen von **[ENTER]** festgelegt haben und das Editor-Promptzeichen „>“ auf dem Bildschirm erschienen ist. Geben Sie jetzt **I** und erneut **[ENTER]** ein.

Am linken Bildschirmrand erscheint nun die durch den Editor automatisch generierte Zeilennummer **10**. Der Editor setzt die automatische Erzeugung der Zeilennummern solange fort, bis Sie den **I**-Modus (siehe unten) verlassen. Geben Sie nun das folgende Programm ein und schließen Sie jede Zeile mit **[ENTER]** ab.

```
10 program hanoi;  
20 var n : integer;  
30 procedure movedisk(sce,dest : integer);  
40 begin  
50 write(sce:1,' to ',dest:1,'. ')  
60 end;  
70 procedure move(n,sce,aux,dest : integer);  
80 begin
```

```

90 if n=1 then movedisk(sec,dest)
100 else
110 begin
120 move(n-1,sce,dest,aux);
130 movedisk(sce,dest);
140 move(n-1,aux,sce,dest)
150 end;
160 end;
170 (*MainBlock*)
180 begin
190 write('Anzahl Scheiben? ');
200 read(n); writeln;
210 move(n,1,2,3)
220 end.
230 [ESC]

```

Beachten Sie, daß Kommentare durch (* *) eingeklammert (siehe Zeile 170) und Pascal-Anweisungen mit einem Semikolon „;“ abgeschlossen werden.

Nach Abschluß der Eingabe befinden Sie sich – erkennbar durch das Promptzeichen > – im Kommando – Modus des Editors.

Geben Sie nun C und [ENTER] ein. Der Compiler übersetzt daraufhin das Programm in Maschinencode und erzeugt ein Compiler-Listing. Nach Abschluß einer fehlerfreien Übersetzung erscheint „RUN?“ auf dem Bildschirm, andernfalls „*ERROR*“. Im Fehlerfall gelangen Sie durch Drücken von E in den Editor zurück. Betätigen Sie dann die [ENTER]-Taste und überprüfen Sie das Programm auf Übereinstimmung mit dem obigen Listing. Korrigieren Sie die fehlerhaften Zeilen, indem Sie deren Zeilennummer, ein Leerzeichen sowie den Zeilentext eingeben und abschließend die [ENTER]-Taste betätigen. Mit C können Sie das Programm erneut übersetzen.

Nach fehlerfreier Übersetzung beantworten Sie die Frage „RUN?“ mit Y. Das Programm wird gestartet und stellt Ihnen die Frage „Anzahl Scheiben?“.

Das Programm ist eine Lösung für das Spiel „Türme von Hanoi“. Es stellt drei Zapfen auf dem Bildschirm dar, von denen einer mit mehreren Scheiben unterschiedlicher Größe belegt ist, die in absteigender Ordnung aufgeschichtet sind.

Die Aufgabe besteht nun darin, alle Scheiben unter Wahrung der Reihenfolge auf anderen Zapfen zu stapeln. Geben Sie die Anzahl der Scheiben ein, die Sie auf den ersten Zapfen setzen wollen und drücken Sie [ENTER]. Beginnen Sie beispielsweise mit 3 Scheiben und geben Sie

3 [ENTER]

ein. Das Programm erzeugt nun eine Auflistung aller Bewegungen, die für die Scheiben-Umsetzung auf einen anderen Zapfen erforderlich sind.

Nach Abschluß des Programms erscheint erneut „RUN?“. Durch Eingabe von Y wird das Programm wiederholt, jede andere Taste stellt Ihnen den Editor zur Verfügung.

So, das war's für's erste! Lesen Sie jetzt bitte die restlichen Kapitel sorgfältig durch.

0.1 Inhaltsübersicht

Dieses Handbuch ist kein Einführungskurs in Pascal, sondern ein Nachschlagewerk für den Anwender von HiSoft Pascal. Als Einführung wird Ihnen das separate Lehrbuch zu HiSoft Pascal oder eines der in jeder Bibliothek erhältlichen Grundlagenbücher zur Programmierung in Pascal empfohlen.

Folgende Themen werden in diesem Handbuch behandelt:

- Abschnitt 1 beschreibt die Syntax und Semantik des Compilers.
- Abschnitt 2 behandelt die vordefinierten Namen (Bezeichner, Identifier) von HiSoft Pascal, von **CONST**anten bis zu **FUNCTI**ONen.
- Abschnitt 3 beinhaltet Informationen über die verschiedenen Compiler – Optionen und das Format der Kommentare.
- Abschnitt 4 beschreibt die Bedienung des in den HP464 integrierten Editors. Dieser Abschnitt sollte von allen Anwendern gelesen werden.
- Anhang 1 enthält die während der Übersetzungs- und Laufzeit auftretenden Fehlermeldungen.
- Anhang 2 listet die vordefinierten Namen und reservierten Wörter auf.
- Anhang 3 beschreibt die interne Datendarstellung von HiSoft Pascal.
- Anhang 4 zeigt einige Beispiel-Programme, die das Schreiben von HiSoft Pascal-Programmen erleichtern sollen.
- Anhang 5 behandelt die Schildkröten-Graphik.
- Anhang 6 beschreibt einige für die Anpassung der CPC464-Firmware erforderliche Pascal-Prozeduren und -Funktionen.

0.2 Übersetzen und Starten von Programmen

Detaillierte Ausführungen zum Erstellen, Ändern, Übersetzen und Starten von HP464-Programmen mit Hilfe des integrierten Editors befinden sich in Abschnitt 4 dieses Handbuchs.

Nach dem Aufruf des Compilers wird ein Listing mit folgenden Angaben erzeugt:

```
xxxx nnnn text of source line
```

wobei: **xxxx** die Anfangsadresse der diesen Code erzeugenden Zeile und
nnnn die Zeilennummer ohne führende Nullen ist.

Beinhaltet eine Zeile mehr als 80 Zeichen, so fügt der Compiler eine neue Zeile ein. Dadurch kann die Zeilenlänge nie mehr als 80 Zeichen betragen. Das Listing kann mit der Compiler-Option **SP** (siehe Abschnitt 3) auf dem Drucker ausgegeben werden.

Durch Drücken einer beliebigen Taste kann der Ausdruck jederzeit angehalten werden. [ESC] ermöglicht die Rückkehr zum Editor, jede andere Taste startet die Auflistung erneut.

Wird während der Übersetzung ein Fehler erkannt, wird die Meldung „*ERROR*“ und eine Fehlernummer (siehe Anhang 1) angezeigt.

Das den Fehler verursachende Symbol ist durch einen nachgestellten Aufwärtspfeil („↑“) kenntlich gemacht.

Zur Änderung der fehlerhaften Zeile kann entweder mit der Taste „E“ der Editor oder mit „P“ der Editor und die vorletzte Zeile (sofern vorhanden) bereitgestellt werden. Jede andere Taste setzt die Übersetzung fort.

Ist das Programm nicht korrekt abgeschlossen worden (z.B. ohne „END.“), erscheint die Meldung

„No more text“,

und die Steuerung wird an den Editor übergeben.

„No Table Space“

weist darauf hin, daß der Tabellenbereich des Compilers übergelaufen ist. Mit dem Alter-Kommando des nun bereitgestellten Editors kann die Tabellengröße neu zugewiesen werden.

Wenn die Übersetzung komplett durchläuft, dabei aber Fehler erkannt wurden, so wird die Anzahl der erkannten Fehler ausgegeben, der Object-Code gelöscht und der Editor bereitgestellt. Einer erfolgreichen fehlerlosen Übersetzung folgt die Meldung „RUN?“, woraufhin das Programm mit der Taste „Y“ oder „y“ sofort gestartet werden kann.

Bei der Ausführung des Object-Codes sind verschiedene Laufzeit-Fehlermeldungen (siehe Anhang 1) möglich.

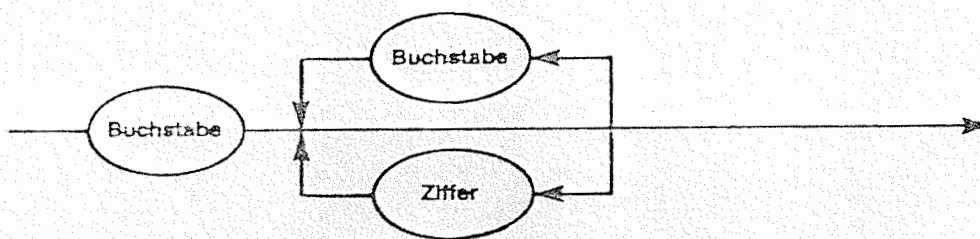
Durch Drücken einer beliebigen Taste kann der Programmablauf ausgesetzt werden. Nachfolgendes Betätigen der [ESC]-Taste führt zum Abbruch, jede andere Taste setzt den Ablauf fort.

ABSCHNITT 1

Syntax und Semantik

Dieser Abschnitt beschreibt Syntax und Semantik (Wortbedeutung) von HiSoft Pascal. Wenn nicht anders vermerkt, entspricht die Implementation den Spezifikationen des Pascal User Manual and Report Second Edition (Jensen/Wirth).

1.1 Bezeichner (Name, Identifier)

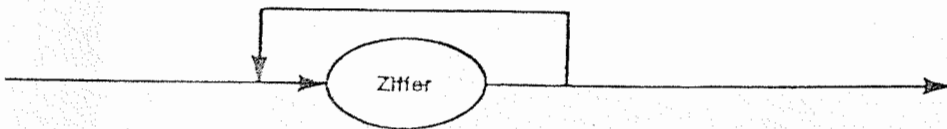


Es werden nur die ersten 10 Zeichen eines Bezeichners ausgewertet.

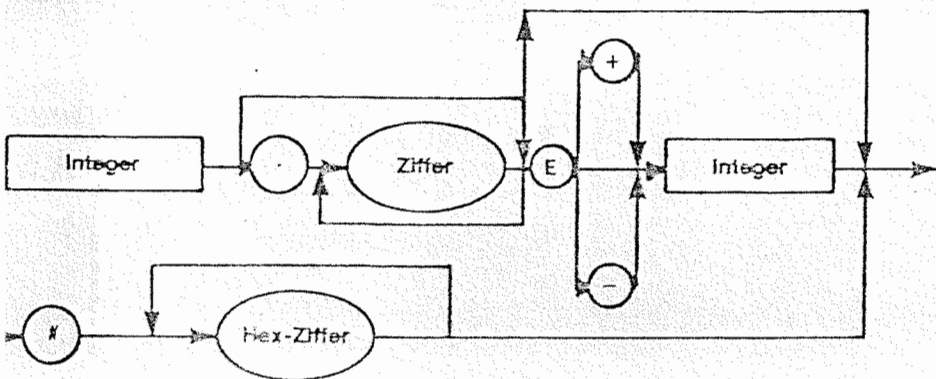
Bezeichner dürfen Groß- und Kleinbuchstaben enthalten, wobei Kleinbuchstaben intern in Großbuchstaben umgewandelt werden. Die Bezeichner HELLO, HELLo und he l l o sind demzufolge äquivalent.

Reservierte Wörter und vordefinierte Bezeichner dürfen ebenfalls Groß- und Kleinbuchstaben enthalten, wobei reservierte Wörter wiederum durch den Editor in Großbuchstaben umgewandelt werden.

.2 Vorzeichenlose ganze Zahlen (Integer)



.3 Vorzeichenlose reelle Zahlen (Real)



Zeichen haben in HiSoft Pascal einen Wertebereich kleiner oder gleich 32767. Große ganze Zahlen werden als Reelle Zahlen dargestellt.

Mantisse einer Reellen Zahl ist 23 Bit lang, wobei die Reelle Zahl auf 7 höchstwertigen Stellen genau ist. Beachten Sie jedoch, daß eine Berechnung ungenauer wird, wenn das Ergebnis sehr viel kleiner als die absoluten Werte der Argumente ist. So führt beispielsweise die Berechnung

1002-2

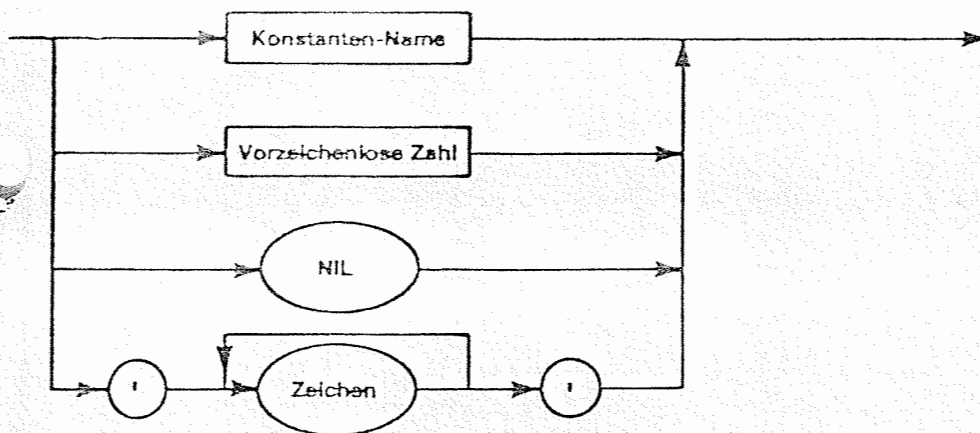
ist exakt zum Ergebnis

1002.

Das End-of-line-Zeichen (CHR(13)) darf in literalen Zeichenketten nicht enthalten sein, da andernfalls die Fehlermeldung „*ERROR* 68“ ausgegeben wird.

Ansonsten kann der volle erweiterte ASCII-Zeichensatz mit seinen 256 Elementen verwendet werden. Um die Kompatibilität mit Standard-Pascal zu gewährleisten, sollte das Zeichen Null nicht durch „' '“ sondern durch CHR(0) dargestellt werden.

1.5 Konstante



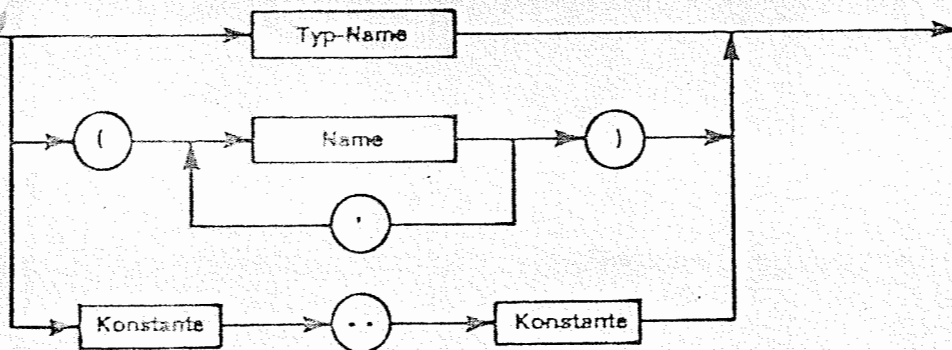
Das in Standard-Pascal nicht implementierte CHR dient zur Darstellung von Steuerzeichen durch Konstanten. Die eingeklammerte Konstante muß vom Typ Integer sein.

Beispiel:

```

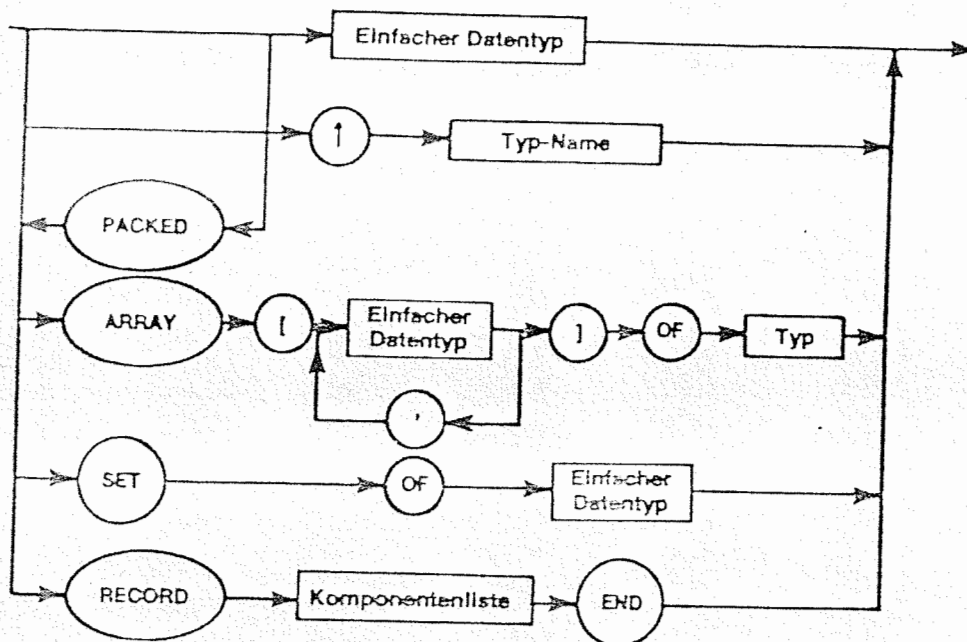
CONST  bs=CHR(8);      )
        cr=CHR(13);   )
  
```

1.6 Einfache Datentypen



Einfache (skalare) Datentypen (Name, Name,) dürfen nicht mehr als 256 Elemente enthalten.

1.7 Strukturierte Datentypen



Das reservierte Wort **PACKED** ist gültig, wird jedoch ignoriert, da Arrays von Zeichen etc. bereits gepackt werden. Der einzige Anwendungsfall, für den ein gepacktes Array Vorteile verspricht, ist ein Array vom Typ Boolean (obgleich hierfür der strukturierte Datentyp Menge günstigere Darstellungsmöglichkeiten bietet).

1.7.1 ARRAY und MENGE

Der Grundtyp (Basistyp) einer Menge (englisch Set) kann bis zu 256 Elemente enthalten, wodurch Mengen vom Typ CHAR (SET OF CHAR) ebenso wie Mengen von jedem durch den Benutzer spezifizierten Typ deklariert werden können. Beachten Sie jedoch, daß nur ein Unterbereich (Subrange) der Integer-Zahlen als Grundtyp verwendet werden kann. Die Untermenge der Integer-Zahlen hat einen Wertebereich von 0 bis 255.

Arrays vom Typ Array, Arrays vom Typ Menge und Records (Verbund) vom Typ Menge werden unterstützt.

Zwei Arrays sind nur dann äquivalent, wenn ihre Definitionen das reservierte Wort **ARRAY** gleichartig anwenden. Die folgenden Typen sind beispielsweise nicht äquivalent:

TYPE

```

tabellea=ARRAY[1..100] OF INTEGER;
tabelleb=ARRAY[1..100] OF INTEGER;
    
```

Eine Variable vom Typ `tabellea` kann beispielsweise nie einer Variablen vom Typ `tabelleb` zugewiesen werden. Hierdurch sind Fehler wie z.B. die Zuweisung von Tabellen unterschiedlichen Datentyps erkennbar. Die obige Einschränkung gilt nicht für Arrays vom Typ `String`, da sie immer Daten des gleichen Typs enthalten.

1.7.2 Zeiger

Mit der Standard-Prozedur `NEW` (siehe Abschnitt 2) kann eine dynamische Variable erzeugt werden. Im Gegensatz zu den statischen Variablen, die im Vereinbarungsteil des Programms definiert werden und einen festen Speicherplatz zugewiesen bekommen, kann auf dynamische Variablen nur über eine Zeigervariable und nicht direkt über einen Namen (dynamische Variablen bekommen keinen Namen) zugegriffen werden. Diese Zeigervariable ist eine statische Variable, der die Adresse der dynamischen Variablen zugewiesen wird. Damit „zeigt“ die Zeigervariable auf die eigentliche dynamische Variable. Für den Zugriff auf die dynamische Variable muß der Zeigervariablen ein Aufwärtspfeil „↑“ nachgestellt werden. Beispiele zur Anwendung der Zeigervariablen befinden sich im Anhang 4.

Für HiSoft Pascal gelten folgende Einschränkungen:

- Zeiger auf nicht deklarierte Datentypen sind unzulässig. Der Aufbau von verketteten Listen wird jedoch nicht eingeschränkt, da Typ-Definitionen auch Zeiger auf sich selbst enthalten dürfen.

Beispiel:

```
TYPE
  item=RECORD
  wert:INTEGER;
  naechsten:↑item
END;
Kette=↑item;
```

- Zeiger, die auf Zeiger weisen, sind unzulässig.
- Zeiger, die auf Daten gleichen Typs weisen, sind äquivalent.

Beispiel:

```
VAR
  erste:Kette;
  weitere:↑item;
```

Die Variablen `erste` und `weitere` sind äquivalent (d.h. strukturgleich) und können einander zugewiesen oder miteinander verglichen werden.

- Die vordefinierte Konstante `NIL` wird unterstützt. Eine Zeigervariable, der die Konstante `NIL` zugewiesen wurde, enthält keine Adresse.

1.7.4 Record

Die Implementierung von Records, also strukturierten Variablen mit einer konstanten Anzahl von Elementen (Komponenten), entspricht Standard Pascal. Die einzige Ausnahme besteht darin, daß HiSoft Pascal keinen Variantenteil in der Komponentenliste erlaubt.

Zwei Records sind nur dann äquivalent, wenn ihre Deklarationen das reservierte Wort RECORD gleichartig anwenden (siehe Abschnitt 1.7.1). In zwei unterschiedlichen RECORD-Definitionen sollten nie die gleichen Komponenten-Namen verwendet werden.

Beispiel:

Nach der Deklaration des RECORDs

```
rec1 = RECORD  
  f1 : integer  
END;
```

sollte der RECORD

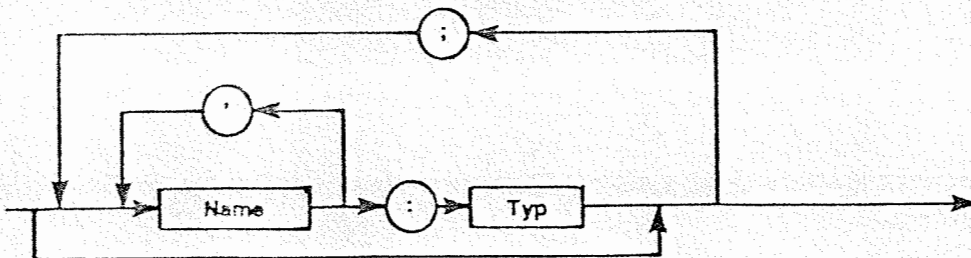
```
rec2 = RECORD  
  f1 : integer  
END;
```

nicht verwendet werden, sondern stattdessen

```
rec2 = RECORD  
  f2 : integer  
END;
```

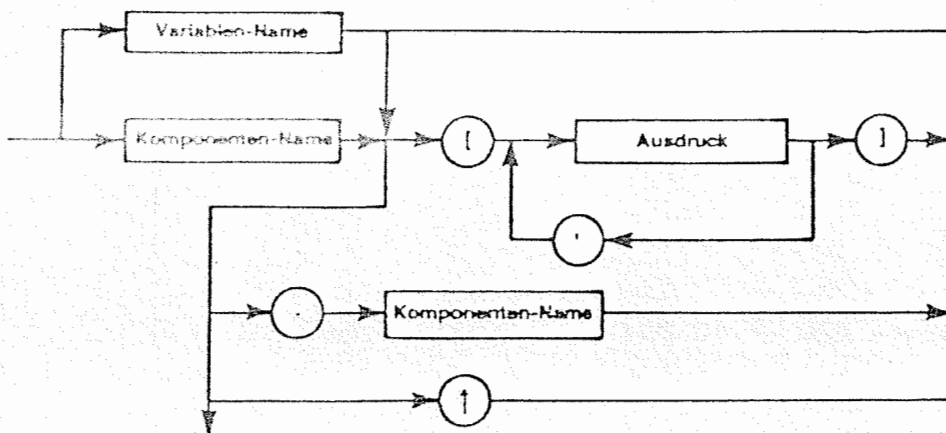
Die WITH-Anweisung dient dem schnellen, einfachen Zugriff auf die einzelnen Komponenten eines Records und ist nicht rekursiv anwendbar. Anhang 4 zeigt ein allgemeines Beispiel für die Anwendung von WITH und RECORD.

1.8 Komponentenliste



Komponentenlisten werden im Zusammenhang mit RECORDs verwendet (siehe Abschnitt 1.7.4 und Anhang 4).

1.9 Variablen



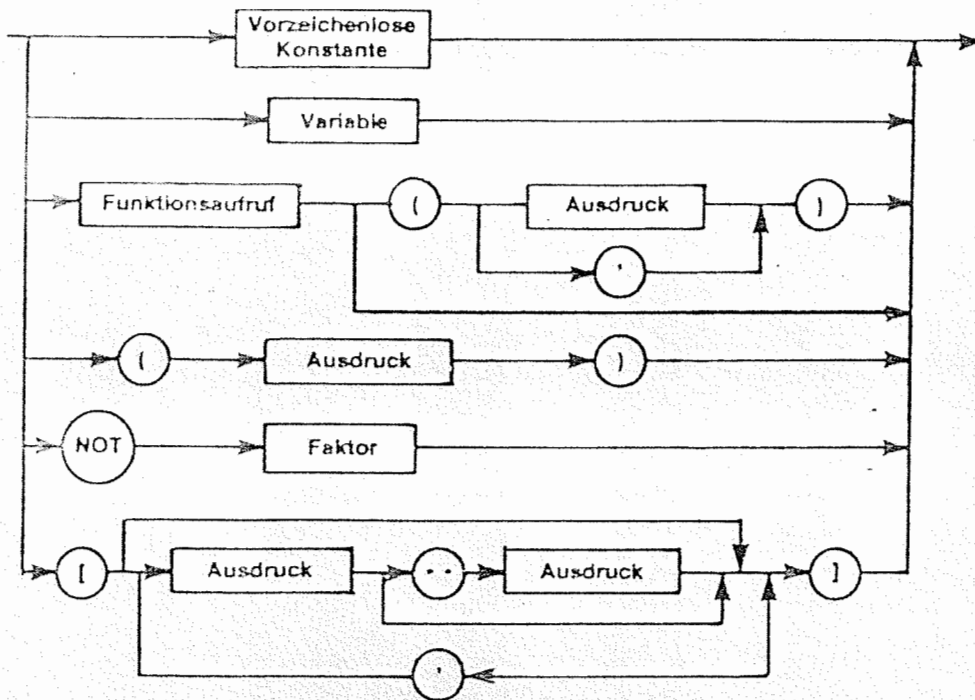
HiSoft Pascal unterstützt statische und dynamische Variablen. Statische Variablen werden im Deklarationsteil des Programms explizit durch **VAR** definiert und bekommen beim Übersetzen einen festen Speicherplatz zugewiesen.

Dynamische Variablen werden erst beim Programmablauf durch die Prozedur **NEW** erzeugt. Sie werden nicht explizit definiert und sind nicht durch einen Namen ansprechbar. Auf dynamische Variablen kann nur indirekt über eine statische Variable vom Typ Zeiger zugegriffen werden, die deren Speicheradresse zugewiesen bekommt (siehe Abschnitt 1.7.2 und 2 sowie Anhang 7).

Der Programmierer ist bei der Spezifikation von Elementen eines mehrdimensionalen Arrays nicht daran gebunden, den im Deklarationsteil definierten Indextyp für den Zugriff auf dieses Array zu verwenden.

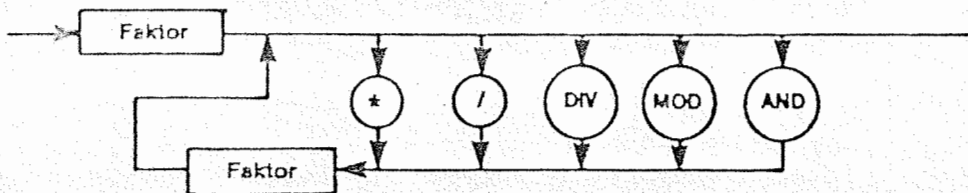
Wenn beispielsweise eine Variable **a** als **ARRAY[1..10] OF INTEGER** deklariert ist, kann ein Zugriff auf das Array-Element (1,1) sowohl durch **[1][1]** als auch durch **[1,1]** erfolgen.

1.10 Faktor



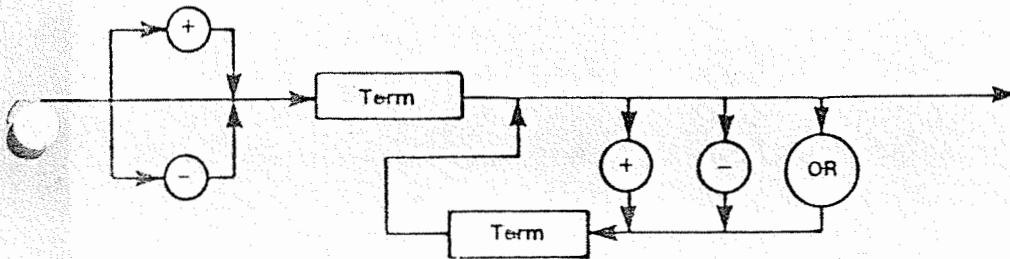
Weitere Einzelheiten sind in Abschnitt 1.13 (AUSDRUCK) und 3 (FUNKTIONEN) enthalten.

1.11 Term



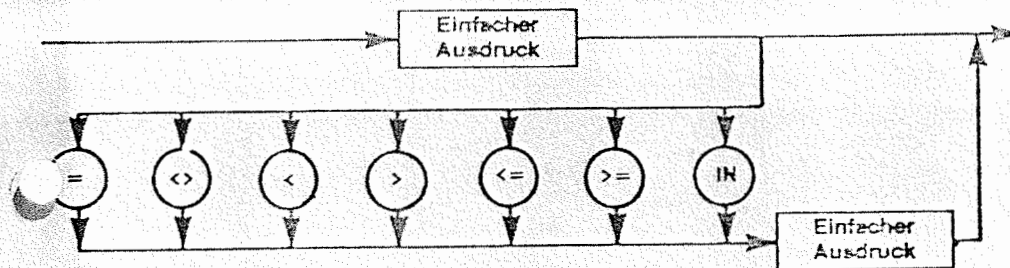
Die kleinste Element-Anzahl einer Menge ist immer 0, während die maximale Element-Anzahl einer Menge immer die maximale Element-Anzahl des Basistyps dieser Menge ist. Daraus folgt beispielsweise, daß die Menge SET OF CHAR immer 32 Bytes belegt (maximal 256 Elemente mit einem Bit pro Element). Ähnliches gilt für die Menge SET OF 0..10, die zur Menge SET OF 0..255 äquivalent ist.

1.12 Einfacher Ausdruck



Das unter Abschnitt 1.11 Ausgeführte gilt ebenfalls für einfache Ausdrücke.

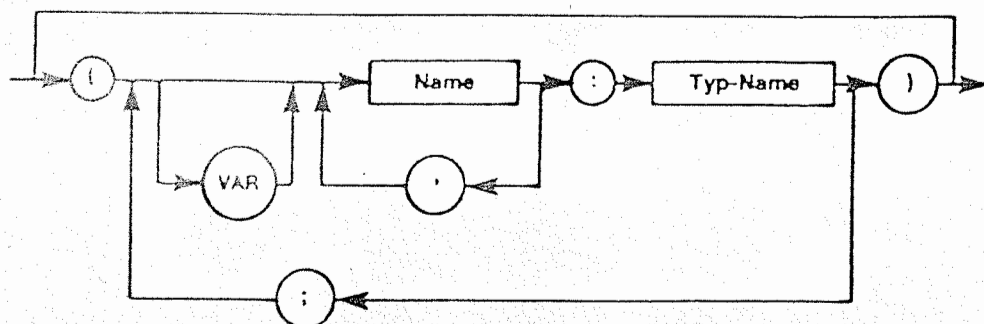
1.13 Ausdruck



Der IN-Operator kann auf Mengen angewendet werden, deren Elemente aus dem vollen Bereich des Typs „Einfacher Ausdruck“ entstammen können. Eine Ausnahme besteht nur für Integer-Argumente, deren Werte auf den Unterbereich [0..255] festgelegt sind.

Die obige Syntax gilt für den Vergleich von Strings gleicher Länge, Zeigern und einfachen (skalaren) Datentypen. Mengen können nur mit den Operatoren >=, <=, <> oder =, Zeiger nur mit den Operatoren = und <> verglichen werden.

1.14 Parameterliste



Dem Doppelpunkt nach Name muß ein Typ-Name folgen, da andernfalls *ERROR* 44 ausgegeben wird. HiSoft Pascal unterstützt sowohl Variablen-Parameter als auch Werte-Parameter. Prozeduren und Funktionen sind als Parameter nicht zugelassen.

1.15 Anweisung

Die Syntaxdiagramme der hier aufgeführten Anweisungen finden Sie auf den folgenden Seiten.

Wertzuweisung:

Abschnitt 1.7 enthält die zulässigen Wertzuweisungen.

CASE-Anweisung:

Eine leere CASE-Anweisung ist unzulässig, d.h. CASE OF END erzeugt eine Fehlermeldung (*ERROR* 13).

Der ELSE-Zweig (bei Anweisungen ohne END) wird ausgeführt, wenn der Selektor mit keiner Markierung der CASE-Anweisung übereinstimmt.

Bei CASE-Anweisungen mit END wird die Anweisung nach END ausgeführt, wenn der Selektor keine dazugehörige Markierung findet.

FOR-Anweisung:

Die Steuervariable einer FOR-Anweisung darf keine strukturierte Variable und kein Parameter sein. Diese Definition ist ein Kompromiß zwischen den Spezifikationen von Jensen/Wirth und dem ISO-Standard.

GOTO-Anweisung:

Ein Sprung mit der Anweisung GOTO ist nur auf eine Marke (Label) zulässig, die sich im gleichen Programmblock wie die GOTO-Anweisung, und auf gleicher Ebene befindet.

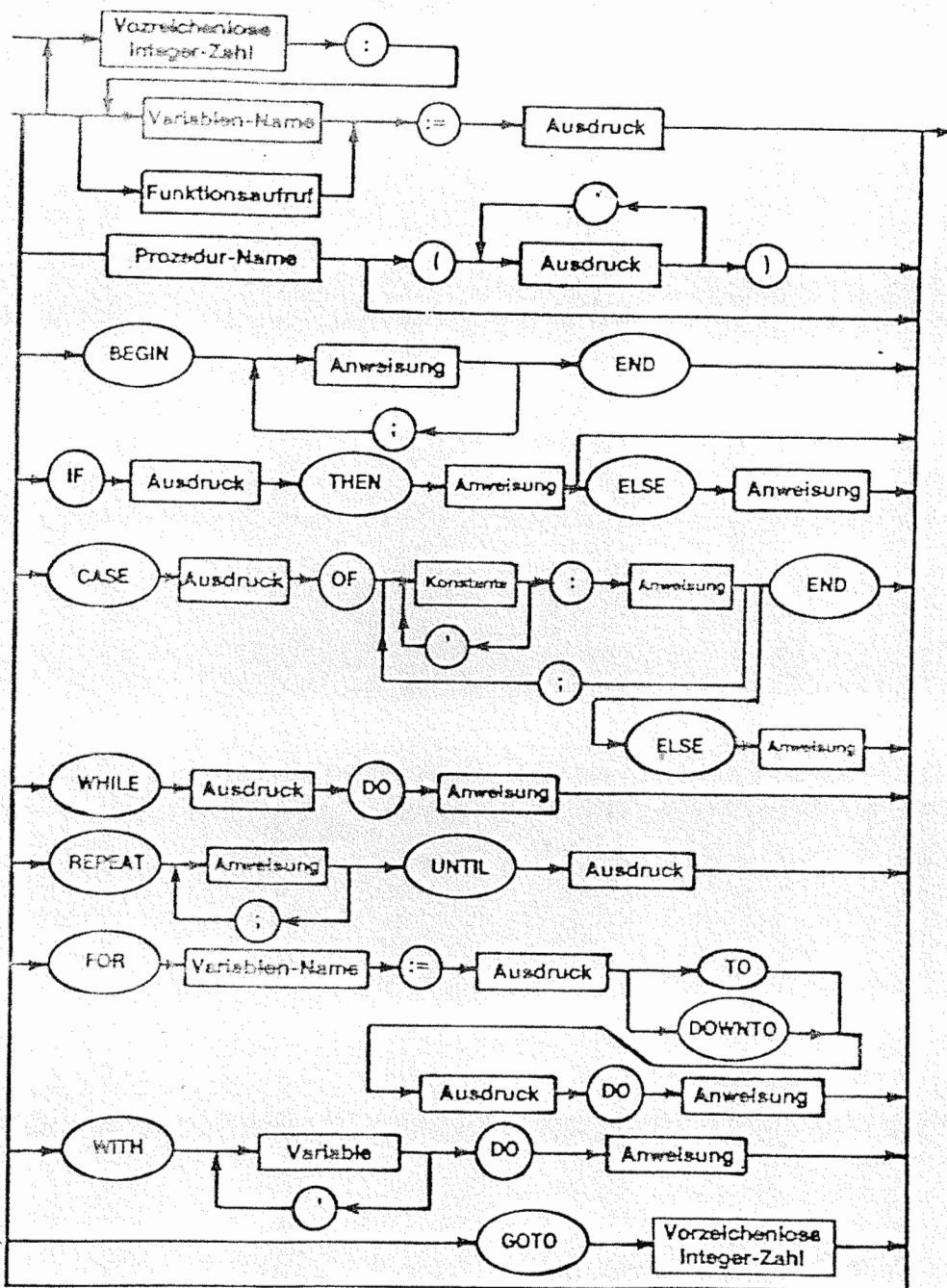
Marken müssen in dem entsprechenden Programmblock mit dem reservierten Wort LABEL deklariert werden.

Eine Marke darf maximal 4 Zeichen, gefolgt von einem Doppelpunkt ":", enthalten und muß am Anfang einer Anweisung stehen. Eine GOTO-Anweisung darf weder zum Verlassen einer FOR...DO-Schleife, noch zum Ausprung aus einer Prozedur oder Funktion verwendet werden.

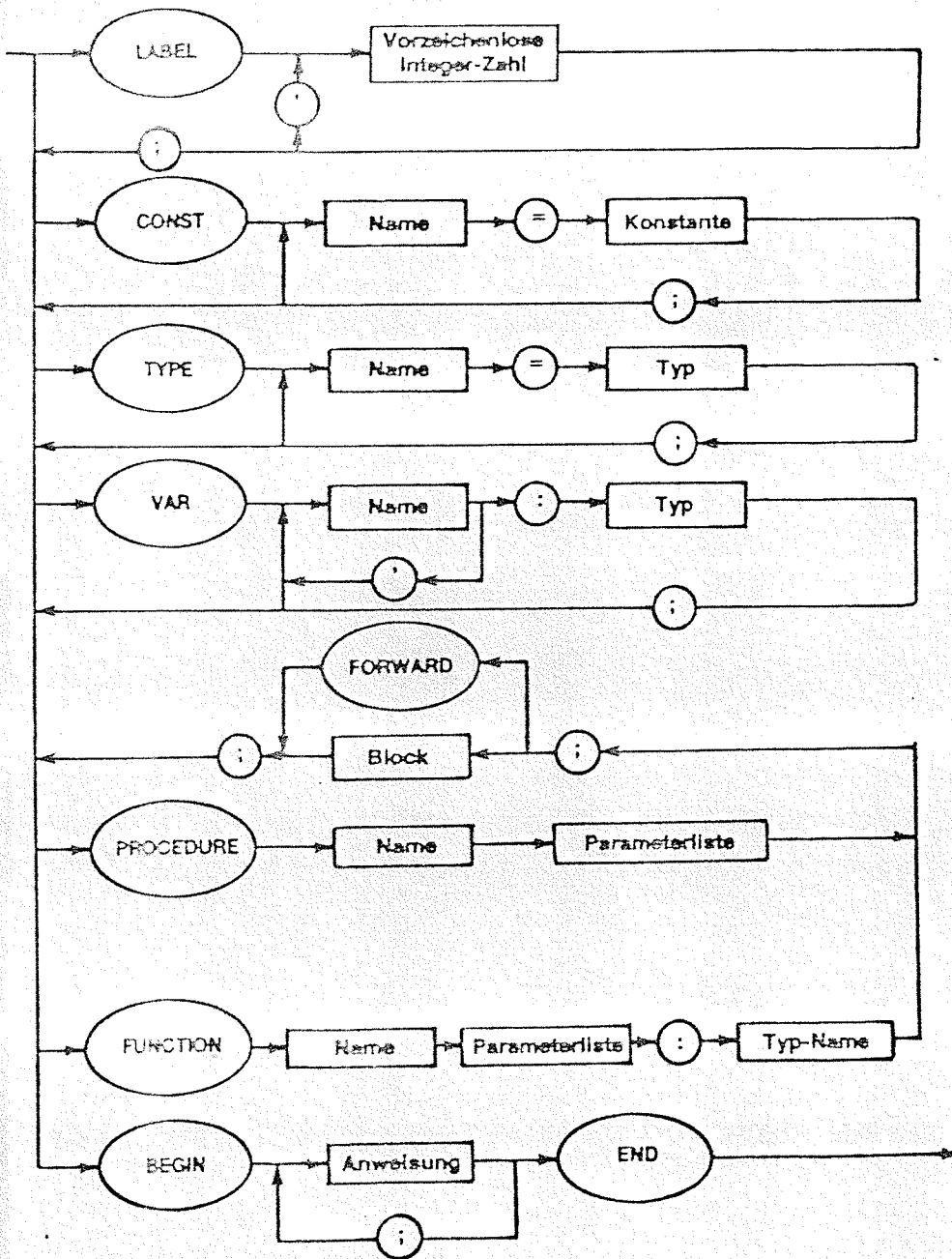
WITH-Anweisung:

WITH-Anweisungen dürfen nicht rekursiv angewendet werden (siehe Abschnitt 1.7.4).

Anweisung:



1.16 Block



Vorwärts-Referenzen

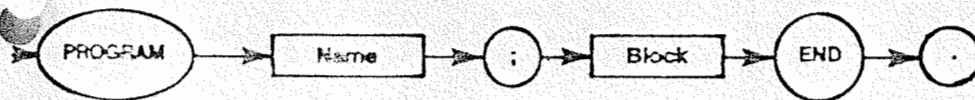
Auf Prozeduren und Funktionen kann mit dem reservierten Wort `FORWARD` zugegriffen werden, bevor sie deklariert sind.

Beispiel:

```
PROCEDURE a(y:t) ; FORWARD;    (*Prozedur a wird als*)
PROCEDURE b(x:t);              (*Vorwärts-Referenz deklariert*)
BEGIN
  ....
  a(p);                         (*Aufruf der Prozedur a*)
  ....
END;
PROCEDURE a;                   (*Tatsächliche Deklaration der*)
BEGIN                           (*Prozedur a*)
  ...
  (q);
  ....
END;
```

Die Parameter und der Ergebnis-Typ der Prozedur `a` sind in diesem Beispiel bereits in der Vorwärts-Referenz (`FORWARD`) deklariert und brauchen daher nicht noch einmal im Deklarationsteil der Prozedur definiert werden.

1.17 Programm



Formale Programm-Parameter (z.B. `INPUT`, `OUTPUT`) sind nicht erforderlich, weil der Datentyp `Datei` in HiSoft Pascal nicht implementiert ist.

1.18 Typgebundenheit

Moderne Programmiersprachen bedienen sich verschiedener Verfahren, um Anwender vor der nicht definitionsgemäßen Handhabung der Daten zu schützen.

Auf unterster Ebene, der Maschinensprache, wird der Typ (TYPE) einer Variablen generell nicht überprüft. Auf nächst höherer Ebene folgt BASIC, die zwischen Zahlen und Strings, teilweise auch zwischen Integer und Real (Integer wird häufig durch das Zeichen „%“ gekennzeichnet) differenziert. Pascal wiederum läßt sogar vom Anwender selbst definierbare Datentypen (Aufzählungstyp) zu. Auf höchster Ebene (jedenfalls momentan) befindet sich die Programmiersprache ADA, in der verschiedene, inkompatible Datentypen definierbar sind.

Die strenge Typgebundenheit der Programmiersprache Pascal beruht auf zwei Hauptmerkmalen, der Struktur- und Namens-Äquivalenz. HiSoft Pascal bedient sich der Namens-Äquivalenz bei RECORDs und ARRAYs (siehe Anhang 1). Nehmen wir beispielsweise einmal an, daß zwei Variablen wie folgt definiert sind:

```
VAR A : ARRAY['A'..'C'] OF INTEGER;  
    B : ARRAY['A'..'C'] OF INTEGER;
```

Aufgrund dieser Definitionen könnte man annehmen, die Zuweisung $A := B$; sei zulässig. Stattdessen wird unter HiSoft Pascal die Fehlermeldung (*ERROR* 10) erzeugt, da durch die obigen Definitionen zwei RECORDs unterschiedlichen Typs geschaffen worden sind. Der Anwender hat also durch diese Definitionen nicht zum Ausdruck gebracht, daß A und B den gleichen Datentyp repräsentieren sollen. Die Definition

```
VAR A,B: ARRAY['A'..'C'] OF INTEGER;
```

schafft zwei RECORDs gleichen Typs und ermöglicht damit eine freie Zuordnung von A und B.

Dieses Namens-Äquivalenz sieht oberflächlich betrachtet etwas kompliziert aus, verhindert jedoch Programmierfehler, da sie den Programmierer zu gründlichen Vorüberlegungen zwingt.

ABSCHNITT 2

Vordefinierte Namen

2.1 Konstanten

MAXINT Größte Integer-Zahl (32767).
FALSE, TRUE Konstanten vom Typ Boolean.

2.2 Datentypen

INTEGER siehe Abschnitt 1.3.
REAL siehe Abschnitt 1.3.
CHAR Vollständiger ASCII-Zeichensatz mit 256 Elementen.
BOOLEAN (**FALSE, TRUE**) Datentyp für logische Operationen (einschließlich Vergleich).

2.3 Prozeduren und Funktionen

2.3.1 Eingabe- und Ausgabe-Prozeduren

2.3.1.1 WRITE

Mit der Prozedur **WRITE** können Daten auf dem Datensichtgerät oder Drucker ausgegeben werden. Wenn der zu schreibende Ausdruck vom Typ **CHAR** ist, übergibt **WRITE (e)** den durch **e** repräsentierten 8-Bit-Wert an den Bildschirm oder Drucker.

Hinweis:

CHR(8) (**[CTRL]H**) gibt einen löschenden Rückwärtsschritt auf dem Bildschirm aus.
CHR(12) (**[CTRL]L**) löscht den Bildschirm oder bewirkt einen Seitenvorschub auf dem Drucker.
CHR(13) (**[CTRL]M**) bewirkt einen Wagenrücklauf/Zeilenvorschub.
CHR(16) (**[CTRL]P**) bewirkt eine Ausgabe an den Drucker, wenn der Bildschirm aktiv ist, und umgekehrt.

Weiterhin gilt:

```
WRITE (P1, P2, ..., Pn); ist äquivalent zu  
BEGINWRITE (P1); WRITE (P2); .....; WRITE (Pn) END;
```

Die WRITE-Parameter P1, P2, ..., Pn können eines der folgenden Formate haben:

<e> oder <e:m> oder <e:m:n> oder <e:m:H>

wobei e, m und n Ausdrücke und H eine literale Konstante ist.

Es sind 5 Fälle zu unterscheiden:

1) e ist vom Typ Integer und das Format <e> oder <e:m>.

Der Wert des Integer-Ausdrucks e wird in einen Zeichenstring mit vorgestelltem Leerzeichen umgewandelt. Die Länge des Strings läßt sich durch vorgestellte Leerzeichen vergrößern, indem mit m die insgesamt auszugebende Zeichenanzahl festgelegt wird. Wenn m wirkungslos bzw. nicht spezifiziert ist, wird e vollständig mit nur einem führenden Leerzeichen ausgegeben und m ignoriert. Ist m auf die Länge von e-1 (ohne führendes Leerzeichen) gesetzt, wird auch kein vorgestelltes Leerzeichen ausgegeben.

2) e ist vom Typ Integer und das Format <e:m:H>.

In diesem Fall wird e hexadezimal ausgegeben. Ist m=1 oder m=2, so wird der Wert $(e \text{ MOD } 16 \uparrow m)$ mit exakt m Zeichen ausgegeben. Bei m=3 oder m=4 erfolgt die hexadezimale Ausgabe von e mit 4 Zeichen. Ist m > 4, werden gegebenenfalls führende Leerzeichen vor dem vollständigen hexadezimalen Wert von e eingefügt. Führende Nullen werden – sofern anwendbar – ebenfalls eingefügt.

Beispiele:

```
WRITE(1025:m:H);
```

Bei m=1 erfolgt die Ausgabe: 1

Bei m=2 erfolgt die Ausgabe: 01

Bei m=3 erfolgt die Ausgabe: 0401

Bei m=4 erfolgt die Ausgabe: 0401

Bei m=5 erfolgt die Ausgabe: _0401

3) e ist vom Typ Real und das Format <e>, <e:m> oder <e:m:n>.

Der Wert von e wird in einen Zeichenstring, der eine reelle Zahl darstellt, umgewandelt. Das Format der Darstellung wird durch n bestimmt.

Bei fehlendem n wird die Zahl in der Exponentialform mit Mantisse und Exponent dargestellt. Eine negative Zahl erscheint mit einem Minuszeichen vor der Mantisse, eine positive mit einem vorgestellten Leerzeichen. Die Zahl hat minimal eine und maximal 5 Dezimalstellen, wobei der Exponent immer vorzeichenbehaftet ist. Die Exponentialzahl besteht aus minimal 8 Zeichen. Wenn die Feldlänge m kleiner als 8 ist, wird immer die volle Länge von 12 Zeichen ausgegeben. Ist m >= 8, so werden eine bis maximal 5 Dezimalstellen (m=12) dargestellt. Bei m > 12 werden führende Leerzeichen eingefügt.

Beispiele:

```
WRITE (-1.23E10:m);
```

m=7	ergibt:	-1.23000E+10
m=8	ergibt:	-1.2E+10
m=9	ergibt:	-1.23E+10
m=10	ergibt:	-1.230E+10
m=11	ergibt:	-1.2300E+10
m=12	ergibt:	-1.23000E+10
m=13	ergibt:	-1.23000E+10

Mit dem Format $\langle e : m : n \rangle$ kann eine Festpunktdarstellung der Zahl e geschrieben werden, wobei n die auszugebenden Dezimalstellen bestimmt. Nur wenn die Feldlänge m entsprechend groß ist, werden führende Leerzeichen ausgegeben. e wird als Integer-Zahl ausgegeben, wenn $n=0$ ist. Sollte e größer als die maximal mögliche Feldlänge sein, so wird die Exponentialform mit der Feldlänge m (siehe oben) gewählt.

Beispiele:

WRITE(1E2:6:2)	ergibt:	_00.00
WRITE(1E2:8:2)	ergibt:	_ _100.00
WRITE(23.455:6:1)	ergibt:	_ _23.5
WRITE(23.455:4:2)	ergibt:	_2.34550E+01
WRITE(23.455:4:0)	ergibt:	_ _23

4) e ist vom Typ CHAR oder String.

Zur Ausgabe von Zeichen mit einer Feldfolge von 1 bzw. Zeichenketten mit der entsprechenden String-Länge können entweder $\langle e \rangle$ oder $\langle e : m \rangle$ verwendet werden. Bei größeren m werden führende Leerzeichen eingefügt.

5) e ist vom Typ Boolean.

In Abhängigkeit von e wird mit $\langle e \rangle$ oder $\langle e : m \rangle$ der Wert „TRUE“ bzw. „FALSE“ ausgegeben. Die Feldlänge beträgt minimal 4 bzw. 5 Zeichen.

2.3.1.2 WRITELN

WRITELN erzeugt eine neue Zeile und ist äquivalent zu `WRITE (CHR (13))`.

Beispiel:

```
WRITELN (P1,P2,.... P3); ist äquivalent zu  
BEGINWRITE (P1,P2,.... P3); WRITELNEND;
```

2.3.1.3 PAGE

Die Prozedur PAGE ist äquivalent zu WRITE (CHR (12)) und löscht den Bildschirm oder bewirkt einen Seitenvorschub auf dem Drucker.

2.3.1.4 READ

Die Prozedur READ dient dem Zugriff auf Daten von der Tastatur. Der Zugriff erfolgt über einen zur Laufzeit bereitgestellten Puffer, der zu Beginn – mit Ausnahme der end-of-line-Marke – leer ist. Diesen Puffer kann man sich als Text-Fenster vorstellen, in dem jeweils ein Zeichen bereit steht. Wenn dieses Text-Fenster über eine end-of-line-Marke positioniert ist, wird vor Abschluß der Lese-Operation eine neue Text-Zeile von der Tastatur in den Puffer gelesen. Beim Einlesen dieser Zeile werden alle in Abschnitt 0 0 angeführten Steuer-codes erkannt.

Ziel:

READ (V1, V2, Vn); ist äquivalent zu
BEGINREAD (V1); READ (V2);; READ (Vn) END;

wobei V1, V2 usw. vom Typ CHAR, String, Integer oder Real sein kann.

Die Anweisung READ (V) zeigt in Abhängigkeit von V unterschiedliche Wirkungen. Es sind vier Fälle zu unterscheiden:

1) V ist vom Typ CHAR.

In diesem Fall liest READ (V) ein Zeichen vom Eingabe-Puffer und weist deren Wert der Variablen V zu. Wenn das Text-Fenster im Puffer auf eine Zeilen-Marke (CHR (13)) positioniert ist, übergibt die EOLN-Funktion den Wert TRUE und liest eine neue Text-Zeile von der Tastatur. Eine weitere Lese-Operation positioniert das Text-Fenster auf den neuen Zeilenanfang.

Wichtiger Hinweis: EOLN ist beim Programmstart vom Wert TRUE. Wenn der erste READ-Aufruf vom Typ CHAR ist, wird der Wert CHR (13) übergeben und eine neue Zeile von der Tastatur gelesen. Eine erneute Lese-Operation vom Typ CHAR liefert das erste Zeichen dieser neuen Zeile, sofern es kein Leerzeichen ist (siehe READLN).

V ist vom Typ String.

Mit READ ist auch eine Zeichenkette einlesbar. Es werden hierbei solange Zeichen eingelesen, bis die durch den String definierte Anzahl Zeichen geholt oder EOLN = TRUE ist. Wurde der String durch diese Operation nicht vollständig gelesen (z.B. weil end-of-line vor dem Ende des gesamten Strings auftrat), so erfolgt eine Auffüllung des verbleibenden String-Bereichs mit Nullen (CHR (0)), wodurch dem Programmierer die Bestimmung der Länge einer eingelesenen Zeichenkette ermöglicht wird.

Der unter 1) angeführte Hinweis gilt auch hier.

3) V ist vom Typ Integer.

In diesem Fall werden Zeichen eingelesen, die Integer-Zahlen darstellen (siehe Definition in Abschnitt 1.3). Alle vorgestellten Leerzeichen und end-of-line-Marken werden ausgeschlossen, d.h. Integer-Zahlen werden unmittelbar eingelesen (siehe Hinweis unter 1)).

Eine eingelesene Integer-Zahl, deren absoluter Wert größer **MAXINT** (32767) ist, erzeugt den Laufzeit-Fehler „**Number too Large**“ (Bereichsüberschreitung) und beendet die Ausführung. Ist das erste, nach dem Überspringen der Leerzeichen und end-of-line-Marken gelesene Zeichen keine Ziffer oder kein Vorzeichen („+“ oder „-“), so erscheint die Laufzeit-Fehlermeldung „**Number expected**“ (Zahl fehlt), und das Programm wird abgebrochen.

4) V ist vom Typ Real.

Hier werden Zeichen eingelesen, die eine reelle Zahl – entsprechend der Syntax in Abschnitt 1.3 – darstellen, wobei alle führenden Leerzeichen und end-of-line-Marken ausgelassen werden. Wie bei den Integer-Zahlen, muß auch hier das erste Zeichen eine Ziffer oder ein Vorzeichen sein. Eine zu kleine oder zu große gelesene Zahl (siehe Abschnitt 1.3) erzeugt die Fehlermeldung „**Overflow**“ (Bereichsüberschreitung). Folgt dem Zeichen „E“ kein Vorzeichen bzw. keine Ziffer, erscheint die Fehlermeldung „**Exponent expected**“ (Exponent fehlt). Fehlt einem Dezimalpunkt die nachgestellte Ziffer, so wird die Fehlermeldung „**Number expected**“ (Zahl fehlt) erzeugt.

Reelle Zahlen können ebenso wie Integer-Zahlen unmittelbar gelesen werden (siehe oben unter 1) und 3)).

2.3.1.5 READLN

Beispiel:

```
READLN(V1,V2,.....Vn); ist äquivalent zu  
BEGIN READ(V1,V2,.....Vn); READLN END;
```

READLN liest einen neuen Puffer von der Tastatur. Während der Eingabe in den Puffer können die in Abschnitt 0.0 beschriebenen Steuerfunktionen verwendet werden. Nach der Ausführung von READLN erhält EOLN den Wert FALSE, wenn die nächste Zeile keine Leerzeile ist.

READLN kann zum Überspringen der Leerzeile, die zu Beginn der Ausführung des Object-Codes ansteht, benutzt werden, was das Einlesen eines neuen Puffers zur Folge hat.

Dieses Vorgehen kann sehr nützlich sein, wenn am Programm-Anfang eine Komponente vom Typ CHAR eingelesen werden soll. Für das Lesen einer Integer- oder reellen Zahl bzw. von Zeichen aus aufeinanderfolgenden Zeilen ergeben sich hieraus keine Vorteile, da end-of-line-Marken übersprungen werden.

Im allgemeinen sollten nach dem Aufruf zur Eingabe (Ausgabe eines Prompts) eine READLN-Anweisung und entsprechende READ-Aufrufe folgen.

Beispiel für das Einlesen einer Variablen vom Typ CHAR mittels READ:

```
PROGRAM READ CHAR;
VAR CH:CHAR;
BEGIN
  REPEAT
    WRITE ('Gib Zeichen ein');
    READLN;
    WHILE NOT EOLN DO
      BEGIN
        READ (CH);
        WRITE ('Das 'CH,' entsprechende ASCII-Zeichen
              ist ',ORD(CH))
      END;
    UNTIL CH='E'
  END.
```

2.3.2 Eingabefunktionen

2.3.2.1 EOLN

Die boolesche Funktion EOLN liefert den Wert TRUE, wenn das nächste einzulesende Zeichen ein end-of-line-Zeichen (CHR(13)) ist. Jedes andere Zeichen übergibt den Wert FALSE.

2.3.2.2 INCH

Die Funktion INCH fragt die Tastatur nach einer gedrückten Taste ab und liefert das dieser Taste zugehörige Zeichen. Wurde keine Taste betätigt, übergibt sie das Zeichen CHR(0). In jedem Fall ist das Ergebnis vom Typ CHAR. Die Funktion sollte mit der C-Option des Compilers (siehe Abschnitt 3) verwendet werden.

3.3 Transferfunktionen

2.3.3.1 TRUNC(X)

Der Parameter X muß vom Typ Real oder Integer sein. Der durch TRUNC übergebene Wert ist die größte Integer-Zahl kleiner oder gleich X, wenn X positiv, oder die kleinste Integer-Zahl größer oder gleich X, wenn X negativ ist. TRUNC schneidet also die Stellen hinter dem Dezimalpunkt ab.

Beispiel:

`TRUNC(-1.5)` ergibt -1. `TRUNC(1.9)` ergibt 1

2.3.3.2 ROUND (X)

Diese Funktion rundet X auf die nächstgelegene Integer-Zahl (entsprechend den Rundungsregeln) auf oder ab, wobei X vom Typ Integer oder Real sein muß.

Beispiel:

`ROUND(-6.5)` ergibt -6 `ROUND(11.7)` ergibt 12
`ROUND(-6.51)` ergibt -7 `ROUND(23.5)` ergibt 24

2.3.3.3 ENTIER (X)

ENTIER liefert für alle X-Werte die größte Integer-Zahl kleiner oder gleich X, wobei X vom Typ Integer oder Real sein muß.

Beispiel:

`ENTIER(-6.5)` ergibt -7 `ENTIER(11.7)` ergibt 11

Hinweis: ENTIER ist keine Standard Pascal-Funktion, sondern ein Äquivalent zu INT unter BASIC. Sie dient dem Schreiben schneller mathematischer Routinen.

2.3.3.4 ORD (X)

Die Funktion ORD liefert die Ordnungsnummer des Wertes von X innerhalb der definierten Wertemenge, wobei X von jedem skalaren Typ – ausgenommen Real – sein kann.

Beachten Sie, daß `ORD(X) = X` ist, wenn X eine Integer-Zahl darstellt.

Beispiele:

`ORD('a')` ergibt 97 `ORD('a')` ergibt 64

2.3.3.5 CHR (X)

CHR liefert das dem Wert des Ausdrucks entsprechende ASCII-Zeichen.

Beispiele:

`CHR(49)` ergibt '1' `CHR(91)` ergibt '['

2.3.4 Arithmetische Funktion

Die Parameter X der folgenden Funktionen müssen vom Typ Real oder Integer sein.

2.3.4.1 ABS (X)

Berechnet den Absolutwert von X (z.B. $ABS(-4.5) = 4.5$). Das Ergebnis ist vom gleichen Typ wie X.

2.3.4.2 SQR (X)

Errechnet das Quadrat ($X * X$) von X. Das Ergebnis ist vom gleichen Typ wie X.

2.3.4.3 SQRT (X)

Berechnet die Quadratwurzel von X. Das Ergebnis ist immer vom Typ Real. Ein negatives Argument X erzeugt die Fehlermeldung „**Maths Call Error**“.

2.3.4.4 FRAC (X)

Berechnet die X-Fraction $FRAC(X) = X - ENTIER(X)$.

Diese Funktion dient (wie **ENTIER**) dem Erstellen schneller mathematischer Routinen.

Beispiele:

$FRAC(1.5) = 0.5$ $FRAC(-12.56) = 0.44$

2.3.4.5 SIN (X)

Berechnet den Sinus von X (X im Bogenmaß). Das Ergebnis ist immer vom Typ Real.

2.3.4.6 COS (X)

Berechnet den Cosinus von X (X im Bogenmaß). Das Ergebnis ist vom Typ Real.

2.3.4.7 TAN (X)

Berechnet den Tangens von X (X im Bogenmaß). Das Ergebnis ist immer vom Typ Real.

2.3.4.8 ARCTAN (X)

Berechnet den Winkel (im Bogenmaß), dessen Tangens den Wert X ergibt. Das Ergebnis ist vom Typ Real.

2.3.4.9 EXP (X)

Berechnet den Wert $e^{\uparrow}X$, mit $e=2.71828$. Das Ergebnis ist immer vom Typ Real.

2.3.4.10 LN (X)

Berechnet den natürlichen Logarithmus (zur Basis e) von X . Das Ergebnis ist vom Typ Real. Ein Wert $X < 0$ erzeugt die Fehlermeldung „Maths Call Error“.

2.3.5 Weitere Vordefinierte Prozeduren

2.3.5.1 NEW (p)

Die Prozedur **NEW (p)** weist einer dynamischen Variablen Speicherplatz zu. Die Variable p ist eine Zeigervariable. Nach der Ausführung von **NEW (p)** enthält p die Adresse der neu zugewiesenen dynamischen Variablen. Die dynamische Variable und die Zeigervariable p sind vom gleichen Typ, der wiederum frei wählbar ist.

Die Zeigervariable p^{\uparrow} dient dem Zugriff auf die dynamische Variable (siehe Beispiel in Anhang 4).

Die Neuzuweisung von Speicherplatz für eine dynamische Variable kann mit den Prozeduren **MARK** und **RELEASE** (siehe unten) erfolgen.

2.3.5.2 MARK (v1)

Diese Prozedur sichert dem Status der „Halde“ von dynamischen Variablen in der Zeigervariablen $v1$.

Die Halde kann auf den Status vor Aufruf der Prozedur **MARK** zurückgesetzt werden, indem die Prozedur **RELEASE** aufgerufen wird.

Der Typ der Variablen, auf die $v1$ zeigt, ist unbedeutend, da $v1$ nur mit **MARK** oder **RELEASE**, jedoch nie mit **NEW**, verwendet wird.

In Anhang 4 befindet sich ein Beispiel-Programm zu **MARK** und **RELEASE**.

2.3.5.3 RELEASE (v1)

Diese Prozedur schafft Platz auf der Halde für die Verwendung von dynamischen Variablen. Der Status der Halde wird auf den Status vor Aufruf von **MARK (v1)** zurückgesetzt. Hierdurch werden alle dynamischen Variablen zerstört, die seit dem Aufruf der **MARK** Prozedur erzeugt wurden. Diese Prozedur sollte daher mit großer Vorsicht verwendet werden.

(Weitere Informationen im Abschnitt 4).

2.3.5.4 INLINE (C1, C2, C3,....)

Mit dieser Prozedur kann Z80 – Maschinencode in ein Pascal-Programm eingebunden werden. Die Werte (C1 MOD 256, C2 MOD 256, C3 MOD 256) werden beginnend mit dem Speicherplatz, der durch den vom Compiler verwalteten Adresszähler adressiert wird, in den Object-Code eingefügt. Die Anzahl der Integer-Konstanten C1, C2, C3 usw. ist beliebig.

Anhang 4 zeigt ein Beispiel zur Anwendung von INLINE.

2.3.5.5 USER (V)

Die Prozedur USER bewirkt einen Aufruf der durch das Integer-Argument V gegebenen Speicheradresse. Da HiSoft Pascal Integer-Zahlen im Zweierkomplement verarbeitet (siehe Anhang 3), müssen Adressen größer als *7FFF (32767) durch negative V-Werte dargestellt werden. Somit ist *9000 gleich -28672. USER (-28762); ruft demzufolge die Speicheradresse *C000 auf. Die Angabe einer Speicheradresse sollte hexadezimal erfolgen.

Die aufgerufene Routine muß mit einem Z80 RET-Befehl (*C9) abgeschlossen werden und darf das IX-Register nicht überschreiben. Die Register A,B,C,D,E,H,L und F werden vor dem Aufruf der Routine mit den Werten RA, RB, RC, RD, RE, RH, RL und RF belegt. Die übergebenen Register-Werte der Routine sind durch die oben angeführten Variablen (siehe Abschnitt 2.4.2) gegeben.

2.3.5.6 HALT

Die Prozedur HALT stoppt die Programm-Ausführung und erzeugt die Meldung „Halt at PC=XXXX“, wobei XXXX die hexadezimale Abbruch-Speicheradresse ist. HALT kann – gemeinsam mit einem Compiler-Listing – verwendet werden, um beim Debuggen den Programm-Ablauf bei Verzweigungen zu verfolgen.

2.3.5.7 POKE (X,V)

POKE speichert den Ausdruck V unter der Adresse X im Speicher. X ist vom Typ Integer, V kann beliebigen Typs – außer Menge – sein (Abschnitt 2.3.5.5 enthält eine Beschreibung der Darstellung von Speicheradressen durch Integer-Zahlen).

POKE(#6000, 'A') legt #41 unter Adresse #6000 ab.
POKE(-28672, 3.6E3) legt 000B8070 unter Adresse #9000 ab.

2.3.5.8 TOUT (NAME, START, SIZE)

Mit der Prozedur TOUT können Variablen auf der Cassette gesichert werden. Der erste Parameter ist vom Typ ARRAY[1..12] OF CHAR und stellt den Namen der zu sichernden Datei dar.

SIZE ist die Anzahl der ab Adresse START auszugebenden Bytes. Beide Parameter sind vom Typ Integer.

Beispiel:

```
TOUT('VAR _____', ADDR(V), SIZE(V))
```

sichert die Variable V unter dem Namen VAR auf der Cassette.

Durch die Spezifikation von Speicheradressen können nicht nur Arrays, sondern auch globale Variablen in einer Datei gesichert werden. Anhang 4 enthält ein Beispiel über die Anwendung von TOUT.

2.3.5.9 TIN (NAME, START)

Diese Prozedur lädt Variablen etc., die vormals mit TOUT gesichert wurden, von der Cassette. Der Parameter NAME ist vom Typ ARRAY[1..12] OF CHAR und START vom Typ Integer. Die Cassette wird nach der Datei mit dem Namen NAME durchsucht, welche dann unter Adresse START gespeichert wird. Die Anzahl der zu ladenden Bytes ist (durch TOUT) auf der Cassette gespeichert.

Beispiel:

```
TIN('VAR _____', ADDR(V))
```

lädt die im Beispiel unter Abschnitt 2.3.5.8 gesicherte Variable.

2.3.5.10 OUT (P,C)

Diese Prozedur kann unter Umgehung der Prozedur INLINE direkt auf das Z80 Ausgabe-Port zugreifen. Der Integer-Parameter P wird in das Register BC, der Zeichen-Parameter C in das Register A geladen und der Assembler-Befehl OUT (C), A ausgeführt.

Beispiel:

```
OUT(1, 'A') gibt das Zeichen 'A' an das Z80 Port 1 aus.
```

2.3.5.11 EXTERNAL (S1,V1,V2,....)

Diese Prozedur ermöglicht Aufrufe in externen ROMs bzw. RSXen und ähnelt dem BASIC-Befehlstyp „I“. Der erste Parameter ist vom Typ String und bezeichnet den Namen des externen Befehls. Die weiteren Parameter beliebiger Anzahl können vom Typ Integer, CHAR oder String sein. Mit ADDR(V) kann eine Variable V an einen externen Befehl übergeben werden. Befehlsnamen mit Kleinbuchstaben werden in Großbuchstaben umgewandelt.

Beispiel:

```
EXTERNAL('BASIC'); bewirkt einen Rücksprung nach BASIC.
```

Eine Haupt-Anwendung dieser Prozedur ist der Zugriff auf das Disketten-System.

Beispiel:

```
EXTERNAL('DIR','*.COM');
```

gibt ein Inhaltsverzeichnis aller auf dem momentanen Laufwerk gespeicherten .COM-Dateien aus.

Eine RSX sollte vor ihrem Aufruf am Anfang des Programms durch die Firmware-Routine KL LOG EXT eingeführt werden. Dies ist notwendig, da nach der Abarbeitung eines Programms alle Ereignis-Warteschlangen und damit auch die RSXen gelöscht werden (siehe CPC464 Firmware Handbuch).

2.3.5.12 AFTER (COUNT, TIMER, PROC)

AFTER entspricht dem BASIC-Befehl gleichen Namens. Der erste Parameter ist vom Typ Integer und gibt die Anzahl der 1/50-Sekunden-Einheiten an, nach denen die durch den dritten Parameter bezeichnete Prozedur aufgerufen wird. Der zweite Parameter wählt den Zeitgeber (Timer) aus und sollte eine Integer-Zahl zwischen 0 und 3 sein.

Beispiel:

```
AFTER(100,1,FRED);
```

startet die Prozedur FRED nach 2 Sekunden durch Zeitgeber 1.

2.3.5.13 EVERY (COUNT, TIMER, PROC)

EVERY entspricht dem BASIC-Befehl gleichen Namens. Die Parameter sind mit denen der Prozedur AFTER identisch (siehe Abschnitt 2.3.5.12). Die Zeitgeber sind im BASIC-Handbuch detailliert beschrieben.

Beispiel:

```
EVERY(300,2,FRED);
```

ruft alle 6 Sekunden die Pascal-Prozedur FRED auf. Die Prozedur wird mit der Priorität 2 in die Warteschlange für normale Ereignisse eingereiht.

2.3.5.14 SOUND (G, K, L, H, M, J, D)

Die Prozedur SOUND hat folgende 7 Integer-Parameter:

1. Kanal und gegenseitige Einstellung (Kanal-Status).
2. Amplituden-Hüllkurve.
3. Ton-Hüllkurve.
4. Ton-Periode.
5. Geräuschperiode.
6. Anfangs-Amplitude.
7. Dauer oder Hüllkurven-Wiederholrate.

2.3.6 Weitere vordefinierte Funktionen

2.3.6.1 RANDOM (X)

RANDOM erzeugt eine positive Pseudozufallszahl vom Typ Integer im Bereich 0 - MAXINT und übernimmt einen Parameter. Wenn dieser Parameter Null ist, stellt RANDOM (0) die nächste Zufallszahl dieser Sequenz zur Verfügung. Jeder andere Parameter startet eine neue Sequenz.

2.3.6.2 SUCC (X)

SUCC (X) ist die sog. Nachfolgefunktion und liefert den Nachfolger (successor) von X. X kann jeden skalaren Typs – außer Real – sein.

Beispiele:

SUCC('A')='B' SUCC('5')='6'

2.3.6.3 PRED (X)

PRED (X) ist die sog. Vorgängerfunktion und liefert den Vorgänger (predecessor) von X. X kann jeden skalaren Typs – außer Real – sein.

Beispiele:

PRED('j')='i' PRED(TRUE)=FALSE

2.3.6.4 ODD (X)

ODD (X) liefert TRUE, wenn der Wert von X ungerade, und FALSE, wenn der Wert von X gerade ist. X muß vom Typ Integer sein.

2.3.6.6 ADDR (V)

Diese Funktion übernimmt einen Variablen-Namen jeden Typs als Parameter und übergibt die Speicheradresse (vom Typ Integer) dieser Variablen. Informationen über die Speicherung von Laufzeit-Variablen unter HiSoft Pascal finden Sie in Anhang 3. Anhang 4 zeigt ein Beispiel zur Anwendung von ADDR.

2.3.6.7 PEEK (X,T)

Der erste Parameter dieser Funktion ist vom Typ Integer und spezifiziert eine Speicheradresse (siehe Abschnitt 2.3.5.5). Das zweite Argument ist der Ergebnis-Typ dieser Funktion.

PEEK liest Daten jeden beliebigen Typs aus dem Speicher. Die Datenoperationen von PEEK und POKE werden mit der unter Hisoft Pascal üblichen Datendarstellung (siehe Anhang 3) vorgenommen. Wenn der Speicher ab Adresse #9000 die hexadezimalen Werte 50 61 73 63 61 6C enthält, sind z.B. folgende Operationen möglich:

```
WRITE(PEEK(#9000, ARRAY[1..6] OF CHAR)) ergibt „Pascal“
```

```
WRITE(PEEK(#9000, CHAR)) ergibt „P“
```

```
WRITE(PEEK(#9000, INTEGER)) ergibt 24912
```

```
WRITE(PEEK(#9000, REAL)) ergibt 2.46227E+29
```

Anhang 3 enthält weitere Informationen über die Daten-Darstellung unter Hisoft Pascal.

2.3.6.8 SIZE (V)

Der Parameter dieser Funktion ist eine Variable, deren Speicherplatzbedarf (in Byte) berechnet wird.

2.3.6.9 INP (P)

INP kann unter Umgehung der Prozedur INLINE direkt auf die Z80-Ports zugreifen. Der Integer-Parameter P wird in das Register BC geladen und der Assembler-Befehl IN A, (C) ausgeführt.

2.3.6.10 INITEVENT (CLASS, PROC)

Diese Funktion initialisiert einen Ereignis-Block für synchrone Ereignisse und übergibt deren Adresse als Integer-Zahl. Der erste Parameter CLASS ist die Ereignis-Klasse (siehe CPC464 Firmware Handbuch). Der zweite Parameter ist eine Prozedur (ohne Parameter), die beim Eintreten des entsprechenden Ereignisses aufgerufen wird.

Diese Funktion dient dem fortgeschrittenen Programmierer zum Zugriff auf den Betriebssystem-Kern, was normalerweise in höheren Programmiersprachen nicht möglich ist.

2.3.6.11 SQ (CHANNEL)

Diese Funktion entspricht der BASIC-Funktion SQ und liefert den Status (Typ Integer) des als Integer-Parameter (CHANNEL) angegebenen Kanals (siehe BASIC Handbuch).

Beispiel:

```
WRITE(SQ(;));
```

ergibt 4, wenn keine SOUND-Befehle für Kanal A ausgegeben wurden.

2.3.6.12 REMAIN (TIMER)

REMAIN entspricht der BASIC-Funktion gleichen Namens und erfordert einen Parameter vom Typ Integer. Diese Funktion liefert einen Integer-Wert, der den Zählerstand des durch den Parameter bezeichneten Zeitgebers (TIMER) angibt. Der Zeitgeber wird gleichzeitig gesperrt.

2.4 Vordefinierte Variablen

2.4.1 ERRFLG und ERRCHK

ERRFLG und ERRCHK sind boolesche Variablen, die fehlerhafte Zahleneingaben erkennen. Wenn der Wert von ERRCHK TRUE ist und der Fehler „Digit Expected“ auftritt, wird nicht das Programm gestoppt, sondern Null eingelesen und ERRFLG auf den Wert TRUE gesetzt. Der Wert von ERRFLG ist im Normalzustand FALSE.

Beispiel:

```
ERRCHK:=TRUE;
REPEAT
  READLN; READ(I)
  IF ERRFLG THEN WRITE ('Bitte Zahl eingeben')
UNTIL NOT ERRFLG;
```

2.4.2 RA, RB, RC, RD, RE, RH, RL und RF

Diese Variablen vom Typ CHAR werden für die USER-Prozedur (siehe Abschnitt 2.3.5.5) benötigt. Ihre Werte werden vor dem Aufruf einer USER-Routine in die Z80-Register geladen und durch die entsprechenden Ergebnisse der Routine überschrieben.

Beispiel:

```
RA:='a'; USER(#BB5A)
```

gibt mit der Betriebssystem-Routine TXT OUTPUT (Adresse #BB5A) das Zeichen „a“ aus.

Nach

```
USER(#BB06);
```

enthält RA das durch die Betriebssystem-Routine KM WAIT CHAR gelieferte Ergebnis. Die vordefinierten Variablen RAF, RBC, RDE und RHL werden bei einer Zuweisung von RA, RB etc. ebenfalls geändert.

2.4.3 RAF, RBC, RDE und RHL

Diese Variablen vom Typ Integer werden für die USER-Prozedur (siehe Abschnitt 2.3.5.5) benötigt. Ihre Werte werden vor dem Aufruf einer USER-Routine in die Z80-Register geladen und durch die entsprechenden Ergebnisse der Routine überschrieben.

```
RA:=CHR(I);
USER(#BCC5); {SOUND T ADDRESS}
IF ODD(RAF) THEN
  WRITE('Adresse der Ton-Huellkurve',RHL:4:H)
ELSE WRITE('Huellkurve fehlt')
```

gibt die Adresse der Hüllkurve I – sofern vorhanden – aus. `ODD (RAF)` ist `TRUE`, wenn das Carry-Flag gesetzt ist.

Eine Zuweisung von `RHL` verändert die entsprechenden Werte von `RH,RL` etc. (siehe Abschnitt 2.4.2).

ABSCHNITT 3

Syntax und Compiler-Optionen

3.1 Kommentare

Ein Kommentar darf zwischen reservierten Wörtern, Zahlen, Namen oder speziellen Symbolen stehen (siehe Anhang 2) und wird durch die Zeichen „{“ und „}“ oder „(“ und „)“ eingeschlossen. Steht nach der öffnenden Kommentar-Klammer das Zeichen „\$“, so werden alle Zeichen bis zur schließenden Klammer „}“ oder „)“ ignoriert, da der Compiler eine Compiler-Option erwartet (siehe unten). Die Zeichen „{“ und „}“ werden durch gemeinsames Drücken von SHIFT und „[“ bzw. SHIFT und „]“ erzeugt.

3.2 Compiler-Optionen

Compiler-Optionen werden zwischen den Kommentar-Klammern und nach dem Symbol „\$“ in das Programm eingefügt und können sowohl Groß- als auch Kleinbuchstaben enthalten.

Beispiel:

`(*$C- ,A-*)` schaltet die Tastatur- und Array-Abfrage aus.

Es stehen folgende Optionen zur Verfügung:

Option L:

Steuert die Auflistung des Programms und der Object-Code-Adressen.

L+ erzeugt ein vollständiges Listing.

L- erzeugt ein Listing der fehlerhaften Zeilen.

Standard-Einstellung: L+

Option O:

Steuert die Abfrage auf Bereichsüberschreitung (Overflow). Integer-Multiplikationen und -Divisionen sowie alle arithmetischen Operationen mit reellen Zahlen werden generell auf Bereichsüberschreitung geprüft.

O+ überprüft Integer-Additionen und -Subtraktionen.

O- obige Überprüfungen werden nicht durchgeführt.

Standard-Einstellung: O+

Option C:

Steuert die Tastaturabfrage während der Ausführung eines Programms. Ist C+ eingestellt, so stoppt jede gedrückte Taste die Programm-Ausführung. Durch nachfolgendes Betätigen von [ESC] wird die Ausführung durch einen HALT (siehe Abschnitt 2.3.5.6) abgebrochen. Jede andere Taste setzt die Ausführung fort.

Da die Abfrage zu Beginn jeder Schleife, Prozedur und Funktion durchgeführt wird, kann in der Testphase eine nicht korrekt arbeitende bzw. verlassene Schleife etc. erkannt werden. Die Abfrage sollte nach dem Debuggen zum schnelleren Programm-Ablauf gesperrt werden.

C+ schaltet die Tastatur-Abfrage ein;
vergrößert das Objekt-Programm nicht.

C- schaltet die Tastatur-Abfrage aus.

Standard-Einstellung: C+

Option S:

Steuert die Stack-Abfrage.

Durch S+ wird der Stack zu Beginn jedes Prozedur- und Funktions-Aufrufs auf Überlauf geprüft. Wenn der Stack überläuft wird die Ausführung abgebrochen und die Meldung „Out of RAM at PC=XXXX“ ausgegeben. Diese Maßnahme schützt nicht in jedem Fall vor Programm-Abstürzen und kann überflüssige Abbrüche bewirken.

S- schaltet die Stack-Abfrage aus.

Standard-Einstellung: S+

Option A:

Steuert die Abfrage auf Bereichsüberschreitung bei Array-Indizes. Array-Indizes müssen innerhalb der durch die Array-Deklaration festgelegten Grenzen liegen.

A+ und ein zu großer bzw. kleiner Array-Index bewirken einen Programm-Stop und die Ausgabe der Meldung „Index too high“ bzw. „Index too low“.

A- schaltet die Array-Abfrage aus.

Standard-Einstellung: A+

Option I:

Eine arithmetische Operation mit 16 Bit-Integer-Zahlen im 2er-Komplement erzeugt einen Überlauf, wenn die Argumente bei den Operationen $>$, $<$, $>=$ oder $<=$ um mehr als **MAXINT (32767)** voneinander abweichen. Das Ergebnis einer derartigen Operation ist demzufolge verfälscht. **I+** sorgt dafür, daß die Ergebnisse des Vergleichs daraufhin überprüft werden. Eine gleichartige Situation tritt bei arithmetischen Operationen mit reellen Zahlen auf, wenn deren Argumente um mehr als **3.4E38** voneinander abweichen. In diesem Fall kann der Überlauf jedoch nicht abgefangen werden.

I- schaltet die Überlauf-Abfrage aus.

Standard-Einstellung: **I-**

Option P:

P schaltet das Ausgabegerät für das Compiler-Listing zwischen Bildschirm und Drucker um. Dieser Option folgt kein „+“ oder „-“.

Standard-Einstellung: Bildschirm

Option F:

F muß ein Leerzeichen und ein 12 Zeichen langer Dateiname folgen. Ein kürzerer Dateiname muß mit Leerzeichen aufgefüllt werden.

Diese Option bewirkt den Einschluß von Pascal Source-Text aus der spezifizierten Datei am Ende der momentanen Zeile. Hierdurch kann der Programmierer Cassetten-Bibliotheken häufig benutzter Prozeduren und Funktionen anlegen und in bestimmte Programme einbinden.

Das Programm läßt sich mit dem **P**-Befehl des Editors sichern.

Beispiel:

(***S F MATRIX** schließe den Text aus der Cassetten-Datei **MATRIX** ein *)

Beim Schreiben sehr langer Programme reicht der Speicherplatz häufig nicht aus, um den Source- und Object-Code gleichzeitig abzulegen. Dieses Programm ist dennoch übersetzbar, indem es auf Cassette gesichert und die **F**-Option verwendet wird. Dadurch befinden sich jeweils nur 128 Bytes Source-Code im RAM, wodurch mehr Platz für den Object-Code zur Verfügung steht.

Diese Option kann nicht verschachtelt werden.

Die Compiler-Optionen sollten zum Austesten von Programmen eingeschaltet und zur Beschleunigung der Abarbeitung ausgetesteter Programme ausgeschaltet werden.

ABSCHNITT 4

Der integrierte Editor

4.1 Einführung

Der mit allen Versionen von HiSoft Pascal ausgelieferte Editor ist ein einfacher, zeilenorientierter Editor, der mit allen Z80-Betriebssystemen verwendet werden kann. Er ist für den Schneider CPC464 um Bildschirm-Editierhilfen erweitert und unterstützt die COPY-Taste sowie die SHIFT-Cursorstasten.

Der editierte Text wird in komprimierter Form im Speicher abgelegt. Die Anzahl der führenden Leerzeichen einer Zeile wird in einem Zeichen am Zeilenanfang abgespeichert. Auch die reservierten Wörter sind derart abgespeichert, was insgesamt zu einer 25%igen Reduzierung des erforderlichen Speicherplatzes führt.

Der Editor meldet sich automatisch nach dem Laden von HiSoft Pascal mit:

```
HiSoft Pascal Schneider CPC464  
Version of 26/9/84  
Copyright HiSoft 1983,4  
All rights reserved
```

Daraufhin erscheint eine Help-Maske und das Prompt-Zeichen „>“. Sie können nun eine Befehlszeile mit dem folgenden Format eingeben:

```
CN1,N2,S1,S2 gefolgt von [ENTER],
```

wobei

C das auszuführende Kommando (siehe Abschnitt 4.2),
N1,N2 je eine Zahl zwischen 1 und 32767 und
S1,S2 je eine bis zu 20 Zeichen lange Zeichenkette ist.

Das Komma dient hier zur Trennung der verschiedenen Parameter (dieses Zeichen ist mit dem Q-Kommando neu definierbar). Leerzeichen werden außerhalb der Zeichenketten ignoriert.

Die einzelnen Parameter sind nicht immer zwingend erforderlich, einige Kommandos werden jedoch ohne N1 und N2 nicht ausgeführt (vergleiche D-Befehl). Der Editor speichert bereits eingegebene Zahlen oder Zeichenketten und verwendet sie, wenn keine neuen Argumente in der Befehlszeile eingegeben werden.

Die Werte N1 und N2 sind beim Programmstart mit 10 vorbelegt, die Zeichenketten (Strings) dagegen grundsätzlich leer. Wenn Sie eine unzulässige bzw. fehlerhafte Befehlszeile, wie z.B. F-1,100,HALL0, eingeben, wird sie ignoriert und die Meldung „Pardon?“ ausgegeben. Sie sollten daraufhin die korrigierte Zeile eingeben, z.B. F1,100,HALL0. Diese Fehlermeldung erscheint auch, wenn beispielsweise S1 oder S2 länger als 20 Zeichen sind. Im ersten Fall (S1 > 20 Zeichen) werden die Zeichen ab Position 21 ignoriert. Bei der Eingabe können Groß- und Kleinbuchstaben verwendet sowie die Steuerfunktionen (siehe unten) aufgerufen werden.

In den folgenden Abschnitten sind die verschiedenen Kommandos des Editors ausführlich beschrieben. Ein durch spitze Klammern eingeschlossener Parameter ist zur Ausführung dieses Kommandos unbedingt anzugeben.

4.2 Die Editor-Kommandos

4.2.1 Texteingabe

Text wird durch Eingabe der Zeilennummer, eines Leerzeichens und des gewünschten Textes erstellt oder mit dem I-Kommando eingefügt. Wenn Sie eine bereits existierende Zeilennummer ohne Text eingeben, wird diese Zeile nach Drücken von [ENTER] gelöscht. Zur Texteingabe stehen folgende Steuerfunktionen bereit:

[CTRL]X löscht die Zeile bis zum Zeilenanfang.
[TAB] springt zur nächsten Tabulatorposition.
[ESC] kehrt in die Kommando-Schleife zurück.
[CTRL]P schaltet die Ausgabe wechselweise auf Drucker/Bildschirm um.
[DEL] löscht das Zeichen links vom Cursor innerhalb einer Zeile.

Der eingegebene Text wird in einem internen Puffer abgelegt. Sollte dieser Puffer voll sein, können mit [DEL] oder [CTRL]X Daten gelöscht und damit neuer Speicherplatz bereitgestellt werden.

Kommando: I n, m

Mit diesem Befehl lassen sich Zeilennummern automatisch erzeugen. Parameter n ist die Zeilennummer und Parameter m die Schrittweite, mit der weitergezählt wird. Den gewünschten Text geben Sie ein, nachdem die Zeilennummer auf dem Bildschirm erscheint. Eine neue Zeile wird dann jeweils automatisch nach dem abschließenden Betätigen der [ENTER]-Taste bereitgestellt. Dieser Modus kann mit [ESC] verlassen werden.

Geben Sie eine Zeile mit einer bereits existierenden Zeilennummer ein, so wird die Nummer der bestehenden Zeile um 1 erhöht und die neue Zeile eingefügt. Eine Zeilennummer größer als 32767 bewirkt den Ausstieg aus dem Einfügens-Modus.

Eine mit mehr als 80 Zeichen (die Puffergröße) eingegebene Zeile stellt automatisch die neue Zeile bereit, woraufhin die Texteingabe fortgesetzt werden kann.

4.2.2 Text-Listing

Sie können ein Listing Ihres Programms auf dem Bildschirm (mit dem L-Kommando) oder dem Drucker (mit dem Z-Kommando) ausgeben.

Kommando: L *n*, *m*

Der Text von Zeilennummer *n* (standardmäßig 1) bis *m* (standardmäßig 32767) wird auf dem Bildschirm ausgegeben. Zum Anzeigen des gesamten Textes kann L ohne Parameter eingegeben werden. Nach jeweils 24 Zeilen (eine Seite) pausiert das Listing, durch Drücken einer beliebigen Taste können die nächsten 24 Zeilen dargestellt werden. [ESC] beendet die Auflistung.

Kommando: Z *n*, *m*

Gibt die Textdatei von Zeile *n* bis *m* auf dem Drucker aus. Wenn der Drucker nicht angeschlossen oder im Zustand „off-line“ ist, erscheint die Meldung „NO PRINTER!“. Fehlende Parameter *n* und *m* bewirken den Ausdruck der gesamten Textdatei.

Durch Drücken einer beliebigen Taste läßt sich die Ausgabe unterbrechen, nachfolgendes Betätigen von [ESC] bricht den Druckvorgang vollständig ab und stellt den Editor bereit. Jede andere Taste setzt die Auflistung fort.

4.2.3 Text Editieren

Nachdem ein Text erstellt wurde, müssen fehlerhafte Zeilen neu editiert bzw. geändert werden. Hierzu stehen verschiedene Befehle zum Löschen und Ändern von Zeilen oder zum Erzeugen neuer Zeilennummern bereit. Diese Befehle werden im Folgenden erklärt.

Doch zuvor einige elementare Grundsätze des Bildschirm-Editierens:

Wenn Sie sich im Kommando-Modus des Editors (gekennzeichnet durch „>“ am Zeilenanfang) befinden, können Sie den Cursor als Lese- oder Schreib-Cursor programmieren, indem Sie erst [SHIFT] und dann eine der Cursortasten gemeinsam niederdrücken. Der Schreib-Cursor verbleibt nun auf der Position des Original-Cursors. Der Lese-Cursor kann hingegen mit [SHIFT] und den Cursortasten über den Bildschirm bewegt werden. Wenn sich der Lese-Cursor auf der gewünschten Position befindet, entriegeln Sie einfach [SHIFT] und die entsprechende Cursortaste. Sie können nun entweder direkt Zeichen über die Tastatur eingeben, die dann auf der Schreib-Cursorposition erscheinen, oder mit Hilfe der [COPY]-Taste Text von der Lese-Cursorposition zur Schreib-Cursorposition kopieren.

Mit [ENTER] kann daraufhin der Kopier-Modus verlassen werden.

Es stehen folgende Zeileneditier-Kommandos zur Verfügung:

Kommando: D <*n*, *m*>

Die Textzeilen *n* bis *m* werden in der Textdatei gelöscht. Wenn *m* < *n* ist oder weniger als 2 Parameter angegeben sind, wird das Kommando zur Vermeidung von Fehlern nicht ausgeführt. Sie können eine einzelne Zeile löschen, indem Sie für *n* und *m* den gleichen Wert einsetzen bzw. nur die entsprechende Zeilennummer und [ENTER] eingeben.

Kommando: **M**<n, m, d>

Dieses Kommando kopiert den Text zwischen den Zeilennummern *n* und *m* in den Bereich ab Zeilennummer *d* und ändert die Zeilennummern entsprechend. Der Original-Textbereich wird dabei gelöscht.

Ein Textbereich kann nicht auf sich selbst kopiert werden.

Kommando: **N**<n, m>

Mit diesem Kommando wird eine Textdatei beginnend mit *n* und in Schritten von *m* neu durchnummeriert. Beide Parameter sind anzugeben. Ist eine neue Zeilennummer größer als 32767, so wird die bisherige Numerierung beibehalten.

Kommando: **F** n, m, f, s

Dieses Kommando sucht im Zeilenbereich *n* bis *m* nach der durch den Parameter *f* bezeichneten Zeichenkette. Sie können die gefundene Zeichenkette im daraufhin bereitgestellten Editier-Modus ändern bzw. bearbeiten oder durch den vor Anwendung dieses Befehls definierten String *s* ersetzen. Das **F**-Kommando ist auch mehrfach auf die gesamte Textdatei anwendbar.

Der Zeilenbereich und die beiden Zeichenketten können vor der Anwendung dieses Kommandos definiert werden (siehe Abschnitt 4.3), wodurch dieser Befehl einfach mit „**F**“ – ohne Angabe von Parametern – aufrufbar ist.

Kommando: **E** n

Schaltet den Editor auf die Zeilennummer *n*. Bei fehlendem *n* wird der Befehl nicht ausgeführt. Die spezifizierte Zeile wird in den Puffer geladen und auf dem Bildschirm dargestellt, sowie die gleiche Zeilennummer (ohne Text) in der darauffolgenden Zeile ausgegeben. Im nun bereitgestellten Editier-Modus kann die Zeile bearbeitet werden. Geändert wird jedoch nur der Puffer-Inhalt und nicht der Text selbst, wodurch der ursprüngliche Text jederzeit wiederherstellbar ist.

In diesem Modus weist ein unsichtbarer Text-Zeiger auf die aktuell bearbeitete Position im Puffer, der durch die folgenden Unterbefehle manipuliert werden kann:

- “(Leerzeichen) – verschiebt den Text-Zeiger innerhalb der Zeile um eine Position nach rechts.
- [DEL] – verschiebt den Text-Zeiger innerhalb der Zeile um eine Position nach links.
- [TAB] – verschiebt den Text-Zeiger innerhalb der Zeile auf die nächste Tabulatorposition.
- [ENTER] – beendet die Zeilen-Editierung und übernimmt die Änderungen.
- Q – beendet die Zeilen-Editierung, ohne die Änderungen zu berücksichtigen.
- R – stellt den Originalzustand des Puffers wieder her und löscht alle Änderungen.
- L – listet den Rest der Zeile hinter der momentanen Zeigerposition auf. Der Zeiger steht anschließend am Zeilenanfang.
- K – löscht das Zeichen auf der momentanen Zeigerposition.
- Z – löscht alle Zeichen rechts (und einschließlich) der momentanen Zeigerposition bis zum Zeilenende.

Durch das F-Kommando (siehe oben) definierten String. Weiterer, gleicher String auffindbar ist, verläßt dieser Un- automatisch den Editier-Modus und übernimmt die Ände- ch dem Auffinden eines gleichen Strings wird der Editier- diese Zeile bereitgestellt. Der Text-Zeiger befindet sich Zeilenanfang der gefundenen Zeichenkette.

mittels F-Kommando (siehe oben) gefundenen String gleichen Kommando definierten String s und führt dar- Interbefehl F aus (siehe Beispiel im Anhang 4.3).

ben auf der momentanen Zeigerposition ein. Mit kann dieser Modus verlassen werden, wobei der Text- der dem zuletzt eingefügten Zeichen steht. [DEL] und [EOL]: diesem Modus anwendbar. Der Cursor wird durch in „*“ dargestellt.

Zeiger auf das Zeilenende und schaltet den Einfügu- (siehe oben).

bt das Zeichen auf der momentanen Zeiger-Position und den Zeiger um ein Zeichen nach rechts. Mit [ENTER] Modus verlassen werden. [DEL] und [TAB] sind in aus ebenfalls anwendbar. Der Cursor wird durch das Zei- dargestellt.

Befehle

ichert, mit G von der Cassette geladen und mit V über-

wird unter dem Namen des Parameters s abgespeichert. tos müssen vor deren Anwendung definiert sein.

ommandos überprüfen Sie bitte, ob Ihr Recorder einge- us ist.

iner Textdatei mit dem Namen des Parameters s durch- ht auffindbar, erscheint eine Fehlermeldung, andernfalls n. Bei einem Ladefehler wird der Vorgang abgebrochen

gang abbrechbar.

in Speicher, so wird die neue Datei dazugeladen und die Zeilennummer 1 in Einserschritten neu durchnummeriert.

(siehe F-Kommando) definierten Strings auf der Cassette gesichert. Der Programm-Code wird in binärer Form abgespeichert, so daß er mit dem BASIC-Befehl `LOAD` geladen werden kann.

Der erzeugte Object-Code enthält Befehle, die bei der Ausführung die Routine `MC START PROGRAM` aufrufen. Diese Routine löscht BASIC und alle in die Firmware integrierten RSXen. Wenn nun zur Laufzeit des Programms eine RSX benötigt wird, muß über die Pascal-Prozedur `USER` die Firmware-Routine `KL LOG EXT (#BCD1)` aufgerufen werden, die wiederum die entsprechende RSX initialisiert.

Da der Object-Code an die Laufzeit-Routinen angehängt wird und damit den Compiler überschreibt, muß er nach einem T-Kommando erneut geladen werden.

Die Sicherung auf Cassette kann unterbrochen werden, indem Sie auf die Frage „OK?“ eine beliebige Taste (außer „Y“) drücken. Hernach steht der voll funktionsfähige Editor erneut zur Verfügung.

4.2.6 Weitere Befehle

Kommando: H

Dieses Kommando stellt sämtliche Editier-Befehle auf dem Bildschirm dar (Help-Kommando).

Kommando: Q, d

Mit Q ist das Abgrenzungszeichen (Standard ist das Komma „`,`“) neu definierbar. Das neue Abgrenzungszeichen muß als Parameter d eingesetzt werden. Ein Leerzeichen ist als Abgrenzungszeichen unzulässig.

Kommando: U

U gibt die letzte Zeile der Textdatei aus und ermöglicht damit das schnelle Auffinden des Datei-Endes.

Kommando: W

Änden wechselweise die Zeichenanzahl pro Zeile von 40 auf 80 Zeichen und umgekehrt.

Kommando: Y

Gibt das momentane Abgrenzungszeichen, den Zeilenbereich (n-m) sowie die Strings f und s (siehe F-Kommando) auf dem Bildschirm aus.

Kommando: X

Zeigt die Start- und Endadresse des Textes in hexadezimaler Form an, womit die Text-Größe in Bytes berechnet werden kann.

Kommando: |

Der senkrechte Strich („|“) ermöglicht den Aufruf von Befehlen im Hintergrund-ROM. Der Befehlsname für das externe ROM und optionell die erforderlichen Parameter sind hinter dem Senkrecht-Strich anzugeben. Die Parameter müssen durch ein Komma getrennt, String-Parameter durch das Zeichen „'“ eingeschlossen werden.

Eine ungültige Kommando-Zeile erzeugt die Fehlermeldung „Pardon“.

Beispiel:

```
|dir, '*.PAS'
```

listet ein Inhaltsverzeichnis aller Disketten-Dateien mit der Extension .PAS auf.

```
|basic
```

bewirkt die Rückkehr nach BASIC.

4.3 Beispielprogramm

Nehmen wir an, das folgende Programm sei beispielsweise mit I 10, 10 eingegeben worden:

```
10 PROGRAM BUBBLESORT
20 CONST
30 Size = 2000;
40 VAR
50 Numbers : ARRAY [1..Size] OF INTEGER;
60 I, Temp : INTEGER;
70 BEGIN
80 FOR I:=1 TO Size DO Number[I] :=RANDOM;
90 REPEAT
100 FOR I:=1 TO Size DO
110 Noswaps := TRUE;
120 IF Number[I] > Number[I+1] THEN
130 BEGIN
140 Temp := Number[I];
150 Number[I] := Number[I+1];
160 Number[I+1] := Temp;
170 Noswaps := FALSE
180 END
190 UNTIL Noswapss;
195 FOR I := 1 TO Size DO WRITE(Number[I]:4)
200 END.
```

Dieses Programm enthält folgende „Fehler“:

- Zeile 10: Semikolon fehlt.
- Zeile 30: Kein Fehler, aber Size soll z.B. 100 sein.
- Zeile 100: Kein Fehler, aber Size soll z.B. Size-1 sein.
- Zeile 110: Diese Zeile soll auf Zeile 95 gesetzt werden.
- Zeile 190: Noswapss soll in Noswaps umbenannt werden.

Weitere : Die Variable Numbers ist zwar deklariert, wird aber mit Number aufgerufen.

Die boolesche Variable Noswaps ist nicht deklariert.

Die Korrekturen können folgendermaßen vorgenommen werden:

- | | |
|------------------------|---|
| F60,210,Number,Numbers | und wiederholter Aufruf des Unterbefehls S. |
| E10 | und X;[ENTER][ENTER]. |
| E30 | und =====KC1[ENTER][ENTER]. |
| F100,100,Size,Size-1 | und Aufruf des Unterbefehls S. |
| M110,95 | |
| E190 | und X DELETE[ENTER][ENTER]. |
| 65Noswaps:BOOLEAN; | |
| N10,10 | Neu-Numerierung in Schritten von 10. |

ANHANG 1

FEHLER- MELDUNGEN

A.1.1 Fehlermeldungen des Compilers

1. Number too large.
Zahl zu groß.
2. Semi-colon expected.
Semikolon fehlt.
3. Undeclared identifier.
Name nicht deklariert.
4. Identifier expected.
Name fehlt.
5. Use '=' not ':=' in a constant declaration.
Verwenden Sie „=“ statt „:=“ in einer Konstanten-Deklaration.
6. '=' expected.
„=“ fehlt.
7. This identifier cannot begin a statement.
Name am Beginn einer Anweisung unzulässig.
8. ':=' expected.
„:=“ fehlt.
9. ')' expected.
„)“ fehlt.
10. Wrong type.
Falscher Typ.
11. '.' expected.
„.“ fehlt.
12. Factor expected.
Faktor fehlt.

13. Constant expected.
Konstante fehlt.
14. This identifier is not a constant.
Name ist keine Konstante.
15. 'THEN' expected.
„THEN“ fehlt.
16. 'DO' expected.
„DO“ fehlt.
17. 'TO' or 'DOWNT0' expected.
„TO“ oder „DOWNT0“ fehlt.
18. '(' expected.
„(“ fehlt.
19. Cannot write this type of expression.
Ausdruck kann nicht geschrieben werden.
20. 'OF' expected.
„OF“ fehlt.
21. ',' expected.
„,“ fehlt.
22. ':' expected.
„:“ fehlt.
23. 'PROGRAM' expected.
„PROGRAM“ fehlt.
24. Variable expected since parameter is a variable parameter.
Variable fehlt (Parameter ist ein Variablen-Parameter).
25. 'BEGIN' expected.
„BEGIN“ fehlt.
26. Variable expected in call to READ.
Variable fehlt im READ-Aufruf.
27. Cannot compare expressions of this type.
Ausdrücke dieses Typs können nicht verglichen werden.
28. Should be either type INTEGER or type REAL.
Typ muß Integer oder Real sein.
29. Cannot read this type of variable.
Dieser Variablentyp kann nicht gelesen werden.
30. This identifier is not a type.
Name ist kein Typ.
31. Exponent expected in real number.
Exponent fehlt in Real-Zahl.
32. Scalar expression (not numeric) expected.
Skalaren (nicht numerischen) Ausdruck angeben.

33. Null strings not allowed (use CHR(0)).
Null-Strings nicht zulässig (CHR(0) verwenden!).
34. '[' expected.
„[“ fehlt.
35. ']' expected.
„]“ fehlt.
36. Array index type must be scalar.
Array-Index muß vom Typ Skalar sein.
37. '..' expected.
„..“ fehlt.
38. ']' or ',' expected in ARRAY declaration.
„]“ oder „,“ fehlt in der ARRAY-Deklaration.
39. Lowerbound greater than upperbound.
Untergrenze größer als Obergrenze.
40. Set too large (more than 256 possible elements).
Menge zu groß (mehr als 256 Elemente).
41. Function result must be type identifier.
Funktionsergebnis muß vom Typ Name sein.
42. ', ' or ']' expected in set.
„,“ oder „]“ fehlt in der Menge.
43. '..' or ', ' or ']' expected in set.
„..“ oder „,“ oder „]“ fehlt in der Menge.
44. Type of parameter must be a type identifier.
Typ des Parameters muß ein Name sein.
45. Null set cannot be the first factor in a non-assignment statement.
Null-Menge als erster Faktor in einer nicht zuweisenden Anweisung unzulässig.
46. Scalar (including real) expected.
Skalar (einschl. Real) fehlt.
47. Scalar (not including real) expected.
Skalar (ohne Real) fehlt.
48. Sets incompatible.
Mengen nicht kompatibel.
49. '<' and '>' cannot be used to compare sets.
„<“ und „>“ beim Vergleich von Mengen unzulässig.
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' or 'BEGIN' expected.
„FORWARD“, „LABEL“, „CONST“, „VAR“, „TYPE“ oder „BEGIN“ fehlt.
51. Hexadecimal digit expected.
Hexadezimale Ziffer fehlt.

52. Cannot POKE sets.
Mengen mit POKE unzulässig.
53. Array too large (>64K).
Array zu groß (>64K).
54. 'END' or ';' expected in RECORD definition.
„END“ oder „;“ fehlt in der RECORD-Definition.
55. Field identifier expected.
Komponenten-Name fehlt.
56. Variable expected after 'WITH'.
Nach „WITH“ fehlt die Variable.
57. Variable in WITH must be of RECORD type.
Variable nach WITH muß vom Typ RECORD sein.
58. Field identifier has not had associated WITH statement.
Komponenten-Name hat keine zugehörige WITH-Anweisung.
59. Unsigned integer expected after 'LABEL'.
Vorzeichenlose Integer-Zahl fehlt nach „LABEL“.
60. Unsigned integer expected after 'GOTO'.
Vorzeichenlose Integer-Zahl fehlt nach „GOTO“.
61. This label is at the wrong level.
Marke (Label) auf falscher Programm-Ebene.
62. Undeclared label.
Marke (Label) nicht deklariert.
63. The parameter of SIZE should be a variable.
SIZE-Parameter muß eine Variable sein.
64. Can only use equality tests for pointers.
Nur Abfragen auf Gleichheit bei Zeigern zulässig.
67. The only write parameter for integers with two ':'s is e:m:H.
Der Schreib-Parameter mit zwei „:“ muß für Integer-Zahlen vom Format e:m:H sein.
68. Strings may not contain end of line characters.
Strings dürfen keine end-of-line-Zeichen enthalten.
69. The parameter of NEW, MARK or RELEASE should be a variable of pointer type.
Die Parameter von NEW, MARK oder RELEASE müssen Variablen vom Typ Zeiger sein.
70. The parameter of ADDR should be a variable.
Der Parameter von ADDR muß eine Variable sein.
71. This parameter must be a procedure.
Der Parameter muß eine Prozedur sein.
72. This parameter must be a parameterless procedure.
Der Parameter muß eine Prozedur (ohne Parameter) sein.
73. No more than 5 sections in an envelope.
Nur 5 Hüllkurven-Abschnitte zulässig.

A.1.2 Laufzeit-Fehlermeldungen

Nach dem Erkennen eines Laufzeit-Fehlers wird eine Fehlermeldung auf dem Bildschirm ausgegeben. Ihr folgt die Speicheradresse XXXX (genauer : „at PC=XXXX“), bei der dieser Fehler auftrat. Häufig ist die Fehlerquelle offensichtlich, andernfalls kann das Compiler-Listing und die Fehleradresse weiterhelfen.

1. Halt
Halt.
2. Overflow
Überlauf.
3. Out of RAM
Speicher zu klein (RAM).
4. /by zero
Division durch Null. (bei / und DIV).
5. Index too low
Index zu klein.
6. Index too high
Index zu groß.
7. Maths Call Error
Mathematischer Fehler.
8. Number too large
Zahl zu groß.
9. Number expected
Zahl fehlt.
10. Line too long
Zeile zu lang.
11. Exponent expected
Exponent fehlt.

Diese Fehlermeldungen bewirken eine Programm-Unterbrechung.

ANHANG 2

Reservierte Wörter und vordefinierte Namen

A.2.1 Reservierte Wörter

AND	ARRAY	BEGIN	CASE	CONST	DIV	DO
DOWNTO	ELSE	END	FORWARD	FUNCTION	GOTO	IF
IN	LABEL	MOD	NIL	NOT	OF	OR
PACKED	PROCEDURE	PROGRAM	RECORD	REPEAT	SET	THEN
TO	TYPE	UNTIL	VAR	WHILE	WITH	

A 2.2 Spezielle Symbole

Die folgenden Symbole haben unter Hisoft Pascal eine syntaktische Bedeutung:

+	-	*	/		
=	<>	<	<=	>=	>
()	[]		
{	}	(*	*)		
:	:=	.	'	;	:
!	..				

A 2.3 Vordefinierte Namen

Die folgenden Namen werden im Deklarationsteil am Anfang des Programms eingeführt und sind in der Regel im gesamten Programm-Block gültig. Sie können jedoch in bestimmten Unter-Blöcken durch den Programmierer umdefiniert werden (siehe Abschnitt 2).

```

CONST      MAXINT=32767;

TYPE      BOOLEAN=(FALSE,TRUE);
          CHAR {Der erweiterte ASCII-Zeichen-
          satz;
          INTEGER=-MAXINT..MAXINT;
          REAL {Eine Untermenge der Realzahlen.
          siehe Abschnitt I.3.

VAR      ERRFLG, ERRCHK: BOOLEAN: RA, RB,
          RC, RD, RE, RF, RH, RL: CHAR;
          RAT, RBC, RDE, RHL: INTEGER;

PROCEDURE WRITE; WRITELN; READ; READLN; PAGE;
          HALT; USER; POKE; INLINE;
          OUT; NEW; MARK; RELEASE; TIN; TOUT;
          AFTER; EVERY; SOUND; ONSQ;
          EXTERNAL; ENV; ENT;

FUNCTION  ABS; SQR; ODD; RANDOM; ORD; SUCC;
          PRED; INCH; EOLN;
          PEEK; CHR; SQRT; ENTIER; ROUND;
          TRUNC; FRAC; SIN; COS;
          TAN; ARCTAN; EXP; LN; ADDR; SIZE;
          INP; REMAIN; INITEVENT; REMAIN;

```


ANHANG 3

Interne Daten-Darstellungen und -Speicherung

A.3.1 Daten-Darstellung

Im folgenden wird ausführlich gezeigt, wie im HiSoft Pascal Daten intern dargestellt werden.

Die Information über den jeweils erforderlichen Speicherbedarf dürfte für die meisten Programmierer sehr nützlich sein (siehe auch SIZE Funktion im Abschnitt 2.3.6.8); andere Einzelheiten könnten für diejenigen von Interesse sein, die Pascal und Maschinencode-Programme mischen möchten.

A 3.1.1 Integer-Zahlen

Integer-Zahlen werden im 2er Komplement dargestellt und belegen jeweils 2 Bytes im Speicher.

Beispiele:

```
1   ≙ #0001
256 ≙ #0100
-256 ≙ #FF00
```

Integer-Zahlen werden durch den Compiler im Z80-Register HL abgelegt.

A 3.1.2 Zeichen, Boolesche und andere Skalare Datentypen

Diese Daten werden binär (vorzeichenlos) dargestellt und belegen jeweils 1 Byte im Speicher.

Zeichen: 8 Bit, erweiterter ASCII-Zeichensatz.

```
'E' ≙ #45
'[' ≙ #5B
```

ean:
 (TRUE)=1 TRUE wird durch eine 1 dargestellt.
 (FALSE)=0 FALSE wird durch eine 0 dargestellt.

e Daten werden im Z80-Register A abgelegt.

3.1.3 Reelle Zahlen

le Zahlen (real) werden im Format (Mantisse, Exponent) als binäre Werte darge-

$2 * 10^0$ oder $1.0_2 * 2^1$
 $1 * 10^0$ oder $1.0_2 * 2^0$
 $.5 \triangleq -1.25 * 10^1$ oder $\triangleq -25 * 2^{-1}$
 $\triangleq 1.0 * 10^{-1}$ oder $\triangleq -11001_2 * 2^{-1}$
 $\triangleq 1.1001_2 * 2^3$ normierte Darstellung.

$\frac{1}{10} \triangleq \frac{1}{1010_2} \triangleq \frac{0.1_2}{101_2}$

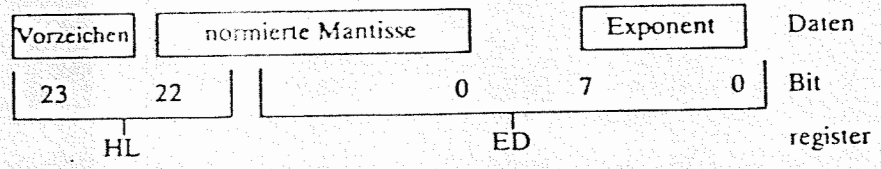
zur Berechnung ist eine binäre Division erforderlich.

0.0001100
 $101 \overline{) 0.1000000000000000}$
 $\underline{101}$
 110
 $\underline{101}$
 1000
 $\underline{101}$

ab hier wiederholt sich die Ziffernfolge

$= \frac{0.1_2}{101_2} = 0.0001100_2$
 $1.1001100 * 2^{-4}$ Ergebnis.

lle Zahlen benötigen jeweils 4 Bytes mit folgendem Format:



Vorzeichen: Das Vorzeichen ist 1 für negative und 0 für positive Mantissen.

Normierte Mantisse: Die Mantisse wird auf das Format 1.xxxxxx normiert, wobei das oberste Bit (Bit 22) immer 1 ist.
Ausnahme: Darstellung von 0 (HL=0, DE=0).

Exponent: Darstellung im binären 2er Komplement.

Beispiel:

```

2 ≙ 0 1000000 00000000 00000000 00000001 (#40, #00, #00, #01)
1 ≙ 0 1000000 00000000 00000000 00000000 (#40, #00, #00, #00)
-12.5 ≙ 1 1100100 00000000 00000000 00000011 (#E4, #00, #00, #03)
0.1 ≙ 0 1100110 01100110 01100110 11111100 (#66, #66, #66, #FC)

```

Register-Belegung für die obigen Reellen Zahlen:

```

2      ≙      LD HL, #4000
          LD DE, #0100
1      ≙      LD HL, #4000
          LD DE, #0000
-12.5 ≙      LD HL, #E400
          LD DE, #0300
0.1    ≙      LD HL, #6666
          LD DE, #FC66

```

Hinweis: Reelle Zahlen werden in der Reihenfolge EDLH im Speicher abgelegt.

A 3.1.4 Records und Arrays

Der für Records erforderliche Speicherplatz entspricht der Summe seiner Komponenten-Speicherplätze.

Der für Arrays erforderliche Speicherplatz entspricht

$n * s$

wobei

n = Anzahl der Array-Elemente und

s = Speicherplatz des Array-Elements

ist.

Beispiele:

```

ARRAY [1..10] OF INTEGER      erfordert 10*2=20 Bytes
ARRAY [2..12, 1..10] OF CHAR  erfordert 11*10*1=110 Bytes

```

A 3.1.5 Mengen

Mengen werden als Bit-Strings gespeichert. Wenn der Basis-Typ der Menge n Elemente enthält, errechnet sich die Anzahl der erforderlichen Bytes zu

$$(n-1) \text{DIV} 8 + 1.$$

Beispiele:

SET OF CHAR erfordert $(256-1) \text{DIV} 8 + 1 = 32$ Bytes.

SET OF (blau, grün, gelb) erfordert $(3-1) \text{DIV} 8 + 1 = 1$ Byte.

A 3.1.6 Zeiger

Zeiger enthalten die Adresse (im Intel-Format, d.h. niederwertiges Byte zuerst) einer Variablen und belegen 2 Bytes.

A 3.2 Variablen-Speicherung zur Laufzeit

Bei der Speicherung von Variablen sind drei Fälle zu unterscheiden:

- a. Globale Variablen – im Haupt-Block des Programms deklariert.
- b. Lokale Variablen – in einem Unter-Block des Programms deklariert.
- c. Parameter und übergebene Werte – übergeben an und übernommen von Prozeduren bzw. Funktionen.

Die einzelnen Fälle werden im folgenden beschrieben und durch ein Beispiel in Anhang 4 verdeutlicht.

Globale Variablen

Globale Variablen sind im Bereich vom oberen Ende des Laufzeit-Stack abwärts gespeichert. Wenn der Laufzeit-Stack z.B. auf Adresse #B000 liegt und die Hauptprogramm-Variablen

```
VAR    i : INTEGER;  
       ch : CHAR;  
       x : REAL;
```

deklariert sind, wird die 2 Byte belegende (siehe vorheriger Abschnitt) Variable i auf den Adressen #B000-2 und #B000-1, d.h. #AFFF, ch (1 Byte) auf #AFFE-1, d.h. #AFFD, und x (4 Bytes) auf #AFF9, #AFFA, #AFFB sowie #AFFC abgespeichert.

Lokale Variablen

Lokale Variablen können nicht direkt über den Stack adressiert werden. Der Zugriff erfolgt über das in jedem Unter-Block neu belegte Register (IX. (IX-4) zeigt dann auf den Anfang der lokalen Block-Variablen.

Beispiel:

```
PROCEDURE test;
VAR
i (2 Bytes) liegt auf den Adressen IX-4-2 und IX-4-1, d.h. IX-6 und IX-5.
j (2 Bytes) liegt auf den Adressen IX-8 und IX-7.
```

Parameter und übergebene Werte

Werte-Parameter werden wie lokale Variablen behandelt. Je früher ein Parameter definiert oder eine lokale Variable deklariert wird, desto höher ist die jeweilige Speicheradresse. Die niedrigste Adresse eines Werte-Parameters ist jedoch im Gegensatz zu lokalen Variablen auf (IX+2) festgelegt.

Beispiel:

```
PROCEDURE test(i:REAL;j:INTEGER);
j (zuerst gespeichert) liegt auf den Adressen IX+2 und IX+3.
i liegt auf den Adressen IX+4, IX+5, IX+6 und IX+7.
```

Variablen-Parameter werden wie Werte-Parameter behandelt. Der einzige Unterschied besteht darin, daß Variablen-Parameter immer 2 Byte für ihre Adresse zugewiesen bekommen.

Beispiel:

```
PROCEDURE test(i:INTEGER;VARx:REAL);
Die Referenz-Adresse zu x liegt auf den Adressen IX+2 und IX+3, d.h. dieser Speicherplatz enthält die Adresse, unter der x gespeichert ist.
Der Wert von i befindet sich unter den Adressen IX+4 und IX+5.
Übergebene Werte von Funktionen werden oberhalb der Parameter gespeichert.
```

Beispiel:

```
FUNCTION test(i:INTEGER):REAL;
i befindet sich auf den Adressen IX+2 und IX+3. Der für zu übergebende Werte reservierte Speicherplatz befindet sich auf den Adressen IX+4, IX+5, IX+6 und IX+7.
```

ANHANG 4

Beispiel-Programme

Die folgenden Programme sollten zum besseren Verständnis von HiSoft Pascal sorgfältig studiert werden.

PROGRAMFACTOR

```
10 (*Programm zur Anwendung der Rekursion*)
20
30 PROGRAMFACTOR;
40
50 (*Dieses Programm berechnet eine arithmetische
   Reihe, dessen Element ueber die Tastatur einge-
   geben wird, mit Hilfe 1) der Rekursion und 2)
   einer annaehrenden (iterativen) Methode.*)
70
80 TYPE
90 POSINT=0..MAXINT;
100
110 VAR
120 METHOD:CHAR;
130 NUMBER:POSINT;
140
150 (*Rekursion*)
160
170 FUNCTIONRFAC(N:POSINT):INTEGER;
180
190 VARF@POSINT;
200
210 BEGIN
220 IF N>1 THEN F:= N * RFAC(N-1) (*RFAC wird N-mal
   aufgerufen*)
230 ELSEF:=1;
240 RFAC:=F
250 END;
260
270 (*Iterative Loesung*)
280
290 FUNCTIONIFAC(N:INTEGER;
```

```

0
0 VARI,F:POSINT;
0 BEGIN
0 F:=1;
0 FORI:=2TRONDOF:=F*I; (*Einfache Schleife*)
0 IFAC:=F
0 END;
0
0 BEGIN
0 REPEAT
0 WRITE ('Geben Sie die Methode (I oder R) und eine
Zahl ein')
0 READLN;
0 READ(METHOD,NUMBER);
0 IF Method = 'R'
0 THEN WRITELN
0 ELSE WRITELN
0 UNTIL NUMBER=0
0 END.

```

PROGRAMREV

```
10 {Programm zum Ausgeben von Zeilen einer Datei in
    umgekehrter Reihenfolge.
20 Das Programm zeigt die Anwendung von Zeigern,
    Records, MARK und RELEASE.}
30
40 PROGRAM ReverseLine;
50
60 TYPE elem=RECORD {Erzeuge eine verkettete Liste.}
70 next:↑elem;
80 ch:CHAR
90 END;
100 link=↑elem;
110
120 VAR prev,cur,heap:link;      {Zeiger auf 'elem'.}
130
140 BEGIN
150 REPEAT                      {Wiederholung}
160 MARK(heap);
    {weise 'heap' die Spitze des Haufens zu}
170 prev:=NIL;                {zeigt noch auf keine Variable}
180 WHILE NOT EOLN DO
190 BEGIN
200 NEW(cur);
    {Erzeuge einen neuen dynamischen RECORD}
210 READ(cur↑ch);             {und weise seiner Komponente
220                             ein Zeichen der Datei zu.}
230 cur↑next:=prev;
    {dieser Komponentenzeiger adressiert}
240 prev:=cur                 {den vorherigen Satz}
250 END
260
270 {gib die Reihe rueckwaerts aus, indem
280 der SET der Records rueckwaerts angegangen wird}
290
300 cur:=prev;
310 WHILE cur<>NILDO          {NIL ist das erste}
320 BEGIN
330 WRITE(cur↑.ch);
    {schreibe die Komponente bzw. das Zeichen.}
340 cur:=cur↑.next {adressiere vorherige Komponente.}
350 END;
360 WRITELN;
370 RELEASE(heap);
    {gib den Bereich fuer die dyn. Variablen frei}
380 READLN                    {warte auf neue Zeile}
390 UNTIL FALSE              {druecke [ESC] zum Abbrechen}
400 END.
```


PROGRAM TINTOUT

Dieses Programm zeigt die Anwendung von TIN und TOUT. Es erstellt ein einfaches Telefon-Verzeichnis auf der Cassette und liest es wieder ein.

PROGRAM TINTOUT;

CONST

Size=10;

TYPE

Entry=RECORD

Name:ARRAY[1..10]OFCHAR;

Number:ARRAY[1..10]OFCHAR

END;

VAR

Directory:ARRAY[1..Size]OFENTRY;

I:INTEGER;

BEGIN

{Aufbau des Verzeichnisses..

FOR I:= 1 TO Size DO

BEGIN

WITH Directory[I] DO

BEGIN

WRITE ('Bitte Name eingeben')

READLN;

READ(Name);

WRITELN;

WRITE ('Bitte Nummer eingeben')

READLN;

READ(Number);

WRITELN

END

;

{fuer die Sicherung auf Cassette bitte}

TOUT('Directory',ADDR(Directory),SIZE(Directory))

{aufrufen. Zum erneuten Enlesen des Arrays bitte}

TIN('Directory',ADDR(Directory))

{aufrufen. Das Verzeichnis kann nun beliebig
erweitert werden.}

END.

PROGRAM DIRTY.

```
10 {Dieses Programm zeigt die Manipulation von
20 Pascal-Variablen durch Maschinen-Code und demon-
30 striert die Anwendung von PEEK, POKE, ADDR und
40 INLINE}
50 PROGRAM divmult2;
60
70 VAR r:REAL;
80
90 FUNCTION divby2(x:REAL):REAL;      {Funktion zur
                                     schnellen Division
                                     durch 2}
100
110 VARI:INTEGER;
120 BEGIN
130 i:=ADDR(x)+1;    {Zeigt auf den Exponenten von x.}
140 POKE(i,PRED(PEEK(i,CHAR)));      {Dekrementiert den
                                     Exponenten von x.}

150 {siehe Anhang 3.1.3}
160 divby2:=x
170 END;
180
190 FUNCTION multby2(x:REAL):REAL;    {Funktion zur
                                     schnellen Multiplikation
                                     mit 2}
200
210 BEGIN
220 INLINE(#DD,#34,3);    {INC (IX+3) ist der Exponent
230                       von x (siehe Anhang 3.2)}
240 multby2:=x
250 END;
260
270 BEGIN
280 REPEAT
290 WRITE ('Geben Sie bitte eine Zahl fuer r ein')
300 READ(r);              {READLN nicht erforderlich
310                       (siehe Abschnitt 2.3.1.4)}
320
330 WRITELN('r dividiert durch 2 ergibt'
           ,divby2(r):7:2);
340 WRITELN('r mit 2 multipliziert ergibt',
           ,multby2(r):7:2)
350 UNTIL r=0
360 END.
```

ANHANG 5

Schildkröten-Graphik

Das Schildkröten-Paket befindet sich unter dem Namen TURTLE auf der Rückseite Ihrer Master-Cassette.

Das Software-Paket ist in Pascal geschrieben und kann aus dem HiSoft Pascal Editor mit dem Kommando „G,,TURTLE“ geladen und an ein beliebiges Programm angehängt werden. Vor dem Schildkröten-Programm müssen jedoch ein normaler Programm-Kopf (PROGRAM-Anweisung), eine Variablen-Deklaration (VAR-Anweisung) sowie optional TYPE-, CONST- und LABEL-Deklarationen eingefügt werden. Vorgestellte Prozedur- oder Funktions-Deklarationen sind nicht erforderlich.

Das Schildkröten-Paket enthält ein Demonstrations-Programm, das mit

```
g,,TURTLE[ENTER]  
C[ENTER]
```

übersetzt und nach Beantwortung der Frage RUN? mit „y“ gestartet werden kann. Die im folgenden beschriebenen Graphik-Routinen lassen sich mit der Kommando-Sequenz

```
d10,40[ENTER]  
d1150,2320[ENTER]  
p1,1140,turtle[ENTER]
```

aus dem Demonstrations-Programm extrahieren. Im Zeilenbereich von 1150 bis 2320 befinden sich noch weitere, für Sie interessante Prozeduren und Funktionen.

Wie die meisten Schildkröten-Graphik-Implementationen erzeugt auch HiSoft Pascal eine künstliche Schildkröte, die mit sehr einfachen Befehlen auf dem Bildschirm bewegt werden kann. Dabei hinterläßt sie eine Spur in verschiedenen, wählbaren Farben, die sichtbar oder unsichtbar sein kann.

Die Ausrichtung und Position der Schildkröte wird in globalen Variablen gespeichert und bei jeder Bewegung oder Drehung derselben neu berechnet. Diese Variablen können jederzeit gelesen oder geändert werden.

Globale Variablen

heading

Diese Variable enthält den Winkel-Wert der Schildkröten-Ausrichtung. Der Winkel ist eine reelle Zahl (REAL) (in Grad) und kann mit der Prozedur TURTLE (siehe unten)

den Wert 0 gesetzt werden, was einer Ausrichtung nach Osten – also von links nach rechts – entspricht. Ein zunehmender Winkel-Wert dreht die Schildkröte gegen den Uhrersinn.

or, Ycor

ese beiden Variablen vom Typ REAL bezeichnen die momentane x, y-Koordinate der Schildkröte. Sie kann auf jeden der mit dem Graphik-Bildschirm des CPC464 möglichen 320x200 Bildpunkte gesetzt werden.

or und Ycor werden mit der Prozedur TURTLE auf Position 300 und 200 (ungefähr Bildschirm-Mitte) initialisiert.

Pen-Status

Die Variable Pen vom Typ BOOLEAN, die den momentanen Status des Pen angibt:

- Pen = TRUE – Pen aktiv (unten).
- Pen = FALSE – Pen inaktiv (oben).

Prozeduren

INK (I, C1, C2: INTEGER)

Setzt die Ink I auf die durch C1 und C2 spezifizierten Farbwerte. Bei C1=C2 bleibt die Ink-Farbe stabil, andernfalls blinkt sie in den beiden Farben.

Beispiel:

- INK (1, 12, 12); setzt Ink 1 auf die Farbe gelb.
- INK (0, 16, 21); setzt Ink 0 wechselweise (blinkend) auf die Farben rosa und limonengrün.

PAPER (I: INTEGER)

Setzt die Hintergrund-Farbe (Paper) des Bildschirms auf die der Ink I zugehörigen Farbe.

PEN (I: INTEGER)

Setzt die Pen-Farbe der Schildkröte auf die der Ink I zugehörigen Farbe.

DOWN (I: INTEGER)

Aktiviert den Pen (Pen-Status = TRUE), wodurch die Schildkröte eine Spur in der der Ink I zugewiesenen Farbe hinterläßt.

UP

Deaktiviert den Pen (Pen-Status = FALSE). Die Schildkröte zeichnet keine Linie (sinnvoll für den Übergang zwischen unterschiedlichen Graphik-Elementen).

SETHD (A:REAL)

weist der globalen Richtungsvariablen (heading) den Real-Parameter A zu und bestimmt damit die Ausrichtung der Schildkröte:

0 entspricht Osten (EAST),
90 entspricht Norden (NORTH),
180 entspricht Westen (WEST),
270 entspricht Süden (SOUTH).

SETXY (X,Y:REAL)

setzt die absolute Position der Schildkröte auf die Koordinaten-Werte (x, y). Die Prozedur überprüft die Koordinaten nicht auf Bereichsüberschreitung; dies ist die Aufgabe der Firmware.

FWD (L:REAL)

bewegt die Schildkröte (in der definierten Ausrichtung) um L Einheiten vorwärts. Eine Einheit entspricht einem Graphik-Bildpunkt (gerundet).

BACK (L:REAL)

bewegt die Schildkröte (entgegen der definierten Ausrichtung, d.h. -180) um L Einheiten rückwärts. Die Ausrichtung bleibt unverändert.

TURN (A:REAL)

ändert die Ausrichtung der Schildkröte um A Grad entgegen dem Uhrzeigersinn. Die Schildkröte wird nicht bewegt.

MODE (M:INTEGER)

versetzt den Bildschirm in den Modus M. M ist eine Integer-Zahl im Bereich 0..2 und entspricht dem Bildschirm-Modus unter BASIC.

RIGHT (A:REAL)

eine Alternative zu TURN und ändert die Ausrichtung der Schildkröte um A Grad im Uhrzeigersinn.

LEFT (A:REAL)

entspricht TURN und ist nur zur Kompatibilität mit RIGHT eingeführt worden.

ARCR (R:REAL,A:INTEGER)

definiert ein Kreissegment mit dem Radius R (gewöhnlich ist $R=0.5$), in dem sich die Schildkröte bewegen kann. Der Winkelbereich ist durch A (im Uhrzeigersinn) bestimmt.

TURTLE

initialisiert die Schildkröte. Sie befindet sich daraufhin in der Bildschirm-Mitte und ist nach Osten (heading=0) ausgerichtet. Die Hintergrund-Farbe (Paper) ist gelb und eine gezeichnete Linie blau. Da die Schildkröte nach dem Laden des Programms nicht initialisiert ist, sollte die Prozedur TURTLE an jedem Programm-Anfang aufgerufen werden.

Beispiel-Programme

In den folgenden Programmen wird vorausgesetzt, daß HiSoft Pascal und das Schildkröten-Paket (Zeilen 10 bis 1140) mit „G„ TURTLE“ geladen, sowie die Zeilen für die Graphik-Demonstration (Zeilen 1150 bis 2320) gelöscht sind.

1. KREISE

```
1 PROGRAM CIRCLES;
2 VAR I:INTEGER;

1150 BEGIN
1160 TURTLE;
1170 FOR I:=1 TO 9 DO
1180 BEGIN
    90 ARCR(0.5,360);
    80 RIGHT(40)
1210 END
1220 END.
```

2. SPIRALEN

```
1 PROGRAM SPIRALS;
2 VAR

1150 PROCEDURE SPIRALS ( L,A:REAL );
1160 BEGIN
1170 FWD(L);
1180 RIGHT(A);
1190 SPIRALS(L+1,A)
1200 END;
1210 BEGIN
1220 TURTLE;
1230 SPIRALS(9,95) (*or (9,90) or (9,121)...*)
1240 END.
```

3. BLUMEN

```
1 PROGRAM FLOWER;
2 VAR

1150 PROCEDURE PETAL (S:REAL);
1160 BEGIN
1170 ARCR(S,60);
1180 LEFT(120);
1190 ARCR(S,60);
1200 LEFT(120)
```

```
1210 END;
1220 PROCEDURE FLOWER (S:REAL);
1230 VAR I:INTEGER;
1240 BEGIN
1250 FOR I =1 TO 6 DO
1260 BEGIN
1270 PETAL(S);
1280 RIGHT(60)
1290 END
1300 END;
1310 BEGIN TURTLE;
1320 SETXY(127,60);
1330 LEFT(90); FWD(10);
1340 RIGHT(60); PETAL(0.2);
1350 LEFT(60); PETAL(0.2);
1360 SETHD(90); FWD(40);
1370 FLOWER(0.4)
1380 END.
```

Zur weiteren Vertiefung Ihrer Studien empfehlen wir Ihnen das Buch „Turtle Geometry“ von Harold Abelson und Andrea di Sessa, herausgegeben von MIT Press, ISBN 0-262-01063-1.

ANHANG 6

Aufrufe an die CPC464 Firmware

In diesem Anhang werden einige Pascal-Prozeduren und -Funktionen beschrieben, die den Aufruf verschiedener Firmware-Routinen unter Hisoft Pascal ermöglichen. Sie sollten nur die für Ihre bestimmte Anwendung benötigten Routinen auswählen und aufrufen. Die Routinen sind zum leichteren Verständnis durch Kommentare ergänzt und damit selbstdokumentierend.

Bibliotheks-Prozeduren für Firmware-Aufrufe

```
10 (* getjoy entspricht der JOY FUNCTION unter BASIC
20 Seine Parameter sind 0 oder 1 ; zurueckgegeben
30 wird - wie in Basic - ein Bit-Wert.*)
40
50 FUNCTION getjoy(stick:integer):integer;
60 BEGIN
70 user(#bb24);
80 IF stick=1 THEN ra:=rl;
90 getjoy:=ord(ra)
100 END;
110
120 (* txtinitialise initialisiert den Text-VDU
folgendermaßen:
130 -Das Text-Paper wird auf Ink 0 gesetzt.
140 -Der Text-Pen wird auf Ink 1 gesetzt.
150 -Das Text-Fenster wird auf den gesamten Bild-
schirm eingeteilt.
160 -Der Text-Cursor ist freigegeben, aber aus-
geschaltet.
170 -Der Zeichen-Schreibmodus wird auf Opaque
gesetzt.
180 -Der VDU ist freigegeben.
190 -Der Graphikzeichen-Schreibmodus ist ausge-
schaltet.
200 -Der Cursor befindet sich in der oberen linken
210 Ecke des fensters. *)
220 PROCEDURE txtinitialise;
230 BEGIN
240 user(#bb4e)
250 END;
260
270 (* txtout uebergibt ein Zeichen oder einen
0 Steuercode an den Text-VDU. Steuercodes
0 koennen z.B. zur Einstellung der Paper-Farben
00 verwendet werden. *)
310
320 PROCEDURE txtout(c:char);
330 BEGIN
340 ra:=c;
350 user(#bb5a)
360 END;
370
```

```

380 (* txdchar liest ein Zeichen von der momentanen
390   Cursor-Position. *)
400
410 FUNCTION txdchar:char;
420 BEGIN
430   user(#bb60);
440   txdchar:=ra
450 END;
460
470 (* winenable setzt die GroeÙe des momentanen
480   Text-Fensters. *)
490 PROCEDURE winenable(col1,col2,row1,row2:integer);
500 BEGIN
510   rh:=chr(col1); rd:=chr(col2);
520   rl:=chr(row1); re:=chr(row2);
530   user(#bb66)
540 END;
550
560 (* getwindow uebergibt einen Variablen-Parameter
570   mit der momentanen FenstergröÙe. *)
580
590 PROCEDURE getwindow(VAR col1,col2,row1,row2:integer);
600 BEGIN
610   user(#bb69);
620   col1:=ord(rn); col2:=ord(rd);
630   row1:=ord(rl); row2:=ord(re)
640 END;
650
660 (* clearwindow loescht das momentane Text-
670   Fenster *)
680 PROCEDURE clearwindow;
690 BEGIN
700   user(#bb6c)
710 END;
720
730 (* setcolumn setzt die horizontale Position des
740   Cursors. *)
750 PROCEDURE setcolumn(c:integer);
760 BEGIN
770   ra:=chr(c);
780   user(#bb6f)
790 END;
800

```

```

810 (* setrow setzt die vertikale Position des
820   Cursors. *)
830 PROCEDURE setrow(:r:integer);
840 BEGIN
850   ra:=chr(r);
860   user(#bb72)
870 END;
880
890 (* setcursor setzt die Cursor-Position auf die
900   angegebene Spalte und Reihe. *)
910
920 PROCEDURE setcursor(c,r:integer);
930 BEGIN
940   rh:=chr(c); rl:=chr(r);
950   user(#bb75)
960 END;
970
980 (* getcursor uebergibt die momentane Cursor-
990   Position und die Anzahl der Scroll (roll)-
1000  Operationen. Die Scroll-Anzahl ist ein
1010  relativer Wert und wird beim Abwaerts-Scroll
1020  inkrementiert und beim Aufwaerts-Scroll
   dekrementiert. *)
1030 PROCEDURE getcursor(VAR col,row,roll:integer);
1040 BEGIN
1050   user(#bb78);
1060   col:=ord(rh); row:=ord(rl); roll:=ord(ra)
1070 END;
1080
1090 (* curenable gibt den Cursor frei. Der Cursor
1100   wird nur dann dargestellt, wenn er freige-
1110   geben und eingeschaltet ist. *)
1120 PROCEDURE curenable;
1130 BEGIN
1140   user(#bb7b);
1150 END;
1160
1170 (* curdisable sperrt den Cursor. *)
1180
1190 PROCEDURE curdisable;
1200 BEGIN
1210   user(#bb7e)
1220 END;
1230
1240 (* curon schreibt den Cursor ein und stellt ihn
1250   dar, wenn er freigegeben ist. *)
1260 PROCEDURE curon;
1270 BEGIN
1280   user(#bb81)
1290 END;

```

```

1300
1310 (* curoff schaltet den Cursor aus. *)
1320
1330 PROCEDURE curoff;
1340 BEGIN
1350   user(#bb84)
1360 END;
1370
1380
1390 (* txtsetpen setzt die Pen-Ink fuer das
1400   Schreiben von Zeichen. *)
1410 PROCEDURE txtsetpen(ink:integer);
1420 BEGIN
1430   ra:=chr(ink);
1440   user(#bb90)
1450 END;
1460
1470 (* txtgetpen uebergibt die momentane Text-Pen-
1480   Ink. *)
1490 FUNCTION txtgetpen:integer;
1500 BEGIN
1510   user(#bb93);
1520   txtgetpen:=ord(ra)
1530 END;
1540
1550 (* txtsetpaper setzt die Ink fuer den Text-
1560   Hintergrund. *)
1570 PROCEDURE txtsetpaper(ink:integer);
1580 BEGIN
1590   ra:=chr(ink);
1600   user(#bb96)
1610 END;
1620
1630 (* gettxtpaper uebergibt die Ink des Text-Hinter-
1640   grunds. *)
1650 FUNCTION gettxtpaper:integer;
1660 BEGIN
1670   user(#bb99);
1680   gettxtpaper:=ord(ra)
1690 END;
1700
1710 (* txtinverse vertauscht die momentanen Pen- und
1720   Text-Inks. *)
1730 PROCEDURE txtinverse;
1740 BEGIN
1750   user(#bb9c)
1760 END;
1770

```

```

1780 (* txtsetback schaltet in den Transparent-
1790   Modus, wenn der Wert seines Parameters "wahr"
1800   ist. Ist der Wert seines Parameters "falsch",
1810   wird der den Hintergrund beschreibende
1820   Opaque-Modus ausgewaehlt. *)
1830 PROCEDURE txtsetback(b:boolean);
1840 BEGIN
1850   ra:=chr(ord(b));
1860   user(#bb9f)
1870 END;
1880
1890 (* txtgetback uebergibt den Wert "wahr", wenn
1900   der Transparent-Modus ausgewaehlt und
1910   "falsch", wenn der Opaque-Modus eingeschaltet
1920   ist. *)
1920 FUNCTION txtgetback:boolean;
1930 BEGIN
1940   user(#bba2);
1950   txtgetback:= ra = chr(1)
1960 END;
1970
1980 (* txtgetmatrix uebergibt die Adresse der
1990   Zeichenmatrix, die dem als Parameter angege-
2000   benen Zeichen entspricht. *)
2010 FUNCTION txtgetmatrix(c:char):integer;
2020 BEGIN
2030   ra:=c;
2040   user(#bba5);
2040   txtgetmatrix:=rhl
2060 END;
2070
2080 (* txtsetmatrix legt die Zeichenmatrix fuer das
2090   Zeichen unter der Adresse adr ab (adr und C
2100   sind Parameter!). *)
2110 PROCEDURE txtsetmatrix(c:char;adr:integer);
2120 BEGIN
2130   ra:=c;
2140   rhl:=adr;
2150   user(#bba8)
2160 END;
2170

```

```

2180 (* setmtable setzt die durch den Anwender
2190   definierte Zeichenmatrix-Adresse (Parameter
2200   addr). Der Parameter C gibt den Offset, be-
2210   zogen auf den Tabellenanfang, an. Wenn sich
2220   der Parameter C nicht im Bereich von 0 bis
2230   255 befindet, wird die Zeichentabelle als
2240   "leer" interpretiert. Normalerweise wird ein
2250   Array zum Speichern der Matrizen und addr zum
2260   Uebergeben der Tabellenadresse verwendet. *)
2270 PROCEDURE setmtable(c,adr:integer);
2280 BEGIN
2290   rde:=c; rhl:=adr;
2300   user(#bbab)
2310 END;
2320
2330 (* txtstrselect waehlt eine im Bereich 0 - 7
2340   liegende Kanal-Nummer aus. Die meisten Attri-
2350   bute des Text-VDU koennen unabhaengig vom Zu-
2360   stand der Kanale gesetzt werden. *)
2370 PROCEDURE txtstrselect(s:integer);
2380 BEGIN
2390   ra:=chr(s);
2400   user(#bbb4)
2410 END;
2420
2430 (* txtswapstreams vertauscht die Bezeichner der
2440   beiden Kanale. *)
2450 PROCEDURE txtswapstreams(s1,s2:integer);
2460 BEGIN
2470   rb:=chr(s1); rc:=chr(s2);
2480   user(#bbb7)
2490 END;
2500
2510 (* graintialise initialisiert den Graphik-VDU
2520   folgendermaßen:
2530   - Setzt das Graphik-Paper auf Ink 0.
2540   - Setzt den Graphik-Pen auf Ink 1.
2550   - Setzt den Anwenderkoordinaten-Ursprung in
2560   die untere linke Ecke des Bildschirms.
2570   - Verschiebt die momentane Position auf den
2580   Anwenderkoordinaten-Ursprung
2590   - Setzt das Graphik-Fenster auf den gesamten
2600   Bildschirm.
2610   - Das Graphik-Fenster wird nicht geloescht.*)
2620 PROCEDURE graintialise;
2630 BEGIN
2640   user(#bbba)
2650 END;
2660
2670
2680
2690
2700

```

```

2640 (* gramoveabsolute setzt die momentane Position
2650   auf die durch die Parameter gegebene absolute
2660   Position. *)
2670 PROCEDURE gramoveabsolute(x,y:integer);
2680 BEGIN
2690   rde:=x; rhl:=y;
2700   user(#bbc8)
2710 END;
2720
2730 (* gramoverelative verschiebt die momentane
2740   Position relativ zur derzeitigen Position. *)
2750
2760 PROCEDURE gramoverelative(x,y:integer);
2770 BEGIN
2780   rde:=x; rhl:=y;
2790   user(#bbc3);
2800 END;
2810
2820 (* graaskcursor uebergibt die momentane Graphik-
2830   Position durch ihre Variablen-Parameter. *)
2840
2850 PROCEDURE graaskcursor(VAR x,y,:integer);
2860 BEGIN
2870   user(#bbc6);
2880   x:=rde; y:=rhl
2890 END;
2900
2910 (* grasetorigin setzt die Position des Anwender-
2920   koordinaten-Ursprungs und bewegt die momen-
2930   tane Position dorthin. *)
2940 PROCEDURE grasetorigin(x,y:integer);
2950 BEGIN
2960   rde:=x; rhl:=y;
2970   user(#bbc9)
2980 END;
2990
3000 (* gragetorigin uebergibt die Position des An-
3010   wenderkoordinaten-Ursprungs. *)
3020 PROCEDURE gragetorigin(VAR x,y:integer);
3030 BEGIN
3040   user(#bbcc);
3050   x:=rde; y:=rhl
3060 END;
3070

```

```

3080 (* grawinwidth setzt die rechte und linke
3090   Begrenzung des Graphik-Fensters. *)
3100
3110 PROCEDURE grawinwidth(x1,x2:integer);
3120 BEGIN
3130   rde:=x1; rhl:=x2;
3140   user(#bbcf)
3150 END;
3160
3170 (* grwinheight setzt die obere und untere
3180   Begrenzung des Graphik-Fensters. *)
3190
3200
3210 PROCEDURE grawinheight(y1,y2:integer);
3220 BEGIN
3230   rede:=y1; rhl:=y2;
3240   user(#bbd2)
3250 END;
3260
3270 (* gragetwidth uebergibt die linke und rechte
3280   Begrenzung des Graphik-Fensters. *)
3290
3300 PROCEDURE gragetwidth(VAR x1,x2:integer);
3310 BEGIN
3320   user(#bbd5);
3330   x1:=rde; x2:=rhl
3340 END;
3350
3360 (* gragetwheight uebergibt die untere und obere
3370   Begrenzung des Graphik-Fensters. *)
3380
3390 PROCEDURE gragetwheight(VAR y1,y2:integer);
3400 BEGIN
3410   user(#bbd8);
3420   y1:=rede; y2:=rhl
3430 END;
3440
3450 (* graclearwindow loescht das Graphik-
3460   Fenster. *)
3470 PROCEDURE graclearwindow;
3480 BEGIN
3490   user(#bbdb)
3500 END;
3510

```



```

3520 (* grasetpen setzt die Graphik-Pen-Ink (zur
3530   Darstellung von Punkten und Linien. *)
3540
3550 PROCEDURE grasetpen(ink:integer);
3560 BEGIN
3570   ra:=chr(ink);
3580   user(#bbde)
3590 END;
3600
3610 (* gragetpen uebergibt die momentane Einstel-
3620   lung der Graphik-Pen-Ink. *)
3630 FUNCTION gragetpen:integer;
3640 BEGIN
3650   user(#bbe1);
3660   gragetpen:=ord(ra)
3670 END;
3680
3690 (* grasetpaper setzt die Hintergrund-Ink fuer
3700   Graphik. *)
3710
3720 PROCEDURE grasetpaper(ink:integer);
3730 BEGIN
3740   ra:=chr(ink);
3750   user(#bbe4)
3760 END;
3770
3780 (* gragetpaper uebergibt die momentane Einstel-
3790   lung der Hintergrund-Ink fuer Graphik. *)
3800 FUNCTION gragetpaper:integer;
3810 BEGIN
3820   user(#bbe7);
3830   gragetpaper:=ord(ra)
3840 END;
3850
3860 (* graplotabsolute stellt mit dem Graphik-
3870   Schreibmodus einen Punkt auf einer absoluten
3880   Position in der momentanen Graphik-Pen-Ink
3890   dar. *)
3900 PROCEDURE graplotabsolute(x,y:integer);
3910 BEGIN
3920   rde:=x;rhl:=y;
3930   user(#bbea)
3940 END;
3950

```

```

3960 (* graplotrelative stellt mit dem Graphik-
3970   Schreibmodus einen Punkt relativ zur momenta-
3980   nen Position in der derzeitigen Graphik-Pen-
3990   Ink dar. *)
4000 PROCEDURE graplotrelative(x,y:integer);
4010 BEGIN
4020   rde:=x; rhl:=y;
4030   user(#bbbed)
4040 END;
4050
4060
4070 (* gratestabsolute verschiebt die momentane
4080   Graphik-Position und uebergibt den Wert der
4090   eingestellten Ink. *)
4100 FUNCTION gratestabsolute(x,y:integer):integer;
4110 BEGIN
4120   rde:=x; rhl:=y;
4130   user(#bbf0);
4140   gratestabsolute:=ord(ra)
4150 END;
4160
4170 (* gratestrelative verschiebt den Graphik-Cursor
4180   relativ zur momentanen Position und uebergibt
4190   den Wert der auf der neuen Position einge-
4200   stellten Ink. *)
4210 FUNCTION gratestrelative(x,y:integer):integer;
4220 BEGIN
4230   rde:=x; rhl:=y;
4240   user(#bbf3);
4250   gratestrelative:=ord(ra)
4260 END;
4270
4280 (* gralineabsolute zeichnet im Graphik-Modus
4290   eine Linie in der derzeitigen Graphik-Ink
4300   und verschiebt dabei die momentane Position
4310   auf den spezifizierten Endpunkt. *)
4320 PROCEDURE gralineabsolute(x,y:integer);
4330 BEGIN
4340   rde:=x; rhl:=y;
4350   user(#bbf6)
4360 END;
4370

```

```

380 (* gralinerrelative zeichnet im Graphik-Modus
390     eine Linie in der derzeitigen Graphik-Ink und
400     verschiebt dabei die momentane Position auf
410     den spezifizierten relativen Endpunkt. *)
420 PROCEDURE gralinerrelative(x,y:integer);
430 BEGIN
440     rde:=x; rhl:=y;
450     user(#bbf9)
460 END;
470
480 (* grawrchar schreibt ein Zeichen auf der momen-
490     tanen Graphik-Position und bewegt die Gra-
500     phik-Position um ein Zeichen nach rechts. *)
510
520 PROCEDURE grawrchar(c:char);
530 BEGIN
540     ra:=c;
550     user(#bbfc)
560 END;
570
580 (* scrinitialise reinitialisiert das Bildschirm-
590     Paket, den Modus und die Inks. *)
600
610 PROCEDURE scrinitialise;
620 BEGIN
630     user(#bbff)
640 END;
650
660 (* scrsetoffset setzt den Offset des ersten
670     Bildschirm-Zeichens. Durch Änderung dieses
680     Offsets kann der Bildschirminhalt hardware-
690     maessig gescrollt werden. *)
700 PROCEDURE scrsetoffset(ink:integer);
710 BEGIN
720     rhl:=ink;
730     user(#bc05)
740 END;
750
760 (* scrgetlocation uebergibt den momentanen Off-
770     et
780     des ersten Bildschirm-Zeichens. *)
790 FUNCTION scrgetlocation:integer;
800 BEGIN
810     user(#bc0b);
820     scrgetlocation:=rhl
830 END;
840

```

```

4850 (* scrsetmode setzt einen neuen Bildschirm-Modus
4860      loescht den Bildschirm und stellt das Fenster
4870      auf den gesamten Bildschirm ein. *)
4880 PROCEDURE scrsetmode(m:integer);
4890 BEGIN
4900   ra:=chr(m);
4910   user(#bc0e)
4920 END;
4930
4940 (* scrgetmode uebergibt den momentanen Bild-
4950      schirm-Modus. *)
4960 FUNCTION scrgetmode:integer;
4970 BEGIN
4980   user(#bc11);
4990   scrgetmode:=ord(ra)
5000 END;
5010
5020 (* scrclear loescht den Bildschirm auf Ink 0. *)
5030
5040 PROCEDURE scrclear;
5050 BEGIN
5060   user(#bc14)
5070 END;
5080
5090 (* scrcharlimits uebergibt die letzte Bild-
5100      schirmspalte und -reihe des momentanen Bild-
5110      schirm-Modus. *)
5120 PROCEDURE scrcharlimits(VAR col,row:integer);
5130 BEGIN
5140   user(#bc17);
5150   col:=ord(rb); row:=ord(rc)
5160 END;
5170
5180 (* scrsetink setzt die Ink-Farben. Wenn die
5190      beiden Farben gleich sind bleibt die Ink-
5200      Farbe stabil, andernfalls wechselt sie
5210      zwischen den beiden Farben. *)
5220
5230 PROCEDURE scrsetink(ink,col1,col2:integer);
5240 BEGIN
5250   ra:=chr(ink); rb:=chr(col1); rc:=chr(col2);
5260   user(#bc32)
5270 END;
5280

```

```

5290 (* scrgetink uebergibt die beiden Ink-Farben. *)
5300
5310
5320 PROCEDURE scrgetink(VAR col1,col2:integer);
5330 BEGIN
5340   user(#bc35);
5350   col1:=ord(rb); col2:=ord(rc)
5360 END;
5370
5380 (* scrsetborder setzt die beiden Bildschirmrand-
5390   Farben. Wenn die beiden Farben gleich sind,
5400   bleibt die Bildschirmrand-Farbe stabil. *)
5410
5420 PROCEDURE scrsetborder(col1,col2:integer);
5430 BEGIN
5440   rb:=chr(col1); rc:=chr(col2);
5450   user(#bc38)
5460 END;
5470
5480 (* scrgetborder uebergibt die beiden Bildschirm
5490   rand-Farben. *)
5500
5510 PROCEDURE scrgetborder(VAR col1, col2:integer);
5520 BEGIN
5530   user(#bc3b);
5540   col1:=ord(rb); col2:=ord(rc)
5550 END;
5560
5570 (* scrsetflashing setzt die Zeitperiode (Blink-
5580   periode), in der jede der beiden Ink- und
5590   Bildschirmrand-Farben auf dem Bildschirm er-
5600   scheint. Diese Einstellung gilt fuer alle
5610   Inks und den Bildschirmrand. Die Blinkperiode
5620   wird in 1/50 - bzw. 1/60 (in den USA) - Ein-
5630   heiten angegeben. Der Standard-Wert ist 10. *)
5640 PROCEDURE scrsetflashing(p1,p2:integer);
5650 BEGIN
5660   rb:=chr(p1); rc:=chr(p2);
5670   user(#bc3e)
5680 END;
5690
5700 (* scrgetflashing uebergibt die oben beschriebe-
5710   ne, momentane Blinkperiode. *)
5720
5730 PROCEDURE scrgetflashing(VAR p1,p2:integer);
5740 BEGIN
5750   user(#bc41);
5760   p1:=ord(rb); p2:=ord(rc)
5770 END;
5780

```

```

5790 (* scrfillbox fuellt einen Zeichenbereich des
5800   Bildschirms mit einer Ink. col1 und col2
5810   sind die linke und rechte Spalte, row1 und
5820   row2 die obere und untere Reihe des zu f#l-
5830   lenden Bereichs. Die Koordinaten sind hier
        physikalische Koordinaten. *)
5840 PROCEDURE scrfillbox(ink,col1,col2,row1,row2:
        integer);
5850 BEGIN
5860   ra:=chr(ink);
5870   rh:=chr(col1); rd:=chr(col2);
5880   rl:=chr(row1); re:=chr(row2);
5890   user(#bc44)
5900 END;
5910
5920 (* scrcharinvert invertiert eine Zeichenposi-
5930   tion. Alle in einer Ink dargestellten Pixel
5940   dieser Zeichenposition werden in der inver-
5950   tierten Ink dargestellt. Die verwendeten Ko-
5960   ordinaten sind physikalische Koordinaten. *)
5970 PROCEDURE schrcharinvert(i1,i2,col,row:integer);
5980 BEGIN
5990   rb:=chr(i1); rc:=chr(i2);
6000   rh:=chr(col); rl:=chr(row);
6010   user(#bc4a)
6020 END;
6030
6040 (* scrhswroll bewegt den gesamten Bildschirm per
6050   Hardware um 8 Pixel (1 Zeichen) auf - oder
6060   abwaerts. Die dabei entstehende Leerzeile
6070   wird mit der Farbe des Parameters ink ge-
        fuellt. *)
6080 PROCEDURE Scrhswroll(up:boolean; ink:integer);
6090 BEGIN
6100   rb:=chr(ord(up)); ra:=chr(ink);
6110   user(#bc4d)
6120 END;
6130
6140 (* scrswroll bewegt einen Bildschirmbereich per
6150   Software um 8 Pixel (1 Zeichen) auf- oder ab-
6160   waerts. Die Bereichskoordinaten sind in
6170   physikalischen Koordinaten auszugeben. *)
6180 PROCEDURE scrswroll(up:boolean;ink,col1,col2,
        row1,row2:integer);
6190 BEGIN
6200   rb:=chr(ord(up)); ra:=chr(ink);
6210   rh:=chr(col1); rd:=chr(col2);
6220   rl:=chr(row1); re:=chr(row2);
6230   user(#bc50)
6240 END;
6250

```

```
7210 (* mcbusyprinter uebergibt den Wert "wahr", wenn
7220     der Centronics-Kanal beschaeftigt ist,
7230     andernfalls wird der Wert "falsch" zurueck-
        gegeben. *)
7240 FUNCTION mcbusyprinter:boolean;
7250 BEGIN
7260     user(#bd2e);
7270     mcbusyprinter:=odd(raf)
7280 END;
7290
```