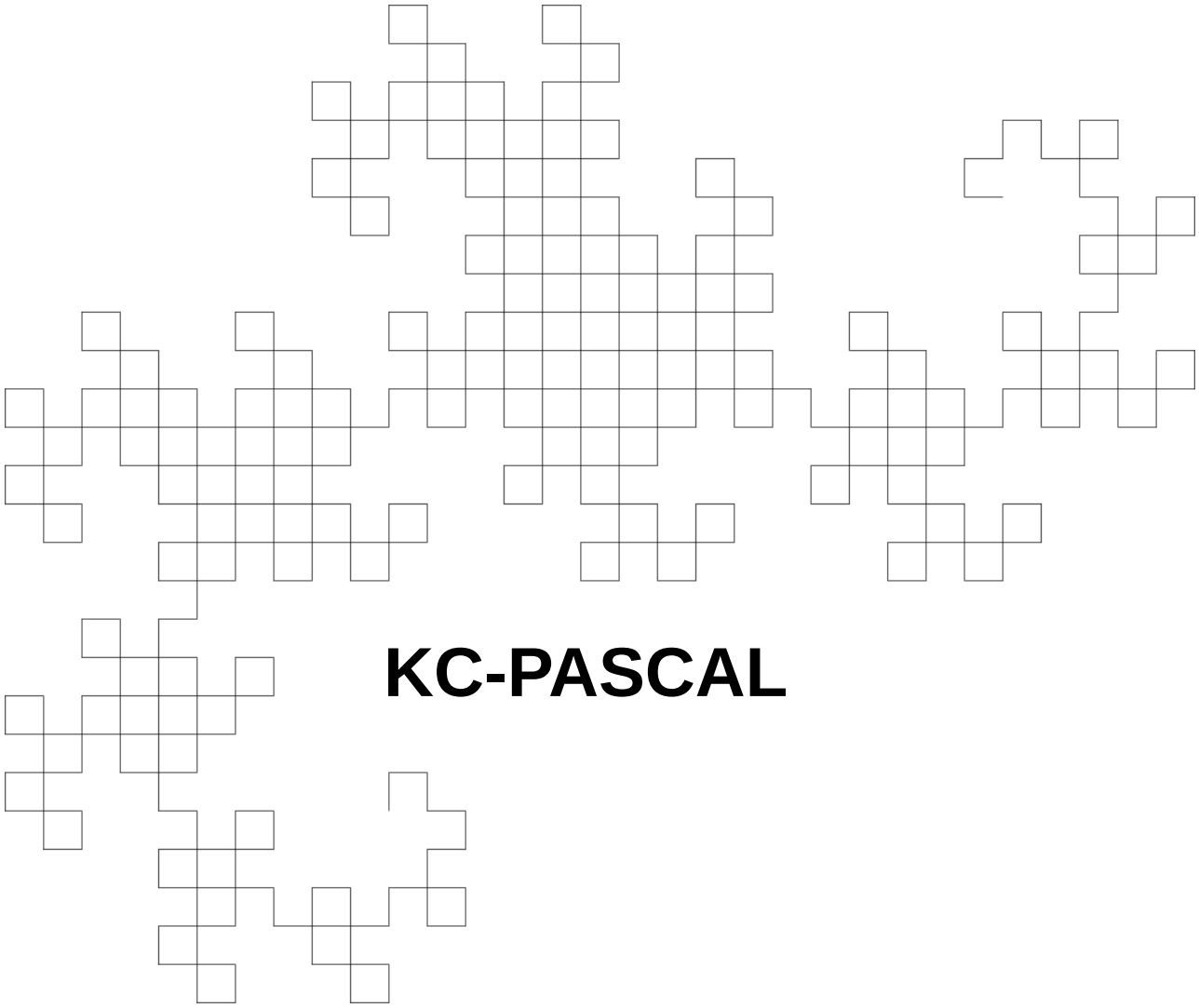


KC-PASCAL

Beschreibung KC-PASCAL

(Version 5)

Ablageort: https://github.com/duhlig/KC85_software/tree/main/_Papier/kcpascal_dok

Dokumentenversion: 1.1

Änderungshistorie

03/2025	Dietmar Uhlig	Text aus [KCPascal], Syntaxdiagramme aus [Hisoft]
05/2025	Dietmar Uhlig	Fehler in den Bildern zu Anweisung und Block korrigiert
12/2025	Dietmar Uhlig	Abschnitt zu PasEx ergänzt, Anpassungen an Version 5.2

Inhaltsverzeichnis

1 Einleitung

1.1 Zum Programm

KC-PASCAL ist eine schnelle, leicht bedienbare und leistungsfähige Pascal-Version für die Kleincomputer KC85/1, KC85/2 und KC85/3, sowie für alle CP/M-kompatiblen Systeme auf Computern wie A5120, A5130 und PC1715. Allerdings wird KC-Pascal ab Version 5.2 nur noch für die CAOS-Betriebsart der Mühlhausener KCs weiterentwickelt. Sie können jedes auf einem Kleincomputertyp erstellte Pascal-Quellprogramm auch auf einem anderen Kleincomputer einlesen und compilieren. Dadurch sind Sie in der Lage, lauffähige Maschinenprogramme für alle Kleincomputertypen zu erzeugen (s. T-Kommando).

Die Version für CP/M-kompatible Computer soll Ihnen gestatten, auf komfortableren Systemen KC-PASCAL-Programme zu erstellen, zu testen, zu drucken und zu speichern. Dabei werden alle kassettenorientierten Funktionen der Kleincomputer mittels Diskettenspeicher realisiert.

Dem vorliegenden Compiler liegt die Version HP4S (HISOFT) zugrunde. Abweichungen von der im "PASCAL User Manual and Report" (Jensen/Wirth) gegebenen Beschreibung sind:

- FILEs sind nicht implementiert, aber Variablen können auf Band gespeichert werden.
- Ein RECORD-Typ darf keinen VARIANT-Teil enthalten
- PROCEDUREs und FUNCTIONs sind nicht als Parameter zugelassen.

Gegenüber den Vorgängerversionen PASCAL V4.2 und PASCAL V4.3 wurden neue Funktionen eingebaut (Compiler Option P, Editor-Kommandos V, X und Z). Mit dem Kommando 'O' ist es jetzt möglich, Turbo-Pascal-Quellen im KC weiter zu verarbeiten, Darüber hinaus können Sie jetzt mit 32 KByte oder 48 KByte Arbeitsspeicher entwickelte und mittels Kommando T übersetzte Programme auch auf Systemen mit nur 16 Kbyte abarbeiten (ab Version 4.3; es sei denn, das COM-File ist länger). Alle System- und Fehlermeldungen erscheinen in deutscher Sprache.

Wesentliche Verbesserungen sind durch die neuen vordefinierten Funktionen und Prozeduren zur Manipulation von INTEGER-Zahlen sowie für die graphische Arbeit und durch die vordefinierte Konstante PI gegeben (PLOT, CLRPLOT, SETC, GETC, GOTOXY, KEYPRESSED, PI, SWAP, HI, LO usw.)

Weiterhin wurde der Editor verbessert und der Zufallszahlengenerator durch eine leistungsfähigere Version ersetzt.

1.2 Was gehört zu KC-PASCAL

PASENTRY.COM	KC-PASCAL Kaltstart für CP/M-kompatible Systeme (Diskette)
PASREC.COM	KC-PASCAL Warmstart für CP/M-kompatible Systeme (Diskette)

PASCAL41.COM	KC-PASCAL für KC85/1 (Kassette)
PASCAL42.COM	KC-PASCAL für KC85/2 und KC85/3 (Kassette) (wie oben, aber mit SETC, GETC, PLOT, CLRPLOT)
PASCAL43.COM	KC-PASCAL für KC85/3 (Kassette) (wie oben, aber zusätzlich LINEPLOT, CIRCLE)
PASCAL52.KCC	KC-PASCAL für KC85/3, /4 und /5 (enthält PasEx2)

1.3 Die Struktur

KC-PASCAL besteht aus verschiedenen Modulen und Tabellenbereichen. Die Reihenfolge der Programmteile und Bereiche ist nachfolgend dargestellt.

- Laufzeitmodul (Runtimes)
- Puffer für mit 'T' übersetzte Programme
- Editor
- Compiler
- Quelltext
- mittels 'C' übersetzte Programme
- Variablenstack

Der Compiler benötigt einschließlich Editor und Laufzeitmodul etwa 20 KByte Speicherplatz. Das Laufzeitmodul umfasst einschließlich der Systemanpassung und einiger Pufferbereiche ca. 5 KByte.

1.4 Laden von KC-PASCAL

Bei Verwendung eines KC85/1 ist der Einsatz eines RAM-Erweiterungsmodules 690 003.5 von 16 KByte erforderlich.

Ein weiteres RAM-Modul (insgesamt 32 KByte Zusatz-RAM) wird automatisch erkannt und kann zur Vergrößerung des Arbeitsspeichers eingesetzt werden. Der Compiler wird ab Adresse 0300H in den Arbeitsspeicher geladen.

Beim KC85/2 oder KC85/3 ist eine Speichererweiterung M022 (16 KByte) im Modulschacht 8 erforderlich. Ein weiteres Modul M022 oder ein 64-KByte-RAM-Modul kann in den Modulschacht C gesteckt werden, womit sich insgesamt 48 Kbyte Speicher nutzen lassen (etwa 1000 PASCAL-Anweisungszeilen). Das zusätzliche RAM-Modul muss vor Verwendung aktiviert werden. Dazu wird im Betriebssystem folgendes Kommando eingegeben:

```
SWITCH C 83
```

Der Compiler wird von Kassette ab Adresse 0280H in den Arbeitsspeicher des KC85/2 oder KC85/3 geladen.

Die CP/M-kompatible Version wird wie üblich durch Eingabe des Namens aufgerufen (PASENTRY, PASREC).

1.5 PasEx2

PasEx ist eine Erweiterung, um unter CAOS bis Version 4.5 neben der Kassette auch die Diskette oder ab PasEx Version 2 das Modul M052 (USB-Stick) nutzen zu können. Ab KC-PASCAL Version 5.1 wird die Erweiterung bereits zusammen mit PASCAL geladen und initialisiert. Ab CAOS Version 4.6 wird PasEx nicht mehr benötigt. Bei der Initialisierung erscheint ein Hinweis darauf, das Device-System von CAOS 4.6 und höher zu verwenden.

PasEx benutzt dieselben Routinen wie DevEx und lädt alle notwendigen Programmteile und Daten in den IRM ab Adresse 0BA00h. Um die SUTAB für USB anzupassen wird im M052-ROM nach „edas“ gesucht und dieses aufgerufen. „edas“ selbst sucht zuerst in den Modulen und bei M052-ROM ab Version 3 auch im RAM nach EDAS. Wenn vorhanden, wird es aufgerufen. Andernfalls bricht die Initialisierung ab. PasEx2 versucht, ein RAM-EDAS vorzugaukeln. Das funktioniert am besten mit einem aktuellen M052-ROM und ohne EDAS-Modul (M027, M062 oder vergleichbare). Dann merkt der Anwender von dieser Krückenlösung nichts.

Falls ein EDAS im ROM-Modul gefunden wird, startet es automatisch. Keine Panik - hier muss man die Speicherabfragen mit Enter quittieren und EDAS sofort manuell verlassen. Man kann mit KC-PASCAL ohne Einschränkungen weiterarbeiten.

Mit dem CAOS-Menüpunkt PASDEV lässt sich das aktuelle Speichergerät anzeigen und ändern. Beispiel:

```
%PASDEV
Dev 02 (0=Tape, 1=Floppy, 2=USB)
%PASDEV 1
Dev 01 (0=Tape, 1=Floppy, 2=USB)
```

Nach einem Reset sollte das M052 mit SWITCH wieder aktiviert werden. Wenn sich dann im KC-PASCAL keine Dateien laden oder speichern lassen, muss vor dem Wiedereintritt in PASCAL der USB-Stick initialisiert werden. Das geht am einfachsten, indem man eine kleine Datei speichert:

```
%usave 1 2
Name :ABFALL
02>
%PASENTRY
```

Innerhalb von KC-PASCAL lassen sich weder das Speichergerät ändern noch der Inhalt des USB-Sticks oder der Diskette anzeigen. Diese Operationen müssen auf CAOS-Ebene durchgeführt werden.

PasEx2 existiert auch als eigenständiges Programm, um einem PASCAL-Programm, das Dateioperationen ausführt, den Zugriff auf Diskette oder USB-Stick zu ermöglichen. PasEx2 sollte vor dem Pascal-Programm geladen werden, um eine Überschneidung im RAM zu vermeiden. Es liegt im Hauptspeicher ab Adresse 6000h und nach der Initialisierung im IRM (s.o.). Der RAM-Bereich ab 6000h ist nach der Initialisierung wieder frei.

1.6 Start von KC-PASCAL

Der Start des KC-PASCAL erfolgt über das Kommando PASENTRY. War der Compiler bereits gestartet und soll vorhandener Quelltext nicht gelöscht werden (z.B. nach zeitweiliger Rückkehr ins Betriebssystem), kann man mit dem Kommando PASREC (auf Betriebssystemebene) in KC-PASCAL zurückkehren.

1.7 Besondere Steuertasten

Folgende Tasten besitzen eine besondere oder abweichende Bedeutung in KC-PASCAL:

KC 85/1	**KC 85/2 /3**	**CP/M**	**Wirkung**
ENTER	ENTER	ENTER	Zeile abschließen
←	←	CTL-H	löscht letztes eingegebenes Zeichen
^	^	CTL-I	zur nächsten TAB-Position
CL LN	Shift DEL	CTL-Y	löscht gesamte Zeile
CLS	Shift HOME	CTL-L	löscht den Bildschirm
STOP	BREAK	CTL-C	unterbricht das Listen oder bricht das Programm ab
CTL-P	HOME	CTL-P	Ausgabe auf Drucker EIN/AUS
INS	Shift SPACE	[	eckige Klammer auf
DEL	¬	]	eckige Klammer zu

Alle Anweisungen müssen in Großschreibung angegeben werden. Beachten Sie bitte, dass beim KC85/2 Kleinbuchstaben zwar eingegeben werden können, aber als Großbuchstaben angezeigt werden. Der Compiler unterscheidet Namen streng nach Groß- und Kleinschreibung. Darüber hinaus werden die Zeichen '[' und ']' bei den Typen KC85/2 und KC85/3 nicht standardgemäß dargestellt.

Ein Ladeprogramm für im KC-PASCAL-Format auf Band aufgenommene Daten und Programme ist im Laufzeitmodul des KC-PASCAL enthalten.

1.8 Die Druckerschnittstelle

KC85/1

Der Drucker ist vor Aufruf zu aktivieren (s. Handbuch KC85/1). Durch Betätigen von CTL-P wird der Drucker parallel zur Bildschirmausgabe geschalten.

KC85/2, KC85/3

Ein Drucker zur Ausgabe von Programmlistings oder anderen Daten kann über einen USER-Kanal des KC85/2 oder KC85/3 erfolgen. Dafür ist ein Modul M003 erforderlich. Das Treiberprogramm entnehmen Sie bitte dem Heft "Kleincomputer KC85 – Beschreibung zu M003 V24". Dazu sollte man das Modul in Schacht "C" stecken (M022 in Schacht "8"). Das Einschalten des linken Kanals erfolgt über das Kommando (auf Betriebssystemebene):

`V24 C 1 2 <ENTER>`

Ist das Listing eines PASCAL-Programmes auf den Drucker ausgegeben worden, wird die Taste "HOME" betätigt. Nochmaliges Betätigen schaltet den Drucker wieder ab. Die Ausgabe auf dem Bildschirm bleibt erhalten. Beim Compilieren wird der Drucker immer deaktiviert (s. auch Compiler-Option P).

CP/M

Der Drucker wird wie üblich mit CTL-P aktiviert.

KC85/1, KC85/2, KC85/3, CP/M

Soll der Drucker innerhalb eines Programms ein- oder ausgeschalten werden, geschieht das mit dem Steuerzeichen 16 (10H) wie folgt:

```
…
100 WRITE(CHR(16));
110 WRITELN('Jetzt ist der Drucker eingeschaltet!');
120 WRITE(CHR(16));
130 WRITELN('...und jetzt wieder aus!');
…
```

1.9 Compilieren und Starten

Der Compiler erzeugt nach dem Aufruf ein Listing der folgenden Form:

```
aaaa nnnn Quelltext-Zeile
```

aaaa ist die Adresse, an der der aus dieser Zeile generierte Code beginnt

nnnn ist die Zeilen-Nr. (führende Nullen werden unterdrückt)

Das Listen kann durch BREAK unterbrochen werden, jede andere Taste setzt das Listen fort. Wird während des Compilierens ein Fehler festgestellt, erscheint die Meldung '*FEHLER*', gefolgt von einem Pfeil, der unter dem Fehler erzeugenden Zeichen steht, und einer Fehlernummer (siehe Fehlerliste).

Das Listen wird gestoppt, 'E' ermöglicht das Editieren der angezeigten Zeile, 'P' das Editieren der vorhergehenden Zeile (falls sie existiert), jede andere Taste setzt das Compilieren fort.

Treten während des Compilierens sehr viele Fehler auf, sollten Sie unbedingt 'E' betätigen und mit ENTER in den Editor zurückkehren. Sonst kann es vorkommen, dass der Compiler nicht wieder in den Editor "zurückfindet". Erwarten Sie bitte nicht, dass der Compiler alle Ihre Fehler exakt behandelt, sondern überprüfen Sie gegebenenfalls vorher Ihre Anweisungen!

Wird während dem Compilieren eine Taste betätigt, folgt eine Unterbrechung. Mit BREAK (CTL-C) erfolgt der Abbruch, jede andere Taste setzt den Compilerlauf fort.

Falls das Programm fehlerhaft endet (z. B. ohne 'END.'), erscheint die Meldung 'Kein Text mehr!', die Steuerung wird an den Editor übergeben.

Wenn die Compilierung erfolgreich abgeschlossen wird, das Programm aber Fehler enthält, wird die Zahl der ermittelten Fehler angezeigt und der Objekt-Code gelöscht. Nach fehlerfreier Compilierung erscheint die Frage '`Lauf?`'. Soll das Programm sofort gestartet werden, muss mit '`J`' geantwortet werden, sonst wird die Steuerung an den Editor übergeben.

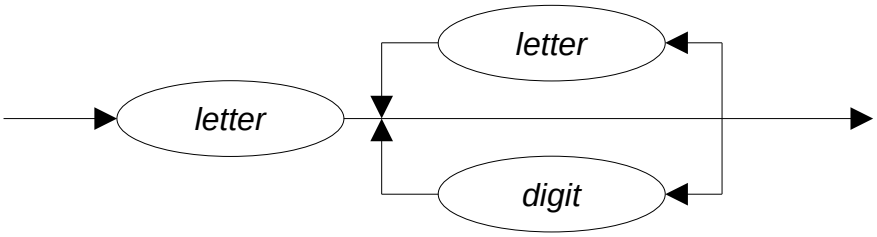
Die beim Lauf des Objekt-Codes möglichen Runtime-Fehler sind der entsprechenden Liste zu entnehmen.

Der Programmlauf kann durch eine beliebige Taste unterbrochen werden. Anschließendes BREAK bricht den Lauf ab, jede andere Taste setzt ihn fort.

2 Syntax

Soweit nicht ausdrücklich anders angegeben, entspricht die Implementierung der im "PASCAL User Manual and Report" (Jensen/ Wirth Second Edition) vorgegebenen.

2.1 Bezeichner (identifier)



Nur die ersten 10 Zeichen sind signifikant. Kleinbuchstaben werden nicht in Großbuchstaben umgewandelt, so dass z. B. die Namen HALLO, HAllo und hallo verschieden sind. Reservierte Worte und vordefinierte Namen dürfen nur in Großbuchstaben eingegeben werden.

2.2 Typisierung

PASCAL erfordert eine relative strenge, durch den Nutzer festzulegende Typisierung (Definition) der verschiedenen Datenelemente. Es gibt in verschiedenen PASCAL-Implementierungen zwei verschiedene Arten der Typisierung:

Strukturäquivalenz und Namensäquivalenz

KC-PASCAL benutzt die Namenäquivalenz für `RECORD`s und `ARRAY`s. Beispiel: Zwei Variablen sind wie folgt definiert

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

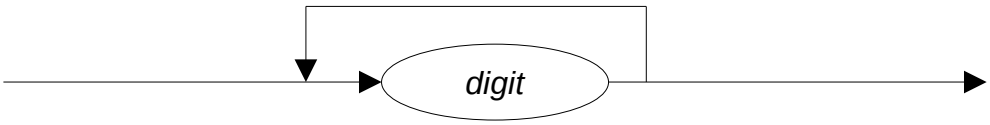
Man könnte annehmen, dass man nun `A:=B;` schreiben könnte, aber dies würde zu `'*FEHLER* 10'` führen, da durch obige Definition zwei getrennte „TYPErecords“ erzeugt werden. D.h. der Programmierer hat entschieden, dass `A` und `B` nicht demselben Datentyp angehören.

Nur ein Datentyp wird erzeugt durch:

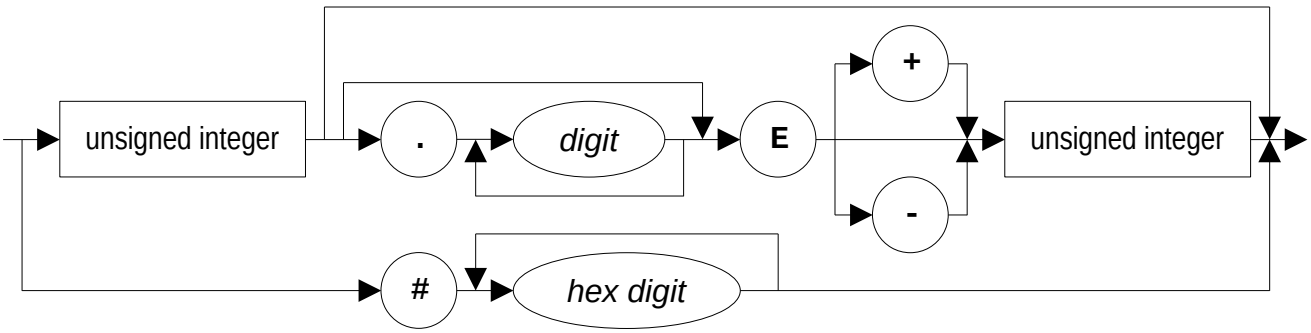
```
VAR A, B: ARRAY ['A'..'C'] OF INTEGER;
```

Jetzt kann zu `A B` zugewiesen werden und umgekehrt. Obwohl dieses Namenäquivalenzprinzip etwas komplizierter erscheint, werden durch die geforderte größere Gewissenhaftigkeit beim Programmieren die Fehler eingeschränkt.

2.3 Natürliche Zahl (unsigned integer)



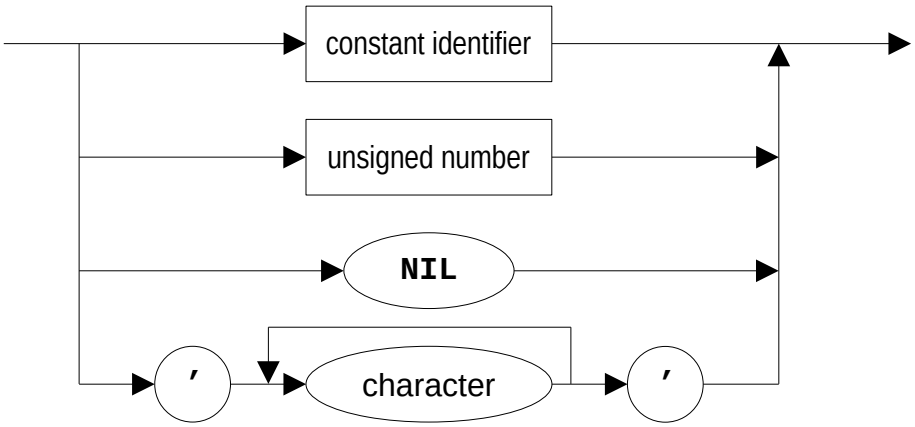
2.4 Vorzeichenlose Zahl (unsigned number)



Ganze Zahlen (`INTEGER`s) haben in KC-PASCAL einen Absolutwert kleiner gleich 32767. Größere Zahlen werden als reelle Zahlen (`REAL`s) betrachtet. Die Mantisse von `REAL`s ist 23 Bit lang. Die Genauigkeit ist demzufolge etwa 7 signifikante Stellen. Beachten Sie bitte, dass die Genauigkeit einer Berechnung abnimmt, wenn der Absolutwert des Ergebnisses viel kleiner als der der Argumente ist, z.B. liefert 2.00002-2 nicht genau 0.00002. Das rührt von der Ungenauigkeit in der Darstellung von Dezimalbrüchen als Binärbrüche her. Dieser Fall tritt nicht auf, wenn ganze Zahlen mäßiger Größe als `REAL`s dargestellt werden, z.B. wird 200002-200000 = 2 exakt berechnet. Die größte mögliche `REAL`-Zahl ist 3.4E38, die kleinste 5.9E-39. Bei der Beschreibung einer Zahl werden nur die Werte der ersten 7 Ziffern betrachtet, bei den weiteren zählt nur der Stellenwert. Führende Nullen sollten vermieden werden, da diese als Ziffer gelten und somit die Genauigkeit beeinträchtigen (z.B. ist 0.000123456 ungenauer als 1.23456E-4).

Hexadezimalzahlen sind verfügbar, um z.B. Speicheradressen darzustellen. Beachten Sie, dass nach '#' mindestens eine Hex-Ziffer stehen muss (sonst `FEHLER 51`).

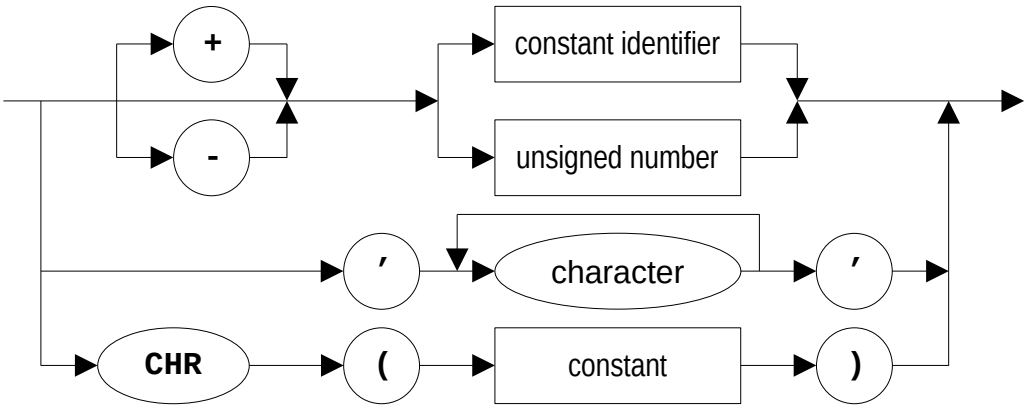
2.5 Vorzeichenlose Konstante (unsigned constant)



Beachten Sie, dass Strings (Zeichenketten) nicht mehr als 255 Zeichen enthalten dürfen. String-Typen sind `ARRAY[1..N] OF CHAR`, wobei `N` eine ganze Zahl von 1 bis 255 ist. Zeichenstrings dürfen kein End-of-line-Zeichen (`CHR (13)`) enthalten, sonst tritt '`FEHLER 68`' auf.

Der gesamte erweiterte ASCII-Zeichensatz mit 256 Elementen ist zulässig. Um die Kompatibilität mit Standard-Pascal aufrechtzuerhalten, wird das Null-Zeichen nicht als `''` dargestellt, sondern als `CHR(0)`.

2.6 Konstante (constant)



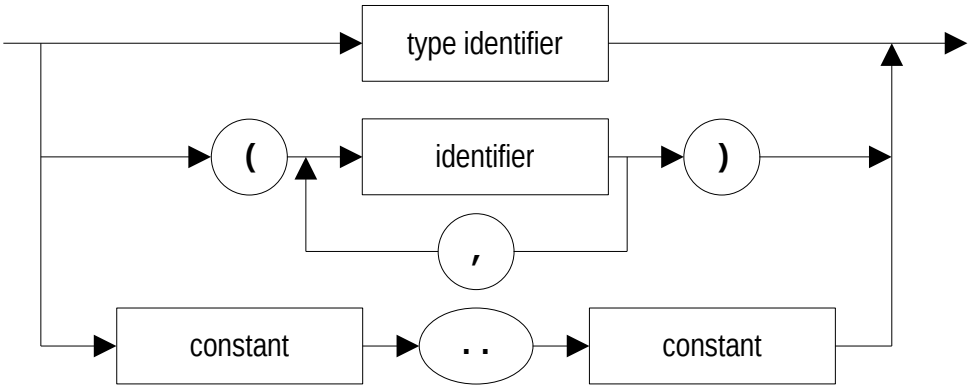
Die nicht standardgemäße `CHR`-Konstruktion ist hier zugelassen, so dass Konstanten auch als Steuerzeichen benutzt werden können. In diesem Fall muss die Konstante in den runden Klammern vom `INTEGER`-Typ sein.

z.B.:

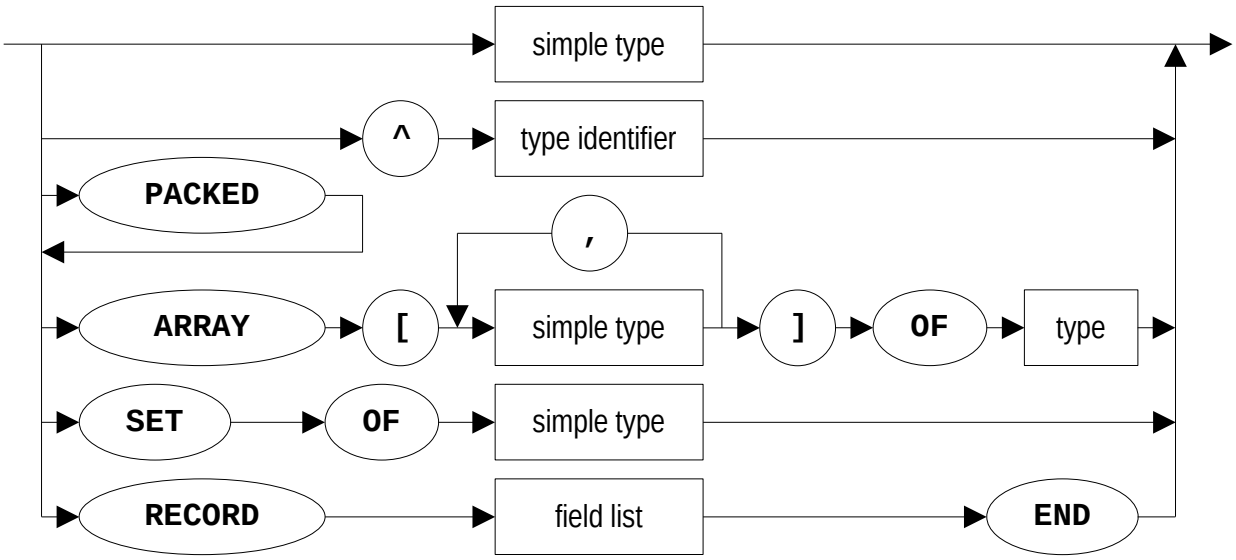
```
CONST bs=CHR(8);  (* BACKSPACE *)
      cr=CHR(13); (* CARRIGE RETURN *)
```

2.7 Einfacher Typ (simple type)

Skalare Aufzählungstypen `(name,name,...)` dürfen nicht mehr als 256 Elemente enthalten.



2.8 Typ (type)



Das reservierte Wort `PACKED` wird akzeptiert aber ignoriert, da Packen bei `ARRAYs OF CHAR` usw. schon stattfindet. Der einzige Fall, in dem Packen von Arrays vorteilhaft wäre, ist der bei `ARRAYs OF BOOLEAN`, aber in diesem Fall wird normalerweise ein SET verwendet, wenn Packen erforderlich ist.

2.8.1 Felder und Mengen

Der Grundtyp eines SETs kann bis zu 256 Elemente enthalten. Das ermöglicht SET OF CHAR zusammen mit SETs von jedem Nutzer-Aufzählungstyp zu deklarieren. Beachten Sie, dass nur Untermengen von INTERGERs als Grundtypen verwendet werden können. Alle Teilmengen von INTEGERs werden als SET OF 0..255 behandelt.

Zugelassen sind auch ARRAY OF ARRAY, ARRAY OF SET, RECORD OF SET usw.

Zwei Array-Typen werden nur dann als äquivalent betrachtet, wenn ihre Definition in ein und derselben Benutzung des reservierten Wertes ARRAY erfolgt. Folgende Typen sind somit nicht äquivalent:

```
TYPE tablea = ARRAY[1..100] OF INTEGER;
     tableb = ARRAY[1..100] OF INTEGER;
```

Eine Variable vom Typ tablea kann also nicht einer Variablen vom Typ tableb zugewiesen werden. Das ermöglicht das Ermitteln von Fehlern, wenn die beiden Tabellen verschiedene Daten darstellen. Die o.g. Einschränkung gilt nicht für den Spezialfall ARRAYs vom Stringtyp, da diese immer zur Darstellung von ähnlichen Daten verwendet werden.

2.8.2 Zeiger

KC-PASCAL erlaubt die Erzeugung dynamischer Variablen durch die Benutzung der Standardprozedur NEW. Im Gegensatz zur statischen Variablen, der ein Speicherplatz überall in dem Block zugewiesen ist, in der sie deklariert ist, kann auf die dynamische Variable nicht direkt durch einen Namen Bezug genommen werden. Diese Zeigervariable ist eine statische Variable und enthält die Adresse der dynamischen Variablen. Auf diese wird mittels eines '^' hinter der Zeigervariablen zugegriffen.

Im KC-PASCAL gibt es folgende Einschränkungen in der Benutzung von Zeigern:

Zeiger auf Typen, die nicht vereinbart wurden, sind nicht erlaubt. Das verhindert nicht die Erzeugung von verketteten Listen, da Typ-Definitionen Zeiger auf sich selbst enthalten dürfen, z.B.:

```
TYPE
  item = RECORD
         value:INTEGER;
         next:^item
         END;
  link = ^item;
```

Zeiger auf Zeiger sind nicht erlaubt.

Zeiger auf den gleichen Typ werden als äquivalent betrachtet, z.B.:

```
VAR first:link;
    current: ^item;
```

Die Variablen `first` und `current` sind äquivalent (d.h. Strukturäquivalenz wird benutzt) und dürfen einander zugewiesen oder verglichen werden.

Die vordefinierte Konstante `NIL` ist zugelassen. Wenn sie einer Zeigervariablen zugewiesen wird, wird sie als keine Adressen enthaltend betrachtet.

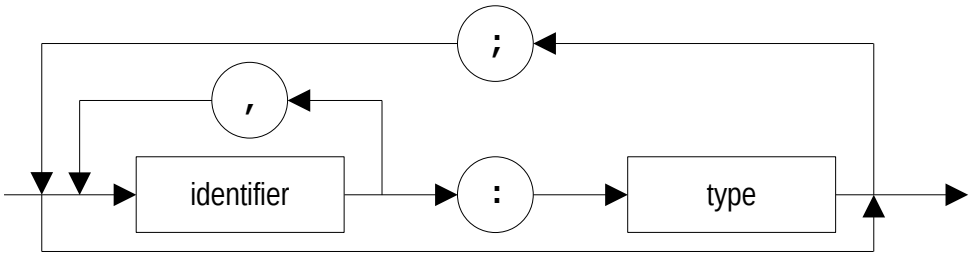
2.8.3 Records

Die Implementierung von `RECORD`s, strukturierten Variablen, die sich aus einer festen Anzahl Bestandteilen, sogenannten Feldern (fields), zusammensetzen, entspricht dem Standard-PASCAL bis auf die Einschränkung, dass ein `VARIANT`-Teil der Feldliste nicht zugelassen ist.

Zwei `RECORD`-Typen werden nur als äquivalent betrachtet, wenn ihre Deklaration zum gleichen Zeitpunkt unter dem gleichen reservierten Wort `RECORD` erfolgt.

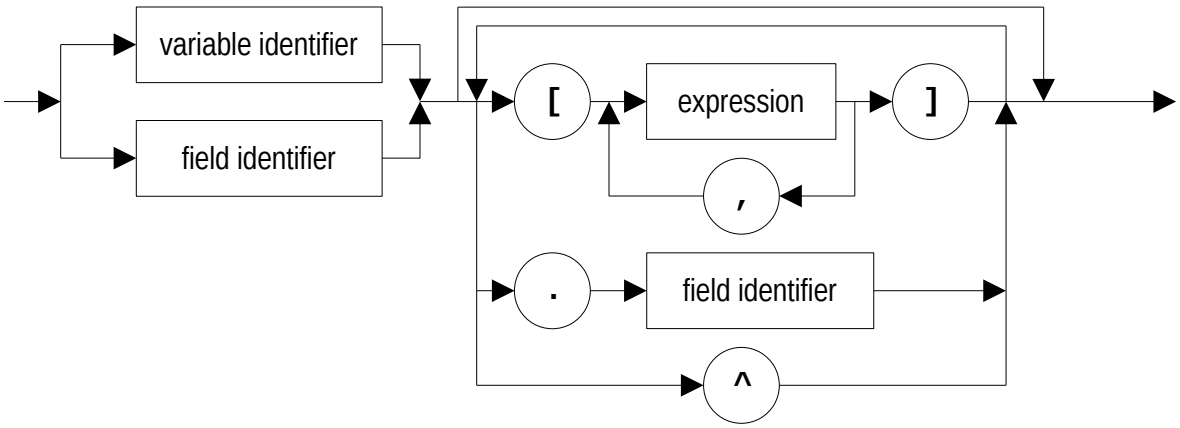
Die `WITH`-Anweisung kann verwendet werden, um den Zugriff zu den einzelnen Feldern innerhalb eines `RECORD`s in kompakterer Form zu ermöglichen.

2.9 Feldliste (field list)



Wird im Zusammenhang mit `RECORD`s verwendet.

2.10 Variable (variable)



In KC-PASCAL sind sowohl statische als auch dynamische Variable zugelassen. Statische Variable werden explizit durch `VAR` deklariert, wobei für sie im gesamten Block, in dem sie deklariert sind, Speicherplatz zugewiesen wird.

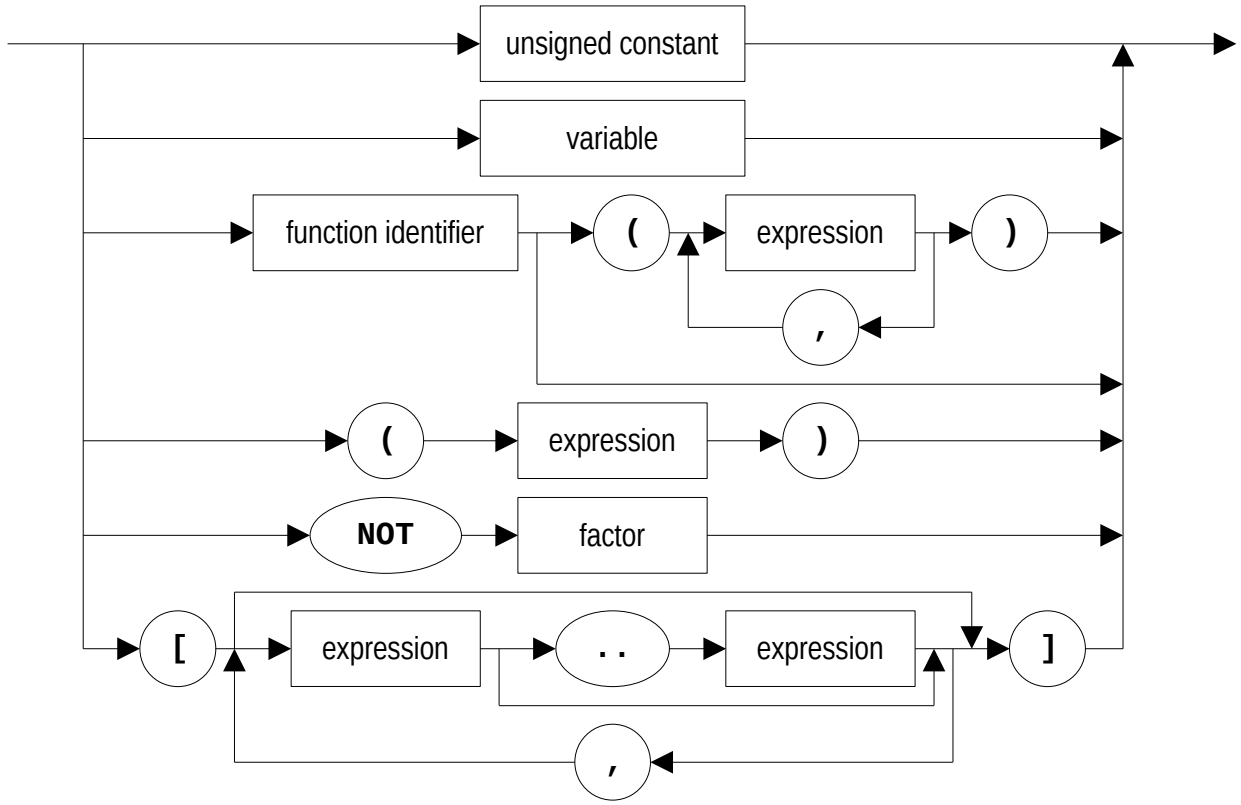
Dynamische Variable werden jedoch während der Programmierung durch die Prozedur `NEW` dynamisch erzeugt. Sie werden nicht explizit deklariert und können nicht durch einen Namen angesprochen werden.

Auf sie wird indirekt durch eine statische Variable vom Pointertyp, die die Adresse der dynamischen Variablen enthält, Bezug genommen.

Beim Beschreiben von Elementen mehrdimensionaler Arrays muss der Programmierer nicht die gleiche Art der Index-Beschreibung in der Bezugnahme verwenden, die in der Deklaration benutzt wurde.

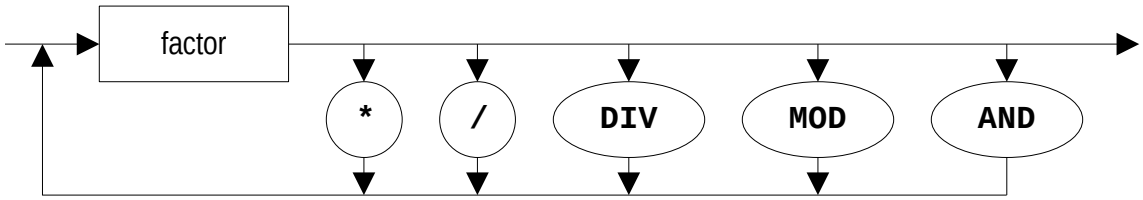
z.B.: Wenn `VAR a` als `ARRAY[1..10] OF ARRAY[1..10] OF INTEGER` deklariert wurde, kann sowohl mit `a[1][1]` als auch mit `a[1,1]` auf das Element (1,1) des Arrays zugegriffen werden.

2.11 Faktor (factor)



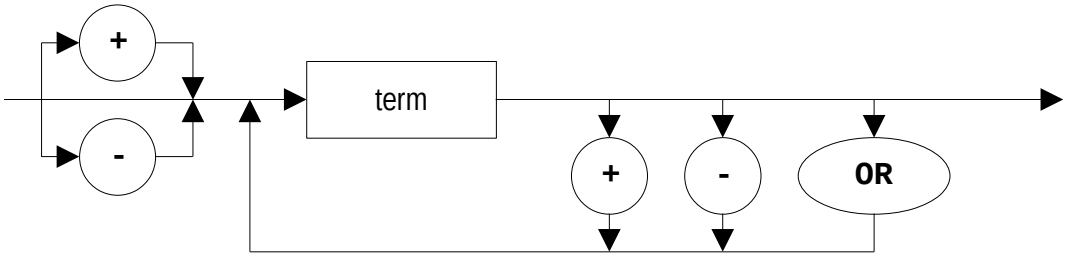
Siehe Ausdrücke und Funktionen.

2.12 Term



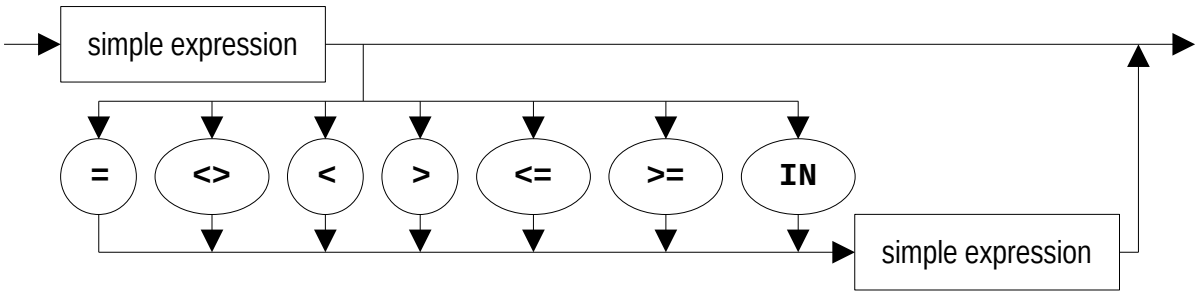
Die Untergrenze einer Menge (`SET`) ist immer Null und die Mengengröße ist immer das Maximum des Grundtyps des `SET`s. Somit belegt ein `SET OF CHAR` immer 32 Bytes (256 Elemente sind möglich – ein Bit für jedes Element). Ebenso ist ein `SET OF 0..10` äquivalent zu einem `SET OF 0..255`.

2.13 Einfacher Ausdruck (simple expression)



s. TERM

2.14 Ausdruck (expression)



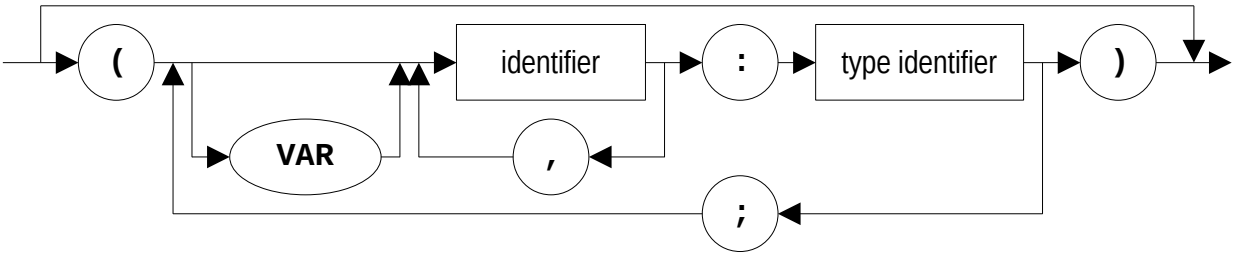
Bei der Benutzung von `IN` umfasst das `SET` immer den vollen Bereich des entsprechenden Grundtyps, nur bei `INTEGER`-Argumenten umfaßt der Grundtyp immer den Bereich [0..255].

Die obige Syntax wird beim Vergleich von Strings gleicher Länge, Zeiger und allen skalaren Typen verwendet. `SET`s können durch >=, <=, <> oder = verglichen werden, Zeiger nur durch = oder <>.

Achtung:

Vergleiche zwischen REAL- und INTEGER-Ausdrücken müssen so notiert werden, dass der REAL-Ausdruck auf der linken Seite des Vergleiches steht.

2.15 Parameterliste (parameter list)



Auf den Doppelpunkt muss ein Typname folgen, sonst tritt 'FEHLER 44' auf.

Sowohl variable Parameter als auch Festwertparameter sind ohne Einschränkungen zugelassen.

Nicht zugelassen sind Funktionen und Prozeduren.

2.16 Anweisung (statement)

siehe Abbildung

2.16.1 Zuweisungen

siehe 'TYPE'

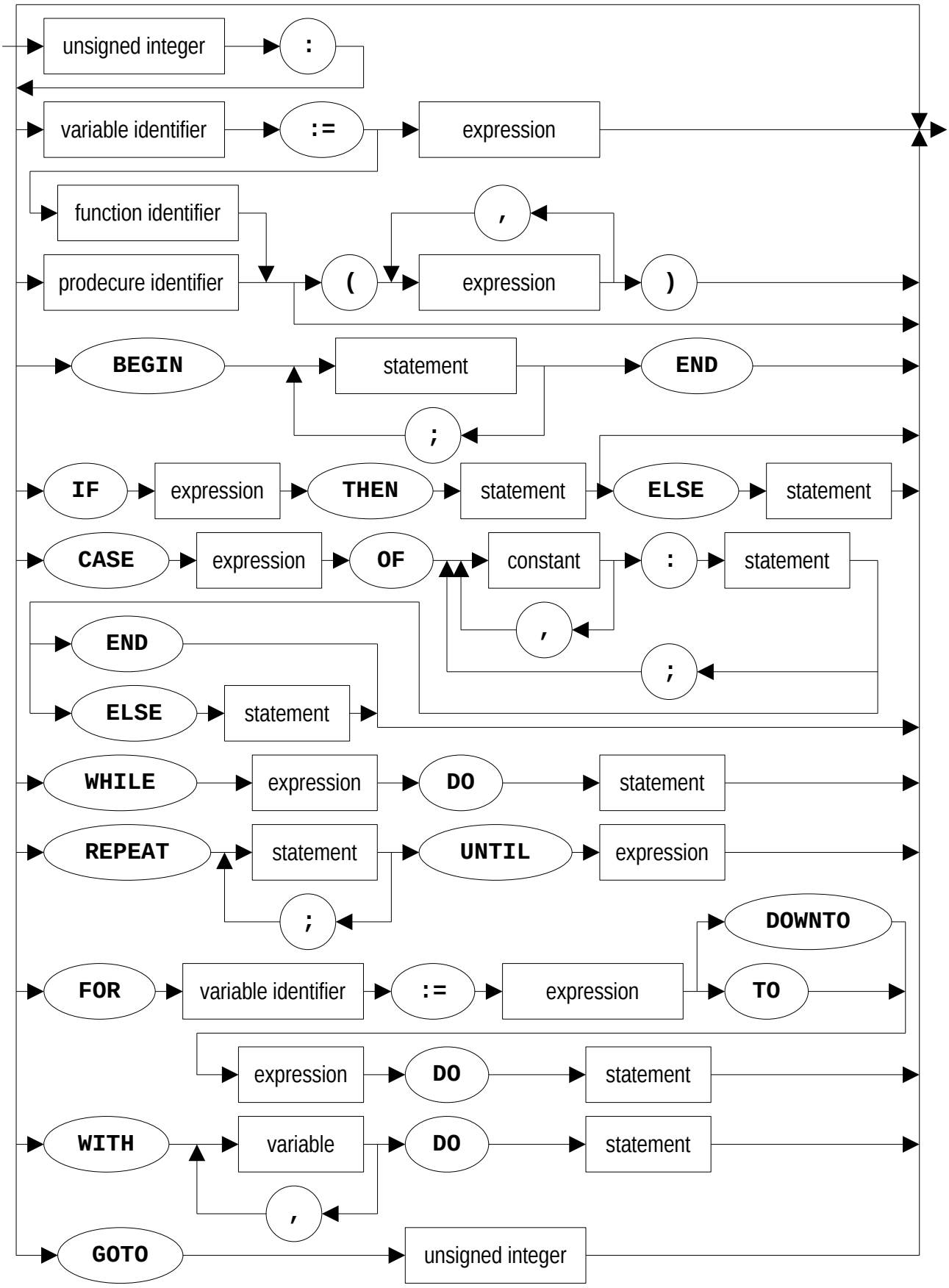
2.16.2 CASE-Anweisung

Eine leere CASE-Liste ist nicht gestattet, d. h. „CASE OF END“ ruft 'FEHLER 13' hervor.

ELSE (welches eine Alternative zu END darstellt) wird ausgeführt, wenn die geforderte CASE-Marke nicht existiert. Wenn mit END abgeschlossen und die geforderte CASE-Marke nicht gefunden wurde, setzt das Programm mit der auf END folgenden Anweisung fort.

2.17 FOR-Anweisung

Die Steuervariable einer FOR-Anweisung darf nur eine unstrukturierte Variable, kein Parameter sein.

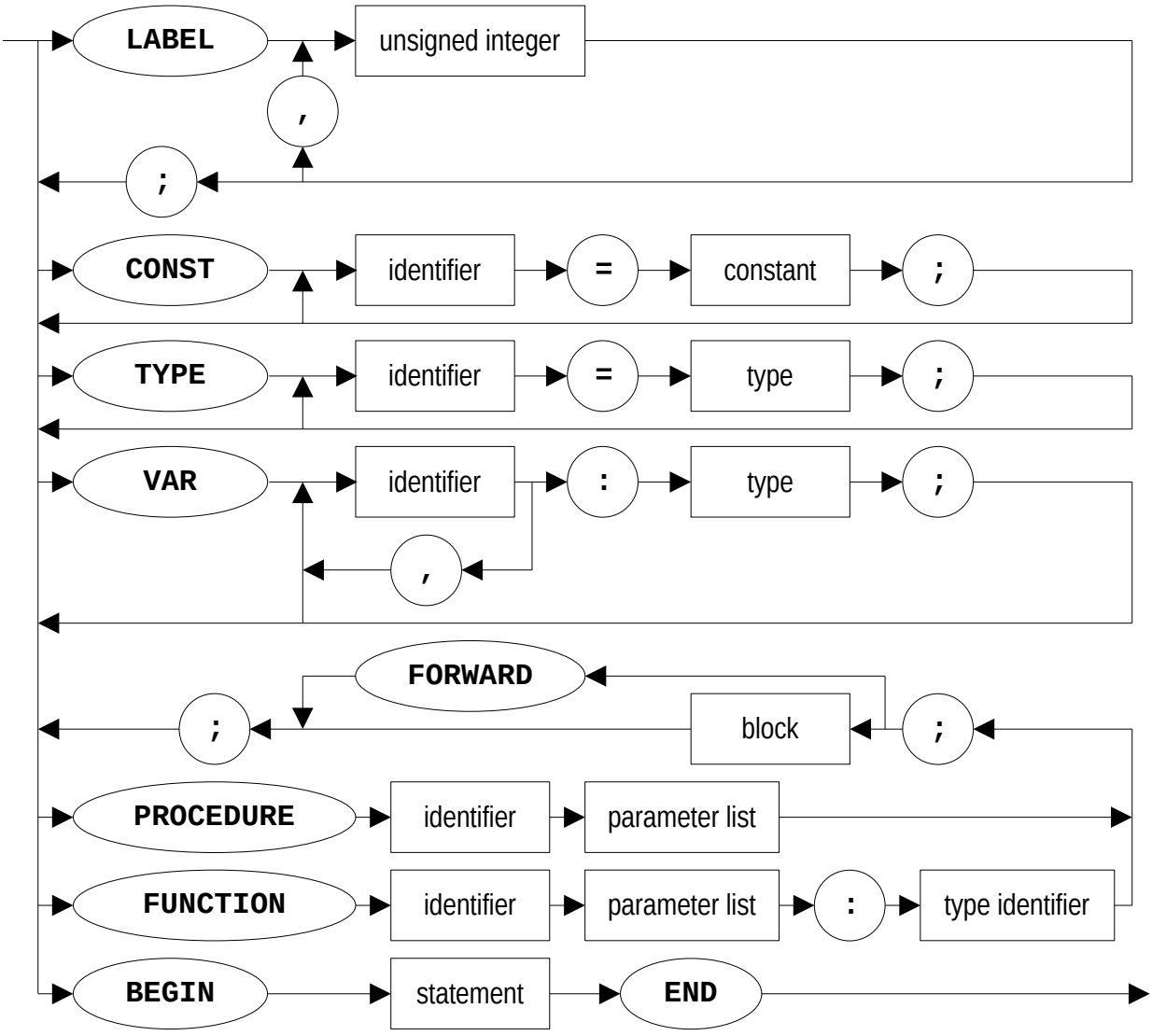

unsigned integer
:
variable identifier
:=
expression
function identifier
,
prodecure identifier
(
expression
)
BEGIN
statement
END
;
IF
expression
THEN
statement
ELSE
statement
CASE
expression
OF
constant
:
statement
,
END
;
ELSE
statement
WHILE
expression
DO
statement
REPEAT
statement
UNTIL
expression
;
DOWNTO
FOR
variable identifier
:=
expression
TO
expression
DO
statement
WITH
variable
DO
statement
,
GOTO
unsigned integer

2.18 GOTO-Anweisung

Es ist nur ein GOTO zu einer Marke möglich, die sich im selben Block und in derselben Ebene befindet.

Marken müssen durch das reservierte Wort 'LABEL' in dem Block, in dem sie benutzt werden sollen, deklariert werden. Eine Marke besteht aus 1 bis 4 Zeichen. Wenn eine Anweisung markiert werden soll, muss die Marke vor der Anweisung stehen und mit einem Doppelpunkt ':' abgeschlossen werden.

2.19 Block



2.20 Vorwärts-Bezüge

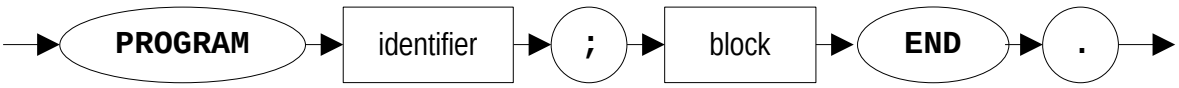
Wie im Pascal User Manual and Report beschrieben, können Prozeduren und Funktionen verwendet werden, bevor sie deklariert wurden. Das ist möglich durch das reservierte Wort FORWARD.

Beispiel:

```
PROCEDURE a(y:t); FORWARD;  (* Procedure a vorwärts deklariert *)
PROCEDURE b(x:t);           (* Procedure b *)
  BEGIN
  ....
  a(p);                     (* Bezug auf Procedure a *)
  ....
  END;
PROCEDURE a;                (* aktuelle Deklaration Procedure a *)
  BEGIN
  ....
  b(q);
  ....
  END;
```

Beachten Sie, dass die Parameter und der Typ des Ergebnisses einer Prozedur zusammen mit FORWARD deklariert werden und in der eigentlichen Deklaration der Prozedur nicht wiederholt werden.

2.21 Programme



Da keine Files implementiert sind, existieren auch INPUT und OUTPUT nicht.

2.22 Konstanten

PI	die Zahl Pi = 3.14159E+00 vom Type REAL
MAXINT	die größte verfügbare ganze Zahl, d.h. 32767
TRUE, FALSE	die Konstanten vom Typ BOOLEAN

2.23 Typen

INTEGER
REAL
CHAR
BOOLEAN

3 Prozeduren und Funktionen

3.1 Ein- und Ausgabe-Prozeduren

3.1.1 WRITE

Die Prozedur WRITE wird verwendet, um Daten auf dem Bildschirm auszugeben. Wenn der auszugebende Ausdruck vom CHARACTER-Typ ist, dann gibt WRITE (e) den 8-bit-Wert, der durch den Wert des Ausdruckes e dargestellt wird, am Bildschirm aus.

Beachte:

CHR (n) ergibt das Steuerzeichen n. Die möglichen Steuerzeichen entnehmen Sie bitte den KC85-Programmierhandbüchern.

Allgemein gilt:

WRITE (P1, P2, ... Pn); entspricht:

```
BEGIN WRITE(P1);WRITE(P2);.....;WRITE(Pn) END;
```

Die Parameter P1,P2,...Pn können eine der folgenden Formen haben:

(e) oder (e:m) oder (e:m:n) oder (e:m:H)

wobei e, m, n Ausdrücke sind und H der unmittelbare Buchstabe ist.

5 Fälle sind zu betrachten:

1) e ist von INTEGER-Typ und (e) oder (e:m) wird benutzt:

Der Wert von e wird in einen Zeichenstring mit abschließendem Leerzeichen umgewandelt. Eine Verlängerung des Strings durch führende Leerzeichen kann durch Angabe von m, welches die Gesamtlänge des Strings angibt, erreicht werden. Wenn m nicht ausreichend ist, um e auszugeben oder m nicht vorhanden ist, dann wird e vollständig mit abschließendem Leerzeichen ausgegeben und m wird ignoriert. Wenn die durch m festgelegte Länge der Länge von e ohne nachfolgenden Leerzeichen entspricht, wird kein abschließendes Leerzeichen ausgegeben.

2) e ist vom INTEGER-Typ und (e:m:H) wird benutzt:

In diesem Fall erfolgt die Ausgabe hexadezimal. Falls m=1 oder m=2 ist, wird der Wert (e MOD 16) ausgegeben, d.h. die m höchstwertigen Hex-Ziffern ausgegeben. Wenn m>4 ist, werden führende Leerzeichen hinzugefügt. Führende Nullen werden angefügt, wo es notwendig ist.

Beispiel:

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

Parameter m	Ausgabe
1	1
2	01
3	0401
4	0401
5	0401

3) `e` ist vom `REAL`-Typ und `(e)`, `(e:m)` oder `(e:m:n)` wird benutzt

Der Wert von `e` wird in einen Zeichenstring, der eine reelle Zahl darstellt, umgewandelt. Das Format der Darstellung wird durch `n` festgelegt. Falls `n` nicht vorhanden ist, wird die Zahl in wissenschaftlicher Zahlendarstellung mit Mantisse und Exponent ausgegeben. Wenn die Zahl negativ ist, wird ein Minuszeichen vor der Mantisse, anderenfalls ein Leerzeichen ausgegeben. Die Zahl wird immer mit mindestens einer Nachkommastelle und mit maximal 5 Nachkommastellen ausgegeben. Der Exponent wird immer mit Vorzeichen notiert. Daraus folgt, dass die minimale Länge der wissenschaftlichen Darstellung 8 Zeichen beträgt. Wenn `m` < 8 ist, wird die vollständige Darstellung von 12 Zeichen genommen. Wenn 8 <= `m` <= 12 ist, werden mehr oder weniger Dezimalstellen ausgegeben. Ist `m` > 12, werden führende Leerzeichen angefügt.

Beispiel:

```
WRITE(-1.23E, 10:m);
```

Parameter m	Ausgabe
7	-1.23000E+10
8	-1.2E+10
10	-1.230E+10
12	-1.23000E+10
13	-1.23000E+10

Wird die Form `(e:m:n)` benutzt, so wird die Zahl `e` in Festkommadarstellung ausgegeben, wobei `n` die Zahl der Nachkommastellen angibt. Solange die Länge `m` nicht ausreichend groß ist, werden keine führenden Leerzeichen ausgegeben. Wenn `n`=0 ist, ist die Ausgabe eine ganze Zahl. Falls `e` zu groß ist, um in dem angegebenen Feld dargestellt zu werden, erfolgt die Ausgabe im wissenschaftlichen Format (siehe oben).

Beispiel:

Quelltext	Ausgabe
WRITE(1E2:6:2)	100.00
WRITE(1E2:8:2)	100.00
WRITE(23.455:6:1)	23.5
WRITE(23.4554:2)	2.34550E+01
WRITE(23.455:4:0)	23

4) e ist vom CHARACTER- oder STRING-Typ:

Sowohl (e) als auch (e:m) können verwendet werden. Das Zeichen oder der String werden mit einer minimalen Länge von 1 (bei Zeichen) oder Länge der Strings (bei STRING-Typen) ausgegeben. Führende Leerzeichen werden angefügt, wenn m ausreichend groß ist.

5) e ist von BOOLEAN-Typ:

(e) und (e:m) können verwendet werden. 'TRUE' oder 'FALSE' werden in Abhängigkeit vom boolschen Wert e ausgegeben, wobei eine minimale Länge von 4 bzw. 5 verwendet wird.

3.1.2 WRITELN

Ausgaben mittels WRITELN schließen mit Zeilenvorschub/Wagenrücklauf ab, d.h. mit einem WRITE (CHR(13)).

WRITELN(P1,P2.....,P3); entspricht

```
BEGIN WRITE(P1,P2,...,P3);WRITELN END;
```

3.1.3 READ

Die Prozedur READ liest Daten von der Tastatur. Dies erfolgt über einen Puffer, der sich in den Runtimes befindet. Dieser ist anfangs leer (bis auf eine Zeilenende-Markierung). Man kann sich den Zugriff auf diesen Puffer so vorstellen, dass ein Textfenster über den Puffer gelegt wird, durch welches jeweils ein Zeichen sichtbar ist. Wenn dieses Textfenster über einer Zeilenende-Markierung liegt, wird vor dem Abschluß der READ-Operation eine neue Textzeile von der Tastatur in den Puffer gelesen.

READ(V1, ...,Vn); entspricht:

```
BEGIN READ(V1); READ(V2);.....;READ(Vn) END;
```

wobei V1, V2, usw. vom Typ CHARACTER, STRING, INTEGER oder REAL sein müssen.

4 Fälle sind zu betrachten:

1) V ist vom CHARACTER-Typ

In diesem Fall liest READ (V) nur ein Zeichen aus dem Eingabepuffer und weist es V zu. Wenn das Textfenster über einer Zeilenmarkierung (CHR (13)) liegt, liefert die Funktion EOLN den Wert TRUE, und eine neue Textzeile wird von der Tastatur gelesen. Wenn anschließend eine READ-Operation ausgeführt wird, wird das Textfenster über den Beginn der neuen Zeile gelegt.

Achtung! Nach dem Start des Programms ist EOLN=TRUE, d.h. falls zuerst ein READ eines CHARACTER-Typ erfolgt, wird ein CHR(13) übergeben und daraufhin eine neue Zeile von der Tastatur gelesen. Ein anschließendes READ eines CHARACTERs übergibt das erste Zeichen dieser Zeile, vorausgesetzt, sie ist nicht leer. Siehe auch unter READLN.

2) V ist vom STRING-Typ

Wird ein String mittels READ gelesen, so werden so viele Zeichen eingelesen, wie bei der Stringdefinition als Länge angegeben wurden, bzw. so viele bis EOLN=TRUE ist. Falls der String durch READ nicht gefüllt wird (d.h. falls das Zeilenende erreicht ist, bevor das Stringende erreicht ist), wird der Rest des Strings mit CHR(0) aufgefüllt. Das ermöglicht dem Programmierer die Länge des eingelesenen Strings zu ermitteln.

Die unter 1) gemachte "Achtung" - Bemerkung gilt auch hier.

3) V ist vom INTEGER-Typ

In diesem Fall wird eine Reihe von Zeichen eingelesen, die eine INTEGER-Zahl darstellen. Alle vorausgehenden Leerzeichen und Zeilenende-Markierungen werden übergangen. (Das bedeutet, dass INTEGER-Zahlen direkt eingelesen werden können). Wenn die eingelesene Zahl größer als MAXINT (32767) ist, wird der Runtime-Fehler '`Zahl zu groß`' ausgegeben und das Programm gestoppt. Wenn das erste eingelesene Zeichen (nachdem Leerzeichen und Zeilenende-Markierung übersprungen wurden) keine Ziffer oder Vorzeichen ist, wird der Fehler '`Zahl erwartet`' angezeigt und das Programm abgebrochen.

4) V ist vom REAL-Typ

Hierbei wird eine Zeichenfolge eingelesen, die eine REAL-Zahl darstellt. Alle führenden Leerzeichen und Zeilenende-Markierungen werden übergangen, und wie bei 3) muss das erste andere Zeichen eine Ziffer oder Vorzeichen sein. Wenn die Zahl zu groß oder zu klein ist, wird der Fehler '`Ueberlauf`' angezeigt, wenn '`E`' ohne nachfolgendes Vorzeichen oder Ziffer eingelesen wird, tritt Fehler '`Exponent erwartet`' auf, und wenn ein Dezimalpunkt ohne nachfolgende Ziffer gelesen wird, kommt es zum Fehler '`Zahl erwartet`'.

3.1.4 READLN

`READLN(V1,V2,....,Vn);` entspricht:

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

READLN liest einen neuen Pufferinhalt von der Tastatur. Nach der Ausführung von READLN wird EOLN = FALSE, es sei denn, die nächste Textzeile ist leer.

READLN kann verwendet werden, um die zu Beginn der Programmausführung vorhandene leere Zeile zu überspringen, d.h. es wird ein neuer Puffer gelesen. Diese Maßnahme ist nützlich, wenn zu Beginn eines Programms ein CHARACTER eingelesen werden soll, aber nicht notwendig, wenn Zahlen (da Zeilenende-Markierungen übersprungen werden) oder Zeichen von späteren Zeilen eingelesen werden sollen.

3.1.5 PAGE

Die Prozedur PAGE entspricht einem `WRITE (CHR(12));` und bewirkt ein Löschen des Bildschirmes.

3.1.6 EOLN

EOLN ist eine boolsche Funktion, die TRUE liefert, wenn das nächste zu lesende Zeichen eine Zeilenende-Markierung (CHR(13)) ist, sonst ist sie FALSE.

3.1.7 KEYPRESSED

Als Ergebnis wird eine Variable vom Typ BOOLEAN übergeben. Wenn eine Taste gedrückt wurde, ist das Ergebnis TRUE, sonst FALSE.

3.1.8 READKBD

Diese Funktion liefert ein Zeichen vom Typ CHAR von der Tastatur (wartet auf Tastenbetätigung).

3.2 Konvertierungsfunktionen

3.2.1 TRUNC (X)

Der Parameter X muss vom Typ REAL oder INTEGER sein. Der übergebene Wert ist die größte INTEGER-Zahl, die kleiner gleich X ist, wenn X positiv ist bzw. die kleinste INTEGER-Zahl, die größer gleich X ist, wenn X negativ ist.

Quelltext	Ergebnis
`TRUNC (-1.5)`	`-1`
`TRUNC (1.9)`	`1`

3.2.2 ROUND (X)

X muss vom Typ REAL sein. Die Funktion liefert die "nächstliegende" INTEGER-Zahl entsprechend den normalen Rechnungsvorschriften.

Quelltext	Ergebnis
`ROUND (-6,5)`	`-6`
`ROUND (-6.51)`	`-7`
`ROUND (11.7)`	`12`
`ROUND (23.5)`	`24`

3.2.3 ENTIER (X)

X muss REAL oder INTEGER sein. ENTIER liefert die größte INTEGER-Zahl, die kleiner gleich X ist.

Quelltext	Ergebnis
`ENTIER (-6.5)`	`-7`
`ENTIER (11.7)`	`11`

3.2.4 ORD (X)

X kann jeder skalare Typ außer REAL sein. Der übergebene Wert ist eine INTEGER-Zahl – die Ordnungszahl von X innerhalb der Menge (SET), die den Typ von X festlegt.

Wenn X vom INTEGER-Typ ist, ist ORD(x)=x; das sollte normalerweise vermieden werden.

Quelltext	Ergebnis
`ORD ('a')`	`97`

3.2.5 CHR(X)

X muss INTEGER sein. CHR liefert einen CHARACTER, der dem ASCII-Wert von X entspricht.

Quelltext	Ergebnis
`CHR (49)`	`'1'`
`CHR (91)`	`'['`

3.3 Arithmetische Funktionen

In diesem Abschnitt muss der Parameter X immer vom Typ REAL oder INTEGER sein.

3.3.1 ABS(X)

liefert den Absolutwert von X (d.h. ABS(-4.5) = 4.5) Das Ergebnis ist vom gleichem Typ wie X.

3.3.2 SQR(X)

liefert den Wert X*X, d.h. das Quadrat von X. Das Resultat ist vom gleichen Typ wie X.

3.3.3 SQRT(X)

liefert die Wurzel aus X. Das Resultat ist immer vom Typ REAL. Ein 'mathematischer Fehler' tritt auf, wenn X negativ ist.

3.3.4 FRAC(X)

liefert den gebrochenen Anteil von X.

Quelltext	Ergebnis
`FRAC (X)`	`X - ENTIER(X)`
`FRAC (1.5)`	`0.5`
`FRAC (-12.56)`	`0.44`

3.3.5 SIN(X)

liefert den Sinus von X. X wird im Bogenmaß (radian) angegeben. Das Resultat ist immer REAL.

3.3.6 COS(X)

liefert den Cosinus von X, weiter wie SIN

3.3.7 TAN(X)

liefert den Tangens von X, weiter wie SIN

3.3.8 ARCTAN(X)

liefert den Winkel im Bogenmaß, dessen Tangens gleich X ist. Das Ergebnis ist REAL.

3.3.9 EXP(X)

liefert den Wert e^X (mit e=2.71828). Das Resultat ist REAL.

3.3.10 LN(X)

liefert den natürlichen Logarithmus von X. Das Resultat ist REAL. Wenn X kleiner gleich Null ist, kommt es zum Fehler '`mathematischer Fehler`'.

3.4 Dynamischer Speicher

3.4.1 NEW(p)

Mittels NEW(p) wird dynamischen Variablen Speicherplatz zugewiesen. p ist eine Zeiger-Variable, und nachdem NEW(p) ausgeführt wurde, enthält p die Adresse der neu zugewiesenen dynamischen Variablen. Der Typ der dynamischen Variablen ist der gleiche Typ wie der der Zeiger-Variablen p, und dies kann jeder Typ sein. Um auf die dynamische Variable zuzugreifen, wird p benutzt. Um den Speicherplatz wieder freizugeben, werden die Prozeduren MARK & RELEASE benutzt (s. unten).

3.4.2 MARK(v1)

Die Prozedur sichert den Zustand der Menge ("heap") der dynamischen Variablen in der Zeigervariablen v1. Der Zustand der Menge, wie sie zum Zeitpunkt der Ausführung von MARK vorlag, kann durch die Prozedur RELEASE (s. unten) wieder hergestellt werden. Der Typ der Variablen, auf die v1 zeigt, ist gleichgültig, da v1 nur mit MARK & RELEASE verwendet werden darf, *niemals mit NEW.*

3.4.3 RELEASE(v1)

Durch diese Prozedur wird vom Ende des Bereichs der dynamischen Variablen her Speicherplatz für dynamische Variable freigegeben. Es wird wieder der Zustand hergestellt, der zum Zeitpunkt der Ausführung von `MARK(v1)` bestand, d.h. alle später erzeugten dynamischen Variablen werden zerstört.

3.5 Systemnahe Routinen

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

Diese Prozedur erlaubt das Einfügen von Z-80 Maschinencode in das Pascal-Programm. Die Werte (`C1 MOD 256`, `C2 MOD 256`,...) werden an dieser Stelle (Adresse, an der sich der Compiler gerade befindet) in das Objekt-Programm einfügt. `C1`, `C2`, `C3`, usw. sind `INTEGER`-Konstanten.

3.5.2 USER(V)

`V` ist ein `INTEGER`-Argument. Die Prozedur bewirkt den Aufruf eines Maschinenprogramms an der Adresse `V`. Da KC-PASCAL `INTEGER`-Zahlen in Zweierkomplement-Form bearbeitet, müssen Adressen, die größer als 7FFFh (32767) sind, als negative Zahlen eingegeben werden. Z.B.:

0C000H ist -16384 und somit bewirkt `USER(-16384)` einen Aufruf an der Speicherstelle 0C000H.

Wenn Konstanten verwendet werden, um Aufrufe zu erzeugen, ist es also einfacher, hexadezimale Zahlen zu verwenden. Die aufgerufene Routine muss mit einem Z-80 RET-Befehl (0C9H) enden und darf den Wert im IX-Register nicht verändern.

3.5.3 HALT

Die Prozedur stoppt die Programmausführung mit der Meldung '`Halt bei PC=XXXX`', wobei `XXXX` die hexadezimale Adresse darstellt, an der das `HALT` ausgegeben wurde. Zusammen mit dem Compilerlisting kann `HALT` verwendet werden, um festzustellen, welcher von zwei oder mehr Programmzweigen durchlaufen wurde, es wird also normalerweise bei der Fehlersuche benutzt.

3.5.4 POKE(X,V)

`POKE` schreibt den Ausdruck `V` angefangen von der Adresse `X` in den Speicher. `X` ist vom `INTEGER`-Typ und `V` kann von jedem Typ außer `SET` sein.

Beispiele:

`POKE(#6000,'A')` schreibt #41 an Adresse #6000

`POKE (-16384,3.6E3)` schreibt 00 0B 80 70 (in Hex) ab Adresse #C000

3.5.5 PEEK(X,T)

X stellt eine Speicheradresse als INTEGER-Zahl dar. T ist der Typ, den die Funktion liefern soll (jeder Typ ist möglich). PEEK wird benutzt, um Daten aus dem Speicher zu lesen. Bei allen PEEK- und POKE-Operationen werden Daten in der KC-PASCAL Darstellung gelesen bzw. geschrieben.

z. B.: Wenn der Speicher ab der Adresse #6000 die HEX-Bytes 50 61 73 63 61 6C enthält, liefern verschiedene Peeks:

Quellcode	Ergebnis
`WRITE(PEEK(#6000,ARRAY[1..6] OF CHAR))`	`'PASCAL'`
`WRITE(PEEK(#6000,CHAR))`	`'P'`
`WRITE(PEEK(#6000,INTEGER))`	`24912`
`WRITE(PEEK(#6000,REAL))`	`2.46227E+29`

3.5.6 OUT(P,C)

Diese Prozedur spricht unmittelbar den Z-80-Output-Port an, ohne dass INLINE benutzt werden muss. Der Wert des INTEGER-Parameters P wird in das BC-Register und der CHARACTER-Parameter C in das A-Register geladen, dann wird der Assembler-Befehl OUT(C), A ausgeführt.

z.B.: OUT(l,'A') gibt das Zeichen 'A' an den Z-80 Port l aus.

3.5.7 INP(P)

Mittels INP läßt sich der Z80-Input-Port (U880) direkt (ohne Benutzung von INLINE) ansprechen. Der Wert von P wird ins BC-Register geladen. Das CHARACTER-Resultat wird dann durch Ausführung des Assembler-Befehls IN A,(C) erhalten.

3.6 Datei-Routinen

3.6.1 TOUT(NAME,START,SIZE)

Mit TOUT werden Variablen auf Band gespeichert. Der NAME ist vom Typ ARRAY[1...8] OF CHAR. Es werden SIZE (=Anzahl) Bytes aus dem Speicher, angefangen von Adresse START, abgespeichert. Diese beiden Parameter sind vom Typ INTEGER. Um z.B. die Variable V unter dem Namen 'VAR' zu speichern, ist zu schreiben:

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

Die Benutzung von konkreten Speicheradressen ermöglicht dem Programmierer mehr Flexibilität als nur die Möglichkeiten, Arrays zu speichern. Wenn ein System beispielsweise mit mehreren Bildschirminhalten (memory mapped screen) arbeitet, können ganze Bildschirminhalte unmittelbar gesichert werden.

3.6.2 TIN(NAME, START)

Mit dieser Prozedur können durch TOUT gesicherte Variablen usw. wieder geladen werden. Für die Typen von NAME und START gilt das oben gesagte. Das Band wird nach dem File mit dem Namen NAME abgesucht ('?' für beliebige Zeichen ist zugelassen), welches dann ab der Adresse START eingelesen wird. Die Anzahl der zu ladenden Bytes wird vom Band entnommen (entspricht den gesicherten).

Beispiel: Um die mit TOUT gesicherte Variable zu laden schreibt man:

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

Da Quelltexte (source files), die durch den Editor gesichert wurden, das gleiche Aufzeichnungsformat haben, wie es auch von TOUT benutzt wird, können mit TIN Quelltexte in ARRAY OF CHAR geladen werden, um sie weiter zuverarbeiten.

3.7 Manipulation von INTEGER-Zahlen

3.7.1 HI(I)

Das Ergebnis der Funktion liefert den Wert des höherwertigen Byte der INTEGER-Zahl I (Ergebnistyp ist INTEGER).

3.7.2 LO(I)

Das Ergebnis dieser Funktion liefert den Wert des niederwertigen Byte der INTEGER-Zahl I (Ergebnistyp ist INTEGER)

3.7.3 SWAP(I)

Mit dieser Funktion werden High-Byte und Low-Byte der INTEGER-Zahl I vertauscht.

3.7.4 SHL(I, N)

Die INTEGER-Zahl I wird um N (Typ INTEGER) Bitstellen bitweise nach links verschoben. Das Ergebnis ist vom Typ INTEGER.

3.7.5 SHR(I, N)

Die INTEGER-Zahl I wird um N (Typ INTEGER) Bitstellen bitweise nach rechts verschoben. Das Ergebnis ist vom Typ INTEGER.

3.7.6 BAND(A, B)

Die INTEGER-Zahl A wird mit der INTEGER-Zahl B bitweise UND-verknüpft. Das Ergebnis ist vom Typ INTEGER.

3.7.7 BOR(A, B)

Die INTEGER-Zahl A wird mit der INTEGER-Zahl B bitweise ODER-verknüpft. Das Ergebnis ist vom Typ INTEGER.

3.7.8 BXOR(A, B)

Die INTEGER-Zahl A wird mit der INTEGER-Zahl B EXCLUSIV-ODER-verknüpft. Das Ergebnis ist vom Typ INTEGER.

3.8 Graphikprozeduren

3.8.1 GOTOXY(X, Y)

X und Y sind INTEGER-Variablen. Der Cursor (für WRITE und WRITELN) wird auf die Spalte X und die Zeile Y eingestellt.

3.8.2 SETC(V, W) [nur KC85/2 und KC85/3]

V und W sind INTEGER-Variablen. V bestimmt die Vordergrundfarbe (Schrift, Punkte usw.) von 0 bis 15 und W die Hintergrundfarbe von 0 bis 7 (siehe Handbücher KC85/2 und KC85/3).

3.8.3 GETC(X, Y) [nur KC85/2 und KC85/3]

X und Y sind INTEGER-Variablen. Mit dieser Funktion kann die Farbe im Punkt X,Y des Pixel-Bildspeicher bestimmt werden. Das Ergebnis der Funktion ist vom Typ INTEGER.

3.8.4 PLOT(X, Y) [nur KC85/2 und KC85/3]

X und Y sind INTEGER-Variablen. Der Bildpunkt an der Position X,Y wird entsprechend der voreingestellten Farbinformation gesetzt.

3.8.5 CLRPLOT(X, Y) [nur KC85/2 und KC85/3]

X und Y sind INTEGER-Variablen. Der Bildpunkt an der Position X,Y wird gelöscht (auf aktuelle Hintergrundfarbe gesetzt).

3.8.6 LINEPLOT(X1, Y1, X2, Y2) [nur KC85/3]

X1, X2, Y1 und Y2 sind INTEGER-Variablen. Es wird eine Linie vom Punkt X1,Y1 zum Punkt X2,Y2 mit der zuletzt eingestellten Farbe (s. SETC) gezogen.

3.8.7 CIRCLE(XM, YM, R) [nur KC85/3]

XM, YM und R sind INTEGER-Variablen. Um den Punkt XM,YM wird ein Kreis mit dem Radius R gezogen (die zuletzt eingestellte Farbe wird verwendet).

3.9 Weitere vordefinierte Funktionen

3.9.1 RANDOM

Das Ergebnis der Funktion ist eine INTEGER-Pseudo-Zufallszahl von 0 bis 255.

3.9.2 SUCC(X)

X kann jeder skalare Typ außer REAL sein. SUCC(X) liefert den Nachfolger von X.

SUCC('A') ergibt 'B'.

SUCC('5') ergibt '6'.

3.9.3 PRED(X)

PRED liefert den Vorgänger von X.

PRED('j') ergibt 'i'.

PRED(TRUE) ergibt FALSE.

3.9.4 ODD(X)

X muss vom INTEGER-Typ sein. ODD liefert das boolesche Resultat TRUE, wenn X ungerade ist und FALSE, wenn X gerade ist.

3.9.5 ADDR(V)

V ist ein Variablen-Name beliebigen Typs. Die Funktion liefert die Speicheradresse der Variablen V als INTEGER-Zahl.

3.9.6 SIZE(V)

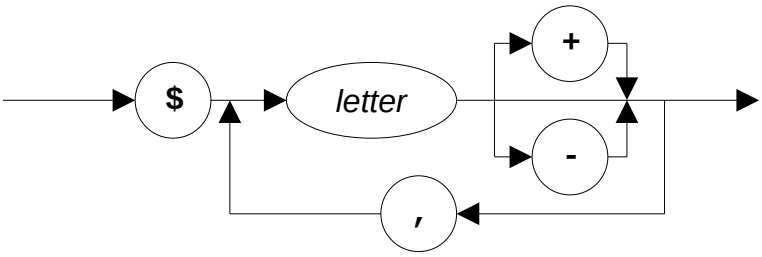
V ist eine Variable. Das INTEGER-Resultat stellt die Anzahl der Bytes dar, die durch diese Variable belegt werden.

4 Kommentare und Compiler-Steuerzeichen

4.1 Kommentare

Ein Kommentar kann überall zwischen zwei reservierten Worten, Zahlen, Namen oder Spezialsymbolen eingefügt werden. Ein Kommentar beginnt mit dem Zeichen '{' oder dem Zeichenpaar '(*'. Außer, wenn das nächste Zeichen ein '$' ist, werden alle Zeichen bis zum nächsten '}' oder '*)' ignoriert. Wird ein '$' gefunden, erwartet der Compiler eine Folge von Compiler-Steuerzeichen (s. unten). Die auf diese Steuerzeichen folgenden Zeichen werden übersprungen bis '}' oder '*)' gefunden wird.

4.2 Compiler-Steuerzeichen



Alle Compiler-Steuerungen können selektiv eingesetzt, d.h. im Programm ein- und ausgeschaltet werden. Somit können fehlerfreie Programmteile beschleunigt und zusammengedrängt werden, indem die entsprechenden Tests in unverschachtelten Programmteilen abgeschaltet werden.

4.2.1 Steuerzeichen `L`

Steuert das Listen von Programmtext und Objektcode-Adressen beim Compilieren.

Bei `L+` wird ein vollständiges Listing ausgegeben.
Bei `L-` werden nur die fehlerhaften Zeilen gelistet.

Vorgabewert ist `L+`.

4.2.2 Steuerzeichen `O`

Legt fest, ob bestimmte Überlauftests gemacht werden. `INTEGER`-Multiplikationen und Divisionen und alle arithmetischen `REAL`-Operationen werden immer auf Überlauf getestet.

Bei `O+` werden außerdem die `INTEGER`-Addition und -Subtraktion getestet.
Bei `O-` nicht.

Vorgabewert ist `O+`.

4.2.3 Steuerzeichen `C`

Legt fest, ob während der Ausführung des Object-Codes Tastaturüberprüfungen durchgeführt werden.

Bei `C+` wird die Programmausführung nach Betätigen von `EDIT` mit der Meldung '`Halt`' angehalten. Dieser Test wird am Beginn aller Schleifen, Prozeduren und Funktionen gemacht. Damit kann der Benutzer bei der Fehlersuche überprüfen, welche Schleifen usw. nicht ordnungsgemäß beendet werden (Endlosschleife).

Bei `C-` werden obige Tests nicht durchgeführt und das Programm läuft somit schneller.

Vorgabewert ist `C+`.

4.2.4 Steuerzeichen `P`

Mit diesem Steuerzeichen wird der Drucker parallel zum Bildschirm geschaltet. Wird das Steuerzeichen wiederholt verwendet, so wird der Drucker wieder ausgeschaltet. Nach dem Compilieren wird der Drucker immer deaktiviert.

4.2.5 Steuerzeichen `S`

Legt fest, ob Stack-Tests durchgeführt werden.

Bei `S+` wird am Beginn jeder Prozedur und jedes Funktionsrufes getestet, ob der Stack in diesem Block vermutlich überlaufen wird. Wenn der Stack droht, den Bereich der dynamischen Variablen oder das Programm zu zerstören, wird die Meldung '`>> RAM bei PC=XXXX`' ausgegeben und das Programm abgebrochen. Wenn innerhalb einer Prozedur ein großer Stackbereich benötigt wird, kann das Programm abstürzen. Benötigt andererseits eine Funktion sehr wenig Stack, während sie rekursiv arbeitet, ist es u.U. unnötig, diesen Test durchzuführen.

Bei `S-` wird kein Stack-Test durchgeführt.

Vorgabewert ist `S+`.

4.2.6 Steuerzeichen `A`

Legt fest, ob getestet wird, dass Feldindizes in dem durch die Felddeklaration festgelegten Bereich liegen.

Bei `A+` wird die Bereichsüberschreitung die Meldung '`Index zu hoch`' oder '`Index zu niedrig`' ausgegeben und das Programm gestoppt.

Bei `A-` wird dieser Test nicht durchgeführt.

Vorgabewert ist `A+`.

4.2.7 Steuerzeichen `I`

Bei der Benutzung der 16 bit-Zweierkomplement-`INTEGER`-Arithmetik entsteht ein Überlauf bei der Ausführung von `>`, `<`, `>=` oder `<=`, wenn sich die Argumente um mehr als `MAXINT` (32767) unterscheiden. In diesem Fall wäre das Ergebnis des Vergleichs falsch. Sollen solche Zahlen nicht verglichen werden, hat das keine weiteren Folgen. `I+` sichert aber auch bei einem solchen Vergleich ein richtiges Ergebnis. Eine ähnliche Situation kann auch beim Vergleich von `REAL`-Zahlen entstehen, wenn diese größer, mehr als etwa 3.4E38, werden. Dann wird ein Überlauf-Fehler ausgegeben.

Bei `I-` wird obige Korrektur nicht durchgeführt.

Vorgabewert `I-`.

5 Der eingebaute Editor

5.1 Einführung

Der mitgelieferte Editor ist ein einfacher zeilenorientierter Editor, der ein einfaches, schnelles und wirksames Editieren der Programme erlaubt.

Eine Eingabezeile kann maximal 80 Zeichen umfassen, weitere Zeichen werden ignoriert. Mit ENTER wird der editierte Text in einer kompakten Form im Quelltextspeicher abgelegt. Die Zahl der führenden Leerzeichen einer Zeile wird in einem Byte am Anfang dieser Zeile abgespeichert. Alle reservierten KC-PASCAL-Wörter werden als Token ebenfalls in einem Byte abgespeichert. Der Editor wird nach dem Laden von KC-PASCAL automatisch aufgerufen und zeigt je nach Version etwa folgende Meldung:

```
****** PASCAL V4.4X ******
  Bearb. von +++ AMXX +++
    (Version XXXXXX)
```

gefolgt vom Zeichen '+' (Editor-Prompt) an.

Dabei gibt X, bzw. XXXXXX die jeweilige Versionsbezeichnung je nach Gerätetyp und Erstellungsjahr an.

Im folgenden ist die Eingabe einer Kommandozeile des Formates:

```
C N1, N2, S1, S2
```

gefolgt von 'ENTER' möglich, wobei

C	das auszuführende Kommando
N1 und N2	Zahlen im Bereich 1 bis 32767 (Ausnahme 'Z')
S1 und S2	STRING OF CHAR mit einer maximalen Länge von 20

darstellen.

Das Komma wird benutzt, um die einzelnen Argumente zu trennen (kann durch das S-Kommando verändert werden). Außer innerhalb der Strings werden Leerzeichen ignoriert. Keines der Argumente ist obligatorisch, aber einige Kommandos (z.B. das 'D'elete-Kommando) werden nicht ausgeführt, ohne dass N1 und N2 spezifiziert werden. Wird ein Argument nicht eingegeben, nimmt der Editor das zuletzt an dieser Stelle eingegebene. Die Werte von N1 und N2 werden anfangs auf 10 gesetzt, die Strings sind leer. Wird eine unzulässige Zeile, wie F-1,100,HELLO eingeben, wird die Zeile ignoriert und 'Pardon?' ausgegeben. Diese Fehlermeldung wird auch ausgegeben, wenn die Länge von S2 20 übersteigt. Wenn S1 länger als 20 ist, werden die überzähligen Zeichen ignoriert. Editier-Kommandos können als Groß- oder Kleinbuchstaben eingegeben werden.

5.2 Editierkommandos

5.2.1 Text einfügen

Text kann in das Textfile eingefügt werden, indem man eine Zeilennummer, gefolgt von mindestens einem Leerzeichen und dem gewünschten Text eingibt, oder indem man das I-Kommando benutzt. (Wenn eine Zeilennummer gefolgt von 'ENTER', d.h. ohne nachfolgenden Text eingegeben wird, wird sie aus dem Textfile gelöscht, falls sie existiert).

Beim Eingeben von Text können die eingangs beschriebenen Steuertasten verwendet werden. Der Text wird immer erst in einen Eingabepuffer eingegeben. Ist dieser voll, wird das Eingeben weiterer Zeichen verhindert.

Kommando `I n,m`

I ruft den automatischen Einfüge-Modus auf. Die Zeilennummern werden, beginnend mit n und mit einem Inkrement von m, automatisch erzeugt und angezeigt. Der gewünschte Text kann unter Benutzung der Steuerkommandos hinter der angezeigten Zeilennummer eingegeben und mit ENTER abgeschlossen werden. Durch BREAK kann man den Einfüge-Modus wieder verlassen. Wenn eine eingegebene Zeile die gleiche Zeilennummer wie eine bereits existierende besitzt, wird die alte (nach ENTER) gelöscht und durch die neu eingegebene ersetzt.
Wenn die automatische Inkrementierung eine Zeilennummer größer als 32767 erzeugen würde, wird der Einfüge-Modus automatisch verlassen. Wenn beim Eingeben von Text das Ende einer Bildschirmzeile erreicht wird, ohne dass der Textpuffer voll ist (80 Zeichen), wird der Bildschirm gerollt und auf der folgenden Zeile kann weiterer Text eingegeben werden. Der Text wird automatisch eingerückt, so dass die Zeilennummern vom Text abgehoben sind.

5.2.2 Text anzeigen

Der Text kann mittels des L-Kommandos durchgesehen werden. Die Anzahl der auf einmal gelisteten Zeilen kann durch das K-Kommando festgelegt werden.

Kommando `L n,m` (List)

Listet den Text von Zeilennummer n bis m. Der Vorgabewert für n ist immer 1 und der für m immer 32767, d.h. er wird nicht von vorher eingegebenen Argumenten übernommen. Um den gesamten Textspeicher zu listen, kann also einfach 'L' benutzt werden. Die Zeilen werden auf dem Bildschirm mit einem linken Rand ausgegeben, so dass die Zeilennummer deutlich erkennbar ist. Wenn die durch das K-Kommando bestimmte Zeilenzahl ausgegeben und m noch nicht erreicht ist, wird das Listen unterbrochen. BREAK bewirkt die Rückkehr zur Editor-Hauptschleife, jede andere Taste setzt das Listen fort.

Kommando `K n`

Legt die Zahl der Zeilen fest, bis das Listen anhält. Der Wert (`n MOD 256`) wird berechnet und gespeichert.

Vorgabewert ist n=15.

5.2.3 Text editieren

Verschiedene Kommandos ermöglichen es, einen erzeugten Text zu verändern, zu löschen, zu verschieben und neu zu nummerieren.

Kommando D n,m (Delete)

Alle Zeilen von n bis m werden aus dem Textfile gelöscht. Wenn m < n ist oder weniger als 2 Argumente eingegeben wurden, wird nichts unternommen, der Text also vor versehentlich falschen Eingaben geschützt. Eine einzelne Zeile kann durch m = n gelöscht werden. Das kann aber auch durch einfaches Eingeben der Zeilennummer gefolgt von ENTER erreicht werden.

Kommando M n,m (Move)

Die Textzeile n wird auf die Textzeile m kopiert und löscht evtl. dort stehenden Text. Die alte Zeile n wird nicht gelöscht! Somit kann eine Textzeile an eine andere Position des Textfiles gesetzt werden. Falls eine Zeilennummer n nicht existiert, geschieht nichts.

Kommando N n,m (Renumber)

Durch N wird dass Textfile neu durchnummeriert. Die neue erste Zeilennummer ist n, der Abstand m. Sowohl m als auch n müssen angegeben werden. Wenn die Neunummerierung eine Zeilennummer größer als 32767 erzeugen müsste, wird die alte Nummerierung beibehalten.

Kommando F n,m,f,s (Find)

Der Text im Zeilenbereich n < x < m wird nach dem String f durchsucht. Wird f gefunden, so wird die entsprechende Zeile angezeigt und der Edit-Modus (s. unten) aufgerufen. Weitere Kommandos im Edit-Modus können verwendet werden, um nach einem weiteren Vorkommen von f im definierten Zeilenbereich zu suchen oder um den gerade gefundenen String f durch den String s zu ersetzen und dann nach einem weiteren Auftreten von f zu suchen (s. auch unten). Beachten Sie, dass der Zeilenbereich und die Strings durch vorhergehende andere Kommandos gesetzt sein können, so dass die Suche u. U. nur durch 'F' initiiert werden kann.

Kommando E n (Edit)

Die Zeile mit der Nummer n wird editiert. Falls n nicht existiert, geschieht nichts. Anderenfalls wird die Zeile in einen Puffer kopiert und auf dem Bildschirm angezeigt. Die Zeilennummer wird darunter nochmals angezeigt und der Edit-Modus aufgerufen. Alle nachfolgenden Handlungen finden im Puffer, nicht im eigentlichen Text statt. Somit kann der ursprüngliche Zustand jederzeit wieder hergestellt werden. In diesem Modus stellt man sich einen Zeiger vor, der sich, beginnend vom ersten Zeichen, durch die Zeile bewegt.

Folgende Sub-Kommandos werden zum Editieren der Zeile benutzt:

→	bewegt den Text-Zeiger ein Zeichen weiter
←	bewegt den Text-Zeiger ein Zeichen zurück (höchstens bis zum ersten Zeichen der Zeile)
^	bewegt den Text-Zeiger zur nächsten TAB-Position, (aber höchstens bis zum Zeilenende)
ENTER	die editierte Zeile wird so übernommen
Q	(QUIT) bricht das Editieren ab, d.h. belässt die Zeile in ihrem alten Zustand (vor dem Aufruf des Edit-Modus)
R	("RELOAD") lädt den Textpuffer nochmals mit der alten Version der Zeile, d.h. macht alle Änderungen rückgängig
L	("LIST") listet den Rest der editierten Zeile, d.h. die Zeile rechts von der gegenwärtigen Zeigerposition. Der Edit-Modus bleibt bestehen und der Zeiger wird wieder an den Beginn der Zeile gesetzt.
K	("KILL") löscht das Zeichen an der aktuellen Zeigerposition
Z	löscht alle Zeichen ab (und einschließlich) der aktuellen Zeigerposition bis zum Ende der Zeile
F	("FIND") sucht das nächste Auftreten des vorher mit dem Kommando 'F' definierten Suchstring f (s. oben). Dieses Subkommando schließt gleichzeitig das Editieren der gerade aufgerufenen Zeile ab (unter Beibehaltung aller Änderungen), falls der String f in dieser Zeile nicht noch einmal auftritt. Wenn (im definierten Zeilenbereich), wird diese Zeile zum Editieren in den Puffer geladen und der Text-Zeiger auf den Beginn der Zeile gesetzt.
S	("SUBSTITUTE") ersetzt den eben gefundenen String f durch den vorher definierten Austauschstring s und führt dann automatisch das Sub-Kommando 'F' aus, d.h. sucht nach dem nächsten Auftreten von f. 'S' wird also zusammen mit 'F' benutzt, um sich durch Textfile hindurchzuarbeiten und jeden String f nach Belieben gegen String s auszutauschen (weiter mit 'S') oder auch nicht (weiter mit 'F').

Achtung

Das Sub-Kommando 'S' darf nur unmittelbar nach einem 'F'-Sub-Kommando oder 'S'-Sub-Kommando benutzt werden.

I	("INSERT") fügt Zeichen an der gegenwärtigen Zeigerposition ein. Dieser Sub-Modus wird beibehalten, bis man mit ENTER in den normalen Edit-Modus zurückgekehrt. Der Zeiger weist dann auf die Position hinter dem letzten eingefügten Zeichen. In diesem Sub-Modus löscht DELETE das Zeichen vor dem Zeiger aus dem Textpuffer, während '^' den Zeiger bis zur nächsten TAB-Position vorrücken lässt, wobei Leerzeichen eingefügt werden.
X	setzt den Zeiger auf das Ende der Zeile und ruft automatisch den I-Sub-Modus auf (s. oben)
C	("CHANGE") Das Zeichen an der aktuellen Zeigerposition wird durch das eingegebene überschrieben und der Zeiger eine Stelle weiter gerückt. Auch dieser Sub-Modus wird beibehalten, bis man durch Drücken von ENTER in den normalen EDIT-Modus zurückkehrt. Der Zeiger weist dann auf die Position hinter dem letzten geänderten Zeichen. DELETE bewegt den Zeiger in diesem Sub-Modus einfach ein Zeichen nach links. '→' hat keine Wirkung.

5.2.4 Tonband-Kommandos

Der Text kann durch die Kommandos 'P' und 'G' auf Band gesichert bzw. vom Band geladen werden.

Kommando P n,m,s (Put)

Der Zeilenbereich n < x < m wird im KC-PASCAL-Format unter dem Dateinamen s auf Band gesichert. Beachten Sie, dass die Argumente durch vorhergehende Kommandos gesetzt sein können (s. "V"). Vor Abschluss dieses Kommandos schalten Sie bitte den Kassettenrecorder auf Aufnahme.

Kommando G ,,s (Get)

Das Band wird nach einem File in KC-PASCAL-Format mit dem Namen s abgesucht. Ist der richtige Filename gefunden, wird der Text in den Speicher geladen. Gefundene Files werden angezeigt. Fragezeichen im gesuchten Filenamen führen zum Überlesen des jeweiligen Zeichens im gesuchten File. Wird '????????' eingegeben, wird das nächste File eingelesen, gleich welchen Namen es besitzt (Typ 'PAS' ist erforderlich). Das Einlesen kann durch BREAK abgebrochen werden.

Wenn bereits ein Textfile im Speicher vorhanden ist, wird das neu geladene File an das existierende Textfile angehängt und danach der gesamte Text, beginnend mit Zeilennummer 1 in 1er Schritten, neu durchnummeriert.

5.2.5 Compilieren und Probelauf

Kommando C n (Compilieren)

Der Text wird, beginnend mit Zeilennummer n, compiliert. Wird keine Zeilennummer angegeben, beginnt die Compilierung mit der ersten existierenden Zeile. Bei fehlerfreier Übersetzung erfolgt die Abfrage 'Lauf?'. Bei Beantwortung mit 'J' wird das Programm ausgeführt.

Kommando R (Run)

Der eben erzeugte Objekt-Code wird gestartet, allerdings nur, wenn der Quelltext mittlerweile nicht erweitert wurde.

Kommando T n,m,s,a (Translate)

Der Quelltext wird wiederum, beginnend mit Zeile n (bzw. ab der ersten existierenden Zeile), bis Zeile m (wenn angegeben) compiliert. War die Übersetzung erfolgreich, wird die Frage 'Ok?' ausgegeben. Antwortet man mit 'J', wird der erzeugten Objekt-Code an das Ende der Runtimes verschoben und zusammen mit diesen auf Band gespeichert (Dabei wird der Compiler zerstört!). Schalten Sie vor Beantworten der Frage mit 'J' ihr Magnetbandgerät auf Aufnahme. Als Filename wird 's' verwendet, oder wenn nicht angegeben der letzte definierte Suchstring. Wird dieses File zu

einem späteren Zeitpunkt in den Rechner geladen, kann es als wie üblich unter dem Namen des zuletzt verwendeten String S1 wieder gestartet werden. Beachten Sie, dass sich der Objekt-Code nach der Antwort 'J' am Ende des Runtimes befindet, d.h. dass der Compiler nach Ausführung von 'T' neu geladen werden muss. Das bringt jedoch keine Probleme, da Sie ja wahrscheinlich nur ein vollständig ausgetestetes, fehlerfreies Programm mit 'T' übersetzen. Antwortet man auf die Frage 'Ok?' nicht mit 'J', sondern mit einer anderen Taste, wird der Objekt-Code nicht verschoben und die Steuerung an den Editor zurückgegeben.

Ab KC-PASCAL 5.2 kann mit dem optionalen vierten Parameter a der Autostart der generierten KCC-Datei aktiviert werden. Dazu muss an dieser Stelle eine 1 angegeben werden.

5.2.6 Systemzellen

Kommando V (Editorvariablen)

Mit diesem Kommando werden die Editorvariablen in der Reihenfolge N1, N2, S1, S2 angezeigt. Als Trennzeichen wird das aktuell vereinbarte verwendet. Damit kommen die aktuellen Zeilennummern und Suchstrings (auch für Tape-E/A) zur Anzeige.

Kommando X (Systemzellen)

Es werden die Systemzellen in folgender Reihenfolge angezeigt:

```
wwww xxxx yyyy zzzz
```

Dabei ist wwww die Anfangsadresse des PASCAL-Quelltextes, xxxx die Endadresse des Quelltextes, yyyy die Endadresse des Arbeitsspeichers und zzzz die Endadresse des Arbeitsspeichers für übersetzte Programme nach dem Kommando 'T'.

Kommando Z n,m (Systemzellen stellen)

Mit diesem Kommando können Sie die Endadresse des Arbeitsspeichers n (s. xxxx oben) und die Endadresse des Arbeitsspeichers m für übersetzte Programm neu wählen. Damit ist es möglich, Speicherbereich für andere Aufgaben zu reservieren, aber auch die übersetzten Programme in Rechnersystemen mit weniger RAM laufen zu lassen. Die Adresseingabe erfolgt dezimal. Nach dem Kommando erfolgt ein Neustart des KC-PASCAL.

ACHTUNG!

Liegen die eingegebenen Werte nicht im RAM, im Compiler usw., hat das einen unweigerlichen Systemabsturz zur Folge. Nach Ausführen des Kommando 'Z' wird ein Kaltstart ausgeführt und vorhandener Quelltext gelöscht. Benutzen Sie dieses Kommando nur nach dem Kaltstart oder nachdem Sie Ihr Programm abgespeichert haben.

5.2.7 Weitere Kommandos

Kommando B (Betriebssystem)

Bewirkt eine Rückkehr zum Betriebssystem.

Kommando O n,m,s (Tokenisieren)

KC-PASCAL speichert alle reservierten Worte in tokenisierter Form (1 Byte) und führende Leerzeichen einer Zeile in einem weiteren Byte. Wenn Sie jedoch Text in vollständiger, ausgeschriebener Form vorliegen haben (vielleicht von einem anderen Editor oder z.B. TURBO-PASCAL-Quellprogramme), kann dieser mit dem Kommando O tokenisiert werden. Der Text wird in der Quellform in einen Puffer gelesen und danach in tokenisierten Form zurück geschrieben. Das dauert u.U. eine gewisse Zeit. Dabei ist n die erste neue Zeilennummer, die aus dem File mit Namen s gebildet wird. Ist m = 1, werden alle Buchstaben des File s in Großbuchstaben gewandelt. Ist m = 2, bleibt die ursprüngliche Schreibweise erhalten.

Beachten Sie bitte, dass die Programmzeilen durch CR/LF (0dh, 0ah) getrennt sein müssen und die Datei mit EOF (01ah) enden muss (n-Mode bei WordStar bzw. TP). Weiterhin muss die Datei einen Vorblock (080h Byte) analog dem KC-Kassettenverfahren besitzen. Zur Tokenisierung von TURBO-PASCAL-Quellen sollten Sie die CP/M-Version von KC-PASCAL benutzen, da dabei beim Einlesen von Diskette automatisch eine Unterscheidung zwischen Quelltext und tokenisierter Form erfolgt und der Vorblock nicht benötigt wird. Die Länge des Quelltextes darf 16 KByte nicht übersteigen.

Kommando S ,,d (Separator-Delemiter)

Dieses Kommando ermöglicht eine Abänderung des Trennzeichens, welches zur Abgrenzung der Argumente in der Kommandozeile benutzt wird. Als neues Trennzeichen wird das erste Zeichen des Strings d benutzt. Nach dem Kaltstart wird das Komma benutzt. Beachten Sie, dass ein definiertes Trennzeichen solange benutzt werden muss (auch im 'S'-Kommando), bis ein neues definiert wurde.

Ein Leerzeichen darf nicht als Trennzeichen definiert werden.

6 Fehlermeldungen

6.1 Compiler-Fehlermeldungen

1. Zahl zu groß
2. Semikolon wird erwartet
3. Nichtdeklarierter Name
4. Name wird erwartet
5. In der Deklaration einer Konstanten muss '=' und nicht ':=' verwendet werden.
6. '=' wird erwartet
7. Eine Anweisung darf nicht mit diesem Namen beginnen.
8. ':=' wird erwartet
9. ')' wird erwartet
10. Falscher Typ
11. '.' wird erwartet
12. Ein Faktor wird erwartet.
13. Eine Konstante wird erwartet.
14. Dieser Name ist keine Konstante.
15. 'THEN' wird erwartet
16. 'DO' wird erwartet
17. 'TO' oder 'DOWNTO' wird erwartet
18. '(' wird erwartet
19. Dieser Ausdrucktyp kann nicht geschrieben werden.
20. 'OF' wird erwartet
21. ',' wird erwartet
22. ':' wird erwartet
23. 'PROGRAM' wird erwartet
24. Eine Variable wird erwartet, da der Parameter ein variabler Parameter ist.
25. 'BEGIN' wird erwartet
26. Bei diesem READ-Aufruf wird eine Variable erwartet.
27. Ausdrücke dieses Typs können nicht verglichen werden.
28. Eine INTEGER- oder REAL-Zahl wird erwartet.
29. Dieser Variablentyp kann nicht gelesen werden.
30. Dieser Name ist kein Typ.
31. Bei reellen Zahlen wird ein Exponent verlangt.
32. Ein skalarer Ausdruck (kein numerischer) wird erwartet.
33. Leerstrings sind nicht erlaubt, (CHR (0)) benutzen.
34. '[' wird erwartet
35. ']' wird erwartet
36. Ein Arrayindex muss vom Typ SCALAR sein
37. '..' wird erwartet
38. In der Deklaration eines ARRAYs wird ']' oder ',' verlangt
39. Obergrenze ist größer als die untere.
40. Das Set ist zu groß (größer als die 256 möglichen Elemente).
41. Das Ergebnis der Funktion muss vom Typ Name sein.

42. Im Set wird ',' oder ' ' erwartet.
43. Im Set wird '..' oder ']' erwartet.
44. Der Parameter muss vom Typ Name sein.
45. Ein leeres Set kann nicht erster Faktor in einer Nicht-Zuweisungs-Anweisung sein.
46. Ein Skalartyp (einschließlich realen Zahlen) wird erwartet.
47. Ein Skalartyp (außer realen Zahlen) wird erwartet.
48. Die Sets sind nicht verträglich.
49. Sets können nicht mit '<' und '>' verglichen werden
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' oder 'BEGIN' wird erwartet
51. Hexadezimalziffer wird erwartet
52. Sets können nicht gePOKEd werden
53. Das Array ist zu groß.
54. Bei der Definition von RECORDs wird 'END' oder '+' erwartet.
55. Ein Feldname wird erwartet.
56. Nach 'WITH' wird eine Variable erwartet.
57. Die Variable bei WITH muss vom RECORD-Typ sein.
58. Der Feldname wurde nicht mit der WITH-Anweisung in Verbindung gebracht.
59. Nach 'LABEL' wird eine vorzeichenlose Integerzahl erwartet.
60. Nach 'GOTO' wird eine vorzeichenlose Integerzahl erwartet.
61. Diese Marke ist in einer falschen Programmebene.
62. Marke ist nicht vereinbart.
63. Der Parameter von SIZE muss eine Variable sein.
64. Auf Zeiger kann nur der Gleichheitstest angewandt werden.
67. Der einzig zulässige WRITE-Parameter für Integerzahlen mit zwei Doppelpunkten ist e:m:H.
68. Strings dürfen keine Zeilenende-Zeichen enthalten.
69. Der Parameter von NEW, MARK oder RELEASE muss eine Zeigervariable sein.
70. Der Parameter von ADDR muss eine Variable sein.

6.2 Runtime-Fehlermeldungen

Tritt während des Programmlaufs ein Fehler auf, so wird eine der folgenden Meldungen gefolgt von 'bei PC=XXXX' ausgegeben, wobei XXXX die Speicheradresse darstellt, an der der Fehler auftrat. Meist wird die Fehlerquelle offensichtlich sein, falls jedoch nicht, kann das Compiler-Listing zu Rate gezogen werden um festzustellen, an welcher Stelle des Programms der Fehler auftrat. XXXX dient dann als Crossreferenz. Gelegentlich liefert das jedoch NICHT das richtige Resultat (Folgefehler).

1. Halt
2. Ueberlauf
3. >> RAM (außerhalb des RAM)
4. / durch Null (Division durch Null; kann auch bei DIV auftreten)
5. Index zu niedrig
6. Index zu hoch
7. mathematischer Fehler (Fehler beim Aufruf einer mathematischen Routine)

8. Zahl zu groß
9. Zahl erwartet
10. Exponent erwartet

Laufzeit-Fehler stoppen die Programmausführung.

7 Reservierte Worte und vordefinierte Namen

7.1 Reservierte Worte

```
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
```

7.2 Spezialsymbole

Die folgenden Symbole werden in KC-PASCAL benutzt und haben eine bestimmte Bedeutung:

```
+         -         *         /
=         <>        <         <=        >=        >
(         )         [         ]
{         }         (*        *)
^         :=        .         ,         ;         :
```

7.3 Vordefinierte Namen

Die folgenden Namen kann man sich als in einem das gesamte Programm umschließenden Block definiert denken. Demzufolge sind sie im gesamten Programm verfügbar, es sei denn, sie wurden durch den Programmierer innerhalb eines Block umdefiniert.

CONST	MAXINT=32767; (INTEGER) PI=3.14159E+00; (REAL) TRUE; (BOOLEAN) FALSE; (BOOLEAN)
TYPE	BOOLEAN=(FALSE,TRUE); CHAR (Erweiterter ASCII-Zeichensatz) INTEGER=-MAXINT..MAXINT; REAL (Eine Untermenge der Realen Zahlen)
PROCEDURE	WRITE; WRITELN; READ; READLN; PAGE; HALT; USER; POKE; INLINE; OUT; NEW; MARK; RELEASE; TIN; TOUT; GOTOXY;
nur KC85/2/3	SETC; GETC; PLOT; CLRPLOT
nur KC85/3	LINEPLOT, CIRCLE
FUNCTION	ABS; SQR; ODD; RANDOM; ORD; SUCC; PRED; EOLN; PEEK; CHR; SQRT; ENTIER; ROUND; TRUNC; FRAC; SIN; COS; TAN; ARCTAN; EXP; LN; ADDR; SIZE; INP; KEYPRESSED; READKBD; SWAP; HI; LO; SHL; SHR; BOR; BAND; BXOR

8 Beispiele

8.1 Beispiel einer „DO – WHILE – Konstruktion“

```
10 PROGRAM DEMO;
11 VAR
12  I:INTEGER;
13 BEGIN
14  I:=1;
15  WHILE I < 10 DO
16  BEGIN
17   WRITE('DER WERT IST: '; I);
18   WRITELN(' UND I*I IST: , I*I);
19   I:=I+1;
20  END;
21  WRITELN('PROGRAMMENDE');
22 END.
```

8.2 Beispiel "Demonstration von Funktionen und Prozeduren"

```
 10 PROGRAM FUT;
 20 VAR
 30  I,J:INTEGER;
 40  R:REAL;
 50 BEGIN
 60  WHILE I<>0 DO
 70  BEGIN
 80   WRITE('Werteingabe: ');
 90   READ(I);
100   WRITELN('WERT =',I);
110   WRITELN('SWAP =',SWAP(I));
120   WRITELN('HI =',HI(I));
130   WRITELN('LO =',L0(I));
140   WRITELN('SHL =',SHL(I));
150   WRITELN('SHR =',SHR(l));
160  END;
170  PAGE;
180  R:=0;
190  I:=1;
200  WHILE I<64 DO
210  BEGIN
220   GOTOXY(I, 8+ROUND(5*SIN(I/5)));
230   WRITE('o');
240   I=I+1;
250  END;
260 END.
```

8.3 Beispiel "Erzeugen von Zufallszahlen"

```
 10 PROGRAM RANDOM;
 20 VAR
 30  I, R: INTEGER;
 40 BEGIN
 50  I:=1;
 60  WHILE I < 11 DO
```

```
 70  BEGIN
 80   R:=RANDOM;
 90   WRITELN('Wert ist: ' ,R);
100   I:=I+1;
110  END;
120 END.
```

8.4 Beispiel TOUT und TIN

Das Programm eignet sich zum Test von PasEx2 unter CAOS Version 4.6 und älter. Dazu wird aus KC-PASCAL heraus mit T ein ausführbares Programm GRUSS.KCC generiert. Es ist nicht wichtig, ob danach erst GRUSS.KCC oder PASEX2.KCC geladen wird, nur müssen beide im Speicher sein. Mit PASDEV stellt man den gewünschten Massenspeicher ein und startet dann GRUSS.

Um auf dem KC 85/4 im Hauptspeicher die geschriebenen und zurück gelesenen Daten ohne großen Aufwand mit dem Betriebssystemkommando DISPLAY zu kontrollieren, kann man nach dem Start von Pascal den verwendeten Speicher mit Z 32766,32766 begrenzen.

Nebenbei zeigt das Programm, wie mit READLN und READ die erste Zeichenkette gelesen wird.

```
 10 PROGRAM GRUSS;
 20 (* vor dem Laden mit G: Z 32766,32766 *)
 30 CONST
 40  datnam='GRUSSDAT';
 50 TYPE
 60  tMit=RECORD
 70   text:ARRAY[1..200] OF CHAR;
 80   jahr,mon,tag:INTEGER
 90  END;
100 VAR
110  ta:CHAR;
120  gruss1,gruss2:tMit;
130
140 BEGIN
150
160  WITH gruss1 DO BEGIN
170   WRITELN('Welches Datum haben wir heute?');
180   WRITE('Tag  : ');READ(tag);
190   WRITE('Monat: ');READ(mon);
200   WRITE('Jahr : ');READ(jahr);
210   WRITELN('Schreiben Sie einen kurzen Gruss an');
220   WRITELN('jemand wichtigen (max. 200 Zeichen):');
230   IF EOLN THEN BEGIN
240    READLN; READ(text); END
250   ELSE
260    READLN(text);
270  END;
280
290  WRITELN;
300  WRITELN('Schreibe Datei ',datnam);
310  WRITELN('ab Adr. ',ADDR(gruss1):4:H,' Laenge ',SIZE(gruss1):4:H);
320  WRITELN('Wenn bereit, dann ENTER');
330  ta:=READKBD;
340  TOUT(datnam,ADDR(gruss1),SIZE(gruss1));
```

```
350
360  WRITELN;
370  WRITELN('Lese Datei ',datnam);
380  WRITELN('ab Adr. ',ADDR(gruss2):4:H);
390  WRITELN('Wenn bereit, dann ENTER');
400  ta:=READKBD;
410  TIN(datnam,ADDR(gruss2));
420  WRITELN;
430  WITH gruss2 DO BEGIN
440   WRITELN('Gruss vom ',tag:2,'.',mon:2,'.',jahr);
450   WRITELN(text);
460  END;
470
480  WRITELN;
490  WRITELN('Bis bald!');
500 END.
```

Literaturverzeichnis

KCPascal: Mugler, Albrecht, KC-Pascal (Version 4.4), 1987, http://kc85.info/index.php/download-topmenu/viewdownload/5-programmiersprachen/43-kc-pascal.html
Hisoft: , Memotech Hisoft Pascal User Guide, 1984, https://www.cpcwiki.eu/imgs/6/64/Hisoft_Pascal_4T_Manual_%28English%29.pdf