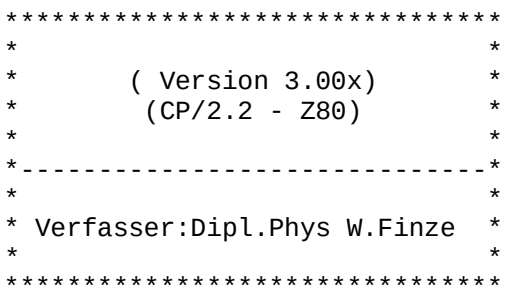
TURBO-Pascal

Nutzerhandbuch



Inhalt

Teil 1 : Bedienungsanleitung

Teil 2 : Abweichungen von TURBO gegenueber Standard-PASCAL

Teil 3 : Zeiger und dynamische Speicherverwaltung

Anlage A : Zusammenfassung

Anlage B : PASCAL-Syntaxnotation

Anlage C : Dateiarbeit

Anlage D : Overlay-Techniken

Anlage E : INLINE-Assembler

Anlage F : Fehlermeldungen

Anlage G : Beispiele

Bedienungsanleitung

1. TURBO - Hauptmenue (Textanfang verstuemmelt)

nachfolgend beschrieben.

D)irectory

Bei diesem Kommando wird nach dem Laufwerk gefragt, dessen Inhalt dargestellt werden soll. Es wird nur ein Buchstabe (A, B,..,D oder @) mit nachfolgendem Return eingegeben. Der Buchstabe weist auf das Laufwerk, @ gilt fuer das aktuelle Laufwerk. Die Inhaltsangabe der Diskette wird 4-spaltig angezeigt bis der Bildschirm voll ist. Mit Return kann die Anzeige fortgesetzt werden. Nach der Anzeige aller Dateien wird mit Return der Supervisor wieder aktiv.

E)ditor

Der Editor ist ein bildschirmorientierter schneller Texteditor. Er ist speziell fuer die Herstellung und Veraenderungen von Texten geeignet. Er ist in Pascal geschrieben und zeigt somit die Leistungsfaehigkeit von Pascal. Er hat sehr viel Aehnlichkeit mit WordStar, wobei Textveraenderungen jedoch erst nach Bestaetigung durch den Programmierer wirksam werden. Damit sind Fehlbedienungen weitgehend ausgeschlossen.
Die erste Zeilmandos/Befehle verwendet werden
(z.B.BDOS- oder BIOS-Aufrufe bzw MSDOS-Aufrufe).
Ueber Zusatzfunktionen gegenueber Standard-PASCAL ist auch in TURBO ein maschinennahes Programmieren moeglich. Z.B. sind im Compiler PORT-Arrays vordefiniert, mit denen sich die I/O-Ports des Prozessors ansprechen lassen. Direkte Speicheroperationen werden ebenfalls unterstuetzt.
Der Arbeitszyklus fuer die Erstellung von lauffaehigen Prgrammen ist gegenueber anderen PASCAL-Implementierungen erheblich verkuerzt und vereinfacht. So lange der Compiler noch Fehler im Quellprogramm erkennt, wird automatisch in den Editor-Mode und an die Fehlerstelle zurueckgesprungen. Ein Programmverbinder ist bei TURBO nicht noetig, da direkt vom Compiler ein abarbeitbarer Maschinencode erzeugt wird, wenn das Quellprogramm syntaktisch fehlerfrei ist.
Je nach gesetzter Compiler-Option kann das so erzeugte Programm direkt ausgefuert werden (ohne Abspeicherung des .COM-Files -> Directmode) oder es wird ein .COM-file auf Floppy erzeugt.
Die Uebersetzungsgeschwindigkeit des Compilers betraegt ca.1000 Zeilen pro Minute (!), die Genauigkeit bei Gleitkommaoperationen betraegt 11 Stellen.
Die Verarbeitungsgeschwindigkeit ist bei TURBO etwas geringer als bei PASCAL-MT+, allerdings ist bei TURBO die Genauigkeit der Gleitkommaoperationen gegenueber PASCAL-MT+ erheblich verbessert und die Grosse der erzeugten COM-files erheblich geringer.

2. Bedienung des Compilers

Der Compiler wird mit dem Namen TURBO aufgerufen und meldet sich mit seinem Namen und den Lizenzvermerken des Herstellers.
Die erste zu beantwortende Frage ist

```
INCLUDE ERROR MESSAGES (Y/N)   (Fehlermeldungen gewuenscht ?)
```

Falls die vollstaendigen Fehlermeldungen des Compilers gewuenscht werden, ist mit Y zu antworten, falls nicht, mit N. Der Compiler laedt, falls gewuenscht, die Fehlermeldungen. Je nach Generierung ist auch ein automatisches Nachladen der Fehlermeldungen moeglich.
Nach Laden der Fehlermeldungen meldet sich der Compiler mit dem Menue und seinem Bereitschaftszeichen.

```
     >
```

Vom Grundmenue aus sind folgende Funktionen erreichbar:

Logged drive, **W**ork file, **M**ain file, **E**dit, **C**ompile,
Run, **S**ave, e**X**ecute, **D**ir, **Q**uit, compiler **O**ptions.

2.1. Menuefunktionen

Die Auswahl der Menuefunktionen erfolgt durch Eingabe des in dieser Beschreibung schraeg gedruckten Buchstabens fuer die jeweilige Funktion. Die Eingabe kann in Gross- oder Kleinbuchstaben erfolgen.

2.1.1. **L**ogged drive

Logged drive ist das aktuelle Laufwerk fuer den Compiler, auf das er immer dann zugreift, wenn kein Laufwerk explizit gefordert ist. Mit Eingabe von **L** laesst sich von der Kommando-/Menueebene das aktuelle Laufwerk aendern.

2.1.2. **W**ork file

Work file ist das gerade in Arbeit befindliche Programm. Falls bei Aufruf des Editors noch kein **W**ork file angegeben ist, fragt der Editor nach dem Namen des Files und versucht es von Diskette zu laden. Ist das File noch nicht auf Diskette, erscheint die Ausschrift "new file" (neue Datei). Ein Wechsel des mit dem Editor zu bearbeitenden Programms ist von der Menueebene aus durch die Eingabe von **W** und der anschliessenden des neuen Namens fuer **W**ork file moeglich.

2.1.3. **M**ain file

Main file ist der Name des aus **W**ork file erzeugten abarbeitbaren Programmes (COM-files). Sollte main-file nicht mit work-file uebereinstimmen, wird beim Aufruf des Compilers das work-file gesichert, das main-file nachgeladen und uebersetzt.

2.1.4. **E**dit

Der Editor ist bewusst an WordStar angelehnt, um ein Umlernen zu vermeiden. Fuer den Zweck 'Programmerstellung' wird die Leistungsfaehigkeit von WordStar erreicht.

Die Zuordnung der Editorfunktionen zu den Control-Funktionen ist installationsabhaengig, die hier beschriebenen Kommandos gelten fuer DAC.

Folgende Befehle sind moeglich:

2.1.4.1. Cursorsteuerung

Funktion	Kommando
Cursor links	^H oder ^S
Cursor rechts	^D
Cursor hoch	^E
Cursor nach unten	^X
Wort links	^A
Wort rechts	^F
Schirm nach unten rollen(eine Zeile)	^W
Schirm nach oben rollen(eine Zeile)	^Z
Schirm nach unten rollen(eine Seite)	^R
Schirm nach oben rollen(eine Seite)	^C
Cursor an linken Zeilenrand	^QS
Cursor an rechten Zeilenrand	^QD
Cursor an Seitenanfang	^QE
Cursor an Seitenende	^QX
Cursor an Dateianfang	^QR
Cursor an Dateiende	^QC
Cursor an Blockanfang	^QB
Cursor an Blockende	^QK
zurueck an letzte Cursorposition	^QP

2.1.4.2. Einfuegen/Loeschen

Funktion	Kommando
Inset-Mode an/aus	^V
Zeile einfuegen	^N

Zeile loeschen	^Y
loeschen bis Zeilenende	^QY
Wort rechts vom Cursor loeschen	^T
Zeichen auf Cursorposition loeschen	^G
Zeichen links vom Cursor loeschen	DEL,DELCH (Tast.abh.)

2.1.4.3. Blockkommandos

Achtung: Bei den Blockkommandos wird Blockanfang/-ende nicht wie bei WordStar mit <B>/<K> markiert. Die Art der Markierung ist installationsabhaengig (meist wird ueber die Bildhelligkeit markiert, bei der vorliegenden Version erfolgt keine sichtbaere Markierung!).

Funktion	Komando
Blockbeginn markieren	^KB
Blockende "	^KK
Wort markieren	^KT
Block sichtbar/unsichtbar	^KH
Block kopieren	^KC
Block verschieben	^KV
Block loeschen	^KY
Datei/Block von Disk lesen	^KR
Datei/Block auf Disk schreiben	^KW

2.1.4.4. zusaetzliche Editorfunktionen

Funktion	Kommando
Ende des Editierens	^KD
TAB	^I
automatischer Tabulator an/aus	^QI
Zeile zurueckspeichern	^QL
Finden	^QF
Finden und ersetzen	^QA
Steuerzeichenvorsatz	^P

2.1.4.5. INCLUDE-Funktion des Editors

Die INCLUDE Funktion des Editors erlaubt das Zusammenfuegen von Programmen aus vorfabrizierten Textbausteinen. INCLUDE muss im Programm gefordert werden und wird wie folgt aktiviert:

{$I <filename>}

Der Compiler laedt bei der Uebersetzung des Programmes die angegebene Datei <filename> und fuegt sie in das Quellprogramm ein. INCLUDE stellt im Directmode die einzige Moeglichkeit dar, andere Programme zuzuladen,da im Directmode keine separat uebersetzten Programme in das zu compilierende Programm eingefuegt oder hizugefuegt werden koennen.
Sollen die eingefuegten Files mit TLIST und der .L+ Option gelistet werden, ist **unbedingt** der vollstaendige Dateiname mit Erwei-

terung anzugeben, da TLIST nicht wie der Compiler bei fehlender Erweiterung standardmaessig .PAS annimmt, sondern immer den vollstaendigen Dateinamen verlangt!

2.1.5. **C**ompile

Uebersetzung des **W**ork file ohne Ausfuerung. Falls compiler **O**ption **C**om-file gesetzt ist, wird das in **M**ain file genannte .COM-File erzeugt.
Waehrend des Compilierens steht auf dem Bildschirm der Text

"Compiling".

Bei Auftreten eines Fehlers im Quellprogramm erscheint die Meldung:

ERROR <nummer> : <fehlertext>. Press <ESC>.

Es ist mit der ESCAPE-Taste zu antworten und TURBO kehrt in den Editormode und an die Stelle, an der der Fehler bemerkt wurde (ist nicht unbedingt die Stelle, wo der Fehler verursacht wird!) zurueck.
Bei syntaktisch fehlerfreier Quelle erscheinen die Meldungen:

```
 Code :      xxxx bytes (adresse von-bis im Hauptspeicher)
 Free :      xxxx  "    (   "     "   "   "       "      )
 Data :      xxxx  "    (   "     "   "   "       "      )
```

Diese Ausschriften zeigen Groesse und Lage von Programm- und Datenbereich in Hauptspeicher an.

2.1.6. **R**un

Wie **C**ompile, allerdings mit anschliessender Ausfuehrung. Ein mit **C**ompile uebersetztes Programm wird mit **R**un direkt gestartet.

2.1.7. **S**ave

Save sichert das **W**ork file auf Diskette. Wenn im **W**ork-file keine Namenserweiterung angegeben wurde, nimmt TURBO automatisch die Erweiterung PAS an. Abgespeicert wird der Programmtext, nicht das abarbeitbare Programm.

2.1.8. e**X**ecute

Das e**X**ecute-Kommando erlaubt aus der TURBO-Menueebene heraus den Aufruf eines anderen Programmes, z.B. TLIST. Vor Ausfuehrung des e**X**ecute-Kommandos wird, falls der aktuelle Stand des work-file nicht gesichert ist, nachgefragt, ob eine Sicherung erfolgen soll (Ausschrift wie beim **Q**uit-Kommando). Nach der Sicherung er-

scheint die Ausschrift

Program:

Hier ist der Name des aufzurufenden Programms anzugeben und mit ET1/ENTER abzuschliessen. Nach Beendigung des mit e**X**ecute gerufenen Programmes laedt TURBO alle noetigen Dateien (auch das work-file) und kehrt in die Menueebene zurueck.

Achtung: Beim Nachladen groesserer Programme von TURBO aus kann es zu Speicherproblemen kommen!

2.1.9. **D**ir

Das **D**ir-Kommando erlaubt aus der Menueebene heraus die Anzeige des Directories. Nach Aufruf von **D**ir erscheint die Ausschrift

"Dir mask:".

Es sind alle aus CP/M bekannten Directory-Masken und die Angabe von Laufwerksbezeichnern zulaessig. Ohne Angabe einer Maske wird das gesamte Directory angezeigt.
Zusaetzlich wird der auf der Diskette verbleibende freie Speicherplatz mit

"Bytes remaining on LW:xxx"

angezeigt.

2.1.10. **Q**uit

Quit beendet das Programm TURBO. Sollte der aktuelle Stand des **W**ork file nicht gesichert sein, erscheint bei Eingabe von **Q** "work file <name> not saved Save (Y/N):"

(Arbeitdatei nicht gesichert, sichern (JA/NEIN):).

Bei Angabe von **Y** wird das **W**ork file gesichert und TURBO beendet, bei **N** wird TURBO ohne Sicherung des **W**ork file beendet.

2.1.11. compiler **O**ptions

Durch Eingabe von **O** laest sich die Arbeitsweise des Compilers steuern. Nach Eingabe von **O** erscheint folgendes Menue:

```
compile -> Memory
           Com-file
           cHain-file
           Find run-time error
           Quit

     >
```

Memory ist Kennzeichen des Directmode (Erzeugung und Abarbeitung

aus dem Hauptspeicher.
Bei Eingabe von **C**om-file und c**h**ain-file erscheint zusaetzlich die Ausschrift

Start address: xxxx (min xxxx)
End address: xxxx (max xxxx)

Der "compile ->" Zeiger wird auf die entsprechende Option umgestellt.
In Klammern sind die kleinstmoegliche Startadresse (TURBO-Bibliothek+1) und die groesstmoegliche Endadresse (BDOS-Anfang - 1) angegeben.

Mit **S** und **E** sind Manipulationen der Start- und Endadresse moeglich. Eine Manipulation der Startadresse kann sinnvoll sein bei Verwendung absoluter Variabler, die dann zwischen TURBO-Laufzeitsystem und dem Programmcode angelegt werden koennen. Eine Manipulation der Endadresse (standardmaessig gleich 700 bis 1000 BYTE kleiner als BDOS-Anfang - 1, ab BDOS-1 liegt der TURBO-interne Lader!) ist sinnvoll, wenn das entwickelte Programm sehr gross ist und in unterschiedlichen CP/M-Systemen mit differierender Groesse des im Speicher vorhandenen Betriebssystems problemlos laufen soll. Es ist dann immer vom groesstmoeglichen Betriebssystem auszugehen. Richtwerte koennen sein: $C100, oder falls das Programm unter MP/M laufen soll: $A100. (Zur Speicheraufteilung und zur Vorbesetzung interner Pointer s.Teil 3 des Handbuches). Eine Startadresse kleiner als der in Klammern gesetzte Wert oder eine Endadresse groesser dem eingeklammerten Wert fuehren zu undefinierten Laufzeitfehlern, da entweder Teile der TURBO-Bibliothek oder Teile des BDOS ueberschrieben werden!

Find run-time error erlaubt das Lokalisieren von Laufzeitfehlern. Die genaue Bedienung s. Anlage 6.

Com-file erzeugt entweder auf "logged drive" oder auf einem im Dateinamen angegebenen Laufwerk ein direkt abarbeitbares Programm und speichert es unter dem Namen <main-file .COM> auf Diskette ab.

c**H**n-file erzeugt entsprechend ein File mit der Erweiterung .CHN. Solche files koennen mit der CHAIN-Funktion aus laufenden Programmen heraus aufgerufen werden.
Die "compiler **O**ptions" sind nach Einstellen der gewuenschten Betriebsart mit **Q**uit zu verlassen
Fuer die Option **C**om-file ist die Angabe eines **M**ain-files vor Ausfuehrung von **C**ompile unbedingt noetig.

2.2. Compiler-Schalter

Compiler-Schalter steuern den Ablauf des Compilierens sowie die Art und Groesse des erzeugten Codes. Sie werden in folgender Form in das Quellprogramm eingefuegt:

```
{$<schalter><symbol1>,<symbol2>,....,<symbolX>}
```

Zu beachten ist, dass vor und nach dem $ kein Blank stehen darf.

Als Symbol zugelassen sind:
-die Buchstaben A B C I R V U W X gefolgt von + oder -
-O Laufwerk (fuer Overlays)

Die Schalter haben im einzelnen folgende Bedeutung:

A Rekursion Default: **A+**
{$A+} --> es wird absoluter Code erzeugt der keine Rekursion zulaesst
{$A-} --> Rekursion ist zugelassen
benoetigt mehr Speicherplatz und Laufzeit

B E/A - Modus-Wahl Default: **+B**
Es wird die Zugehoerigkeit der Standard E/A-Files input und output zu den logischen E/A-Geraeten CON und TRM festgelegt. Der Gueltigkeitsbereich bezieht sich auf den ganzen Programm-Block und kann im Programm nicht geaendert werden.
Es gilt: {$B+} --> CON
{$B-} --> TRM

C ^S und ^C Default: **C+**
Es wird definiert, ob die Ctrl-Zeichen bei der Eingabe ueber die Console intrpretiert werden oder nicht. Gueltigkeitsbereich ist der gesamte Programm-Block.
Es gilt: {$C+} --> ^C fuehrt zum Programmabbruch
^S stoppt die Bildschirmausgabe
{$C-} --> Ctrl-Zeichen werden nicht interpretiert
Die aktive Direktive verlangsamt die Bildschirmausgabe.

I E/A-Fehler-Behandlung Default: **I+**
Definiert die Behandlung von E/A-Fehlern.
{$I+} --> Fehlerbehandlung durch das System
{$I-} --> Fehlerbehandlung kann mittels ioresult selbst definiert werden.

R Index- und Unterbereichsueberwachung Default: **R-**
Legt die Ueberwachung von Indexgrenzen von Arrays und die Zuweisungen an Variable von Aufzaehlungs- und Unterbereichstypen zur Laufzeit fest. Die Ueberwachung verlangsamt die Abarbeitung.
{$R+} --> Ueberwachung eingeschaltet
{$R-} --> Ueberwachung ausgeschaltet

U Benutzer-Interrupt Default: **U-**
{$U+} --> die Eingabe von ^C bricht Programm zur Laufzeit ab; fuehrt zur Verlaengerung der Laufzeit
{$U-} --> keine Unterbrechung durch ^C

V Test von Referenzparametern Default: **V+**

Die Direktive bezieht sich auf Referenzparameter vom Typ string.

{$V+} --> die Laenge von aktuellem und formalen Parameter muss uebereinstimmen.

{$V-} --> die Laengen koennen unterschiedlich sein

W Schachtelung von with - Anweisungen Default: **W2**

Der Schalter legt fest, wieviel Records innerhalb eines Blockes eroeffnet werden duerfen. Dem Buchstaben W muss eine Ziffer zwischen 0..9 folgen.

X ARRAY - Optimierung Default: **X+**

{$X+} --> Felder werden optimiert (Zugriff wird schneller aber speicherplatzintensiver

{$X-} --> Felder werden nicht optimiert

O laufwerk

Der Schalter weist dem Overlay das angegebene Laufwerk zu.

3. Bedienung des Druckprogramms **TLIST**

Der Druck von Quelltexten erfolgt mit dem Programm **TLIST**. Eine Druckausgabe des Programmtextes waehrend des Compilierens ist nicht vorgesehen.

Das Programm TLIST wird unter seinem Namen aufgerufen, entweder direkt vom Betriebssystem aus oder vom TURBO aus mit dem e**X**ecute-Kommando.
In TLIST koennen folgende Options gesetzt werden:

L - Die Programmtextzeilen werden fortlaufend numeriert,mit INCLUDE eingefuegte Textbausteine werden im Druck durch ein I vor der Zeilennummer gekennzeichnet.

M - reservierte PASCAL-Worte werden unterstrichen (auch in Kommentaren)

Beide Options koennen kombiniert werden.

3.1. Drucksteuerkommandos

Auf Quelltextebene sind ebenfalls Drucksteuerkommandos moeglich. Sie sind als Kommentare in das Quellprogramm einzufuegen, das Symbol fuer Kommentarbeginn muss in Spalte 1 stehen, der Punkt in Spalte 2.

Funktion	Kommando
Seitenlaenge beim Druck	.PL<zahl> (zB .PL72)
Seitenwechsel	.PA
neue Seite,wenn fuer <zahl> Zeilen nicht mehr genug Platz ist	.CP<zahl>
<zahl> Leerspalten am linken Rand	.PO<zahl>
<Kopftext> (wird auf jeder neuen Seite	

oben gesetzt)	.HE<kopftext>
<fusstext> (wird auf jeder neuen Seite unten gesetzt)	.FO<fusstext>
	in Kopf- oder Fusstext kann eine Seitennummer mit # gesetzt werden
Unterdruecken/Anschalten der Ausgabe	.L-/.L+
mit INCLUDE in den Quelltext eingefuegte Teile werden mit gedruckt/nicht gedruckt	.I+/.I-

Drucksteueranweisungen werden bei der Zeilenzaehlung mitgezaehlt, genauso durch Setzen von L- nicht gedruckte Quelltextzeilen.

Abweichungen von TURBO

gegenueber Standard-PASCAL

1. TURBO-Spracherweiterungen

Bevor die in TURBO realisierten Spracherweiterungen im Einzelnen beschrieben werden, muss auf folgendes hingewiesen werden:
Bedingt durch den nicht vorhandenen Linker ist ein **Einbinden** von in anderen Programmiersprachen (speziell Assembler) geschriebenen Programmteilen prinzipiell nicht moeglich! Ebenso ist eine Verwen-dung von TURBO-Programmteilen in anderen Progammiersprachen normalerweise ausgeschlossen.
Alle beschriebenen Spracherweiterungen sind TURBO-spezifisch. Es ist nicht gesichert, dass in anderen PASCAL-Implementierungen gleiche Erweiterungen vorhanden sind. (Ein Standard fuer Erweiterungen exitiert (noch!) nicht). Falls Programme unter dem Gesichtspunkt der Portabilitaet entwickelt werden, ist die Verwendung der Erweiterungen sorgfaeltig zu ueberpruefen.
Das gilt insbesondere fuer direkte Betriebssystemaufrufe (BDOS/BIOS) und prozessorspezifische Funktionen (PORT, STACKPTR, MEM) und die in TURBO moegliche wahlfreie Reihenfolge der Deklarationen (Standardreihenfolge ist LABEL, CONST, TYPE, VAR!) sowie die Verwendung der in TURBO moeglichen nichtnumerischen Marken (Standard : nur numerische Marken!).

1.1. Bildschirmsteuerung

Zur Steuerung der Bildschirmausgabe stellt TURBO folgende Proze-duren bereit:

CLRSCR - Loeschen Bildschirm von Beginn bis Ende,Cursor auf Bildschirmanfang.

CLREOL - Loeschen bis Zeilenende.

GOTOXY(x,y) - Positioniert den Cursor in die X-te Spalte,Y-te Zeile Bildschirmanfang=0,0.x und y koennen Konstante, Variable oder numerische Ausdruecke sein.

DELAY(t) - Verzoegert die Programmabarbeitung um ca t ms. (t vom Typ Integer)

Die Prozeduren DELLINE, INSLINE fuer loeschen und Einfuegen einer Zeile ab Cursorposition sind in ihrer Wirkung abhaengig von der Installation (in vorliegender Variante nicht installiert). Die Funktionen lassen sich ueber die fuer die jeweilige CP/M-Version ueblichen Control-Funktionen zur Bildschirmsteuerung simulieren.

Diese Control-Funktionen sind unter DAC,BCU,SCPX:
^L - Loeschen Bildschirm,Cursor oben links

```
 ^T    - Loeschen  Bildschirm  ab  Cursorposition,Cursor  bleibt
         stehen

 ^H    - Cursor 1 Pos. links
 ^U    - Cursor 1 Pos. rechts
 ^M    - Cursor auf Zeilenanfang
 ^J    - Cursor 1 Zeile nach unten in gleicher Spalte
 ^Z    -   "    1   "    "   oben  "     "       "
```

Die Prozeduren NORMVIDEO, LOWVIDEO und HIGHVIDEO zur Helligkeits-steuerung der Bildschirmanzeige sind in der vorliegenden Variante aus technischen Gruenden nicht installiert.

1.2. Verwendung von Control-Funktionen in Quellprogrammen

TURBO bietet fuer die Verwendung von Control-Funktionen in Quellprogrammen folgende Unterstuetzung:
Fuer die Angabe von Control-Funktionen koennen ASCII-Zeichen oder die dezimlen Werte der Control-Funktionen direkt verwendet wer-den.
ASCII-Zeichen werden mit dem Symbol ^ eingeleitet, numerische Werte mit dem Symbol #. Diese Zeichenkombinationen koennen in Ein-Ausgabeanweisungen verwendet werden.

Beispiele:

```
 CONTROL-A  =^A
            =#1

 z.B.  WRITE(^L);    { Bildschirm loeschen          }
       WRITE(#12);   { dezimal ^L                   }

        IF A=^C THEN BDOS(0); { WARMSTART           }
```

Zur Beachtung : Angabe ohne '' (keine Zeichenketten!)

1.3. zusaetzliche Datentypen

In TURBO-PASCAL sind gegenueber Standard-PASCAL zwei zusaetzliche Datentypen vorhanden:

Datentyp **BYTE**
BYTE entspricht einer ganzen Zahl zwischen 0 und 255. Der Typ ist vertraeglich mit **INTEGER** und **REAL**.

Datentyp **STRING[<zahl>]**
STRING entspricht einer Zeichenkette der Laenge <zahl>. In PASCAL-Notation wuerde der Typ STRING entsprechen:

```
  TYPE STRING=PACKED ARRAY[1..<zahl>] OF CHAR;
```

Mit STRING ist eine bequeme Manipulation von Zeichenketten moeg-lich.

1.4 String-Manipulation in TURBO

In TURBO-PASCAL stehen zur direkten String-Manipulation folgende Funktionen/ Prozeduren zur Verfuegung:

Prozeduren

DELETE, INSERT, STR, VAL

Funktionen

CONCAT, COPY, LENGTH, POS
Die maximale Stringlaenge betraegt 255 Zeichen.

DELETE(S,P,N) S:STRING[x],P,N:INTEGER

Im String S werden ab dem p-ten Byte n Byte geloescht. Sollte p groesser sein als die Stringlaenge, tritt ein Laufzeitfehler(Programmabbruch) ein!

INSERT(Q,Z,P) Q,Z:STRING,P:INTEGER

Im String Z wird ab Pos. P der String Q eingefuegt.

STR(V,S) V:INTEGER | REAL,S:STRING

Der Zahlenwert V wird in einen String S umgewandelt.

VAL(S,V,C) S:STRING,V:INTEGER | REAL,C:INTEGER

Der String S wird in den Zahlenwert V umgewandelt. Sollte in S ein nicht umwandelbares Zeichen stehen, wird die Umwandlung nicht ausgefuehrt und C zeigt die Position des fehlerhaften Zeichens an. Bei korrekter Umwandlung ist C=0.

CONCAT(S1,...Sn):STRING S1,...Sn:STRING

Die Strings S1,...Sn werden zu einem String verkettet, wobei die maximale Stringlaenge nicht ueberschritten werden darf.

COPY(S,P,N):STRING S:STRING,P,N:INTEGER

Aus dem String S werden ab Pos.P N Zeichen entnommen und dem Zielstring zugewiesen.

LENGTH(S):INTEGER S:STRING

Es wird die aktuelle Laenge (die tatsaechlich belegte Laenge) des String S ermittelt und an die Zielvariable uebergeben.

POS(Q,Z):INTEGER Q,Z:STRING

Im String Z wird die Position des ersten Auftretens des Strings Q ermittelt. Ist Q nicht in Z enthalten,wird die Zielvariable auf 0 gesetzt.

Das folgende Beispiel demonstriert die Moeglichkeiten der String-manipulation:

```
PROGRAM STRDEMO;
{                                                                    }
{ Demonstrationsprogramm zur Stringmanipulation                      }
{                                                                    }
VAR I      :INTEGER;
    R      :REAL;
    KETTE1 :STRING[50];
    KETTE2 :STRING[20];
    KETTE3 :STRING[10];
BEGIN
   CLRSCR;
   WRITELN('Testprogramm STRING-Verabeitung':50);
   WRITELN;
   KETTE1:='ABCDE';
   I:=LENGTH(KETTE1);     { AKTUELLE STRINGLAENGE              }
   WRITELN(KETTE1,' IST ',I,' Zeichen lang');
   KETTE2:='FGH';
   KETTE1:=CONCAT(KETTE1,KETTE2);  { VERKETTUNG                }
   I:=LENGTH(KETTE1);
   WRITELN(KETTE1,' ist jetzt ',I,' Zeichen lang');
   KETTE3:=COPY(KETTE1,4,3);       { KOPIEREN                  }
WRITELN(KETTE3,'wurde gebildet aus',KETTE1,' ab Pos.4,Laenge 3');
   KETTE3:='E';
   I:=POS(KETTE3,KETTE1);          { Positionn ermitteln       }
   WRITELN('In ',KETTE1,' steht ',KETTE3,' an ',I,'-ter Stelle');
   WRITELN;
   DELETE(KETTE1,4,3);             { ZEICHEN LOESCHEN          }
   WRITELN(KETTE1,' nach Loeschung von 3 Zeichen ab Pos 4');
   KETTE3:='XXX';
   INSERT(KETTE3,KETTE1,4);        { Kette einfuegen           }
   WRITELN(KETTE1,' nach Einfuegen von ',KETTE3,' ab Pos 4');
   KETTE2:='12.123';
   VAL(KETTE2,R,I);            { Umwandeln String in REAL-Zahl  }
   WRITELN(R:3:3,'  ',I,'=Pos.fehlerhaftes Zeichen,falls I>0');
   KETTE2:='12,123';
   VAL(KETTE2,R,I);
   R:=100.0;
  WRITELN(R:3:3,'  ',I,'=Fehlerort');{ Fehlerfall,I=Fehlerort(3)}
END.
```

1.5. zusaetzliche numerische Funktionen

An numerischen Funktionen werden zusaetzlich bereitgestellt:
FRAC:Gibt bei REAL-Zahlen den Teil der Zahl nach dem Komma an. Bei INTEGER-Zahlen ergibt FRAC den Wert 0.
INT :Gibt bei REAL-Zahlen den Teil der Zahl vor dem Komma an.INT bei INTEGER-Zahlen ergibt den Ursprungswert, allerdings als REAL-Groesse.

Beispiele:

```
 VAR A,B,C:REAL;
     D    :INTEGER;
  .
```

```
  A:=1.2345;
  D:=34;
  B:=FRAC(D);  { B hat den Wert 0.0    }
  C:=INT(D);   { C hat den Wert 34.0   }
  B:=FRAC(A);  { B hat den Wert 0.2345 }
  C:=INT(A);   { C hat den Wert 1.0    }
```

1.6. Vordefinierte Arrays

In Turbo sind zwei Arrays vom Typ BYTE vordefiniert: **PORT** und **MEM**

PORT gestattet einen direkten Portverkehr; **MEM** direkte Speicheroperationen.

Ausgeben eines Wertes auf ein Port:
PORT[<portnummer>]:=<wert> (portnummer,wert :integer)

Lesen eines Wertes von einem Port:
<wert>:=port[<portnummer>]

Beispiel: Umschalten des BAB3 auf Grossdarstellung (480 Zeichen):

```
  .
VAR GROSS,PORTNR : INTEGER;
  .
  .
GROSS:=$02;
PORTNR:=$40;
PORT[PORTNR]:=GROSS;
  .
  .
```

Direkte Speicheroperationen:

```
  .
VAR I:BYTE;
  .
  .
MEM[$F800}:=$41;{AUSGABE DES BYTES $41 AUF ADRESSE $F800       }
I:=MEM[$F8A1];{ I ENTHAELT DEN INHALT DES SPEICHERPLATZES $F8A1}
 .
```

1.7. BDOS und BIOS-Aufrufe

In TURBO (Version fuer CP/M Z80)sind direkte Aufrufe von BDOS und BIOS moeglich.
Es werden dafuer die Funktionen/Prozeduren BDOS/BDOSHL UND BIOS/BIOSHL bereitgestellt. Die Funktionen/Prozeduren xxxx und xxxxHL unterscheiden sich in der Parameteruebergabe, welche verwendet wird, muss an Hand der konkreten BDOS/BIOS-Funktion entschieden werden.

Moegliche Arten des Aufrufes sind:

```
 <z1>:=BDOS(<nummer>,<z2>);
       nummer=Nummer der Funktion,z1,z2:INTEGER
```

```
 BDOS(<nummer>,<z2>);

  ,<z2>  kann bei einigen Funktionen weggelassen werden.(s.Anlage
     1,Liste der BDOS-Funktionen)
```

Analoges gilt fuer BIOS/BIOSHL.

Die Bedeutung von <z1>,<z2> ist je nach BDOS/BIOS-Funktion unterschiedlich und muss aus entsprechender Spezialliteratur entnommen werden. Das folgende Beispiel soll als Demonstration dienen, nicht als Gebrauchsanweisung. Generell sollte von BDOS- u.BIOS-Funktionen nur dann Gebrauch gemacht werden, wenn ihre Leistungen und Wirkungen im konkreten System voll ueberblickt werden.

Beispiel:

```
PROGRAM BDOSTEST;
{                                                              }
{direkter Aufruf der BDOS-Funktionen mit TURBO-PASCAL          }
{                                                              }
VAR I,J,K,DRIVE:INTEGER;
    ZEILE      :STRING[80];
    ZEICHEN    :BYTE;
{$I PWIHEX.PAS}{ Prozedur WRITE_I_HEX f.hex-Ausgabe v.INTEGER   }
BEGIN
    CLRSCR;
    WRITELN('BDOS-Funktionen in TURBO-PASCAL':30);
    WRITELN;
    I:=BDOS(12);{ BDOS-Funktion 12 -> Versionsnummer           }
    WRITE(I,'/');   { dezimale Ausgabe von $0022 = 34          }
    WRITE_I_HEX(I); { hexadezimale Ausgabe 0022 -> CP/M 2.2    }
    WRITELN;
    ZEILE:='DAS IST EIN TEXT$';
    J:=ADDR(ZEILE); { Adresse der Variablen 'Zeile'            }
    BDOS(9,J);      { BDOS-Funktion 9 -> Ausgabe von 'ZEILE'   }
    WRITELN;
    WRITE(J,'/');{Ausgabe der Adresse der Zeichenkette dezimal  }
    WRITE_I_HEX(J); { und hexadezimal                          }
    WRITELN;
    ZEICHEN:=$41;   { hexadezimal 41 = Zeichen A               }
    BDOS(5,ZEICHEN); { BDOS-Funktion 5 -> Zeichenausgabe Drucker}
    WRITELN(CHR(ZEICHEN));
    BDOS(13);       { Ruecksetzen Diskettensystem (DISK RESET)  }
    WRITELN;
    I:=BDOSHL(31);{Adresse LW-Parameter,wird in HL uebergeben ! }
    WRITE_I_HEX(I);{Ausschrift ist abhaengig v.konkreten System }
END.
```

1.8. Retten/setzen vordefinierter Pointer

In TURBO sind fuer die Speicherverwaltung folgende Pointer vordefiniert:

```
 STACKPTR  -  CPU-Stackpointer
 RECURPTR  -  Stackpointer bei Rekursionen
 HEAPPTR   -  Kellerspeicher-Pointer
```

Es ist darauf zu achten, dass immer gilt :

```
 HEAPPTR < RECURPTR < STACKPTR
```

Sollte das bei Manipulationen an den genannten Pointern nicht gewaehrleistet sein, koennen Programmabstuerze auftreten, da von TURBO nicht geprueft wird, ob sich die durch diese Pointer definierten Variablen nicht gegenseitig ueberschreiben (s.Teil 3 - dynamische Variable und Pointer).

Als (Nicht zur Nachahmung empfohlenes) Beispiel sei hier ein Zugriff auf den Stackpointer der CPU gezeigt.

```
 <var> : INTEGER;
STACKPTR:=<var>;    { schreiben Stackpointer                    }
<var>:=STACKPTR;    { lesen Stackpointer                        }
```

BEISPIEL:

```
PROGRAM STACKTST;
{                                                               }
{     Testprogramm lesen/schreiben STACKPOINTER                 }
{                                                               }
VAR I:INTEGER;
{$I PWIHEX.PAS}{ einfuegen Prozedur WRITE_I_HEX                 }
BEGIN
  CLRSCR;
  I:=0;
  WRITELN('TEST STACKPTR':46);
  I:=STACKPTR;
  WRITE(I,'/');
  WRITE_I_HEX(I);
  STACKPTR:=I;
  WRITELN;
END.
```

Fuer die Ein-/Ausgabe sind folgende Pointer vordefiniert:

Zeiger	vordefinierte Adresse
CONSTPTR	CONST-Funktion aus BIOS
CONINPTR	CONIN-Funktion aus BIOS
CONOUTPTR	CONOUT-Prozedur aus BIOS
LSTOUTPTR	LSTOUT-Prozedur aus BIOS
AUXOUTPTR	AUXOUT-Prozedur aus BIOS
AUXINPTR	AUXIN-Funktion aus BIOS
USROUTPTR	USROUT-Prozedur aus BIOS
USRINPTR	USRIN-Funktion aus BIOS

Alle Ein-/Ausgaben laufen bei TURBO standardmaessig direkt ueber BIOS. Deshalb ist normalerweise eine Protokollierung der Bildschirmausgaben mit ^P in TURBO nicht moeglich. Eigene Ein-/Ausgaberoutinen sind dadurch anschliessbar, das den vordefinierten Pointern die Adresse der eigenen Routine zugewiesen wird. Ein Beispiel dafuer ist in der Beispielsammlung enthalten.
Die Uebergabebedingungen des BIOS sind auch bei eigenen Prozeduren/Funktionen einzuhalten.

1.9. CHAIN und EXECUTE

CHAIN und EXECUTE erlauben den Aufruf von Programmen von anderen Programmen aus

CHAIN(<filename>); <filename> ist ein ungetyptes File, dem mit der ASSIGN-Funktion der Name des aufgerufenen Programmes zugewiesen wird. Dieses Programm muss die Erweiterung .CHN haben und wird mit der compiler **O**ption cHn-file erzeugt. Das aufgerufene Programm kehrt nicht ins rufende Programm zurueck, eine Datenuebergabe rufend->gerufen erfolgt nicht.

Eine Uebergabe von Werten ist nur folgendermassen moeglich:
Im aufrufenden und im gerufenen Programm mussen die Variablen, die zu uebergebende Werte enthalten sollen, an gleichem Ort und in gleicher Reihenfolge definiert werden. Sie muessen am Anfang der Variablenliste stehen.
Eine weitere Moeglichkeit ist die Verwendung absoluter Variabler (s.Anlage 4).

Ein mit der Compiler-Option c**H**n-file uebersetztes Programm ist um ca.8 kByte kleiner als ein mit der Option **C**om-file uebersetztes Programm, da das Laufzeitsystem nicht mit abgespeichert wird (es wird vom rufenden Programm uebernommen !).
<filename> und Programmname muessen nicht uebereinstimmen.

EXECUTE gestattet den Aufruf von beliebigen Programmen (Namenserweiterung COM). Das gerufene Programm wird auf Adresse $100 geladen, ueberschreibt also das rufende Programm.

Ein gerufenes Programm kann bestimmen, ob es von einem TURBO-Programm mit CHAIN/EXECUTE oder anders aufgerufen wurde:
TURBO legt in der Kommandozeile auf $80 den Wert $FF ab, falls CHAIN oder EXECUTE aktiviert werden. Beim Aufruf von nicht-TURBO-Programmen ist darauf zu achten, dass sie beim Aufruf nicht die Kommandozeile in Adresse $80 belegen, da sonst unklare Zustaende auftreten koennen.

Im folgenden Beispiel ruft das Programm CHTEST das Programm CHAIN auf. Es ist zu beachten, dass in der ASSIGN-Prozedur der vollstaendige Name des aufzurufenden Programmes anzugeben ist.

```
program CHTEST;

{ TESTPROGRAMM FUER CHAIN-FUNKTION,KEINE DATENUEBERGABE         }
var chfile:file;
begin
  assign(chfile,'a:chain.chn');
  clrscr;
  writeln('Programm 1');
  chain(chfile); { Aufruf des Programmes CHAIN.CHN von Lw.A     }
end.

program CHAIN;
begin
```

```
  writeln('Programm 2');
end.
```

CHAIN und EXECUTE sind im Direktmode nicht ausfuehrbar!

Neben CHAIN und EXECUTE sind ab TURBO-Version 2.00x echte Overlays moeglich. (s.dazu Anlage 4), die ebenfalls im Directmode nicht ausfuehrbar sind.

1.10. Erweiterte CASE-Anweisung

In der CASE-Anweisung ist als letzte Verzweigung ein **ELSE** zugelassen.

Beispiel:

```
  .
 VAR A:CHAR;
  .
 CASE A OF
    'A':WRITE('A');
    'B':WRITE('B');
 ELSE
        WRITE('anderer Buchstabe');
 END;
```

1.11. getypte Konstante

In TURBO ist es moeglich, Konstanten mit einer Typangabe zu versehen. Diese getypten Konstanten koennen im Programm wie normale Konstante oder wie Variable verwendet werden. Getypte Konstanten erlauben damit eine Anfangswertzuweisung an Variable.

Beispiele:

```
CONST  A:INTEGER   =104;
       B:REAL      =12.1234;
       C:STRING[20]='Das ist Text';
```

2. vordefinierte Textdateien

Die Standard-Textdateien **input** und **output** werden vom Compiler eroeffnet. Jede Verwendung von read/readln oder write/writeln stellt einen Zugriff auf diese Dateien dar.
Ausser den Standard-Textdateien **input** und **output**, die **CON** bzw. **TRM** zugeordnet sind, existieren im TURBO - Pascal weitere vordefinierte Textdateien. Bei diesen Dateien ist generell der Aufruf von **assign, reset, rewrite** und **close** verboten. Der Zugriff auf diese Dateien erfolgt durch Verwendung ihres Namens in Ein/Ausgabeanweisungen. Die Angabe mehrerer Eingabevariabler in READ/READLN ist zulaessig. Sie sind in der Variablenliste durck Komma zu trennen, bei der Eingabe durch Leerzeichen.

Beispiele:

```
 WRITE(LST,'Text auf Drucker'); { TEXTAUSGABE AUF DRUCKER        }
 READLN(KBD,PASSWORT);          { EINGABE DER VARIABLEN 'PASSWORT'}
                              { OHNE ANZEIGE AUF DEM BILDSCHIRM    }
```

Es gilt folgende Zuordnung:

input - CON oder TRM zugeordnet (Primaere Eingabedatei)
output - CON oder TRM zugeordnet (Primaere Ausgabedatei)
con - CON zugeordnet
trm - TRM zugeordnet
kbd - KBD zugeordnet,die Eingaben werden nicht auf dem Bildschirm angezeigt,eine Ausgabe auf KBD ist nicht moeglich.
lst - LST zugeordnet,die Ausgaben erfolgen auf den Drucker, eine Eingabe von der Datei LST ist nicht moeglich
aux - AUX zugeordnet.Die Ein-/Ausgaben werden ueber den Leser/Stanzerkanal des Betriebssystems gefuehrt.Ist kein Leser/Stanzerkanal im Betriebssystem -> nicht verwenden.
usr - USR zugeordnet

Die Proceduren **seek** und **flush** sowie die Funktionen **filepos** und **filesize** sind nicht auf Dateien des Types Text anwendbar !

Bei der Benutzung der Funktionen **eoln** und **eof** ist zu beachten, dass sie sich bei Diskettendateien immer auf das naechste im Puffer anstehende Zeichen beziehen, dagegen bei logischen Geraeten auf das zuletzt gelesene Zeichen !

2.1. formatierte Ausgabe mit **write/writeln**

Bei der Verwendung von Textdateien ist eine Formatierung der Ausgabe moeglich.

Aufruf: write (f,e);

Parameter: VAR f:text
e ein Ausdruck mit Formatangaben der Form e:n oder e:n:m (m,n:integer)

e:char nach n-1 Blank wird char ausgegeben; fehlt n wird n=1 gesetzt
e:string der String e wird rechtsbuendig in ein Feld von n Zeichen eingetragen; fehlt n wird n=length (e) gesetzt
e:boolean es werden 'true' bzw. 'false' rechtsbuendig in ein Feld von n Zeichen geschrieben
e:integer der Wert von e wird ins Dezimalsystem konvertiert und rechtsbuendig in das Feld mit der Laenge n eingetragen
e:real der Wert von e wird ins Dezimalsystem konvertiert und rechtsbuendig in ein Feld der Groesse n mit m Stellen nach dem Dezimalpunkt eingetragen.
Fehlt m (0 <= m <= 24) erfolgt die Ausgabe in ein

Feld von n Zeichen in der Form
x.xxxxxxxxxE+dd (e >= 0)
-x.xxxxxxxxxE+dd (e <= 0)
Wenn die Angaben fuer m und n fehlen, wird der Wert 18 fuer die Feldgroesse eingesetzt
Die Angabe eines unzulaessigen Parameters fuehrt zu Syntaxfehler.

Durch Verwendung von WRITELN ohne Variable wird ein Zeilenvorschub erzeugt.

3. Einschraenkungen von TURBO gegenueber Standard-PASCAL

In TURBO sind gegenueber Standard-PASCAL folgende Einschraenkungen:

PUT und **GET** sind nicht implementiert. Der Dateizugriff erfolgt mit den Prozeduren WRITE/WRITELN und READ/READLN, die in Standard-PASCAL vorhanden sind.

GOTO-Spruenge aus dem jeweils aktivierten Block heraus sind nicht zulaessig.

PAGE ist als Standard-Prozedur nicht implementiert, da es kein in allen CP/M-Systemen einheitliches Steuerzeichen fuer Seitenvorschub gibt.

PACKED wird als Schlueselworte vom Compiler akzeptiert, hat aber keine Wirkung. Die Prozeduren PACK und UNPACK sind nicht implemen-tiert.

Prozeduren und Funktionen sind nicht als Parameter in Prozeduren und Funktionen zugelassen.

In NEW sind keine Variant-Record-Strukturen zugelassen. Diese Restriktion kann durch Verwendung der GETMEM-Prozedur umgangen werden.

In rekursiven Prozeduren sind keine lokalen Variablen als Parameter zugelassen.

Zeiger und dynamische Speicherverwaltung

1. Pointer

Der POINTER-Typ ist definiert durch das Zeichen ^ gefolgt vom Typbezeichner der dynamischen Variablen, auf die mit diesem Pointer bezug genommen weden soll.
Pointer gleichen Typs koennen verglichen werden mit = <>, das Ergebnis ist vom Typ BOOLEAN. Andere direkte Operationen mit Pointern sind nicht zulaessig.
Der vordefinierte Pointer NIL ist mit allen Pointertypen vertraeglich und hat den Wert 0.

Beispiel fuer Pointerdefinition:

```
  .
 TYPE PersonPointer = ^Person;
      Person        = RECORD
                        name      :STRING[50];
                        beruf     :STRING[50];
                        naechster :PersonPointer;
                      END;
 VAR Erste_Person,Letzte_Person,Neue_Person:PersonPointer;
  .
```

Die mit VAR definierten Variablen sind Zeigervariable (Pointer), die auf Records des Typs PERSON zeigen koennen.

Die Zuordnung von Speicherplatz zu einer dynamischen Variablen kann mit **NEW** (pointervariable) erfolgen, z.B. NEW (Erste_Person). Dabei wird genau soviel Speicherplatz zugewiesen wie die durch den Pointer definierte Variable benoetigt.

Eine andere Moeglichkeit stellt die Verwendung von GETMEM dar.

GETMEM(Pvar,I) Pvar:POINTER;I:INTEGER

Pvar ist eine beliebige Pointervariable, I gibt den zur durch den Pointer definierten dynamischen Variablen zuzuordnenden Speicherplatz an. Die Steuerung der Groesse des Speicherplatzes liegt hier also beim Programmierer.

Dynamische Variable, denen Speicherplatz zugewiesen wurde, werden in einem Kellerspeicher (heap genannt) abgelegt. Dieser Speicher beginnt beim ersten freien Byte im Hauptspeicher (also hinter dem Programm) und dehnt sich in Richtung auf die hoeheren Adressen aus. TURBO steuert den heap durch Wertzuweisung an den heappointer, der zu Programmbeginn auf die erste freie Adresse gestellt wird. Bei jeder Zuweisung mit NEW/GETMEM wird der heappointer auf die naechste freie Adresse gestellt.

Die Standardfunktion **MEMAVAIL** liefert als INTEGER-Wert die Groesse des momentan noch verfugbaren heap in BYTE, **MAXAVAIL** die Groesse des groessten zusammenhaengenden freien Speicherplatzes des heap in BYTE.

Die Standardfunktionen **ORD** und **PTR** dienen der gegenseitigen Umwandlung von Pointer in Integer-Groessen.

ORD(pointer) liefert als INTEGER die im Pointer enthaltene Adresse.

PTR(var) wandelt die INTEGER-Variable var in einen Pointer, der auf die in var als INTEGER angegebene Adresse zeigt.

Die Freigabe dynamisch belegten Speicherplatzes erfolgt entweder mit **MARK/RELEASE** oder **DISPOSE**, soweit er durch NEW angelegt wurde oder im Falle des Anlegens mit GETMEM durch **FREEMEM**. Dabei kann mit FREEMEM nur genausoviel Speicherplatz freigegeben werden, wie mit dem letzten GETMEM belegt wurde.

Eine Freigabe von Werten im Kellerspeicher kann nur von der letzten belegten Variablen her erfolgen, eine Freigabe eines Wertes mitten im heap ist nicht moeglich. Das gilt genauso fuer Zugriffe auf dynamisch verwaltete Variable.
Prizip:"first in -> last out" .

Die Freigabe kann auf zwei Arten erfolgen:
Mit DISPOSE (pointervar) werden alle durch pointervar definierten Variablen freigegeben, einZugriff auf sie ist nicht mehr moeglich.
Mit MARK(poiintervar) wird eine durch pointervar definierte Variable markiert, RELEASE(pointervar) loescht den heap bis zur durch MARK markierten Stelle.
Die Feststellung der absoluten Adresse einer Variablen kann auch durch die ADDR-Funktion erfolgen.
ADDR(var) liefert die Adresse der Variablen var als INTEGER-Zahl. Ist var ein strukturierter Typ, gibt ADDR die Adresse des ersten durch var belegten BYTES an. Ein Zugriff auf Elemente einer strukturierten Variablen ist moeglich.

2. Kellerspeicher(heap) und Stacks

Der Stackpointer ist vordefiniert auf die freie Stelle des Hauptspeichers mit der hoechsten Adresse. Der RECURPTR ist vordefiniert 1 kBYTE ($400) kleiner als der Stackpointer.
Das System prueft bei jeder Zuweisung mit NEW/GETMEM oder bei rekursiven Prozeduren, ob HEAPPTR<RECURPTR. Wenn das nicht mehr der Fall ist, tritt ein Laufzeitfehler auf. Keine Pruefung erfolgt, ob nicht eventuell der Stackpointer in Richtung RECURPTR ueberlaeuft (das kann auftreten, wenn eine rekursive Prozedur sich mehr als 300 bis 400 mal aufruft).
Fuer diesen Fall muss STACKPTR vergroesert werden. Dazu dient folgende Zuweisung:

```
RECURPTR:=STACKPTR-2*maxdepth-512;
```

maxdepth ist die maximale Schachtelungstiefe der Aufrufe der rekursiven Prozedur; die 512 BYTE werden extra als Platz fuer Parametertransfer und fuer Zwischenergebnisse benoetigt.

3. Speicherbelegung

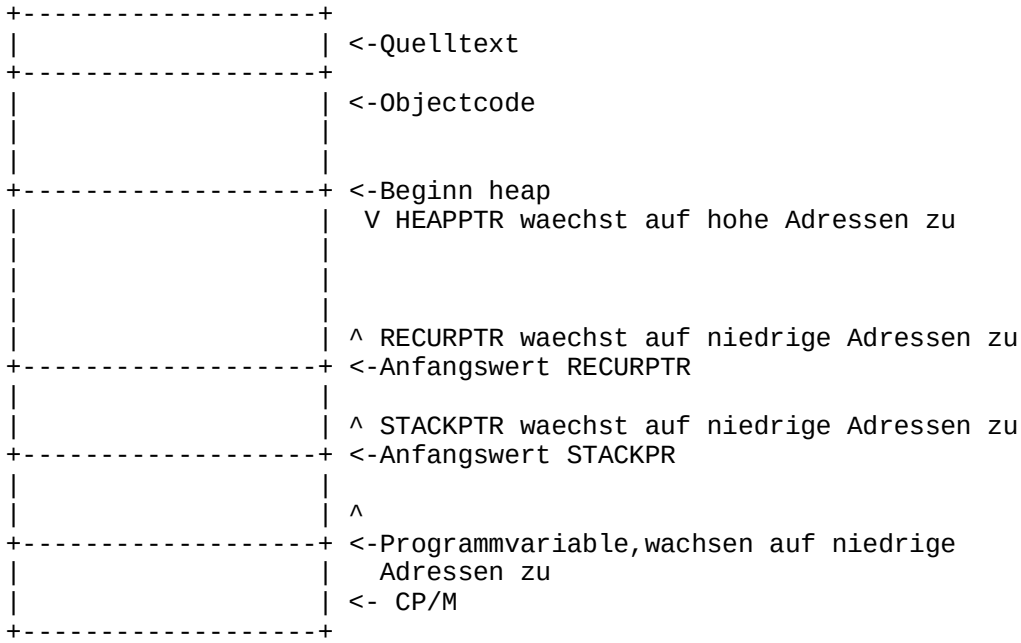
3.1. Uebersetzung im Speicher (Directmode)

3.2. Uebersetzung in Richtung Diskette (Optionen C u.H)

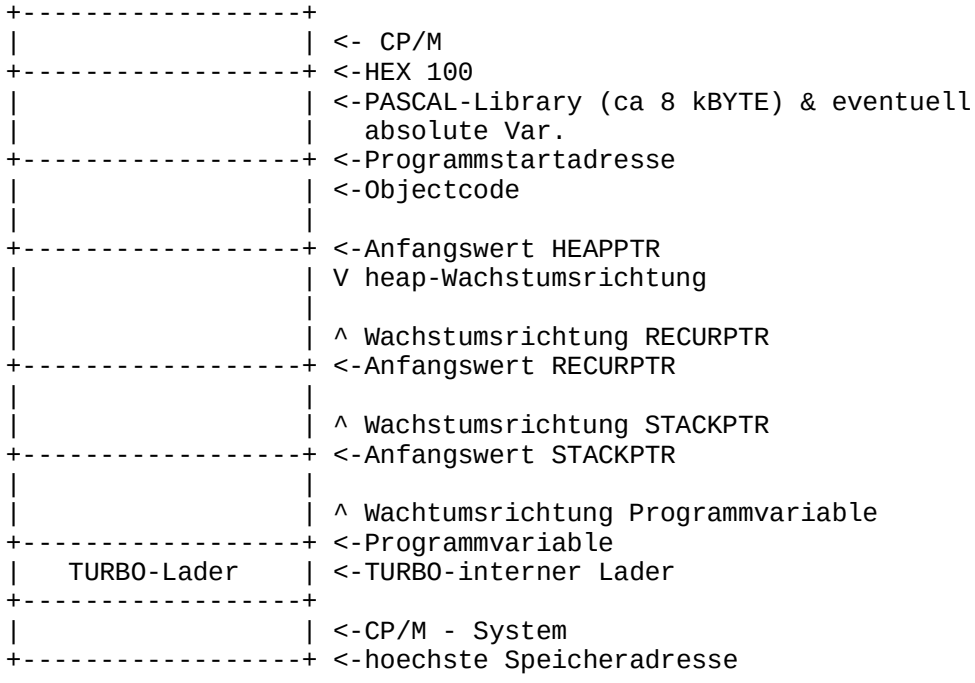
Der Speicheraufbau ist bis Quelltext identisch. Anstelle des Objectcodes ist ein (relativ kleiner) Pufferbereich angelegt, der die uebersetzten Programmteile aufnimmt, bis sie auf Diskette ausgelagert werden.
Der hoeherwertige Teil des Speichers ist wieder identisch mit 3.1. (also Symboltabelle, CPU-Stack und Betriebssystem).

3.3. Ausfuerung im Speicher

Bis zum Quelltext ist der Speicher wieder identisch zu 3.1., d.h. der Quelltext bleibt auch bei Ausfuerung im Speicher. Oberhalb der Quelltextes ergibt sich folgender Aufbau:



3.4. Ausfuerung eines COM-Files



ZUSAMMENFASSUNG

Die vorliegende Zusammenfassung gilt fuer den gesamten in TURBO-PASCAL zur Verfuegung stehenden Sprachumfang. Eine Unterscheidung zwischen Standard-PASCAL und TURBO-PASCAL wird dabei nicht getroffen.

Konstanten

false	-	boolean	
true	-	boolean	
pi	-	real	3.1415926536E+00
maxint	-	integer	32767
nil	-	pointer	leer

Datentypen

boolean - true und false wobei false < true gilt

byte - 0..255 Unterbereichstyp von integer und
mit diesem kompatibel

char - Darstellung 8 Bit ASCII - Zeichensatz

integer - Zahlenbereich von -32768..32767 bzw.
hexadezimal $0000..$FFFF
0.. 32767 entspricht hex. $0000..$7FFF
-1..-32768 entspricht hex. $FFFF..$8000

real - Zahlenbereich von 1E-38 bis 1E+38
11 signifikante Stellen (intern 48 Bit)
overflow fuer zu Fehlerabbruch,
underflow fuehrt zum Wert 0

text - entsprechend ISO - Pascal

string[i] - Zeichenkette mit einer Laenge 0<i<=255
(i:integer)

set - Menge (bis zu 255 Elemente)

Vordefinierte Textdateien

AUX - auxiliary device

CON - console: Ausgabe ueber Bildschirm
Eingabe ueber Tastatur
Eingabe erfolgt zeilenweise gepuffert
(BDOS - Ruf 10)

INPUT - standard Eingabefile vom Typ Text

KBD - keybord: Eingabe erfolgt zeichenweise

ohne Echo auf den Bildschirm

LST - list device: nur Ausgabe auf Drucker

OUTPUT - standard Ausgabefile vom Typ Text

TRM - Terminal: wie bei CON, Zeichen erscheinen aber als Echo auf dem Bildschirm, zeichenweise Eingabe

USR - user device

Prozeduren

Name	Parameter	Wirkung
assign(f,str)	Var f:file_typ str:string	der Datei f wird der log Name str zugewiesen
bdos(i,j)	VAR i,j:integer	bdos - Funktion i mit Parameter j
bios(i,j)	VAR i,j:integer	bios - Funktion i mit Parameter j
blockread(f,v,n)	VAR f:file VAR v:beliebig n:integer	vom File f werden n Records von 128 Byte in v eingelesen
blockwrite(f,v,n)	VAR f:file VAR v:beliebig n:integer	auf das File f werden n Records von 128 Byte aus v geschrieben
chain(f)	VAR f:file_typ	Aufruf des Programmes, dessen Name mittels assign an die ungetypte filevariable f gebunden wurde
close(f)	VAR f:file_type	Inhalt des Puffers wird nach f geschrieben, Datei wird geschlossen.
clreol	-	loescht von aktueller Cursorposition bis Zeilenende
clscr		loescht Bildschirm und setzt den Cursor in die obere linke Ecke
crtexit	-	gibt Terminal - Reset-String aus
crtinit	-	gibt Terminal - Init-String aus

delay (t)	t:integer	Verzoegerung von ca t ms
delete(s,p,n)	VAR s:string p:integer n:integer	im string s werden ab p n Byte geloescht p+n > s problemlos p > s Laufzeitfehler
delline	-	loescht die Zeile in der sich der Cursor befindet
erase(f)	VAR f:file_type	Datei f wird geloescht. Muss vorher mit close geschlossen worden sein.
execute(f)	VAR f:file_type	wie CHAIN, jedoch werden beliebige COM-files aufgerufen.
dispose(p)	p:pointer	der Speicherplatz von p^ wird freigegeben
fillchar(x,n,w)	x:beliebig n:integer w:char	ab 1. Byte von x werden n- Byte mit w gefuellt
flush(f)	VAR f:file_typ	Puffer wird nach f geschrieben
freemem(v,i)	v:pointer i:integer	vom heap werden i Byte freigegeben, i muss den Wert von vorherigen getmem Aufruf haben.
getmem(v,i)	v:pointer i:integer	dem heap werden fuer v i Byte zugeteilt
gotoxy(i,k)	i,k:integer	Kursor geht in i-te Spalte und k-te Zeile
halt		Programmabbruch
insline	-	fuegt an Kursorposition eine Leerzeile ein
insert(q,z,p)	q:string VAR z:string p:integer	im string z wird an Pos. p der string q eingefuegt
lowvideo	-	normal (wenn install.)
mark(p)	p:pointer	mark & release haben die gleiche Wirkung wie dispose
move(v1,v2,n)	v1:beliebig	es werden n Byte von V1 nach V2 kopiert

new(p)	p:pointer	es wird Speicherplatz fuer die Variable p^ bereitgestellt und p bekommt die Anfangsadr. von p^
normvideo	-	intensiv (wenn install.)
ovrdrive(d)	VAR d:integer	steuert das Laufwerk,von dem ein Overlay gelesen wird. 1 = A, 2 = B usw. (erst ab Version 3)
randomize	-	initialisiert den Zufallszahlengenerator
read(f,d1,.. ,dn)	VAR f:file_typ VAR di	aus Datei f werden dx Datennsaetze gelesen
readln(f)	VAR f:text	der Filezeiger von f geht auf den Beginn einer neuen Zeile
release(p)	p:pointer	
rename(f,str)	VAR f:file_typ str:string	Datei f wird in str umbenannt, muss vorher mit close geschlossen worden sein
reset(f)	VAR f:file_typ	Filezeiger wird auf Satz 0 gestellt, Datei muss bereits existieren
rewrite(f)	VAR f:file_typ	Filezeiger wird auf Satz 0 gestellt, best. Datei wird geloescht
seek(f,n)	VAR f:file_typ n:integer	der Filezeiger wird auf den Datensatz n gestellt
str(v,s)	v:integer oder v:real VAR s:string	der in der Form v:d (int) bzw. v:d:s (real) angegebene Zahlenwert wird in einen String umgewandelt
val(s,v,c)	s:string VAR v:integer oder v:real c:integer	besteht der string s aus Ziffern, so wird der Variablen v der Wert zugewiesen, wenn kein Fehler auftritt, ist c = 0, ansonsten beinhaltet c die Position des ersten fehlerhaften Zeichens

write(f,d1,..,dn)	VAR f:file_typ dx	in die Datei f werden dx Datensaetze geschrieben
writeln(f)	VAR f:text	in die Datei f wird eine end-of-line Marke (CR/LF) eingetragen

Funktionen

addr(var)	var:any type (integer)	liefert absolute Speicheradressevon var
bdos(i,j)	i,j:integer (integer)	setzen bdos-funktion
bdoshl(i)	i : integer (integer)	bdos-Funktion mit Uebergabe in HL
bios(i,j)	i,j:integer (integer)	setzen bios-funktion
bioshl(i)	i : integer (integer)	setzen bios-Funktion mit Uebergabe in HL
chr(x)	x:integer (char)	ASCII-Zeichen
concat(s1,..sn)	si:string	die Strings werden verkettet; Gesamtlaenge <= 255
copy(s,p,n)	s:string p:integer n:integer (string)	aus s werden ab p n Zeichen entnommen; p+n > length(s) ok. p <> 1..255 ERROR
eof(f)	VAR f:file_typ (boolean)	**true** wenn Filezeiger auf Dateiende zeigt
eoln(f)	VAR f:text (boolean)	**true** wenn Filezeiger auf **CR** steht; **true** wenn eof(f)=true
filepos(f)	VAR f:file_typ (integer)	Nummer des Records auf das der Filezeiger zeigt; (1. Record hat Nummer 0)
filesize(f)	VAR f:file_typ (integer)	Anzahl der Datensaetze der Datei f
ioresult	(integer)	enthaelt nach einer E/A-Operation den Error - Code
length(s)	s:string	Anzahl Zeichen von s

	(integer)	
maxavail	- (integer)	Wert des groessten zusammenhaengenden Speicherplatzes des heap
memavail	- (integer)	verfuegbarer Speicherplatz des heap in Byte
pos(q,z)	q:string z:string (integer)	im string z wird die 1. Position des Teilstring q ermittelt.
seekeof(f)	VAR f:file_typ (boolean)	wahr,wenn EOF,sonst FALSE (erst ab Version 3)
seekeoln(f)	VAR f:file_typ (boolean)	(erst ab Version 3)
upcase(x)	x:char	Grossbuchstabe von x

arithmetische Operatoren

Stufe	Operator	Operandentyp	Ergebnistyp	Bedeutung
0	NOT	integer	integer	bitweise Negation
1	*	real	real	Multiplikation
	*	integer	integer	Multiplikation
	*	integer,real	real	Multiplikation
	/	integer	real	Division
	/	real	real	Division
	/	integer,real	real	Division
	DIV	integer	integer	Division ohne Rest
	MOD	integer	integer	Rest der Division
	AND	integer	integer	bitweise UND
	SHL	integer	integer	Linksshift
	SHR	integer	integer	Rechtsshift
2	+	integer	integer	Addition
	+	real	real	Addition
	+	real,integer	real	Addition
	-	integer	integer	Subtraktion
	-	real	real	Subtraktion
	-	real,integer	real	Subtraktion
	OR	integer	integer	bitweises ODER
	XOR	integer	integer	bitweises exklusiv ODER

arithmetische Funktionen

Funktion	Argument	Ergebnistyp	Bedeutung
abs(x)	integer	integer	Betrag von x
abs(x)	real	real	Betrag von x
arctan(x)	real,integer	real	Hauptwert von arctan im Bogenmass
cos(x)	real,integer	real	cos x im Bogenmass
exp(x)	real,integer	real	Exponentialfunktion
frac(x)	real,integer	real	gebrochener Teil von x
int(x)	real,integer	real	groesste ganze Zahl <= x wenn x > 0 kleinste ganze Zahl >= x wenn x < 0
ln(x)	real,integer	real	nat. Logarithmus (x > 0)
sin(x)	real,integer	real	sin x im Bogenmass
sqr(x)	integer	integer	Quadrat von x
sqr(x)	real	real	Quadrat von x
sqrt(x)	integer,real	real	Quadratwurzel von x mit x > 0
ord(x)	Ordinaltyp	integer	Nummer von x im Ordinaltyp
pred(x)	Ordinaltyp	integer	Vorgaenger von x
succ(x)	Ordinaltyp	integer	Nachfolger von x
trunc(x)	real	integer	groesste ganze Zahl <= x (x > 0) kleinste ganze Zahl >= x (x < 0)
round(x)	real	integer	round(x)=trunc(x+0.5) wenn x > 0 round(x)=trunc(x-0.5) wenn x < 0
hi(x)	integer	integer	linkes (high) Byte von x
lo(x)	integer	integer	rechtes (low) Byte von x
random	-	real	0 <= Zufallszahl < 1
random(x)	integer	integer	0 <= Zufallszahl < x
sizeof(x)	beliebig	integer	Anzahl Bytes von x

swap(x)	integer	integer	vertauschen high & low-Byte

logische Operatoren

Der Ergebnistyp aller logischen Operatoren ist boolean.

Stufe	Operator	Operand	Bedeutung
0	NOT	boolean	Negation
1	AND	boolean	Konjunktion
2	OR	boolean	Disjunktion
	XOR	boolean	Antivalenz
3	=	einf. Typ	gleich
	<>	einf. Typ	ungleich
	<	einf. Typ	kleiner
	<=	einf. Typ	kleiner oder gleich
	>	einf. Typ	groesser
	>=	einf. Typ	groesser oder gleich

logische Funktionen

Funktion	Argumenttyp	Ergebnistyp	Bedeutung
odd(x)	integer	boolean	true wenn x ungerade false wenn x gerade
keypressed	-	boolean	true wenn Taste gedrueckt wurde

Mengenoperatoren

Stufe	Operator	Operandentyp	Ergebnistyp	Bedeutung
1	*	menge	menge	Schnittmenge
2	+	menge	menge	Vereinigungsmenge
	-	menge	menge	Mengendifferenz
3	=	menge	boolean	Test auf Gleichheit
	<>	menge	boolean	Test auf Ungleichheit
	>=	menge	boolean	Test auf Teilmenge
	<=	menge	boolean	Test auf Teilmenge

```
      IN       links:expr.  boolean     Test auf Enthaltensein
               rechts:menge
-------------------------------------------------------------
```

BDOS - Funktionen

Die eingeklammerten Werte geben das/die zur Uebergabe verwendete Register an. Inwieweit die BDOS-Funktionen in der jeweiligen Variante von CP/M 2.2 realisiert sind (speziell Leser/Stanzer), muss der Spezialliteratur entnommen werden.
Als Fehlercode wird im Fehlerfall im Allgemeinen $FF (dez.255) uebergeben.

Nr.	Operation	uebergebener Wert	erhaltener Wert
0	Warmstart		
1	Konsoleingabe		Zeichen (A)
2	Konsolausgabe	Zeichen (E)	
3	Lesereingabe		Zeichen (A)
4	Stanzerausgabe	Zeichen (E)	
5	Druckerausgabe	Zeichen (E)	
6	direkte Konsol Ein/Ausgabe	FF -> Eingabe Zeichen -> Ausgabe	0 fuer n.bereit oder Zeichen (A)
7	lies IO-BYTE		Byte (A)
8	setze IO-BYTE	Byte (E)	
9	Ausgabe Puffer	Adresse (DE)	
10	lies Puffer	Adresse (DE)	
11	lies Konsolenstat		Byte (A) (00/FF)
12	lies Versionsn.		Byte (H),Byte (L)
13	Laufwerk Grund-stellung -> DISK RESET		
14	Laufwerkanwahl	Laufwerk-Nr. (E)	
15	Eroeffne Datei	FCB-Adresse (DE)	Fehlercode (A)
16	Schliesse Datei	FCB-Adresse (DE)	Fehlercode (A)
17	suche erste Datei	FCB-Adresse (DE)	Fehlercode (A)
18	suche naechste Datei	Fehlercode (A)	
19	loesche Datei	FCB-Adresse (DE)	Fehlercode (A)
20	lies sequentiell	FCB-Adresse (DE)	Fehlercode (A)
21	schreibe sequent.	FCB-Adresse (DE)	Fehlercode (A)
22	Erstelle Datei	FCB-Adresse (DE)	Fehlercode (A)
23	Datei umbenennen	FCD-Adresse (DE)	Fehlercode (A)
24	Bestimme moment. Laufwerke		Vektor (HL)
25	Finde Standardlw.		Laufwerk (A)
26	setze DMA-Adresse	DMA-Adresse (DE)	
27	lies Belegtabelle		Vektor (HL)
28	setze Schreibsch.		
29	finde R/O-Laufw.		Vektor (HL)
30	setze Dateiattr.	FCB-Adresse (DE)	
31	lies Diskpara-meterblock		Blockadresse (HL)
32	lies oder setze Benutzernummer	FF (E) -> lesen Nummer (E) ->setzen	Benutzer-Nummer (A)

33	lies direkt	FCB-Adresse (DE)	Fehlercode (A)
34	schreibe direkt	FCB-Adresse (DE)	Fehlercode (A)
35	lies Dateigroesse	FCB-Adresse (DE)	
36	set Direkt-Satznr	FCB-Adresse (DE)	
37	Laufwerk(e) zu ruecksetzen	Laufwerkvektor (DE)	0=ok. (A)

Die Funktionen 28 u.32 sollten nicht benutzt werden -> Kompatibilitaet zu MP/M

BIOS-Funktionen

Die in Klammern gesetzten Buchstaben geben das Register an.

Nr.	Operation/Funktion	uebergebener,	erhaltener Wert
0	Warmstart		
1	Konsolenstatus		FF/00 (A)
2	Konsoleingabe		Zeichen (A)
3	Konsolausgabe	Zeichen (C)	
4	Listenausgabe	Zeichen (C)	
5	Stanzerausgabe	Zeichen (C)	
6	Lesereingabe		Zeichen (A)
7	Spur 0 anwaehlen		
8	Laufwerk anwaehlen	Lwnr. (C)	DPH-Adresse (HL)
9	Spur-Nr.setzen	Spur-Nr (BC)	
10	Sektor-Nr.setzen	Sektor-Nr(BC)	
11	Pufferadr.setzen	PMA-Adr. (BC)	
12	Sektor lesen		Fehlercode (A)
13	Sektor schreiben		Fehlercode (A)
14	Status Listenausg.		0/1 (A)
15	Sektorkonvertierung logisch -> physisch	log.Sekt.(BC) Tab. in (DE)	phys.Sektornr. (HL)

ASCII - Zeichensatz

Dez.	Hex.	^Char	Char	Dez.	Hex	Char	Dez.	Hex.	Char	Dez.	Hex.	Char
0	00	^@	NUL	32	20	SP	64	40	@	96	60	'
1	01	^A	SOH	33	21	!	65	41	A	97	61	a
2	02	^B	STX	34	22	"	66	42	B	98	62	b
3	03	^C	ETX	35	23	#	67	43	C	99	63	c
4	04	^D	EOT	36	24	$	68	44	D	100	64	d
5	05	^E	ENQ	37	25	%	69	45	E	101	65	e
6	06	^F	ACK	38	26	&	70	46	F	102	66	f
7	07	^G	BEL	39	27	'	71	47	G	103	67	g
8	08	^H	BS	40	28	(	72	48	H	104	68	h
9	09	^I	HT	41	29	)	73	49	I	105	69	i
10	0A	^J	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	^K	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	^L	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	^M	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	^N	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	^O	SI	47	2F	/	79	4F	O	111	6F	o
16	10	^P	DLE	48	30	0	80	50	P	112	70	p
17	11	^Q	DC1	49	31	1	81	51	Q	113	71	q
18	12	^R	DC2	50	32	2	82	52	R	114	72	r
19	13	^S	DC3	51	33	3	83	53	S	115	73	s
20	14	^T	DC4	52	34	4	84	54	T	116	74	t
21	15	^U	NAK	53	35	5	85	55	U	117	75	u
22	16	^V	SYN	54	36	6	86	56	V	118	76	v
23	17	^W	ETB	55	37	7	87	57	W	119	77	w
24	18	^X	CAN	56	38	8	88	58	X	120	78	x
25	19	^Y	EM	57	39	9	89	59	Y	121	79	y
26	1A	^Z	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	^[	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	^\	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	^]	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	^^	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	^_	US	63	3F	?	95	5F	_	127	7F	DEL

Ersetzungen im deutschen ASCII - Zeichensatz

[= AE \ = OE] = UE { = ae | = oe } = ue
~ = sz @ = Par.

NUL	NIL	DC2	divice control 2
SOH	start of heading	DC3	divice control 3
STX	start of text	DC4	divice control 4
EOT	end of text	NAK	negative acknowledge
ENQ	enquiry	SYN	synchronization
ACK	acknowledge	ETB	end of block
BEL	bell	CAN	cancel
BS	backspace	EM	end of medium
HT	horizontal tabulator	SUB	substitution
LF	line feed	ESC	escape
VT	vertical tabulator	FS	form separator
FF	form feed	GS	group separator
CR	carriage return	RS	separator
SO	shift out	US	separator

SI   shift in
DC1 divice control 1

reservierte Worte

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

NOTATION von Pascal

Bezeichner, Zahlen und Zeichenketten

```
<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
             N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
             a | b | c | d | e | f | g | h | i | j | k | l | m |
             n | o | p | q | r | s | t | u | v | w | x | y | z |
             @

<digit>  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
             A | B | C | D | E | F  {nur in HEX-Zahlen zugelassen}

<spezial symbol> ::= {reserved words extra}
                     +  | -  | * | / | = | <> | < | > |
                     <= | >= | ( | ) | [ | ]  | { | } |
                     := | .  | , | ; | : | '  | ^ |
     {the following are additional or substitutions:}
                     (. | .) |   | ? | 1 | |  | $ | & | #

                     !     und | sind Synonyme
          _               wird innerhalb Namen ueberlesen
```

Ein Kommentar beginnt mit (* und muss mit *) enden. Er darf aber auch mit { beginnen und mit } enden.

```
<comment> ::= { <any characters>} | (*<any characters>*)
```

Die Form mit (*..*) darf innerhalb der Form {...} stehen.

```
<identifier>    ::= <letter>{<letter or digit or underscore>}
     <letter or digit or underscore> ::= <letter> | <digit> | _

<digit sequence>   ::= <digit> {<digit>}

<unsigned integer> ::= $<digit sequence>| #<digit sequence> |
                         <digit sequence>
<unsigned real>    ::= <unsigned integer>. <digit sequence> |
                       <unsigned integer> . <digit sequence>
                       E <scale factor>                     |
                       <unsigned integer> E <scale factor>
<unsigned number>  ::= <unsigned integer> | <unsigned real>
<scale factor>     ::= <unsigned interger> |
                       <sign><unsigned integer>
<sign>             ::= + | -

<string>           ::= ' <character> {<character>] ' | ''
```

Unterstreichung ist zulaessig zwischen Buchstaben und Ziffern in einem Bezeichner und wird ueberlesen. Klein- und Grosschreibung wird ignoriert, intern werden Grossbuchstaben verwendet, ausser in Zeichenketten. Ein "$" vor einer ganzen Zahl interpretiert diese Zahl als Hexadezimal. Dann sind die Buchstaben A..F als

Ziffern gueltig.

```
<constant identifier> ::= <identifier>
<constant>            ::= <unsigned number>           |
                          <sign><unsigned number>     |
                          <constant identifier>       |
                          <sign><constant identifier> | <string>
<constant definition> ::= <identifier> = <constant>
```

Zusaetzlich ist die Nullkette erlaubt mit nullstr = '';

Typenvereinbarungen

```
<type>                ::= <simple type>     |
                          <structured type> | <pointer type>
<type definition>     ::= <identifier> = <type>

<simple type>         ::= <scalar type>   |
                          <subrange type> | <type identifier>
<type identifier>     ::= <identifier>

<scalar type>         ::= (<identifier> {,<identifier>})
```

Folgende Typen sind Standard in TURBO-PASCAL

```
     INTEGER | REAL | BOOLEAN | CHAR | BYTE | STRING

<subrange type>       ::= <constant>..<constant>

<structured type>     ::= <array type>   |
                          <record type>  |
                          <set type>     |
                          <file type>

<array type>      ::= <normal array> | <string array>
<string array>    ::= STRING <max lenght>
<max lenght>      ::= [<intconst>]
<intconst>        ::= <unsigned integer> | <intconst id>
<intconst id>     ::= <identifier>
<normal array>    ::= ARRAY [<index type>{,<index type>}] OF
                      <component type>
<index type>      ::= <simple type>
<component type>  ::= <type>


<record type>     ::= RECORD <field list> END
<field list>      ::= <fixed part>                  |
                      <fixed part> : <variant part> |
                      <variant part>
<fixed part>      ::= <record section> {;<record section>}
<record section>  ::= <field identifier>{,<field  identifier>}
                      : <type> | <empty>
<variant part>    ::= CASE <tag field> <type identifier> OF
                          <variant> {;<variant>}
<variant>         ::= <case label list> : (<field list>) |
                          <empty>
```

```
<case label list> ::= <case label> {,<case label>}
<case label>      ::= <constant>
<tag field>       ::= <identifier> : | <empty>

<set type>        ::= SET OF <base type> <base type> ::=
                      <simple type>
<file type>       ::= FILE {OF <type>}
```

Es sind getypte und ungetypte Files erlaubt. Ungetypte Files sind fuer CHAIN und ebenso fuer BLOCKREAD/BLOCKWRITE und SEEK verwendbar.

```
<pointer type>      ::= ^ <type identifier>
```

Zeiger sind als INTEGER-Typ aufzufassen und duerfen in arithmetischen Ausdruecken vorkommen.
Operationen mit verschiedenen Typen sind verboten. Vor dieser Anwendung ist Typgleichheit durch Konvertierung herzustellen.

Variablenvereinbarung und -Beschreibung

```
<variable declaration>::= <variable> : <type> | <empty>
<variable>            ::= <var> | <external var> | <absolute var>
<external var>        ::= <var> EXTERNAL <addr>
<absolute var>        ::= <var> ABSOLUTE <addr>
<var>                 ::= <entire variable>     |
                          <component  variable> |
                          <referenced variable>
<addr>                ::=absolute memory address
<entire variable>     ::= <variable identifier>
<variable identifier> ::= <identifier>
<component variable>  ::=<indexed variable>|<field designator>|
                         <file buffer>
<indexed variable>    ::= <array variable> [<expression>
                {,<expression>}] <array variable> ::= <variable>
<field designator>    ::= <record variable> . <field identifier>
<file buffer>         ::= <file variable> ^ <file variable>
                      ::= <variable>
<referenced variable> ::= <pointer variable>^<pointer variable>
                      ::= <variable>
```

Ausdruecke

```
<unsigned constant> ::= <unsigned number> | <string> | NIL |
                         <constant identifier>
<factor>            ::= <variable> | <unsigned constant> |
                         <function designator>          |
                        (<expression>) | <logical not operator>
                         <factor> <set>
<set>               ::= [ <element list> ]
<element list>      ::= <element> {,<element>} | <empty>
<element>           ::= <expression>|<expression>..<expression>
<term>              ::= <factor><multiplying operator><factor>
<simple expression> ::= <term> |
                    <simple expression><adding operator<>term>
```

```
<expression>        ::= <simple expression> |
    <simple expression><relational operator><simple expression>

<logical not operator> ::= NOT | ?     {? fuer 16 Bit}
<multiplying operator> ::= * | / | DIV | MOD | AND | &
                                       {& fuer 16 Bit}
<adding operator>      ::= + | - | OR | | | !
                                       {! und | fuer 16 Bit}

<relational operator>  ::= = | <> | < | <= | > | >= | IN

<function designator>  ::= <function identifier> |
                           <function identifier> (<parm>{,<parm>})
<function identifier>  ::= <identifier>
```

Anweisungen

```
<statement>            ::= <label> : <unlabelled statement> |
                           <unlabelled statement>
<unlabelled statement> ::= <simple statement> |
                           <structured statement>
<label>                ::= <unsigned integer>
<simple statement>     ::= <assignmenr statement> |
                           <procedure statement>  |
                           <goto statement>       |
                           <empty statement>
<empty statement>      ::= <empty>
<assign statement>     ::= <variable> := <expresion> |
                           <function identifier> := <expression>

<procedure satement>   ::= <procedure identifier>
                           ( <parm>{,<parm>} )       |
                           <procedure identifier>

<procedure identifier> ::= <identifier>
<parm>                 ::= <expression> | <variable>

<goto statement>       ::= GOTO <label>

<structured statement> ::= <repetitive statement>  |
                           <conditional statement> |
                           <compound statement>    |
                           <with statement>

<compound statement>   ::= BEGIN <statement> {,<statement>} END

<conditional statement>::= <case statement> | <if statement>
<if statement>         ::= IF <expression> THEN <statement>
                                  ELSE <statement> |
                           IF <expression> THEN <statement>

<case statement>       ::= CASE <expression> OF <case list>
                            {;<case list>} {ELSE <statement>}
                           END
<case list>            ::= <label list> : <statement> | <empty>
<label list>           ::= <case label> {,<case label>}
<case label>           ::= <non-real short scalar constant>
```

```
<repetitive astatement>::= <repeat statement> |
                            <while statement> | <for statement>
<while statement>      ::= WHILE <expression> DO <statement>
<repeat statement>     ::= REPEAT <statement> {;<statement>}
                              UNTIL <expression>
<for statement>        ::= FOR <ctrlvar> := <for list> DO
                              <statement>

<for list>             ::= <expression> DOWNTO <expression> |
                           <expression> TO <expression>

<ctrlvar>              ::= <variable>

<with statement>       ::= WITH <record variable list> DO
                              <statement>

<record variable list> ::=<record variable> {,<record variable>}
```

Prozedurvereinbarungen

```
<procedure declaration>::= EXTERNAL <procedure heading> |
                           <procedure heading> <block>
<block>                ::= <label declaration part>
                           <constant declaration part>
                           <type declaration part>
                           <variable declaration part>
                           <procfunc declaration part>
                           <statement part>
<procedure heading>    ::= PROCEDURE <identifier> <parmlist>; |
                           PROCEDURE <identifier>;
<parmlist>             ::= ( <fparm> {,<fparm>}
<fparm>                ::= <procedure heading> |
                           <function heading>  |
                           VAR <parm group>    |
                           <parm group>
<parm group>           ::= <identifier {,<identifier>} :
          <type identifier> | <identifier> {,<identifier>} :
          <conformant array>

<conformant array>     ::= VAR ARRAY [<indxtyp> {;<indxtyp>} ]
                              OF <conarray2>

<conarray2>            ::=<type identifier> | <conformant array>

<indxtyp>              ::= <identifier>..<identifier> : <ordtyp>
<ordtyp>               ::= <scalar type identifier> |
                           <subrange type identifier>
<scalar type identifier>  ::= <identifier>
<subange type identifier> ::= <identifier>
<label declaration part>  ::= <empty> |
                              LABEL <label> {,<label>};

<constant definition part>::= <empty> |
           CONST <constant definition> {;<constant definition>};

<type definition part>    ::= <empty> |
```

```
                    TYPE <type definition> {;<type definition>};

<variable declaration part>::= <empty> |
           VAR <variable declaration> {;<variable declaration>};

<procfunc declaration part>::= {<proc or func>;}

<proc or func>             ::= <procedure declaration> |
                               <function declaration>

<statement part>           ::= <compound statement>
```

Filebehandlung:

IORESULT	WRITE	RESET	REWRITE
ASSIGN	CLOSE	ERASE	SEEK
CHAIN	READ	BLOCKREAD	BLOCKWRITE
READLN	WRITELN	FLUSH	EXECUTE

Die dynamischen Prozeduren NEW und DISPOSE entsprechen NEW, MARK und RELEASE

Funktionsvereinbarungen

```
<function decl>     ::= <function heading> |
                        <function heading> <block>
<function heading>  ::= FUNCTION <identifier><parmlist> :
                                    <result type>;|
                        FUNCTION <identifier> : <result type>;

<result type>       ::= <type identifier>
```

Ein- und Ausgabe

```
<readcall>     ::= <read or readln> { ({<filevar>,}{<varliste})}
<read or readln> ::= READ | READLN
<filevar>        ::= <variable>
<varlist>        ::= <variable> {,<variable>}
<writecall>    ::= <write or writeln>{({<filevar>,}{<exprlist>})}
<write or Writeln> ::= WRITE | WRITELN

<exprlist>     ::= <wexpr> {,<wexpr>}
<wexpr>        ::= <expression> {:<width expr>{:<dec expr>}}
<width expr>   ::= <expression>
<dec expr>     ::= <expression>
```

PROGRAMME

```
<program>      ::= <program heading>
                   <label declaration part>
                   <constant declaration part>
                   <type definition part>
```

```
                    <variable declaration part>
                    <procfunc declaration part>
                     END.

<program heading> ::= PROGRAM <identifier> {(<prog parms>)};

<prog parms>      ::= <identifier> {,<identifier>}
```

Dateiarbeit

Der Komplex vordefinierte Textdateien wurde in Teil 2 des Handbuches behandelt und wird hier nicht noch einmal gesondert betrachtet.
Zur Dateiarbeit werden in TURBO folgende Prozeduren und Funktionen bereitgestellt:

Prozeduren
==========

ASSIG REWRITE RESET READ WRITE SEEK FLUSH CLOSE ERASE RENAME

BLOCKREAD/BLOCKWRITE -> nur fuer ungetypte Dateien
WRITELN/READLN -> nur fuer Textdateien

Funktionen
==========

EOF FILEPOS FILESIZE

Beschreibung:

ASSIGN(filevar,str) filevar:FILE;str:STRING

ASSIGN ordnet dem logischen File 'filevar' die sich auf einem Datentraeger befindliche Datei mit dem Namen 'str' zu. Dabei ist es gleichgueltig, ob sich eine Datei mit dem Namen 'str' auf dem Datentraeger befindet oder nicht, eine Pruefung findet nicht statt.

REWRITE(filevar) filevar:FILE

Die Datei 'filevar' wird fuer Schreiboperationen eroeffnet. Es wird auf Datentraeger eine leere Datei mit dem bei ASSIGN zugewiesenen Namen angelegt, der Satzzeiger wird auf den ersten Satz gestellt. Sollte sich auf dem Datentraeger bereits eine Datei gleichen Namens befinden, wird sie ueberschrieben.

RESET(filevar) filevar:FILE

Eine sich auf einem Datentraeger befindende Datei mit dem bei ASSIGN zugewiesenen Namen wird fuer eine weitere Verarbeitung eroeffnet. Der Satzzeiger wird auf Dateianfang gestellt.
Sollte sich auf dem Datentraeger keine Datei mit dem zugewiesenen Namen befinden, tritt ein I/O-Fehler auf.

READ(filevar,var1,...,varN) filevar:FILE;varX:jeder Typ

Lesen der Variablen var1...varN von der Datei filevar. Nach jeder Leseoperation wird der Satzzeiger auf die naechste Komponente gestellt. Die Variabenliste in READ ist durch Komma zu trennen.

WRITE(filevar,var1,...varN) filevar:FILE;varX:jeder Typ

Schreiben der Variablen var1...varN auf die Datei filevar. Nach jeder Schreiboperation wird der Satzzeiger auf die naechste Komponente bzw. das naechste freie Feld gestellt.

SEEK(filevar,n) filevar:FILE;n:INTEGER

Der Satzzeiger wird auf die n-te Komponente der Datei filevar positioniert. Die Position der ersten Komponete ist 0. Es ist moeglich, auf eine Komponente nach dem Fileende zu positionieren (z.B. fuer Erweiterungen von Dateien).

FLUSH(filevar) filevar:FILE;

FLUSH schreibt den internen Schreib-Lesepuffer in die Datei filevar, d.h. wenn eine Screiboperation nach dem letzten Diskupdate stattfindet,wird der Puffer in die Datei geschrieben. FLUSH ist normalerweise nur in multi-user-Systemen noetig, wenn mehrere Nutzer gleichzeitig auf einen Datenbestand zugreifen. FLUSH sollte nicht auf eine geschlossene Datei angewandt werden.

CLOSE(filevar) filevar:FILE;

Die Datei filevar wird geschlossen, das Directory der Disk wird aktualisiert. In multi-user-Systemen ist es guenstig, eine Datei explizit zu schliessen, auch wenn nur lesend zugegriffen wurde.

ERASE(filevar) filevar:FILE;

ERASE loescht die Datei. Vor dem Loeschen sollte die Datei mit CLOSE geschlossen werden.

RENAME(filevar,str) filevar:FILE;str:STRING;

Das mit filevar verbundene Diskfile wird umbenannt in den durch str gegebenen Namen, das Diskettenverzeichnis wird aktualisiert. Alle Operationen mit filevar beziehen sich nach RENAME auf den neuen Namen. RENAME darf nicht auf ein eroeffnetes File angewandt werden.

EOF(filevar) :BOOLEAN filevar:FILE;

EOF liefert TRUE, wenn das Dateiende bei Leseoperationen erreicht wird, ansonsten FALSE.

FILEPOS(filevar) :INTEGER filevar:FILE;

FILEPOS liefert die aktuelle Position des Satzzeigers der Datei. Der erste Satz hat die Position 0.

FILESIZE(filevar) :INTEGER filevar:FILE;

FILESIZE liefert die Satzanzahl der Datei filevar.
Fuer eine Dateierweiterung (d.h. Positionieren auf den ersten Satz nach EOF ist folgende Konstruktion gut brauchbar:

```
SEEK(filevar,FILESIZE(filevar));
```

Ungetypte Dateien

Ungetypte Dateien stellen eine Struktur von 128 Byte dar. Sie sind mit anderen Dateien kompatibel und ein Zugriff auf sie ist sehr schnell und speichereffizient. Sie werden vorzugsweise fuer **erase** und **rename** u.a. E/A - Operationen benutzt.
Die Prozeduren **read, write** und **flush** sind nicht anwendbar.
Der Zugriff auf ungetypte Dateien erfolgt mit den Prozeduren BLOCKREAD und BLOCKWRITE. Fuer ungetypte Dateien gelten ebenfalls ASSIGN, RESET und REWRITE, d.h. es muss ein Name zugewiesen und die Datei muss eroeffnet werden.
Auch eine ungetypte Datei sollte mit CLOSE geschlossen werden.

BLOCKREAD(filevar,var,recs)
filevar:FILE;var:jeder Typ;recs:INTEGER

Leseoperation.recs gibt die Anzahl der 128-BYTE-Sektoren an, die zwischen Disk-File und Speicher bei einem Zugriff transferiert werden sollen. Der Transfer beginnt beim ersten BYTE der Variablen var. Durch den Programmierer ist abzusichern, dass var ausreichend Speicherplatz fuer den Transfer hat, eine Pruefung durch TURBO erfolgt nicht.

BLOCKWRITE(filevar,var,recs)
filevar:FILE;var:jeder Typ;recs:INTEGER

Schreiboperation. Sonst wie BLOCKREAD.

FILEPOS,FILESIZE und **SEEK**

benutzen Komponenten von 128 Byte.

Arbeit mit Dateien/Dateitypen

In TURBO sind sequentielle,Direktzugriffs- und indexsequentielle Dateien moeglich. Die Positionierung in der Datei erfolgt entweder sequentiell oder mit SEEK auf eine frei waehlbare Satznummer. Von dieser Satznummer aus kann sequentiell weitergearbeitet werden oder es kann erneut positioniert werden.
Fuer Textdateien ist **nur** eine **sequentielle Arbeit** moeglich.

Overlay-Techniken

In TURBO sind mehrere Moeglichkeiten der Programmueberlagerung gegeben. Diese sind:

* echte Overlays mit automatischer Rueckkehr ins rufende Programm und der Moeglichkeit der Parameteruebergabe.
* Aufrufe einen Programmes aus dem gerade laufenden Programm aus (CHAIN/ EXECUTE). Hierbei erfolgt kein autmatischer Ruecksprung ins rufende Programm, eine Parameteruebergabe ist beim Aufruf nicht moeglich.
* Verwendung externer Prozeduren/Funktionen

Generell sind Programmueberlagerungen im Directmode nicht zulaessig und erzeugen beim Uebersetzen eine Syntaxfehler, falls im Directmode gearbeitet wird.

Echte Overlays

Ab TURBO-Version 2.00x sind echte Overlays moeglich. Sie werden durch das reservierte Wort OVERLAY eingeleitet und muessen zur Uebersetzungszeit im Hauptprogramm vorhanden sein bzw. mit INCLUDE eingefuegt werden. Overlays erhalten den Hauptprogrammnamen und als Namenserweiterung eine fortlaufende dreistellige Nummer. Bedingt durch den fehlenden Linker ist eine getrennte Uebersetzung von Wurzelsegment und Overlay nicht moeglich.

Beispiel fuer eine einfache Overlaystruktur:

```
PROGRAM OVTEST;
{                                                          }
{              Testprogramm fuer Overlays                  }
{                                                          }
VAR A,B,C:INTEGER;
{                     Beginn Overlay                       }
OVERLAY PROCEDURE SUMME(A,B:INTEGER;VAR C:INTEGER);
BEGIN
  WRITELN('OVERLAY SUMME');
  C:=A+B;
  DELAY(2000);
END;
{             Beginn Hauptprogramm                         }
BEGIN
  CLRSCR;
  WRITELN('OVERLAYTEST':50);
  A:=1;
  B:=9;
  C:=20;
  WRITELN('C VOR OVERLAY-AUFRUF=',C:2);
  SUMME(A,B,C);    { Aufruf des Overlays                   }
  WRITELN('C NACH OVERLAY-AUFRUF=',C:2);
END.
```

TURBO erzeugt hier ein File OVTEST.COM sovie ein File OVTEST.000.

Die Steuerung der Laufwerke fuer die Overlays kann mit dem O-Schalter des Compilers erfolgen.

Ab TURBO-3.xx ist eine Zuweisung des fuer eine Overlaygruppe aktuellen Laufwerkes waehrend der Abarbeitungszeit moeglich. Dazu dient die Prozedur **ovrdrive** (drive_nr).

Overlays gleicher Hierarchiestufe werden von TURBO in einer gemeinsamen Overlaydatei zusammengefasst, die Steuerung der Zugriffe auf diese Datei uebernimmt TURBO. Die Zugehoerigkeit zur gleichen Gruppe wird durch die Anordnung der Overlays im Programm festgelegt. Overlays, die direkt hintereinander folgen, werden in einer Gruppe zusammengefasst. Die Trennung zwischen mehreren Gruppen erfolgt durch "normale" Prozeduren.

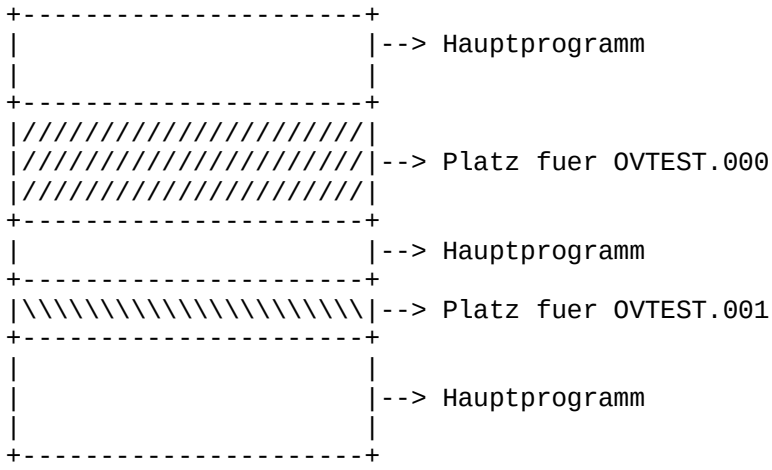
```
PROGRAM OVTEST;
VAR I,J,K,L:INTEGER;
OVERLAY PROCEDURE EINS;   ---------
BEGIN                               |
  WRITELN('OVERLAY EINS');          |
END;                                |---> BILDEN OVTEST.000
OVERLAY PROCEDURE ZWEI;             |
BEGIN                               |
  WRITELN('OVERLAY ZWEI');          |
END;                        ---------
PROCEDURE DREI;      ------------> TRENNUNG ZWISCHEN 000 UND 001
BEGIN
  WRITELN('PROZEDUR DREI');
END;
OVERLAY PROCEDURE VIER;   ---------
BEGIN                               |
  WRITELN('OVERLAY VIER');          |---> BILDET OVTEST.001
END;                        ---------
BEGIN
  WRITELN('ANFANG');
  EINS;
  ZWEI;
  DREI;
  READ(I);{ Festlegung des Laufwerkes f.d.folgenden Overlays }
  OVRDRIVE(I);
  VIER;
  DELAY(9000);
END.
```

Es gilt folgende Einschraenkung:

Prozeduren der gleichen Overlaygruppe koennen sich nicht gegenseitig aufrufen.

Nach dem Wechsel des Laufwerkes werden alle Overlays vom angewaehlten Laufwerk geholt.

Die Speicherbelegung bei der Arbeit mit Overlays ist wie folgt:



Der Platz fuer die Overlaygruppen ist so gross, dass der groesste Modul der Overlaygruppe Platz hat. Man sollte deshalb darauf achten, dass nach Moeglichkeit die Overlays einer Gruppe in ihrer Groesse nicht zu stark differieren. Des weiteren ist aus der Speicheraufteilung ersichtlich, dass ein nur aus einem Modul bestehendes Overlay keine Speicherplatzeinsparung bringt.
Wert- und Parameteruebergabe bei Overlays erfolgen nach den fuer Prozeduren/Funktionen gueltigen Regeln.

Programmaufrufe aus dem laufenden Programm heraus

Programmaufrufe aus laufenden Programmen heraus erfolgen mit CHAIN und EXECUTE. Der Unterschied zwischen diesen beiden Arten ist, dass mit CHAIN nur Programme geladen werden koennen, die mit der Compiler-Option H erzeugt wurden. Solche Files enthalten nur den Programmcode kein Laufzeitsystem <sie nutzen das Laufzeitsystem des rufenden Programmes); mit EXECUTE koennen beliebige COM-files geladen werden. Rufendes und gerufenes Programm ueberschreiben sich gegenseitig!
Falls trozdem eine Wertuebergabe erfolgen soll, ist neben der bereits in Teil 2 des Handbuches genannten noch folgende Moeglichkeit gegeben:

Die Variablen, die Uebergabewerte enthalten sollen, sind im rufenden und im gerufenen Programm als absolute Variable zu definieren und auf identische Speicherplaetze zu legen.

Beispiel:

```
VAR ZAHL :INTEGER ABSOLUTE $9000;
```

Hier wird die Variable 'ZAHL' fest auf $9000 gelegt. Wenn im gerufenen Programm eine INTEGER-Variable auf der gleichen Adresse definiert wurde, kann damit eine Wertuebermittlung rufend -> gerufen erfolgen.

Mit absoluten Variablen ist auch der Aufbau ueberlagerter Variabler moeglich.

```
Z.B. VAR STR    : STRING[36];
         STRLEN : INTEGER ABSOLUTE-STR;
```

Hier wird die Variable STRLEN auf die ersten beiden Byte von STR definiert, beide Variable benutzen also den gleichen Speicherplatz.

Achtung:Eine Definition wie VAR I,J,K:INTEGER ABSOLUTE $8000; ist unzulaessig, es kann in einer Anweisung nur jeweils eine absolute Variable definiert werden. Falls absolute Variable zur Wertuebergabe verwendet werden sollen, ist es guenstig,sie zwischen TURBO-Library und Programmcode anzulegen (s.Teil1, Manipulation der Startadresse).

Absolute Variable ermoeglichen auch den Zugriff auf Bestandteile des Betriebssystems, sofern diese Bestandteile genau lokalisierbar sind (z.B. die Kommandozeile)

Beispiel:

```
VAR CMDLINE : STRING[127] ABSOLUTE $80;
```

Hier ist die CP/M-Kommandozeile als Variable definiert. Damit besteht ein direkter Zugriff auf die Kommandozeile.

```
VAR IOBYTE  : BYTE ABSOLUTE $03;
```

Direkter Zugriff auf IO-BYTE moeglich.

Externe Prozeduren und Funktionen

Externe Prozeduren/Funktionen (im allgemeinen Routinen in Maschinensprache) bestehen im PASCAL-Programm nur aus ihrem Namen, der Parameterliste, dem Schluesselwort EXTERNAL, gefolgt von der absoluten Adresse der Prozedur/Funktion im Hauptspeicher.

Beispiel:

```
PROCEDURE PLOT1(X,Y,Z:INTEGER); EXTERNAL $C000;
```

Weitere Angaben sind im PASCAL-Programm nicht noetig.

Die Parameteruebergabe erfolgt ueber Z80-Stack.Parameter muessen von der externen Routine mit PUSH/POP vom Stack geholt und in den Stack transportiert werden.
Die Parameteruebergabe ist abhaengig von Art und Typ der Variablen.
Bei VAR-Parametern wird im Stack die absolute Adresse des ersten BYTES der Variablen uebergeben.
Bei Wertparametern gilt:
Skalare (INTEGER, CHAR, BOOLEAN, BYTE) werden im Stack als Wort transportiert, wenn die Variable nur ein Byte belegt, ist das hoeherwertige BYTE Null. Im allgemeinen wird ein Wort mit POP HL transportiert.

REAL-Groessen belegen 6 BYTE im Stack. Die Uebergabe ist wie folgt:

```
  POP  HL
  POP  DE
  POP  BC
```

L enthaelt den Exponenten,H das fuenfte BYTE,usw, B enthaelt das erste (hoeherwertigste) BYTE.

STRINGS-Variable: Wenn ein String an der Spitze des Stacks ist, enthaelt das BYTE, auf das SP zeigt, die Laenge des Strings. Die BYTES der Adressen SP+1..SP+N (N=Stringlaenge) enthalten die Zeichen des Strings. Die folgende Assemblerroutine soll als Beispiel fuer eine Stringuebergabe dienen:(holen vom Stack und Spoeichern in STRBUF)

```
 LD  DE,STRBUF
 LD  HL,0
 LD  B,H
 ADD HL,SP
 LD  C,(HL)
 INC BC
 LDIR
 LD  SP,HL
```

SET-Variable: Ein Set belegt 32 BYTE des Stacks. Die folgende Assemblerroutine demonstriert die Uebergabe vom Stack in die Variable SETBUF.

```
 LD  DE,SETBUF
 LD  HL,0
 ADD HL,SP
 LD  BC,32
 LDIR
 LD  SP,HL
```

POINTER werden im Stack als Wort uebergeben und enthalten die Adresse der dynamischen Variablen.

ARRAYS/RECORDS. Es eird in einem Wort die Adresse des ersten BYTES uebergeben. Diese Adresse kann dann in einer Blockkopieroperation verwendet werden.

Die Uebergabe der Ergebnisse erfolgt in der gleichen Weise. Werte von Skalaren und Pointern muessen im HL-Registerpaar uebergeben wewrden. Wenn das Resultat nur ein BYTE enthaelt, ist in L zu uebergeben, H muss 0 sein.

INLINE-Assembler

In TURBO ist ein INLINE-Assembler enthalten. Er erlaubt die Verwendung von Maschinenprogrammen an beliebigen Stellen eines TURBO-Programms. Eine INLINE-Anweisung besteht aus dem reservierten Wort **INLINE**, gefolgt von einem oder beliebig vielen Assemblerbefehlen. Der gesamte Teil der Assemblerbefehle ist in runde Klammern einzufassen, die Anweisungen, Variablen usw sind durch Schraegstriche "/" zu trennen.

Verwendet wird der numerische Operationscode fuer Z80.

Konstanten koennen Literale oder Konstantenbezeichner sein. Sie muessen vom Typ INTEGER sein. Literale benoetigen 1 BYTE des Codes, wenn sie im Bereich von 0 bis 255 liegen, ansonsten 2 BYTE (byte reversed format). Konstantenbezeichner belegen immer 2 BYTE. Variablenbezeichner belegen 2 BYTE und enthalten die Speicheradresse der Variablen (byte reversed format).

Ein Bezug auf den Befehlszaehler besteht aus einem Stern "*", gefolgt von einem Offset, der aus einem Plus- oder Minuszeichen und einer INTEGER-Konstante besteht. Ein Stern allein benoetigt zwei BYTE, die den aktuellen Stand des Befehlszaehlers enthalten (byte reversed format). Ein Stern gefolgt von einem Offset enthaelt den Befehlszaehlerstand, veraendert um den Wert des Offsets.

Das folgende Beispiel einer INLINE-Prozedur wandelt alle Kleinbuchstaben in einem String in Grossbuchstaben: Die Sprungbefehle des Beispiels dienen nur zur Demonstration der Technik der Spruenge.

```
PROCEDURE UpCase(VAR STRG:STING[255]);
BEGIN
 INLINE ($2A/STRG/                  {   LD  HL,(STRG)          }
         $46/                       {   LD  B,(HL)             }
         $04/                       {   INC B                  }
         $05/                       {L1:DEC B                  }
         $CA/*+20/                  {   JP  Z,L2               }
         $23/                       {   INC HL                 }
         $7E/                       {   LD  A,(HL)             }
         $FE/$61/                   {   CP  'a'                }
         $DA/*-9/                   {   JP  C,L1               }
         $FE/$7B/                   {   CP  'z'+1              }
         $D2/*-14/                  {   JP  NC,L1              }
         $D6/$20/                   {   SUB 20H                }
         $77/                       {   LD  (HL),A             }
         $C3/*-20)                  {   JP  L1                 }
                                    {L2:EQU $                  }
END;
```

INLINE-Befehle koennen alle CPU-Register benutzen.

ACHTUNG:
Der Inhalt des Stackpointerregisters SP muss nach der Beendigung

der INLINE-Befehle den gleichen Stand wie vor Beginn der INLINE-Befehle haben.

Im weiteren folgen vier Beispiele fuer nuetzliche INLINE-Routinen.

(aus c't 6/85)

```
{ hexadezimale Ausgabe einer integer-Groesse }

procedure write_i_hex(i:integer);
begin
    inline ($cd/$48d);
end;



{ hexadezimale Ausgabe eines bytes }

procedure write_b_hex(b:byte);
begin
    inline($cd/$492);
end;


{ hexadezimale Ausgabe einer integer-Groesse }

function write_i_hex(i:integer):char;
begin
    inline ($cd/$48d);
    write_i_hex:=chr(0);
end;


{ hexadezimale Ausgabe eines Bytes }

function write_b_hex(b:byte) : char;
begin
    inline($cd/$492);
    write_b_hex:=chr(0);
end;
```

Achtung: Bei diesen Prozeduren/Funktionen wird die TURBO-Library "angezapft". Die Ausgabe erfolgt nur ueber CON. Fuer andere Einheiten muss der CONOUTPTR umgestellt werden.

Cursorsteuerung

Funktion	Kommando
Cursor links	^H oder ^S
Cursor rechts	^D
Cursor hoch	^E
Cursor nach unten	^X
Wort links	^A
Wort rechts	^F
Schirm nach unten rollen(eine Zeile)	^W
Schirm nach oben rollen(eine Zeile)	^Z
Schirm nach unten rollen(eine Seite)	^R
Schirm nach oben rollen(eine Seite)	^C
Cursor an linken Zeilenrand	^QS
Cursor an rechten Zeilenrand	^QD
Cursor an Seitenanfang	^QE
Cursor an Seitenende	^QX
Cursor an Dateianfang	^QR
Cursor an Dateiende	^QC
Cursor an Blockanfang	^QB
Cursor an Blockende	^QK
zurueck an letzte Cursorposition	^QP

Einfuegen/Loeschen

Funktion	Kommando
Inset-Mode an/aus	^V
Zeile einfuegen	^N
Zeile loeschen	^Y
loeschen bis Zeilenende	^QY
Wort rechts vom Cursor loeschen	^T
Zeichen auf Cursorposition loeschen	^G
Zeichen links vom Cursor loeschen	DEL,DELCH (Tast.abh.)

Blockkommandos

Achtung: Bei den Blockkommandos wird Blockanfang/-ende nicht wie bei WordStar mit <B>/<K> markiert. Die Art der Markierung ist installationsabhaengig (meist wird ueber die Bildhelligkeit markiert, bei der vorliegenden Version erfolgt keine sichtbaere Markierung!).

Funktion	Komando
Blockbeginn markieren	^KB
Blockende "	^KK
Wort markieren	^KT
Block sichtbar/unsichtbar	^KH
Block kopieren	^KC
Block verschieben	^KV
Block loeschen	^KY

Datei/Block von Disk lesen	^KR
Datei/Block auf Disk schreiben	^KW

zusaetzliche Editorfunktionen

Funktion	Kommando
Ende des Editierens	^KD
TAB	^I
automatischer Tabulator an/aus	^QI
Zeile zurueckspeichern	^QL
Finden	^QF
Finden und ersetzen	^QA
Steuerzeichenvorsatz	^P

Fehlermeldungen

1. E/A-Fehlermeldungen

Wenn der Compiler-Schalter **I** aktiv ist, kommt es bei einer fehlerhaften E/A - Operation zu einer Fehlermeldung des Systems in der Form:

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

Dabei ist NN die hexadezimale Nummer des Fehlers und adr die Adresse im Objektcode.

Wenn der Compiler-Schalter **I** nicht aktiv ist, so werden die weiteren E/A - Operationen bis zur Auswertung des Fehlers mit der Funktion **ioresult** nicht ausgefuehrt. Sollte vor Auswertung mittels **ioresult** nach einer fehlerhaften E/A - Operation eine weitere fehlerhafte E/A-Operation erfolgen, kann das Programm abstuerzen!

Werte von ioresult/NN

00	korrekte E/A - Operation
01	file does not exist
02	file not open for input
03	file not open for output
04	file not open
10	error in numeric format
20	operation not allowed on a logical device
21	not allowed in direct mode
22	assign to std files not allowed
90	record length mismatch
91	seek beyond end-of-file
99	unexpected end-of-file

F0 disk write error
F1 directory full
F2 file size overflow
FF file disappeared

Die Funktion **ioresult** wird bei folgenden Prozeduren gesetzt:

close	read	rewrite	chain
blockread	erase	readln	seek
blockwrite	execute	rename	write
flush	reset	writeln	

2. Laufzeitfehler

Die Ausschrift bei Laufzeitfehlern ist:

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

Die Laufzeitfehler sind im Einzelnen

01 Floating point overflow
02 Division by zero attempt
03 Sqrt argument error
    (Argument einer Wurzel <0)
04 Ln argument error
    (Argument von LN <=0)
10 Stringth length error
    (entweder ist die Stringlaenge >255 oder es wurde versucht,einen String der Laenge <>1 in CHAR zu konvertieren)
11 Invalid string index
    (tritt bei copy,delete,insert auf,wenn der Indexausdruck nicht zwischen 1 und 255 liegt)
90 Index out of range
    (Zugriff auf nicht definietenIndex)
91 Skalar or subrange out of range
    (Zugriff auf nicht definiertes Elementeines Teilbereichstyps)
92 Out of integer range
    (bei TRUNC/ROUND sll eine Zahl groesser der groessten Integer-Zahl erzeugt werden)
FF Heap/Stack-collision

3. Lokalisierung des Fehlerortes

Bei der Arbeit im Directmode springt der Editor nach Quittung auf die Fehlermeldung automatisch an die Stelle, an der der Fehler bemerkt wurde.

Bei Laufzeitfehlern in COM- oder CHN-Files ist folgendermassen zu verfahren:

Die in der Fehlermeldung in PC genannte Adresse ist zu notieren. Es ist TURBO aufzurufen und das Programm,das den Fehler brachte, als work-file zu laden. Danach ist die Compiler-Option **O** aufzurufen und das **F**-Komando zu geben. Es erscheint: **Enter PC:** Hier ist die aus der Fehlermeldung bekannte Adresse einzugeben, danach das O-Menue mit Q zu verlassen. Wenn jetzt **E** aufgerufen wird, stellt sich der Cursor an die Stelle, an der der Fehler bemerkt wurde.
Falls der Laufzeitfehler in einem Overlay auftrat, gilt folgendes: Die Fehleradresse zeigt auf den Ort, wo der Fehler in der Overlaygruppe das erste mal auftrat. Das kann der Fehlerort sein, der Fehler kann aber genauso in einem anderen Untermodul der Overlaygruppe aufgetreten sein.

Es gibt keinen allgemeingueltigen Algorithmus,den zum Zeitpunkt des Fehlers gueltige Untermodul der Overlaygruppe zu lokalisieren, wenn nicht aus Ausschriften o. aehnlichem der Modul bestimmt werden kann.

Vor einer Feststellung des Fehlerortes ist also zuerst der den Fehler verursachende Modul zu lokalisieren. Hierbei bietet TURBO keine Untestuetzung.

Um den Fehler im Overlamodul selbst zu lokalisieren, muss der entsprechende Modul beim Uebersetzen so angeordnet sein, dass er der erste der entsprechenden Overlaygruppe ist, dann arbeitet der **F**-Schalter einwandfrei und lokalisiert den Fehlerort !

Beispielprogramme fuer PASCAL

Die folgende Sammlung von Beispielprogrammen ist gedacht als Unterstuetzung fuer das Selbststudium der Programmiersprache PASCAL und als Ergaenzung des Nutzerhandbuches TURBO-PASCAL. Alle hier vorgestellten Beispiele wurden getestet und sind lauffaehig. Die Programme wurden teilweise PASCAL-Lehrbuechern entnommen, aber an TURBO-Besonderheiten angepasst. Soweit die Programme aus Zeitschriften entnommen wurden, wurde das vermerkt.

Beispielprogramme fuer die einfachen Steueranweisungen

* Berechnung der Wurzel aus einer reelen Zahl

```
PROGRAM WURZEL;
{ Beispiel fuer IF - THEN - ELSE        }
VAR ZAHL :REAL;
BEGIN
  CLRSCR;
  WRITE('Eingabe einer Zahl:');
  READLN(ZAHL);
  WRITE('Wurzel aus ',ZAHL:10:5,' ist:');
  IF ZAHL>0 THEN WRITE(SQRT(ZAHL):10:5)
            ELSE WRITE('i * ',SQRT(-ZAHL):10:5);
END.
```

* Programm zur Erstellung von Rechnungen

```
PROGRAM RECHNUNG;
{ DEMONSTRATION DER REPEAT-ANWEISUNG UND AUSGABE AUF DRUCKER }
VAR PREIS,SUMME   :REAL;
    ANZAHL,GESAMT :INTEGER;
    NAME          :STRING[20];
BEGIN
    CLRSCR;
    SUMME:=0;
    GESAMT:=0;
    WRITELN('Ende der R.Erstell.durch Eingabe des Namens ENDE');
    WRITELN(LST,'Artikelname':20,'Anzahl':16,'Preis/Stueck':
                                                13,'Gesamt':13);
    REPEAT
        WRITE('Artikelname                :');
        READLN(NAME);
        IF NAME<>'ENDE' THEN
        BEGIN
          WRITE('Anzahl(ganze Zahl)          :');
          READLN(ANZAHL);
          WRITE('Preis/Stueck(m.Dezimalpunkt):');
          READLN(PREIS);
          WRITE(LST,NAME:20,ANZAHL:16,PREIS:13:2);
          PREIS:=PREIS*ANZAHL;
```

```
          WRITELN(LST,PREIS:13:2);
          GESAMT:=GESAMT+ANZAHL;
          SUMME:=SUMME+PREIS;
        END;
    UNTIL NAME='ENDE';
    WRITELN(LST,'Gesamtanzahl':20,GESAMT:16,'Gesamtpreis':
                                              13,SUMME:13:2);
END.

* Programm zur Darstellung der Bitbelegung einer INTEGER-Zahl

PROGRAM WANDELN;
VAR ZAHL,ZAHL1,I :INTEGER;
    BITMUSTER    :ARRAY[1..16] OF CHAR;
BEGIN
  REPEAT
    CLRSCR;
    WRITELN('Anzeige BIT-Belegung einer INTEGER-Zahl':60);
    WRITELN;
    WRITE('Gewuenschte Zahl:');
    READ(ZAHL);
    ZAHL1:=ZAHL;
    IF ZAHL1<0 THEN ZAHL:=ZAHL+1;
    FOR I:=1 TO 15 DO
      BEGIN
        IF ZAHL1>=0 THEN
        BEGIN
          IF TRUNC(ZAHL/2)*2=ZAHL THEN BITMUSTER[17-I]:='0'
                                  ELSE BITMUSTER[17-I]:='1';
        END
           ELSE
             BEGIN
              IF TRUNC(ZAHL/2)*2=ZAHL THEN BITMUSTER[17-I]:='1'
                                      ELSE BITMUSTER[17-I]:='0';
             END;
        ZAHL:=TRUNC(ZAHL/2);
      END;
    IF ZAHL1<0 THEN BITMUSTER[1]:='1'
               ELSE BITMUSTER[1]:='0';
    WRITELN(' ':6,'Bitmuster=',BITMUSTER);
    READLN(ZAHL);
  UNTIL ZAHL1=0;
END.


* Programm zum Plotten des Verlaufes der Sinusfunktion

PROGRAM SINUSPLOT;
{ BEISPIEL FUER REPEAT UND VARIABLE FORMATIERTE AUSGABE }
VAR  LAUF      :REAL;
     POSITION  :INTEGER;
 BEGIN
     CLRSCR;
     LAUF:=0;
     GOTOXY(25,8);
     WRITE('Druck laeuft');
     REPEAT
{  Die Zahlen 41 und 40 sind abhaengig von der  Papierbreite,hier
```

```
81 Zeichen! }
         POSITION:=41+TRUNC(SIN(LAUF)*40);
         WRITELN(LST,'*':POSITION);
         LAUF:=LAUF+PI/12;
     UNTIL LAUF>2*PI;
END.
```

* Ermittlung der Teiler einer ganzen Zahl

```
PROGRAM TEILER;
{ BEISPIEL WHILE-ANWEISUNG                                    }
VAR  ZAHL,COUNT:INTEGER;
BEGIN
    CLRSCR;
    WRITE('Eingabe der Zahl(ganze Zahl):');
    READLN(ZAHL);
    COUNT:=1;
    Write('Teiler von ',ZAHL,' sind :');
    WHILE COUNT<ZAHL DO
      BEGIN
          IF ZAHL MOD COUNT = 0 THEN WRITE(COUNT,' ');
          COUNT:=COUNT+1;
      END;
END.
```

* Erstellung eines Wochenplanes

```
PROGRAM PLAN;
{ BEISPIEL FUER CASE-ANWEISUNG                                                }
{ ACHTUNG:Als Selektoren fuer CASE sind auch in TURBO nur skalare
zugelassen!}
{ DH:STRING-GROESSEN, FELDER ODER RECORDS SIND NICHT ZUGELASSEN               }
VAR TAG1 :INTEGER;
    TAG  :STRING[10];
BEGIN
    CLRSCR;
    WRITE('Eingabe der Nummer des Wochentages(Montag=1):');
    READLN(TAG1);
    CASE TAG1 OF
      1:TAG:='Montag';
      2:TAG:='Dienstag';
      3:TAG:='Mittwoch';
      4:TAG:='Donnerstag';
      5:TAG:='Freitag';
      6:TAG:='Sonnabend';
      7:TAG:='Sonntag';
      ELSE WRITELN('kein gueltiger Wochentag');
    END;
    IF (TAG1>0) AND (TAG1<8) THEN
        BEGIN
            WRITELN('Heute ist ',tag,':Es ist zu tun:');
            CASE TAG1 OF
                    1,2,3,4:BEGIN
                              WRITELN('frueh aufstehen');
                              writeln('Arbeiten gehen');
                            END;
```

```
                          5:BEGIN
                               WRITELN('frueh aufstehen');
                               WRITELN('Arbeiten gehen');
                               WRITELN('abends feiern!');
                             END;
                         6,7:WRITELN('endlich ausschlafen!');
            END;
    END;
END.
```

* Tabellierung der Wurzel-Funktion

```
PROGRAM TABELLE;
{BEISPIEL FUER FOR-ANWEISUNG                    }
VAR  I   :INTEGER;
     WERT:REAL;
BEGIN
    CLRSCR;
    WRITELN('Wurzeltabelle');
    FOR I:=1  TO 14 DO
      BEGIN
        WERT:=I;
WRITELN('Wurzel aus ',WERT:2:0,' ist gleich:',SQRT(WERT):6:3 );
      END;
END.
```

* Tabellierung der Sinus-funktion

```
PROGRAM TABELLE1;
{ BEISPIEL FUER FOR-ANWEISUNG MIT DOWNTO              }
VAR I:INTEGER;
    A:REAL;
BEGIN
    CLRSCR;
    A:=0;
    FOR I:=15 DOWNTO 0 DO
      BEGIN
        WRITELN(I:2,'  SIN(',A:6:4,')=',SIN(A):6:5);
        A:=A+PI/15;
      END;
END.
```

* Beispiel fuer Eingabepruefung numerisch-nichtnumerisch

```
PROGRAM PRUEF;
{ PRUEFUNG EINER EINGABE AUF NUMERISCH MIT DER VAL-PROZEDUR }
VAR    STR      :STRING[20];
       ZAHL     :REAL;
       FEHLERORT:INTEGER;
BEGIN
    CLRSCR;
    WRITELN('Abbruch -> 0 eingeben');
    STR:='A';
    WHILE STR<>'0' DO
```

```
      BEGIN
          WRITE('Zahl eingeben:');
          READLN(STR);
          VAL(STR,ZAHL,FEHLERORT);
          IF FEHLERORT=0 THEN WRITELN('numerisch richtig!')
                         ELSE WRITELN('*':FEHLERORT+14);
      END;
END.
```

Beispielprogramme fuer Prozeduren und Funktionen
---

* Verwendung einer Prozedur

```
PROGRAM UNTERPROGRAMME;
{ BEISPIEL FUER PROZEDUREN OHNE PARAMETER                    }
VAR SUM,A,B:INTEGER;
PROCEDURE ADDIER;
    BEGIN
       SUM:=A+B;
     END;
{ HIER BEGINNT DAS HAUPTPROGRAMM                             }
BEGIN
    CLRSCR;
    READLN(A);
    READLN(B);
    ADDIER;
    WRITELN(' Summe aus ',A,' und ',B,' ist ',SUM);
END.
```

* Verwendung rekursiver Prozeduren

```
PROGRAM HANOI;
{$A-}            { SCHALTER FUER REKURSIVEN CODE              }
{ BEISPIEL FUER PROZEDUREN MIT PARAMETERN : TUERME VON HANOI  }
{ ACHTUNG - HIER WERDEN REKURSIVE PROZEDUREN VERWENDET,DESHALB }
{ IST ES NOETIG,DEN A- SCHALTER ZU SETZEN!(STANDARD IST A+    }
{
{ SPIELREGELN                                                 }
{ DIE AUF TURM 1 BEFINDLICHEN PLATTEN SIND UNTER ZUHILFENAME  }
{ VON TURM 2 NACH TURM 3 ZU BRINGEN.ES DARF NIE EINE GROESSERE }
{ PLATTE UEBER EINER KLEINEREN LIEGEN!                        }
{ DIE PLATTEN AUF TURM 1 SIND GEORDNET,DH ES LIEGT NIE EINE   }
{ GROESSERE PLATTE UBER EINER KLEINEREN!                      }
{ SIE KOENNEN GERNE VERSUCHEN,MIT WENIGER ZUEGEN ALS DER      }
{ AUSZUKOMMEN                                                 }
VAR TOTAL:INTEGER;
    DFLAG:CHAR;
PROCEDURE SCHIEBETURM(HOEHE,VON,NACH,MIT : INTEGER);
  PROCEDURE SCHIEBPLATTE(NIMMWEG,SETZE : INTEGER);
      BEGIN     { SCHIEBPLATTE                                }
IF (DFLAG='J') OR (DFLAG='j')THEN WRITELN(LST,NIMMWEG,'->',SETZE)
                              ELSE  WRITELN(NIMMWEG,'->',SETZE);
      END;
```

```
BEGIN           { SCHIEBETURM                                    }
    IF HOEHE>0 THEN
        BEGIN
            SCHIEBETURM(HOEHE-1,VON,MIT,NACH);
            SCHIEBPLATTE(VON,NACH);
            SCHIEBETURM(HOEHE-1,MIT,NACH,VON);
         END;
 END;

BEGIN            { HAUPTPROGRAMM                                  }
    CLRSCR;
WRITE('Sollen die Zuege des Rechners ausgedruckt werden (J/N):');
    READLN(DFLAG);
    WRITE('Anzahl der Platten auf dem ersten Turm:');
    READLN(TOTAL);
    WRITELN('angezeigt werden die noetigen Bewegungen der
                                  Platten von -> nach');
    SCHIEBETURM(TOTAL,1,3,2);
END.


* Verwendung von Funktionen

PROGRAM FUNKTIONEN;
{ BEISPIEL FUER FUNKTIONEN                                       }
VAR X1,X2,Y1,Y2 : REAL;

FUNCTION DISTANCE(X1,X2,Y1,Y2:REAL) : REAL;
    BEGIN
        DISTANCE:=SQRT(SQR(X2-X1)+SQR(Y2-Y1));
    END;

BEGIN  { HAUPTPROGRAMM                                           }
    CLRSCR;
    WRITELN('Abbruch mit X1=X2 und Y1=Y2');
    REPEAT
        WRITE(' X1,Y1 (Punkt 1) eingeben:');
        READLN(X1,Y1);
        WRITE(' X2,Y2 (Punkt 2) eingeben:');
        READLN(X2,Y2);
        WRITE('Der Abstand von Punkt 1 nach Punkt 2 ist:');
        WRITELN(DISTANCE(X1,X2,Y1,Y2):16:5);
    UNTIL (X1=X2) AND (Y1=Y2);
END.



Beispielprogramme zur Dateiarbeit
---------------------------------

* Aufbauen und Ausdrucken einer Datei,Verwendung von Records


PROGRAM FILE1;
{ BEISPIEL FUER ANLEGEN EINER DATEI UND SCHREIBEN  }
TYPE ADR = RECORD
               NAME       :STRING[20];
```

```
              WOHNORT    :STRING[20];
              STRASSE    :STRING[20];
              NUMMER     :INTEGER
            END;
 VAR ADRESSE    :ADR;
     DATEINAME  :STRING[8];
     DATEI      :STRING[10];
     VERZEICHNIS:FILE OF ADR;
     I          :INTEGER;
     LAUFWERK   :STRING[2];
 BEGIN
     CLRSCR;
     WRITELN('Aufbau eines Adressenverzeichnisses':60);
     WRITE('Eingabe des Dateinamens (max 8 Zeichen!):');
     READLN(DATEINAME);
     WRITE('Eingabe des Laufwerkes  (Form lw:)      :');
     READLN(LAUFWERK);
     WRITELN;
     DATEI:=CONCAT(LAUFWERK,DATEINAME);
     ASSIGN(VERZEICHNIS,DATEI);
     REWRITE(VERZEICHNIS);   { FUER SCHREIBEN OEFFNEN          }
     WRITELN('Eingabeende -> fuer Name * eingeben ');
     ADRESSE.NAME:='A';

     REPEAT
         WRITE('Name          :');
         READLN(ADRESSE.NAME);
         IF ADRESSE.NAME<>'*' THEN
             BEGIN
                 WRITE('Wohnort       :');
                 READLN(ADRESSE.WOHNORT);
                 WRITE('Strasse       :');
                 READLN(ADRESSE.STRASSE);
                 WRITE('Hausnummer    :');
                 READLN(ADRESSE.NUMMER);
                 WRITE(VERZEICHNIS,ADRESSE);
             END;
     UNTIL ADRESSE.NAME='*';
     CLOSE(VERZEICHNIS);
     WRITELN('Druck der Adressenliste');
     RESET(VERZEICHNIS);    { FUER LESEN OEFFNEN              }
     WRITELN(LST,'Aktuelle Adressenliste');
     WRITELN(LST,'Eine Aktualisierung ist ueber die Folgenummer
                                              moeglich!');
     I:=0;
     WHILE NOT EOF(VERZEICHNIS) DO
     BEGIN
         WRITE(LST,I:3,' ':2);
         READ(VERZEICHNIS,ADRESSE);
          WRITE(LST,ADRESSE.NAME:20,ADRESSE.WOHNORT:20,ADRESSE.
                                        STRASSE:20);
         WRITELN(LST,ADRESSE.NUMMER:3);
         I:=I+1
     END;
     WRITELN(LST,'Ende der Adressenliste');
     CLOSE(VERZEICHNIS);
 END.
```

```
* Beispiel fuer aktualisieren einer Datei


PROGRAM FILE2;
{ AKTUALISIERUNG DER IN PROGRAMM FILE1 ERSTELLTEN DATEI    }
{ DIREKTZUGRIFF UEBER SEEK                                 }
{ ACHTUNG -> KEINE DATEIERWEITERUNG VORGESEHEN,DAFUER      }
{ PROGRAMM FILE3 BENUTZEN!                                 }
TYPE ADR = RECORD
               NAME       :STRING[20];
               WOHNORT    :STRING[20];
               STRASSE    :STRING[20];
               NUMMER     :INTEGER
           END;
 VAR ADRESSE    :ADR;
     DATEINAME  :STRING[8];
     DATEI      :STRING[10];
     VERZEICHNIS:FILE OF ADR;
     I          :INTEGER;
     LAUFWERK   :STRING[2];
 BEGIN
     CLRSCR;
     WRITELN('Aktualisierung des Adressenverzeichnisses':60);
     WRITE('Eingabe des Dateinamens (max 8 Zeichen!):');
     READLN(DATEINAME);
     WRITE('Eingabe des Laufwerkes  (Form lw:)      :');
     READLN(LAUFWERK);
     WRITELN;
     DATEI:=CONCAT(LAUFWERK,DATEINAME);
     ASSIGN(VERZEICHNIS,DATEI);
     RESET(VERZEICHNIS);   { FUER LESEN/SCHREIBEN OEFFNEN      }
     WRITELN('Eingabeende -> fuer Folgenummer 999 eingeben ');
     WRITE('hoechst zulaessige Folgenummer ist:');
     WRITELN(FILESIZE(VERZEICHNIS)-1);
     I:=0;
     REPEAT
         WRITE('Folgenummer:');
         READLN(I);
         IF I<999  THEN
             BEGIN
 IF I>=FILESIZE(VERZEICHNIS) THEN WRITELN('Dateiende erreicht')
                 ELSE
                   BEGIN
                     SEEK(VERZEICHNIS,I);
                     READ(VERZEICHNIS,ADRESSE);
                    WRITELN('Name          :',ADRESSE.NAME:20);
                    WRITELN('alter Wohnort :',ADRESSE.WOHNORT);
                    WRITE('neuer Wohnort  :');
                    READLN(ADRESSE.WOHNORT);
                    WRITELN('alte Strasse  :',ADRESSE.STRASSE);
                    WRITE('neue Strasse   :');
                    READLN(ADRESSE.STRASSE);
                    Writeln('Hausnummer alt:',ADRESSE.NUMMER);
                    WRITE('Hausnummer neu:');
                    READLN(ADRESSE.NUMMER);
{ DAS ZWEITE SEEK IST HIER UNBEDINGT NOETIG,DA NACH DER  LESEOPE}
{RATION DER SATZZEIGER DER DATEI AUTOMATISCH UM EINS VORGESTELLT}
```

```
{WURDE       }
{ OHNE DAS ZWEITE SEEK WUERDE DER NACHFOLGESATZ UEBERSCHRIEBEN !}
                     SEEK(VERZEICHNIS,I);
                     WRITE(VERZEICHNIS,ADRESSE);
                   END;
             END;
     UNTIL I>=999;
     CLOSE(VERZEICHNIS);
     WRITELN('Druck der aktualisierten Adressenliste');
     RESET(VERZEICHNIS);{F.LESEN OEFFNEN,DATEIZEIGER AUF ANFANG }
     WRITELN(LST,'Aktuelle Adressenliste');
     WRITELN(LST,'Eine Akt.ist ueber die Folgenummer moeglich!');
     I:=0;
     WHILE NOT EOF(VERZEICHNIS) DO
     BEGIN
         WRITE(LST,I:3,' ':2);
         READ(VERZEICHNIS,ADRESSE);
WRITE(LST,ADRESSE.NAME:20,ADRESSE.WOHNORT:20,ADRESSE.STRASSE:20);
         WRITELN(LST,ADRESSE.NUMMER:3);
         I:=I+1
     END;
     WRITELN(LST,'Ende der Adressenliste');
     CLOSE(VERZEICHNIS);
END.


PROGRAM FILE3;
{ ERWEITERN EINER BESTEHENDEN DATEI                    }
TYPE ADR = RECORD
               NAME       :STRING[20];
               WOHNORT    :STRING[20];
               STRASSE    :STRING[20];
               NUMMER     :INTEGER
            END;
 VAR ADRESSE    :ADR;
     DATEINAME  :STRING[8];
     DATEI      :STRING[10];
     VERZEICHNIS:FILE OF ADR;
     I          :INTEGER;
     LAUFWERK   :STRING[2];
 BEGIN
     CLRSCR;
     WRITELN('Erweiterung eines Adressenverzeichnisses':60);
     WRITE('Eingabe des Dateinamens (max 8 Zeichen!):');
     READLN(DATEINAME);
     WRITE('Eingabe des Laufwerkes  (Form lw:)      :');
     READLN(LAUFWERK);
     WRITELN('Datei enthaelt ',FILESIZE(VERZEICHNIS),' Saetze');
     WRITELN;
     DATEI:=CONCAT(LAUFWERK,DATEINAME);
     ASSIGN(VERZEICHNIS,DATEI);
     RESET(VERZEICHNIS);
     WRITELN('Eingabeende -> fuer Name * eingeben ');
     ADRESSE.NAME:='A';
{NACHFOLGEND  POSITIONIERUNG  AUF  DATEIENDE,DAMIT  ERWEITERT   }
{WERDEN KANN  }
     SEEK(VERZEICHNIS,FILESIZE(VERZEICHNIS));
     REPEAT
```

```
         WRITE('Name          :');
         READLN(ADRESSE.NAME);
         IF ADRESSE.NAME<>'*' THEN
             BEGIN
                 WRITE('Wohnort       :');
                 READLN(ADRESSE.WOHNORT);
                 WRITE('Strasse       :');
                 READLN(ADRESSE.STRASSE);
                 WRITE('Hausnummer    :');
                 READLN(ADRESSE.NUMMER);
                 WRITE(VERZEICHNIS,ADRESSE);
             END;
     UNTIL ADRESSE.NAME='*';
     CLOSE(VERZEICHNIS);
     WRITELN('Druck der Adressenliste');
     RESET(VERZEICHNIS);    { FUER LESEN OEFFNEN              }
     WRITELN(LST,'Erweiterte Adressenliste');
     I:=0;
     WHILE NOT EOF(VERZEICHNIS) DO
     BEGIN
         WRITE(LST,I:3,' ':2);
         READ(VERZEICHNIS,ADRESSE);
WRITE(LST,ADRESSE.NAME:20,ADRESSE.WOHNORT:20,ADRESSE.STRASSE:20);
         WRITELN(LST,ADRESSE.NUMMER:3);
         I:=I+1
     END;
     WRITELN(LST,'Ende der Adressenliste');
     CLOSE(VERZEICHNIS);
END.
```

Spezialprogramme
~~~~~~~~~~~~~~~~
Die hier vorgestellten Spezialprogramme sollen Beispiele fuer maschinennahes Programmieren zeigen.

* Dump vom Bildschirm

```
PROGRAM BSDRUCK;
{ DRUCK DES BILDSCHIRMINHALTES                                  }
CONST   BSANF       =$F800;  { fuer BAB1 $FC00                  }
        ZEICHENZAHL =1920;   { fuer BAB1 1024                   }
        SPALTENZAHL =80;     { fuer BAB1 64                     }
{ Moeglich waere hierauch eine Prozedurvereinbarung der Form :  }
{ PROCEDURE BSDRUC(BSANF,ZEICHENZAHL,SPALTENZAHL:INTEGER);      }
VAR     I           :INTEGER;
        ZEICHEN     :BYTE;
BEGIN
    WRITELN(LST);
    FOR I:=1 TO ZEICHENZAHL DO
      BEGIN
        ZEICHEN:=MEM[I+BSANF-1];
        IF ZEICHEN<$20 THEN ZEICHEN:=$20;
        IF I MOD SPALTENZAHL=0 THEN WRITELN(LST);
        WRITE(LST,CHR(ZEICHEN));
      END;
```
~~~~~~~~~~~~~~~~

```
      WRITELN(LST);
END.


* Programm zur Abarbeitung beliebiger CHN-Files


(*    anwenderprogramm ohne laufzeit-bibliothek        *)
(*    ubersetzen (h-option  dateiextensio .chn)        *)
(*    Utility zum Aufruf von TURBO-PASCAL-Programmen   *)
(*     ohne Laufzeit-Bibliothek                        *)
(*      mc 7/85 s 66                                   *)
program pascal (input,output);
var dname:string[128];(*string fuer den Dateinamen *)
    fileident:text;(*fileidentifier*)
    loop:integer; (*schleifenvariable*)
 begin
  delete (dname,1,length(dname));(*loeschen des strings*)
  for loop:=1 to mem[$80]-1 do (*kopieren der kommandozeile*)
    dname:=dname+char(mem[$81+loop]);
  insert ('.chn',dname,length(dname)+1);(*dateityp anfuegen*)
  assign(fileident,dname); (*dateinamen vereinbaren*)
  (*$i-*) chain(fileident);(*$i+*) (*programm aufrufen*)
  if ioresult <> 0 then (*datei nicht gefunden ?*)
    repeat
      writeln; (*fehlermeldung ausgeben*)
      writeln ('Datei mit namen ',dname,' nicht gefunden !');
      writeln; (* nach neuem dateinamen fragen*)
      write('korr.Namen oder Return fuer Ende eingeben:  ');
      readln(dname); (*dateinamen einlesen*)
      if length (dname)=0 then halt;(*leere eingabe?*)
insert ('.chn',dname,length(dname)+1); (*dateityp anfuegen*)
      assign(fileident,dname);(*$i+*) (*programm aufrufen*)
    until ioresult =0
end.



* Programm zum menuegesteuerten Aufruf von COM-Files


program CPMMenue;
var   dma             : array[0..127] of byte;
      ComFiles        : array[0..63]  of string[11];
      NoOfComs        : byte;
      selection       : byte;
      firstspace      : byte;
      ch              : char;
      progname        : string[12];
      progfile        : file;
      i               : integer;

procedure ScanDirectory (var entries   : byte);

var fcb               : array[0..35] of byte;
    a,i               : integer;
    counter           : byte;
```

```
procedure StoreEntry (NofEnt  : integer);

begin
  NofEnt:=NofEnt*32+1 ;
  ComFiles [counter]:='';
  for i:=0 to 11 do
      ComFiles [counter] :=
          concat (ComFiles [counter],
              chr (dma[NofEnt+i] mod 128));
      if copy (ComFiles[counter], 9, 3) = 'COM' then
      begin
      delete (ComFiles[counter], 9, 3);
      counter:=succ (counter);
      end;
end;

begin
   counter:=0;
   fillchar(fcb,12,ord('?'));
   fcb[0] := 0;
   fcb[12]:= 0;
   bdos($1A,addr(dma));
   a := bdos($11,addr(fcb));
   while a<255 do begin
      StoreEntry(a);
      a := bdos($12,addr(fcb));
   end;
   entries:=counter;
end; (* Directory *)

(*----------------------------------------*)


procedure DisplayProgName  (n  : byte);
var     x   : byte;
        y   : byte;

begin
  x:=n mod 5 * 12 + 1;
  y:=n div 5 + 4;
  gotoxy (x,y);
  write (ComFiles [n]);
end;

begin
  crtinit;
  clrscr;
  ScanDirectory (NoOfComs);
  writeln('<SPACE> to advance,<CR> to select');
  for i := 0 to NoOfComs -1 do
      DisplayProgName (i);
    DisplayProgName(0);
    selection:=0;
    repeat
      read (kbd, ch);
      if ch = ' ' then begin
         displayprogname (selection);
         selection := succ (selection);
```

```
         if selection = NoOfComs then selection := 0;
         DisplayProgName (selection);
      end;
      until ch = chr(13);
      progname:=ComFiles[selection];
      firstspace := pos (' ',progname);
        if firstspace > 0 then
           delete (progname, firstspace,
           length (progname)-firstspace+1);
    progname := progname+'.COM';
    clrscr;
    WRITE(PROGNAME);
   BDOS($1A,$80);
    assign (progfile, progname);
(*$I-*)    execute (progfile);(*$I+*)
end.



* Prozedur zum Umweisen der Ausgaben


{ Mit dieser Prozedur ist ein Parallelschalten des Druckers mit}
{^P moeglich.                                                  }
{ aus : mc 7/85                                                }
procedure sysinit;
var echocheck:char;
function sysinput:char;
begin echocheck:=chr(bdos(1,0));
      if echocheck=chr($7f)
      then bdos(2,32);
      sysinput:=echocheck
end;
procedure sysoutput(zeichen:char);
begin if zeichen<>echocheck
      then bdos(2,ord(zeichen))
      else echocheck:=chr(0);
      if bdos(11,0)<>0
      then if bdos(1,0)=3
           then bdos(0,0);
end;
begin ConInPtr:=addr(sysinput);
      ConOutPtr:=addr(sysoutput);
      echocheck:=chr(0)
end;
```