robotron

SOFTWARE
DOKUMENTATION

Anwendung des BASIC-Systems

Anwendung des BASIC-Systems

VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt
VEB Robotron-Bueromaschinenwerk
Soemmerda

Die vorliegende 1. Auflage der Dokumentation "Anwendung des BASIC-Systems" DCP 3.20 entspricht dem Stand vom 29.2.88 und unterliegt nicht dem Aenderungsdienst.

Die Dokumentation wurde durch ein Kollektiv des

VEB Robotron Buchungsmaschinenwerk

Karl-Marx-Stadt

Software-Zentrum

erarbeitet.

Bitte senden Sie uns Ihre Hinweise, Kritiken, Wuensche oder Forderungen zur Dokumentation zu.

VEB Robotron Buchungsmaschinenwerk
Karl-Marx-Stadt
Postschliessfach 129
Karl-Marx-Stadt
9010

III-12-12 Kv 312/89

I N H A L T S V E R Z E I C H N I S

		Seite
0.	Einleitung	5
1.	Textaufbereitung	5
2.	Ein-/Ausgabe	10
2.1.	Eingabe	10
2.2.	Ausgabe auf Bildschirm (Bildschirmarbeit)	13
2.3.	Ausgabe auf Drucker	15
2.4.	Spezielle Steuerzeichenfolgen zur Bildschirm- und Tastatursteuerung fuer BASIC-Compiler	16
3.	Arbeit mit Ferldern/Bereichen	18
3.1.	Ueberblick und Erklaerung der Begriffe	18
3.2.	Eindimensionale Felder	18
3.3.	Zweidimensionale Felder	21
4.	Programmueberlagerung	22
4.1.	Einmischen von Programmen oder Programmteilen mit dem Kommando MERGE	22
4.2.	Verketten von Programmen mit CHAIN	24
4.3.	Uebergabe von Variablen	26
5.	Fehlerbehandlung	28
5.1.	Fehlerbehandlung mit ON ERROR	28
5.2.	Erzeugen von Fehlercodes mit ERDR	30
6.	Suchen, Sortieren, Mischen und Gruppieren von Daten	30
6.1.	Suchverfahren	31
6.1.1.	Sequentielles Suchen	31
6.1.2.	Binaeres Suchen	32
6.2.	Sortieren	35
6.2.1.	Zahlen unmittelbar sortieren	35
6.2.2.	Sortieren ueber Zeiger	38
6.2.3.	Zeichenketten sortieren	40
6.3.	Mischen von Dateien	41
6.4.	Gruppieren von Daten (Gruppenwechsel)	42
7.	Dateiarbeit	43
7.1.	Sequentielle Dateien	43
7.1.1.	Menuesteuerung	43
7.1.2.	Dateiweise Datenverkehr	44
7.1.3.	Verarbeitung von Feldern in Unterprogrammen	45
7.2.	Direktzugriffsdateien	48
7.2.1.	Datei mit konstanter Datensatzlaenge	49
7.2.2.	Overlay durch Verkettung von Programmen	50
7.2.3.	Datensatzweiser Datenverkehr	50

7.2.4. Direkt Adressierung des Datensatzes 51
7.3. Indexsequentielle Datei 58
7.4. Gekettete Liste 62
7.5. Binaerer Baum 66
7.6. Verkettete Dateien und Datenbank 72

8. Grafik 74
8.1. Ueberblick 74
8.2. Text-Modus 74
8.3. Grafik-Modus 75
8.3.1. Grafik mittlerer Aufloesung 75
8.3.2. Grafik hoher Aufloesung 77
8.4. Programmbeispiele 78

9. Musik 82
9.1. Anweisungen SOUND und PLAY 82
9.2. Programmieren eines Liedes 83

10. Datenfernverarbeitung 86
10.1. Serielle Kommunikationsparameter 86
10.2. Anweisungen und Funktionen fuer Datenfernverarbeitun 87
10.2.1 Eroeffnen einer Datenfernverarbeitungsdatei 87
10.2.2 Datenfernverarbeitungsein-/-ausgabe 88
10.3. Operationen der Steuersignale 88
10.3.1 Steuerung der Ausgabesignale mit OPEN 89
10.3.2 Benutzung der Eingabesteuersignale 89
10.4. Beispiel 89

11. Prozessorarbeit 91
11.1. Hauptspeicherbelegung 91
11.2. Hauptspeicherzuordnung fuer einige Unterprogramme 93
11.2.1 Unter DCP geladene Unterprogramme fuer BASIC 94
11.2.2 Unterprogramme innerhalb des BASIC-Datensegments 94
11.2.3 Unterprogramme oberhalb des BASIC-Datensegments 95
11.3. Kommunikation zwischen BASIC und Unterprogrammen in Maschinensprache 96
11.4. Die Anweisung CALL 98
11.5. Laden und Aufrufen von Unterprogrammen in Maschinensprache 99
11.5.1 Laden mit POKE und Zuordnung zu einem Bereich 100
11.5.2 Aufrufen eines Unterprogrammes von einer Datei mit BLOAD 102
11.5.3 Laden eines Unterprogrammes als residente DCP-Erweiterung 108

0. Einleitung

Die vorliegende Dokumentation ist erarbeitet worden, um besonders Anfaengern einige Hinweise zur Programmierung in BASIC zu geben.
Diese Dokumentation dient als Ergaenzung der Dokumentation "Bedienungsanleitung und Sprachbeschreibung fuer BASIC-Interpreter (BASI)" und beinhaltet verschiedene praktische Programmierbeispiele.
Besondere Aufmerksamkeit wurde der Dateiarbeit gewidmet, d.h. dem Erstellen und Bearbeiten von Datenbestaenden. An vielen Beispielen werden Zugriffsarten auf Dateien und die Reihenfolge der einzelnen Programmschritte erlaeutert.

In den Kapiteln Grafik und Musik werden die Moeglichkeiten des BASIC-Interpreters gezeigt, grafische Abbildungen bzw. Diagramme darzustellen und Toene zu erzeugen.

Die Erfahrung zeigte, dass es in verschiedenen Anwendungsfaellen notwendig ist, Assembler-Unterprogramme in BASIC-Programme einzubinden. In der vorliegenden Dokumentation werden einige detaillierte Hinweise zum Einbinden von Assembler-Unterprogrammen in BASIC-Programme, die im Interpreter-Status abgearbeitet werden sollen, gegeben.

Die angefuehrten Beispiele und Programmausschnitte sollen als Anregung dienen und koennen nach Bedarf in Anwenderprogrammen verwendet werden.

1. Textaufbereitung

In diesem Kapitel werden einige Programmbeispiele gegeben, die der Aufbereitung einer Zeichenkette (eines Textes) bzw. eines numerischen Wertes dienen.
Es werden die verschiedenen Standardfunktionen verwendet, die BASIC fuer die Zeichenkettenverarbeitung bereitstellt.

Beispiel: **Text unterstreichen**

Hier wird das Unterstreichen ueber eine Schleife und ueber die Funktion **STRING$(L,45)** gezeigt.
Fuer **STRING$(L,45)** kann man auch **STRING$(L,"_")** schreiben.

```
100 '*********************************************************
110 '
120 '          Programm UNT
130 '      (Unterstreichen Text)
140 '
150 '*********************************************************
160 PRINT "Text unterstreichen"
170 PRINT "Text eingeben"
180 INPUT T$
190 LET L=LEN(T$)
200 PRINT STRING$(L,45)
210 PRINT T$
220 FOR I=1 TO L
230   PRINT "_";
240 NEXT I :END
```

Beispiel: **Zahl um fuehrende Nullen erweitern**

```
100 '*********************************************************
110 '
120 '          Programm NULL
130 '    (Zahl um fuehrende Nullen erweitern)
140 '
150 '*********************************************************
160 PRINT
170 PRINT "Zahl um fuehrende Nullen erweitern"
180 PRINT"====================================="
190 INPUT "Anzahl der Stellen insgesamt: ",A
200 INPUT "Eingabe Zahl: ";Z%
210 LET Z$=STR$(Z%)
220 LET Z$=RIGHT$(Z$,LEN(Z$)-1)
230 LET Z$=RIGHT$("00000000000000"+Z$,A)
240 PRINT Z$
250 END
```

Beispiel: **Zentrieren Zeile**

Ein eingegebener Text wird mittig auf einer Bildschirmzeile waehlbarer Laenge dargestellt.
Vor und nach dem Text wird die Zeile mit variablen Zeichen aufgefuellt.

```
100 '************************************************************
110 '
120 '          Programm ZENT
130 '      (Zentrieren von Text)
140 '
150 '************************************************************
160 '
170 'T$ :        TEXTZEILE
180 'L  :        ZEILENLAENGE
190 'V$, N$ :    ZEICHEN VOR/NACH DEM TEXT
200 'V, N   :    ANZAHL ZEICHEN VOR/NACH DEM TEXT
210 '
220 '************************************************************
230 '
240 PRINT
250 LINE INPUT "Text? ",T$
260 INPUT "Zeilenlaenge ";L
270 INPUT "Zeichen vor/nach dem Text ";V$,N$
280 PRINT
290 LET L=L-LEN(T$)
300 LET V=INT(L/2)
310 LET N=L-V
320 PRINT
330 PRINT STRING$(V-1,V$);
340 PRINT " ";T$;" ";
350 PRINT STRING$(N-1,N$)
360 GOTO 250
```

Beispiel: **Austausch von Zeichenketten**

Dieses Programm durchmustert die eingegebene Zeile und sucht eine bestimmte Zeichenfolge, die gegen eine neue ausgetauscht werden kann. Die Laengen duerfen unterschiedlich sein.

```
100 '*********************************************************
110 '
120 '          Programm AUSTZK
130 '     (Korrektur von Zeichenfolgen)
140 '
150 '*********************************************************
160 '
170 'T$     : zu korrigierende Textzeile
180 'A$     : alter Text
190 'N$     : neuer Text
200 'A,N    : Laenge alter/neuer Text
210 'P      : Position des gefundenen Textes
220 '
230 '*********************************************************
240 '
250 PRINT
260 PRINT "zu korrigierende Textzeile"
270 LINE INPUT T$
280 PRINT "alte Zeichenkette, neue Zeichenkette";
290 INPUT A$,N$
300 GOSUB an Austausch
330 '
310 PRINT "->";T$
320 END
330 PRINT
340 LET A=LEN(A$)
350 LET N=LEN(N$)
360 LET P=1
370 '
380 LET P=INSTR(P,T$,A$)
390 IF P=0 THEN RETURN
400 '
410 LET T$=LEFT$(T$,P-1)+N$+MID$(T$,P+A)
420 LET P=P+N
430 GOTO 380
```

Beispiel: **Automatischer Randausgleich**

Hier wird eine Eingabezeile durch Einfuegen von Leerzeichen in eine Ausgabezeile konstanter Laenge umgewandelt.

```
100 '*********************************************************
110 '
120 '          Programm RAND
130 '    (Automatischer Randausgleich)
140 '
150 '*********************************************************
160 '
170 'EIN$, LE: Eingabezeile, Laenge von EIN$
180 'AUS$, LA: Ausgabezeile, Laenge von AUS$
190 'BE:       Anzahl von Leerzeichen in EIN$
200 'BA:       Anzahl von Leerzeichen in AUS$ hinzufuegen
210 'BV:       Anzahl von Leerzeichen gerade verarbeitet
220 'Z$:       Zeichen zum Hinzufuegen
230 '
240 '*********************************************************
250 '
260 PRINT "Automatischer Randausgleich"
270 PRINT "============================"
280 PRINT
290 INPUT "Eingabezeile: " ;EIN$
300 LET LE=LEN(EIN$)
310 INPUT "Laenge fuer Ausgabezeile" ; LA
320 PRINT
330 LET BE=0 : LET BA=LA-LE : LET AUS$=""
340 FOR Z=1 TO LA
350     IF MID$(EIN$,Z,1)=" " THEN LET BE=BE+1
360 NEXT Z
370 FOR Z=1 TO LA
380     LET Z$=MID$(EIN$,Z,1)
390     LET AUS$=AUS$+Z$
400     IF Z$ <>" "THEN 480 ELSE 410
410        LET BV = INT (BA/BE)
420        IF BV<1 THEN 470 ELSE 430
430           FOR X=1 TO BV
440             LET AUS$=AUS$+" "
450             LET BA=BA-1
460           NEXT X
470        LET BE=BE-1
480     NEXT Z
490 PRINT "---------/---------/---------/---------/"
500 PRINT EIN$ : PRINT AUS$
510 END
```

2. Ein-/Ausgabe

2.1. Eingabe

Bereits bei der Eingabe sollen Daten auf Richtigkeit (Grenzwerte, Kapazitaet usw.) kontrolliert werden.

Beispiel: Datumpruefung

Es wird geprueft, ob ein eingegebenes Datum in einer vorgegebenen Zeitspanne liegt. Ueber die Funktion **VAL** wird das Datum in einen numerischen Wert gewandelt, um den Vergleich vornehmen zu koennen.

```
100 '*******************************************************
110 '
120 '          Programm DATUM
130 '         (Datumpruefung)
140 '
150 '*******************************************************
160 '
170 DIM D$(3)        'Zeichenkettenfeld fuer Datumangabe
180 DIM T(3)         'Feld fuer Tage
190 DIM M(3)         'Feld fuer Monate
200 DIM J(3)         'Feld fuer Jahr
210 '
220 '*******************************************************
230 '
240 PRINT "Pruefen Datum"
250 INPUT "Untere Datumgrenze",D$(1)
260 INPUT "Obere Datumgrenze ",D$(2)
270 INPUT "Testdatum         ",D$(3)
280 FOR I=1 TO 3
290   LET T(I)=VAL(LEFT$(D$(I),2))
300   LET M(I)=VAL(MID$(D$(I),4,2))
310   LET J(I)=VAL(RIGHT$(D$(I),2))
320 NEXT I
330 IF J(3)>J(2) THEN PRINT "Jahr zu gross" : GOTO 270
340 IF J(3)<J(1) THEN PRINT "Jahr zu klein" : GOTO 270
350 IF J(3)=J(2) AND M(3)>M(2) THEN PRINT
                 "Monat zu gross" : GOTO 270
360 IF J(3)=J(1) AND M(3)<M(1) THEN PRINT
                 "Monat zu klein" : GOTO 270
370 IF J(3)=J(2) AND M(3)=M(2) AND T(3)<T(1) THEN PRINT
                 "Tag zu klein" : GOTO 270
380 IF J(3)=J(1) AND M(3)=M(1) AND T(3)>T(2) THEN PRINT
                 "Tag zu gross" : GOTO 270
390 PRINT
400 PRINT "Datum im gueltigen Bereich"
410 END
```

Beispiel: **Maskierte Eingabe**

Es wird eine maskierte Eingabe auf dem Bildschirm ueber ein Unterprogramm ermoeglicht, wobei die Eingabeposition, die Maske und die Anzahl der Interpunktionszeichen (wie Punkt, Strich ....) in der Maske festzulegen sind.

Im Unterprogramm erfolgen verschiedene Eingabekontrollen:

- Warten bis eine Taste betaetigt wird.

```
1270 H$=INKEY$ : IF H$="" THEN GOTO 1270
```

- Wird eine Taste mit dem Code <48 oder >57 (also keine Zifferntaste) bedient, wird die Eingabe ignoriert.

```
1300 IF ASC(H$)<48 OR ASC(H$)>57 THEN GOTO 1210
```

- Bei Betaetigen der Rueckschrittaste (Code 8) wird die Eingabe geloescht. Es kann neu eingegeben werden.

```
1280 IF ASC(H$)=8 THEN H3$="": GOTO 1080
```

- Bei <ENTER>-Betaetigung (Code 13) wird die Eingabe beendet und das UP verlassen.

```
1290 IF ASC(H$)=13 THEN GOTO 1510
1510 RETURN
```

- Die maximale Eingabelaenge entspricht der Laenge der Eingabemaske.

Beispiel: MASKE

```
 10 '************************************************************
 20 '
 30 '           Programm MASKE
 40 '         (Maskierte Eingabe)
 50 '
 60 '************************************************************
 70 '
 80 'zu uebergebende Parameter:
 90 'SP%     : Positionieren Spalte
100 'ZE%     : Positionieren Zeile
110 'MASK%   : Eingabemaske
120 'HV%     : max. vorkommende Anzahl Formatzeichen +1
130 '          (im Beispiel: Punkte bzw. Striche)
140 '
150 'interne Parameter:
160 'H%, H1%, H2%, H3%, HI%, H%(), H$, H1$,H2$,H3$
170 '
180 CLS
190 '
200 '************************************************************
210 '
```

```
 220 '1. Eingabe mit Maske
 230 '
 235 H2$="."
 240 HV%=3
 250 ZE%=2 : SP%=40
 260 MASK$="##.##.##"
 270 LOCATE 2,1:PRINT "Datum: ";
 280 GOSUB 1080 : PRINT : PRINT
 290 '
 300 '2. Eingabe mit Maske, wobei die Maske veraendert wurde.
                          Spaltenposition  bleibt erhalten
 310 ZE%=4
 320 MASK$="#####"
 330 PRINT "Nummer: ";
 340 GOSUB 1080 : PRINT : PRINT
 350 '
 360 '3. Eingabe
 370 '
 380 H2$="-"
 390 HV%=3
 400 ZE%=6
 410 MASK$="####-##-######"
 420 PRINT "Kontonummer:    ";
 430 GOSUB 1080
 440 '
 450 '     usw.
 460 '
 470 '
 900 END
1000 '*************************************************************
1010 '
1020 '       Unterprogramm fuer maskierte Eingabe
1030 '
1040 '*************************************************************
1050 '
1060 'Anzeige Maske auf definierter Zeile / Spalte
1070 '
1080 H%=0 : H3%=SP% : GOSUB 1480 : PRINT MASK$
1090 '
1100 'Eingabe in Maske
1110 '
1120 'Ermittlung der Interpunktionszeichen in der Maske
      (Positionen stehen in Feld H%)
1130 'H2% = Laenge der Maske mit Formatzeichen
1140 'H3% = Endeposition Eingabefeld -1
1150 '
1160 H1$="" : H1%=1 : H2%=LEN(MASK$)
1170 FOR HI%=1 TO HV%
1180 H%(HI%)=INSTR(H1%,MASK$,H2$) :
     IF H%(HI%)=0 THEN H1$=H1$+RIGHT$(MASK$,H2%-H%(HI%-1)) :
     GOTO 1210
1190 H1%=H%(HI%)+1 :
     H1$=H1$+MID$(MASK$,H%(HI%-1)+1,H%(HI%)-H%(HI%-1) -1)
1200 NEXT HI%
1210 H3%=SP%+H2%-1 : GOSUB 1480 : H$=""
1220 '
```

```
1230 '
1240 'Eingabe der Ziffer mit Eingabekontrollen
1250 'in H$ steht eingegebenes Zeichen
1260 '
1270 H$=INKEY$ : IF H$="" THEN GOTO 1270
1280 IF ASC(H$)=8  THEN H3$="" : GOTO 1080
1290 IF ASC(H$)=13 THEN GOTO 1510
1300 IF ASC(H$)<48 OR ASC(H$)>57 THEN GOTO 1210
1310 IF H%=LEN(H1$) THEN GOTO 1270 ELSE H%=H%+1 :
     H1$=H1$+H$ : H1$=RIGHT$(H1$,LEN(H1$) -1) : H3$=""
1320 '
1330 'Anzeige der eingegebenen Zahl in aufbereiteter Form
      nach jeder Zifferneingabe
1340 '
1350 'Verschieben Ziffernfolge um eine Stelle nach links mit
      Beruecksichtigung Formatzeichen und Eintragen letzte
      eingetastete Ziffer an 1.Stelle von rechts
1360 '    H1$ = eingegebene Zahl ohne Formatzeichen
1370 '    H3$ = eingegebene Zahl mit Formatzeichen
1380 '
1390 '
1400 FOR HI%=1 TO HV%
1410    IF H%(HI%)=0 THEN H3$=H3$+RIGHT$(H1$,H2%-H%(HI%-1)) :
        GOTO 1440
1420    H3$=H3$+MID$(H1$,H%(HI%-1)+(2-HI%),
            H%(HI%)-H%(HI%-1) -1)+H2$
1430 NEXT HI%
1440 H3%=SP% : GOSUB 1480 : PRINT H3$ : GOTO 1210
1450 '
1460 '
1470 ' Kursorpositionierung
1480 LOCATE ZE%,H3%:     RETURN
1490 '
1500 'Ruecksprung aus Unterprogramm
1510 RETURN
```

2.2. Ausgabe auf Bildschirm (Bildschirmarbeit)

- Menuetechnik:

Die Menuetechnik erleichert den benutzergesteuerten Dialog. Ueber das Menue als Auswahluebersicht steuert der Benutzer den Ablauf des Programms.
Die Menuetechnik kann sich auf das Arbeiten innerhalb eines Programms wie auch auf das Verbinden mehrerer Programme beziehen.

Beispiel **Demonstration der Menuetechnik**

Das Programm MENUE bietet dem Benutzer verschiedene Wahlmoeglichkeiten am Bildschirm an. Dabei ist kennzeichnend:

- Auswahl einer Taetigkeit aus dem Menue: Das Menue wird am Bildschirm angezeigt, bis der Benutzer eine gueltige Auswahl getroffen hat

- Ausfuehrung dieser Taetigkeit in einem Unterprogramm

- Wiederholtes Menueangebot mit Programmende ueber das Menue.

Nach dieser Ausfuehrung wird das Menue erneut angezeigt. Abgebrochen wird der Programmlauf stets ueber das Menue bzw. ueber das Steuerprogramm, nicht aber ueber ein Unterprogramm.

```
  5 '*************************************************************
  6 '
 10 '                      Programm MENUE
 11 '              Demonstration der Menuetechnik
 12 '
 13 '*************************************************************
 14 '
 20 PRINT "Demonstration der Menuetechnik mit Wahl in DATA":
          PRINT
 30 READ N: DIM WAHL$(N)
 40 FOR I=1 TO N
 50 READ WAHL$(I)
 60 NEXT I
 70 PRINT "**** Menue - Angebot ****"
 80 FOR I=1 TO N
 90 PRINT I;" ";WAHL$(I)
100 NEXT I
110 PRINT "--------------"
120 INPUT "Ihre Auswahl ";W
130 IF W<1 OR W>N THEN
          PRINT "Zahl liegt ausserhalb.":GOTO 70
140 ON W GOSUB 500, 600, 700, 800, 900, 1000
150 IF W=6 THEN PRINT "Ende.":END
160 INPUT "Weiter mit Taste";W:CLS:GOTO 70
170 DATA 6,Einzahlung,Auszahlung,
180 DATA Konto eroeffnen, Konto loeschen
190 DATA Gesamtliste, Programmende
500 PRINT "Hier meldet sich das Unterprogramm";
          Wahl$(W):RETURN
600 PRINT "Hier meldet sich das Unterprogramm";
          Wahl$(W):RETURN
700 PRINT "Hier meldet sich das Unterprogramm";
          Wahl$(W):RETURN
800 PRINT "Hier meldet sich das Unterprogramm";
          Wahl$(W):RETURN
```

```
 900 PRINT "Hier meldet sich das Unterprogramm";
           Wahl%(W):RETURN
1000 PRINT "Hier meldet sich das Unterprogramm";
           Wahl%(W):RETURN
```

- Aufbau von Bildschirmmasken

Mit Hilfe von Bildschirmmasken kann z.B. ein Formular auf dem Bildschirm aufgebaut werden, um an den vorgegebenen Stellen die Eingaben zu realisieren. Das Programm "BILDMASK" zeigt eine Moeglichkeit des Aufbaus einer solchen Bildschirmmaske.

```
  5 '*********************************************************
  6 '
 10 '                    Programm BILDMASK
 11 '              Aufbau einer Bildschirmmaske
 12 '
 13 '*********************************************************
 14 '
 20 CLS:PRINT "Aufbau einer Bildschirmmaske"
 30 DIM K%(5)  'Fuenf Datenfelder eines Kundensatzes
 40 LOCATE 11,1
 50 PRINT "1. Numerierung der Datenfelder":
         EIN% = INPUT%(1)
 60 CLS
 70 FOR I=1 TO 5:LOCATE 5+I,3:PRINT I:NEXT I
 80 PRINT "2. Bezeichnung der Datenfelder":
       EIN% = INPUT%(1)
 90 LOCATE 6,6 : PRINT "Kundennummer:"
100 LOCATE 7,6 : PRINT "Kundenname:"
110 LOCATE 8,6 : PRINT "Kontostand:"
120 LOCATE 9,6 : PRINT "Umsatz:"
130 LOCATE 10,6: PRINT "Letzte Rechnung:"
140 PRINT "3. Eintraege in Datenfelder"
150 FOR I=1 TO 5
160 LOCATE 5+I,24
170   INPUT;EIN%
180 NEXT I
190 END
```

2.3. Ausgabe auf Drucker

Die Anweisung LPRINT nimmt normalerweise eine Zeilenbreite von 80 Zeichen an und fuegt automatisch an das Ende der Zeile Zeilenschaltung/Wagenruecklauf an. Die Breite der Zeile kann mit der Anweisung WIDTH geaendert werden:

```
WIDTH "LPT1:",n
```

Der Parameter n ist ein Ausdruck im Bereich von 0 bis 255.

Schmaldruck kann mit folgender Anweisung eingeschaltet werden:

```
LPRINT CHR$(15)
```

Die Rueckkehr in den Normaldruck erfolgt mit der Anweisung:

```
LPRINT CHR$(18)
```

Beispiel: Ausgabe von Text rechtsbuendig

```
 50 '*********************************************************
 55 '
 60 '                    Programm RBUEND
 70 '              Rechtbuendige Ausgabe von Text
 80 '
100 '*********************************************************
105 '
110 PRINT "Text rechtsbuendig ausgeben"
120 INPUT "1.Textzeile ";T1$
130 INPUT "2.Textzeile ";T2$
140 INPUT "Rechte Begrenzung ";R
150 B$ = SPACE$(R)
160 LET T1$ = RIGHT$(B$+T1$,R)
170 LET T2$ = RIGHT$(B$+T2$,R)
180 LPRINT T1$:LPRINT T2$
190 END
```

2.4. Spezielle Steuerzeichenfolgen zur Bildschirm- und Tastatursteuerung fuer BASIC-Compiler

Die unten aufgefuehrten ESCAPE - Steuerzeichenfolgen koennen in Quellprogrammen angewendet werden, die mit dem BASIC-Compiler uebersetzt werden. Randbedingung fuer die Uebersetzung ist, dass in der Systemkonfigurationsdatei **CONFIG.SYS** der erweiterte Bildschirm- und Tastaturgeraetetreiber **ANSI.SYS** eingebunden ist. Das Einbinden erfolgt durch die Anweisung:

```
device = [d][path]ansi.sys
```

Format der Steuerfolgen fuer Kursor

```
ESC [parameter Kommando
 |       |         |
 |       |         |
 |       |         +---->  spezifizierter Zeichenkettenwert
 |       #     ---------->  spezifiziert einen numerischen
 |                          Parameter.
1BH                         Ist kein numerischer Wert oder 0
                            spezifiziert, dann wird der
                            Standardwert benutzt.
```

Tabelle der Steuerfolgen fuer Kursor

Kdo	Wirkung	Bemerkung
ESC[#A	Kursor nach oben	# gibt Anzahl der Zeilen an. Wird ignoriert, wenn Kursor bereits in 1. Zeile ist (Standard fuer # ist 1).
ESC[#B	Kursor nach unten	analog ESC[#A wird ignoriert, wenn Kursor bereits in Fusszeile
ESC[#;#H ESC[#;#f	Kursor positionieren	Par.: Zeile, Spalte ohne Par.: Home-Position
ESC[#C	Kursor nach rechts	# gibt Anzahl der Spalten an. Keine Aenderung der Zeile (Standard fuer # ist 1). Wird ignoriert, wenn Kursor am rechten Zeilenende.
ESC[#D	Kursor nach links	analog ESC[#C
ESC[2J	Loeschen Bildschirm	Bildschirm loeschen und Kursor in Home-Position
ESC[K	Loeschen Zeile	Ab Kursorposition bis Zeilenende, einschliesslich Kursorposition
ESC[S	Sichern Kursorposition (Save)	Sichern der aktuellen Kursorposition. Diese Position kann mit Restore wiederhergestellt werden
ESC[U	Wiederherstellen Kursorposition (Restore)	Entsprechend dem Stand bei Kdo Save

3.___Arbeit_mit_Feldern/Bereichen

3.1._Ueberblick_und_Erklaerung_der_Begriffe

Ein Feld ist eine Menge von Werten (Elementen), die unter einem gemeinsamen Namen abgespeichert sind.

Ein Feld M z.B. hat 5 Elemente, wobei im Element M (3) die Zahl 77 gespeichert ist.

```
    M(0)  M(1)  M(2)  M(3)  M(4)
   +-------------------------------+
   | 12  |  9  |  1  | 77  | 2.5  |
   +-------------------------------+
```

Es gibt drei Grundtypen von Feldern

```
                    |
              +-----------|-----------+
              |           |           |
Typ:          Integer     Real        Zeichenkette
Bezeichner:   M%          M           M$
```

eindimensional:

```
    DIM M%(4)      DIM M(4)        DIM M$(4)

    121  M%(0)     65.01  M(0)     ZANGE   M$(0)
    105  M%(1)      3.25  M(1)     HAMMER  M$(1)
    199  M%(2)     12.50  M(2)     MEISEL  M$(2)
     50  M%(3)      7.752 M(3)     KELLE   M$(3)
   2508  M%(4)     99.00  M(4)     BOHRER  M$(4)
```

zweidimensional:

```
    DIM M%(3,2)    DIM M(3,2)          DIM M$(3,2)

     1  2  3        1.4  2.5  1.1      HANS   OTTO   EMIL
     9  9  9       17.1  0.7  1.0      EVA    KLAUS  CARLA
    34  5  9        0.3  0.4  0.5      ERNST  MARIA  JULIA
     1 11  7       11.1 12.7  0.9      MAX    LENA   ANNA
```

3.2._Eindimensionale_Felder

Ein eindimensionales Feld kann man sich waagerecht als Zeile oder senkrecht als Spalte angeordnet vorstellen.

Das Programm **LAGREGAL** veranschaulicht diese Datenstruktur. Mit **DIM R(7)** wird ein Feld mit 8 Elementen vereinbart. Das Element 0 bleibt unberuecksichtigt (man reserviert es meist fuer besondere Eintragungen).

Beispiel: **LAGREGAL**

```
 10 '*****************************************************
 20 '
 30 '          Programm LAGREGAL
 40 '       (Bsp. Eindimensionales Feld)
 50 '
 60 '*****************************************************
 70 '
 80 DIM R(7): 'FELD(1..7) als Regal
 90 'I:        Lauf- bzw. Indexvariable
100 'M:        Summe der Faecher'
110 '
120 '*****************************************************
130 '
140 PRINT "Eindimensionales Feld (Lagerregal)"
150 PRINT: PRINT "Eingabe in Regalfaecher: "
160 FOR I=1 TO 7
170   PRINT "Menge fuer Fach";I;
180   INPUT R(I)
190 NEXT I
200 '
210 PRINT:PRINT "Fach:          Menge:"
220 FOR I=1 TO 7
230   PRINT I,R(I): LET M=M+R(I)
240 NEXT I
520 PRINT "SUMME: ",M : END
```

Dimensionierung eines Feldes

```
                |
     +----------------------+
     |                      |
  dynamisch              statisch

100 INPUT "Anzahl:";A     100 DIM D%(40)
110 DIM D%(A)

Feld D% mit A             Feld D% mit konstant
Elementen                 40 Elementen
(variable Anzahl
durch Tastatureingabe)
```

Das Programm **VOKABEL** erlaeutert die Methode der parallelen Felder. Die Anordnung der Elemente der beiden Felder D%() und F%() kann man durch 2 nebeneinanderstehende Spalten darstellen. Dabei gehoeren die waagerechten Elemente jeweils zusammen.

Zweck: tabellenfoermige Abspeicherung von Daten

	Feld_D$		Feld_F$
(1)	Mann	(1)	homme
(2)	Frau	(2)	femme
(3)	Kind	(3)	enfant
:		:	
(n)	Mond	(n)	lune

gleiche Anzahl von n Elementen

Beispiel: VOKABEL

```
 10 '*****************************************************
 20 '
 30 '     '      Programm Vokabel
 40 '
 50 '*****************************************************
 60 '
 70 PRINT "Uebung Franzoesisch - Deutsch"
 80 INPUT "Anzahl der Vokabeln";A
 90 DIM D$(A): 'ZK-Feld fuer Deutsch(D)
100 DIM F$(A): 'ZK-Feld fuer Franz. (F)
110 'A$:        Jeweilige Antwort
120 '
130 '*****************************************************
140 '
150 PRINT "Paarweise eintippen: D, F"
160 FOR I=1 TO A
170   INPUT D$(I), F$(I)
180 NEXT I
190 PRINT :PRINT "Beginn der Uebung: "
200 FOR I=1 TO A
210   PRINT D$(I);" heisst "; ; INPUT A$
220   IF A$=F$(I)
        THEN PRINT "Gut."
        ELSE PRINT "Falsch.";D$(I);" heisst ";F$(I)
230 NEXT I
240 PRINT "Ende": END
```

3.3. Zweidimensionale Felder

Die Elemente eines zweidimensionalen Feldes sind in Zeilen und Spalten angeordnet. Am Beispiel **ABTABELL** soll diese Datenstruktur naeher betrachtet werden.
Das Feld **R** dient der Speicherung der Absatzmengen von 5 Kunden (≙ Zeilen 1 bis 5) in 4 Quartalen (≙ Spalten 1 bis 4).
Die Tastatureingabe der 5 * 4 = 20 Absatzmengen vollzieht sich ueber 2 geschachtelte Zaehlerschleifen (typisch fuer zweidimensionale Felder).

```
210 FOR I=1 TO Z          Aeussere Schleife: Kunden 1..5
230    FOR J=1 TO S       Innere Schleife: Quartale 1..4
250       INPUT R(I,J)    Eingabe nach Element R(I,J)
260    NEXT J             Innere Schleife beenden
270 NEXT I                Aeussere Schleife beenden
```

Beispiel: **ABTABELL**

```
 10 '*********************************************************
 20 '
 30 '                Programm ABTABELL
 40 ' (Absatztabelle Kunde / Quartal als 2-dimensionales Feld)
 50 '
 60 '*********************************************************
 70 '
 80 PRINT "Absatztabelle Kunde/Quartal"
 90 INPUT "Anzahl der Zeilen (waagerecht)";Z
100 INPUT "Anzahl der Spalten (senkrecht)";S
110 DIM R(Z,S) 'Feld dynamisch dimensioniert
120 'AR:        jeweilige Antwort
130 '
140 '*********************************************************
150 '
160 PRINT :PRINT "Eingabe zeilenweise "
170 FOR I=1 TO Z
180  PRINT : PRINT "Naechste Zeile, naechster Kunde:" : PRINT
190   FOR J=1 TO S
200     PRINT " Kunde ";I;" Quartal ";J;
210     INPUT R(I,J)
220   NEXT J
230 NEXT I
240 FOR I=1 TO Z          'zeilenweise summieren nach Spalte 0
250   FOR J=1 TO S
260     LET R(I,0)=R(I,0)+R(I,J)
270   NEXT J
280 NEXT I
290 FOR I=1 TO Z          'Gesamtsumme nach R(0,0) bringen
300   LET R(0,0)=R(0,0)+R(I,0)
310 NEXT I
320 FOR J=1 TO S          'spaltenweise summieren nach Zeile 0
330   FOR I=1 TO Z
340     LET R(0,J)=R(0,J)+R(I,J)
```

```
350   NEXT I
360 NEXT J
370 PRINT :PRINT "Uebersicht:"
380 PRINT
390 FOR I=0 TO Z
400   FOR J=0 TO S
410     PRINT USING "##### ";R(I,J)
420   NEXT J
430   PRINT
440 NEXT I
450 PRINT "Ende" : END
```

Uebersicht:

	Quartal 1	Quartal 2	Quartal 3	Quartal 4	
1500	150	300	450	600	
100	10	20	30	40	Kunde1
200	20	40	60	80	Kunde2
300	30	60	90	120	Kunde3
400	40	80	120	160	Kunde4
500	50	100	150	200	Kunde5

Die Elemente mit Index 0 werden haeufig zur Ablage besonderer Werte verwendet.

In diesem Beispiel werden in Zeile 0 die Quartals-Summen (Spaltensummen) gespeichert. Die Spalte 0 enthaelt die Kundenabsatzmengen, also 5 Zeilensummen.

Element R(0,0) beinhaltet die Gesamtjahresabsatzmenge.

4. Programmueberlagerung

BASIC stellt die Anweisungen **MERGE**, **CHAIN** und **COMMON** bereit, um Programme zu einem Programmsystem zu verbinden.

4.1. Einmischen von Programmen oder Programmteilen mit dem Kommando MERGE

Mit dem Kommando **MERGE** wird ein im Hauptspeicher abgelegtes Programm mit einem Programm ueberlagert, das sich auf der Diskette befindet.
Dieses Ueberlagern geschieht folgendermassen:

- Zeilen mit gleichen Zeilennummern werden ueberschrieben, d.h. die Zeilen des Programmes auf der Diskette ueberschreiben die im Speicher stehenden Zeilen.

- Zeilen mit ungleichen Zeilennummern werden hinzugefuegt.

Das Kommando **MERGE** kann unter anderem dazu benutzt werden, um oft benoetigte Routinen oder Programmteile (z.B. zur Druckersteuerung, Bildschirmgestaltung) in neue Programme einzubinden. Damit kann unnoetiger Erfassungsaufwand vermieden werden. Zu beachten ist, dass die Zeilennummern dieser Routinen entsprechend hoch liegen, damit keine Zeilen des rufenden Programmes ueberschrieben werden.

Folgende Arbeitsschritte sollten ausgefuehrt werden:

1. Die Routine wird mit dem Kommando **RENUM** neu numeriert (Verwendung hoher Zeilennummern).

2. Mit dem Kommando **SAVE** und der Option **A** wird die Routine als ASCII-Datei bzw. Textdatei abgespeichert.

3. Laden des rufenden Programmes. Mit Hilfe des Kommandos **MERGE** wird die Routine an das rufende Programm angefuegt und kann als Unterprogramm eingebunden werden.

4. Anschliessend kann das gesamte Programm mit Hilfe des Kommandos **SAVE** abgespeichert werden.

Die Variablen, die im rufenden Programm erzeugt werden, koennen auch in den eingemischten Programmteilen verwendet werden, sie behalten ihren Inhalt.

Beispiel:

Die Berechnung einer Formel wird als Routine zur Verfuegung gestellt und kann in jedes Programm eingebunden werden.

rufendes Programm: PROGALT

```
10 '*********************************************************
11 '
12 '          Programm PROGALT
13 '          Rufendes Programm
14 '
15 '*********************************************************
20 PRINT "Beginn rufendes Programm":PRINT
30 INPUT "Eingabe ganze Zahl:",I%
40 INPUT "Eingabe Zahl mit zwei Kommastellen:";D
50 GOSUB 1000
60 PRINT "Rueckkehr aus der Routine":PRINT
70 PRINT "Ergebnis = ";C:PRINT
80 PRINT "Ende des Programmes"
90 END
SAVE "PROGALT"
OK
```

Routine: MODUL

```
1000 PRINT: PRINT "Beginn der Routine"
1010 PRINT "I% = "; I%, "D = "; D:PRINT
1020 C = (2*(D+I%)+(D-I%)^3)/5
1030 RETURN
SAVE "MODUL",A
OK
```

Verbinden der beiden Programme:

```
LOAD "PROGALT"
OK
MERGE "MODUL"
OK
SAVE "PROGNEU"
OK
RUN

Beginn rufendes Programm
Eingabe ganze Zahl: 100
Eingabe Zahl mit zwei Kommastellen: 250.25

Beginn der Routine
I% = 100          D = 250.25
Rueckkehr aus der Routine
Ergebnis = 678520.7
Ende des Programmes
OK
```

4.2. Verketten von Programmen mit CHAIN

Mittels der Anweisung **CHAIN** kann ein Programm waehrend des Programmlaufes ein anderes Programm von der Diskette in den Hauptspeicher laden und dieses kann ausgefuehrt werden.
Dabei wird das rufende Programm geloescht. Ebenfalls werden alle Variablen geloescht, wenn in der Anweisung CHAIN nicht angegeben ist, dass alle Variablen an das gerufene Programm uebergeben werden sollen.

Sinnvoll ist diese Programmiertechnik z.B. dann, wenn von einem Menueprogramm verschiedene eigenstaendige Teilprogramme aufgerufen werden und nach Abarbeitung dieser Teilprogramme die Steuerung wieder an das Menueprogramm uebergeben wird.

Mit Hilfe des Parameters **ALL** in der Anweisung CHAIN koennen alle Variablen an das Teilprogramm uebergeben werden.

Beispiel:

Ausgehend von einem Menueprogramm werden verschiedene Teilprogramme aufgerufen. Nach Abarbeitung dieser Teilprogramme erfolgt der Aufruf des Menueprogrammes, um weitere Teilprogramme abarbeiten zu koennen.

Menueprogramm: MENUE

```
  5 '*********************************************************
  6 '
  7 '            Programm MENUE
  8 'Menueprogramm zur Auswahl der verschiedenen Programmzweige
  9 '
 10 '*********************************************************
 12 COLOR 7,0
 15 CLS
 16 LOCATE 10,1
 20 PRINT "Erstellen einer Datei          : E"
 30 PRINT "Druck der erfassten Datei      : D"
 50 PRINT "Verlassen des Programmes       : A"
 60 PRINT :INPUT "Bitte waehlen Sie ",W$
 70 IF W$="E" OR W$="e" THEN CHAIN "ERFAS"
 80 IF W$="D" OR W$="d" THEN CHAIN "DRUCK"
 95 CLS
100 END
```

Erfassungsprogramm: ERFAS

```
  5 '*********************************************************
  6 '
  7 '            Programm ERFAS
  8 '         Programm zum Erstellen einer Datei
  9 '
 10 '*********************************************************
 15 CLS
 20 PRINT "Hier ist das Teilprogramm ERFAS"
 25 OPEN "MITAR" AS #1
 26 FIELD #1, 4 AS N$,30 AS NAM$, 30 AS A$
 30 LOCATE 10,1
 40 PRINT "Eingabe der Nummer     :"
 50 PRINT "Eingabe des Namens     :"
 60 PRINT "Eingabe der Adresse    :"
 70 LOCATE 10,30:COLOR 2,0:INPUT "",NR%
 75 IF NR%=999 THEN 145
 80 LOCATE 11,30:LINE INPUT "",NA$
 90 LOCATE 12,30:LINE INPUT "",AD$:COLOR 7,0
 95 LSET N$=MKI$(NR%)
 96 LSET NAM$=NA$
 97 LSET A$=AD$
100 PUT #1,NR%
110 LOCATE 10,30:PRINT SPACE$(40)
120 LOCATE 11,30:PRINT SPACE$(40)
130 LOCATE 12,30:PRINT SPACE$(40)
```

```
140 GOTO 70
145 COLOR 7,0
146 CLOSE
150 CHAIN "MENUE"
160 END
```

Druckprogramm: DRUCK

```
 10 '*********************************************************
 11 '
 12 '           Programm DRUCK
 13 '     Programm zum Drucken/Anzeigen der erstellten Datei
 14 '
 15 '*********************************************************
 20 CLS
 30 PRINT "Hier ist das Teilprogramm DRUCK"
 40 OPEN "MITARB" AS #1
 50 FIELD #1,4 AS N$,30 AS NAM$,30 AS A$
 55 LOCATE 10,1:PRINT "Bitte geben Sie die Nummer ein:  "
 56 LOCATE 12,1:PRINT "Name      :"
 57 LOCATE 13,1:PRINT "Adresse   :"
 67 LOCATE 10,35
 68 COLOR 14,0
 70 INPUT "",NR%
 75 LOCATE 12,35:PRINT SPACE$(40)
 76 LOCATE 13,35:PRINT SPACE$(40)
 80 IF NR%=999 THEN 140
 90 GET #1,NR%
 91 IF EOF(1) THEN LOCATE 15,1:PRINT "Dateiende":GOTO 140
 96 NA$=NAM$:AD$=A$
100 LOCATE 12,35:PRINT NA$
110 LOCATE 13,35:PRINT AD$
120 LOCATE 10,35:PRINT SPACE$(10)
130 GOTO 67
140 CLOSE
145 COLOR 7,0
150 CHAIN "MENUE"
160 END
```

4.3. Uebergabe von Variablen

Die Anweisung **CHAIN** mit dem Parameter **ALL** erlaubt es, alle Variablen vom rufenden Programm an das gerufene Programm zu uebergeben, um dort mit diesen Werten weiterzuarbeiten.
Sollen nicht alle Variablen uebergeben werden, oder soll das Programm spaeter mit dem Compiler uebersetzt werden, muss die Anweisung **COMMON** verwendet werden. Mit dieser Anweisung werden gemeinsame Variablen vereinbart, die dann auch vom rufenden Programm an das gerufene Programm uebergeben werden.

Beispiel:

Das Gesamtprogramm besteht aus zwei Teilprogrammen. Das Teilprogramm 1 ruft das Teilprogramm 2 auf und uebergibt die Variablen A$, B$, Tel%.

Teilprogramm 1 TEILPRO1

```
10 '*********************************************************
11 '
12 '           Programm TEILPRO1
13 '
14 '*********************************************************
15 COLOR 14,1
20 PRINT "Hier meldet sich das Teilprogramm1";
25 COLOR 15,0
26 PRINT : PRINT
30 COMMON A$,B$, Tel%
40 INPUT "Betrieb :",B$
50 INPUT "Adresse :",A$
60 INPUT "Telefon :",Tel%
70 CHAIN "TEILPRO2"
80 END
```

Teilprogramm 2 TEILPRO2

```
 5 '*********************************************************
 6 '
 7 '             Programm TEILPRO2
 8 '      Teilprogramm 2 mit den Variablen aus Teilprogramm 1
 9 '
10 '*********************************************************
12 PRINT
15 COLOR 10,4
20 PRINT "Hier meldet sich das Teilprogramm 2";
25 COLOR 15,0
26 PRINT:PRINT
30 PRINT "Betrieb   :";B$
40 PRINT "Adresse   :";A$
50 PRINT "Telefon   :";Tel%
55 PRINT
60 CHAIN "TEILPRO1"
70 END
```

5. Fehlerbehandlung

BASIC definiert 78 Fehler. Jeder Fehler enthaelt eine Fehlernummer (siehe Sprachbeschreibung und Bedienungsanleitung BASIC-Interpreter Anlage C). Tritt bei der Ausfuehrung einer Programmzeile ein Fehler auf, gibt BASIC eine Fehlermeldung aus. In Abhaengigkeit vom Fehler wird die Programmabarbeitung fortgesetzt oder in die Befehlsebene zurueckgekehrt.

BASIC stellt in der Variablen **ERR** den jeweiligen Fehlercode und in der Variablen **ERL** die fehlerverursachende Zeile zur Verfuegung.

5.1. Fehlerbehandlung mit ON ERROR

Um bei Fehlern das Abbrechen des Programmes zu vermeiden und um den Fehler innerhalb des Programmes zu behandeln, wird am Anfang des Programmes die Anweisung **ON ERROR GOTO Zeile** verwendet.
Nach Ausfuehrung dieser Anweisung wird bei Auftreten eines Fehlers immer zu **Zeile** verzweigt (Zeile - spezifiziert den Beginn der Fehlerbehandlungsroutine).
In der Fehlerbehandlungsroutine kann durch Abfrage der Variablen **ERR** die Fehlerart bzw. durch Abfrage der Variablen **ERL** die fehlerverursachende Zeile ermittelt werden.

Die Anweisung **RESUME** muss die Fehlerbehandlungsroutine abschliessen.

Eine Fehlerbehandlung laeuft allgemein in folgenden Schritten ab:

1. Fehlerbehandlung aktivieren (eroeffnen):
   ON ERROR GOTO 1000

2. Fehlerbehandlungroutine ab Zeile 1000 ...
   Fehlercode in **ERR** abfragen.
   Fehlerzeile in **ERL** abfragen.
   Fehlerhinweise ausgeben, Fehlerbeseitigung durchfuehren.

3. Programmablauf fortsetzen mit **RESUME**
   (zu beachten sind die verschiedenen Moeglichkeiten der Anweisung RESUME)

4. Fehlerbehandlung inaktivieren (schliessen)
   ON ERROR GOTO 0

Beispiel:

```
  5 '*********************************************************
  6 '
  7 '            Programm FEHLER
  8 '
  9 '*********************************************************
 10 ON ERROR GOTO 500
 20 A="100"
 30 ...
  :
  :
  :
100 ON ERROR GOTO 0
110 END
120 '
500 '
505 '          Beginn der Fehlerbehandlungsroutine
506 '
510 IF ERR<>13 THEN 600
520 PRINT  "Fehler:    Zeichenkette  einer  numerischen
          Variablen zugewiesen"
530 RESUME NEXT
600 PRINT "Fehlercode:";ERR
610 PRINT "Zeilennummer mit Fehlerursache:";ERL
620 PRINT "Weiter mit Taste:":E$=INPUT$(1)
630 RESUME 100
```

Wird dieses Programmbeispiel ohne Fehlerbehandlung abgearbeitet, so wird bei Ausfuehrung der Zeile 20 die Fehlermeldung "Type mismatch in 20" auf dem Bildschirm angezeigt und das Programm sofort abgebrochen.

Wurde jedoch eine Fehlerbehandlung eingebaut (siehe Beispiel), so wird nach Ausfuehrung der Anweisung ON ERROR GOTO beim Auftreten eines Fehlers nach Zeile 500 verzweigt (Beginn der Fehlerbehandlungsroutine).
Ist der Fehler mit dem Fehlercode 13 aufgetreten, so wird eine Mitteilung ausgegeben und anschliessend mit der naechsten Zeile nach der Fehlerzeile das Programm fortgesetzt.
Wurde ein Fehlercode ungleich 13 festgestellt, so werden dieser und die fehlerverursachende Zeile ausgegeben. Anschliessend wird das Programm mit der Zeile 100 fortgesetzt und hier die Fehlerbehandlung geschlossen, bevor das Programm beendet wird.

5.2. Erzeugen von Fehlercodes mit ERROR

Mit der Anweisung ERROR koennen BASIC-Fehler simuliert werden, indem in der ERROR-Anweisung eine Fehlernummer angegeben wird, die von BASIC belegt ist (Fehlernummern von 1 bis 76). Weiterhin ist es moeglich, dass eigene Fehlercodes definiert werden, indem in der ERROR-Anweisung eine Fehlernummer groesser 76 verwendet wird.

Beispiel:

In diesem Beispiel wird gezeigt, wie alle von der Tastatur eingegebenen Zahlen, die groesser 999 sind, abgewiesen werden. Dafuer wird der Fehlercode 100 verwendet.

```
  4 '*******************************************************
  5 '
  6 '            Programm FEHLER1
  7 '            eigene Fehlerdefinition
  8 '
  9 '*******************************************************
 10 ON ERROR GOTO 500
 20 WHILE EIN<>-999
 30 INPUT "Zahl eingeben (-999 = Ende)  ",EIN
 40 IF EIN>999 THEN ERROR 100
 50 WEND
 60 END
 70 '
500 '
501 '          Fehlerbehandlungsroutine
502 '
510 IF ERR=100 THEN PRINT "Zahl unter 999 eingeben"
520 IF ERL= 40 THEN RESUME 50
```

6. Suchen, Sortieren, Mischen und Gruppieren von Daten

Es gibt vier Hilfsverfahren der Dateiverarbeitung:

- Suchen: sequentielles Suchen, binaeres Suchen
- Sortieren:
  - unmittelbares Umsortieren (Zahlen selbst)
  - paarweiser Austausch (Bubble Sort)
  - Shell-Sort (aehnlich Bubble-Sort)
  - ueber Zeiger
- Mischen: Datei unter Beruecksichtigung der Sortierfolge zusammenmischen
- Gruppieren: Zusammenfassen bzw. Verdichten von Daten Bilden von Gruppensummen

6.1. Suchverfahren

6.1.1. Sequentielles Suchen

Das sequentielle Suchverfahren besteht darin, die Datei Satz fuer Satz in der Reihenfolge der Speicherung zu durchsuchen. Das Suchen muss immer bis zum Dateiende erfolgen.

Dieses Suchverfahren setzt einen vorsortierten Datenbestand voraus. Das Suchen kann abgebrochen werden, sobald der Suchbegriff gefunden wird.

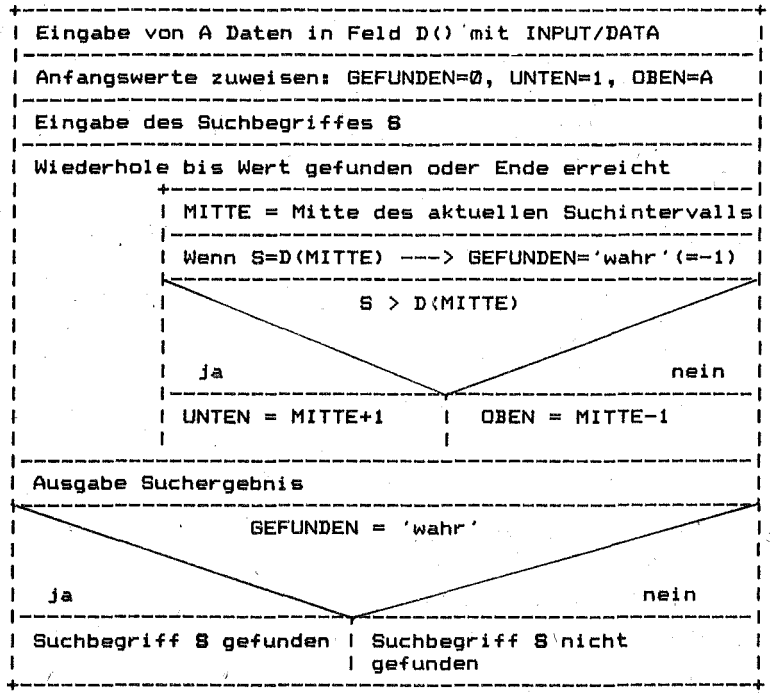
Beispiel: SUCHEN

```
 10 '*********************************************************
 20 '
 30 '                    Programm SUCHEN
 40 '                    (sequentiell)
 50 '
 60 '*********************************************************
 70 '
 80 CLS :PRINT "Sequentielles Suchen im Bereich"
 90 DATA 10,114,116,119,125,150,160,189,202,215,240
100 READ A            'Anzahl der Daten
110 DIM D(A+1)        'Dynamisch dimensionieren
120 FOR Z=1 TO A:READ D(Z):NEXT Z
130 '
140 '*********************************************************
150 '
160 PRINT
170 PRINT "0 ENDE."
180 PRINT "1 SEQUENTIELL VORWAERTS SUCHEN"
190 PRINT
200 INPUT "WAHL 0/1";W$
210 LET W=VAL(W$)
220 IF W=0 THEN PRINT "ENDE" : END
230 FOR Z=1 TO A
240    PRINT D(Z);
250 NEXT Z
260 PRINT :INPUT "Welcher Suchbegriff ";SUCH
270 ON W GOTO 300
280 '
300 LET Z=0
310 LET D(A+1)=SUCH
320 WHILE D(Z)<>SUCH
330 LET Z=Z+1:PRINT Z;
340 WEND :PRINT
350 IF Z=A+1 THEN PRINT "..Nicht gefunden"
             ELSE PRINT SUCH; "an";Z;".Stelle."
360 GOTO 170
```

6.1.2. Binaeres Suchen

Bei diesem Suchvorgang muessen die Daten sortiert auf einem Direktzugriffsspeicher vorliegen.
Die Datenmenge wird stets halbiert, um die Position des Suchbegriffs einzugrenzen.
Im folgenden Beispiel soll in einem Feld von 7 Zahlen gesucht werden (45, 76, 78, 80, 90, 95, 100):
Um den Wert 90 zu suchen, wird zunaechst der mittlere Wert angenommen (immer ganzzahlig). Der Vergleich 80 < 90 zeigt, dass in der oberen Haelfte 90 bis 100 weiterzusuchen ist. Man nimmt wieder die Mitte und der Vergleich 95 > 90 verweist auf die untere Haelfte.

Struktogramm zum Programm SUCHBIN (binaeres Suchen)



Beispiel: SUCHBIN

```
  10 '*******************************************************
  20 '      *
  30 '                          Programm SUCHBIN
  40 '                (Binaeres Suchen als schnelle Suchmethode)
  50 '
  60 '*******************************************************
  70 '
  80 'A:              Anzahl der Daten
  90 'D(A):           Feld mit A Daten als Suchgegenstand
 100  DIM D$(80):     'Zeichenkettenfeld als Suchgegenstand
 110 'Gefunden, Nichtgefunden: -1(wahr) bzw. 0(unwahr)
 120 'Unten, Mitte
 130 'Haelfte:        Suchhaelfte
 140 'S,S$:           Suchbegriffe numerisch bzw.Text
 150 '
 160 '*******************************************************
 170 '
 180 PRINT "Binaeres Suchen als schnelle Suchmethode."
 190 PRINT "0     Ende"
 200 PRINT "1     Suchen in einem numerischen Feld (Anzahl variabel)
 210 PRINT "2     Suchen in einem Zk-Feld(Anzahl fest unter DATA)
 220 PRINT "3     Suchen in einem Zk-Feld (74 Zeichen)"
 230 INPUT "Wahl 0-3";W$:LET W=VAL(W$)
 240 IF W=0 THEN PRINT "ENDE":END
 250 ON W GOSUB 1000,2000,3000
 260 IF INKEY$="" THEN 260
 270 CLS:GOTO 190
 280 '
1000 '*******************************************************
1010 '
1020 '               Unterprogramm SUCHBIN1
1030 '               (Suche in numerischem Feld)
1040 '
1050 '*******************************************************
1060 '
1070 INPUT "Anzahl der Daten";A: DIM D(A)
1080 PRINT A;"Daten aufsteigend sortiert eingeben!"
1090 FOR I=1 TO A:INPUT D(I):NEXT I
1100 LET GEFUNDEN =0:LET UNTEN =1:LET OBEN = A
1110 INPUT "Welchen Wert suchen?";S
1120 PRINT :PRINT "Suchprotokoll zum Halbieren:"
1130 WHILE (UNTEN<=OBEN)AND (GEFUNDEN =0)
1140    LET MITTE=INT((UNTEN+OBEN)/2)
1150    PRINT "Unten:";UNTEN;",Mitte";MITTE;", Oben";OBEN
1160    LET GEFUNDEN=S=D(MITTE)     '-1(wahr) oder 0(unwahr)
1170    IF S>D(MITTE) THEN LET UNTEN=MITTE+1
                      ELSE LET OBEN=MITTE-1 UNTEN ODER OBEN SUCHEN
1180 WEND
1190 PRINT :PRINT "Suchergebnis: ";
1200 IF GEFUNDEN THEN PRINT S; "gefunden."
                 ELSE PRINT S; "nicht gefunden."
1210 RETURN
```

```
2000 '*****************************************************
2010 '
2020 '                  Unterprogramm SUCHBIN2
2030 '                  (Suche in Zk-Feld)
2040 '
2050 '*****************************************************
2060 '
2070 DATA AKELEI,CLEMATIS,FLIEDER,JASMIN,MARGERITE,MOHN,NELKE
2075 DATA ROSE,TULPE,ZZZZZ
2080 RESTORE:FOR I=1 TO 10:READ D$(I):PRINT D$(I);" ";:NEXT I
2090 PRINT :INPUT "Welchen Begriff suchen";S$
2100 LET MITTE=5:LET HAELFTE =MITTE:LET GEFUNDEN=0:LET NICHTGEFUNDEN=0
2110 WHILE NOT (GEFUNDEN OR NICHTGEFUNDEN)
2120   PRINT "Mitte:";MITTE;",Haelfte:";HAELFTE
2130   IF S$=D$(MITTE) THEN LET GEFUNDEN =-1:GOTO 2170
2140   IF HAELFTE=1 THEN LET NICHTGEFUNDEN=-1:GOTO 2170
2150     LET HAELFTE =INT(HAELFTE/2 + .5)
2160     IF S$>D$(MITTE) THEN LET MITTE = MITTE+HAELFTE
                         ELSE LET MITTE = MITTE-HAELFTE
2170 WEND
2180 IF GEFUNDEN THEN PRINT S$;" steht an Stelle ";MITTE
                 ELSE PRINT S$;" nicht gefunden. "
2190 RETURN
'
3000 '*****************************************************
3010 '
3020 '                   Unterprogramm SUCHBIN3
3030 '                   (Suchen in Zk-Feld 74 Zeichen)
3040 '
3050 '*****************************************************
3055 '
3060 LET J=0
3070 RESTORE:FOR I=49 TO 122
3080   LET J=J+1
3090   LET D$(J)=CHR$(I):PRINT D$(J);
3100 NEXT I:PRINT
3110 INPUT "Welches einzelne Zeichen suchen";S$
3120 LET MITTE=37:LET HAELFTE=MITTE:LET GEFUNDEN=0:LET NICHTGEFUNDEN=0
3130 WHILE NOT (GEFUNDEN OR NICHTGEFUNDEN)
3140   PRINT "Mitte:";MITTE;", Haelfte:";HAELFTE
3150   IF S$=D$(MITTE) THEN LET GEFUNDEN=-1:GOTO 3190
3160   IF HAELFTE=1 THEN LET NICHTGEFUNDEN=-1:GOTO 3190
3170     LET HAELFTE=INT(HAELFTE/2 + .5)
3180     IF S$>D$(MITTE) THEN LET MITTE=MITTE+HAELFTE
                         ELSE LET MITTE=MITTE-HAELFTE
3190 WEND
3200 IF GEFUNDEN THEN PRINT S$;" steht an Stelle ";MITTE
                 ELSE PRINT S$;" nicht gefunden. "
3210 RETURN
```

6.2. Sortieren

Sortieren heisst:

- Daten im internen Speicher oder mit Ein-/Auslagern von/zu einem externen Speicher sortieren
- Daten als Zahlen oder als Text sortieren
- Daten selbst sortieren oder nur deren Adressen bzw. Speicherplaetze

6.2.1. Zahlen unmittelbar sortieren

Unmittelbar sortieren heisst, die zu sortierenden Zahlen selbst umzuordnen und nicht ihre Plaetze.
Im nachfolgenden Beispiel werden die verschiedenen Sortierverfahren dargestellt.

Sortierverfahren:

- Austausch nach Auswahl:

Sollen z.B. Zahlen im Feld D() sortiert werden, ergibt sich folgender Ablauf:

Minimum in D() suchen und in STELLEMIN speichern

D(I) mit D(STELLEMIN) austauschen

Wiederholung ab 1. Schritt, aber jetzt mit D(I+1) beginnen

Das Tauschen von D(I) mit D(STELLEMIN) vollzieht sich ueber die Anweisung

SWAP D(I),D(STELLEMIN)

Das Sortieren erfolgt ueber zwei geschachtelte Zaehlerschleifen.

- Paarweiser Austausch bzw. Bubble-Sort:

Die ersten zwei Zahlen im Feld werden verglichen und falls sie nicht in der richtigen Sortierfolge sind, durch die Anweisung **SWAP D(I),D(J)** ausgetauscht. Dann werden die beiden folgenden Zahlen verglichen usw..
Auf diese Art wird bei jedem Schleifendurchlauf eine am Anfang des Feldes stehende grosse Zahl ans Ende verschoben. Dieses Verfahren arbeitet recht umstaendlich, da es nichts aus vorangegangenen Sortierschritten lernt.

- Lineare Auswahl:

Mit **LET MIN=99999** wird eine grosse Zahl als vorlaeufiges Minimum angenommen. Dann wird - durch Vergleich mit 99999 - das Minimum ermittelt und an die erste Stelle von S gespeichert.

Durch **LET D(STELLEMIN) = 99999** ersetzt man das Minimum 101 durch 99999, um diese Position in D fuer den nachfolgenden Schleifendurchlauf zu sperren.

- Shell-Sort:

Beim Shell-Sort vergleicht und vertauscht man wie beim Bubble-Sort Elemente paarweise, nur liegen diese Elemente nicht unbedingt direkt nebeneinander.
Elemente, die ...16,8,4,2,1 Stellen voneinander entfernt liegen, werden zusammengefasst und wie beim Bubble-Sort verglichen. Die Anfangsschrittweite wird in diesem Beispiel abhaengig von der Elementeanzahl eingestellt.

Der Vorteil des Shell-Sort gegenueber dem Bubble-Sort liegt darin, dass das Programm bei jedem Durchlauf aus jeweils vorhergehenden Durchlaeufen gelernt hat.

Beispiel: SORTNUM

```
100  '*******************************************************
110  '
120  '                 Programm SORTNUM
130  '          (numerische Daten sortieren)
140  '
150  '*******************************************************
160  '
170 PRINT "Sortieren nach vier grundlegenden Verfahren"
180 PRINT "(Numerische Daten selbst sortieren, nicht Zeiger)."
185 '
190 'STELLEMIN,MIN:  Stelle mit vorlaeufigem Minimum, Minimum
200 DIM D(6),S(6):   '6 unter DATA unsortierte bzw. sortierte Daten
205 '
210 '*******************************************************
220 '
230 RESTORE:FOR I=1 TO 6:READ D(I):NEXT I
240 DATA 102,101,109,106,104,105.
250 PRINT "0  Ende"
260 PRINT "1  Sortieren durch 'Austausch nach Auswahl'"
270 PRINT "2  Sortieren durch 'Paarweisen Austausch (Bubble Sort)'"
280 PRINT "3  Sortieren durch 'Lineare Auswahl'"
290 PRINT "4  Sortieren durch 'Shell Sort'"
300 INPUT "Wahl 0-4";W$:LET W=VAL(W$)
310 ON W GOSUB 1050,2060,3060,4060
320 IF W=0 THEN PRINT "Ende.":END
330 IF INKEY$=""THEN 330
340 CLS:GOTO 230
350 '
```

```
1000 '*******************************************************
1010 '
1020 '                  Austausch nach Auswahl
1030 '
1040 '*******************************************************
1045 '
1050 PRINT
1055 PRINT "Sortierprotokoll zum 'Austausch nach Auswahl':"
1060 FOR I=1 TO 5
1070    LET STELLEMIN = 1
1080    FOR J=I+1 TO  6
1090      IF D(J)<D(STELLEMIN) THEN LET STELLEMIN=J
1100    NEXT J
1110    SWAP D(I),D(STELLEMIN)
1120    FOR Y=1 TO 6:PRINT D(Y);:NEXT Y:PRINT
1130 NEXT I
1140 RETURN
1150 '
2000 '*******************************************************
2010 '
2020 '                     Bubble Sort
2030 '
2040 '*******************************************************
2050 '
2060 PRINT "Sortierprotokoll zum 'Bubble Sort':"
2070 FOR I=1 TO 6
2080    FOR J=1 TO 6
2090      IF D(J)>D(I) THEN SWAP D(I),D(J)
2100      FOR Y=1 TO 6:PRINT D(Y);:NEXT Y:PRINT
2110    NEXT J
2120 NEXT I
2130 RETURN
2140 '
3000 '*******************************************************
3010 '
3020 '                    Lineare Auswahl
3030 '
3040 '*******************************************************
3050 '
3060 PRINT "Sortierprotokoll zur 'Linearen Auswahl':"
3070 FOR I=1 TO 6
3080    LET MIN=99999!
3090    FOR J=1 TO 6
3100       IF D(J)<MIN THEN LET MIN=D(J):LET STELLEMIN=J
3110    NEXT J
3120    LET S(I)=D(STELLEMIN):LET D(STELLEMIN)=99999!
3130    FOR Y=1 TO 6:PRINT S(Y);:NEXT Y:PRINT
3140 NEXT I
3150 FOR I=1 TO 6:LET S(I)=0:NEXT I
3160 RETURN
3170 '
```

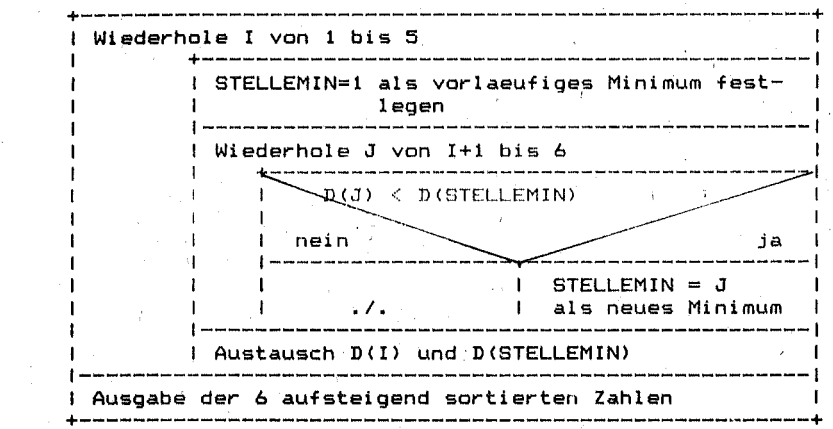
```
4000 '*********************************************************
4010 '
4020 '                      Shell Sort
4030 '
4040 '*********************************************************
4060 PRINT "Sortierprotokoll zum 'Shell Sort':"
4070 LET SCHRITT=4
4080 WHILE SCHRITT<6
4090   LET SCHRITT=SCHRITT+SCHRITT
4100 WEND
4110 LET SCHRITT=SCHRITT-1:LET ENDE=0
4120 WHILE NOT ENDE
4130 LET SCHRITT =INT(SCHRITT/2)
4140   IF SCHRITT<1 THEN ENDE =-1:GOTO 4220
4150   FOR I=1 TO 6-SCHRITT
4160     FOR J=I TO 1 STEP-SCHRITT
4170       IF D(J+SCHRITT)>D(J) THEN LET J=1:GOTO 4190
4180       SWAP D(J),D(J+SCHRITT)
4190     NEXT J
4200     FOR Y=1 TO 6:PRINT D(Y);NEXT Y:PRINT
4210   NEXT I
4220 WEND
4230 RETURN
```

6.2.2. Sortieren ueber Zeiger

Bei umfangreicheren Datenbestaenden ist es guenstiger, nur die Speicherplaetze dieser Zahlen ueber Pointer zu sortieren, die Zahl selbst aber unbewegt zu lassen.
Das Programm **SORTZEIG** demonstriert das mit denselben Daten und dem Sortierverfahren 'Austausch nach Auswahl' wie im Programm **SORTNUM.**

Unsortiertes Feld:	Sortiertes Zeiger-Feld:	Sortiertes Feld (ueber Zeiger):
D(1) = 102	Z(1) = 2	D(Z(1)) = 101
D(2) = 101	Z(2) = 1	D(Z(2)) = 102
D(3) = 109	Z(3) = 5	D(Z(3)) = 104
D(4) = 106	Z(4) = 6	D(Z(4)) = 105
D(5) = 104	Z(5) = 4	D(Z(5)) = 106
D(6) = 105	Z(6) = 3	D(Z(6)) = 109

Struktogramm zum Programmteil 'Austausch nach Auswahl':



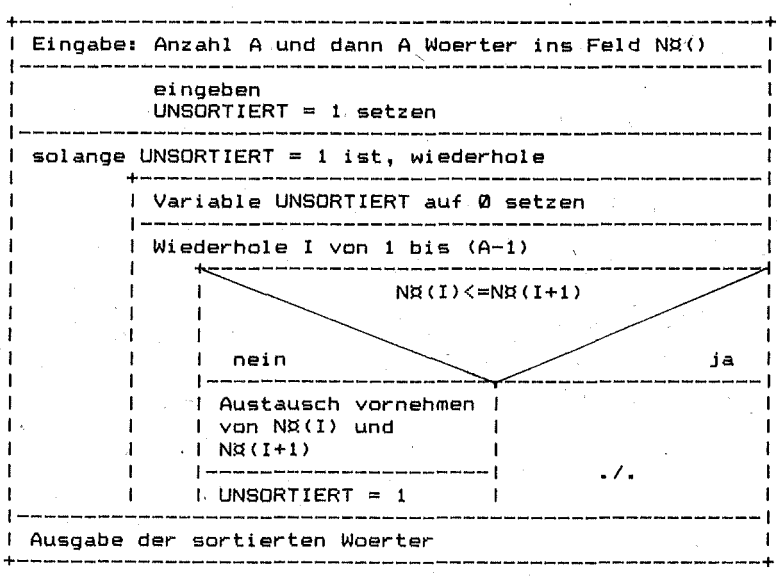
Beispiel: SORTZEIG

```
100  '************************************************************
110  '
120  '            Programm SORTZEIG
130  '
140  '************************************************************
150  '
160  PRINT "Sortieren nach dem Verfahren 'Austausch nach Auswahl
170  PRINT "(Numerische Daten ueber Zeiger sortieren.)
180  'STELLEMIN: Stelle mit vorlaeufigem Minimum
190  DIM Z(6):    '6 Zeiger bzw. Pointer
200  DIM D(6):    '6 unter DATA im Programm gespeicherte Daten
205  '
210  '************************************************************
220  '
230  FOR I=1 TO 6: LET Z(I)=I:NEXT I
240  PRINT :PRINT "  6 Daten umsortieren:"
250  FOR I=1 TO 6:READ D(I):PRINT D(I);:NEXT I
260  DATA 102,101,109,106,104,105
270  PRINT :PRINT "Sortierprotokoll der 6 Zeiger:"
280  FOR I=1 TO 5
290    FOR Y=1 TO 6:PRINT Z(Y);:NEXT Y:PRINT
300    LET STELLEMIN=I
310    FOR J=I+1 TO 6
320      IF D(J) < D(Z(STELLEMIN)) THEN LET STELLEMIN=J
330    NEXT J
340    SWAP Z(I),Z(STELLEMIN)
350  NEXT I
360  PRINT "6 Daten ueber Zeiger sortieren:"
370    FOR I=1 TO 6:PRINT D(Z(I));:NEXT I
380  PRINT :PRINT "Ende.":END
```

6.2.3. Zeichenketten sortieren

Es sollen Namen aus einem Zeichenkettenfeld sortiert werden. In der Variablen **UNSORTIERT** bleibt solange eine Ø bis kein Austausch mehr erfolgt.

Struktogramm zum Sortieren ZK:



Beispiel: **SORTZK**

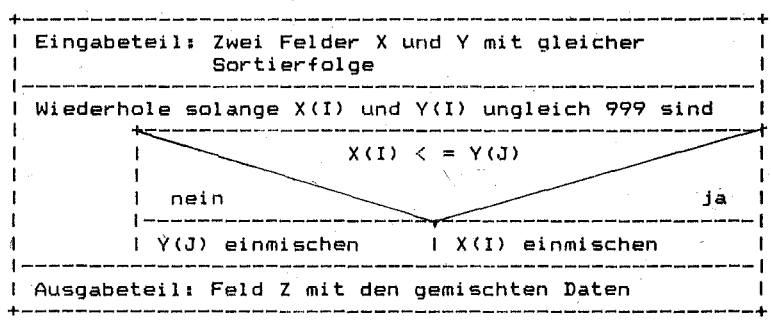
```
100  '*************************************************************
110  '
120  '                Programm SORTZK
130  '            (Sortieren Zeichenketten)
140  '
150  '*************************************************************
160  '
170  'NØ(A),A:          A Namen im Feld NØ gespeichert
180  'UNSORTIERT :      Wahrheitswert Ø = unsortiert
190  '                                1  = sortiert
200  '
210  '*************************************************************
220  '
230  PRINT : INPUT "Anzahl der Namen";A
240  DIM NØ(A)
250  PRINT A ; "Namen einzeln eintippen:"
260     FOR I=1 TO A
270     INPUT NØ(I)
280     NEXT I
```

```
290  PRINT "Kontrollausgabe zum Sortieren"
300  LET UNSORTIERT=1
310  WHILE UNSORTIERT
320     LET UNSORTIERT=0
330     FOR Y=1 TO A: PRINT N%(Y) ;""
340     NEXT Y :PRINT
350     FOR I=1 TO A-1
360       IF N%(I)<=N%(I+1) THEN 390 ELSE 370
370       SWAP N%(I),N%(I+1)
380       LET UNSORTIERT=1
390     NEXT I
400  WEND
410  PRINT :PRINT "Programmende" : END
```

6.3. Mischen von Daten

Mischen heisst, Daten unter Beruecksichtigung ihrer Sortierfolge zu einer Datenstruktur zusammenzufuegen.
Im Beispiel sollen ein 5-Elemente-Feld X() und ein 4-Elemente-Feld Y() zum 9-Elemente-Feld Z() gemischt werden. Fuer die Endebehandlung wird je ein zusaetzliches Element fuer die groesste Zahl 999 verwendet.

Struktogramm zum Mischen von Daten:



Beispiel: MISCHDAT

```
100  '*******************************************************
110  '
120  '              Programm MISCHDAT
130  '
140  '*******************************************************
150  '
160  PRINT "Feld X und Y zu einem Feld Z mischen"
170  DIM X(6),Y(5),Z(9):  'drei numerische Felder
175  '
```

```
180  '*******************************************************
190  '
200  PRINT :PRINT "Datenbestand 1:"
210    FOR I=1 TO 5:READ X(I):PRINT X(I);:NEXT I
220    DATA 10,20,30,40,50
230  PRINT :PRINT "Datenbestand 2:"
240    FOR I=1 TO 4:READ Y(I):PRINT Y(I);:NEXT I
250    DATA 15,20,25,45
260  LET I=1 : LET J=1 : LET K=1
270  WHILE NOT ((X(I)=999) AND (Y(J)=999))
280    IF X(I) <= Y(J)
          THEN LET Z(K)=X(I):I=I+1:X(6)=ABS(999*(I=6))
          ELSE LET Z(K)=Y(J):J=J+1:Y(5)=ABS(999*(J=5))
290  LET K=K+1
300  WEND
310  PRINT :PRINT "Datenbestand 1 und 2 gemischt:"
320    FOR K=1 TO 9:PRINT Z(K);:NEXT K
330  END
```

6.4. Gruppieren von Daten (Gruppenwechsel)

Daten werden abhaengig von Ordnungsbegriffen (Auftragsnummer, Kundennummer, ...) zu Gruppen zusammengefasst. Bei Wechsel des Ordnungsbegriffes werden z.B. Gruppensummen gebildet.
Der Gruppenwechsel kann auch mehrstufig sein (Hauptgruppe, Untergruppen).
Im folgenden Beispiel wird beim Wechsel der Auftragsnummer die Summe ausgegeben.

Beispiel: GRUPPDAT

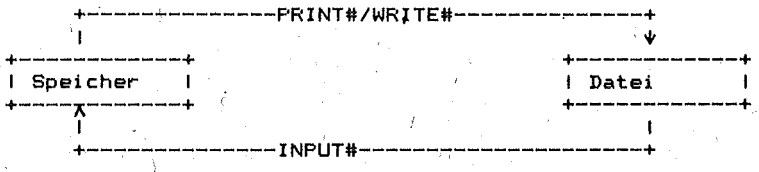
```
100 '*******************************************************
110 '
111 '                Programm "GRUPPDAT"
112 '                  (Gruppenwechsel)
113 '
120 '*******************************************************
125 '
130 'A2, M :  Datensatz mit den Elementen Auftrag und Menge
140 'A1, S1:  Auftrag alt und Gruppensumme
145 '
150 '*******************************************************
155 '
160 INPUT "Autrag,Menge ";A2,M
170 LET A1=A2
180 WHILE A2<>0
190   WHILE A2=A1          'kein Gruppenwechsel bei A2=A1
200     LET S1=S1+M        'Gruppensumme erhoehen
210     INPUT "Auftrag,Menge ";A2,M
220   WEND
230   PRINT A1,"mit Gruppensumme ";S1
240   LET S1=0:LET A1=A2
250 WEND
260 PRINT "Ende.":END
```

7.____Dateiarbeit

In diesem Abschnitt werden verschiedene Formen des Dateizugriffs an Beispielen erlaeutert.
Es handelt sich dabei um sequentiellen Zugriff, Direktzugriff und indexsequentielle Verarbeitung. Zeigerverkettete Listen und binaere Baeume als dynamische Datenstrukturen werden ebenfalls betrachtet.

7.1.___Sequentielle_Dateien

Eine sequentielle Datei wird fortlaufend, Datensatz fuer Datensatz bearbeitet.



7.1.1._Menuesteuerung

Nach dem Aufruf des Programms wird ein Menue mit mehreren Auswahlmoeglichkeiten angezeigt.

Beispiel: Telefonliste als sequentielle Datei

Menue zur Verwaltung der Telefondatei

0	Beenden	
1	Laden	der Datei
2	Speichern	der Datei extern
3	Drucken	Gesamtverzeichnis
4	Eingeben	von Daten
5	Suchen	eines Datensatzes
6	Aendern	eines Datensatzes
7	Loeschen	eines Datensatzes
8	Einfuegen	eines Datensatzes
9	Sortieren	der Gesamtdatei

7.1.2. Dateiweiser Datenverkehr

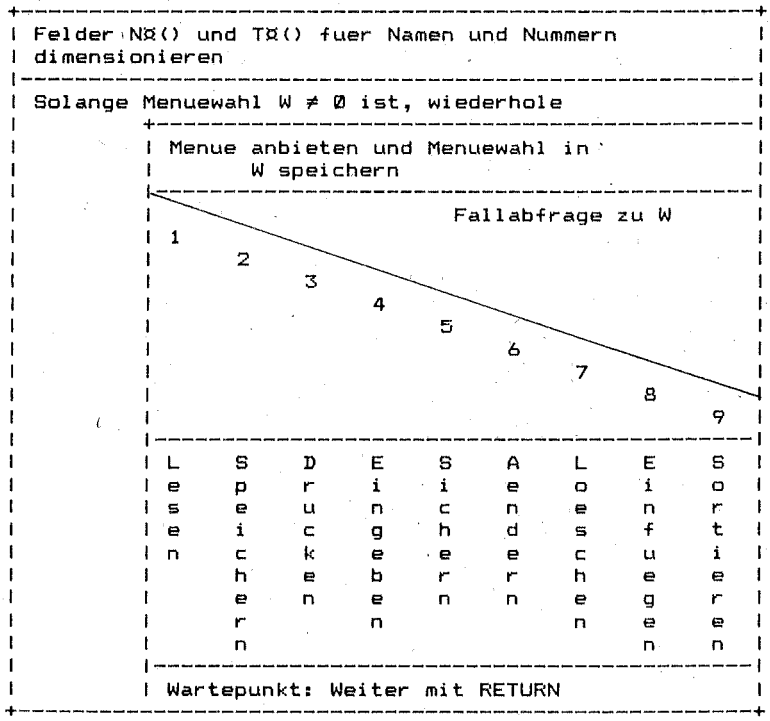
Die Datei wird Datensatz fuer Datensatz auf die Diskette als externe Datei abgespeichert bzw. als komplette Datei in den Speicher eingelesen.

Vorteil : schnelle interne Verarbeitung

Nachteil: Datei ist groessenmaessig durch den Speicherplatz begrenzt

Beim dateiweisen Datenverkehr wird meist das Prinzip der parallelen Felder angewandt, bei dem gleiche Indizes auf denselben Datensatz verweisen. Im Beispiel der Telefonliste wird ein Feld fuer die Namen **N$(I)** und ein Feld fuer die Telefonnummern **T$(I)** verwendet.

Struktogramm zum Programm SEQUEN



7.1.3. Verarbeitung von Feldern in Unterprogrammen

Das Programm besteht aus einer Schleife, die eine Fallabfrage enthaelt. Es koennen 8 Unterprogramme aufgerufen werden.

Lesen:
- Datei eroeffnen
- Datei verarbeiten, Lesen der Datensaetze in die Felder N%, T$.
  Erster Dateieintrag ist die Satzanzahl N.
- Datei schliessen

Speichern:
- Schreiben Satzanzahl
- Schreiben N Datensaetze auf die Datei

Suchen:
- Sequentielles Suchen
- Zaehlerschleife hat nur einen Ausgang
- dient der Ablaufsteuerung

Loeschen:
- physisches Loeschen
  In der Zaehlerschleife werden alle Eintraege ab dem zu loeschenden Eintrag um ein Element in den Feldern N%() und T$() vorgerueckt.

Einfuegen:
- Die Zaehlerschleife
  FOR Z=N TO W+2 STEP -1
  rueckt vom letzten Satz N ausgehend Eintraege um jeweils eine Position nach hinten, um den neuen Eintrag einzufuegen.

Sortieren:
- Sortiert wird nach dem Prinzip
  'Austausch nach Auswahl'.
  Es werden Zeichenketten sortiert, die Anzahl der Sortierbegriffe N ist variabel.

Beispiel: SEQUEN

```
100 '*********************************************************
110 '
120 '          PROGRAMM SEQUEN
130 '
140 '*********************************************************
150 '
160 PRINT "Telefonliste als sequentielle Datei." : PRINT
165 '
170 DIM N$(100): '100-Elemente-Feld fuer die Namen
180 DIM T$(100): '100-Elemente-Feld fuer die Telefonnummern
190 'W,W$:       Wahlmoeglichkeit bei Menueauswahl
200 'F$:         Dateiname(Filename) fuer die sequentielle Datei
210 'I,Z,F:      Laufvariable bzw. Flags
215 '
220 '*********************************************************
230 '
240 LET W=1
250 WHILE W<>0
260   GOSUB 500          'Aufruf Unterprogramm Menueangebot
270   CLS
275   END
280   ON W GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9000
290   PRINT "Weiter mit RETURN"; : LET W$=INPUT$(1) : CLS
300 WEND
310 PRINT "Programmende." : PRINT
315 '
320 '*********************************************************
330 '
500 PRINT "Menue zur Verwaltung der Telefon-Datei"
510 PRINT "--------------------------------------"
520 PRINT " 0   Beenden"
530 PRINT " 1   Lesen       der Datei"
540 PRINT " 2   Speichern   der Datei extern"
550 PRINT " 3   Drucken     Gesamtverzeichnis"
560 PRINT " 4   Eingeben    von Eintraegen"
570 PRINT " 5   Suchen      eines Eintrags"
580 PRINT " 6   Aendern     eines Eintrags"
590 PRINT " 7   Loeschen    eines Eintrags"
600 PRINT " 8   Einfuegen   eines Eintrags"
610 PRINT " 9   Sortieren   der Gesamtdatei"
620 PRINT "--------------------------------------"
630 INPUT "Wahl 0-9";W$ : LET W=VAL(W$)
640 IF W<0 OR W>9 THEN PRINT "Eingabe bitte zwischen 0 und 9." :
645 GOTO 630
650 IF W<>INT(W) THEN PRINT "Eingabe bitte ganzzahlig vornehmen." :
655 GOTO 630
660 RETURN
665 '
670 '*********************************************************
680 '
1000 INPUT "Name der Datei";F$
1010 OPEN F$ FOR INPUT AS #1
1020 INPUT #1,N
```

```
1030   FOR I=1 TO N : INPUT #1,N$(I),T$(I) : NEXT I
1040 PRINT N ; "Eintraege von Datei ";F$;" in den Hauptspeicher."
1050 CLOSE #1
1060 RETURN
1065 '
2000 INPUT "Name der Ausgabedatei"; F$
2010 INPUT "Bisherige Datei zerstoeren (ja/nein)";W$
2020 IF W$<>"ja" THEN 2080
2030 OPEN F$ FOR OUTPUT AS #1
2040 PRINT #1,N
2050   FOR I=1 TO N : PRINT #1,N$(I);",";T$(I) : NEXT I
2060 PRINT N;"Eintraege vom Hauptspeicher in die Datei ";F$;"."
2070 CLOSE #1
2080 RETURN
2090 '
3000 PRINT "Name:                Telefonnummer:"
3010 PRINT "--------------------------------"
3020 FOR I=1 TO N
3030   PRINT N$(I);TAB(24); T$(I)
3040   IF INT(I/10)=I/10 THEN INPUT "Weiter blaettern",W$
3050 NEXT I
3060 PRINT "Dateiende nach ";N;"Eintraegen."
3070 RETURN
3080 '
4000 LET N=N+1
4010 INPUT "Name  (0=Ende)";N$(N)
4020   IF N$(N)="0" THEN LET N=N-1 : GOTO 4040
4030 INPUT "Telefonnummer";T$(N) : GOTO 4000
4040 RETURN
4045 '
5000 INPUT "Zu suchender Name"; W$
5010 LET F=0
5020 FOR I=1 TO N
5030   IF LEFT$(N$(I),LEN(W$))=W$
         THEN PRINT "Gefundene Nummer: ";T$(I): LET I=N:
5040 NEXT I
5050 IF NOT F THEN PRINT W$; " nicht gefunden."
5060 RETURN
5065 '
6000 INPUT "Name des zu aendernden Eintrags";W$ : LET F=0
6010 FOR I=1 TO N
6020   IF LEFT$(N$(I),LEN(W$))=W$ THEN 6030 ELSE 6090
6030     WHILE W$<>"ja"
6040       PRINT N$(I)
6050       PRINT T$(I);" aendern in "; : INPUT T$(I)
6060       PRINT N$(I);" ";T$(I);" korrekt (ja/nein) "; : INPUT W$
6070     WEND
6080   LET I=N : LET F=-1
6090 NEXT I
6100 IF NOT F THEN PRINT "Eintrag ";W$ ;" nicht gefunden."
6110 RETURN
6115 '
7000 INPUT "Name des zu loeschenden Eintrags";W : LET F=0
7010 FOR I=1 TO N
7020   IF LEFT$(N$(I),LEN(W$)<>W$ THEN 7100
7030   PRINT N$(I);" wirklich loeschen (ja/nein)"; : INPUT W$
```

```
7040   IF W$<>"ja" THEN 7090
7050   FOR Z=1 TO N
7060     LET N$(Z)=N$(Z+1) : LET T$(Z)=T$(Z+1)
7070   NEXT Z
7080   LET N=N-1
7090   LET I=N : LET F=-1
7100 NEXT I
7110 IF NOT F THEN PRINT W$;"nicht gefunden. Kein Loeschen moeglich."
7120 RETURN
8000 PRINT "Datei ";F$;" hat";N;"Eintraege. Nach welchem"
8010 INPUT "Eintrag einfuegen (Satznummer tippen)";W
8020 LET N=N+1
8030 FOR Z=N TO W+2 STEP -1
8040   LET N$(Z)=N$(Z-1)
8050 NEXT Z
8060 PRINT "Nachfolgende Eintraege sind verschoben."
8070 INPUT "Einzufuegender Name  ";N$(W+1)
8080 INPUT "Einzufuegende Nummer ";  T$(W+1)
8090 RETURN
8095 '
9000 PRINT "Sortieren von ";N;"Datensaetzen beginnt."
9010 FOR I=1 TO N-1     'Sortiermethode "Austausch nach Auswahl"
9020   LET STELLMIN=I : LET NAMMIN$=N$(I) : LET TELMIN$=T$(I)
9030   FOR Z=(I+1) TO N
9040     IF N$(Z)<NAMMIN$ THEN LET STELLMIN=Z: NAMMIN$=N$(Z):TELMIN$=T$(Z)
9050   NEXT Z
9060   LET N$(STELLMIN)=N$(I) : LET N$(I)=NAMMIN$
9070   LET T$(STELLMIN)=T$(I) : LET T$(I)=TELMIN$
9080 NEXT I
9085 '
9090 PRINT  "Sortieren im Hauptspeicher beendet."
9100 RETURN
```

7.2. Direktzugriffsdateien

Alle Datensaetze haben gleiches Format und werden ueber Satznummern adressiert. Eine beliebige Reihenfolge der Nummern ist zulaessig. Bei nicht benutzten Satznummern wird der Bereich entsprechend der Satzlaenge auf der Diskette frei gehalten, so dass eine nachtraegliche Erweiterung der Datei leicht moeglich ist.

Der direkte Dateizugriff soll anhand des Programmes "ARTIKEL" erlaeutert werden. Dieses Programm verwaltet eine Artikeldatei.

Wesentliche Merkmale:

- Menuesteuerung
- Direktzugriffs-Datei
- Datensaetze mit konstanter Satzlaenge
- Overlay durch Verkettung von fuenf eigenstaendigen Programmen
- Satzweiser Datenverkehr
- Direkte Adressierung des Datensatzes

7.2.1. Datei mit konstanter Datensatzlaenge

Dateiname: ARTDATEI
Satzlaenge: 23 Stellen (4 Elemente)

Datensatzbeschreibung:

Inhalt:	Artikel-nummer	Bezeichnung	Menge	Stueckpreis
Stellen	2	15	2	4
Datentyp	Integer	Zeichenkette	Integer	Real-einfach
Variablen-name	A1	A2$	A3	A4
Puffer-name	P1$	P2$	P3$	P4$
Bsp.	1002	Bleistift	50	0.20

In BASIC hat die Direktzugriffs-Datei im Gegensatz zur sequentiellen Datei eine feste Datensatzlaenge, die in der OPEN-Anweisung angegeben werden muss.

```
OPEN "ARTDATEI" AS #1 LEN=23
        |         |    |
        +---------|----|-------  Dateiname
                  +----|-------  Dateinummer
                       +-------  Datensatzlaenge fest
```

Die Datensatzlaenge einer eingerichteten Datei kann spaeter nicht mehr veraendert werden. Ohne die Angabe LEN werden 128 Stellen als Satzlaenge angenommen.

7.2.2. Overlay durch Verkettung von Programmen

Im Menueprogramm **ART-M** wird ueber eine Kennziffer (1..4) das gewuenschte Programm ausgewaehlt, und mit der Anweisung

RUN "Programmname"

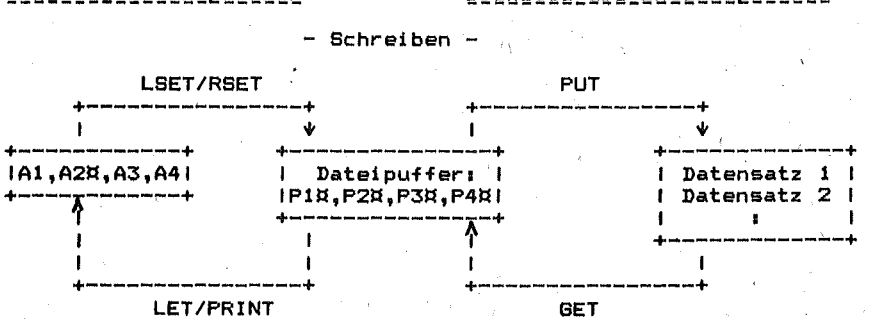
wird **ART-M** ueberlagert.
Nach der Ausfuehrung eines mit **RUN** gerufenen Programms laedt dieses durch **..RUN "ART-M"** das Menueprogramm wieder in den Speicher.
Mit dem Overlay werden alle vom rufenden Programm bislang erzeugten Variablenwerte zerstoert.

5 Programme

ART-M	(Menue)
ART-A	(Anlegen der Datei)
ART-S	(Schreiben Datensaetze)
ART-L	(Lesen Datensaetze)
ART-F	(Fortschreiben Bestaende)

7.2.3. Datensatzweiser Datenverkehr

Im Gegensatz zum dateiweisen Datenverkehr wird hier jeweils nur ein einzelner Satz gelesen, geschrieben oder geaendert. Dadurch kann die Artikeldatei groesser sein als der Speicher. Es wird mit einem Dateipuffer, Teil des Speichers, gearbeitet, in dem die Datensaetze formgerecht bereitgestellt werden.



Mit der Anweisung **FIELD** wird der Dateipuffer in Datenfelder eingeteilt, Feldlaenge und Puffervariable werden festgelegt;

```
FIELD #1, 2 AS P1$, 15 AS P2$, 2 AS P3$, 4 AS P4$.
```

Puffervariable sind immer vom Typ Zeichenkette, deshalb muessen numerische Werte immer gewandelt werden.

```
LSET P1$ = MKI$(1002)
```

wandelt die Integerzahl **1002** in eine Zeichenkette der Laenge **2** um und traegt diese linksbuendig in die Puffervariable **P1$** ein.

Beim Lesen von Datensaetzen mit **GET** (z.B. GET #1,34 heisst Lesen 34. Satz) muessen die Zeichenketten gegebenenfalls wieder in numerische Werte zurueckgewandelt werden.

```
LET A1 = CVI(P1$)
```

wandelt die 2 Byte-Zeichenkette der Puffervariablen **P1$** in eine Integerzahl um.
Im Programm **ART-A** wird das Anlegen einer neuen Datei gezeigt. Die alte Datei wird geloescht, und die neue Datei kann mit Leersaetzen beschrieben werden.
Durch die Anweisungen

```
FOR S=I TO ANZ
  PUT #1,S
NEXT S
```

wird die Direktzugriffsdatei seriell beschrieben, da **S** in der Schleife jeweils um 1 hochgezaehlt wird.

7.2.4. Direkte Adressierung des Datensatzes

Die Nummer des Datensatzes ist abhaengig von einem Ordnungsbegriff, in diesem Programmbeispiel abhaengig von der Artikelnummer. Es besteht ein Zusammenhang zwischen Artikelnummer als Ordnungsbegriff einerseits und der relativen Satznummer als Speicherort andererseits.
Der erste Artikel hat die Nummer 1001 und ist somit der 1. Satz der Datei. Die Satznummer ergibt sich beim Schreiben aus Artikelnummer -1000.

```
LET S=A1-1000
PUT #1,S
```

Beim Lesen eines Datensatzes wird die Satznummer aus dem Suchbegriff ermittelt.

```
INPUT "Artikel-Nr. "; SUCH
LET S=SUCH-1000
GET #1,S
```

Bei dieser direkten Adressierung spielt die Reihenfolge der Speicherung keine Rolle. Bei nicht benutzten Satznummern (Artikelnummern) bleibt der Speicherplatz auf der Diskette leer. Es entstehen Luecken. Deshalb sollte der Direktzugriff moeglichst nur bei einer relativ geschlossenen Folge von Ordnungsbegriffen angewandt werden.

Beispiel: VERWALTUNG ARTIKELDATEI

```
100 '*********************************************************
110 '
120 '          Verwaltung Artikeldatei
130 '              (Menueprogramm)
140 '
150 '*********************************************************
160 '
170 'E$,E:                    Eingabe bei Menueauswahl
180 'F$:                      Dateiname
190 'ANZ:                     Anzahl Datensaetze
200 'S:                       Satznummer (Artnr. - 1000)
210 'A1,A2$,A3,A4:            4 Felder im Datensatz
220 'P1$,P2$,P3$,P4$:         4 Felder im Puffer
230 'SUCH:                    Artnr. als Suchbegiff
240 'ZUAB:                    Zu- oder Abgang bei Bestandsaenderungen
250 '
260 '*********************************************************
270 '
280 '          Programm ART-M
290 '         (Menueprogramm)
300 '
310 '*********************************************************
320 '
330 PRINT
340 PRINT "  1 = Neue Datei anlegen"
350 PRINT "  2 = Datensaetze schreiben"
360 PRINT "  3 = Datensaetze lesen"
370 PRINT "  4 = Bestand fortschreiben"
380 PRINT "  5 = Liste der Artikel"
390 PRINT "  6 = Ende"
400 INPUT "Auswahl 1..6 ",E$
410 LET E = VAL(E$)
420 IF E = 6 THEN PRINT "Ende !" : END
425 '
430 ON E GOTO 460,470,480,490,500
440 PRINT "Auswahl wiederholen !" : GOTO 400
450 '
460 RUN "ART-A"
470 RUN "ART-S"
480 RUN "ART-L"
490 RUN "ART-F"
500 RUN "ART-D"
```

Beispiel: ART-A

```
100 '*********************************************************
110 '
120 '          Progr amm ART-A
130 '          (Datei anlegen)
140 '
150 '*********************************************************
155 '
160 LET F$ ="ARTDATEI"
170 PRINT "Dateiname: ";F$
180 OPEN "R",#1,F$,23
190 FIELD #1,2 AS P1$,15 AS P2$,2 AS P3$,4 AS P4$
200 PRINT "Datei ";F$;" neu eroeffnet!"
210 PRINT
220 PRINT "Leersaetze auf Datei schreiben (j/n)?";
230 LET E$=INPUT$(1) : PRINT E$
240 IF E$<>"j" THEN 320
250 INPUT "Wieviel Leersaetze? ";ANZ
260 LSET P1$=MKI$(0):LSET P2$=" "
270 LSET P3$=MKI$(0):LSET P4$=MKI$(0)
280 FOR S=1 TO ANZ
290   PUT #1,S
300 NEXT S
310 PRINT ANZ;" Leersaetze geschrieben!"
315 '
320 CLOSE #1
330 RUN "ART-M"
```

```
100 '************************************************************
110 '
120 '          Programm ART-S
130 '    (Datensaetze schreiben)
140 '
150 '************************************************************
160 '
170 OPEN "R",#1,"ARTDATEI",23
180 FIELD #1,2 AS P1$,15 AS P2$,2 AS P3$,4 AS P4$
190 PRINT "Saetze schreiben "
200 PRINT "Nummer, Bezeichnung, Bestand, Preis:"
204 LET A1=1
205 WHILE A1<>0
210 INPUT;"",A1,A2$,A3,A4
215 '
230 LSET P1$=MKI$(A1) : LSET P2$=A2$
240 LSET P3$=MKI$(A3) : LSET P4$=MKS$(A4)
250 LET S=A1-1000       'Adressrechnung
260 PUT #1,S
270 PRINT TAB(40);
280 INPUT"WEITER ?  J/N ";X$
290 IF X$="N" OR X$="n" THEN LET A1=0
300 WEND
305 '
310 CLOSE #1
320 RUN "ART-M"
```

Beispiel: ART-L

```
100 '*********************************************************
110 '
120 '          Programm ART-L
130 '       (Lesen Datensaetze)
140 '
150 '*********************************************************
160 '
170 OPEN "R",#1,"ARTDATEI",23
180 FIELD #1, 2 AS P1$, 15 AS P2$, 2 AS P3$, 4 AS P4$
190 PRINT "Nummer  Bezeichnung      Bestand   Preis"
200 INPUT ;"",SUCH
205 WHILE SUCH >1000
210   LET S=SUCH-1000
220   ON ERROR GOTO 310
230   GET #1,S
240   IF EOF(1) THEN PRINT "DATEIENDE!":GOTO 330
250   LET A1=CVI(P1$) : LET A2$=P2$
260   LET A3=CVI(P3$) : LET A4=CVS(P4$)
270   IF A1 = 0 THEN PRINT "Nr. nicht vorhanden!" : GOTO 295
280 PRINT SPC(2) A2$ TAB(28) USING" ####"; A3 ;
290 PRINT SPC(2) USING "###.##";A4
295 INPUT;"",SUCH
300 WEND
310 IF (ERR>0) AND (ERL=230) THEN PRINT "Artnr ";SUCH;"nicht gefunden !" :
                              RESUME 320
320 CLOSE #1
330 RUN"ART-M"
```

Beispiel: **ART-F**

```
100 '************************************************************
110 '
120 '          Programm ART-F
130 '     (Bestand fortschreiben)
140 '
150 '************************************************************
160 '
170 OPEN "R",#1,"ARTDATEI",23
180 FIELD #1, 2 AS P1$, 15 AS P2$, 2 AS P3$, 4 AS P4$
190 LET SUCH =1001
200 WHILE SUCH > 1000
210 INPUT "Artikelnr. "             ;SUCH
220 IF SUCH = 0 THEN 370
230   LET S = SUCH - 1000
240   ON ERROR GOTO 380
250   GET #1,S
260   LET A1=CVI(P1$): LET A2$=P2$
270   LET A3=CVI(P3$): LET A4=CVS(P4$)
280   PRINT "Artnr.       ";A1
290   PRINT "Art-Bez:      ";A2$
300   PRINT "Bestand:     ";USING "#####";A3
310   PRINT "Preis:       ";USING "##.##";A4
320   INPUT "Bestandsaenderung + -" ; ZUAB
330   LET A3=A3+ZUAB
340   LSET P3$=MKI$(A3)
350   PUT #1,S
360 PRINT "Fortgeschrieben auf "         ;A3
365 '
370 WEND
380 IF (ERR>0) AND (ERL=250) THEN PRINT "Artnr. nicht gefunden"
390 CLOSE #1
400 RUN "ART-M"
```

```
100 '************************************************************
110 '
120 '          Programm ART-D
130 '       (Liste der Artikel)
140 '
150 '************************************************************
160 '
170 OPEN "R",#1,"ARTDATEI",23
180 FIELD #1, 2 AS P1$,15 AS P2$,2 AS P3$,4 AS P4$
190 FOR S=1 TO 20
200 ON ERROR GOTO 300
210 GET #1,S
220 LET A1=CVI(P1$)
230 IF A1=0 THEN 270
240 LET A2$=P2$
250 LET A3=CVI(P3$) : LET A4=CVS(P4$)
260 PRINT A1,A2$,USING "#####";A3;
265 PRINT SPC(2) USING"##.##";A4
270 NEXT S
300 IF (ERR>0) AND (ERL=210) THEN PRINT "FEHLER !"
310 CLOSE #1
320 RUN "ART-M"
```

7.3. Indexsequentielle Datei

Das Prinzip des indexsequentiellen Zugriffs soll am folgenden Beispiel "Kundendatei" erlaeutert werden. Auf diese Kundendatei wird ueber eine Indexdatei als Inhaltsverzeichnis zugegriffen.

1. Zugriff: ueber Index (KEY)
2. Zugriff: direkt (RANDOM)

Ein Programmpaket mit 3 Teilprogrammen

INDEX-S (Schreiben)
INDEX-L (Lesen in den Speicher)
INDEX-T (Sortieren im Speicher)

verwaltet eine indexsequentiell organisierte Datei.

Dateiaufbau:

Name der Kundendatei: **KUNDE**

Kundennummer	Name	Umsatz
K	K$	U
P1$	P2$	P3$

(Dateipuffer)

Name der unsortierten Indexdatei: **IKUNDE**

Kundennummer	Satznummer
K	S
P4$	P5$

Name der sortierten Indexdatei: **ISORT**

Kundennummer	Satznummer
K	S
P6$	P7$

Mit **INDEX-S** werden die Kunden ueber Tastatur erfasst und in der Datei **KUNDE** gespeichert. Parallel dazu wird in die Indexdatei **IKUNDE** die Kundennummer mit der dazugehoerigen Satznummer geschrieben.
Mit dem Sortierprogramm **INDEX-T** wird die Indexdatei in den Speicher gelesen, dort sortiert und dann in die sortierte Indexdatei **ISORT** geschrieben.

Mit **INDEX-L** koennen die Dateien gelesen werden. Es ist zu beachten, dass bei der index-sequentiellen Organisation zu **einer** Datei mehrere Indexdateien angelegt werden koennen.

Beispiel:

KUNDE unsortiert			IKUNDE unsortiert		ISORT sortiert	
K	KN	U	K	S	K	S
104	MEIER	250.00	104	1	101	2
101	FISCHER	780.00	101	2	104	1
110	MANN	1500.00	110	3	109	4
109	SAUER	960.00	109	4	110	3
:			:		:	
:			:		:	

Die Indexdatei wird mit dem Parameter "R" als Direktzugriffsdatei eroeffnet, spaeter aber sequentiell beschrieben. Das ist erforderlich, weil nachtraeglich die Satzanzahl als 1. Satz in die Datei geschrieben werden soll. Dieser Wert wird im Sortierprogramm benoetigt.

Mit dem Programmteil **INDEX-L** koennen die Kunden sortiert nach Kundennummern gelesen werden.
Der Zugriff erfolgt in zwei Schritten:

1. sequentiell auf sortierte Indexdatei
2. direkt auf Datendatei

Zu dieser Datei koennen noch zusaetzlich sortierte Indexdateien erstellt werden.

Primaerindexdatei: - Kundennummer

Sekundaerindexdatei: - Indexdateien bezogen auf andere Ordnungsbegriffe des Datensatzes (z.B. fuer Name und Kennsatz)

Voll-Index: - zu jedem Satz der Datei erfolgt ein Eintrag in die Indexdatei (Bsp. Kundendatei)

Teil-Index: - nur bei sortierten Dateien moeglich z.B. erhaelt nur jeder 10. Satz einen Eintrag in die Indexdatei

Arbeitsweise:

1. Sequentieller oder direkter Zugriff auf Indexdatei
2. Direktzugriff auf Datendatei
3. ggf. sequentiell in Datendatei weitersuchen

Beispiel1 VERWALTUNG KUNDENDATEI

```
100 '*********************************************************
110 '
120 '          Verwaltung Kundendatei
130 '            (indexsequentiell)
140 '
150 '*********************************************************
160 '
170 'F$,FI$:          Name Kundendatei / Indexdatei
180 'L$,LI$:          Laufwerke (A oder B)
190 'K,K$,U:          Datensatz Kundendatei
200 '                 Kundennummer, Kundenname, Umsatz
210 'K,S:             Datensatz Indexdatei
220 '                 Kundennummer, Satznummer
230 'Hinweis: Satzanzahl S ist als 1. Indexsatz gespeichert
240 '
250 '*********************************************************
260 '
270 '          Programm INDEX-S (Schreiben)
280 '
290 '*********************************************************
300 '
310 ON ERROR GOTO 610
320 INPUT "Laufwerk fuer Kundendatei ";L$
330 INPUT "Laufwerk fuer Indexdatei ";LI$
340 LET F$="KUNDE" : LET FI$="IKUNDE"
350 ON ERROR GOTO 600
360 LET S=1
370 OPEN L$+":"+F$ AS #1 LEN =20
380 FIELD #1, 2 AS P1$, 14 AS P2$, 4 AS P3$
390 OPEN LI$+":"+FI$ AS #2 LEN =4
400 FIELD #2, 2 AS P4$, 2 AS P5$
410 GET #2,1
420 LET K=CVI(P4$) : LET S=CVI(P5$)
430 PRINT "Kunr.; Name, Umsatz"
440 PRINT "(0=Ende)
450 LET K=1
460 WHILE K<>0
470   LET S=S+1
480   INPUT;" ",K
485   IF K=0 THEN 570
490   INPUT ;" ",K$
500   INPUT " ",U
510   LSET P1$=MKI$(K) : LSET P2$=K$ : LSET P3$=MKS$(U)
520   PUT #1,S
530   LSET P4$=P1$ : LSET P5$=MKI$(S)
540   PUT #2,S
550 WEND
```

```
560 '
570 LSET P4$=MKI$(0) '1.SATZ = SATZANZAHL
580 PUT #2,1
590 CLOSE
595 PRINT
600 PRINT "Datei geschlossen !" : END
610 IF (ERR>0) AND (ERL=540) THEN PRINT "FEHLER !" : GOTO 590
```

Beispiel: **INDEX-T**

```
1000 '*****************************************************
1010 '
1020 '         Programm INDEX-T
1025 '      (Sortieren Kundendatei)
1030 '
1040 '*****************************************************
1050 '
1060 PRINT "Sortieren Kundendatei"
1070 OPEN "IKUNDE" AS #2 LEN=4
1080 FIELD #2, 2 AS P4$, 2 AS P5$
1090 OPEN "ISORT" AS #3 LEN =4
1100 FIELD #3, 2 AS P6$, 2 AS P7$
1110 GET #2,1 'Satzanzahl aus IKUNDE lesen
1120 LET S=CVI(P5$)
1130 DIM I(S,2)    'Indextabelle dynamisch dimensionieren
1140 FOR Z=1 TO S
1150   GET #2,Z    'Index aus IKUNDE in Tabelle I einlesen
1160   LET I(Z,1)=CVI(P4$) 'Kundennr.
1170   LET I(Z,2)=CVI(P5$) 'Satznr.
1180 NEXT Z
1190 PRINT S, "Saetze in Indextabelle I gelesen
1200 PRINT
1210 PRINT "Sortieren Indextabelle"
1220 LET SORT=1
1230 FOR X= 2 TO (S-1)
1240   FOR Z=2 TO (S-1)
1260     IF I(Z,1)<=I(Z+1,1) THEN 1300 ELSE 1270
1270       SWAP I(Z,1),I(Z+1,1)
1280       SWAP I(Z,2),I(Z+1,2)
1300   NEXT Z
1310 NEXT X
1320 PRINT "Indextabelle aufsteigend sortiert"
1330 PRINT "Indextabelle sortiert in Datei ISORT schreiben"
1340 PRINT "Kundennr.","Satznr."
1350 FOR Z=1 TO S
1360 LSET P6$=MKI$(I(Z,1))
1370 LSET P7$=MKI$(I(Z,2))
1380 PUT #3,Z
1390 PRINT I(Z,1),I(Z,2)
1400 NEXT Z
1410 PRINT "Datei geschlossen !"
1420 CLOSE
1430 END
```

Beispiel: INDEX-L

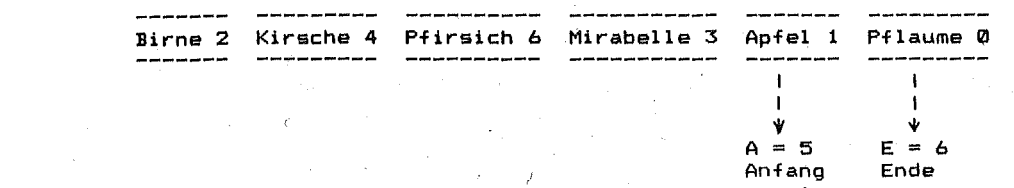
```
2000 '*************************************************************
2010 '
2020 '          Programm INDEX-L
2030 ' (Lesen sortierte Kundendatei)
2040 '
2050 '*************************************************************
2060 '
2070 PRINT "Lesen sortierte Kundendatei"
2080 OPEN "KUNDE" AS #1 LEN = 20
2090 FIELD #1, 2 AS P1$, 14 AS P2$,4 AS P3$
2100 OPEN "ISORT" AS #3 LEN =4
2110 FIELD #3, 2 AS P6$, 2 AS P7$
2120 PRINT "Nummer","Kundenname","                    Umsatz"
2130 FOR I=2 TO 9999
2140 ON ERROR GOTO 2210
2150 GET #3,I
2160 LET S=CVI(P7$)
2170 GET #1, S
2180 LET K=CVI(P1$): LET K$=P2$ : LET U = CVS(P3$)
2190 PRINT K,K$,USING "#,###.##";U
2200 NEXT I
2210 CLOSE:END
```

7.4 Gekettete Liste

Um auf Daten einer Datei schnell zugreifen zu koennen, ist eine sortierte Datei anzuwenden. Beim Hinzufuegen von Daten muesste stets neu sortiert werden.
Mit der geketteten Liste steht eine dynamische Datenstruktur zur Verfuegung, bei der neue Daten einfach ans Ende angefuegt werden koennen.
Zu jedem Datensatz wird je ein Zeiger auf seinen Nachfolger (Sohn) gespeichert. Dadurch bleibt die physische Speicherungsfolge erhalten, und ueber ein Zeigerfeld wird eine logische Speicherungsfolge aufgebaut.

Im folgenden Beispiel soll eine Artikeldatei mit verschiedenen Obstsorten erstellt und gepflegt werden. Zur Vereinfachung besteht jeder einzelne Datensatz nur aus einem Namensfeld und einem Zeigerfeld. Diese Datenstruktur nennt man lineare gekettete Liste.

Darstellung einer linearen geketteten Liste mit 6 Elementen:



LØ() = Feld fuer Listenelemente (Namen)
Z() = Feld fuer Listenzeiger (naechster Name)

Die folgende Uebersicht zeigt, welche Werte die Variablen bei der Abarbeitung des Beispiels annehmen.

Arbeitsgang	LØ()	Z()	A	E
leere Liste	blanc	Ø	Ø	Ø
Name eingeben	BIRNE	Ø	1	1
Name anfuegen	BIRNE	2		
	KIRSCHE	Ø	1	2
Name anfuegen	BIRNE	2		
	KIRSCHE	3		
	PFIRSICH	Ø	1	3
Name anfuegen	BIRNE	2		
	KIRSCHE	4		
	PFIRSICH	Ø		
	MIRABELLE	3	1	4
Name anfuegen	BIRNE	2		
	KIRSCHE	4		
	PFIRSICH	Ø		
	MIRABELLE	3		
	APFEL	1	5	5
Name anfuegen	BIRNE	2		
	KIRSCHE	4		
	PFIRSICH	6		
	MIRABELLE	3		
	APFEL	1		
	PFLAUME	Ø	5	6

Gekettete Liste als Datei extern ablegen:

Die Datei fuer die Namen ist sequentiell organisiert. Im ersten Datensatz stehen die beschriebenen Daten wie Listenanfang A und Listenende E.
Die Datenfelder werden durch "," und die Datensaetze durch CHR$(13) bzw. 'Wagenruecklauf' getrennt.

```
7040 PRINT #1,L$(I);",";Z(I)
```

Beispiel: LILIST

```
100 ************************************************************
101 '
102 '              Programm LILIST
103 '              (gekettete Liste)
104 '
105 '************************************************************
106 '
110 PRINT "Demonstration: Gekettete Liste (Linked List)"
120 PRINT "als dynamische Datenstruktur":PRINT
130 '
140 'L$(100):       Maximal 100 Elemente der linearen geketteten Liste
150 'Z%(100):          "     100 Zeiger     "       "          "       "
160 'A, E:          Zeiger "Anfang der Liste" und "Ende der Liste"
170 'H:             Hilfszeiger
180 'I:             Laufvariable, Listenzeiger
190 'F$:            Dateiname zur externen Speicherung der Liste
200 'L$(I):         Datensatz mit zwei Datenfeldern "Listenelement"
210 'Z(I):          und "Listenzeiger"
220 'W$, E$:        Hilfsvariablen fuer die Eingabe
230 '
240 PRINT "1   Leere Liste erzeugen"
250 PRINT "2   Neue Elemente eingeben"
260 PRINT "3   Liste logisch ausgeben"
270 PRINT "4   Liste physisch ausgeben"
280 PRINT "5   Datei mit Liste laden"
290 PRINT "6   Liste in Datei speichern"
300 INPUT "Ihre Wahl (0=Ende)";W
310 IF W=0 THEN PRINT "ENDE." : END
320 ON W GOSUB 1000,2000,4000,5000,6000,7000
330 PRINT :PRINT "Weiter mit Taste";: LET E$=INPUT$(1):PRINT
340 CLS
350 GOTO 240
360 '
1000 LET A=0     'Anfang der Liste          IM INTERNSPEICHER
1010 LET H=0     'Hilfszeiger               LEERE LISTE ERZEUGEN
1020 LET I=0     'Listenzeiger zwischen A und H als Laufvariable
1030 IF E<>0 THEN ERASE L$,Z   'Array loeschen zum Redimensionieren
1040 LET E=0     'Ende der Liste
1050   DIM L$(100)      'Liste mit den Listenelementen selbst
1060  DIM Z(100)      'Zeiger auf L$
1070 PRINT "Liste leer dimensioniert."
1080 RETURN
```

```
1090 '
2000 INPUT "Neues Element (0=Ende)";E$     'NEUE LISTENELEMENTE EINGEBEN
2010 WHILE E$<>"0"
2020   LET E=E+1
2030   LET L$(E)=E$
2040   GOSUB 3000       'E$ einordnen
2050   INPUT "Neues Element (0=Ende)";E$
2060 WEND
2070 RETURN
2080 '
3000 LET I=A                                'ELEMENT SORTIERT EINORDNEN
3010 IF E$<=L$(I) OR I=0 THEN 3050
3020   LET H=I
3030   LET I=Z(I)
3040 GOTO 3010
3050 IF I<>A THEN 3060 ELSE 3070
3060   LET Z(E)=I : LET Z(H)=E : GOTO 3080
3070   LET Z(E)=A : LET A=E
3080 RETURN
3090 '
4000 LET I=A                                'LISTE IN SORTIERFOLGE AUSGEBEN
4010 WHILE I<>0
4020   PRINT L$(I)
4030   LET I=Z(I)
4040 WEND
4050 RETURN
4060 '
5000 FOR I=1 TO E                           'LISTE IN SPEICHERFOLGE AUSGEBEN
5010   PRINT L$(I); TAB(10); Z(I)
5020 NEXT I
5030 RETURN
5040 '
6000 GOSUB 1000   'Leere Liste.             'LISTE IN INTERNSPEICHER LADEN
6010 INPUT "Von welcher Datei laden";F$
6020 OPEN F$ FOR INPUT AS #1
6030 INPUT #1, A,E
6040 FOR I=1 TO E
6050   INPUT #1, L$(I), Z(I)
6060 NEXT I
6070 CLOSE #1
6080 PRINT "Liste aus "; F$; " in den Hauptspeicher geladen."
6090 RETURN
6100 '
7000 INPUT "Dateiname "; F$                 'LISTE EXTERN ALS DATEI ABSPEICHERN
7010 OPEN F$ FOR OUTPUT AS #1
7020 PRINT #1, A; ","; E
7030 FOR I=1 TO E
7040   PRINT #1, L$(I) ; "," ; Z(I)
7050 NEXT I
7060 CLOSE #1
7070 PRINT "Liste auf Diskette in der Datei "; F$ ; " gespeichert."
7080 RETURN
```

7.5. Binärer Baum

Der binaere Baum gehoert wie die zeigerverkettete Liste zu den dynamischen Datenstrukturen. Der Unterschied zur verketteten Liste besteht darin, dass jedes Baumelement (Knoten = node) stets zwei Zeiger hat.

Darstellung eines binären Baumes in BASIC:

Das BASIC-System hat fuer Baeume und Zeiger keine besonderen Sprachelemente. Deshalb wird die Datenstruktur **'Binaerbaum'** abstrakt in einem Feld dargestellt. Fuer jeden Baumknoten werden mindestens vier Eintraege benoetigt.

Element, Vorgaenger, linker und rechter Nachfolger

B$(I) I L(I) R(I)

Intern im Hauptspeicher wird der Binaerbaum in 3 Feldern abgelegt, extern wird er als sequentielle Datei (NUMDATEI) in 8 Saetzen (8 Artikelnummern) mit den 3 Datenfeldern **Element, linker Sohn** und **rechter Sohn** gespeichert.

Binaerbaum

intern

I	B$()	L()	R()
0	CHR$(126)	1	8 = Anzahl N
1	ME421000	3	2
2	NA421030	0	4
3	HA093320	5	6
4	SI		
5	BO		
6	KE		
7	BO		
8	PL		
-			

extern

NUMDATEI

mit der Speicherungsfolge

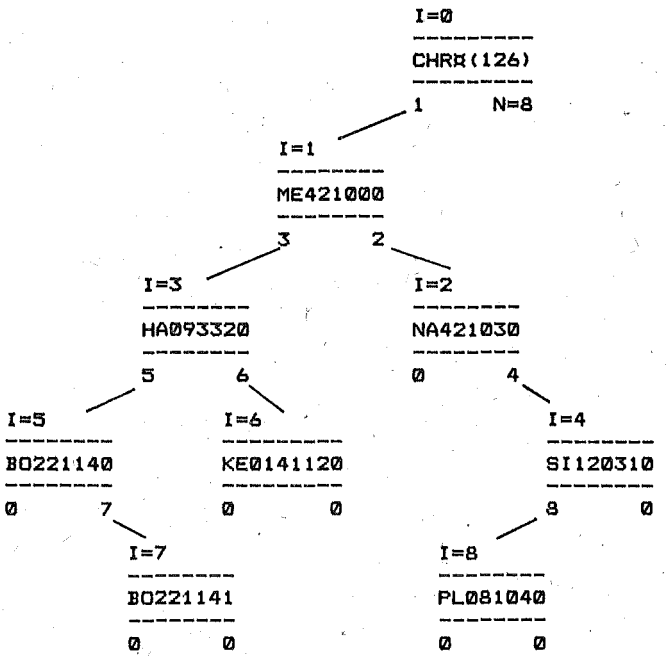
```
CHR$(126) 1 8 /
ME421000  3 4 /
NA421030  0 4..
```

Anmerkung zu Satz 0:

L(0)=1 deutet auf Wurzel
R(0)=8 Anzahl der Knoten des Baumes

Grafische Darstellung eines Binaerbaums:

8 Artikelnummern als Binaerbaum mit 8 Knoten strukturiert



Aufbau der Artikelnummer:

```
BO221140
| | |  |
+-|-|--|-----  BOHRER als Artikelbezeichnung
  +-|--|-----   22    als Lagerort
    +--|-----  114    als Lieferantennummer
       +-----    0    als Unterscheidung
```

Die Anordnung der Artikelnummern als Binaerbaum hat den Vorteil, schnell zugreifen zu koennen und bei Aenderungen nicht alle Nummern bewegen zu muessen.

Eingeben von Elementen in den Binaerbaum

Die Artikelnummern werden seriell gespeichert. Die logische Verankerung geschieht nur ueber Zeiger fuer den linken und den rechten Nachfolger.

1. Schritt: Die Knotenanzahl **N** wird um 1 erhoeht, um die Artikelnummer **EX** durch **LET BX(N)=EX** hinten anzuhaengen.

2. Schritt: In einer Suchschleife wird von der Wurzel ausgehend gefragt, ob rechts eingetragen werden soll (**IF BX(N) > BX(I)** erfuellt) oder links (Bedingung nicht erfuellt).
Bei erfuellter Bedingung gibt es zwei Faelle:
Existiert ein rechter Nachfolger (**IF R(I) <> 0** erfuellt), dann wird dorthin gegangen (**LET I=R(I)**) und die Suchschleife wiederholt (**WEND**).
Gibt es noch keinen rechten Nachfolger, wird die Artikelnummer durch Setzen des rechten Nachfolge-Zeigers (**LET R(I)=N**) an dieser Stelle - logisch - abgelegt und die Suchschleife beendet (**LET FLAG = -1**).
Die Eintragungen links im Baum erfolgen entsprechend.

Regel:

- Ein Element tritt nur einmal auf.
- Ein Vater (Wurzel) hat maximal zwei Soehne (direkte Nachfolger), 0 heisst kein Sohn.
- Der linke Sohn ist alphanumerisch kleiner als der Vater.
- Der rechte Sohn ist alphanumerisch groesser als der Vater.

Binaerbaum sortiert ausgeben

Waehrend die unsortierte Ausgabe des Baumes ueber eine **FOR**-Schleife entsprechend der physischen Speicherung realisiert wird, erfolgt die sortierte Ausgabe gemaess der logischen Folge durch die Zeigervermerke.

1. Schritt: 'Kleinste Nummer suchen', die sich ganz links aussen befindet.
Die Schleife tastet sich vom Stamm (**I=0**) ausgehend immer weiter nach links vor (**LET I=L(I)**), bis endlich kein linker Sohn mehr auftaucht (**IF L(I)=0** ist erfuellt). Die gefundene kleinste Nummer wird ausgegeben.

2. Schritt: Suchen der naechst hoeheren Nummer
Diese ist entweder der Vater oder ein rechter Sohn. Ist kein rechter Sohn da, so wird der Vater ausgegeben. Ist dagegen ein rechter Sohn vorhanden (wie in diesem Beispiel **IF R(I)=0** nicht erfuellt), so geht man zu diesem Sohn (**LET I=R(I)**), um dann wie in Schritt 1 in die aeusserste linke Ecke zu gelangen.
Der rechte Sohn wird als Wurzel eines Teilbaumes aufgefasst, in dem sich das 'Suchen der kleinsten Nummer ganz links' genauso vollzieht wie im Gesamtbaum. Ist dieses Minimum gefunden, dann wird erneut der Vorgang 'Suchen der naechst hoeheren Nummer' aufgerufen.
Ein solches "Aufrufen von sich selbst" nennt man **Rekursion.**
In diesem Beispiel werden im Suchfeld **S(J)** die beim 'Vortasten' durchlaufenen Knoten (**LET J=J+1**) gespeichert, um dann auf dem gleichen Weg zurueckgehen zu koennen (**LET J=J-1**)
BR(0) enthaelt einen grossen 'Start-Wert', in diesem Beispiel **CHR$(126)**, dass entspricht **sz.** Das bewirkt, dass der eigentliche Anfangsknoten **BR(1)** stets ein linker Sohn des Hilfsknotens **BR(0)** ist.

Binaerbaum_als_Datei_extern_ablegen

Die Datei wird sequentiell unter **NUMDATEI** geschrieben oder gelesen. Der 1. Datensatz enthaelt 1 fuer die Wurzel und die Knotenanzahl 8. Dann folgen die Datensaetze mit je 3 Feldern.

Beispiel: BIBAUM

```
  90 '*********************************************************
  91 '
  92 '          Programm BIBAUM
  93 '    (dynamische Datenstruktur)
  94 '
  95 '*********************************************************
  96 '
 110 PRINT "Demonstration: Binaerer Baum als dynamische Datenstruktur"
 112 '
 130 'B$(100):          Maximal 100 Baumelemente bzw. Knoten
 140 'R():              Rechte Soehne als Nachfolger
 150 'L():              Linke Soehne als Nachfolger
 160 'S():              Suchfeld als Hilfsvariable beim Sortieren
 170 'N:                Anzahl der Baumelemente, in R(0) abgelegt
 180 'B$(I),L(I),R(I): 3-Felder-Datensatz fuer I. Baumelement in der Datei
 190 'F$:               Name der sequentiellen Datei
 200 'I,J,Z:            Hilfsvariablen
 210 'Flag:             Variable als Schleifensteuerung
 220 '
 225 '*********************************************************
 230 '
 240 PRINT "0  Ende"
 250 PRINT "1  Leeren Binaerbaum erzeugen"
 260 PRINT "2  Neue Elemente eingeben"
 270 PRINT "3  Baum sortiert ausgeben"
 280 PRINT "4  Baum unsortiert ausgeben"
 290 PRINT "5  Datei mit Baum laden"
 300 PRINT "6  Baum in Datei speichern"
 310 INPUT "Wahl 0-6";Z : PRINT : IF Z=0 THEN PRINT "Ende." : END
 320 ON Z GOSUB 1000,2000,3000,4000,5000,6000
 330 PRINT : PRINT "Weiter mit Taste"; : LET E$=INPUT$(1) : PRINT
 340 CLS : GOTO 240
 350 '
1000 DIM B$(100)                        'LEEREN BINAERBAUM ERZEUGEN
1010 DIM L(100), R(100), S(100)
1020 LET I=0            'bei Wurzel 0 beginnen
1030 LET N=0            'Anzahl der Knoten 0
1040 LET B$(0)=CHR$( 126)     'Wurzel mit hohem Codewert
1050 PRINT "Binaerbaum leer eingerichtet."
1060 RETURN
1070 '
2000 PRINT N+1; ". Element (0=Ende) "; 'NEUE ELEMENTE IN BINAERBAUM EINGEBEN
2010 INPUT E$
2020 WHILE E$<>"0"
2030   LET N=N+1 : LET B$(N)=E$ : LET I=0 : LET FLAG=0
2040   WHILE NOT FLAG
2050     IF B$(N)>B$(I) THEN 2080
2060     IF L(I)<>0 THEN LET I=L(I)
                    ELSE LET R(I)=N:  LET FLAG=-1
2070     GOTO 2090
2080     IF R(I)<>0 THEN LET I=R(I)
                    ELSE LET R(I)=N:  LET FLAG=-1
2090   WEND
```

```
2100   LET FLAG=0
2110   PRINT N+1;". Element (0=Ende) "; : INPUT EB
2120 WEND
2130 LET R(0)=N
2140 RETRUN
2150 '
3000 LET I=1    'Index in Feld B$()      BINAERBAUM SORTIERT AUSGEBEN
3010 LET Z=0    'Rangplatz fuer Sortierung
3020 LET J=0    'Index im Suchfeld S()
3030 PRINT "Links: Suchfeld J,S(J)" : PRINT "Rechts: Knoten Z,B$(I)"
3040 IF L(I)=0 THEN 3060
3050 GOSUB 3200 : LET I=L(I) : GOTO 3040    'links aussen lesen
3060 GOSUB 3180 : IF Z=N THEN 3160
3070 IF R(I)=0 THEN 3090
3080 GOSUB  : LET I=R(I) : GOTO 3040        'rechts lesen
3090 IF I<>L(S(J)) THEN 3120
3100 GOSUB 3220 : GOSUB 3180 : IF Z=N THEN 3160
3110 GOTO 3070
3120 IF J<2 THEN 3160
3130 GOSUB 3220
3140 IF I<>R(S(J)) THEN 3090
3150 IF I>1 THEN 3130
3160 PRINT "Ende des Sortierens."
3170 RETURN
3180 '
3185 LET Z=Z+1                          'UPRO ELEMENT AUSGEBEN
3190 PRINT "   ";Z;". Element ";B$(I : RETURN
3200 LET J=J+1                          'Upro in Suchfeld weiter
3210 LET S(J)=I : PRINT J; S(J) : RETURN
3220 LET I=S(J)                         'Upro in Suchfeld zurueck
3230 LET J=J-1 : PRINT J;S(J) : RETURN
3240 '
4000 PRINT "Reihenfolge: ";             'Baum unsortiert ausgeben
4010 PRINT "I,B$(I),L(I),B$(L(I)),R(I),B$(R(I))"
4020 FOR I=1 TO N
4030   PRINT I;" ";B$(I);L(I);" ";B$(L(I));R(I);" ";B$(R(I))
4040 NEXT I
4050 PRINT "Ende der unsortierten Ausgabe."
4060 RETURN
4070 '
5000 GOSUB 1000                         'BINAERBAUM AUS DATEI IN SPEICHER LESEN
5010 INPUT "Dateiname",F$ : OPEN F$ FOR INPUT AS #1
5020 INPUT #1, B$(0),L(0),R(0)
5030 LET I=0 : LET N=R(0)
5040 FOR I=1 TO N
5050   INPUT #1, B$(I),L(I),R(I)
5060 NEXT I
5070 CLOSE #1
5080 PRINT "Binaerbaum eingelesen."
5090 RETURN
5100 '
6000 INPUT "Dateiname zum Speichern" ; F$     'BINAERBAUM EXTERN ABSPEICHERN
6010 OPEN F$ FOR OUTPUT AS #
6020 FOR I=0 TO N
6030   PRINT #1, B$(I);",";L(I);",";R(I)
6040 NEXT I
```

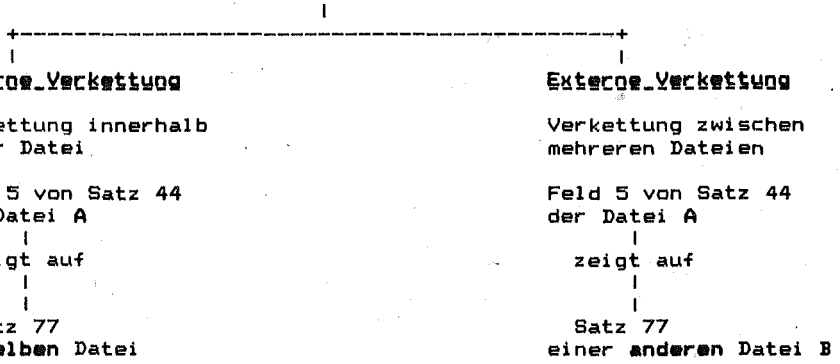
```
6050 CLOSE #1
6060 PRINT "Binaerbaum in ";FR$;" gespeichert."
6070 RETURN
```

7.6. Verkettete Dateien und Datenbank

Der Begriff der **Datenbank** ist aeusserst vielschichtig, hat aber stets etwas mit Dateien zu tun, die zu einem gemeinsamen Datenbestand verkettet sind.
Es gibt eine verkettete Speicherung ueber Zeiger, bei der in jedem Datensatz zwei zusaetzliche Datenfelder mit Zeigern angefuegt werden.
Strukturiert man die Datensaetze als '**Gekettete Liste (Linked List)**', dann werden damit ebenfalls Saetze innerhalb einer Datei verkettet. Dies nennt man auch **interne Verkettung.**

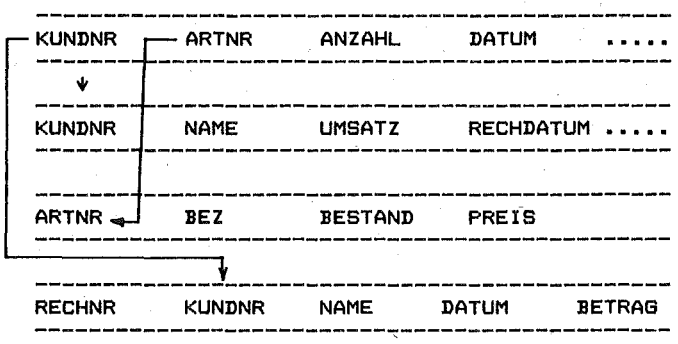
Das Prinzip der Verkettung laesst sich auch auf mehrere Dateien anwenden. Der Schluessel des Datenfeldes einer Datei **A** wird als Zeiger auf den Satz einer Datei **B** betrachtet. Man spricht dann von einer **externen Verkettung.**
In einer Datenbank koennen beide Typen der Verkettung angewendet werden.

Verkettung durch Zeiger



Beispiel: Externe Verkettung von 4 Dateien

Fakturierung mit Bestelldatei, Kundendatei, Artikeldatei, Offene-Posten-Datei



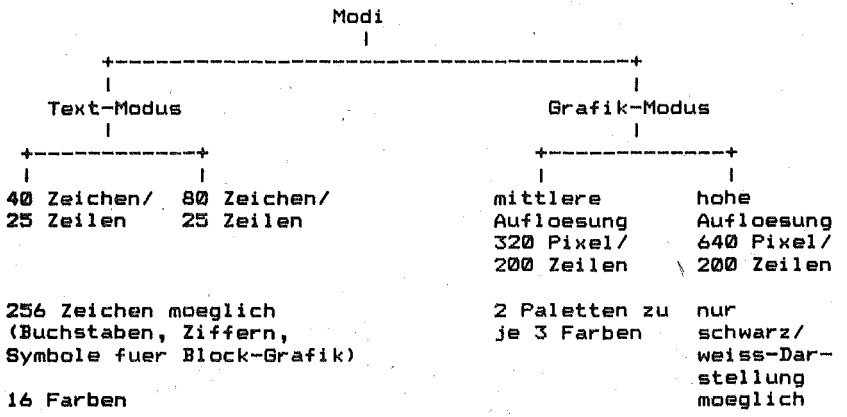
In einer Fakturierung werden die Tagesbestellungen in einer **BESTELLDATEI** erfasst, um dann fuer die Rechnungslegung verwendet zu werden.
Jeder Bestellsatz enthaelt **KUNDNR**, **ARTNR**, **ANZAHL** und **DATUM**. Das Feld **KUNDNR** wird zum Zeiger auf die **KUNDENDATEI**, aus der die entsprechenden Kundenstammdaten abgerufen werden. Das Feld **ARTNR** ist der Zeiger auf die **ARTIKELDATEI**, aus dieser werden die Angaben zum Artikel auf die Rechnung gedruckt.
Die **KUNDNR** zeigt ausserdem noch auf die **OPOS-DATEI**. Da ist ersichtlich, ob gerade offene oder angemahnte Rechnungen vorliegen.
Datenfelder, ueber die Datensaetze derselben oder einer anderen Datei unverwechselbar bzw. eindeutig identifiziert werden koennen, nennt man **eindeutige Schluessel**.
Eine Postleitzahl oder ein Name koennen z.B. nicht als solche Schluessel verwendet werden, da mehrere Kunden am gleichen Ort wohnen bzw. Namen mehrfach auftreten koennen.

8. Grafik

8.1. Ueberblick

Die BASIC-Anweisungen, die softwareseitig die Grafik- und Farbmoeglichkeiten des Computers unterstuetzen, ermoeglichen vielfaeltige und sehr einfach zu programmierende grafische Darstellungen.
Es sind zwei verschiedene Modi zu unterscheiden:

- Text-Modus zur Darstellung von Zeichen
- Grafik-Modus zur punktweisen Darstellung (Pixelgrafik)

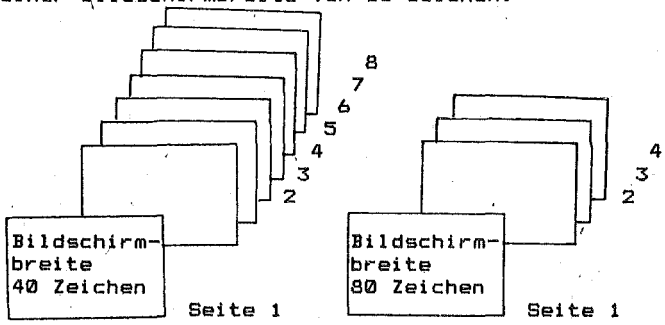


8.2. Text-Modus

Der Text-Modus wird mit der Anweisung **SCREEN 0** eingestellt. **WIDTH** stellt die Bildschirmbreite und **COLOR** die Farbe ein. Im Text-Modus koennen nur Zeichen (keine Bildpunkte) dargestellt werden. Unter den 256 Zeichen des ASCII-Zeichensatzes befinden sich jedoch auch fuer die Grafik interessante Zeichen, wie z.B. die Blockgrafik-Symbole.
Daraus ist ersichtlich, dass die Betriebsart Text-Modus zwar begrenzte, aber sehr leistungsfaehige Anwendungen erlaubt.

Im Text-Modus kann mehr als eine Bildschirmseite gespeichert werden.
Dadurch wird es moeglich, eine Seite anzuzeigen, waehrend die andere Seite erstellt wird.

Bei einer Bildschirmbreite von 40 Zeichen koennen acht Textseiten gespeichert werden, und vier Textseiten sind moeglich bei einer Bildschirmbreite von 80 Zeichen.



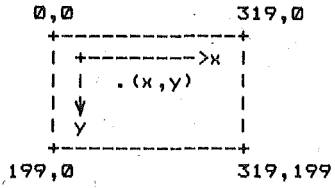
Mit Hilfe der Anweisung **SCREEN** kann die aktuelle Seite (mit der gearbeitet wird) und die visuelle Seite (die angezeigt wird) ausgewaehlt werden. Zu beachten ist hierbei, dass alle Seiten denselben Kursor verwenden.
Das bedeutet zum Beispiel, wenn auf die Seite 1 ausgegeben wird und der Kursor steht am Anfang der Zeile 13, so wird beim Umschalten auf eine andere Seite die Ausgabe am Anfang der Zeile 13 fortgesetzt.
Deshalb ist es notwendig, die Kursorkoordinaten mit Hilfe der Anweisung **CSRLIN** und der Funktion **POS(0)** zu retten.

8.3. Grafik-Modus

8.3.1. Grafik mittlerer Aufloesung

Im Grafik-Modus mittlerer Aufloesung wird der Bildschirm in 200 Zeilen zu je 320 Spalten eingeteilt. Dabei ist jeder Bildpunkt (Pixel) einzeln ansteuerbar.
Die linke obere Ecke des Bildschirm besitzt die Koordinaten (0,0), die rechte untere Ecke die Koordinaten (319,199).
Es wird zuerst die Spalte und dann die Zeile angegeben, im Gegensatz zum Text-Modus.

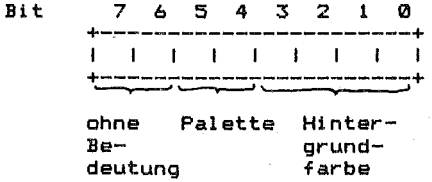
Die Anweisung **SCREEN** stellt den Bildschirmmodus ein und setzt im Grafik-Modus den "zuletzt angesprochenen Punkt" auf die Bildschirmmitte (160,100 bei der mittlerer Aufloesung).



Im Grafik-Modus wird der gesamte Bildschirmspeicher benoetigt, so dass nur eine Bildschirmseite existieren kann.
Mit Hilfe der Anweisung **WINDOW** ist es moeglich, ein neues Koordinatensystem zu definieren. Hierbei wird ein rechteckiger Bereich spezifiziert durch Angabe zweier Koordinatenpaare (x1,y1) und (x2,y2) als die diagonal gegenueberliegenden Ecken des Rechteckes.
Diese Definition eines neuen Koordinatensystems kann notwendig werden, wenn ein Programm auf Geraeten mit verschiedenen Bildschirmen abgearbeitet werden soll.
Gleichzeitig ist es moeglich, eine Vergroesserung bzw. eine Verkleinerung des Objektes zu erhalten.

Die **VIEW**-Anweisung kann verwendet werden, um Ausschnitte auf dem Bildschirm zu definieren. Ein Ausschnitt ist ein Rechteck auf dem Bildschirm, in dem die folgenden Grafikoperationen angezeigt werden.
Die Groesse wird mit Hilfe von zwei Koordinatenpaaren festgelegt, die die diagonal gegenueberliegenden Ecken des Rechteckes bestimmen.
Mit **VIEW** kann man einen "Bildschirm innerhalb des Bildschirms" definieren und die Grafikanweisungen auf den aktuellen Ausschnitt beschraenken.
Ausschnitte werden durch physikalische Bildschirmkoordinaten definiert und nicht durch vorangestellte **WINDOW**-Anweisungen beeinflusst.
Da **VIEW** und **WINDOW** sich nicht gegenseitig beeinflussen, spielt die Reihenfolge der Ausfuehrung dieser beiden Anweisungen keine Rolle.

Im Grafikmodus ist es moeglich, eine Farbe aus zwei verschiedenen Paletten auszuwaehlen.
Mit Hilfe einer Portausgabe auf die Portadresse **3D9H** koennen zwei weitere Paletten ausgewaehlt werden, die jeweils die intensiven Farben der Palette 0 und 1 enthalten.
Das auszugebende Byte ist folgendermassen aufgebaut:



Mit den Bit 0...3 kann die Hintergrundfarbe ausgewaehlt werden (Farbe 0...15). Bit 4 und 5 waehlen die Palette aus.

Bit 5	Bit 4	
0	0	Palette 0
1	0	Palette 1
0	1	Palette 0 intensiv
1	1	Palette 1 intensiv

Beispiel:

```
10 SCREEN 1
20 OUT &H3D9,53
30 CIRCLE (100,100),50,1
```

In der Zeile 20 wird fuer den Hintergrund die Farbe 5 (violett) und die Palette 1 intensiv ausgewaehlt.

8.3.2. Grafik hoher Aufloesung

In diesem Modus stehen 640 Spalten und 200 Zeilen zur Verfuegung. Jedoch ist dieser Modus nicht so vielseitig verwendbar wie die mittlere Aufloesung, da nur zwei Farben moeglich sind: schwarz und weiss.
Eine Aenderung der Vordergrundfarbe ist moeglich, indem ueber eine Portausgabe auf die Portadresse **3D9H** die entsprechende Farbnummer ausgewaehlt wird. Hierbei sind die Farbnummern 0...15 zugelassen.

Beispiel:

```
10 CLS: SCREEN 2
20 OUT &H3D9,4
30 CIRCLE (100,100),50
```

Der Kreis wird in der Vordergrundfarbe rot gezeichnet. Der Hintergrund bleibt schwarz. Diese Einstellung gilt fuer alle weiteren Ausgaben. Bei Aenderung der Farbe aendert sich die Vordergrundfarbe aller Abbildungen.

8.4. Programmbeispiele

- Das Programm "MODUS" demonstriert die verschiedenen Betriebsarten: Textmodus, Grafikmodus mittlere und hohe Aufloesung.

```
  90 '*********************************************************
 100 '
 101 '          Programm MODUS
 102 '     Demonstration der Modi TEXT und GRAFIK
 103 '
 104 '*********************************************************
 110 PRINT "Demonstration der Grafik-Betriebsarten bzw."
 120 PRINT "Grafik-Modi Text, mittlere und hohe Aufloesung"
 130 INPUT "Weiter: Taste ",E$
 140 LET TEXT$="Text-Modus Farbe mit 40 Zeichen:"
 150 SCREEN 0
 160 WIDTH 40
 170 GOSUB 1000
 180 LET TEXT$="Text-Modus Farbe mit 80 Zeichen:"
 190 WIDTH 80
 200 GOSUB 1000
 250 LET TEXT$="Grafik-Modus mittlere Aufloesung (320*200):"
 260 KEY OFF:SCREEN 1,0
 270 GOSUB 2000
 280 LET TEXT$="Grafik-Modus hohe Aufloesung (640*200):"
 290 SCREEN 2
 300 GOSUB 2000
 310 SCREEN 0,0:COLOR 7,0,0:WIDTH 80:KEY ON
 320 PRINT "ENDE.":END
 330 '
1000 FOR HINTERGRUND=0 TO 7
1010   FOR VORDERGRUND=1 TO LEN(TEXT$)
1020     LET ZEICHENFARBE=VORDERGRUND MOD 32
1030     COLOR ZEICHENFARBE,HINTERGRUND
1040     PRINT MID$(TEXT$,VORDERGRUND,1);
1050   NEXT VORDERGRUND
1060   PRINT:PRINT
1070 NEXT HINTERGRUND
1080 COLOR 7,0,0:INPUT "Weiter: Taste ",E$
1090 RETURN
1100 '
2000 CLS
2010 PRINT TEXT$
2020 PRINT "100 Kreise zeichnen:"
2030 FOR RADIUS=1 TO 100
2040   CIRCLE(200,110),RADIUS
2050 NEXT RADIUS
2060 IF INSTR(TEXT$,"hohe")<>0 THEN 2120
2070 PRINT "15 Hintergrundfarben:"
2080 FOR FARBE=1 TO 15
2090   COLOR FARBE
2100   FOR ZEIT=1 TO 1000:NEXT ZEIT
2110 NEXT FARBE
2120 INPUT "Weiter: Taste ",E$
2130 RETURN
```

- Zeichnen einer Parabel mit Angabe des Koordinatensystems

```
  9 '*********************************************************
 10 '
 11 '           Programm PARABEL
 12 '        Zeichnen einer PARABEL (mit Achsen)
 13 '
 14 '*********************************************************
 20 CLS:KEY OFF
 30 SCREEN 1
 39 '
 40 '           Zeichnen horizentale Linie
 41 '
 50 DRAW "c1 bm48,176"
 60 DRAW "c1 m208,176"
 70 FOR I=6 TO 26
 80    LOCATE 24,I+1
 90    IF I/5=INT(I/5) THEN PRINT I-5;
100   PSET (8*I,175)
110   IF (I-6)/5=INT((I-6)/5) THEN PSET (I*8,174):PSET (I*8,173)
120 NEXT I
129 '
130 '          Zeichnen vertikale Linie
131 '
140 DRAW "c1 bm48,176"
150 DRAW "c1 m48,56"
160 FOR I=0 TO 14
170   IF I/5=INT(I/5) THEN PSET (49,I*8+56):PSET (51,I*8+56)
180   PSET (49,I*8+56)
190   LOCATE 8+I,1
200   IF I/5=INT(I/5) THEN PRINT USING "###";150-10*I;
210 NEXT I
215 COLOR ,1
219 '
220 '          Zeichnen  Parabel
221 '
230 FOR X=0 TO 20 STEP .1
240   PSET (8*X+48,176-.8*(X-10)^2),2
250 NEXT X
260 END
```

- Verwendung der Anweisungen **VIEW** und **WINDOW**

Die Koordinaten in der VIEW-Anweisung werden als physikalische Koordinaten angegeben.
Die Anweisung WINDOW definiert ein eigenes Koordinatensystem.

```
  4 '*********************************************************
  5 '
  6 '              Programm WINDOW
  7 '          Verwendung von VIEW und WINDOW
  8 '
  9 '*********************************************************
 10 '
 11 '            Verwendung VIEW und dann WINDOW
 12 '
 20 SCREEN 2:CLS:KEY OFF
 30 VIEW (150,50)-(450,150),,1
 40 WINDOW (0,0)-(100,100)
 50 GOSUB 140
 60 REM
 70 FOR I=1 TO 10:BEEP:NEXT I
 79 '
 80 '            Verwendung WINDOW und dann VIEW
 81 '
 90 SCREEN 2:VIEW:CLS
100 WINDOW (0,0)-(100,100)
110 VIEW (100,50)-(500,150),,1
120 GOSUB 140
130 END
139 '
140 '           Zeichnen Kreis und Linie
141 '
150 CIRCLE (50,50),25
160 LINE (25,25)-(75,75)
170 RETURN
```

- Erstellen einer Jahresuebersicht als Balkendiagramm

Erstellen einer Jahresuebersicht. Die 12 Monatswerte werden als Balkendiagramm in mittlerer Aufloesung gezeichnet (Umsaetze, Kontostaende, usw.)

Es werden 4 Unterprogramme aufgerufen, um die 12 Messwerte einzugeben, den groessten Messwert zu suchen, das Koordinatenkreuz und die Balken zu zeichnen.

Tastatureingabe (ab Zeile 1000):

Maximumsuche (ab Zeile 2000):
In der Variablen **MAXIMUM** wird der Index bzw. die Stelle des groessten Messwertes im Feld **WERT** gespeichert.

Koordinatenkreuz (ab Zeile 3000):
Mit der Anweisung **SCREEN 1** wird in den Grafik-Modus mittlere Aufloesung umgeschaltet.

Die **DRAW**-Anweisung zeichnet die Achsen und anschliessend werden die Monatsbezeichnungen unter die X-Achse geschrieben.

Balkendiagramm (ab Zeile 4000):
Es wird die Hoehe der einzelnen Balken errechnet. Jeder Balken ist jeweils 24 Pixel breit.

```
 80 '*******************************************************
 90 '
100 '          Programm BALKEN
105 '       Erstellen eines Balkendiagrammes
106 '
107 '*******************************************************
108 '
110 PRINT "Demonstration zur Grafik mittlere Aufloesung"
120 PRINT "Balkendiagramm fuer die 12 Monate des Jahres"
125 '
130 '*******************************************************
135 '
140 DIM MONAT$(12)      'Feld fuer die Monatsnamen
150 FOR I=1 TO 12:READ MONAT$(I):NEXT I
160 DATA Januar,Februar,Maerz,April,Mai,Juni,Juli
170 DATA August,September,Oktober,November,Dezember
180 DIM WERT(12)
225 '
230 '*******************************************************
235 '
240 PRINT "Weiter: Taste druecken";:LET E$=INPUT$(1)
250 KEY OFF
260 CLS
270 GOSUB 1000
280 GOSUB 2000
290 GOSUB 3000
300 GOSUB 4000
310 LOCATE 2,1:LET E$=INPUT$(1)
320 KEY ON
330 SCREEN 0
340 WIDTH 80
350 END
360 '
1000 PRINT "Eingabe der 12 Messwerte: "
1010 FOR I=1 TO 12
1020 LOCATE I+10,9
1030 PRINT "Wert im ";MONAT$(I);SPACE$(20)
1040 LOCATE I+10,27:INPUT ;"",E$
1050 LET WERT(I)=VAL(E$)
1060 IF WERT(I)<0 THEN BEEP:LOCATE I+10,25:PRINT SPACE$(20):GOTO 1040
1070 NEXT I
1080 LOCATE 25,1:PRINT "Weiter: Taste druecken":LET E$=INPUT$(1)
```

```
1090 RETURN
1100 '
2000 FOR I=1 TO 12
2010 IF WERT(I)>WERT(MAXIMUM) THEN LET MAXIMUM=I
2020 NEXT I
2030 RETURN
2040 '
3000 SCREEN 1,1
3010 DRAW "BM10,180 u180"
3020 DRAW "bm10,180 r300"
3030 LOCATE 24,4
3040 PRINT "Ja Fe Ma Ap Ma Ju Ju Au Se Ok No De";
3050 LOCATE 1,1:PRINT STR$(WERT(MAXIMUM))
3060 LOCATE 23,2:PRINT "0"
3070 RETURN
3080 '
4000 LET EINHEIT=180/WERT(MAXIMUM)
4010 DRAW "bm10,180"
4020 FOR I=1 TO 12
4030 LET HOEHE=INT(WERT(I)*EINHEIT)
4040 DRAW "u=HOEHE;"
4050 DRAW "r24"
4060 DRAW "d=hoehe;"
4070 NEXT I
4080 RETURN
```

9.___Musik

Musik kann verwendet werden, um verschiedene Programmteile zu betonen, z.B. in Spiel- oder Lernprogrammen bei richtigen Antworten.
Die Programmiersprache **BASIC** unterstuetzt die Erzeugung von Musik bzw. Toenen durch die beiden Anweisungen **SOUND** und **PLAY**.

9.1._Anweisungen_SOUND_und_PLAY

Die beiden Anweisungen generieren Toene ueber den Lautsprecher.
SOUND erzeugt einen Ton, der exakt durch Frequenz und Dauer festgelegt ist.

Die Frequenz kann zwischen 37 und 32767 Hertz liegen. Das menschliche Ohr kann aber nur Toene bis zu einer Frequenz von 20000 Hertz wahrnehmen.
Wird beim Abarbeiten eines Programms eine **SOUND**-Anweisung ausgefuehrt, so setzt **BASIC** die Programmabarbeitung fort, waehrend der Ton generiert wird. Bei Erreichen einer weiteren **SOUND**-Anweisung wird die Abarbeitung des Programms gestoppt, bis die vorhergehende **SOUND**-Anweisung abgearbeitet ist. Wurde jedoch fuer den Parameter Dauer 0 (Null) angegeben (z.B. **SOUND 1300,0**), so wird der Ton, der durch die vorhergehende **SOUND**-Anweisung erzeugt wurde, abgebrochen.
Eine Pause kann erzeugt werden, indem als Frequenz 32767 angegeben wird.

Beispiel:

Verwendung von FOR-NEXT-Schleifen zum Erzeugen von Toenen.

```
10 FOR X=10 TO 200 STEP 20
20 FOR A=100 TO 1000 STEP 100
30  SOUND Y+A, 4.55
40 NEXT A
50 FOR B=1200 TO 2000 STEP 100
60  SOUND Y+B, 4.55
70 NEXT B
80 NEXT X
90 END
```

Beispiel:

Erzeugen der Polizeisirene mit Hilfe von SOUND.

```
10 FOR I=1000 TO 540 STEP -10
20  SOUND I,.5
30 NEXT I
40 FOR I=540 TO 1000 STEP 10
50  SOUND I,.5
60 NEXT I
70 GOTO 10
```

PLAY eignet sich zum Spielen mehrerer Toene hintereinander. Die Anweisung **PLAY** weist als Argument stets eine Zeichenkette auf, die die zu spielende Tonfolge enthaelt. Durch Angabe der Parameter A-G, L, M..., N, O, P, T und X koennen die zu spielenden Noten, die Laenge, Oktave, Pausen, Musikvordergrund oder Musikhintergrund usw. festgelegt werden.
Mit dem Parameter X ist es moeglich, Noten einer Zeichenkettenvariablen zuzuweisen und diese Variable dann in der **PLAY**-Anweisung anzugeben.

9.2. Programmieren eines Liedes

Mit Hilfe des Programms **PLAYMU** werden die wichtigsten Anwendungen der Anweisung **PLAY** demonstriert.
Der Ablauf der Programmierung eines Liedes ist in drei Teile gegliedert:

- Aufbau der Typzeichenkette **TYP$**, um den Musiktyp fuer das Lied festzulegen (Oktave, Tempo, Tonlaenge)

- Aufbau der Notenzeichenketten, die die Noten des Liedes enthalten.
  Es ist sinnvoll, mehrere Teilzeichenketten zu verwenden, falls Teile mehrmals zu spielen sind.

- Spielen des Liedes mit der Anweisung **PLAY**.

Festlegen der Typzeichenkette:

Die Normaleinstellung ist

TYP$ = "O3T120MN" (Oktave 3, Tempo 120, Musik normal)

Es ist aber auch moeglich, die einzelnen Parameterwerte ueber die Tastatur einzugeben. In diesem Fall muessen die einzelnen Teilzeichenketten mit "+" verknuepft werden.

Festlegen der Notenzeichenkette des Liedes:

Bei den meisten Liedern werden die einzelnen Liedzeichenketten NOTEN$ in mehrere Teilzeichenketten aufgeteilt, dann muessen diese Liedpassagen nur einmal codiert bzw. zugewiesen werden.

In diesem Beispiel werden 5 Teilzeichenketten M0$...M4$ verwendet.

Die Leerstellen in den Zeichenketten markieren die Takte. Die Leerstellen koennen auch weggelassen werden. Mit der Anweisungszeile **M0$ = "P2P4"** wird ein 3/4-Auftakt mit **P2** fuer eine halbe und **P4** fuer eine Viertel-Note definiert.

Die 4 Zeichenketten **M1$, M2$,** M3$ und **M4$** enthalten das eigentliche Lied, in diesem Beispiel "Der Mond ist aufgegangen".

```
 50 '*********************************************************
 60 '
 70 '            Programm PLAYMU
 80 '            Lied spielen
 90 '
100 '*********************************************************
110 Input "Oktave (0-6, 3 = mittlere Oktave)"; OKT$
120 Input "Tempo (32-255, 120 = Normaltempo)"; TEMPO$
130 Input "Normal,Legato,Staccato (N, L, oder S)"; Laenge$
140 TYP$ = "O" + OKT$ + "T" + TEMPO$ + "M" + LAENGE$
150 M0$ = "P2P4"                  '3/4 Auftakt
160 M1$ = "L4C DCFE L2DL4C"       'Leerstellen trennen Takte
170 M2$ = "E EEAG L2FL4E"
180 M3$ = "E EEFE L2DP4"
190 M4$ = "L4E EEFE DDCP4"
200 PLAY TYP$
210 PLAY "XM0$;XM1$;XM2$;XM3$;XM1$;XM2$;XM4$;"
220 INPUT "Noch einmal wiederholen (J/N)"; W$
230 IF W$ = "J" THEN 110
240 END
```

Ein weiteres Beispiel zeigt, wie Noten ueber die Tastatur gespielt werden koennen. Wird z.B. die Taste "C" gedrueckt, so wird das mittlere "C" gespielt, wird die Taste "F" gedrueckt, ertoent das "F".

Bei laengerem Druecken der Taste bleibt der Ton "stehen". Die Eingabe von "0" (Null) schaltet diese Musik-Tastenbelegung wieder ab.

```
 50 '*********************************************************
 60 '
 70 '           Programm NOTEN
 80 '           Noten ueber Tastatur spielen
 90 '
100 '*********************************************************
110 PRINT "Eingabe: Noten C, D, E, F, G, A, H oder
         0 / fuer Ende"
120 WHILE NOTE$ <> "0"
130   NOTE$ = INKEY$
140   IF NOTE$ = ""  THEN 130
150   IF NOTE$ = "0" THEN 270
160   IF NOTE$ = "C" THEN FREQ = 523: GOTO 250
170   IF NOTE$ = "D" THEN FREQ = 587: GOTO 250
180   IF NOTE$ = "E" THEN FREQ = 659: GOTO 250
190   IF NOTE$ = "F" THEN FREQ = 698: GOTO 250
200   IF NOTE$ = "G" THEN FREQ = 784: GOTO 250
210   IF NOTE$ = "A" THEN FREQ = 880: GOTO 250
220   IF NOTE$ = "H" THEN FREQ = 988: GOTO 250
230   PRINT:  PRINT "Eingabe C,D,E,F,G,A,H oder 0 (=Ende)"
240   GOTO 270
250   SOUND FREQ,6
260   PRINT NOTE$; " ";
270 WEND
280 END
```

Das folgende Beispiel demonstriert, wie parallel zu einem Prozess eine Hintergrundmusik gespielt werden kann. Dazu ist der Parameter **MB** anzugeben.
Fuer den Hintergrund kann eine Folge bis zu 32 Noten angegeben werden.

```
 50 '*********************************************************
 60 '
 70 '           Programm HINMUSIK
 80 '           Musik im Hintergrund
 90 '
100 '*********************************************************
110 FOR Z=1 TO 3
120   PLAY "MB02 T220L4CDEF L2GG L4AAA L1G L4AAAA L1G"
130   FOR M = 1 TO 200
140 PRINT "Musik im Hintergrund"
150   NEXT M: PRINT
160   NEXT Z
170 PLAY "L4FFFF L2EE L4DDDD L1C"
180 END
```

10. Datenfernverarbeitung

In diesem Kapitel wird beschrieben, wie die Kopplung mit anderen Computern oder Peripheriegeraeten ueber die serielle Schnittstelle V.24 fuer asynchrone Datenfernverarbeitung unterstuetzt wird.
Diese Kopplung wird ueber den Adapter fuer serielle Kommunikation mit zwei V.24-Anschluessen ermoeglicht, der asynchron eingestellt werden muss.
Seriell bedeutet, dass Daten Bit fuer Bit gesendet oder empfangen werden.
Asynchron bedeutet, dass zeichenweise uebertragen wird und die einzelnen Zeichen von Start- und Stopbits eingeschlossen sind.

10.1. Serielle Kommunikationsparameter

Baud-Rate
Baudrate ist die Geschwindigkeit, mit der Daten uebertragen werden. Dieser Parameter wird ausgewiesen in Bit pro Sekunde (bps). Typisch sind Geschwindigkeiten von 300 bis 9600 bps.

Anzahl Datenbits
Die Anzahl der Datenbits legt fest, aus wieviel Bits ein Datenwort besteht (5,6,7 oder 8).
In der seriellen Kommunikation wird jedes Datenwort uebertragen als eine Folge von Bits.

Paritaet
Es ist eine Methode, Fehler in der Datenkommunikation festzustellen. Das Paritaetsbit wird an das Ende des Datenwortes angefuegt.
Es gibt verschiedene Moeglichkeiten, das Paritaetsbit zu berechnen:

EVEN: Das Paritaetsbit wird so gesetzt, dass die Summe aller Datenbits eine gerade Zahl ist.
Z.B. wird ein 7-bit-Datenwort mit gerader Paritaet ausgegeben.
Ist das Datenwort 1010010, dann wird das Paritaetsbit auf 1 gesetzt, damit die Summe aller Datenbits des Wortes gerade ist. Das endgueltige Datenwort ist jetzt 10100101.
Ist das Datenwort 0001010, dann wird das Paritaetsbit 0 sein. Das komplette Datenwort ist nun 00010100.

ODD: Das Paritaetsbit wird so gesetzt, dass die Summe aller Datenbits im Wort (einschliesslich Paritaetsbit) eine ungerade Zahl ist.

MARK: Das Paritaetsbit ist immer 1.

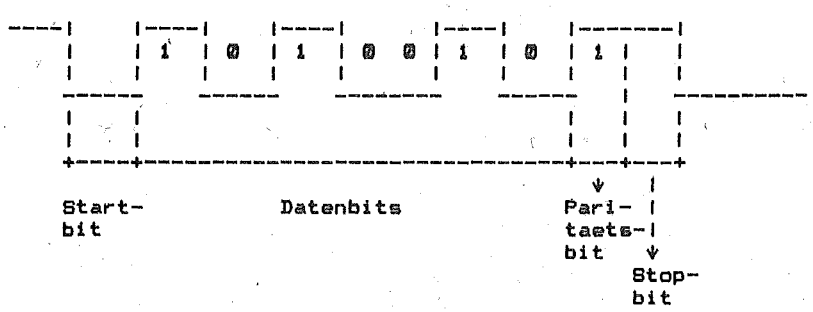
SPACE: Das Paritaetsbit ist immer 0.

NONE: Es wird kein Paritaetsbit am Ende des Datenwortes angefuegt, d.h., beim Senden und Empfangen von Daten wird nicht auf Bit-Paritaet geprueft.

Stop-Bit
In der asynchronen seriellen Kommunikation werden 1, 1 1/2 oder 2 Stop-Bit an das Ende jeden Datenwortes angefuegt. Dieses Bit teilt dem Empfaenger mit, wo das Datenwort zu Ende ist.

Beispiel:

Es wird ein 7-Bit-Datenwort (1010010) mit gerader Paritaet und einem Stop-Bit dargestellt.



10.2. Anweisungen und Funktionen fuer die Datenfernverarbeitung

Zu Einzelheiten ist bei der jeweiligen Anweisung oder Funktion in der Anwenderdokumentation BASIC-Interpreter BASI, Kapitel 7, nachzuschlagen.

10.2.1. Eroeffnen einer Datenfernverarbeitungsdatei

Durch **OPEN "COM n:** wird ein Puffer fuer die Ein-/ Ausgabe in der gleichen Art wie bei der Anweisung OPEN fuer Diskettendateien angelegt.
Dabei werden die Parameter fuer Geschwindigkeit, Paritaet, Anzahl Datenbits und Stop-Bits sowie Satzlaenge festgelegt.
"n" bedeutet 1 oder 2 und legt die Nummer des Anschlusses auf dem Adapter fuer serielle Kommunikation fest.

10.2.2. Datenfernverarbeitungsein-/-ausgabe

Da jeder Datenfernverarbeitungsanschluss als Datei eroeffnet wird, sind die Ein-/Ausgabeanweisungen und Funktionen der Dateiarbeit, die den Zugriff auf Diskettendateien ermoeglichen, auch fuer die Datenfernverarbeitung gueltig.

Fuer die sequentielle Ein-/Ausgabe sind dies:

```
INPUT#
LINE INPUT#
PRINT#
PRINT# USING
WRITE#
```

Fuer Ein-/Ausgabe fester Laenge werden die Anweisungen

```
GET
PUT
```

verwendet.
Hier werden anstelle der Satz-nummer die Anzahl der Bytes angegeben, die in oder aus dem Dateipuffer uebertragen werden sollen.
Ausserdem koennen noch die Funktionen

```
LOC(f)
LOF(f)
EOF(f)
INPUT$
```

genutzt werden (f=Dateinummer).

10.3. Operationen der Steuersignale

Die Ausgabe des asynchronen Datenfernverarbeitungsanschlusses entspricht dem V.24-Standard fuer Schnittstellen zwischen Datenendeinrichtungen und Datenuebertragungseinrichtungen. Dieser Standard definiert eine Anzahl Steuersignale, die vom Personalcomputer uebertragen oder empfangen werden, um den Austausch von Daten zwischen einem anderen Computer oder einer peripheren Einheit zu steuern. Diese Signale sind Geraetesteuerspannungen, die entweder Ein (groesser als +3 Volt) oder Aus (kleiner als -3 Volt) sind.

10.3.1. Steuerung der Ausgabesignale mit OPEN

Wenn man BASIC im Personalcomputer startet, sind die Leitungen RTS (Request To Send - Sendeteil einschalten) und DTR (Data Terminal Ready - Datenstation betriebsbereit) ausgeschaltet. Wird die Anweisung OPEN "COM... ausgefuehrt, werden grundsaetzlich beide Leitungen eingeschaltet. Jedoch kann man die Auswahl RS in der Anweisung OPEN "COM... angeben, um das RTS-Signal zu unterdruecken. Die Leitungen bleiben eingeschaltet, bis die Datenfernverarbeitungsdatei geschlossen wird (durch CLOSE, END, NEW, RESET, SYSTEM oder RUN, ohne die Angabe R). Sogar wenn in der Anweisung OPEN "COM... ein Fehler auftritt (wie unten beschrieben), wird die DTR-Leitung (und RTS-Leitung, falls anwendbar) eingeschaltet und bleibt bereit. Dies erlaubt eine Wiederholung des OPEN, ohne zuvor ein CLOSE auszufuehren.

10.3.2. Benutzung der Eingabesteuersignale

Ist eine der Leitungen CTS (Clear To Send - Sendebereitschaft) oder DSR (Data Set Ready - Betriebsbereitschaft) ausgeschaltet, wird die Anweisung OPEN "COM... nicht ausgefuehrt. Nach einer Sekunde uebergibt BASIC den Fehler "Device Timeout" (Einheitenzeitsperre) (Fehlercode 24). Der CD (Carrier Detect - Empfangssignalpegel) kann entweder Ein oder Aus sein. Er hat auf den Ablauf des Programms keinen Einfluss.

Man kann jedoch angeben, wie diese Leitungen mit Hilfe der Auswahlen RS, CS, DS und CD in der Anweisung OPEN "COM... zu testen sind. Siehe unter Anweisung OPEN "COM... in Kapitel 7 der Anwenderdokumentation des BASIC-Interpreters BASI.
Wird eines der zu testenden Signale ausgeschaltet, waehrend das Programm ausgefuehrt wird, werden die Ein-/Ausgabeanweisungen fuer die Datenfernverarbeitungsdatei nicht ausgefuehrt. Wird z.B. die Anweisung PRINT# ausgefuehrt, nachdem die Leitung CTS oder DSR ausgeschaltet wurde, erscheint der Fehler "Device Fault" (Einheitenfehler) (Fehlercode 25) oder "Device Timeout" (Einheitenzeitsperre) (Fehlercode 24). Die Leitungen fuer RTS und DTR bleiben eingeschaltet, wenn einer dieser Fehler auftritt.

10.4. Beispiel

Mit dem folgenden Programm kann der Personalcomputer als konventionelle "dumme" Datenstation verwendet werden. In diesem Beispiel werden eine 300-bps-Leitung und ein Eingabepuffer von 256 Bytes angenommen (die Auswahl /C: wurde beim Laden von BASIC nicht benutzt).

```
  10 REM Beispiel "dumme" Datenstation
  20 'setzen Bildschirm auf schwarz/weiss Textmodus
  30 ' und Breite auf 40
  40 SCREEN 0,0:WIDTH 40
  50 'Funktionstastenanzeige ausschalten; Bildschirm
  60 '  loeschen; Dateien abschliessen
  70 KEY OFF: CLS: CLOSE
  80 'numerische Variablen als Integer definieren
  90 DEFINT A-Z
 100 'wahr und falsch definieren
 110 FALSE=0: TRUE= NOT FALSE
 120 'die Zeichen XON und XOFF definieren
 130 XOFF$=CHR$(19): XON$=CHR$(17)
 140 'Verbindung eroeffnen fuer Dateinr. 1, 300 bps,
 150 'Paritaet gerade, 7 Datenbits
 160 OPEN "COM1:300,E,7" AS #1
 170 'Benutzen Bildschirm als Datei
 180 OPEN "SCRN:" FOR OUTPUT AS 2
 190 'Kursor einschalten
 200 LOCATE ,,1
 400 PAUSE = FALSE: ON ERROR GOTO 9000
 490 '
 500 'senden Tastatureingabe zur DFV-Leitung
 510 B$=INKEY$: IF B$<>"" THEN PRINT #1,B$;
 520 'falls keine Zeichen im DFV-Puffer, Eingabe pruefen
 530 IF EOF(1) THEN 510
 540 'falls Puffer mehr als halbvoll, Pausezeichen
 550 '   fuer Eingabeunterbrechung setzen, XOFF zum Host-
 560 '   rechner senden, um Uebertragung zu stoppen
 570 IF LOC(1)>128 THEN PAUSE=TRUE: PRINT #1, XOFF$;
 580 'Inhalt des DFV-Puffers lesen
 590 A$=INPUT$(LOC(1),#1)
 600 'Zeilenvorschub zur Vermeidung von Doppellehrzeilen
 610 '  bei Anzeige der Bildschirmeingabe unterdruecken
 620 LFP=0
 630 LFP=INSTR(LFP+1,A$,CHR$(10)) 'suchen Zeilenvorschub
 640 IF LFP>0 THEN MID$(A$,LFP,1)=" ": GOTO 630
 650 'DFV-Eingabe anzeigen und auf weitere Eingabe pruefen
 660 PRINT #2,A$;: IF LOC(1)>0 THEN 570
 670 'ist Uebertragung durch XOFF unterbrochen, durch
 680 '  senden von XON wiederaufnehmen
 690 IF PAUSE THEN PAUSE=FALSE: PRINT #1,XON$;
 700 'wieder auf Tastatureingabe pruefen
 710 GOTO 510
9000 'bei Fehler die Fehlernr. anzeigen und erneut versuchen
9010 PRINT "FEHLERNR.";ERR: RESUME
```

Programmhinweise:

- Asynchrone Datenfernverarbeitung erlaubt Zeichenein-/-ausgabe als Zeichen- oder Blockein-/-ausgabe. Deshalb werden alle Anweisungen PRINT (entweder zur Datenfernverarbeitungsdatei oder zum Bildschirm) mit einem Semikolon (;) beendet. Dabei wird die Zeilenschaltung unterdrueckt, die normalerweise am Ende der auszugebenden Werteliste erfolgt.

- Die Zeile 90, in der alle numerischen Variablen als Integer definiert sind, wurde deshalb so programmiert, weil jedes Programm, das schnell laufen soll, in Schleifen mit Integer-Zahlen - falls moeglich - arbeiten sollte.

- Man beachte, dass in Zeile 510 INKEY$ eine leere Zeichenkette uebergeben wird, wenn kein Zeichen mehr verfuegbar ist.

11. Prozessorarbeit

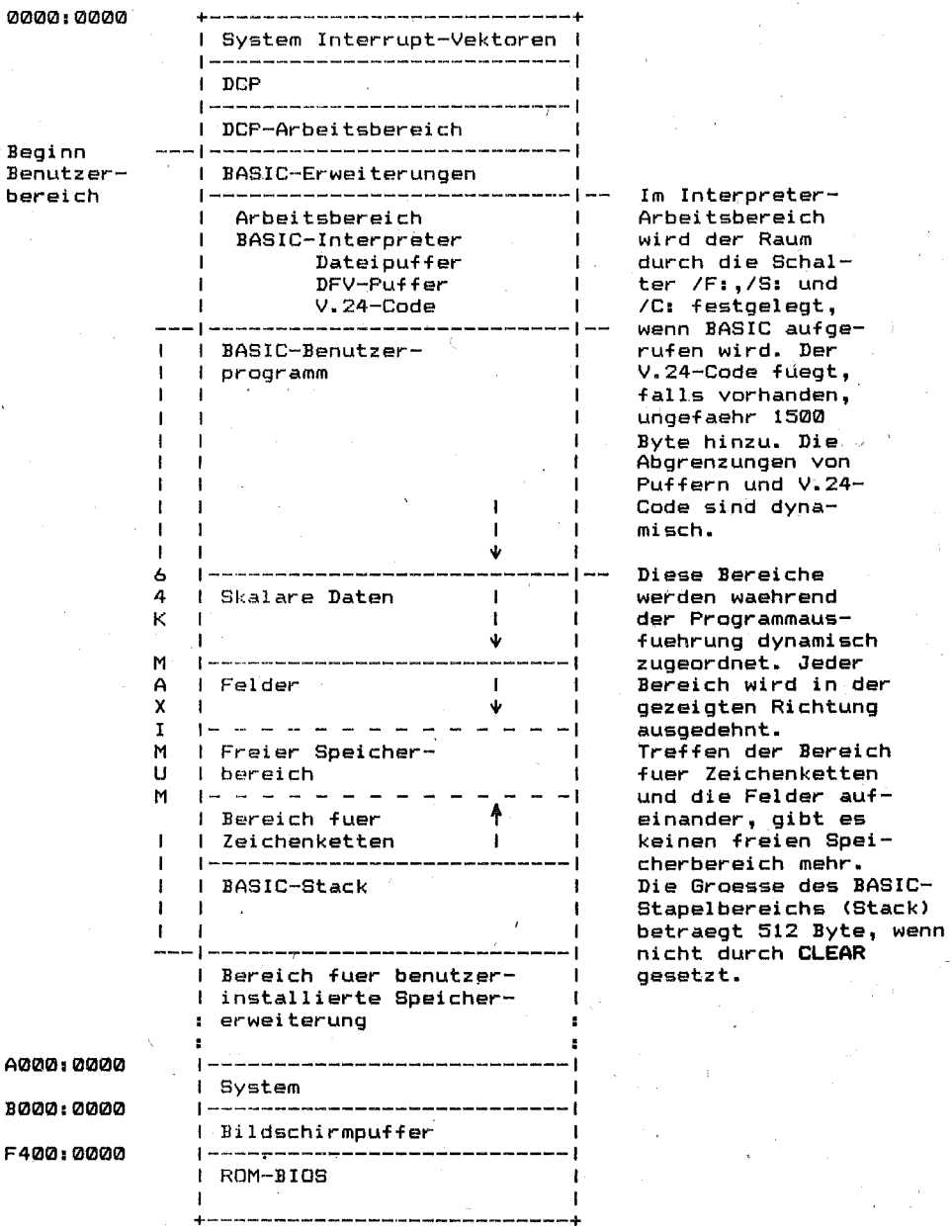
In diesem Kapitel wird beschrieben, wie BASIC Verbindungen zu Unterprogrammen in Maschinensprache (Assemblerprogramme) herstellt. Insbesondere wird beschrieben,

- wo sich Unterprogramme in Maschinensprache im Hauptspeicher befinden,
- wie BASIC eine Verbindung zu Unterprogrammen in Maschinensprache herstellt,
- wie Unterprogramme in Maschinensprache aufgerufen und ausgefuehrt sowie Parameter uebergeben werden.

Dieses Kapitel ist fuer Programmierer gedacht, die Erfahrungen in der Assemblersprache besitzen.

11.1. Hauptspeicherbelegung

Die Adressen sind in der folgenden Hexadezimalform **Segment:Relativzeiger** angegeben.


0000:0000
System Interrupt-Vektoren
DOP
DOP-Arbeitsbereich
Beginn Benutzerbereich
BASIC-Erweiterungen
Arbeitsbereich BASIC-Interpreter
Dateipuffer
DFV-Puffer
V.24-Code
BASIC-Benutzerprogramm
Skalare Daten
Felder
MAXIMUM
Freier Speicherbereich
Bereich fuer Zeichenketten
BASIC-Stack
Bereich fuer benutzerinstallierte Speichererweiterung
A000:0000
System
B000:0000
Bildschirmpuffer
F400:0000
ROM-BIOS
Im Interpreter-Arbeitsbereich wird der Raum durch die Schalter /F:,/S: und /C: festgelegt, wenn BASIC aufgerufen wird. Der V.24-Code fuegt, falls vorhanden, ungefaehr 1500 Byte hinzu. Die Abgrenzungen von Puffern und V.24-Code sind dynamisch.
Diese Bereiche werden waehrend der Programmausfuehrung dynamisch zugeordnet. Jeder Bereich wird in der gezeigten Richtung ausgedehnt. Treffen der Bereich fuer Zeichenketten und die Felder aufeinander, gibt es keinen freien Speicherbereich mehr. Die Groesse des BASIC-Stapelbereichs (Stack) betraegt 512 Byte, wenn nicht durch CLEAR gesetzt.

11.2. Hauptspeicherzuordnung fuer eigene Unterprogramme

BASIC verwendet standardmaessig den ganzen verfuegbaren Hauptspeicherbereich vom Startpunkt des Datenbereichs bis zu einem Maximum von 64K Byte. In diesem Bereich sind BASIC-Programme, Daten, der Interpreter-Arbeitsbereich sowie der BASIC-Stack enthalten.

Fuer Unterprogramme in Maschinensprache kann der Hauptspeicherplatz innerhalb oder ausserhalb dieses BASIC-64K-Arbeitsbereichs angelegt werden. Wo diese Unterprogramme untergebracht werden, haengt von

- der im System installierten Speichergroesse,
- der Groesse des BASIC-Programms und
- den Charakteristika des Unterprogramms in Maschinensprache ab.

Nur die hoechsten Hauptspeicherplaetze koennen fuer Unterprogramme reserviert werden. Will man z.B. die hoechsten 4K Bytes des BASIC-64K-Arbeitsbereichs fuer Unterprogramme in Maschinensprache reservieren, kann man dies folgendermassen tun:

```
10 CLEAR,&HF000
```

oder BASIC mit dem DCP-Befehl starten:

```
BASIC /M: &HF000
```

Jede dieser Anweisungen schraenkt die Groesse des BASIC-Arbeitsbereichs auf hex F000 (60K) Bytes ein, so dass man die hoechsten 4K Bytes fuer Unterprogramme in Maschinensprache nutzen kann.

Es gibt mehrere Methoden, Unterprogramme in Maschinensprache zu installieren. Es wird empfohlen, sie mit einer der folgenden Moeglichkeiten zu speichern:

- Als residenten Teil von DCP, der vor dem Aufruf von BASIC geladen wird.

- Als Integer-Feld innerhalb des BASIC-Datensegments.

- Oberhalb des BASIC-Datensegments.

11.2.1. Unter DCP geladene Unterprogramme fuer BASIC

Diese Methode wird empfohlen, um von BASIC aus mit Unterprogrammen in Maschinensprache zu kommunizieren, da hierdurch die Kompatibilitaet der Programme mit DCP und BASIC gewaehrleistet ist. Durch diese Methode koennen ausserdem moegliche Fehlerquellen beim Kommunizieren mit den Unterprogrammen vermieden werden.

Vorteile:

- Diese Methode ermoeglicht die Arbeit mit Unterprogrammen mit mehreren Segmenten (bis zu 64K) und mit nicht selbstverschiebbaren Unterprogrammen, die mit den Anweisungen **BLOAD** oder **DATA** nicht verarbeitet werden koennten.

- Das Unterprogramm wird ein residenter Teil von DCP, ohne dass es dem Zugriff von BASIC entzogen oder von BASIC oder DCP Platz fuer das Unterprogramm reserviert oder geschuetzt werden muesste.

Ueberlegungen:

- DCP-geladene Routinen bleiben nach dem Laden resident und koennen nur durch Laden eines neuen Programmes entfernt werden. Zusaetzlich koennen mehrere Kopien des Unterprogramms resident werden, wenn dieses mehrere Male aufgerufen wird.

- Wenn Unterprogramme, die mit **INT27** resident sind, aufgerufen werden, kann eine gewisse Ueberlagerung stattfinden. Zusammen mit dem residenten Teil des Codes erhaelt der Benutzer eine Kopie der DCP-Umgebung mit einem Minimum von 128 Byte und einem PSP (Programm-Segment-Praefix) von 256 Byte.

- Es ist nicht immer moeglich, Unterprogramme getrennt von BASIC zu starten, da diese sofort beim Start einmal abgearbeitet werden.

11.2.2. Unterprogramme innerhalb des BASIC-Datensegments

Relativ kurze Unterprogramme in Maschinensprache koennen am leichtesten und sichersten gespeichert werden, indem sie in ein Integer-Feld in BASIC geladen werden.

Vorteile:

- Da innerhalb des BASIC-Datensegments gearbeitet wird, ist es nicht erforderlich, auf ein externes Segment fuer das Unterprogramm mit der Anweisung **DEF SEG** zuzugreifen.

- Weil das Unterprogramm in das BASIC-Datensegment und in einen Datenbereich geladen wird, der von BASIC zugeordnet und verwaltet wird, besteht keine Gefahr, dass das Programm den kritischen Bereich innerhalb oder ausserhalb von BASIC ueberschreitet oder der Code ausserhalb von BASIC die Daten ueberlagert.

Ueberlegungen:

- Wenn die BASIC-Anwendung den groessten Teil des verfuegbaren Datenbereichs innerhalb des BASIC-Datensegments benoetigt, wird diese Methode nicht empfohlen. Hier wird zusaetzlicher Platz fuer Daten benoetigt, wenn das Unterprogramm einem Integer-Feld zugeordnet wird.

- Das Unterprogramm sollte selbst-verschiebbar sein. Es darf also keinerlei Bezug auf ein Segment enthalten, dessen Wert davon abhaengt, wohin das Unterprogramm geladen wurde. Zeiger muessen also in relativen Ausdruecken angegeben werden, so dass alle Referenzen innerhalb des Unterprogramms bleiben.

11.2.3. Unterprogramme oberhalb des BASIC-Datensegments

Diese Methode, mit Unterprogrammen in Maschinensprache zu kommunizieren, sollte nur verwendet werden, wenn dies unbedingt erforderlich ist, da der Nutzer evtl. selbst Operationen zur Speicherverwaltung vornehmen muss.

Vorteile:

- Es wird kein Platz innerhalb des BASIC-Datensegments fuer die Routinen benutzt.

- Bei einem System mit einem grossen Speicher koennen Unterprogramme geladen werden, die nicht in das BASIC-Datensegment passen.

Ueberlegungen:

Wird ein Unterprogramm ausserhalb des BASIC-Datensegments geladen, muss der vom Unterprogramm belegte Speicherbereich vom Bediener selbst verwaltet werden, um das Funktionieren des Programms zu gewaehrleisten:

- Die Groesse des adressierbaren Speicherbereichs von BASIC kann durch Eingabe der Anweisung **CLEAR** oder des Schalters **/M:** in der BASIC-Befehlszeile eingegrenzt werden. Dadurch kann das Unterprogramm nicht von BASIC-Daten ueberlagert werden.

- Innerhalb des Programms muss die Segment-ID berechnet und an die Anweisung **DEF SEG** uebergeben werden.

- Wenn die Groesse des verfuegbaren Speichers errechnet wird, muessen Routinen, die beim Aufrufen des Unterprogramms aktiv sind (z.B. PRINT, MODE usw.), beruecksichtigt werden.

11.3. Kommunikation zwischen BASIC und Unterprogrammen in Maschinensprache

BASIC unterstuetzt zwei Schnittstellen, um vom Anwenderprogramm aus Unterprogramme in Maschinensprache aufzurufen:

die Anweisung **CALL** und die Funktion **USR**.

Es wird empfohlen, die Anweisung **CALL** zu nutzen. In allen Beispielen wird diese Anweisung verwendet.
Informationen zur Funktion **USR** siehe Sprachbeschreibung **BASI**.

Beim Aufrufen des Unterprogramms ist folgendes zu beachten:

- Zu Beginn werden die Segmentregister **DS**, **ES** und **SS** auf den gleichen Segmentwert gesetzt, und zwar auf die Segment-ID des BASIC-Datensegments. Dies ist der Standardwert fuer **DEF SEG**.

- Das Codesegmentregister **CS** enthaelt die Segment-ID des letzten Wertes, der in der Anweisung **DEF SEG** angegeben wurde. Standardmaessig ist dies die BASIC-Segment-ID.
  Wurde **DEF SEG** nicht spezifiziert oder spezifizierte die letzte **DEF SEG** keinen Ueberschreibungswert, ist der Wert in **CS** der gleiche wie in den anderen drei Segmentregistern.

- Beim Einsprung gibt das Stapelspeicherzeigerregister (Stackpointer SP) an, dass im Stapelspeicher (Stack), der fuer das Unterprogramm benutzt werden kann, mindestens acht Worte (16 Byte) verfuegbar sind.
  Weiterer Stapelspeicherbereich ist fuer Unterbrechungen (wie z.B. TIMER) und DCP- oder BIOS-Aufrufe, die vom Unterprogramm in Maschinensprache aufgerufen werden, reserviert. Wird fuer das Unterprogramm mehr Stack benoetigt, kann vom Benutzer ein eigener Stack erstellt werden (mindestens 128 reservierte Byte fuer Systembenutzung oberhalb der Erfordernisse fuer das Unterprogramm). Bei der Rueckkehr zu DCP muessen das Stapelspeichersegment (Stacksegment SS) und die Stackpointer (SP) wieder die gleichen Werte haben, die sie hatten, als das Unterprogramm von BASIC aus aufgerufen wurde. Dass heisst, sie muessen vorher gerettet werden, bevor zurueckverzweigt wird.

Ist der Eingabeparameter eine Zeichenkette, ist der uebergebene Wert der Zeiger eines Drei-Byte-Bereichs im BASIC-Datensegment, der **Zeichenkettenbeschreibung** genannt wird.

Byte 0 enthaelt die Laenge der Zeichenkette (0 bis 255).

Byte 1 enthaelt die niedrigen acht Bit des Relativzeigers der Zeichenkette im Bereich fuer Zeichenketten innerhalb des BASIC-Datensegments.

Byte 2 enthaelt die hoechsten acht Bit des Zeigers der Zeichenkette im Bereich fuer Zeichenketten innerhalb des BASIC-Datensegments.

Achtung:

Keines der drei Byte der **Zeichenkettenbeschreibung** darf vom Unterprogramm geaendert werden.

Das Unterprogramm darf den **Inhalt** der Zeichenkette aendern, aber nicht ihre **Laenge**.
Veraendert das Unterprogramm den Inhalt einer Zeichenkette, kann dies das Programm veraendern. Durch das folgende Beispiel wird moeglicherweise die Zeichenkette "TEXT" in einem BASIC-Programm veraendert:

```
A$ = "TEXT"
CALL SUBRT(A$)
```

Im folgenden Beispiel wird das Programm nicht veraendert, weil BASIC infolge der Zeichenkettenverknuepfung die Zeichenkette in den Zeichenkettenbereich kopiert, wo er geaendert werden kann, ohne den Originaltext zu beeinflussen.

```
A$= "BASIC"+""
CALL SUBRT(A$)
```

Beim Verlassen des Unterprogramms muss beachtet werden

- alle Unterbrechungsfunktionen, die vom Unterprogramm inaktiviert wurden, werden wieder aktiviert.

- **RET** verwenden, da jeder Unterprogrammaufruf von BASIC eine **FAR**-Prozedur ist (Siehe Assembler-Handbuch: Pseudo-Operation "PROC").

- alle Segmentregister und den Stackpointer SP auf den alten Stand bringen. Alle anderen Register und Zeiger koennen **geaendert bleiben.**

11.4. Die Anweisung CALL

Maschinenspracheunterprogramme koennen mit Hilfe der BASIC-Anweisung **CALL** aufgerufen werden. Das Format der Anweisung **CALL** ist:

CALL numerische Variable [(Variablenliste)]

numerische Variable ist der Name einer numerischen Variablen. Ihr Wert ist der Relativzeiger des Segments, das durch **DEF SEG** gesetzt wurde, d.h. der Startpunkt des Unterprogramms im Hauptspeicher, das aufgerufen wird.

Variablenliste enthaelt die Variablen, getrennt durch Komma, die als Argument an das Unterprogramm uebergeben werden (die Argumente duerfen keine Konstanten sein).

Die Ausfuehrung der Anweisung **CALL** bewirkt folgendes:

- Fuer jeden Parameter in der Variablenliste wird die Adresse der Variablen in den Stack gebracht. Die Adresse wird als Zwei-Byte-Zeiger im BASIC-Datensegment angegeben. Ueber die Parameter koennen Werte an BASIC uebergeben werden, indem die Werte der Variablen, auf die in der Parameterliste gezeigt wird, geaendert werden.
  Das aufgerufene Unterprogramm muss wissen, wieviele Parameter uebergeben wurden. Parameter werden dadurch angesprochen, dass ein positiver Zeiger zu **BP** addiert wird, nachdem das aufgerufene Unterprogramm den aktuellen Stackpointer nach **BP** uebertraegt. Die ersten Instruktionen im Unterprogramm muessen wie folgt aussehen:

```
PUSH BP     ;RETTEN BP
MOV BP,SP   ;UEBERGEBEN SP NACH BP
```

Der Zeiger irgendeines Parameters im Stack wird wie folgt berechnet:

Zeiger von BP = 2*(n-m)+6

n ist die Gesamtzahl der uebergebenen Parameter

m ist die Position eines spezifischen Parameters in der Parameterliste der BASIC-Anweisung **CALL**. m muss sich im Bereich von 1 bis n befinden.

Im Macro-Assembler stellt die Pseudo-Operation **"STRUC"** die normale Methode dar, den Inhalt des Stack zu definieren, wobei die Position der Parameter gezeigt wird.

```
FRAME     STRUC
SAVE BP   DW      ?
RET OFF   DW      ?
RET SEG   DW      ?
PARMN     DW      ?   ;Zeiger zum n. Parameter
          :
          :
          :
PARM1     DW      ?   ;Zeiger zum 1. Parameter
FRAME     ENDS

PARMSIZE EQU OFFSET PARM1-OFFSET RETSEG ;
```

Achtung:

Es muss sichergestellt sein, dass die Parameter in der Anweisung **CALL** in Anzahl, Typ und Laenge mit den Parametern, die vom Unterprogramm erwartet werden, uebereinstimmen; z.B. (8-Byte-)Werte doppelter Genauigkeit.

- Die Steuerung wird an das Unterprogramm in Maschinensprache mit dem in der ersten **DEF SEG**-Anweisung angegebenen Segment und dem in der Anweisung **CALL** angegebenen Zeiger uebergeben.

- Das Rueckkehrsegment **ID**, im Register **CS** angegeben, und der Zeiger werden in den Stack gebracht.

- Wird das Unterprogramm mit einer Anweisung **CALL**, die Parameter angibt, aufgerufen, muss das Unterprogramm mit **RET N** zurueckverzweigen, wobei **N** zweimal die Anzahl der Parameter in der Liste ist. Dadurch wird der Stack in den urspruenglichen Zustand gebracht.

- Eine Anweisung **CALL** muss nicht immer Parameter haben. Wenn sich die Routine in einem Integerfeld befindet, koennen Parameter direkt in das Feld gebracht werden anstatt mit der Anweisung **CALL**.

11.5. Laden und Aufrufen von Unterprogrammen in Maschinensprache

Im folgenden werden Moeglichkeiten angegeben, wie ein Unterprogramm in Maschinensprache in den Hauptspeicher geladen werden kann:

- mit der Anweisung **POKE** aus dem BASIC-Programm in den Hauptspeicher,

- mit **BLOAD** von einer Diskettendatei,

- von einer Diskettendatei als einen residenten DCP-Teil.

11.5.1. Laden mit POKE und Zuordnung zu einem Bereich

Man kann Unterprogramme in Maschinensprache mit Hilfe der Anweisung **POKE** vom Benutzerprogramm direkt in den Hauptspeicher laden. Auf diese Weise wird das Unterprogramm Teil des BASIC-Programms.

Vorteile:

- Fuer diese Methode wird kein Assembler benoetigt.

- Da sich alle Codes in einer Datei befinden, muessen nicht zwei oder mehr Dateien erstellt werden.

Ueberlegungen:

- Das Codieren jeder Anweisung des Unterprogramms in Hexadezimalwerten ist sehr langwierig und fehleranfaellig und sollte deshalb nur fuer kurze Unterprogramme verwendet werden.

- Wird das Unterprogramm mit **POKE** ausserhalb des BASIC-Datensegments geladen, muss der Speicherbereich vom Benutzer verwaltet werden, um Platz fuer das Unterprogramm vor BASIC oder DCP zu schuetzen.

Zuordnen eines Unterprogramms im Speicher

Mit dieser Prozedur wird der Maschinencode einem Integer-Feld zugeordnet und das Unterprogramm aufgerufen:

1. Festlegen des Maschinencodes fuer das Unterprogramm.

2. Anschliessend muss ein Integer-Feld auf die benoetigte Groesse dimensioniert werden. Dabei muss beachtet werden, dass eine Integer-Zahl 2 Byte des Speichers benoetigt, so dass die Groesse des erstellten Feldes der Haelfte der Byte des Unterprogramms entspricht.

   Es gibt mehrere Gruende die fuer die Benutzung eines Integer-Feldes (2 Byte pro Element) sprechen anstatt eines Feldes einfacher Genauigkeit (4 Byte pro Element) oder doppelter Genauigkeit (8 Byte pro Element). Wenn ein Unterprogramm einem Integer-Feld zugeordnet wird, bleibt nie mehr als 1 Byte des Hauptspeichers ungenutzt; bei doppelter Genauigkeit koennen 7 Byte ungenutzt bleiben. Bei Verwendung von Integer-Feldern erfolgt eine Wortzuordnung zu den Feldelementen und keine Bytezuordnung, wobei ein Wort aus zwei Byte besteht. Wenn jedem Feldelement vier Woerter zugeordnet wuerden, koennten leicht Fehler auftreten, da die CPU im System den Maschinencode in einem Niedrigbyte-Hochbyte-Format (L-Byte, H-Byte) erwartet.

Im folgenden Beispiel werden Informationen ueber die Anordnung der Byte, wie sie in der Assembler-Liste erscheinen, angegeben.

3. Definiert alle Skalare, die in der Anwendung benutzt werden. Nachdem ein Feld dimensioniert wurde, verschiebt jeder erstellte Skalarwert das Feld im Speicher nach oben. Siehe auch "Hauptspeicherbelegung" in diesem Kapitel.

4. Jetzt kann der Maschinencode den Feldelementen Wort fuer Wort zugeordnet werden. Dazu den hexadezimalen Wert (&Hxx-Format) jedes Bytes des Codes in mehrere **DATA**-Anweisungen eintragen. Dabei ist zu beachten, dass das System das niedrigste Byte zuerst speichert und anschliessend die hoeherwertigen Byte, also Daten im Niedrigbyte-Hochbyte-Format speichert.

5. Eine **FOR-NEXT-Schleife** ausfuehren, die jedes Daten-Byte liest und in das Feld eintraegt, das fuer das Unterprogramm ausgewaehlt wurde.

6. Der Zeiger des Unterprogramms innerhalb des BASIC-Datensegments sollte mit der Funktion **VARPTR** definiert werden.

7. Das Unterprogramm wird mit der Anweisung **CALL** oder der Funktion **USR** aufgerufen.

Beispiel:

```
10 DEFINT A-Z: OPTION BASE 1: DIM ARRAY(3)
20 DATA &HCD55              ;REM 55H Push BP
30 DATA &H5D05              ;REM CD05H INT 5
40                          ;REM 5DH POP BP
50 DATA &H90CB              ;REM CBH RET; 90H NOP
70 FOR I = 1 TO 3: READ ARRAY(I): NEXT I
80 SUBRT = VARPTR(ARRAY(1)): CALL SUBRT
```

In diesem Beispiel wird demonstriert, wie mit Hilfe einer BASIC-Routine ein Hardcopy vom Bildschirm auf den Drucker ausgefuehrt wird durch Nutzen des DCP-Interrupts **INT 5H**. Dabei wird die Assembler-Subroutine in ein BASIC-Integer-Feld geladen.

Mit der Anweisung **DATA** werden Daten Wort fuer Wort in das Feld gelesen. Die Assembler-Liste listet die Byte in der Reihenfolge auf, in der sie im kommentierten Teil des Programms erscheinen. Das Programm aendert die Byte-Reihenfolge in den Worten um, wenn sie den Feldelementen zugeordnet werden, da die einzelnen Woerter im Niedrigbyte-Hochbyte-Format dargestellt sind.

Es wird empfohlen, **VARPTR** und **CALL** immer zusammen in eine Zeile zu schreiben, so dass **CALL** nie ausgefuehrt wird, ohne den Platz des Feldes mit dem Unterprogramm festzustellen.

Laden eines Unterprogramms in den Hauptspeicher mit Hilfe von POKE

Im folgenden Beispiel wird dasselbe Unterprogramm in Maschinensprache benutzt. Hier bewirkt die BASIC-Anweisung **POKE**, dass die Byte in einem Feld abgelegt werden. Die Reihenfolge der Bytes wird ebenfalls von **POKE** gesteuert.

Beispiel:

```
10 DEFINT A-Z:OPTION BASE 1:P=0:I=0:J=0:
   DIM ARRAY(3)
20 DATA &H55              ;REM 55H Push BP
30 DATA &HCD, &H05        ;REM CD05H INT 5
40 DATA &H5D              ;REM 5DH POP BP
50 DATA &HCB              ;REM CBH RET FAR
60 DATA &H90              ;REM 90H NOP
70 P=VARPTR(ARRAY(I)):FOR I=0 TO 4:READ J:
   POKE(P+I),J:NEXT I
80 SUBRT = VARPTR(ARRAY(1)): CALL SUBRT
```

Mit **POKE** koennen Daten irgendwo im Speicher abgelegt werden. Um diese Routine mit **POKE** in ein anderes Segment zu bringen, muesste Zeile 70 wie folgt aussehen:

```
DEF SEG = SEGMENT: FOR I=0 TO 4:READ J:
POKE(I,J):NEXT
```

Da das Unterprogramm nicht im BASIC-Datensegment abgelegt wird, wird **VARPTR** nicht benoetigt, um den Zeiger des Unterprogramms zu erhalten.

11.5.2. Aufrufen eines Unterprogramms von einer Datei mit BLOAD

Mit dem BASIC-Befehl **BLOAD** kann man eine Hauptspeicherabbilddatei direkt in den Hauptspeicher laden. Diese Datei kann eine Hauptspeicherabbilddatei sein, die mit **BSAVE** gespeichert wurde, oder es kann ein Unterprogramm in Maschinensprache sein, das mit einem **BLOAD**-Leitsatz in Maschinensprache umgewandelt, verkettet und mit **EXE2BIN** in eine **.COM**-Datei umgewandelt wurde. Weitere Angaben ueber **EXE2BIN** sind im DCP-Handbuch zu finden.
Unterprogramme in Maschinensprache koennen mit **BLOAD** in ein Integer-Feld oder oberhalb des BASIC-Datensegments geladen werden.

Vorteile:

- Wenn ein Unterprogramm in das BASIC-Datensegment geladen wird, wird die ganze Speicherverwaltung von BASIC durchgefuehrt, und der Benutzer braucht sich nicht mehr um das Funktionieren des Programms bemuehen.

- Werden Unterprogramme ausserhalb des BASIC-Datensegments geladen, kann ein Programm erstellt werden, dass die normale Grenze von 64K in BASIC ueberschreitet.

Ueberlegungen:

- Fuer diese Methode wird ein Assembler benoetigt.

- Ein Unterprogramm in Maschinensprache, das mit **BLOAD** in den Hauptspeicher geladen werden soll, muss zuvor mit **BSAVE** gespeichert worden sein. Dazu wird ein **7-Byte-BSAVE-Leitsatz** als erstes Segment des Unterprogramms verkettet. Diese 7 Byte dienen zum Laden; sie werden von **BLOAD** ueberprueft und anschliessend geloescht. Der Anfang des Programms im Hauptspeicher ist das erste Byte, das diesem Leitsatz folgt.

Im folgenden Beispiel wird dargestellt, wie ein Unterprogramm mit **BLOAD** in ein Integer-Feld in BASIC geladen und anschliessend aus BASIC aufgerufen wird. Das Unterprogramm addiert den Inhalt des ersten und des zweiten Parameters und legt das Ergebnis in den dritten Parameter ab.
Ausserdem wird der verschiebbare Code erklaert und eine schrittweise Erklaerung der Prozedur gegeben.

Folgende Schritte sind notwendig:

1. Unterprogramm umwandeln und verketten.

2. Die Datei mit **EXE2BIN** in eine **.COM**-Datei umwandeln.

3. Von BASIC aus ein Integer-Feld auf die benoetigte Groesse dimensionieren. (Die Groesse des Feldindex entspricht der Haelfte der vom Unterprogramm beanspruchten Byte).

4. Alle in der Anwendung benutzten Skalare definieren, einschliesslich aller Parameter, die mit **CALL** uebergeben werden koennen.

5. Den Relativzeiger des Feldes innerhalb des BASIC-Datenbereichs mit der Funktion **VARPTR** abfragen.

6. Das Unterprogramm mit **BLOAD** in das Feld laden.

7. Das Unterprogramm mit diesem Relativzeiger aufrufen.

Beispielprogramm:

Mit diesem Unterprogramm werden der Inhalt des ersten und des zweiten Parameters addiert und das Ergebnis in den dritten Parameter eingetragen. Dies ist ein praktisches Beispiel, wie Parameter an ein Unterprogramm, das sich in einem Integer-Feld befindet, uebergeben werden koennen. Parameter koennen uebergeben werden, indem sie als Elemente des Feldes gespeichert werden, in das das Unterprogramm mit **BLOAD** geladen wurde.

Im folgenden Programm sind Kommentare eingefuegt, durch die die wichtigsten Punkte des Programmablaufs und der Programmstruktur verdeutlicht werden.

```
AAAA                    SEGMENT PARA PUBLIC 'CODE'
              DB        0FDH              ;BSAVE id
              DW        0,0
              DW        TRAILER-HEADER ;Modulgroesse
AAAA          ENDS
              PAGE
              BASDATSEG SEGMENT BYTE PUBLIC 'CODE'
              ASSUME CS:BASDATSEG,DS:BASDATSEG
HEADER EQU    $     ;Start des Speicherabbildes
SUBRT  PROC   FAR
       NOP            ;Kann ersetzt werden durch 0CCH
;                      (INT 3) als Pruefpunkt fuer
;                      DEBUG.
       CALL   SKIP ;Legt den Zeiger fuer 'BASE' im Stack
;                     ab
BASE:                 ;Arbeitsbereich ermitteln
WORKA  DW 0 ;Erster Eingabeparameter,
;            BASIC's ARRAY(3)
WORKB  DW 0 ;Zweiter Eingabeparameter,
;            BASIC's ARRAY(4)
WORKC  DW 0 ;Ergebnis, BASIC's ARRAY(5)
SKIP:  POP    BX      ;In BX wird der Zeiger abgelegt, wo
;                      sich 'BASE' im BASIC-Datensegment
;                      befindet
       MOV    AX,WORKA - BASE[BX] ;1. Parameter lesen
       ADD    AX,WORKB - BASE[BX] ;2. Parameter addieren
;                                    zu ACC
       MOV    WORKC-BASE[BX],AX ;Im 3. Parameter ablegen
       RET
SUBRT      ENDP
TRAILER    EQU      $        ;Ende des Speicherabbildes
BASDATSEG ENDS
           END
```

Erlaeuterung zum Beispielprogramm

Mit diesem Beispiel wird ein Leitsatz erstellt, der aussieht, als ob er mit **BSAVE** erstellt worden waere. Er muss an den Beginn des Lademoduls angekettet werden, indem ein Segmentname benutzt wird, der im Alphabet am Anfang steht.

```
AAAA            SEGMENT PARA PUBLIC 'CODE'
                DB      0FDH            ;BSAVE id
                DW      0,0
                DW      TRAILER-HEADER ;Modulgroesse
AAAA            ENDS
                PAGE
```

Der Wert fuer die Modulgroesse, dividiert durch 2, ist der Wert, der fuer **N** in obigem Beispiel eingesetzt wird.

Die Byte-Ausrichtung der folgenden **SEGMENT**-Anweisung wird benoetigt, damit das Speicherabbild als naechstes Byte nach dem Schein-Leitsatz fuer **BSAVE** ohne Fuellbyte geladen wird.

```
BASDATSEG SEGMENT BYTE PUBLIC 'CODE'
```

Dieser Code soll in ein Integer-BASIC-Feld im BASIC-Datensegment geladen werden. Die Zeiger in der Assemblerliste sind relativ zum Beginn des Unterprogramms und nicht zum Beginn des BASIC-Datensegments, auf das **CS** und **DS** zeigen. Das Unterprogramm ermittelt seine eigenen Hauptspeicheradressen, indem es prueft, wo es im BASIC-Datensegment geladen wurde und anschliessend mit diesem Wert die Adressen entsprechend aendert.

```
                ASSUME CS:BASDATSEG,DS:BASDATSEG
HEADER EQU      $       ;Start des Speicherabbildes
SUBRT  PROC     FAR
       NOP              ;Kann auch ersetzt werden durch 0CCH
;                        (INT 3) als Pruefpunkt fuer DEBUG.
       CALL     SKIP    ;Legt den Zeiger fuer 'BASE' im
;                        Stack ab.
BASE:                   ;Arbeitsbereich ermitteln
WORKA  DW 0 ;1. Eingabeparameter, BASIC's ARRAY(3)
WORKB  DW 0 ;2. Eingabeparameter, BASIC's ARRAY(4)
WORKC  DW 0 ;Ergebnis, BASIC's ARRAY(5)
SKIP:  POP      BX      ;In BX wird der Zeiger abgelegt, wo
;                        sich 'BASE' im BASIC-Datensegment
;                        befindet.
       MOV      AX,WORKA - BASE[BX] ;1. Parameter lesen
       ADD      AX,WORKB - BASE[BX] ;2. Parameter addieren
;                                     zu ACC
```

Hierbei ist zu beachten, wie der lokale Speicher adressiert werden muss. Durch Einfuegen von **BX** wird die benoetigte Verschiebung durchgefuehrt, lokale Zeiger werden in **DS**-Zeiger (BASIC-Datensegment) geaendert.

```
        MOV     WORKC-BASE[BX],AX ;Ablegen im 3. Parameter
        RET
SUBRT   ENDP
TRAILER EQU     $         ;Ende Speicherabbild
BASDATSEG ENDS
        END
```

Die aufrufende BASIC-Routine wuerde wie folgt aussehen:

```
 10 OPTION BASE 1
 20 DEFINT A-Z
 40 SUBRT = 0
 50 DIM ARRAY(10)
 60 'Subroutine in das Feld laden
 70 BLOAD "SUB2.BLO",SUBRT
 80 'Parameter in das Feld eintragen
 90 ARRAY(3)=2: ARRAY(4)=3
100 'Beginn des Feldes finden
110 SUBRT=VARPTR(ARRAY(1))
120 'Parameter sind im Feld abgelegt
130 CALL SUBRT
140 'Ergebnis anzeigen
150 PRINT ARRAY(5)
```

Mit dem folgenden Beispiel wird die gleiche Funktion durchgefuehrt wie mit dem vorhergehenden. Hier werden Parameter uebergeben, indem Variablennamen in die in Klammern stehende Liste in der Anweisung **CALL** eingetragen werden.

```
 10 OPTION BASE 1
 30 SUBRT%=0: PARMC%=0
 40 DIM ARRAY%(12)
 50 'Beginn des Feldes finden
 60 SUBRT%=VARPTR(ARRAY%(1))
 70 'Routine in das Feld laden
 80 BLOAD "SUB1.BLO", SUBRT%
 90 'Parameter an Subroutine uebergeben
100 PARMA% = 2: PARMB% = 3
110 'Lokalisieren Subroutine
120 SUBRT% = VARPTR(ARRAY%(1))
130 'Subroutine rufen
140 CALL SUBRT%(PARMA%,PARMB%,PARMC%)
150 'Ergebnis anzeigen
160 PRINT PARMC%
```

Das Unterprogramm **SUB1.BLO** sieht wie folgt aus:

```
PARM    STRUC       ;Beschreibung der Parameterliste
SAVEBP  DW      0   ;Sichern des BP-Registers
RETOFF  DW      0   ;Zeiger, von wo RET erfolgt
RETSEG  DW      0   ;Retten Segment
PARMC   DW      0   ;Zeiger zum 3. Parameter
PARMB   DW      0   ;Zeiger zum 2. Parameter
PARMA   DW      0   ;Zeiger zum 1. Parameter
PARM    ENDS

AAA     SEGMENT PARA
        DB      0FDH            ;BSAVE ID
        DW      0,0
        DW      TRAILER-HEADER  ;Modulgroesse
AAA     ENDS
BASDATSEG SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:BASDATSEG,DS:BASDATSEG

HEADER  EQU     $   ;Start des Speicherabbildes
SUBRT   PROC    FAR
        NOP         ;Zum Testen kann dies durch 0CCH
;                    (INT 3) als Pruefpunkt fuer DEBUG
;                    ersetzt werden.
        PUSH    BP  ;Sichern des BP-Registers von BASIC
        MOV     BP,SP ;Setzen Adressmoeglichkeit auf Para-
;                      meterbereich im Stack
        MOV     SI,[BP].PARMA  ;Lesen Zeiger zum Para-
;                              meter 1
        MOV     AX,[SI]  ;Lesen Wert von Parameter 1
        MOV     SI,[BP].PARMB  ;Lesen Zeiger zum Para-
;                              meter 2
        MOV     AX,[SI]  ;Addieren Wert von Parameter 2
;                        zu ACC
        MOV     DI,[BP].PARMC  ;Lesen Zeiger zum Para-
;                              meter 3
        MOV     [DI],AX  ;Uebergabe Ergebnis zum Para-
;                         meter 3
        POP     BP     ;Ruecksetzen BASIC's BP
        RET     6      ;Zurueck zu BASIC, verwerfen der
;                       3 Parameter
SUBRT   ENDP
TRAILER EQU     $      ;Ende Speicherabbild
BASDATSEG ENDS
        END
```

Das Unterprogramm in Maschinensprache kann mit dem BASIC-Befehl **BLOAD** in diesen externen Datenbereich bzw. mit **POKE** in den Hauptspeicher geladen werden.

Das Unterprogramm kann ab Offset 0 des ersten Abschnitts nach dem BASIC-Datensegment geladen werden. Die Segment-ID kann berechnet werden, indem zum Wert der BASIC-Segment-ID die hoechstmoegliche Anzahl Abschnitte, an die BASIC adressieren kann, addiert wird.

Mit folgender Methode wird berechnet, wohin das Unterprogramm mit **BLOAD** geladen werden soll:

- Das aktuelle Segment mit der Anweisung **DEF SEG** auf Segment 0 setzen. Die Segment-ID des BASIC-Datensegments befindet sich nun im niedrigen Speicherbereich, Segment 0, Offsets &H510 und &H511.

- Diese beiden Werte mit **PEEK** ermitteln und das Ergebnis einer Variablen zuordnen.

- Zu diesem Wert muss die hoechstmoegliche Anzahl Abschnitte, die BASIC adressieren kann, addiert werden (Standard: 4096 oder &H1000). Dieser Wert kann durch die Anweisung **CLEAR** oder den Schalter /M: geaendert werden.

- Mit der Anweisung **DEF SEG** diesen Wert als neue Segment-ID deklarieren.

- Das Unterprogramm bei Offset 0 mit **BLOAD** in dieses Segment laden.

Das BASIC-Programm sieht wie folgt aus:

```
10 DEF SEG = 0  'Niedriger Speicherbereich
20 'Lesen Segment-ID
30 V = PEEK(&H510) + (256*PEEK(&H511))
40 SEGID = V + &H1000  '4096 addieren
50 'Segment definieren zum naechsten Segment nach BASIC
60 DEF SEG = SEGID
70 'In dieses Segment ab Zeiger 0 laden
80 BLOAD "SUBROUT.COM" , 0
```

11.5.3. Laden eines Unterprogramms als residente DCP-Erweiterung

Das Unterprogramm in Maschinensprache kann unter DCP wie ein Befehl geladen werden. Anschliessend kann BASIC das Unterprogramm aufrufen.

Vorteile:

- Der Speicherbereich, der das Unterprogramm enthaelt, ist durch DCP geschuetzt. Es muss keinerlei Speicherverwaltung durchgefuehrt werden, um das Programm vor Loeschen zu schuetzen.

- Der Maschinencode muss nicht verschiebbar sein.

- Mit dieser Methode koennen sehr grosse Module geladen werden. DCP teilt den benoetigten Speicherbereich ein, um sich selbst und seine Erweiterungen zu laden, bevor Platz fuer BASIC zugeordnet wird.

- Da das Unterprogramm resident bleibt, wenn es von DCP geladen wurde, kann ein Unterprogramm geschrieben werden, das von mehreren Programmen benutzt wird. Das Unterprogramm muss nur einmal geladen werden.

Ueberlegungen:

- Fuer diese Methode wird ein Assembler benoetigt.

- Unter DCP geladene Unterprogramme bleiben resident, bis ein neues Programm geladen wird.

Der unter DCP geladene Modul (das Unterprogramm in Maschinensprache) besteht aus zwei Teilen: dem BASIC-Unterprogramm und dem Teil, der fuer den Programmlader benoetigt wird. Der Modul kann als externer DCP-Befehl aufgerufen werden (Dateiname ohne Dateityp eingeben).

Folgende Schritte muessen im Programmladeteil des Unterprogramms in Maschinensprache ablaufen:

- Der Doppelwortvektor, der auf das Unterprogramm zeigt, muss im niedrigen Speicherbereich stehen.

- Es muss DCP mitgeteilt werden, wieviel des Moduls resident bleiben soll (nur das BASIC-Unterprogramm, nicht der Programmlader).

- Mit **INT 27H** ist zu DCP zurueckzukehren, wodurch das Unterprogramm resident bleibt.

Ein Problem bei von DCP geladenen Modulen besteht darin, dass BASIC mitgeteilt werden muss, wo sich das Unterprogramm nach dem Laden befindet. Ein fuer Benutzer reservierter Vektor im ersten K des Hauptspeichers kann ausgewaehlt werden. Der Uebertragungsbereich fuer interne Anwendungen (16 Byte bei &H4F0 - 4FF im Segment 0) kann auch benutzt werden, um den 4-Byte-Vektor zu speichern, der auf das Unterprogramm zeigt.

Der Programmlader sollte den Zeiger und die Segment-ID des Unterprogramms an einer bestimmten Stelle in Segment 0 speichern. BASIC fuehrt anschliessend **DEF SEG = 0** und **PEEK** bei den auf Offset &H4F0 folgenden 4 Byte aus, um das Unterprogramm zu suchen. Dies ist ein aehnlicher Vorgang wie das Finden des BASIC-Standardwertes fuer **DEF SEG**.

Um zu verhindern, dass mehrere Kopien des Unterprogramms resident werden, kann eine **.BAT**-Datei definiert werden, die das Unterprogramm laedt und BASIC und das BASIC-Programm aufruft. Anschliessend sollte das Programm den **BIOS**-Code fuer das einleitende Programmladen aufrufen und ausfuehren, um zu verhindern, dass das Unterprogramm im Hauptspeicher bleibt.

Die .BAT-Datei sollte wie folgt aussehen:

```
Unterprogramm - Name

BASIC-Anwenderprogramm - Name
```

Der Programmladerteil des Unterprogramms setzt den Zeiger fuer den Uebertragungsbereich der internen Anwendung und laesst die Routine resident:

```
SEGZERO SEGMENT AT 0
        ORG 4F0H    ;Uebertragungsbereich fuer
;                    interne Anwendungen
COMAREA_OFF  DW  ?      ;Vektor, der auf das
COMAREA_SEG  DW  ?      ;Unterprogramm zeigt.
SEGZERO ENDS

GRP     GROUP  AAAA,ZLOADER

AAAA    SEGMENT 'CODE' ;Der Start des Lademoduls wird
;                       definiert und dieser vor "DATA"
AAAA    ENDS            aktiviert.

SUBDAT  SEGMENT PARA PUBLIC 'DATA'
SUBDAT  ENDS

SUBSEG  SEGMENT PARA PUBLIC 'CODE'
        ASSUME  CS:SUBSEG
SUBRT   PROC    FAR     ;BASIC Eintrittspunkt
        PUSH    BP      ;BASIC-Register sichern
        MOV     BP,SP   ;Parameter in Stack laden
        PUSH    ES      ;BASIC-Register sichern
        MOV     AX,SUBDAT
        MOV     ES,AX
        ASSUME  ES:SUBDAT
;Einfuegen der BASIC-Subroutine
        POP     ES      ;Ruecksetzen BASIC-Register
        POP     BP
        RET     n       ;Zurueck zu BASIC,
;                        Parameter von Stack retten
SUBRT   ENDP
SUBSEG  ENDS

ZLOADER SEGMENT BYTE 'ZLOAD'
```

```
        ASSUME  CS:ZLOADER,SS:ZSTACK
;                       Stack nicht verfuegbar
;                       fuer die Subroutine
ENDRES  EQU     $           ;Ende der BASIC-Subroutine.
;                            Der Rest der Subroutine
;                            bleibt nicht resident.
LOADER  PROC    FAR         ;Eintritt von DCP
        PUSH    ES
        XOR     AX,AX       ;'RET' zurueck zu INT 27H,
;                            Zeiger 0 in PSP.
        PUSH    AX
        MOV     BYTE PTR ES:1,27H
;                      Aendern INT 20H auf
;                      INT 27H auf Beginn PSP
        XOR     AX,AX ;Loeschen Register
        MOV     ES,AX ;Setzen Extra-Segment auf 0
        ASSUME  ES:SEGZERO
;                       ;Segment-Register zeigt auf
;                        niedrigen Speicherbereich
        MOV     COMAREA_OFF, OFFSET SUBRT
;                      Einstellen Zeiger auf
;                      Eintrittspunkt Subroutine
        MOV     COMAREA-SEG,SEG SUBRT
;                      Einstellen Subroutinen-Segment
        MOV     DX,OFFSET GRP:ENDRES+100H
;                      Einstellen Zeiger auf Ende
;                      residenter Code.
;                      Bei Austritt soll CS auf den PSP
;                      zeigen, welcher die Austritts-
;                      instruktion 27H hat.
;                      Dies setzt DX auf den Zeiger
;                      von CS.
;                      Dies ist das Ende des residenten
;                      Codes.
        RET            ;Zurueck zu DCP, INT 27H in PSP
;                       laesst die Subroutine resident.
LOADER  ENDP
ZLOADER ENDS
ZSTACK  SEGMENT PARA STACK 'STACK'
        DB      16 DUP ("STACK")
                      ;Stack wird nur waehrend der
;                      Prozedur genutzt.
ZSTACK  ENDS
        END     LOADER
```

Das Unterprogramm mit dem Programmlader wird umgewandelt und verkettet. Anschliessend ist es zur Ausfuehrung als ein externer **.EXE**-Befehl von DCP bereit.

Sobald das BASIC-Anwenderprogramm geladen ist, kann das Unterprogramm wie folgt aufgerufen werden:

```
 20 DEFINT A-Z
 30 DEF SEG = 0
 40  SUBOFF = PEEK(&H4F0)+(256*PEEK(&H4F1))
 50  SUBSEG = PEEK(&H4F2)+(256*PEEK(&H4F3))
  :
  :
100 DEF SEG = SUBSEG
110 CALL SUBOFF(PARMA,PARMB,PARMC)
120 DEF SEG
```

Nach Beenden der Ausfuehrung kann ein einleitendes Programmladen durchgefuehrt und dadurch das Unterprogramm entfernt werden:

```
1000 DEF SEG = &HF000 : SUBOFF = &HFFF0 : CALL SUBOFF
```