M20

PERSONAL COMPUTER

BASIC 8000, Band I

Sprachelemente, alphabetisch

olivetti

Das Handbuch dient der Information, sein Inhalt ist ohne ausdrückliche schriftliche Vereinbarung nicht Vertragsgegenstand. Technische Änderungen behalten wir uns vor. Die angegebenen Daten sind lediglich Nominalwerte.

VORWORT

Das vorliegende Handbuch ist Band I der zweiteiligen Dokumentation über BASIC 8000 unter PCOS. Die Beschreibung basiert auf dem Release-Stand 2.Ø des Betriebssystems **PCOS**. Auch Release 1.3 ist darin abgedeckt. Band I beschreibt den Befehlsvorrat in BASIC 8000; die Sprachelemente sind rein alphabetisch geordnet.

In Band II (BASIC 8000, Anwendung der Sprachelemente) sind u.a. beschrieben: Programm- und Datenstruktur, Aufruf von BASIC 8000 aus PCOS, Arbeitsweise in der BASIC-Ebene, Einsatz der BASIC-Sprachelemente im thematischen Zusammenhang, Möglichkeiten mit peripheren Einheiten, Schnittstelle PCOS/BASIC 8000, BASIC-Fehlermeldungen.

Die folgenden zusätzlichen Dokumentationen können im Papierlager der

Deutschen Olivetti DTS GmbH
Schmickstraße 14
6000 Frankfurt/M-Osthafen

bestellt werden:

* M20, Bedienungs- und Installationshandbuch
* PCOS, Betriebssystem
* V24-Peripherie, Programmierhandbuch
* IEC-Schnittstelle, Programmierhandbuch
* ISAM, indexsequentielle Dateiverwaltung
* Befehlsliste PCOS/BASIC 8000
* Bedienungsanleitungen für die verschiedenen Drucker

Zur Anwendung von Standardprogrammen, zusätzlichen Programmiermöglichkeiten oder zum Erlernen von BASIC können - ebenfalls beim Papierlager - deutsche Dokumentation mit den folgenden Themen bestellt werden:

* Multiplan
* Oliword
* Olisort
* Oliterm
* Olicom
* Bedienungsanleitung Fakturiersystem
* Bedienungsanleitung Finanzbuchführung
* Bedienungsanleitung Lohn-/Gehaltsabrechnung
* Datenverarbeitung mit BASIC (Programmierte Unterweisung)
* M20, Software-Katalog

Die gültigen Drucknummern entnehmen Sie bitte dem aktuellen M20-Software-Katalog.

Bei der

Deutschen Olivetti DTS GmbH
Lyoner Straße 34
6000 Frankfurt/M 71 (Niederrad)

finden außerdem laufend Kurse statt.

Themenkreise:

* Programmierung
* Betriebssysteme
* Textverarbeitung
* Betriebskalkulation mit Personal-Computer
* Dialog mit Anwenderprogrammen am Personal-Computer

Die Kurse erfordern unterschiedliche Vorkenntnisse.

Von Anwendern, die sich neu in ein Programm oder System einarbeiten wollen, bis hin zu Fachleuten, die sich Spezialkenntnisse aneignen wollen ——— alle finden etwas in diesem breiten Schulungsangebot.

Bitte fordern Sie dazu - unverbindlich und kostenlos - im Ausbildungszentrum an:

* den Ausbildungsplan des Ausbildungszentrums
* das Kursangebot der Olivetti-Personal-Computer-Schule.

ABS 1
AND 3
ASC 5
ATN 7
AUTO 9
CALL 11
CDBL 15
CHAIN 17
CHR$ 21
CINT 23
CIRCLE 25
CLEAR 29
CLOSE 31
CLOSE WINDOW 33
CLS 35
COLOR 37
COLOR= 41
COMMON 45
CONT 47
COS 49
CSNG 51
CURSOR 53
CVD 59
CVI 61
CVS 63
DATA 65
DATE$ 67
DEFDBL 69
DEF FN 71
DEFINT 75
DEFSNG 77
DEFSTR 79

DELETE 81
DIM 83
DRAW 85
EDIT 89
END 97
EOF 99
EQV 101
ERASE 103
ERL 105
ERR 107
ERROR 109
EXEC 111
EXP 113
FIELD 115
FILES 119
FIX 123
FN 125
FOR 127
FRE 133
GET 135
GET% 139
GOSUB 143
GOTO 145
HEX$ 147
IF...THEN...ELSE 149
IMP 153
INKEY$ 155
INPUT 157
INPUT# 161
INPUT$ 165
INSTR 167

INT 169
KILL 171
LEFT$ 173
LEN 175
LET...= 177
LINE 179
LINE INPUT 183
LINE INPUT# 187
LIST 189
LLIST 193
LOAD 197
LOC 199
LOF 201
LOG 205
LPOS 207
LPRINT 209
LPRINT USING 213
LSET 215
MERGE 217
MID$...= 219
MID$ 221
MOD 223
MKD$ 225
MKI$ 227
MKS$ 229
NAME 231
NEW 233
NEXT 235
NOT 237
NULL 239
OCT$ 241
ON ERROR GOTO 243

ON...GOSUB 245
ON...GOTO 247
OPEN 249
OPTION BASE 255
OR 257
PAINT 259
POINT 261
POS 263
PRESET 265
PRINT 267
PRINT# 271
PRINT USING 275
PRINT# USING 277
PSET 279
PUT 281
PUT% 285
RANDOMIZE 289
READ 291
REM 293
RENUM 295
RESTORE 299
RESUME 301
RETURN 303
RIGHT$ 305
RND 307
RSET 309
RUN 311
SAVE 315
SCALE 319
SCALEX 323
SCALEY 325

SGN 327
SIN 329
SPACE$ 331
SPC 333
SQR 335
STOP 337
STR$ 339
STRING$ 341
SWAP 343
SYSTEM 345
TAB 347
TAN 349
TIME$ 351
TROFF 353
TRON 355
USING 357
VAL 369
VARPTR 371
WEND 373
WHILE 375
WIDTH 377
WINDOW 379
WINDOW% 391
WRITE 393
WRITE# 395
XOR 399
? 401
' 403

FUNKTION: **ABS**

ZWECK: liefert den Absolutwert (Betrag) eines numerischen Werts

FORMAT: **ABS**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet und in seinen positiven Wert verwandelt.

BEMERKUNGEN:
- Das Ergebnis der Funktion ist von der Genauigkeit des Arguments.

BEISPIELE:

```
10  PRINT ABS(12.5);ABS(-12.5)
RUN
 12.5  12.5
```

VERWEISE: Kapitel 2.4 und 4.13

OPERATOR: **AND**

FUNKTION: Verknüpfung zweier logischer Ausdrücke mit Hilfe der UND-Verknüpfung bzw. Bildung des logischen Produkts zweier Integerwerte (Bit-für-Bit-Verknüpfung mit **AND**)

FORMAT: { log.Ausdruck / Integerwert } **AND** { log.Ausdruck / Integerwert }

WIRKUNG: Das Ergebnis der **AND**-Verknüpfung zweier Bedingungen ist -1 (wahr), wenn beide Bedingungen erfüllt sind; es ist Ø (falsch) in allen anderen Fällen.
Bei der **AND**-Verknüpfung zweier Integerwerte wird die **AND**-Operation Bit für Bit durchgeführt und das Ergebnis berechnet (vgl. Kapitel 2.6.4).

BEISPIELE:

```
10 A%=5:B%=3:C!=2.47
15 C%=A% AND B%  ' C% ist jetzt 1
20 IF A%(B% AND C!)3 THEN GOSUB 1000 ELSE GOSUB 2000
```

VERWEISE: Kapitel 2.6.4
Operatoren: **NOT, OR, XOR, EQV, IMP**

FUNKTION: ASC

ZWECK: liefert den ISO-Code eines Zeichens

FORMAT: **ASC**(Stringausdruck)

WIRKUNG: Der Stringausdruck wird errechnet und das erste Zeichen bestimmt. Von diesem wird der ISO-Code ermittelt.

BEMERKUNGEN:
- Ist das Ergebnis von 'Stringausdruck' der Leerstring, wird der Fehler "Illegal function call" gemeldet.
- **ASC** ist die Umkehrung der Funktion **CHR$**.

BEISPIELE:

```
10  PRINT ASC("abc")
RUN
 97
```

VERWEISE: Kapitel 4.3.4.3 und 4.3.17
Funktion: **CHR$**

FUNKTION: **ATN**

ZWECK: liefert von einem numerischen Wert seinen Arcustangens

FORMAT: **ATN**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet, als Tangens interpretiert und der daraus resultierende Winkel im Bogenmaß ermittelt.

BEMERKUNGEN:
- Das Ergebnis ist im Bogenmaß ausgedrückt. Es liegt zwischen -pi/2 und +pi/2. Umrechnungen müßten ggf. in Unterprogrammen oder mit einer vom Anwender definierten Funktion (s. **DEF FN**) erfolgen.
- Die Funktion ist die Umkehrung der Funktion **TAN**.

BEISPIELE:

```
10 PI!=3.14159
20  PRINT ATN(1.6)*180/PI!;"Altgrad"
RUN
 64.4711 Altgrad
```

VERWEISE: Kapitel 2.4 und 4.13
Funktion: **TAN**

BEFEHL: **AUTO**
(automatic line numbering)

FUNKTION: Automatische Erzeugung von Zeilennummern für einzugebende Anweisungen eines Programms

FORMAT: **AUTO** [{Zeilennummer}] [,] [Schrittweite]

Zeilennummer: positive ganze Zahl zwischen 1 und 65529, welche die Nummer der nächsten Anweisung angibt

Schrittweite: positive ganze Zahl, welche die jeweilige Erhöhung zur nächsten Zeilennummer bestimmt

WIRKUNG: Nach jedem Drücken der Eingabeabschlußtaste wird eine Zeilennummer generiert. 'Zeilennummer' bestimmt den Anfangswert, den die erste einzugebende Anweisung erhält. 'Schrittweite' gibt an, um wieviel die laufende Zeilennummer zur nächsten Anweisung erhöht werden soll. Als Default-Wert wird für beide Parameter der Wert 10 angenommen.

BEMERKUNGEN:
- Fehlt einer der Parameter 'Zeilennummer' oder 'Schrittweite', wird für diesen Parameter der Standardwert 10 angenommen.
- Mit **AUTO.** beginnt die Generierung der Zeilennummern ab der letzten definierten Zeilennummer.

- **AUTO.,** erzeugt Zeilennummern ab der zuletzt definierten Zeilennumer mit der vorher gewählten Schrittweite (eventuell Standardwert 10).

- Wird bei der automatischen Zeilennumerierung eine Zeilennummer erzeugt, unter der bereits eine Programmzeile im Arbeitsspeicher ist, erscheint neben der Zeilennummer ein *. Durch Drücken der Eingabeabschlußtaste unmittelbar nach * bleibt diese Programmzeile im Arbeitsspeicher erhalten. Jegliche Eingabe ändert die im Arbeitsspeicher vorhandene Programmzeile.

- C beendet die automatische Zeilennumerierung. Die Zeilennummer, bei der C gedrückt wurde, wird nicht gespeichert. Das System befindet sich anschließend im Direkt-Mode.

- Wird bei der automatischen Zeilennumerierung der höchste Wert (65528) erreicht, wird die Zeilennumerierung beendet; das System befindet sich danach im Direkt-Mode.

BEISPIELE:

```
AUTO 2500,2
AUTO .,1
```

VERWEISE: Kapitel 3.3

ANWEISUNG: CALL

FUNKTION: Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine

FORMAT: **CALL** { Stringausdruck | Name d. Routine (Liste d. Parameter) }

Stringausdruck: enthält Namen der Routine und Parameter in der Form, wie sie in der PCOS-Ebene einzugeben sind.

Name d. Routine: Stringausdruck, der den Namen der abzuarbeitenden Routine bestimmt.

Liste d. Parameter: Folge von Integer- und/oder Stringvariablen oder -konstanten, die durch Komma getrennt sind.

WIRKUNG: Das System prüft, ob die aufgerufene Assembler-Routine bereits im Arbeitsspeicher vorhanden ist. Falls nicht, wird diese Routine auf der Diskette gesucht und in den Arbeitsspeicher geladen. Anschließend wird diese Routine sofort abgearbeitet. Der momentane Zustand des Arbeitsspeichers bleibt erhalten.

BEMERKUNGEN:
- **CALL** erledigt die gleichen Aufgaben wie **EXEC**, ist aber leistungsfähiger.

- Nach Ausführung der Assembler-Routine wird das BASIC-Programm mit der der **CALL**-Anweisung folgenden Anweisung fortgesetzt.

- Werden an die Assembler-Routine numerische Variablen übergeben, müssen diese Variablen als Integervariablen deklariert werden.

- Sollen Parameter in Form von Variablen übergeben und bestimmte Parameter im aktuellen Zustand gelassen werden, muß das Format mit 'Stringausdruck' verwendet werden. Die Parameter sind über Stringverarbeitung in einem Stringausdruck auszudrücken. Für die nicht zu ändernden Parameter muß nur der Platzhalter , (Komma) gesetzt werden (vgl. PCOS-Handbuch).

- Variablen, deren Werte von der Assembler-Routine an das BASIC-Programm übergeben werden, müssen vor dem Variablennamen das §-Zeichen stehen haben.

- Werden Werte von PCOS an Variablen übergeben, müssen die betreffenden Variablen vor dem ersten Aufruf einen Wert erhalten haben (z.B. durch Wertanweisungen wie **T%**=Ø oder **A$=SPACE$(**255) im Vereinbarungsteil). Solche Stringvariablen müssen vor jedem **CALL** eine ausreichende Länge haben, um beim **CALL** die übergebenen Werte aufnehmen zu können.

- Die Parameter in 'Liste der Parameter' müssen in der richtigen Reihenfolge und formatgerecht (nur Integer oder String) sein.

- Zur Beschleunigung des Aufrufs von Assembler-Routinen kann vorher der PCOS-Befehl **pl** verwendet werden. Dann bleiben die Routinen ständig im Arbeitsspeicher erhalten.

- Die am meisten verwendeten PCOS-Routinen innerhalb von BASIC-Programmen sind:

  lt zur Abfrage, welche Eingabeabschlußtaste betätigt wurde
  pk Anwender-definierte Tastenbelegung
  la Ausgabe von Text in der graphischen Arbeitsweise
  sp Hardcopy (Text und Graphik)
  pl Laden einer Assembler-Routine in den Arbeitsspeicher als resident gehaltene Routine
  ci Kommunikation mit dem RS-232-(V24)-Driver
  ls Ausgabe des Textinhaltes eines Windows auf dem Drucker (ohne Graphik)
  bv Abfragen Diskettenname, freie Sektoren oder Inhaltsverzeichnis
  ck Ändern Feststellfunktionen, Tastenbelegung, Tastenfunktionen
  pu Entladen Assembler-Routine
  ss Ändern Parameter für PCOS
  sf Ändern Druckerparameter

- Wechsel des Ein/Ausgabe-Mediums (**+/- s/d...**) darf nur mit **EXEC** aufgerufen werden.

BEISPIELE:

```
10 CALL "pl %n la":CALL "ss %n,,,,,0"
15 CLEAR:SCALE 0,511,0,255
20 CALL "la 'OLIVETTI',0,180,4,0"
30 CALL "la"("TEXT",300,10,8,1)
35 V%=2
40 T$="la '"+DATE$+"',"+MID$(STR$(SCALEX(310)),2)+","+
   MID$(STR$(SCALEY(180)),2)+","+MID$(STR$(V%),2)+",0"
50 CALL T$
60 GOTO 60
```

```
10 CLEAR:OHO%=-1
20 LINE INPUT "Ihre Eingabe";E$
150 CALL "lt"(@OHO%)  'Variable von PCOS nach BASIC
160 ON OHO%+1 GOSUB 1001,2001,3001 'je nach Abschlußtaste
```

VERWEISE: Kapitel: 4.3.15
Anweisung: **EXEC**
Funktion: **VARPTR**

FUNKTION: **CDBL**
(convert double precision)

ZWECK: Konvertiert einen numerischen Wert in doppelte Genauigkeit

FORMAT: **CDBL**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet und gemäß Rundungsregeln in doppelte Genauigkeit umgewandelt.

BEMERKUNGEN: ACHTUNG:

Auch bei Anwendung der Funktion **CDBL** kann bei der Umwandlung eines Wertes in einfacher Genauigkeit Signifikanzverlust eintreten; d.h. das Ergebnis der Funktion ist nur annähernd gleich dem Anfangswert (vgl. Kapitel 2.5.1.4, Punkt 5. sowie 2.7.1)! Diese Problematik kann mit einer Anweisung der Art D# =**VAL**(**STR$**(G!)) umgangen werden.

BEISPIELE:

```
10 DEF FNSUM#(S1#,S2#)=S1#+S2#
20 A!=2.45:B!=2.15
30 PRINT FNSUM#(CDBL(A!),CDBL(B!))
```

VERWEISE: Kapitel 2.4, 2.5.1.5 und 4.13

ANWEISUNG: **CHAIN**

FUNKTION: Aufruf eines Programms, eventuell mit Übergabe von Variablen vom gegenwärtigen Programm an das aufgerufene sowie gegebenenfalls Laden von Overlays und/oder Löschen von Programmzeilen

FORMAT: **CHAIN** [**MERGE**] Filename [, [Startzeile] [,**ALL**]] [,**DELETE** {Zeilennr. 1 - Zeilennr. 2 / - Zeilennr. 2}]

Filename : Stringausdruck, dessen Ergebnis den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

Startzeile : numerischer Ausdruck, dessen Ergebnis eine Zeilennummer definiert

Zeilennr. 1,
Zeilennr. 2 : Konstante, die eine Zeilennummer angibt.

WIRKUNG: Die Verarbeitung des laufenden Programms wird beendet. Das Programm mit dem Namen 'Filename' wird von Diskette geladen und die Ausführung ab der ersten Zeile bzw. der 'Startzeile' begonnen. Das zugeladene Programm muß im ASCII-Format gespeichert sein (**SAVE...,A**).

Parameter **ALL**

Ist der Parameter **ALL** gesetzt, werden alle Variablen des Ausgangs-Programms an das Folgeprogramm übergeben, wobei der Wert der Variablen erhalten bleibt. Sollen nur bestimmte Variablen übergeben werden, siehe unter Anweisung **COMMON.**

Parameter **MERGE**

Wird der Parameter **MERGE** gesetzt, wird das unter 'Filename' angegebene Programm als Overlay in das gegenwärtige Programm zugeladen. Danach wird das Gesamtprogramm gestartet. Das durch 'Filename' bestimmte Programm muß in Form eines ASCII-Files auf Diskette vorliegen (Z.B. **SAVE "1:ABC",A)**.

Parameter **DELETE**

Der Parameter **DELETE** wirkt sich nur aus, wenn auch der Parameter **MERGE** gesetzt wurde. Es werden die unter **DELETE** angegebenen Zeilennummern des gegenwärtigen Programms gelöscht, bevor **MERGE** (Laden des Overlay-Programms) ausgeführt wird.

BEMERKUNGEN:

- Die Zeilennummern 'Zeilennr.1' und 'Zeilennr.2' werden durch den Befehl **RENUM** unnumeriert, während 'Startzeile' nicht (!) unnumeriert wird.
- Durch **CHAIN** werden Daten-Files nicht geschlossen.
- Der aktuelle Wert von **OPTION BASE** bleibt bei **CHAIN** erhalten.
- Wird der Parameter **MERGE** benutzt, werden die Daten-Files geschlossen, alle Variablen gelöscht und die Routinen **FOR...NEXT, WHILE...WEND** und **GOSUB...RETURN** abgeschlossen.
- Disketten- und File-Password sind anzugeben, falls vorhanden. Die Angabe des Disketten-Passwords kann entfallen, wenn seit seiner letzten Angabe kein Diskettenwechsel und kein Neuladen des Betriebssystems erfolgt ist.

- Der Prozeß des Zuladens bei **CHAIN MERGE** kann mehrere Minuten dauern. Es empfiehlt sich, zur Beschleunigung das Overlay ans jeweilige Ende des Arbeitsspeicherinhalts zuzuladen.

BEISPIELE:

```
10 CLS:LINE INPUT "Ihre Programmwahl (1-4)";W$
20 IF ASC(W$)<49 OR ASC(W$)>52 THEN 10
30 ON ASC(W$) GOTO 40,50,60
35 CHAIN "1:PROG4.prg"
40 CHAIN "1.PROG1.prg",100
50 CHAIN "1:PROG2.prg",,ALL
60 CHAIN MERGE "1:SEGMENT3.asc",100,ALL,DELETE 100-999
100 REM weiter mit PROG1.prg bzw. eingebundenem SEGMENT3.asc
999 END
```

VERWEISE: Kapitel: 4.3.13
Befehl: **SAVE**
Anweisung: **COMMON**

FUNKTION: **CHR$**

ZWECK: liefert dasjenige Zeichen, das durch einen ISO-Code bestimmt ist

FORMAT: **CHR$(**num.Ausdruck**)**

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit kaufmännisch gerundet. Es wird dasjenige Zeichen ermittelt, das dem ISO-Code des Zeichens entspricht.

BEMERKUNGEN:
- Das Ergebnis von 'num.Ausdr.' muß zwischen Ø und 255 liegen, sonst wird der Fehler "Illegal function call" gemeldet.
- **CHR$** ist die Umkehrung der Funktion **ASC.**
- Zahlreiche nicht druckbare Zeichen mit den ISO-Codes Ø bis 31 dienen als Steuerzeichen am Bildschirm oder Drucker.
- Häufig werden nicht druckbare Zeichen als Trennzeichen zwischen Daten eingesetzt (vgl.z.B. **PRINT#, INPUT#**).

BEISPIELE:

```
10  PRINT CHR$(66)
RUN
B
```

VERWEISE: Kapitel 4.3.4.3 und 4.3.17
Funktion: **ASC**

FUNKTION: **CINT**
(convert integer)

ZWECK: Konvertiert einen numerischen Wert in einen ganzzahligen (Integer)-Wert

FORMAT: **CINT**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet und gemäß den Rundungsregeln in einen Integerwert verwandelt.

BEMERKUNGEN:
- Ist das Argument nicht im Intervall zwischen -32768 und 32767, wird "Overflow" gegeben.

BEISPIELE: **CINT**(3.7) liefert 4 und **CINT**(-3.7) liefert -4.
CINT(3.2) liefert 3 und **CINT**(-3.2) liefert -3.

```
10 DEFDBL X
20 X=32.5746584#:LPRINT X;CINT(X)
RUN
 32.5746584  33
```

VERWEISE: Kapitel 2.4, 2.5.1.5 und 4.13
Funktionen: **FIX, INT**

ANWEISUNG: CIRCLE

FUNKTION: Zeichnet einen Kreis oder eine Ellipse

FORMAT: **CIRCLE** [% Window-Nr.,](X,Y), Radius[, [Farbindex] [,Achsenverhältnis] [,Operand]]

Window-Nr.: numerischer Ausdruck (Wert: 1 bis 16)

X,Y: numerische Ausdrücke, die den Mittelpunkt bestimmen

Radius: numerischer Ausdruck, der den Radius des Kreises, bezogen auf die gültige Skalierung der X-Achse, bestimmt.

Farbindex: numerischer Ausdruck, der die Farbe des Kreises bestimmt.
(Beim Vier-Farbschirm: bezogen auf die aktuelle globale **COLOR**=-Anweisung; Wert: Ø-3; sonst identisch mit 'Farbcode')

Achsenverhältnis: numerischer Ausdruck, der das Verhältnis der gültigen Radien von X- zu Y-Richtung bestimmt.

Operand: Operand kann sein: **AND, OR, XOR, NOT, PSET** oder **PRESET**

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, wird der Kreis in diesem Window gezeichnet, andernfalls im aktiven Window. Das Koordinatenpaar **(X,Y)** bestimmt den Mittelpunkt des Kreises in Absolut-Koordinaten bezogen auf das für das Window gültige **SCALE** für die X-Achse.

Radius

Dieser Parameter bestimmt den Radius des Kreises. Der Wert ist bezogen auf das für das Window geltende **SCALE** in horizontaler (X-)Richtung.

Farbindex

Dieser Parameter bestimmt, in welcher Farbe der Kreis gezeichnet wird. Dabei gibt der Parameter 'Farbindex' für den Vier-Farb-Schirm die Position (Ø-3) der entsprechenden Farbe in der globalen **COLOR**=-Anweisung an. Beim Schwarz/Weiß- und Acht-Farb-Schirm sind 'Farbindex' und 'Farbcode' identisch.

Achsenverhältnis

Fehlt dieser Parameter, wird ein Kreis gezeichnet. Ist 'Achsenverhältnis' angegeben, wird eine Ellipse nach folgenden Regeln gebildet:
Als Ausgangswert für die horizontale und vertikale Halbachse wird der gültige Wert von 'Radius' im jeweils gültigen Maßstab gemäß **SCALE** genommen.
Der Parameter 'Achsenverhältnis' wird für die vertikale Halbachse als Faktor für den sich aus Radius und Maßstab ergebenden Wert genommen.
Aus den sich dadurch ergebenden Werten für die Halbachse wird eine Ellipse gezeichnet.

Operand

Operand kann eines der Sprachelemente **AND, OR, XOR, NOT, PSET** oder **PRESET** sein. Es wird dann auf dem für die Darstellung maßgeblichen Bereich der Bitmap (vgl. Kapitel 4.3.18) die entsprechende logische Operation für jeden Punkt durchgeführt.

Wird **PSET** angegeben, werden alle betroffenen Punkte auf die angegebene Farbe gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Farbe des Hintergrundes gesetzt.

BEMERKUNGEN:

- Kreise und Ellipsen werden gezeichnet, indem jeweils die Punkte in der entsprechenden Farbe dargestellt werden, die den sich aus der Rechnung ergebenden Werten am nächsten liegen.
- Es wird jeweils der Teil des Kreises oder der Ellipse gezeichnet, der innerhalb des Windows liegt.
- Je kleiner der Wert für 'Achsenverhältnis' ist, desto flacher wird die Ellipse, je größer dieser Wert ist, um so höher wird die Ellipse.

BEISPIELE:

```
0 CLEAR:CALL "ss %n,,,,,0"
1 SCALE 0,511,0,255
10 CIRCLE(511/2,255/2),40
20 CIRCLE(511/2,255/2),60,,0.8
100 PRINT "weiter: beliebige Taste:":K$=INPUT$(1)
1000 CLEAR:FOR I%=1 TO 17
1010 CIRCLE(250,180-5*I%),40+5*I%
1020 CIRCLE(250,80+5*I%),40+5*I%
1030 CIRCLE(295+5*I%,130),40+5*I%
1040 CIRCLE(215-5*I%,130),40+5*I%
1050 NEXT
1060 CALL "sp"  ' nur hardcopy-fähige Drucker
```

VERWEISE: Kapitel : 4.3.14 und 4.3.18
Anweisungen: **COLOR=, SCALE**
Funktion : **WINDOW**

FUNKTION: Löschen des Datenbereichs und des gesamten Bildschirms; Schließen aller Files; setzen des gesamten Bildschirms auf Default-Hintergrundfarbe und auf das zuletzt im PCOS-Befehl **ss** gesetzte Format; Aufhebung aller Window-Vereinbarungen; ggf. Definition der logischen Größe des Arbeitsspeichers und/oder des für Stack-Speicher zu reservierenden Platzes im Arbeitsspeicher.

FORMAT: **CLEAR** [, [Arbeitsspeichergröße][, Stack-Größe]]

Arbeitsspeicher-Größe: num.Ausdruck
Stack-Größe: num.Ausdruck

WIRKUNG: Der gesamte Datenbereich im Arbeitsspeicher wird gelöscht (incl. aller Variablen, die aufgrund einer **COMMON**-Anweisung weitergereicht werden könnten). Bei späterem Aufruf (ohne vorherige Zuweisung) haben alle numerischen Variablen den Wert Ø und alle String-Variablen den Wert Leerstring ("")

Alle Vereinbarungen über den Bildschirm (Windows, Farbauswahl) werden aufgehoben (Setzen der PCOS-Default-Werte). Der Bildschirm wird komplett auf die Default-Hintergrundfarbe gelöscht.

Mit 'Arbeitsspeicher-Größe' kann definiert werden, auf wieviele Bytes der physisch dem Anwender zur Verfügung stehende Platz im Arbeitsspeicher reduziert werden soll.

Mit 'Stack-Größe' kann der Wert, der für Stack-Speicher im Arbeitsspeicher in Bytes standardmäßig oder aufgrund der letzten **CLEAR**-Anweisung reserviert wurde, verringert oder vergrößert werden.

BEMERKUNGEN:

- **CLEAR** schließt alle offenen Files.
- **CLEAR** löscht auch **COMMON**-Bereichs-Variablen.
- Die Summe aus 'Arbeitsspeicher-Größe' und 'Stack-Größe' muß kleiner sein als der mit PCOS-Befehl **sb** zur Verfügung des Anwenders freigegebene Platz im Arbeitsspeicher.
- Für einen Stack-Speicher müssen zwischen 9.7 Bytes (bei einer sehr geringen Anzahl) und 6.1 Bytes (bei einer sehr hohen Anzahl) kalkuliert werden (zur Anzahl vgl. Kapitel 4.3.5.1).
- Eine durch **CLEAR** definierte 'Arbeitsspeicher-Größe' kann nur durch Neuladen des Systems rückgängig gemacht werden.
- Eine durch **CLEAR** definierte 'Stack-Größe' kann (auch in neuen Programmen) erst durch eine neue **CLEAR**-Anweisung oder durch Neuladen des Systems rückgängig gemacht werden.

BEISPIELE:

```
10 CLEAR  'Auflösen aller Window-Vereinbarungen
          aus Vorprogramm + Löschen Schirm
```

VERWEISE: Kapitel 4.3.2 und 4.3.5.1

Befehle: **RUN, LOAD**

Anweisungen: **GOSUB,ON...GOSUB,RETURN,FOR...NEXT, WHILE...WEND,COMMON,CHAIN, CLOSE WINDOW**

Fehler-Code: 7

FUNKTION: Schließen von einem, mehreren oder allen externen Datenfiles und Schreiben des restlichen Filepuffer-Inhalts auf Diskette.

FORMAT: **CLOSE** [[#] Filenr. [,[#]Filenr] ...]

Filenr.: numerischer Ausdruck
Es ist die Nummer anzugeben, mit der das Datenfile bei **OPEN** geöffnet wurde.

WIRKUNG: Der ganzzahlige Zeil des numerischen Ausdrucks wird ermittelt.
Daten, die sich noch im File-Puffer befinden und noch nicht gespeichert sind, werden automatisch in das entsprechende File geschrieben, wenn:

- sich ein sequentielles File im Output-Mode befindet (siehe Parameter "O" und "A" bei der Anweisung **OPEN**)
- bei einem Random-File die Anweisung **PUT** ausgeführt, aber das Schreiben physisch noch nicht erfolgt ist.

Sämtliche Informationen, die sich auf das File beziehen, werden anschließend im Arbeitsspeicher gelöscht, d.h. die dem File zugeordnete Nummer (der zugeordnete File-Puffer) wird wieder frei.

BEMERKUNGEN:
- Die **END**-, die **CLEAR**- und die **CHAIN**-Anweisung mit Parameter **MERGE** sowie die Befehle **NEW** und **RUN** bewirken automatisch ein **CLOSE** für alle geöffneten Files.

- Files sollten im Programm immer durch **CLOSE**-Anweisungen geschlossen werden. Von jedem anderen Schließen ist wegen des möglichen Datenverlusts abzusehen.
- Die **STOP**-Anweisung und **CTRL C** schließen Files nicht.

BEISPIELE:

```
10 'öffnen Files S.seq, R.rnd + S1.seq
20 OPEN "I",1,"1:S.seq"
30 A$="S1.seq":O$="I"
40 OPEN O$,2,"1:"+A$
50 OPEN "R",3,"1:R.rnd",10
60 FIELD 3,10 AS B.PUFFER3$
                :
80 FOR I%=1 TO 3:GOSUB 1010:NEXT

1000 'Schließen File  I%
1010 CLOSE I%:RETURN
```

VERWEISE: Kapitel : 4.3.1Ø.2
Anweisung: **OPEN, CLEAR**
Befehle : **RUN, NEW**

ANWEISUNG: **CLOSE WINDOW**

FUNKTION: Auflösen von Windows

FORMAT: **CLOSE WINDOW** [%Window-Nr.]

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 2 und 16 enthalten muß

WIRKUNG: Wird nur **CLOSE WINDOW** angegeben, werden alle zur Zeit definierten Windows mit Ausnahme des Windows 1 aufgelöst. Es existiert also nur noch ein Window (Gesamt-Schirm), und dieses ist aktiv.

Wird die 'Window-Nr.' angegeben, wird dieses Window aufgelöst und der freiwerdende Bereich einem anderen Window zugeordnet.

Es gelten dann für den Gesamtbereich aus aufgelöstem Window und dem Window, dem dieser Bereich zugeordnet wurde, die Parameter des verbleibenden Windows.

Der Bereich des/der aufgelösten Windows wird vollständig mit der Hintergrundfarbe gefüllt (wie bei **CLS**).

BEMERKUNGEN:
- Window 1 kann nie aufgelöst werden.
- Man beachte den Unterschied zwischen dem Auflösen eines Windows mit **CLOSE WINDOW** und dem Löschen des Inhalts eines Windows mit **CLS** .

- Der durch die Auflösung des Windows frei werdende Bereich wird einem anderen Window zugeordnet. Da dieses Window sich dadurch vergrößert, ergibt sich eine Änderung des geltenden Maßstabes in diesem Window.
- **CLEAR** löst ebenfalls alle offenen Windows auf.
- Es ist empfehlenswert, vor dem Programmende alle eröffneten Windows aufzulösen (am sichersten mit **CLEAR**, falls nicht Daten übergeben werden sollen), da andernfalls nachfolgende Arbeiten durch eröffnete Windows fehlerhaft werden können.

BEISPIELE:

```
10 CLEAR:DEFINT W
20 W=WINDOW(0,0):W1=WINDOW(3,40)
30 INPUT "Sie wollen in Window Nr.";WN
40 IF WN>2 OR WN<1 THEN 30 ELSE WINDOW%WN
                ⋮
800 CLOSE WINDOW%W1
```

VERWEISE: Kapitel 4.3.18
Anweisungen: **WINDOW%, CLEAR, CLS, SCALE**
Funktion: **WINDOW**

ANWEISUNG: **CLS**
(clear screen)

FUNKTION: Löscht den Inhalt des aktiven oder des angegebenen Windows.

FORMAT: **CLS** [%Window-Nr.]

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 1 und 16 erhalten muß.

WIRKUNG: Wird nur **CLS** angegeben, wird der Inhalt des zur Zeit aktiven Windows gelöscht und das Window in der Hintergrundfarbe dargestellt. Wird zusätzlich die 'Window-Nr.' angegeben, wird der Inhalt des dadurch spezifizierten Windows gelöscht und das Window in der Hintergrundfarbe dargestellt.

BEISPIELE:

```
10 CLS:CURSOR(1,9):PRINT "Beliebige Taste drücken"
20 K$=INPUT$(1):COLOR 0,1:CLS:PRINT "schwarz auf weiß"
30 K$=INPUT$(1):COLOR 1,0:CLS:PRINT "weiß auf schwarz"
40 GOTO 20
```

VERWEISE: Kapitel 4.3.14

FUNKTION: Auswahl oder Neufestlegung von Vorder- und Hintergrundfarbe in einem Window

FORMAT: **COLOR** [%Window-Nr.,] Vordergrundfarbe [,Hintergrundfarbe]

Window-Nr.: num.Ausdruck (Wert: 1-16)

Vordergrundfarbe: num.Ausdruck (Wert beim Vier-Farb-Schirm Ø-3, beim Schwarz-Weiß-Schirm Ø-1, beim Acht-Farb-Schirm Ø-8), dessen Ergebnis die Position der gewünschten Vordergrundfarbe bestimmt ('Farbindex' aufgrund der globalen **COLOR**=-Anweisung für den Vier-Farb-Schirm; 'Farbcode' beim Schwarz-/Weiß- oder Acht-Farb-Schirm).

Hintergrundfarbe: num.Ausdruck dessen Ergebnis die Position der gewünschten Hintergrundfarbe bestimmt ('Farbindex' aufgrund der globalen **COLOR**=-Anweisung beim Vier-Farb-Schirm; 'Farbcode' beim Schwarz/Weiß- oder Acht-Farb-Schirm).

WIRKUNG: Wird der Parameter % Window-Nr. angegeben, bezieht sich die Anweisung auf das damit bestimmte Window, anderenfalls auf das aktive Window.

Mit 'Vordergrundfarbe' wird festgelegt, welche Farbe zur Vordergrundfarbe werden soll.
Mit 'Hintergrundfarbe' wird die Hintergrundfarbe ausgewählt.

BEMERKUNGEN:

- 'Vordergrundfarbe' und 'Hintergrundfarbe' sind beim Vier-Farb-Schirm 'Farbindex' (vgl. Kapitel 4.3.18.3) und geben an, der wievielte Farbcode der globalen **COLOR**=-Anweisung gewünscht wird. Dabei steht Ø für den dort zuerstgenannten, 1 für den zweitgenannten, 2 für den drittgenannten und 3 für den zuletztgenannten.
  Wurde noch keine **COLOR**=-Anweisung abgearbeitet, erfolgt die Auswahl aus den Default-Werten, bei denen die Reihenfolge der 'Farbcodes' festliegt ('Farbindex', vgl. **COLOR**=-Anweisung).
  Für den Acht-Farb-Schirm und den Schwarz/Weiß-Schirm decken sich 'Farbcode' und gewünschte 'Vordergrundfarbe' bzw. 'Hintergrundfarbe'.
- Wird 'Hintergrundfarbe' nicht angegeben, bleibt die bisherige Hintergrundfarbe erhalten. Unterscheidet sich dabei die Vordergrundfarbe nicht mehr von der aktuellen Hintergrundfarbe, wird die **COLOR**-Anweisung jedoch nicht beachtet.
- Die **COLOR**-Anweisung wirkt sich nicht sofort auf das gesamte Window aus, sondern erst bei den einzelnen Anweisungen, die den Inhalt des Windows ändern, (z.B. **PRINT**, **CLS** oder graphische Anweisungen). Es wird immer nur der aktuell ausgegebene Text angepaßt.

BEISPIELE:

```
10 CLEAR:COLOR 1,0
20 CALL "la 'COLOR 1,0',20,20,5,0"
30 COLOR 0,1
40 CALL "la 'COLOR 0,1',20,150,5,0"
50 CURSOR(5,10):PRINT "PRINT-Ergebnis"
60 COLOR 1,0     :END
```

VERWEISE: Kapitel 4.3.18

Anweisungen: **COLOR=, WINDOW%, CLS**

ANWEISUNG: COLOR=

(globale **COLOR**-Anweisung)

FUNKTION: Auswahl von vier aus acht Farben für das Arbeiten mit dem Vier-Farb-Schirm

FORMAT: **COLOR** = Farbcode 1, Farbcode 2, Farbcode 3, Farbcode 4

Farbcode: numerischer Ausdruck, dessen Ergebnis gerundet wird und eine Zahl zwischen Ø und 7 ergeben muß

WIRKUNG: Die Anweisung wirkt sich nur auf den Vier-Farb-Schirm aus. Beim Bildschirm schwarz/weiß oder dem Acht-Farb-Schirm hat sie keine Bedeutung. Es werden durch Angabe der Farbcodes aus der nachstehenden Tabelle 4 Farben ausgewählt, die dann gleichzeitig auf dem Bildschirm dargestellt werden können. Der gesamte Bildschirm wird sofort an diese Werte angepaßt.

Farbcode	Farbe
Ø	schwarz
1	grün
2	blau
3	cyan (türkis)
4	rot
5	gelb
6	magenta (purpur)
7	weiß

Farbtabelle für farbigen Bildschirm

BEMERKUNGEN: - Wird keine **COLOR**=-Anweisung benutzt, werden vom System die folgenden Defaultwerte gesetzt:

einfarbiger Schirm	Farb-Schirm	Farb-Code	Farbindex	
schwarz	schwarz	Ø	Ø	Hinter-grund
weiß	grün	1	1	Vorder-grund
	blau	2	2	
	rot	4	3	

Defaultwerte für Bildschirm

- Die zuerst genannte Farbe (Default-Wert:schwarz) ist die Hintergrundfarbe.
- Die an zweiter Position genannte Farbe ist die Vordergrundfarbe.
- Der Cursor wird in der zuletzt genannten Farbe dargestellt, am Acht-Farb-Schirm immer in weiß.
- Die Vorder- und Hintergrundfarbe kann mit der Anweisung **COLOR** für jedes Window definiert oder geändert werden (insbesondere wichtig für Schwarz/Weiß- und Acht-Farb-Schirm).
- Die mit **COLOR**= ausgewählten Farben gelten für alle Windows, d.h., in allen Windows kann am Vier-Farb-Schirm nur mit diesen Farben gearbeitet werden.

- Für den Vier-Farb-Schirm sind die bis zu vier ausgewählten Farben für alle Anweisungen, die später das Arbeiten mit Farben ermöglichen, unter den Nummern Ø bis 3 ('Farbindex') in der Reihenfolge der mit 'Farbcode' in der **COLOR**=-Anweisung ausgewählten Farben anzusprechen. Für den Schwarz/Weiß- und den Acht-Farb-Schirm bleibt der 'Farbcode' maßgebend.

- ACHTUNG: Vorherige **COLOR**-Anweisungen wirken sich auf die **COLOR**=-Anweisung aus (auch in neue Programme hinein)! Mit Hilfe von **CLEAR** kann der Default-Zustand wiederhergestellt werden.

BEISPIELE:

```
10 COLOR=3,6,4,7  '4 aus 8 Farben, nur Farbschirm
20 ' Hintergrund: türkis, Vordergrund: purpur,
30 ' außerdem folgende Farben: rot + weiß
40 COLOR 3,1 ' weiß wird Vordergrund, purpur Hintergrund
```

VERWEISE: Kapitel 4.3.14 und 4.3.18.3
Anweisung: **COLOR**

FUNKTION: Definition der Variablennamen, deren Werte bei der Verkettung von Programmen mit **CHAIN** übernommen bzw. übergeben werden

FORMAT: **COMMON** {Array( )|Variable} [,{Array( )|Variable}] ...

Array: Name des Arrays (String oder numerisch) (ggf. mit Typkennzeichen)

Variable: Name der String- oder numerischen Variablen (ggf. mit Typkennzeichen)

WIRKUNG: Die **COMMON**-Anweisung legt für das laufende Programm die Variablen fest, deren Werte an ein anderes Programm übergeben und/oder von einem anderen Programm übernommen werden sollen.

BEMERKUNGEN:
- **COMMON**-Anweisungen sind nur im Zusammenhang mit der Anweisung **CHAIN** zu verwenden.
- Ein Programm kann eine oder mehrere **COMMON**-Anweisungen enthalten.
- Die mit **CHAIN** verketteten Programme müssen die gleichen **COMMON**-Variablennamen verwenden.
- Beziehen sich Deklarationsanweisungen (wie **DEFINT, DEFSNG, DEFDBL** oder **DEFSTR**) auf **COMMON**-Variablen, sind diese im Folgeprogramm zu wiederholen. Dabei müssen die Deklarationsanweisungen jeweils vor den **COMMON**-Anweisungen stehen
- Die **CLEAR**-Anweisung löscht den **COMMON**-Bereich.

- Arrays, die über **COMMON**-Anweisung weitergereicht werden, müssen durch die Anweisung **DIM** dimensioniert sein. Dies kann vor oder nach der **COMMON**-Anweisung erfolgen.
- Eine Variable darf innerhalb der **COMMON**-Anweisungen nur einmal auftreten.

BEISPIELE:

```
10 COMMON A!():DIM A!(5,2)
30 FOR I%=1 TO 5:FOR K%=1 TO 2
40 A!(I%,K%)=I%*K%
50 NEXT K%,I%
60 CHAIN "1:FOLGEPROG.prg"
```

```
10 COMMON A!() 'FOLGEPROG.prg
20 FOR I%=1 TO 5:FOR K%=1 TO 2
30 PRINT A!(I%,K%)
40 NEXT K%,I%
```

VERWEISE: Kapitel 4.3.13

BEFEHL: **CONT**

FUNKTION: Fortsetzung des Programms nach einer Unterbrechung durch **STOP**-Anweisung, C oder Fehlerkondition ohne aktive Fehlerbehandlungsroutine

FORMAT: **CONT**

WIRKUNG: Die Programmausführung wird mit der Anweisung fortgesetzt, die auf diejenige Anweisung folgt, bei der das Programm unterbrochen wurde.

BEMERKUNGEN:

- Eine Programmunterbrechung kann erfolgen durch
  1. Betätigen der Taste C
  2. Ausführen der Anweisung **STOP**
  3. Auftreten einer Fehlermeldung
- Beim Auftreten einer Fehlermeldung kann mit der Programmausführung nur dann fortgesetzt werden, wenn die Fehlermeldung keinen Fehler anzeigt, der die sinnvolle Fortsetzung des Programms verhindert (vgl. Kapitel 4.3.12).
- Wird während der Ausführung der Anweisung **INPUT** oder **LINE INPUT** die Taste C gedrückt und anschließend mit **CONT** die Programmausführung fortgesetzt, wird die Anweisung **INPUT** bzw. **LINE INPUT** wiederholt.

- Wurde vor **CONT** im Edit-Mode gearbeitet bzw. im Command-Mode eine neue Anweisungszeile eingegeben, sind alle Variableninhalte wieder auf Ø bzw. Leerstring gesetzt sowie alle Stack-Speicher (z.B. für **GOSUB...RETURN** oder **FOR...NEXT**) gelöscht. In diesem Fall erscheint die Meldung "Cant't continue", falls z.B. in einer Schleife unterbrochen wurde.

BEISPIELE:

```
10 PRINT "Es folgt eine STOP-Anweisung;
   zur Fortsetzung CONT eingeben":STOP
20 PRINT "Es geht weiter."
```

VERWEISE: Kapitel 3
Anweisung: **STOP**

FUNKTION: COS

ZWECK: Liefert den Cosinus eines Winkels im Bogenmaß

FORMAT: **COS**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet, als Bogenmaß interpretiert und davon der Cosinus ermittelt.

BEMERKUNGEN:
- Das Argument ist im Bogenmaß anzugeben. Umrechnungen müßten ggf. über Unterprogramm oder eine vom Anwender definierte Funktion (s. **DEF FN**) erfolgen.
- Das Argument muß im Intervall -65537.574≤X≤ 65534.427 liegen, sonst erfolgt die Fehlermeldung "Illegal function call".
- Der Arcus-Cosinus kann berechnet werden über ARCCOS(X)=-**ATN**(X/**SQR**(-X*X+1))+1.5708

BEISPIELE:

```
10 PI!=3.14159:WIN!=40
20 PRINT "COS von";WIN!;"Grad =";SIN(WIN!*PI!/180)
RUN
COS von 40 Grad = .766045
```

VERWEISE: Kapitel 2.4 und 4.13

FUNKTION: **CSNG**
(convert single precision)

ZWECK: Konvertiert einen numerischen Wert in einfache Genauigkeit.

FORMAT: **CSNG(num.Ausdr.)**

WIRKUNG: Der numerische Ausdruck wird berechnet und gemäß Rundungsregeln in einfache Genauigkeit verwandelt.

BEMERKUNGEN:
- Die siebte signifikante Stelle von doppelt genauen Werten wird kaufmännisch gerundet.

ACHTUNG:

Werden doppelt genaue Werte außerhalb des für einfach genaue Werte zulässigen Intervalls (vgl. Kapitel 2.5.1.2) konvertiert, sind zwei Fälle zu unterscheiden:

1. Liegt der zu konvertierende, doppelt genaue Wert außerhalb des Intervalls -1.8Ø925139D75≤x≤1.8Ø925139D75, wird die Meldung "Overflow" gegeben. Es wird jedoch mit den verschiedensten völlig falschen Werten weitergerechnet!!!
2. Liegt der zu konvertierende doppelt genaue Wert zwar unterhalb des Intervalls zu 1., aber über dem Intervall -3.4Ø282D38≤x≤3.4Ø282D38, erfolgt keine Overflow-Meldung. Intern wird mit dem niedrigsten bzw. höchsten in einfacher Genauigkeit intern darstellbaren Wert (vgl. Intervall) weitergerechnet!

BEISPIELE:

```
10 DEFDBL X-Z:X=123456789#
20 PRINT USING(X)
RUN
 1.23457E+08
```

VERWEISE: Kapitel 2.4, 2.5.1.5 und 4.13

FUNKTION: Steuerung des Text- oder des Graphik-Cursors und gegebenenfalls Festlegung der Cursor-Eigenschaften

FORMAT: **CURSOR** [**POINT**] [(X,Y)] [Schalter] [, Rate] [, Darstellung]

X,Y : numerische Ausdrücke, deren Ergebnisse die Spalten- und Zeilen-Position des Cursors bestimmen

Schalter : numerischer Ausdruck, dessen Ergebnis gerundet wird und dann Ø oder 1 enthalten muß; gibt an, ob Cursor angezeigt werden soll oder nicht

Rate : numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen Ø und 2Ø enthalten muß; gibt an, wie oft der Cursor pro Sekunde blinken soll

Darstellung: Element eines eindimensionalen Integer-Arrays, ab dem die Cursor-Darstellung gespeichert ist.

WIRKUNG: Der alphanumerische oder der graphische Cursor wird in der durch den Parameter 'Darstellung' definierten Form auf die Position (X,Y) im aktiven Dokument gesetzt. Dabei bestimmt der Parameter 'Schalter', ob der Cursor sichtbar ist oder nicht und der Parameter 'Rate' legt die Rate fest, mit der der sichtbare Cursor blinkt.

POINT

Wird das Schlüsselwort **POINT** angegeben, bezieht sich die Anweisung auf den graphischen Cursor und somit auf die durch **SCALE** definierten Anwender-Koordinaten. Fehlt das Schlüsselwort **POINT**, wird der Text-Cursor über Textspalte und Textzeile angesprochen.

X,Y

Die Ergebnisse der numerischen Ausdrücke werden auf Ganzzahligkeit gerundet. Beim Text-Cursor bestimmt X die Text-Spalte und Y die Text-Zeile, in der der Cursor positioniert wird. Beim graphischen Cusor bestimmt X die horizontale und Y die vertikale Anwenderkoordinate (vgl. **SCALE**). Die Positionen beziehen sich grundsätzlich auf das aktive Window.

Schalter

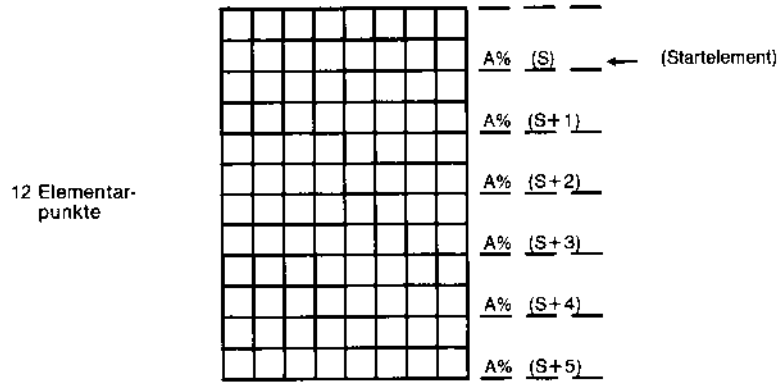
Hat das gerundete Ergebnis des numerischen Ausdrucks 'Schalter' den Wert Ø, wird der gesetzte Cursor nicht sichtbar angezeigt. Besitzt es den Wert 1, wird der Cursor auf der definierten Position angezeigt. Bei den Anweisungen **INPUT** bzw. **LINE INPUT** wird der Cursor grundsätzlich angezeigt.

Rate

Durch den Parameter 'Rate' wird die Rate bestimmt, mit der Cursor blinkt. Dabei können Werte von Ø bis 2Ø benutzt werden. Ist der Wert gleich Ø, bleibt der Cursor ständig sichtbar, d.h. er blinkt nicht. Ist der Wert z.B. 5, heißt das, der Cursor soll pro Sekunde 5 mal blinken.

Darstellung

Der Cursor besteht aus einer 8 x 12-Matrix von Elementarpunkte:



Ordnet man jeweils zwei Reihen von Elementarpunkten (à 8 Punkten) einem bestimmten Element eines eindimensionalen Integer-Arrays zu, kann die Darstellung des Cursors in sechs aufeinanderfolgenden Elementen eines Integer-Arrays gespeichert werden. Durch Angabe des ersten Elements im Array - im weiteren 'Startelement' genannt - dieses 6-elementigen (Teil)-Arrays (im Bild: Elemente mit den Indices S bis S+5) ist die Cursor-Darstellung eindeutig definiert.

In jedem Element des Integer-Arrays wird in 2 Bytes die 16-Bit-Maske einer Zweierreihe von Elementarpunkten gespeichert.

Die blinkenden Punkte ergeben sich aus dem jeweiligen Integerwert. Dort sind diejenigen Bits zu setzen, die bei Darstellung des Cursors in der entsprechenden Zweierreihe blinken sollen. Zur internen Darstellung von Integerwerten vgl. Kapitel 2.6.3, interne Speicherung von Integer-Werten im Arbeitsspeicher.

Beispiel:

Wird für das 6. Element des Arrays der Wert -1 vergeben (alle anderen seien Ø), sind dort alle 16 Bits gesetzt und die letzten zwei Zeilen des Cursors werden blinken.

BEMERKUNGEN:

- Es ist immer nur ein Cursor aktiv, entweder der graphische oder der alphanumerische.
- Die Anweisung **CURSOR** bezieht sich immer auf das aktive Window.
- Das Startelement des durch 'Darstellung' definierten eindimensionalen Arrays muß nicht das Element mit dem Index Ø oder 1 (bzgl. **OPTION BASE**) sein, sondern kann ein beliebiges Array-Element sein, dem 5 weitere direkt folgen.
- Die **CURSOR**-Anweisung hat solange Gültigkeit, bis ein neuer Cursor definiert wird, **CLEAR**-Anweisung erfolgt oder durch RESET bzw. Ausschalten des Systems wieder der Default-Cursor gesetzt wird.

- Eine neue Darstellung des Text-Cursors gilt nicht im Command- und Edit-Mode und nicht bei **INPUT**- bzw. **LINE INPUT**-Anweisungen.

BEISPIELE:

```
10 OPTION BASE 1        :DEFINT A-Z
30 A(1)=32:A(2)=4104:A(3)=3086:A(4)=3596
35 A(5)=2064:A(6)=8192
40 CURSOR(2,4)0 ' keine Anzeige
50 CURSOR(2,7)1 ' blinkend
60 CURSOR(2,8)1,10 '10 * /Sek. blinkend
70 CURSOR(2,11)1,4,A%(1)    'neue Form
80 CURSOR POINT (300,100)1,5  ' Graphik-Cursor
```

```
10 CLEAR:OPTION BASE 0:DIM CF%(7)
20 FOR I%=0 TO 4:CF%(I%)=0:NEXT
30 CF%(5)=-1 ' untere 2 Zeilen belegt
35 CURSOR(1,10):PRINT "Ihre Eingabe"
40 CURSOR(14,10)1,0,CF%(0):S$="":K$=""
50 WHILE  K$<>CHR$(13)
60 PRINT K$;:S$=S$+K$:K$=INPUT$(1):WEND
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Funktion: **WINDOW, POS**

FUNKTION: **CVD**
(convert double)

ZWECK: Verwandlung der ersten 8 Bytes eines Strings in einen doppelt genauen numerischen Wert

FORMAT: **CVD**(Stringausdr.)

WIRKUNG: Der Stringausdruck, der früher über **MKD$** gewonnen wurde, wird berechnet. Seine ersten acht Bytes werden in einen numerischen Wert (doppelt genaue Zahl) verwandelt.

BEMERKUNGEN:
- **CVD** ist die Umkehrfunktion von **MKD$**.
- Ist 'Stringausdruck' mehr als 8 Zeichen lang, werden nur die ersten acht Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als 8 Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:

```
10 CLEAR:D#=9.99999999999998D+149:S!=1E-20:I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 PRINT D$;LEN(D$);CVD(D$)
40 PRINT S$;LEN(S$);CVS(S$)
50 PRINT I$;LEN(I$);CVI(I$)
RUN
.P5 8  9.99999999999998D+149
< 4  1E-20
U 2  32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion: **MKD$**

FUNKTION: **CVI**
(convert integer)

ZWECK: Verwandlung der ersten 2 Bytes eines Strings in einen Integer-Wert

FORMAT: **CVI**(Stringausdr.)

WIRKUNG: Der Stringausdruck, der früher über **MKI$** gewonnen wurde, wird berechnet. Seine ersten zwei Bytes werden in einen numerischen Wert (Integerzahl) verwandelt.

BEMERKUNGEN:
- **CVI** ist die Umkehrung der Funktion **MKI$**.
- War ein Wert einfach oder doppelt genau, bevor er mit **MKI$** in einen String verwandelt wurde, wird bei der Rückumwandlung mit **CVI** sein ganzzahliger Teil (vgl. Kapitel 2.5.1.1) erzeugt.
- Ist 'Stringausdruck' mehr als 2 Zeichen lang, werden nur die ersten 2 Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als 2 Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:
```
10 CLEAR:D#=9.99999999999998D+149:S!=1E-20:I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 PRINT D$;LEN(D$);CVD(D$)
40 PRINT S$;LEN(S$);CVS(S$)
50 PRINT I$;LEN(I$);CVI(I$)
RUN
.P5 8  9.99999999999998D+149
( 4  1E-20
U 2  32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion: **MKI$**

FUNKTION: CVS
(convert single)

ZWECK: Verwandlung der ersten 4 Bytes eines Strings in einen einfach genauen numerischen Wert

FORMAT: **CVS**(Stringausdr.)

WIRKUNG: Der Stringausdruck, der früher über **MKS$** gewonnen wurde, wird berechnet. Seine ersten vier Bytes werden in einen numerischen Wert (einfache Genauigkeit) verwandelt.

BEMERKUNGEN:
- **CVS** ist die Umkehrung der Funktion **MKS$**.
- War ein Wert doppelt genau, bevor er mit **MKS$** in einen String verwandelt wurde, ist er nach der Rückumwandlung mit **CVS** an der siebten signifikanten Stelle kaufmännisch gerundet.
- Ist 'Stringausdruck' mehr als 4 Zeichen lang, werden nur die ersten 4 Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als 4 Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:
```
10 CLEAR:D#=9.99999999999998D+149:S!=1E-20:I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 PRINT D$;LEN(D$);CVD(D$)
40 PRINT S$;LEN(S$);CVS(S$)
50 PRINT I$;LEN(I$);CVI(I$)
RUN
.P5 8  9.99999999999998D+149
< 4  1E-20
ü 2  32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion: **MKS$**

FUNKTION: Erzeugung eines internen Files von Daten, die später mit **READ**-Anweisung(en) Variable(n) zugewiesen werden können

FORMAT: **DATA** { num.Konst. / Stringkonst. } [, { num. Konst. / Stringkonst. }] ...

WIRKUNG: Vor Beginn der Programmausführung wird im Arbeitsspeicher eine Tabelle erzeugt, die alle Konstanten aller **DATA**-Anweisungen des Programms enthält. Ein Pointer zeigt auf das erste Element der Tabelle. (Die Konstanten der Tabelle werden später mit **READ**-Anweisung(en) auf Variable(n) zugewiesen.) Nach jedem **READ** zeigt der Pointer auf das jeweils nächste Element der Tabelle. (Mit der Anweisung **RESTORE** kann der Pointer wieder auf das erste Element einer **DATA**-Zeile positioniert werden.)

BEMERKUNGEN:
- Die **DATA**-Anweisungen können an jeder Stelle im Programm stehen. Die Reihenfolge des Abstellens der Konstanten im internen Datenfile wird bestimmt durch die physische Reihenfolge der **DATA**-Anweisungen gemäß aufsteigender Zeilennummer.
- **DATA**-Anweisungen werden beim Programmlauf selbst nicht mehr ausgeführt.
- Folgen innerhalb einer Programmzeile einer **DATA**-Anweisung andere mit : angehängte Anweisungen, werden diese dem logischen Programmablauf gerecht abgearbeitet.

- **DATA**-Anweisungen können zwar angesprungen werden, erzeugen dabei aber keine neuen Elemente im internen Datenfile!
- Stringkonstanten in einer **DATA**-Anweisung, die am Anfang oder am Ende ein Blank oder irgendwo ein Komma enthalten, müssen in Anführungszeichen stehen. Ansonsten können die Anführungszeichen weggelassen werden.
- Anführungszeichen können nicht als Bestandteile von Strings übernommen werden.
- Die Konstanten-Typen (String, numerisch) in der **DATA**-Anweisung müssen mit den Variablen-Typen der **READ**-Anweisung übereinstimmen.
- **DATA**-Anweisungen können zwar im Direkt-Mode ohne Fehlermeldung eingegeben werden, haben aber keine Wirkung.

BEISPIELE:

```
10 FOR I%=1 TO 10:READ T$(10):NEXT
620 RESTORE 9060
630 FOR I%=1 TO 2:READ PSEUDO$:NEXT:READ NWERT!
9050 DATA 2.555,57,Frage1,3.45027," Name,Vorn.  "
9060 DATA 345,1111,5.66,-1,text3
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **READ, RESTORE**

DATE$

FUNKTION: von der Echtzeituhr verwaltetes Tagesdatum

FORMAT: **DATE$**

WIRKUNG: Die Variable **DATE$** enthält das aktuelle Datum in der Form:

TT/MM/JJJJ

Der Variablen **DATE$** können Werte zugewiesen werden, die ein gültiges Datum darstellen.
Gültig ist:

Ø1≤TT≤31	(Tag)
Ø1≤MM≤12	(Monat)
ØØ≤JJ≤99	(Jahr) oder
	1ØØØ≤JJJJ≤ 9999

und ein beliebiges druckbares Trennzeichen, das keine Ziffer sein darf.
Eine Zuweisung in diesem Format bewirkt eine Übernahme vom System als gültiges Datum. Wurde das Jahr zweistellig definiert, wird automatisch 19ØØ addiert.
Wird versucht, auf **DATE$** einen String zuzuweisen, der nicht diesem Format entspricht, erfolgt keine Zuweisung und das aktuelle Datum bleibt unverändert.

BEMERKUNGEN:
- Die Variable **DATE$** ist auch ohne Wertzuweisung definiert, da der Wert vom PCOS-Befehl **ss** her ständig zur Verfügung steht.

- Die Überprüfung des Tagesdatums durch das System selbst erfolgt nicht in Abhängigkeit vom geltenden Monat, d.h. der 31.02.82 ist ein nach dem Format zulässiges Datum.
- Das Datum wird von der Echtzeituhr richtig auch unter Berücksichtigung der Schaltjahre verwaltet. Das Datum wird vom System jeweils verändert, wenn die Uhrzeit ØØ/ØØ/ØØ beträgt (vgl. **TIME$**).
- **DATE$** darf nur in Anweisungen verwendet werden, die keine Wertänderung bewirken. Die Wertänderung ist nur über die Anweisung **LET...=** möglich. Das Schlüsselwort LET muß allerdings weggelassen werden.
- Liegt die US-ASCII-Tastatur statt einer europäischen Tastatur zugrunde, wird **DATE$** in der Form MM/TT/[JJ]JJ verwaltet.

BEISPIELE:

```
10 LINE INPUT "Datum ";D$
20 LINE INPUT "Zeit ";T$
30 C$="ss "+D$+","+T$:EXEC C$
40 ' oder:  DATE$=D$:TIME$=T$
60 PRINT DATE$,TIME$ ' nur bei zulässigem
       Format in D$ und T$ verändert!
```

VERWEISE: Kapitel 4.3.11
Spezielle Variable: **TIME$**
PCOS-Befehle: **ss,sl**

ANWEISUNG: DEFDBL

(define double precision)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als doppelt genaue numerische Variblen

FORMAT: **DEFDBL** Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.1]]...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä,Ö,Ü und ß); gibt den Anfangsbuchstaben der als doppelt genau vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen ohne Typkennzeichen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st. 1' (incl.) und 'Anf.b.st. 2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von # an den Namen - als doppelt genaue numerische Variablen interpretiert.

BEMERKUNGEN:
- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2', ebenso 'Anf.buchst.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFSNG, DEFINT** oder **DEFSTR** hebt **DEFDBL** auf. Dabei erhalten alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen ohne Typkennzeichen wieder den Default-Wert (Ø für numerische; ""(=Leerstring) für Strings)! Auch alle betreffenden Arrays werden auf DefaultGröße reduziert.

- Beim Verketten von Programmen mit Übergabe aller Variablen (vgl. Anweisung **CHAIN** mit Parameter **ALL**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen. Bei Anwendung der **COMMON**-Anweisung müssen die betreffenden Variablen neu definiert werden.
- **DEFDBL** bezieht sich bei vom Anwender definierten Funktionen (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.
- Durch Anhängen von **%, !** oder **$** an Variablennamen, die wegen **DEFDBL** normalerweise doppelt genaue Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.

  ACHTUNG:

  Ist z.B. **DEFDBL** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR# nicht mehr möglich. Eine Wertänderung von KUNDENNR bewirkt dann automatisch eine Wertänderung von KUNDENNR#.

BEISPIELE:

```
10 DEFDBL V-X,Z
20 X=12345678#    ' identisch mit X#=12345678#
30 X%=7   ' andere Variable
40 Y=123456789#   ' wird zu einfacher Genauigkeit!
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2

Anweisungen: **DEFINT, DEFSNG, DEFSTR, DEF FN, COMMON, CHAIN**

ANWEISUNG: DEF FN

FUNKTION: Definition einer numerischen oder einer Stringfunktion innerhalb einer BASIC-Zeile durch den Anwender

FORMAT: **DEF** { **FN**Var.Name (Parameterliste) = num.Ausdruck
FNStr.Var.Name (Parameterliste) = Stringausdr. }

Var.Name : beliebiger numerischer Variablenname mit Ausnahme eines Arrays

Str.Var.Name : beliebiger Stringvariablenname mit Ausnahme eines Arrays

Parameterliste: Folge von numerischen und/oder String-Variablen, die durch Komma getrennt werden. Es sind keine Array-Elemente zugelassen.

WIRKUNG: Die in der **DEF**-Anweisung innerhalb der Klammern stehenden Parameter sind lokale Variablen, die keinen Bezug zu gleichnamigen Variablen außerhalb der Funktion haben und deren Inhalt im Gesamtprogramm nicht beeinflussen. Beim Funktionsaufruf (s. **FN**) sind diese Parameter durch entsprechende Ausdrücke zu ersetzen, wodurch den lokalen Variablen der aktuelle Wert der Ausdrücke zugewiesen wird.

Auf der rechten Seite vom Gleichheitszeichen können außer den Elementen der Parameterliste auch andere Variablen, als globale Variablen, benutzt werden.

Die Funktion **FN**Var.Name oder **FN**Str.Var.Name wird durch den Ausdruck auf der rechten Seite definiert. Die Funktion wird in einem Programm jedesmal ausgeführt, wenn ihr Name **FN**Var.Name oder **FN**Str.Name zusammen mit den aktuellen Parametern im Programm aufgerufen wird (vgl. **FN**).

BEMERKUNGEN:

- Eine Funktion darf nur ein einziges Mal, und zwar vor dem ersten Aufruf, definiert werden und in dieser Form nur eine Zeile und eine Anweisung umfassen.
- Verwendet man bei der Definition einer Funktion einen Stringvariablennamen, ist das Ergebnis der Funktion ein String; benutzt man den Namen einer num.Variablen, ist das Ergebnis ein numerischer Wert. Der Typ des Ergebnisses wird durch Anhängen des gewünschten Typenkennzeichens an den Variablennamen festgelegt.
- Für den Variablennamen/Stringvariablennamen sind Typendeklarationen, z.B. mit **DEFSTR**, möglich.
- Wird eine Funktion vor ihrer Definition aufgerufen, erfolgt die Meldung "Undefined user function".
- Es dürfen in der Parameterliste nicht mehr als zwölf Variablennamen (lokale Variablen) verwendet werden. Es muß aber mindestens ein Variablenname darin vorkommen.
- Direkte rekursive Aufruffolgen
  (z.B. 1Ø **DEF FNA$(Y)=FNA$(X)+" "+R$**) sind unzulässig.

- Aufrufe anderer bereits definierter Funktionen rechts vom Gleichheitszeichen sind möglich (z.B. 3Ø **DEF FNS$(**X,R$**)**= **FNA$(**X**)**+" "+R$).
- Die beim Aufruf der Funktion in der Klammer eingereichten Parameter müssen in Typ, Anzahl und Reihenfolge mit den in der 'Parameterliste' aufgeführten lokalen Variablen übereinstimmen.
- Funktionsdefinitionen dürfen nicht im Direkt-Mode eingegeben werden.
- Syntaxfehler in **DEF FN**-Anweisungen werden beim Aufruf der Funktion durch **FN** gemeldet.
- ACHTUNG:
  Durch **DEFINT, DEFSTR, DEFDBL, DEFSNG** getroffene Vereinbarungen beziehen sich auch auf den 'Var. Namen' und auch auf die in der 'Parameterliste' angegebenen lokalen Variablen!

BEISPIELE: numerische selbstdefinierte Funktion:

```
30 DEF FNNETTO#(BRUTTO#,MWSTSATZ!)=BRUTTO#/
(1+MWSTSATZ!/100):PRINT FNNETTO#(114,14)
```

selbstdefinierte Stringfunktion:

```
40 DEFSTR T
50 DEF FNTFUNKT(T,U1%,U2%)=CHR$(U1%)+T+CHR$(U2%)
```

VERWEISE: Kapitel 4.3.7
Anweisung: **FN**

ANWEISUNG: **DEFINT**

(define integer)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als numerische Integer-Variablen

FORMAT: **DEFINT** Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.1]]...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä,Ö,Ü und ß); geben den Anfangsbuchstaben der als 'Integer' vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen ohne Typkennzeichen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st. 1' (incl.) und 'Anf.b.st. 2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von **%** an den Namen - als numerische Integer-Variablen interpretiert.

BEMERKUNGEN:
- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2', ebenso 'Anf.buchst.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFDBL, DEFSNG** oder **DEFSTR** hebt **DEFINT** auf. Dabei erhalten alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen ohne Typkennzeichen wieder den Default-Wert (Ø für numerische; ""(=Leerstring) für Strings)! Auch alle betroffenen Arrays werden auf Default-Größe reduziert.

- Beim Verketten von Programmen mit Übergabe aller Variablen (vgl. Anweisung **CHAIN** mit Parameter **ALL**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen. Bei Anwendung der **COMMON**-Anweisung müssen die betreffenden Variablen neu deklariert werden.
- **DEFINT** bezieht sich bei einer vom Anwender definierten Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.
- Durch Anhängen von **!**, **#** oder **$** an Variablennamen, die wegen **DEFINT** normalerweise Integer-Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.

  ACHTUNG:

  Ist z.B. **DEFINT** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR**%** nicht mehr möglich. Eine Wertänderung von KUNDENNR**%** bewirkt dann automatisch eine Wertänderung von KUNDENNR.

BEISPIELE:

```
10 DEFINT V-X,Z
20 X=12345        ' identisch mit X%=12345
30 X!=7   ' andere Variable
40 Y=123456789#   ' wird zu einfacher Genauigkeit!
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2

Anweisungen: **DEFDBL, DEFSNG, DEFSTR, DEF FN, COMMON, CHAIN**

ANWEISUNG: **DEFSNG**

(define single precision)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als einfach genaue numerische Variablen

FORMAT: **DEFSNG** Anf.b.st.1 [-Anf.b.st.2] [, Anf.b.st.k [-Anf.b.st.l]]...

Anf.b.st.: Klein- und Grpßbuchstabe (außer ä, ö, ü, Ä, Ö, Ü und ß); geben den Anfangsbuchstaben der als einfach genau vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen ohne Typkennzeichen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st.1' (incl.) und 'Anf.b.st.2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von ! an den Namen - als einfach genaue numerische Variable interpretiert.

BEMERKUNGEN:
- Eine neue Typvereinbarung durch **DEFDBL**, **DEFINT** oder **DEFSTR** hebt eine vorher erfolgte auf. Dabei erhalten alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen ohne Typkennzeichen wieder den Default-Wert (Ø für numerische, "" (=Leerstring) für Strings)! Auch alle betroffenen Arrays werden auf Default-Größe reduziert.

- Beim Verketten von Programmen mit Übergabe aller Variablen (vgl. Anweisung **CHAIN** mit Parameter **ALL**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen. Bei Anwendung der **COMMON**-Anweisung müssen die betreffenden Variablen neu deklariert werden.
- **DEFSNG** bezieht sich bei einer vom Anwender definierten Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.
- Durch Anhängen von **%**, **#** oder **$** an Variablennamen, die wegen **DEFSNG** normalerweise einfach genaue Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.

  ACHTUNG:

  Ist z.B. **DEFSNG** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR! nicht mehr möglich. Eine Wertänderung von KUNDENNR! bewirkt dann automatisch eine Wertänderung von KUNDENNR.

BEISPIELE:

```
10 DEFSNG V-X,Z
20 X=123456       ' identisch mit X!=123456
30 X%=7   ' andere Variable
40 Y=123456789#   ' wird zu einfacher Genauigkeit!
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2

Anweisungen: **DEFINT, DEFDBL, DEFSTR, DEF FN, COMMON, CHAIN**

ANWEISUNG: **DEFSTR**

(define string)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben und ohne Typkennzeichen als String-Variablen

FORMAT: **DEFSTR** Anf.b.st.1 [-Anf.b.st.2] [, Anf.b.st.k [-Anf.b.st.l]]...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä, ö, ü, Ä, Ö, Ü und ß); geben den Anfangsbuchstaben der als String vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen ohne Typkennzeichen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st.1' (incl.) und 'Anf.b.st.2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von **$** an den Namen - als String-Variablen interpretiert.

BEMERKUNGEN:
- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2' ebenso 'Anf.b.st.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFDBL**, **DEFSNG** oder **DEFINT** hebt eine vorher erfolgte auf. Dabei erhalten alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen ohne Typkennzeichen wieder den Default-Wert (Ø für numerische)!

- Beim Verketten von Programmen mit Übergabe aller Variablen (vgl. Anweisung **CHAIN** mit Parameter **ALL**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen. Bei Anwendung der **COMMON**-Anweisung müssen die betreffenden Variablen neu deklariert werden.
- **DEFSTR** bezieht sich bei einer Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.
- Durch Anhängen von **%**, **!** oder **#** an Variablennamen, die wegen **DEFSTR** normalerweise String-Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.

  ACHTUNG:

  Ist z.B. **DEFSTR** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNAME und KUNDENNAME**$** nicht mehr möglich. Eine Wertänderung von KUNDENNAME**$** bewirkt dann automatisch eine Wertänderung von KUNDENNAME.

BEISPIELE:

```
10 DEFSTR T
20 X="Guten Tag"   'FALSCH: Es muß heißen:
20 X$="Guten Tag"
30 T="Hallo"       'OK wegen DEFSTR T
40 T=1234          'FALSCH: Es muß heißen:
40 T!=1234
```

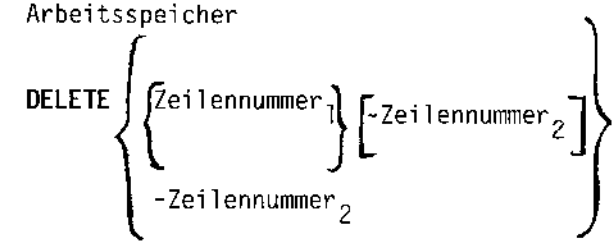
VERWEISE: Kapitel 2.5.3.1 und 4.3.2

Anweisungen: **DEFINT, DEFSNG, DEFDBL, DEF FN, COMMON, CHAIN**

BEFEHL: DELETE

FUNKTION: Löschung einer oder mehrerer Programmzeilen im Arbeitsspeicher

FORMAT:



$Zeilennumer_1$: ganze positive Zahl, die die Nummer der zu löschenden Zeile oder die erste Zeile des zu löschenden Abschnitts angibt

$Zeilennummer_2$: ganze positive Zahl, die die letzte Zeile des zu löschenden Abschnitts angibt

WIRKUNG: Die durch '$Zeilennummer_1$' angegebene Programmzeile oder alle Programmzeilen des durch '$Zeilennummer_1$' und '$Zeilennummer_2$' (inklusive) begrenzten Abschnitts werden im Arbeitsspeicher gelöscht. Ist der erste Operand der Punkt (.), wird für '$Zeilennummer_1$' die zuletzt angesprochene oder abgearbeitete Programmzeile genommen. Bei Angabe von '-$Zeilennumer_2$' allein werden alle Programmzeilen vom Anfang des Programms bis zu '$Zeilennummer_2$' gelöscht.

BEMERKUNGEN:
- Die im Befehl angegebenen Zeilennummern müssen im Programm vorhanden sein (sonst: "Illegal function call").

- Die Eingabe von 'Zeilennummer' und unmittelbares Drücken einer Eingabeabschlußtaste hat dieselbe Wirkung wie **DELETE** Zeilennummer.
- Nach Ausführen des Befehls **DELETE** befindet sich das System im Direkt-Mode.

BEISPIELE:

```
DELETE 2255-3450

DELETE -150
```

VERWEISE: Kapitel 3.3

ANWEISUNG: DIM

FUNKTION: Festlegung der Dimension und der maximalen Werte der Indices von Arrays

FORMAT: **DIM** Arrayname(Indexliste) [,Arrayname(Indexliste)]
...
Format der Indexliste:
num.Ausdr.1 [,num.Ausdr.2] ... [, num.Ausdr.k]
... [,num.Ausdr.n]
Arrayname: Name des Arrays
num.Ausdr.k: Gibt den höchsten Wert des Index in der Dimension k an
n: Dimension des Arrays

WIRKUNG: Der num.Ausdr.k wird berechnet und gerundet. Für jeden angegebenen Array wird die Dimension und der maximale Wert der Indices definiert.

BEMERKUNGEN:
- Für nicht mit **DIM** definierte Arrays gilt: Der Default-Wert der höchsten Indices beträgt 10.
- In Abhängigkeit von **OPTION BASE** ist der kleinste mögliche Index-Wert eines Arrays gleich 0 oder 1.
- Der vom Interpreter zugelassene Höchstwert für die Dimension ist 255, und der höchste Index pro Dimension beträgt 32767. Die tatsächlichen Obergrenzen ergeben sich aus der Größe des jeweils vorhandenen Arbeitsspeichers.

- Die **DIM**-Anweisung muß vor dem ersten Auftreten des entsprechenden Arrays stehen.
- Erfolgen Typvereinbarungen durch **DEFINT, DEFSNG, DEFDBL** oder **DEFSTR**, müssen diese vor den **DIM**s abgearbeitet werden, da durch **DEF...** alle betroffenen **DIM**s wieder aufgehoben werden.
- Werden den Elementen der Arrays keine Werte zugewiesen, haben die Elemente der numerischen Arrays den Wert Ø und die Elemente der Stringarrays den Wert Leerstring (="").
- Verschieden dimensionierte Arrays dürfen nicht den gleichen Namen haben. (z.B. ist **DIM** A(2), A(3,3) nicht zulässig).
- Zum Löschen von Arrays (er ist dann nicht mehr dimensioniert) siehe **ERASE**.
- Da für die Indices numerische Ausdrücke zugelassen sind, ist eine dynamische Dimensionierung von Arrays möglich. Vor einer Redimensionierung muß die Anweisung **ERASE** verwendet werden. Ansonsten erfolgt die Fehlermeldung "Duplicate definition".

BEISPIELE: siehe Anweisung **OPTION BASE**

```
0 CLEAR:DEFINT Q:T%=250
10 DIM P!(5,2,9),Q11(2*T%),TE$(2,3,5)
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisungen: **OPTION BASE, ERASE**
Fehler-Codes: 7, 9, 1Ø

FUNKTION: Festlegung von Zeichenvorgängen

FORMAT: **DRAW** [% Window-Nr.] Befehlsstring

Window-Nr. : numerischer Ausdruck; 1 bis 16

Befehlsstring: Stringkonstante oder Stringvariable, die eine Befehlsfolge enthält.

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, wird das angesprochene Window gewählt. Fehlt dieser Parameter, bezieht sich die Anweisung auf das aktive Window. Der Befehlsstring beschreibt, wie gezeichnet werden soll. Der Befehlsstring besteht aus Buchstaben und Koordinatenwerten. Die Form der Darstellung kann mit der Vorstellung einer "Bewegungen" durchführenden "Feder" verbunden werden.
Koordinaten-Angaben werden hier mit 'x' und 'y' bezeichnet, wenn es Absolutkoordinaten sind, und mit 'dx' und 'dy' bezeichnet, falls es sich um Relativkoordinaten handelt.
Folgende Elemente für den Befehlsstring sind zulässig:

B Dieses Element kann anderen Elementen vorangestellt werden und bedeutet, daß nur Bewegungen der "Feder" ohne Zeichnen ausgeführt werden. Ist **B** nicht angegeben, wird eine Zeichenbewegung ausgeführt.

C Farbindex Bewirkt die Vorwahl der durch 'Farbindex' angegebenen Farbe, bezogen auf die für das Window gültige **COLOR**-Anweisung

J x,y Es wird die Feder linear zum Punkt mit den Koordinaten (x,y) bewegt

M dx,dy Bewirkt eine lineare Federbewegung um dx und dy Einheiten

L dx Bewirkt eine Federbewegung um dx nach links

R dx Bewirkt eine Federbewegung um dx nach rechts

U dy Bewirkt eine Federbewegung um dy nach oben

D dy Bewirkt eine Federbewegung um dy nach unten

Jedem dieser Befehlselemente außer **C** kann ein zusätzlicher Operand nachgestellt werden, der eine logische Operation für jeden Elementarpunkt bestimmt. Dieser Operand entspricht in seiner Wirkung denen der Anweisungen **LINE, CIRCLE** und **PUT %**. Diese Operanden sind:

P für **PSET**
X für **XOR**
O für **OR**
R für **PRESET**
N für **NOT**
A für **AND**

Sie müssen ohne Trennzeichen unmittelbar nach dem Befehlselement **J, M, L, R, U** oder **D** folgen.

BEMERKUNGEN:

- Alle Angaben von Koordinaten (absolut und relativ) beziehen sich auf die gültige **SCALE**-Anweisung.
- Die Buchstaben für die Befehle können als Klein- oder Großbuchstaben im Befehlsstring enthalten sein.
- Koordinatenwerte müssen als num.Konstanten oder Variablen im String enthalten sein und sind durch Komma zu trennen. Wird ein Koordinatenwert durch eine Variable angegeben, muß die Variable durch Voran- und Nachstellen des Zeichens '=' gekennzeichnet werden. Die Verwendung von Blanks zwischen Befehlscodes und Koordinaten ist beliebig.
- In einem Befehlsstring können hintereinander mehrere Befehlselemente mit ihren Operanden definiert sein.
- Der Befehlsstring muß den Vorschriften entsprechen. Ist der String fehlerhaft, erfolgt die Fehlermeldung "Syntax error".

BEISPIELE:

```
10 CLEAR:N%=WINDOW(2,32)
20 WINDOW%N%:FOR J%=1 TO 2
26 SCALE %J% 0,100,0,100:CLS %J%:NEXT
30 DRAW "BJ 0,0":DRAW "J100,100":DRAW "D50"
40 X1%=20:Y1%=40:RELX%=-5:RELY%=50
50 FIGUR$="BJ=X1%=,=Y1%= M=RELX%=,=RELY%="
60 DRAW %1 FIGUR$
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **SCALE, COLOR=, COLOR**
Funktion: **WINDOW**

BEFEHL: **EDIT**

FUNKTION: Ändern bestehender Programmzeilen eines Programms im Arbeitsspeicher

FORMAT: **EDIT** {Zeilennummer / .}

Zeilennummer: ganze positive Zahl, die die Nummer der zu ändernden Programmzeile angibt

WIRKUNG: Der Command-Mode wird verlassen und der Edit-Mode gewählt. Die Zeilennummer der durch 'Zeilennummer' angegebenen Programmzeile wird angezeigt. Nach einem Edit-Subbefehl können nun Änderungen in der Programmzeile vorgenommen werden.
Wird anstelle von 'Zeilennummer' der Parameter '.' verwendet, bezieht sich **EDIT** auf die laufende Zeile.

Edit-Subbefehle

Edit-Subbefehle werden verwendet, um den Cursor innerhalb der Zeile zu bewegen oder um Zeichen ändern, löschen, einfügen oder anfügen zu können (vgl. auch Kapitel 3.4.4).
Edit-Subbefehle werden am Bildschirm nicht angezeigt. Werden Zeichen, die keinen Subbefehl bilden, eingegeben, ertönt das akustische Signal.
Anmerkung:
In der nachfolgenden Beschreibung bedeuten:

Zeichen ein beliebiges Zeichen

Text a) eine beliebige Zeichenfolge mit beliebiger Länge

b) eine beliebige Zeichenfolge mit genau n Zeichen, falls dem Subbefehl der Parameter 'n' vorangestellt ist.

n Ganze positive Zahl, die einigen Subbefehlen vorangestellt werden kann und einen Wiederholungsfaktor für diesen Subbefehl darstellt. Ist 'n' nicht angegeben, wird 1 angenommen.

Subbefehle zur Cursorsteuerung

Blank bewegt den Cursor um eine Stelle nach rechts. Das soeben durchlaufene Zeichen wird angezeigt. Danach ist ein neuer Subbefehl nötig. Wird das Zeilenende erreicht, haben nachfolgende Blanks keine Wirkung.

nBlank bewirkt die n-fache Widerholung des Subbefehls Blank. Danach ist ein neuer Subbefehl nötig.

H Bewegt den Cursor um eine Stelle nach links (zurück). Die Zeichen, um die der Cursor nach links verschoben wird, werden nicht mehr angezeigt. Die durchlaufenen Zeichen bleiben erhalten. Ausnahme: nach den Subbefehlen **I**, **H** oder **X**. Werden mehr **H** angegeben, als der Cursorposition innerhalb der Zeile entspricht, bleibt der Cursor am ersten Zeichen hinter der Zeilennr., das nicht Blank ist.

n H Bewirkt die n-fache Wiederholung des Subbefehls H. Es wird der Cursor um n-Stellen nach links oder an den Zeilenanfang gesetzt. Alle Zeichen, die dabei durchlaufen werden, werden in der Reihenfolge des Durchlaufens, also in umgekehrter Reihenfolge, angezeigt.

Subbefehle für das Einfügen von Zeichen

IText Die als 'Text' eingegebene Zeichenfolge wird rechts von der aktuellen Cursorposition eingefügt.

Während des Einfügens kann H wie bei Eingaben außerhalb des Edit-Modes zum Löschen von fehlerhaft eingegebenen Zeichen verwendet werden.

Wird durch das Einfügen versucht, in einer Zeile mehr als 255 Zeichen einzugeben, ertönt das akustische Signal und weitere Zeichen werden nicht akzeptiert.

Das Einfügen kann beendet werden durch:

HOME beendet das Einfügen und der Edit-Mode bleibt erhalten. Der Cursor steht an der Stelle nach dem eingefügten Zeichen. Danach ist ein neuer Subbefehl nötig. Auch CR beendet das Einfügen.

XText Erlaubt das Anfügen von Zeichen an eine bestehende Zeile. Es wird der Cursor an die Stelle nach dem letzten Zeichen der Zeile gesetzt. Alle unter **I** beschriebenen Möglichkeiten stehen ohne weiteren Subbefehl zur Verfügung.

Subbefehle für das Löschen von Zeichen

D löscht das Zeichen rechts vom Cursor. Danach ist ein neuer Subbefehl nötig.

n**D** löscht n Zeichen beginnend ab der Cursorposition. Danach ist ein neuer Subbefehl nötig. Ist n größer als die Anzahl der Zeichen rechts vom Cursor, wird die Zeile bis zum Ende gelöscht. Gelöschte Zeichen werden zwischen dem Zeichen Ö angezeigt (bei deutschem Zeichensatz). Der Cursor befindet sich vor dem ersten nicht gelöschten Zeichen.

H Es werden alle Zeichen rechts vom Cursor bis zum Zeilenende gelöscht und das Einfügen von Zeichen wie beim Subbefehl I ermöglicht. Der Subbefehl **H** ist besonders zum Ersetzen von Anweisungen oder Anweisungsteilen ab einer bestimmten Zeilenposition sinnvoll.

Subbefehle für das Ersetzen von Zeichen

CZeichen Das Zeichen rechts von der Cursorposition wird durch das eingegebene Zeichen ersetzt. Nach Eingabe des Zeichens ist ein neuer Subbefehl nötig.

nCText Es werden - beginnend ab dem Zeichen rechts von der Cursorposition - 'n' Zeichen durch die im 'Text' angegebene Zeichenfolge ersetzt. Danach ist ein neuer Subbefehl nötig. Es müssen unbedingt 'n' Zeichen eingegeben werden.
Enthält 'Text' mehr als 'n' Zeichen, werden die nachfolgenden Zeichen aus 'Text' ignoriert. Der Cursor befindet sich nach **C** an der Stelle vor dem ersten nicht geänderten Zeichen.

Subbefehle für das Suchen von Zeichen

SZeichen
nSZeichen
Es wird das 'n'-te Auftreten von 'Zeichen' in der Zeile ab der Cursorposition gesucht und der Cursor an die Stelle vor diesem Zeichen gesetzt. Danach ist ein neuer Subbefehl nötig. Das Zeichen an der aktuellen Cursorposition wird in die Suche nicht einbezogen. Wird das 'Zeichen' nicht oder nicht genügend oft gefunden, wird der Cursor an das Ende der Zeile gesetzt. Die bei der Suche durchlaufenen Zeichen werden angezeigt.

KZeichen
nKZeichen
Der Subbefehl **K** hat eine ähnliche Wirkung wie **S**, es werden jedoch alle beim Suchen durchlaufenen Zeichen gelöscht. Danach ist ein neuer Subbefehl nötig. Der Cursor wird an die Stelle vor dem gefundenen 'Zeichen' gesetzt. Die gelöschten Zeichen werden zwischen Ö angezeigt, (bei deutschem Zeichensatz).

Subbefehle für die Steuerung

CR Bewirkt die Anzeige der restlichen Zeile, speichert die durchgeführten Änderungen und beendet den Edit-Mode.

E Die Änderungen werden gespeichert und der Edit-Mode verlassen. Im Gegensatz zu CR wird der Zeilenrest jedoch nicht angezeigt.

Q Beendet den Edit-Mode ohne Speicherung der durchgeführten Änderungen. Die Zeile bleibt in ihrer ursprünglichen Form gespeichert.

L Bewirkt die Ausgabe der restlichen Zeile. Die Zeilennummer wird erneut angezeigt und der Cursor befindet sich an der ersten Stelle der Zeile.

A Erlaubt den Neubeginn des Änderns einer Zeile. Die ursprüngliche Zeile wird wieder bereitgestellt und der Cursor wieder an die erste Stelle der Zeile gesetzt.

BEMERKUNGEN:

- Der Edit-Mode wird vom System automatisch gewählt, wenn während der Programmausführung ein Syntax error erkannt wird.
- Editing löscht alle Variablenwerte. Sollen daher (z.B. nach einem Syntaxfehler) zuerst Variablenwerte überprüft werden, muß der Edit-Mode mit dem Subbefehl **Q** verlassen werden.
- Wird versucht, **EDIT** für ein mit dem Parameter **P** geschütztes Programm auszuführen, oder wird der Edit-Mode dort automatisch infolge eines Syntaxfehlers aufgesucht, wird "Illegal function call" gemeldet.

BEISPIELE:

```
10 PRIMT "Hallo"
EDIT
```

VERWEISE: Kapitel 3.4
PCOS-Befehle: **ed**

ANWEISUNG: **END**

FUNKTION: Festlegung eines logischen Programmendes

FORMAT: **END**

WIRKUNG: Beendigung der Programmausführung, Schließen aller Files und Puffer und Übergang in den Command-Mode

BEMERKUNGEN:
- Die Anweisung **END** kann an jeder Stelle in einem Programm stehen.
- Das Herbeiführen eines Programmendes ohne Ausführung der **END**-Anweisung gewährleistet nicht, daß die benutzten Files ordnungsgemäß geschlossen werden.
- Die Ausführung der **END**-Anweisung verursacht bei interpretativem Arbeiten die Meldung 'Ok' und nicht "Break in line nnnnn" wie bei Anwendung der Anweisung **STOP**.
- Die Meldung 'Ok' erscheint auch, wenn die letzte Anweisung eines BASIC-Programms interpretativ abgearbeitet ist, ohne daß eine **END**-Anweisung das physische Programmende anzeigt.
- Ein Programm kann keine oder mehrere **END**-Anweisungen haben.

BEISPIELE:
```
10 ' 1. Programmzeile
                :
999 END    ' 1. logisches Programmende
                :
1090 K$=INKEY$:IF K$<>"" THEN END ' 2. Programmende
```

VERWEISE: Kapitel 4.3.5

FUNKTION: **EOF**
(end of file)

ZWECK: Abfrage auf Ende eines sequentiellen Files beim Lesen von sequentiellen Files

FORMAT: **EOF(**Filenr.**)**

WIRKUNG: Die Funktion **EOF** liefert den Wahrheitswert -1, falls beim Versuch eines Lesens (Anweisungen **INPUT#** oder **LINE INPUT#** bzw. Funktion **INPUT$(...** ,#Filenr.**)**) von einem sequentiellen File das Fileende erreicht wurde.

BEMERKUNGEN:
- Wird der Versuch eines Inputs über das Fileende hinaus gemacht, ohne mit **EOF** abzufragen, wird der Fehler 62 ("Input past end") gemeldet.

BEISPIELE:

```
10 OPEN "O",1,"0:DATEN.seq"
20 FOR I%=1 TO 10:PRINT#1,I%:NEXT
30 CLOSE 1:OPEN "I",1,"0:DATEN.seq"
40 WHILE NOT EOF(1)
50 INPUT#1,I%:PRINT I%
60 WEND:CLOSE 1      :END
```

VERWEISE: Kapitel 4.3, 10.2
Anweisungen: **INPUT#** , **LINE INPUT#**
Funktion: **INPUT$**
Fehler-Code: 62

OPERATOR: **EQV**
(equivalent)

FUNKTION: Verknüpfung zweier logischer Ausdrücke mit Hilfe der Äquivalenz-Verknüpfung bzw. Bildung der Bijunktion zweier Integerwerte (Bit-für-Bit-Verknüpfung mit **EQV**)

FORMAT: {log.Ausdruck / Integerwert} **EQV** {log.Ausdruck / Integerwert}

WIRKUNG: Das Ergebnis der **EQV**-Verknüpfung zweier Bedingungen ist
-1 (wahr), wenn beide Bedingungen erfüllt oder beide nicht erfüllt sind (gleicher Wahrheitswert beider Bedingungen);
Ø (falsch), wenn die beiden Bedingungen sich im Wahrheitswert unterscheiden.
Bei der **EQV**-Verknüpfung zweier Integerwerte wird die **EQV**-Operation Bit für Bit durchgeführt und das Ergebnis berechnet (vgl. Kapitel 2.6.4).

BEISPIELE:

```
10 A%=5:B%=3:C!=2.47
15 C%=A% EQV B%  ' C% ist jetzt -7
20 IF A%(B% EQV C!)3 THEN GOSUB 1000 ELSE GOSUB 2000
```

VERWEISE: Kapitel 2.6.4
Operatoren: **NOT, AND, OR, XOR, IMP**

ANWEISUNG: ERASE

FUNKTION: Aufheben der Dimensionierung eines Arrays

FORMAT: **ERASE** Arrayname [,Arrayname ...]

Arrayname: Name des Arrays (ohne Klammern und Indices)

WIRKUNG: Für jeden angegebenen Array wird die Dimensionierung, die in der zuletzt durchlaufenen **DIM**-Anweisung für den/die Array/s gegeben wurde,aufgehoben.

BEMERKUNGEN:
- Nach der Aufhebung der Dimensionierung kann der Array neu dimensioniert werden; ansonsten gelten die Default-Werte (siehe **DIM**-Anweisung).
- Es kann mit **ERASE** auch die Dimensionierung von Arrays aufgehoben werden, die mit Default-Werten dimensioniert wurden,also ohne **DIM**-Anweisung.

BEISPIELE:

```
10 CLEAR:DIM ZAHL!(49)
20 FOR I%=1 TO 49 ZAHL!(I%)=RND:NEXT
820 ERASE ZAHL!:DIM ZAHL!(49) ' alle
    Elemente auf 0
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisungen **DIM, OPTION BASE**

RESERVIERTE VARIABLE: ERL

ZWECK: enthält die Zeilennr. derjenigen Programmzeile, in der der zuletzt entdeckte BASIC-Fehler auftrat bzw. die Zeilennr., in der die letzte **ERROR**-Anweisung auftrat

FORMAT: **ERL**

EINSATZ: Mit Hilfe einer Anweisung der Art

IF ERL=Fehlerzeile **THEN** Zeilennr. / Anweisung(sfolge) ...

kann eine Behandlung von aufgetretenen ("echten" bzw. mit **ERROR** "künstlich" erzeugten) Fehlern in Abhängigkeit von der Programmzeile, in der der Fehler auftrat (='Fehlerzeile'), erreicht werden. Ein Fehler führt immer zu sofortigem Programmabbruch und Fehlermeldung, wenn er nicht in einer Fehlerbehandlungsroutine bearbeitet wird (**ON ERROR GOTO**-Anweisung). (Ausnahme: gewisse Fälle des "Overflows"; vgl. z.B. Kapitel 2.5.1.2).

BEMERKUNGEN:
- **ERL** enthält einen ganzzahligen Wert.
- Wird im Gegensatz zur obigen Formulierung die Abfrage in der Form

  IF Fehlerzeile=**ERL THEN ...**

  durchgeführt, wird bei einer Neu-Numerierung des Programms (vgl. Befehl **RENUM**) der Parameter 'Fehlerzeile' nicht angepaßt.

- Auf die reservierte Variable **ERL** kann mit **LET** kein Wert zugewiesen werden, sondern nur über Umwege mit der Anweisung **ERROR**.
- Tritt ein Fehler im Direkt-Mode auf, enthält die reservierte Variable **ERL** stets den Wert 65535. Diese Bedingung kann mit

  IF 65535=**ERL THEN ...**

  in einer aktiven Fehlerbehandlungsroutine abgefangen werden.

BEISPIELE:

```
9  ON ERROR GOTO 1001
10 OPEN "I",1,"1:KONTROLLFILE"

1001 IF ERL=10 AND ERR=57 THEN PRINT "Schreib-/
Lesefehler (Schreibschutz?)":X$=INPUT$(1):RESUME 0
1999 ON ERROR GOTO 0:STOP ' sonstige Fehler
```

VERWEISE: Kapitel 4.3.12 und 6
Anweisungen: **ON ERROR GOTO, ERROR, RESUME**
Reservierte Variable: **ERR**

RESERVIERTE VARIABLE: ERR

ZWECK: enthält den Fehler-Code des zuletzt entdeckten BASIC-Fehlers bzw. nach einer **ERROR**-Anweisung den dadurch zugewiesenen Wert

FORMAT: **ERR**

EINSATZ: Mit Hilfe einer Anweisung der Art

IF ERR=Fehler-Code **THEN** Zeilennr. / Anweisung(sfolge) ...

kann eine Behandlung von aufgetretenen ("echten" bzw. mit **ERROR** "künstlich" erzeugten) Fehlern in Abhängigkeit vom Fehler-Code erreicht werden.
Ein Fehler führt jedoch immer zu sofortigem Programmabbruch und Fehlermeldung, wenn er nicht in einer Fehlerbehandlungsroutine bearbeitet wird (**ON ERROR GOTO**-Anweisung). (Ausnahmen: gewisse Fälle des "Overflows"; vgl. Kapitel 2.5.1.2;).

BEMERKUNGEN:
- **ERR** enthält einen ganzzahligen numerischen Wert.
- Auf die reservierte Variable **ERR** kann ein Wert nur mit der Anweisung **ERROR** zugewiesen werden.
- PCOS-Fehler werden nicht in **ERR** festgehalten; nach Auftreten der meisten PCOS-Fehler enthält **ERR** jedoch den Wert 40.
- Tritt im Direkt-Mode ein Syntax-Fehler auf, wird dieser zwar gemeldet, **ERR** hat aber danach den Wert Ø.

- **ERR** hat den Wert Ø, nachdem mit **RESUME** aus einer Fehlerbehandlungsroutine ins Hauptprogramm zurückverzweigt worden ist.

BEISPIELE:

```
9  ON ERROR GOTO 1001
10 OPEN "I",1,"1:KONTROLLFILE"

1001 IF ERL=10 AND ERR=57 THEN PRINT "Schreib-/
Lesefehler (Schreibschutz?)":K$=INPUT$(1):RESUME 0
1999 ON ERROR GOTO 0:STOP ' sonstige Fehler
```

VERWEISE: Kapitel 4.3.12 und 6
Anweisungen: **ON ERROR GOTO, ERROR, RESUME**
Reservierte Variable: **ERL**

FUNKTION: Zuweisung eines Fehler-Codes auf die reservierte Variable **ERR** oder Simulation eines BASIC-Fehlers durch den Anwender

FORMAT: **ERROR** num.Ausdr.

WIRKUNG: Der numerische Ausdruck wird berechnet und zur Ganzzahligkeit kaufmännisch gerundet.
Entspricht der Wert einem standardmäßig für den Interpreter definierten Fehlercode, wird die entsprechende Fehlermeldung gegeben und das Programm angehalten bzw. bei vorhandener Fehlerbehandlungsroutine in diese verzweigt (vgl. **ON ERROR GOTO**).
Anderenfalls wird der Fehler "Unprintable error" gemeldet und das Programm angehalten.
Dies geschieht nicht, wenn in einer vorher aktivierten Fehlerbehandlungsroutine der betreffende Fehlercode, z.B. mit einer "künstlichen" Fehlermeldung, behandelt wird.

BEMERKUNGEN:
- Das Ergebnis des 'numerischen Ausdruck' muß größer als Ø und kleiner als 255 sein.
- Es empfiehlt sich, "künstlichen" Fehlercodes Werte über 82 (höchster dem Interpreter bekannter Fehler-Code) zu geben.
- PCOS-Fehler können nicht simuliert werden.

BEISPIELE:

```
5 ON ERROR GOTO 1001
752 IF LEFT$(TIME$,2)>"17" THEN ERROR 125
900 ' Arbeit nach 17:00 Uhr

1210 IF ERR=125 THEN PRINT "Keine Lust mehr":RESUME 900
```

VERWEISE: Kapitel 4.3.11, 4.3.12 und 6
Anweisungen: **ON ERROR GOTO, RESUME**
Reservierte Variablen: **ERR, ERL**

ANWEISUNG: **EXEC**

FUNKTION: Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine

FORMAT: **EXEC** { Stringausdruck / "Name d. Routine Liste d. Parameter" }

Stringausdruck: enthält Namen der Routine und Parameter in der Form, wie sie in der PCOS-Ebene einzugeben sind

Name d. Routine: bezeichnet den Namen der Assembler-Routine

Liste d. Parameter: Folge von Integer- und/oder Stringkonstanten, die durch Komma getrennt sind; muß angegeben werden wie in der PCOS-Ebene

WIRKUNG: Das System prüft, ob die aufgerufene Assembler-Routine bereits im Arbeitsspeicher vorhanden ist. Falls nicht, wird die Routine auf der Diskette gesucht und in den Arbeitsspeicher geladen und abgearbeitet. Der momentane Zustand des Arbeitsspeichers bleibt erhalten.

BEMERKUNGEN:
- **EXEC** erledigt die gleichen Aufgaben wie **CALL**; es können aber Variablen nur in Form von Strings über Stringverarbeitung an PCOS übergeben werden. Von PCOS können keine Variablen übernommen werden.

- Mit der Anweisung **EXEC** können keine Werte von der Assembler-Routine an das BASIC-Programm übergeben werden.
- Die Parameter in 'Liste der Parameter' müssen in der richtigen Reihenfolge und formatgerecht (Integer oder String) sein.
- Sollen Parameter in Form von Variablen übergeben werden, müssen diese über Stringverarbeitung in den Stringausdruck eingebracht werden. Für nicht zu ändernde Parameter ist nur der Platzhalter , zu setzen (vgl. PCOS-Handbuch).
- Zur Beschleunigung des Aufrufs von Assembler-Routinen kann der PCOS-Befehl **pl** verwendet werden. Danach beleiben die Routinen ständig im Arbeitsspeicher erhalten.
- Wechsel des Ein-/Ausgabe-Mediums (**+/- s/d ...**) darf nur mit **EXEC** erfolgen.

BEISPIELE:

```
EXEC "vl +dprt: 1:":EXEC "-dprt:"
```

VERWEISE: Kapitel 4.3.15, ANHANG IV
Anweisung: **CALL**
Funktion: **VARPTR**

FUNKTION: **EXP**
(exponent)

ZWECK: Berechnung einer Potenz der Eulerschen Zahl e

FORMAT: **EXP(num.Ausd.)**

WIRKUNG: Der numerische Ausdruck wird berechnet. Die Eulersche Zahl e wird mit diesem Wert potenziert.

BEMERKUNGEN:
- e selbst ist mit **EXP(1)** zu bestimmen (einfache Genauigkeit!).
- e kann über eine doppelt genaue Variable mit dem Inhalt 2.7182818284590 5# genauer verwaltet werden.
- Ist das Argument größer als 88, wird "Overflow" gemeldet.
- Ist das Argument kleiner als - 87, wird mit Ø weitergerechnet.

BEISPIELE:

```
10 PRINT EXP(1),EXP(2.645)
 2.71828   14.0834
```

VERWEISE: Kapitel 2.4 und 4.3.6

ANWEISUNG: FIELD
(Feldvereinbarung)

FUNKTION: Vereinbarung der Feldlängen in einem bestimmten Random-File-Puffer und damit des Record-Aufbaus; Festlegung von Variablennamen für die Felder, Festlegung ihrer Länge und Position im Random-File-Puffer sowie Bereitstellung des Random-File-Puffers für folgende **GET**- oder **PUT**-Anweisungen

FORMAT: **FIELD** [#] Filenr.,Feldlänge **AS** Feldvariable [,...]

(Die Folge ,Feldlänge **AS** Feldvariable kann wiederholt werden.)

Filenr.: num.Ausdr. (Ergebnis: 1-15)

Feldlänge: num.Ausdr. (Ergebnis: größer Ø, aber kleiner oder gleich der bei **OPEN** festgelegten Record-Länge)

Feldvariable: Stringvariable

WIRKUNG: Für das unter 'Filenr.' geöffnete Random-File wird im betreffenden Random-File-Puffer Platz und Position für die 'Felddvariablen' vereinbart, und zwar jeweils für die in 'Feldlänge' vereinbarte Anzahl Zeichen (Bytes). Die Felder können von da an unter den 'Feldvariablennamen' angesprochen werden.

BEMERKUNGEN:
- In einem Random-File sind alle Daten in Form von Strings gespeichert.

- Die Daten können nur über Stringvariablen vom File in den Puffer bzw. vom Puffer in das File übertragen werden (**GET** bzw. **PUT**).
- Die logische Länge des Random-File-Puffers ist vorher mit **OPEN**-Anweisung zu definieren.
- Durch **FIELD** werden automatisch die dort genannten Feldvariablen auf die aktuelle Länge 'Feldlänge' gebracht.
- Die 'Feldvariablen' des Puffers dürfen ihre Inhalte nur mit Hilfe der Anweisungen **LSET** bzw. **RSET** oder durch Lesen mit Hilfe von **GET** erhalten.
- Die **FIELD**-Anweisung überträgt keine Variableninhalte in den Puffer, sondern stellt nur Platz im Puffer bereit.
- Nach der **FIELD**-Anweisung sind die Feldvariablen mit nicht verwendbaren Zeichenfolgen (abhängig vom Status von BASIC) gefüllt. Sie müssen deshalb zuerst über **GET** bzw. **RSET** oder **LSET** sinnvolle Anfangswerte erhalten.
- Für dieselbe 'Filenr.' können beliebig viele **FIELD**-Anweisungen gegeben werden. Bei mehreren **FIELD**-Anweisungen für dieselbe 'Filenr.' gelten alle gleichzeitig. Die verschiedenen **FIELD**-Anweisungen beschreiben dann jeweils eine andere Aufteilung des Puffers (unterschiedliche Recordaufbauten bzw. Feldstrukturen, evtl. mit unterschiedlichen Längen der Felder). Der Record (Puffer) mit der bei **OPEN** angegebenen 'Recordlänge' muß jedoch immer als Ganzes in einer **FIELD**-Anweisung beschrieben werden.

- Für zwei verschiedene offene Files (**OPEN**-Anweisung) dürfen in den jeweiligen **FIELD**-Anweisungen keine Feldvariablennamen gleich sein. Für dasselbe File kann bei Gültigkeit mehrerer verschiedener **FIELD**-Anweisungen in diesen derselbe 'Feldvariablenname' mit verschiedener 'Feldlänge' verwendet werden.
- Überschreitet 'Feldlänge' bzw. die Summe der Feldlängen aller für eine bestimmte 'Filenr.' vereinarten 'Feldvariablen' die bei **OPEN** festgelegte Record-Länge, wird der Fehler "FIELD overflow" gegeben. Die Default-Recordlänge ist *256* Bytes.

  ACHTUNG:

  Nach einer **FIELD**-Anweisung dürfen die Inhalte der darin aufgeführten Feldvariablen nur noch durch **LSET** oder **RSET** bzw. **GET** verändert werden. Andere Zuweisungen (z.B. über **LET** oder **INPUT**) verschieben die Pointer-Position in den Daten-Bereich des Arbeitsspeichers!!! Dies führt dazu, daß falsche Werte in das Random-File geschrieben werden.

BEISPIELE:

```
10 OPEN "R",1,"10:RDATEN1.rnd",128
20 OPEN "R",2,"10:RDATEN2.rnd",24
5001 GOSUB 9101 'Record-Typ 1 von RDATEN1.rnd
5010 ' Verarbeitung
5031 GOSUB 9201 ' Felder von RDATEN2.rnd
5610 GOSUB 9111 'Record-Typ 2 von RDATEN1.rnd
5615 ' Verarbeitung
6999 CLOSE 1,2 END
9101 FIELD 1,2 AS I$(1),16 AS K$(1),4 AS H$(1),10 AS
E$(1),32 AS DUMMY1$,2 AS I$(2),16 AS K$(2),4 AS H$(2)
10 AS E$(2),32 AS DUMMY2$                    :RETURN
9111 FIELD 1,2 AS ZAEHLER$ 1 AS SCHALTER$,120 AS SONST$,
5 AS FILLER1$                                :RETURN
9201 FIELD 2,24 AS GESAMTPUFFER$              RETURN
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, RSET, LSET, GET, PUT**

Funktionen: **MKI$, MKS$, MKD$**

Fehler-Code: 5Ø

BEFEHL: **FILES**

FUNKTION: Ausgabe des kurzen Inhaltsverzeichnisses einer Diskette oder der Namen von Files mit gemeinsamen Eigenschaften auf dem Bildschirm

FORMAT: **FILES** [Diskettenspezifikation: / Filename(nauswahl)]

Diskettenspezifikation: Stringausdruck; Ergebnis muß einen String ergeben, der den Regeln zur Bildung von Diskettenspezifikationen entspricht (vgl. Kapitel 4.1.3)

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3); mit Hilfe der Parameter * und ? kann eine Auswahl bestimmter Files mit gemeinsamen Eigenschaften im Namen getroffen werden (vgl. PCOS-Handbuch).

WIRKUNG: Es wird das kurze Inhaltsverzeichnis der Diskette ausgegeben. Ist kein Operand angegeben, wird das Inhaltsverzeichnis der Diskette angezeigt, die zuletzt durch einen Befehl oder eine Anweisung angesprochen wurde. Bei Angabe des Operanden 'Diskettenspezifikation' wird das Inhaltsverzeichnis der durch 'Diskettenspezifikation' bestimmten Diskette ausgegeben. Ist der Operand 'Filename' angegeben, wird nur der Name des angesprochenen Files angezeigt.

Ist eine 'Filenamenauswahl' angegeben, werden die Namen der dadurch selektierten Files ausgegeben. Im Falle, daß nur die Filespezifikatior angegeben ist, muß die Station, auf der sich das File befindet, aktiv sein. Der Befehl **FILES** gibt folgende Informationen:

1. Station (**Ø** oder **1**), in welcher sich die Diskette befindet
2. Diskettenbezeichnung
3. Anzahl freier Sektoren auf der Diskette.

Anschließend werden die File-Identifier - in 4 Spalten aufgeteilt - in der Reihenfolge ausgegeben, wie sich die Files im Inhaltsverzeichnis der Diskette befinden.

BEMERKUNGEN:

- Der PCOS-Befehl **vq** entspricht dem BASIC-Befehl **FILES.**
- Soll das kurze Inhaltsverzeichnis am Drucker ausgegeben werden, ist die Anweisung

  CALL "vq Diskettenspezifikation: **+dprt:"**

  einzugeben.
  Das ausführliche Inhaltsverzeichnis kann durch

  CALL "vl Disk-Spez.:Filenamen(auswahl) **+dprt:"**

  abgerufen werden.
  Anschließend muß in beiden Fällen **EXEC "-dprt:"** eingegeben werden, um den Drucker wieder abzustellen.

BEISPIELE:

```
FILES "1:"
FILES "DISKA:TEST1.prg"
FILES "10:*.prg"
```

VERWEISE: Kapitel 4.1.3
PCOS-Befehle: **vq**, **vl**

FUNKTION: FIX

ZWECK: Ermittlung des ganzzahligen Teils eines numerischen Werts

FORMAT: **FIX**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet.

Vom Ergebnis wird der Nachkomma-Anteil abgeschnitten (ohne Rundung).

BEMERKUNGEN: - Das Ergebnis ist in der Genauigkeit des Arguments.

BEISPIELE: **FIX**(3.7) liefert 3 und **FIX**(-3.7) liefert -3
FIX(3.2) liefert 3 und **FIX**(-3.2) liefert -3

```
 10 A!=2.8572
 20 PRINT A!;FIX(A!)
RUN
 2.8572  2
```

VERWEISE: Kapitel 2.4 und 4.13
Funktionen: **INT, CINT**

FUNKTION: **FN**

ZWECK: Aufruf einer vom Anwender definierten Funktion

FORMAT: { **FN**Var.Name / **FN**Str.Var.Name } (Liste von Ausdrücken)

Var.Name, Str.Var.Name: Siehe Anweisung **DEF FN**

Liste von Ausdrücken: Folge von num. oder Stringausdrücken, die durch Komma getrennt sind.

WIRKUNG: Die einzelnen Ausdrücke werden berechnet und entsprechend der Parameterliste an die lokalen Variablen in der entsprechenden **DEF FN**-Anweisung übergeben. Anschließend wird die Funktion berechnet. Der Typ des Ergebnisses (num. Wert, String) hängt ab vom Namen der Funktion.

BEMERKUNGEN:
- Die Elemente von 'Liste von Ausdrücken' müssen in Typ (!!) und Anzahl mit den Elementen von 'Parameterliste' der zugehörigen **DEF FN**-Anweisung übereinstimmen.
- siehe Bemerkungen bei der Anweisung **DEF FN**

BEISPIELE:

```
10 DEF FNSUM#(S1#,S2#)=S1#+S2#
30 S1#=7:A#=9:B!=-6
40 PRINT FNSUM#(A#,CDBL(B!)):S1#
RUN
 3  7
```

VERWEISE: Kapitel 4.3.7
Definition einer Funktion durch den Anwender siehe Anweisung **DEF FN**
Fehler-Code: 18

FUNKTION: Startanweisung für die mehrmalige Ausführung von Anweisungen innerhalb einer Schleife

FORMAT: **FOR** Laufvar.=Anf.wert **TO** Endwert [**STEP** Schr.weite]

.
.
.

NEXT [Laufvar.]

Laufvar.: numerische Variable; darf kein Array-Element und nicht doppelt genau sein
Anf.wert: numerischer Ausdruck (Anfangswert)
Endwert: numerischer Ausdruck
Schr.weite: numerischer Ausdruck (Schrittweite)

WIRKUNG: Die zwischen **FOR** und **NEXT** liegende Anweisung(sfolge) wird so oft ausgeführt, wie es sich aus den Parametern 'Anfangswert', 'Endwert' und 'Schrittweite' ergibt.

1. Stößt das Programm auf eine **FOR**-Anweisung, wird zu dieser die zuzuordnende **NEXT**-Anweisung ermittelt.
2. Der 'Laufvariablen' wird der 'Anfangswert' zugewiesen. Es wird der aktuelle Wert des 'Endwerts' und der 'Schrittweite' ermittelt und im Arbeitsspeicher festgehalten. Diese ermittelten Werte stellen von nun an feste, der betreffenden **FOR...NEXT**-Schleife zugeordnete Parameter dar.

3. Vor Ausführung der folgenden BASIC-Anweisungen wird überprüft, ob der 'Endwert' überschritten ist (falls die 'Schrittweite' größer Ø ist) bzw., ob der 'Endwert' unterschritten ist (falls die 'Schrittweite' kleiner Ø ist).
4. Ist 'Endwert' über- bzw. unterschritten, wird mit derjenigen Anweisung fortgefahren, die auf die vorher ermittelte, der **FOR**-Anweisung zugeordnete **NEXT**-Anweisung folgt.
5. Anderenfalls werden die auf **FOR.....** folgenden Anweisungen ausgeführt.
6. Stößt das Programm dann auf die **NEXT**-Anweisung, wird zum aktuellen Wert der Laufvariablen der Wert der 'Schrittweite' addiert. (Im Falle von negativer 'Schrittweite' vermindert sich der Wert von 'Laufvariable' entsprechend.) Dieser Wert wird an die 'Laufvariable' zugewiesen und das Programm verzweigt wieder zu Punkt 3.

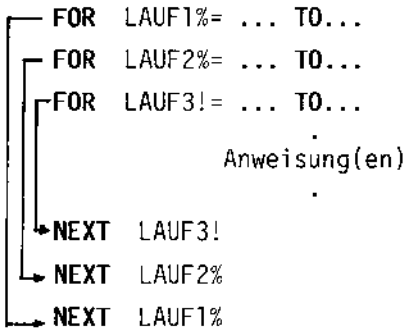
BEMERKUNGEN:
- Ist die Schrittweite nicht angegeben, wird der Default-Wert 1 angesetzt.
- Ist die 'Schrittweite' Ø, wird die Schleife nie verlassen, falls der Endwert anfangs nicht sofort über- bzw. unterschritten ist.
- Zur Beschleunigung der Schleife empfiehlt es sich (wenn aufgrund der 'Schrittweite' möglich), die 'Laufvariable' als Integer zu deklarieren (z.B. durch Anhängen von % an den Variablennamen).

- Ist die 'Laufvariable' ein Array-Element, erscheint die Fehlermeldung "Syntax error". Ist die 'Laufvariable' doppelt genau definiert, wird "Type mismatch" gemeldet.
- Der Name der 'Laufvariablen' bei der **NEXT**-Anweisung kann weggelassen werden. Die **NEXT**-Anweisung bezieht sich dann auf diejenige davorstehende **FOR**-Anweisung, die noch nicht aufgrund von 'Endwert'-Unter/Überschreitung abgearbeitet ist.
- Jeder **FOR**-Anweisung muß genau eine **NEXT**-Anweisung zugeordnet sein, sonst Fehlermeldung "FOR without NEXT".
- Der Wert von 'Laufvariable ' wird während der Abarbeitung verändert.

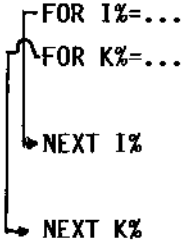
Verschachtelung von **FOR...NEXT**-Schleifen:

- Fallen mehrere **NEXT**-Anweisungen an einer Programmstelle zusammen, kann dies durch die Angabe aller betroffenen 'Laufvariablen' hinter einem **NEXT**, getrennt durch Komma, erklärt werden.
  Diese Formulierung kann jedoch bei bestimmten Datenkonstellationen zu einer fehlerhaften Anzahl von Schleifendurchläufen führen!

Die 'Laufvariablen' müssen in der Reihenfolge genannt werden, daß die Zuordnung nach folgendem Schema möglich ist:



Eine Verschachtelung nach dem Schema



oder ähnlichen ist nicht zulässig (Fehlermeldung: "NEXT without FOR").

- In verschachtelten **FOR...NEXT**-Schleifen müssen sich die Namen der 'Laufvariablen' unterscheiden.

- Achtung:

  Bei vorzeitigem Aussprung aus **FOR-NEXT**-Schleifen (z.B. mit **GOTO**) ohne späteren Rücksprung bleibt ein Stack-Speicher belegt. Dies kann bei wiederholtem Auftreten in einem Programm zu Speicher-Overflows führen. Zum Löschen von Stack-Speichern in solchen Fällen siehe Beispiel in Kapitel 4.3.6.

BEISPIELE:

```
5 INPUT N%:IF N%)20 OR N%<1 THEN 5
10 S!=1:FOR I%=N% TO 2 STEP -1
20 S!=S!*I%
30 NEXT
40 PRINT N%:"Fakultät =";S!
```

```
1 LINE INPUT "Name ";N$
10 FOR I%=1 TO LEN(N$)
20 PRINT ASC(MID$(NAM$,I%,1)) :NEXT
```

```
1 DEF FNP!(X!)=X!^3
10 SCALE -2,2,FNP!(-2),FNP!(2)
30 FOR I!=-2 TO 2 STEP .1
40 PSET(I!,FNP!(I!)):NEXT I!
```

VERWEISE: Kapitel 4.3.5 und 4.3.6
Anweisung: **NEXT, CLEAR**
Fehler-Codes: 1,7,2,6

ZWECK: Ermittlung des im Arbeitsspeicher noch freien Platzes (in Bytes) und ggf. Bereinigung des Arbeitsspeichers

FORMAT: FRE({Ø / Stringausdr.})

WIRKUNG: Bei Angabe der Ø wird nur der freie Platz (in Bytes) im Arbeitsspeicher ermittelt.
Bei Angabe von 'Stringausdr.' wird zusätzlich der Datenbereich im Arbeitsspeicher bereinigt: nach Stringmanipulationen (insbesondere -verlängerungen) nicht mehr benötigte Bereiche werden entfernt; alle Strings werden in aktueller Länge hintereinander neu abgelegt.

BEMERKUNGEN:
- Ist 'Stringausdruck' eine Variable, wird ihr Wert nicht verändert.
- Eine Bereinigung des Arbeitsspeichers kann bis zu 2 Minuten dauern.
- Durch eine Bereinigung des Arbeitsspeichers kann einem Fehler 14 ("Out of string space") eine Zeitlang vorgebeugt werden.
- Nach einer Bereinigung des Arbeitsspeichers wird eine neue Bereinigung weniger Zeit benötigen und die Rechengeschwindigkeit möglicherweise größer sein.

- Eine Bereinigung des Arbeitsspeichers kann nur durch **FRE**(Str.Ausdr.) erzwungen werden. Bei starker Stringverarbeitung im Programm sollte dies gelegentlich vorgenommen werden.; erfolgt dies nicht, nimmt der Interpreter evtl. selbständig eine Bereinigung vor. Dies kann zu plötzlichen längeren "Verarbeitungspausen" führen.

BEISPIELE:

```
10 PRINT FRE(0)' freie Bytes
20 Z!=FRE("A") ' freie Bytes + garbage collection
```

VERWEISE: Kapitel 4.3.4 und 4.13
Anweisung: **CLEAR**
Fehler-Code: 14

ANWEISUNG: **GET**
(bei Random-Files)

FUNKTION: Übertragen eines logischen Records von einem Random-File in den Random-File-Puffer ("Lesen").

FORMAT: **GET** [#] Filenr. [,Record-Nr.]

Filenr.: num.Ausdruck (Ergebnis: 1-15)
Record-Nr.: num.Ausdr. (Ergebnis 1-32767)

WIRKUNG: Von dem unter 'Filenr.' geöffneten Random-File wird der durch 'Record-Nr.' spezifizierte Record gelesen und in die mit der/den **FIELD**-Anweisung(en) für die betreffende 'Filenr.' definierten Feldvariablen aufgeteilt ("Lesen"). Es werden soviel Bytes in den Random-File-Puffer übertragen, wie beim zugehörigen **OPEN** mit 'Recordlänge' festgelegt wurde.

BEMERKUNGEN:
- Nach **GET** stehen die Feldinhalte in den Feldvariablen der **FIELD**-Anweisung(en) für 'File-Nr.' zur Verfügung.
  Diejenigen Stringvariablen, die numerische Werte beinhalten, können mit Hilfe der Funktionen **CVI, CVS** oder **CVD** zurückverwandelt werden.
- Records, die vorher noch nicht beschrieben wurden (vgl. **PUT**), sind mit so vielen Zeichen mit ISO-Code Ø belegt, wie bei **OPEN** unter 'Record-Länge' angegeben wurde. Dies muß nicht immer der Fall sein (vgl. zweite Bemerkung am Ende von Kapitel 4.3.10).

- 'Record-Nr.' muß ein positiver Wert sein; Ø liefert die Fehlermeldung "Bad record number"; der maximale Wert ist 32767.
- Wird 'Record-Nr.' weggelassen, wird derjenige Record des Files mit 'Filenr.' in den Puffer übertragen, der auf den letzten durch **GET** gelesenen oder mit **PUT** geschriebenen Record folgt ("sequentielles Lesen").
- Mit der Funktion **LOC** kann die letzte angesprochene Record-Nr. abgefragt werden.
- Wird ein **GET**-Versuch über das File-Ende hinaus gemacht, wird kein Fehler gemeldet, sondern über das eigentliche Ende des Files hinaus gelesen!!!
- Der Inhalt des Puffers oder Teile können auch mit **INPUT#** bzw. **LINE INPUT#** Variablen(listen) zugewiesen werden.

BEISPIELE:

```
10 OPEN "R",1,"DFILE.rnd",1  'zeichenweise bearbeiten
20 FIELD 1,1 AS BYTE$
30 INPUT "ab Byte Nr. ";N%:Z%=N%
40 PRINT "Welche Zeichen "
45 Z$=INPUT$(1):PRINT Z$
50 LSET BYTE$=Z$:PUT 1,Z%
60 Z%=Z%+1:IF Z%>32767 THEN 70 ELSE  45
70 Z%=1:CLS:WHILE Z$<>CHR$(0)
80 GET 1,Z% PRINT BYTE$;
90 WEND     :CLOSE 1:   END
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, FIELD, INPUT#, LINE INPUT#**

Funktionen: **EOF, LOC, LOF, CVI, CVS, CVD**

Fehler-Code: 54, 62, 63

FUNKTION: Ein rechteckiger (Teil-)Bereich eines Windows wird in einem Array gespeichert

FORMAT: **GET**[%Windownr.,](X1,Y1)-(X2,Y2),Arrayelement

Window-Nr. : numerischer Ausdruck, Wert: 1 bis 16

X1, Y1 : numerische Ausdrücke, die den (linken) Anfangspunkt des Bereiches definieren

X2, Y2 : numerische Ausdrücke, die den (rechten) Endpunkt des Bereichs definieren

Arrayelement: Bezeichnung des Elements eines eindimensionalen Integer-Arrays, bei dem mit der Speicherung begonnen wird

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, wird das entsprechende Window gewählt. Fehlt dieser Parameter, bezieht sich die Anweisung auf das aktive Window.

Die Koordinatenpaare (X1,Y1) und (X2,Y2) definieren, bezogen auf die gültige **SCALE**-Anweisung, den linken unteren und den rechten oberen Endpunkt eines achsenparallelen Rechtsecks. Der diesem Rechteck entsprechende Bereich der Bitmap wird in den angegebenen Integer-Array so übertragen, daß das angegebene Arrayelement die Anzahl Elementarpunkte des Rechtecks in der horizontalen Richtung enthält, das nächste die Anzahl in vertikaler Rich-

tung und jedes weitere Arrayelement je 16 Bits der Bitmap, aufsteigend von links oben nach rechts unten.

BEMERKUNGEN:

- Die maximal benötigte Anzahl der erforderlichen Arrayelemente ergibt sich mit mindestens (Anzahl Elementarpunkte im abzuspeichernden Bereich)/16 + Anzahl Elementarpunktzeilen.
- Liegen Ausgangs- und Endpunkt des Rechtecks außerhalb des Windows, wird nur der im Window liegende Teil des Rechtecks erfaßt.
- Ist der Integer-Array zu klein, um den angegebenen Bereich aufzunehmen, wird "Parameter out of range" gemeldet.
- Zur Ermittlung der benötigten Anzahl Elemente im Integer-Array empfiehlt sich folgendes Vorgehen: Array mit sehr großem Maximalindex dimensionieren, mit **GET%** gewünschten Bereich einlesen, anschließend mit einer Schleife (z.B. im Direkt-Mode) abfragen, ab welchem Array-Element nur noch Ø besteht; danach den maximalen Array-Index entsprechend reduzieren.
- Wurden Bilder mit PCOS-Release 1.3 oder älter im Arbeitsspeicher gespeichert und in einem sequentiellen oder Random-File auf Diskette abgespeichert und sollen diese unter Release 2.Ø nach Einlesen mit **PUT%** wieder abgebildet werden, muß vorab der betreffende File-Inhalt für Release 2.Ø konvertiert werden.
  Dazu existiert das BASIC-Hilfsprogramm getconv.bas.

BEISPIELE: Siehe **PUT %**

```
10 CLEAR:SCALE 0,511,0,255:CURSOR POINT 0
20 OPTION BASE 0:DIM SPEICHER%(15000)
40 FOR I%=1 TO 7
50 CIRCLE(250,180-5*I%),40+5*I%,,1.2
60 CIRCLE(250,80+5*I%),40+5*I%
70 CIRCLE(295+5*I%,130),40+5*I%,,.8
80 CIRCLE(215-5*I%,130),40+5*I%
90 NEXT
100 GET (0,0)-(511,255),SPEICHER%(0)
110 FOR I=1 TO 5000:NEXT
120 CLS COLOR 0,1:PRINT "Nach Abspeichern
im RAM durch GET% erfolgt jetzt PUT%"
130 PUT (0,0)-(511,255),SPEICHER%(0)
140 CALL "ep" 'nur graphikfähige Drucker
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **PUT %, SCALE**
Funktion: **WINDOW**

FUNKTION: Bewirkt den Sprung zu einer bestimmten Anweisung, bei der ein Unterprogramm beginnt

FORMAT: **GOSUB** Zeilennummer

WIRKUNG: Das Programm fährt bei der durch die Zeilennumer definierten Anweisung fort. Die zuletzt ausgeführte Anweisung des Unterprogramms muß **RETURN** sein, damit das Programm zu der Anweisung nach **GOSUB** zurückspringt.

BEMERKUNGEN:
- In einem Unterprogramm können auch mehrere **RETURN**-Anweisungen vorkommen.
- Ein Unterprogramm kann auch **GOSUB**-Anweisungen enthalten. Durch **RETURN** wird jedesmal zu der Anweisung nach dem letzten **GOSUB** gesprungen.
- Die Anzahl der möglichen Schachtelungen von Unterprogrammen ist abhängig vom Platz, der bei der Ausführung des Programms im Arbeitsspeicher zur Speicherung der Rücksprungadressen freibleibt (Stack-Speicher).
- Der rekursive Aufruf von Unterprogrammen ist möglich.
- Wird versucht, eine nicht vorhandene Zeilennummer anzuspringen, wird "Undefined line number" gemeldet.

BEISPIELE:

```
40 UX=1:GOSUB 100 'Sprung ins 1. UP
45 GOSUB 300      'Sprung ins 3. UP
50 UX=2:GOSUB 500 'Sprung ins 2. UP
55 GOSUB 300      'Sprung ins 3. UP
90 END
100 PRINT "Sie befinden sich in UP";UX:RETURN
300 PRINT "zurück aus UP";UX:RETURN
500 PRINT "noch nicht":GOSUB 300:RETURN
```

VERWEISE: Kapitel 4.3.5 und 4.3.7

Anweisungen: **RETURN, ON...GOSUB, CLEAR**

Fehler-Code: 3, 8

ANWEISUNG: **GOTO**

FUNKTION: Sprung zu einer bestimmten Zeile des Programms

FORMAT: **GOTO** Zeilennummer

WIRKUNG: Die Verarbeitung des Programms wird mit der in 'Zeilennummer' bezeichneten Zeile fortgesetzt.

BEMERKUNGEN:
- Sprünge aus Schleifen oder Unterprogrammen hinaus und später wieder zurück sind möglich, sofern dabei nicht neue Schleifen oder Unterprogramme errichtet werden.
- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.

BEISPIELE:

```
10 INPUT "2 Werte, durch , getrennt ";A!,B!
20 PRINT "Das Produkt ist";A!*B!
30 GOTO 10
```

VERWEISE: Kapitel 4.3.5
Anweisung: **ON...GOTO**
Fehler-Code: 8

FUNKTION: HEX$

ZWECK: liefert den hexadezimalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung.

FORMAT: **HEX$(**num.Ausdr.**)**

WIRKUNG: Der numerische Ausdruck wird berechnet, kaufmännisch gerundet und das Ergebnis als Dezimalzahl interpretiert. Davon wird die hexadezimale Darstellung ermittelt und in einen String verwandelt.

BEMERKUNGEN:
- Das Ergebnis von 'num.Ausdruck' muß zwischen Ø und 65536 liegen, sonst wird "Overflow" gemeldet. Negative Zahlen bis -65536 werden ebenfalls umgeformt; es werden allerdings die 16-Bit-Komplemente der positiven Zahl entwickelt.
- Die hexadezimale Zahlendarstellung selbst wird in einen String verwandelt. Dabei wird kein führendes Blank und kein **&H**-Präfix vorangestellt.

BEISPIELE:

```
10 INPUT "Wert ";A%
20 PRINT HEX$(A%)
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.4.3
Funktionen: **OCT$, VAL**

ANWEISUNG: IF ... THEN ... ELSE
(falls...dann...andernfalls)

FUNKTION: Bedingte Verzweigung(en) in einem Programm

FORMAT: IF log.Ausdr. **THEN** { Anweisung(en) / Zeilennr. }
[**ELSE** { Anweisung(en) / Zeilennr. }]

logischer Ausdruck: Vergleichsausdruck, logischer Ausdruck oder Verknüpfung von Vergleichsausdrücken bzw. logischen Ausdrücken durch logische Operatoren; siehe Kapitel 2.7.3.

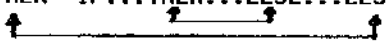
WIRKUNG: Der logische Ausdruck wird berechnet und sein Wahrheitswert gebildet. Ist der logische Ausdruck eine Verknüpfung von Vergleichen, die durch Boole'sche Operatoren wie **AND, OR** etc. verknüpft sind, wird der Wahrheitswert der Verknüpfung gebildet.
Ist das Ergebnis des logischen Ausdrucks "wahr" (also ungleich Ø) wird der **THEN**-Zweig ausgeführt.
Ist das Ergebnis des logischen Ausdrucks "unwahr", (also gleich Ø) wird der **THEN**-Zweig ignoriert und wenn vorhanden der **ELSE**-Zweig ausgeführt. Ist kein zugeordneter **ELSE**-Zweig vorhanden, wird im letzten Falle die nächste Programmzeile verarbeitet. Für beide Fälle gilt: Erfolgt keine Sprunganweisung in dem jeweiligen Zweig, wird die Verarbeitung mit der nächsten Programmzeile fortgesetzt.

BEMERKUNGEN:

- Wird versucht, eine nicht vorhandene Zeilennumer anzuspringen, wird "Undefined line number" gemeldet.
- Vor dem Schlüsselwort **THEN** darf ein Komma stehen (Übersichtlichkeit des Programms).
- Alle im **THEN**- bzw. **ELSE**-Zweig durch : verbundenen Anweisungen werden jeweil verarbeitet, aber es wird der **THEN**-Zweig nur bei "wahr" und der **ELSE**-Zweig nur bei "falsch abgearbeitet.
- **IF**-Anweisungen dürfen verschachtelt werden. Die Tiefe der Verschachtelung ist nur durch die Zeilenlänge begrenzt.
- Liegt eine verschachtelte IF-Folge vor, sind drei Fälle zu unterscheiden:
  . Anzahl **THEN** = Anzahl **ELSE**; jedes einzelne **ELSE** bezieht sich auf das zuletzt davor aufgetretene **THEN**.
  . Anzahl **THEN** größer Anzahl **ELSE**; jedes **ELSE** bezieht sich auf dasjenige vorangehende **THEN**, dem noch kein **ELSE** zugeordnet wurde.

  Beispiel:

  IF...THEN IF...THEN IF...THEN...ELSE...ELSE

  Das erste **THEN** hat kein **ELSE**. Es ist somit nicht möglich, bei Nichterfüllung der ersten Bedingung auf eine Anweisung innerhalb der **IF**-Folge zu kommen.
  . Anzahl **THEN** kleiner **ELSE**; die überflüssigen **ELSE** werden ignoriert.

BEISPIELE:

```
10 INPUT "A,B ";A%,B%
20 IF A%<B% THEN PRINT "A<B":C%=B%-A%
            ELSE IF A%=B% THEN PRINT "A=B":GOTO 10
                          ELSE PRINT "A>B":C%=A%-B%
30 ' wird nur bei A>B bzw. A<B  erreicht !!!
40 PRINT "Unterschied =";C%
```

VERWEISE: Kapitel 4.3.5
Fehler-Code: 8

OPERATOR: IMP
(implication)

FUNKTION: Verknüpfung zweier logischer Ausdrücke mit Hilfe der Implikation-Verknüpfung bzw. Bildung der Subjunktion zweier Integerwerte (Bit-für-Bit-Verknüpfung mit **IMP**)

FORMAT: {log.Ausdruck / Integerwert} **IMP** {log.Ausdruck / Integerwert}

WIRKUNG: Das Ergebnis der **IMP**-Verknüpfung zweier Bedingungen ist

-1 (wahr), wenn die zweite Bedingung den gleichen Wahrheitswert hat wie die erste oder aber die erste falsch ist;

Ø (falsch), wenn die erste Bedingung wahr ist, die zweite aber falsch.

Bei der **IMP**-Verknüpfung zweier Integerwerte wird die **IMP**-Operation Bit für Bit durchgeführt und das Ergebnis berechnet (vgl. Kapitel 2.6.4).

BEMERKUNGEN:
- **IMP** ist die einzige verfügbare logische Operation, bei der es auf die Reihenfolge der Operatoren ankommt.

BEISPIELE:

```
10 A%=5:B%=3:C!=2.47
15 C%=A% IMP B%  '  C% ist jetzt -5
20 IF A%<B% IMP C!>3 THEN GOSUB 1000 ELSE GOSUB 2000
```

VERWEISE: Kapitel 2.6.4
Operatoren: **NOT, AND, OR, XOR, EQV**

FUNKTION: INKEY$

ZWECK: Abfrage des ersten Zeichens im Tastaturpuffer

FORMAT: **INKEY$**

WIRKUNG: Das Ergebnis der Funktion **INKEY$** ist das erste Zeichen im Tastaturpuffer (also ein Ein-Zeichen-String oder der Leerstring). Ist im Moment der Abfrage kein Zeichen im Tastaturpuffer, liefert **INKEY$** den Leerstring. In beiden Fällen wird nach Abfrage der Funktion das Programm fortgesetzt. Das Ergebnis der Funktion wird nicht auf dem Bildschirm dargestellt.

BEMERKUNGEN:
- Das Programm arbeitet auch ohne Betätigung irgendeiner Taste weiter (im Gegensatz zu der Funktion **INPUT$**).
- Es ist jederzeit eine Eingabe möglich, ohne daß das Programm auf den Abschluß der Eingabe durch die Eingabeabschlußtaste wartet (im Gegensatz zu den Anweisungen **INPUT** bzw. **LINE INPUT**).
- In einer Schleife kann ständig geprüft werden, ob eine Taste betätigt wurde und wenn, welche.
- Es können alle Steuerzeichen eingegeben werden.
- Alle Eingabeabschlußtasten (=**CHR$(13)**) gelten als Eingabezeichen.
- C bewirkt eine Unterbrechung des Programmablaufs.
- Auch H (=**CHR$(8)**) gilt als eingegebenes Zeichen.

- Soll der Tastaturpuffer gelöscht werden, ist eine Anweisung der Art

  81 **IF INKEY$= "" THEN** 81

  nötig.
- Der Tastaturpuffer wird durch folgende Anweisungen bzw. Funktionen weiter geleert:

  INKEY$ nächstes Zeichen, falls vorhanden
  INPUT$ nächstes Zeichen, muß vorhanden sein
  INPUT alle Zeichen bis zum nächsten **CHR$**(13)
  LINE INPUT alle Zeichen bis zum nächsten **CHR$**(13)
- Wird ein Tastaturinput über **INKEY$** programmiert, kann dieser nicht durch Inputs aus der V24-Schnittstelle bzw. aus Datenfiles (Wechsel des Ein-/Ausgabemediums, vgl. PCOS-Handbuch ersetzt werden.

BEISPIELE:

```
10 FOR I%=1 TO 10000:NEXT
20 IF INKEY$<>"" THEN 20 'Löschen Tastaturpufferinhalt
30 K$=INKEY$:IF K$="" THEN GOSUB 1001
40 PRINT K$;:GOTO 30 'Verarbeitung des eingegebenen Zeichens
1001 'Verarbeitung, während nichts eingeht
1010 'hier keine Tastatureingaben programmieren!
1099 RETURN
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **INPUT, CURSOR, WINDOW %**
Funktion: **INPUT$**
Assembler-Routine: **lt**

FUNKTION: Das Programm wartet auf Eingaben über die Tastatur und weist die Eingaben auf Variable(n) zu

FORMAT: **INPUT** [;][Stringkonstante][{;/,}] Variablen(liste)

Variablenliste: Folge von numerischen und/oder Stringvariablen, die durch Kommas getrennt sind

WIRKUNG: Die 'Stringkonstante' und ggf. ein Fragezeichen werden angezeigt und das Programm wartet auf die Dateneingabe. Der Bediner kann nun die angeforderten Daten eingeben. Werden mehrere Daten erwartet, müssen diese durch Kommas getrennt eingegeben werden. Die Eingaben werden den Variablen der Variablenliste der Reihe nach zugewiesen. Die Zuweisung erfolgt erst, wenn der Programmablauf nach dem Eintasten aller Werte durch eine zulässige Eingabeabschlußtaste fortgesetzt wird.

BEMERKUNGEN:
- Es ist nicht möglich, bereits am Bildschirm vorhandene Eingaben durch Auslösen einer zulässigen Eingabeabschlußtaste zu übernehmen! Die Eingaben müssen erneut über Tastatur eingegeben werden. Nur die aktuelle Tastatureingabe wird übernommen!!
- Die Eingabewerte müssen mit dem Typ der entsprechenden Variablen übereinstimmen.

- Die Anzahl der eingegebenen Werte muß mit der Anzahl der Variablen in 'Variablenliste' übereinstimmen.
- Es ist nicht möglich, mit **INPUT** Eingabefelder vor der aktuellen Eingabe zu begrenzen.
- Es ist nicht möglich, vor **INPUT**-Anweisungen die Elemente der 'Variablenliste' (seien es String- oder numerische Variablen) auf eine maximale oder minimale Länge zu begrenzen.
- Stringeingaben können ohne Anführungszeichen eingegeben werden. Soll eine Stringeingabe führende oder am Ende stehende Blanks oder Kommas enthalten, muß der String mit Anführungszeichen eingegeben werden.
- Anführungszeichen können über **INPUT** nicht zum Bestandteil einer Stringeingabe werden.
- Der Strichpunkt unmittelbar nach dem Schlüsselwort **INPUT** bewirkt das Unterbleiben des Zeilenvorschubs, der sonst stets nach dem Betätigen jeder zulässigen Eingabeabschlußtaste erfolgt. Dieser Strichpunkt muß auf jeden Fall gesetzt werden, wenn eine Eingabe in der letzten Bildschirm- bzw. Windowzeile erfolgen soll. Dadurch wird das Scrolling (automatischer Vorschub des Bildschirm- bzw. Window-Inhalts um eine Zeile nach oben) verhindert.
- Der Strichpunkt direkt vor der 'Variablenliste' bewirkt, das die Eingabe direkt hinter dem Fragezeichen und nicht in der nächsten Zeile erwartet wird.

- Das Komma direkt vor der 'Variablenliste' bewirkt, daß das Fragezeichen nicht erscheint und die Eingabe hinter der 'Stringkonstanten' erwartet wird.
  ACHTUNG: Im Falle des Kommas muß vorher eine 'Stringkonstante' definiert sein, notfalls der Leerstring.
- Werden bei der **INPUT**-Anweisung zu viele oder zu wenige Daten (d.h. eigentlich, zu wenige oder zu viele Kommas zur Trennung der Eingabewerte) eingegeben oder wird der Versuch gemacht, einer numerischen Variablen eine Stringeingabe durch die Eingabe zuzuordnen, erscheint die Meldung "?Redo from start". Es findet keine Zuweisung statt und alle Eingabedaten dieser **INPUT**-Anweisung müssen neu eingegeben werden.
- Der Inhalt der reservierten Variablen **ERR** und **ERL** wird durch "?Redo from start" nicht verändert; ist eine Fehlerbehandlungsroutine vorhanden, wird nicht in diese verzweigt.
- Die Fehlermeldung "Overflow" kann zwar in einer Fehlerbehandlungsroutine abgefangen werden (Fehlercode 6); die Anzeige nach Rückkehr aus der Routine (**RESUME**-Anweisung) findet dennoch statt, wenn mit **RESUME** oder **RESUME Ø** die **INPUT**-Anweisung wiederholt werden soll.
- Wird nichts eingegeben, bzw. bei Variablenlisten nur die benötigte Anzahl Kommas, so erhalten num.Variablen den Wert Ø und Strings den Wert Leerstring zugewiesen.

- Ein **INPUT** auf die speziellen Variablen **DATE$** und **TIME$** sowie auf die reservierten Variablen **ERR** und **ERL** ist nicht möglich.

BEISPIELE: ▲: Position des Cursors beim Erwarten der Eingabe

1.) 15 **INPUT**;"Eingabe ";E$
RUN
EINGABE ?▲

2.) 25 **INPUT**;"",E$
RUN
▲

3.) 37 **INPUT**;"Ku.-Nr.,PLZ,Ort";N%,P%,O$
RUN
Ku.-Nr.,PLZ,Ort?▲

Eingabe + zulässige Eingabeabschlußtaste	Inhalt von N%	P%	O$ *
4711,6000,Ffm	4711	6000	"Ffm"
,,	0	0	""(Leerstring)
,7,	0	7	""(Leerstring)
3,,5	3	0	"5"
3,	Meldung: ?Redo from Start		
,,"Ffm, Stadt "	0	0	"Ffm, Stadt "

* Die Anführungszeichen dienen hier nur der Kenntlichmachung und sind selbst nicht Bestandteil von O$.

VERWEISE: Kapitel 4.3.8
Anweisungen **LINE INPUT, TAB, POS, CURSOR, WINDOW**
Funktionen: **INPUT$, INKEY$**
Assembler-Routine: **lt**

FUNKTION: Lesen von Daten von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf Variablen

FORMAT: **INPUT#** Filenr., Variable(nliste)

Filenr.: num.Ausdr. (Ergebnis: 1-15)
Variablenliste: Folge von numerischen und/oder Stringvariablen, die durch Komma getrennt sind

WIRKUNG: a) sequentielle Files:
Aus dem unter 'Filenr.' zum Input geöffneten File (Zugriffsart: "I") wird ab der Position des Pointers gelesen.

b) Random-Files:
Aus dem Random-File-Puffer, der dem unter 'Filenr.' geöffneten Random-File (Zugriffsart: "R") per **FIELD**-Anweisung zugeordnet wurde, wird von links gelesen.
Die Daten werden von links kommend elementweise auf die einzelnen Variablen der 'Variablenliste' zugewiesen.
Als Trennzeichen zwischen zwei Daten auf dem File bzw. im Random-File Puffer werden interpretiert:

1) das Komma (,) =**CHR$**(44)
2) LF =**CHR$**(10)
3) CR =**CHR$**(13)

4) nur bei numerischen Daten:
   das Blank =**CHR$**(32)
5) nur bei Strings:
   das Anführungszeichen ("): =**CHR$**(34),
   aber nur wenn der String-
   Anfang durch ein Anführungs-
   zeichen markiert war.

- numerische Werte:
  Das erste Zeichen seit dem letzten Trennzeichen, das nicht ein Blank, LF oder CR ist, wird als Beginn des Wertes interpretiert.
  Auf Ende des Wertes wird erkannt, sobald eines der Zeichen zu 1) bis 4) auftritt.
- Strings:
  Das erste Zeichen seit dem Trennzeichen, das nicht ein Blank, LF oder CR ist, wird als Beginn des Werts interpretiert.
  Auf das Ende des Werts wird erkannt, sobald eines der Zeichen zu 1) bis 3) auftritt oder auf die betroffene Stringvariable bereits 255 Zeichen zugewiesen worden sind.
  Ist das erste für einen String zulässige Zeichen nach dem Trennzeichen jedoch das Anführungszeichen, wird auf das dem Anführungszeichen folgende Zeichen als erstes Zeichen des Strings erkannt (auch wenn dieses Blank, LF oder CR ist). Auf Ende des Strings wird dann erst beim nächsten Anführungszeichen erkannt. Zwischen " " stehende Trennzeichen werden in diesem Falle also nicht als solche interpretiert.

BEMERKUNGEN:

- Die eingelesenen Daten müssen für die Zuweisung formatgerecht sein.
- Bei **INPUT#** wird anders als bei **INPUT** kein ? auf dem Bildschirm ausgegeben.
- Wird während der Ausführung der **INPUT#**-Anweisung das File- bzw. Puffer-Ende erreicht, wird keine Fehlermeldung gegeben und das Ende des gerade betroffenen Datums sowie des Lesens angenommen.
- Das Anführungszeichen (") kann mit **INPUT#** nicht zum Bestandteil eines Strings gemacht werden.
- Führende Blanks oder Kommas können zum Bestandteil eines Strings gemacht werden, wenn der Wert des Strings in Anführungszeichen eingeschlossen auf das File gebracht wurde (vgl. **WRITE#** und **PRINT#**).
- Ein **INPUT#** auf die speziellen Variablen **DATE$** und **TIME$** und die reservierten Variablen **ERR** und **ERL** ist nicht möglich.

BEISPIELE:

```
10 OPEN "O",1,"1:DATEN.seq"
20 A%=1:B!=2.7:C$="Streichhölzer"
30 PRINT#1,A%,B!,C$
35 ERG!=A%*B!:PRINT#1,ERG!
40 CLOSE 1:CLS
50 OPEN "I",1,"1:DATEN.seq"
60 LINE INPUT#1,D$ ' alles von Zeile 30
70 PRINT D$
75 INPUT#1,PROD!:PRINT "Produkt =";PROD!
80 CLOSE 1:END
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, LINE INPUT#, PRINT#, PRINT# USING, WRITE#**

Fehler-Code: 54

FUNKTION: INPUT$

ZWECK: Einlesen einer bestimmten Zeichenanzahl aus dem Tastaturpuffer oder von einem Datenfile

FORMAT: **INPUT$**(Anzahl Zeichen [,[#]Filenr.])

Anzahl Zeichen: num.Ausdruck;
bestimmt die Anzahl Zeichen, aus der das Ergebnis der Funktion besteht

Filenr.: num.Ausdruck;
es ist die Nummer anzugeben, unter der das Datenfile bei der Anweisung **OPEN** geöffnet wurde

WIRKUNG: Die 'Filenr.' wird berechnet und der ganzzahlige Anteil bestimmt. 'Anzahl Zeichen' wird berechnet und gerundet. Ist der Parameter '#Filenr.' angegeben, wird aus dem durch 'Filenr.' definierten Datenfile (siehe Anweisungen **OPEN**) die durch 'Anzahl Zeichen' bestimmte Anzahl Zeichen eingelesen. Fehlt der Parameter '#Filenr.', ist das Ergebnis von **INPUT$** eine Folge der im Tastaturpuffer vorhandenen Zeichen. Es werden immer nur soviel Zeichen betrachtet, wie mit 'Anzahl Zeichen' festgelegt. Das Ergebnis der Funktion wird nicht auf dem Bildschirm dargestellt.

BEMERKUNGEN:
- 'Anzahl Zeichen' muß einen Wert zwischen 1 und 255 aufweisen.

- Beim Aufruf von **INPUT$** wartet das Programm, bis die durch 'Anzahl Zeichen' bestimmte Anzahl Zeichen aus dem Tastaturpuffer eingelesen werden können. Im Gegensatz zu der Funktion **INKEY$** müssen erst soviel Zeichen im Tastaturpuffer vorhanden sein (eingegeben werden), wie durch 'Anzahl Zeichen' festgelegt'.
- Der Abschluß der Eingabe muß nicht mit Eingabeabschlußtaste erfolgen (im Gegensatz zu der Anweisung **INPUT**), sondern wird erkannt, sobald die in 'Anzahl Zeichen' spezifizierte Anzahl Tasten bestätigt wurden.
- Alle Eingabeabschlußtasten (=**CHR$(**13**)**) gelten als Eingabezeichen.
- Es können alle Steuerzeichen eingegeben werden.
- **C** bewirkt eine Unterbrechung des Programmablaufs.
- **H** löscht bereits eingegebene Zeichen. **H** (=**CHR$(**8**)**) gilt als Betätigen einer Taste und wird mitgezählt.

BEISPIELE:

```
10 WHILE K$<>CHR$(13)
20 K$=INPUT$(1):PRINT K$;
30 WEND:GOTO 10

10 LINE INPUT "Filename ";F$
20 OPEN "I",1,F$
30 WHILE NOT EOF(1)
40 PRINT INPUT$(1,#1);
50 WEND CLOSE 1:END
```

VERWEISE: Kapitel 4.3.8 und 4.3.9.1

Anweisungen: **OPEN, INPUT, INPUT#**

Funktion: **INKEY$**

Fehler-Code: 54

Assembler-Routine: **lt**

FUNKTION: INSTR
(instruct position)

ZWECK: Aufsuchen der Anfangsposition eines zu suchenden Strings in einem String

FORMAT: **INSTR(** [Position,] Ausgangsstring, Teilstring)

Position: num.Ausdruck;
Position, ab der die Suche nach 'Teilstring' beginnen soll

Ausgangsstring: Stringausdruck, in dem gesucht werden soll

Teilstring: Stringausdruck, der gesucht wird

WIRKUNG: Die 'Position' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet. Als Ergebnis wird die Stelle des 'Ausgangsstrings' geliefert, an der der 'Teilstring' im 'Ausgangsstring' beginnt. Ist 'Position' nicht angegeben, beginnt die Suche beim ersten Zeichen des Ausgangsstrings. Ist der 'Teilstring' nicht genau im 'Ausgangsstring' vorhanden, liefert die Funktion Ø.

BEMERKUNGEN:
- 'Position' muß einen Wert zwischen 1 und 255 aufweisen.
- Die Funktion **INSTR** liefert als Ergebnis den Wert Ø, falls:
  . 'Position' größer als die Länge des 'Ausgangsstrings' ist.
  . der 'Ausgangsstring' der Leerstring ist.

. der 'Teilstring' im 'Ausgangsstring' nicht genau enthalten ist.

- Ist der 'Teilstring' der Leerstring, so ist das Ergebnis der Funktion **INSTR** gleich 'Position', falls diese angegeben wurde, anderenfalls gleich 1.

BEISPIELE:

```
10 LINE INPUT "zu durchsuchender Text ";A$
15 PRINT "Suche in ";A$;" nach ";
16 LINE INPUT "",S$
17 P%=INSTR(A$,S$)
19 IF P%=0 THEN PRINT S$;" in ";A$;" nicht enthalten."
20    ELSE PRINT S$," beginnt beim";P%;". Zeichen von ";A$
30 END
```

VERWEISE: Kapitel 4.3.4

FUNKTION: **INT**
(integer)

ZWECK: Ermittlung derjenigen ganzen Zahl aus einem numerischen Wert, die als nächste auf der Zahlenskala links von dem angegebenen num.Wert liegt ("Gauß-Funktion")

FORMAT: **INT**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet. Es wird diejenige Zahl bestimmt, die als nächste links auf der Zahlenskala liegt.

BEMERKUNGEN:
- Das Ergebnis ist in der Genauigkeit des Arguments.
- Ist das Argument selbst ganzzahlig, ergibt es sich als Ergebnis.

BEISPIELE: **INT**(3.7) liefert 3 und **INT**(-3.7) liefert -4.
INT(3.2) liefert 3 und **INT**(-3.2) liefert -4.

```
10 A!=2.4:B!=-2.4
20 PRINT INT(A!):INT(B!)
RUN
 2 -3
```

VERWEISE: Kapitel 2.4 und 4.13
Funktionen: **FIX, CINT**

BEFEHL: KILL

FUNKTION: Löschen des Namens eines auf Diskette gespeicherten Files aus dem Inhaltsverzeichnis und Freigabe der vom File auf Diskette belegten Sektoren für andere Files

FORMAT: **KILL** Filename

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Es wird geprüft, ob das mit 'Filename' bestimmte File vorhanden ist. Wenn ja, wird dieses File aus dem Inhaltsverzeichnis gelöscht. Der dadurch frei gewordene Platz kann für die Speicherung weiterer Files wieder verwendet werden.

BEMERKUNGEN:
- Der PCOS-Befehl **fk** hat die gleiche Wirkung wie der BASIC-Befehl **KILL**.
- Mit dem PCOS-Befehl **rk** kann ein gelöschtes File reaktiviert werden, sofern an seine Stelle nicht bereits ein neues File getreten ist.
- Die Diskette oder das File 'Filename' darf keinen Schreibschutz haben.
- File-Password und/oder Disketten-Password, wenn vorhanden, müssen angegeben werden.
- Nachdem das File 'Filename' gelöscht wurde, kann 'Filename' zur Speicherung eines neuen Files wiederverwendet werden.

- **KILL** löscht nur das betreffende File aus dem Inhaltsverzeichnis der Diskette und vermerkt dort, daß die betroffenen Sektoren wieder frei sind. Die Sektoren des Files selbst hingegen werden nicht mit **CHR$(Ø)** vorbelegt!

BEISPIELE:

```
KILL "D:PROG1.prg"
KILL "DISKC:T20500.ase"
```

VERWEISE: Kapitel 4.1.3

FUNKTION: LEFT$

ZWECK: Einem String wird von links kommend ein Teilstring entnommen

FORMAT: **LEFT$(Ausgangsstring, Anzahl Zeichen)**

Ausgangsstring: Stringausdruck
Anzahl Zeichen: num.Ausdruck;
bestimmt die Anzahl Zeichen des Teilstrings

WIRKUNG: Die 'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet. Aus dem 'Ausgangsstring' wird von links kommend die berechnete Anzahl von Zeichen entnommen.

BEMERKUNGEN:
- Ist die 'Anzahl Zeichen' größer als die Länge des 'Ausgangsstrings', wird der gesamte Ausgangsstring übergeben.
- Ist die 'Anzahl Zeichen' = Ø, wird kein Zeichen entnommen und der 'Teilstring' ist der Leerstring.
- 'Anzahl Zeichen' muß einen Wert zwischen Ø und 255 aufweisen.

BEISPIELE:

```
10 A$="Personal-Computer M 20"
20 LPRINT LEFT$(A$,8);"***"
RUN
Personal***
```

VERWEISE: Kapitel 4.3.4
Funktion: **RIGHT$, MID$**

FUNKTION: **LEN**

ZWECK: Bestimmung der Länge eines Stringausdrucks

FORMAT: **LEN**(Stringausdruck)

WIRKUNG: Die Funktion **LEN** ermittelt die im Stringausdruck enthaltene Anzahl von Zeichen.

BEMERKUNGEN: - Alle nicht druckbaren Zeichen und Blanks werden mitgezählt.

BEISPIELE:

```
5 T$=" HALLO "+CHR$(7)  'akustisches Signal
10 PRINT T$;" besteht aus";LEN(T$);"Zeichen!"
RUN
 HALLO  besteht aus 8 Zeichen!
```

VERWEISE: Kapitel 4.3.4

ANWEISUNG: LET...=

FUNKTION: Zuweisung des Werts eines Ausdrucks an eine Variable

FORMAT:

```
[LET] { numerische Variable   = numerischer Ausdruck }
      { Stringvariable        = Stringausdruck       }
```

WIRKUNG: Der Ausdruck wird berechnet und das Ergebnis der Variablen zugewiesen.

BEMERKUNGEN:

- Das Schlüsselwort **LET** muß nicht eingegeben werden.
- Die aktuelle Länge der Stringvariablen in einer Zuweisung ist gleich der Anzahl Zeichen, die sich als Ergebnis des Stringausdrucks ergeben.
- Die Zuweisung erfolgt in dem Datentyp, in dem die 'numerische Variable' definiert ist.

  ACHTUNG:

  Bei der Zuweisung eines einfach genauen num. Werts auf eine doppelt genaue Variable kann Signifikanzverlust eintreten; d.h., der Inhalt der doppelt genauen Variablen ist nur annähernd gleich dem einfach genauen Wert (vgl. Kapitel 2.5.1.4, Punkt 5 sowie 2.7.1)! Diese Problematik kann mit einer Anweisung der Art D#=**VAL(STR$(G!))** umgangen werden.
- Mehrfachzuweisungen sind nicht möglich. Sie müssen durch mehrere **LET**-Anweisungen bewirkt werden.

BEISPIELE:

```
60 PRINT B%  'automatischer Anfangswert = 0
70 A%=10:LET B%=A%  'kürzer:  B%=A%
80 PRINT A%;B%
RUN
 0
 10  10
```

VERWEISE: Kapitel 4.3.3

ANWEISUNG: **LINE**

FUNKTION: Zeichnen von Linien oder Rechtecken oder mit einer Farbe gefüllten Rechtecken

FORMAT: **LINE** [% Window-Nr.,] [**STEP**] [(X1,Y1)]
[**STEP**]-(X2,Y2) [,Farbindex[, **B**[**F**]][, Operand]]

Window-Nr.: numerischer Ausdruck; Wert 1 bis 16

X1,Y1: numerische Ausdrücke, deren Wert den Anfangspunkt bestimmen

X2,Y2: numerische Ausdrücke, deren Wert den Endpunkt bestimmen

Farbindex: numerischer Ausdruck; bestimmt die Farbe, beim Vier-Farb-Schirm bezogen auf die aktuelle globale **COLOR**=-Anweisung (Wert: Ø-3), beim Schwarz/Weiß- und Acht-Farb-Schirm identisch mit 'Farbindex'.

Operand: Operand kann sein: **AND, OR, XOR, NOT, PSET, PRESET**

WIRKUNG: Es wird eine Strecke oder ein Rechteck oder ein ausgefülltes Rechteck gezeichnet. Ist der Parameter **%** Window-Nr. angegeben, erfolgt die Ausgabe im angegebenen Window.
Fehlt dieser Parameter, erfolgt die Ausgabe im aktiven Window.

1. Zeichnen von Strecken in Absolutkoordinaten
   Eine Strecke in Absolutkoordinaten wird gezeichnet, wenn die Anweisungselemente **STEP** und **BF** nicht angegeben sind.

X1 und Y1 bestimmen den Anfangspunkt der Strecke, bezogen auf das für das Window gültige **SCALE**. Fehlt dieser Parameter, wird die Strecke vom vorangehenden Endpunkt zu den angegebenen Koordinaten gezogen.

2. Zeichnen von Strecken in Relativkoordinaten
   Wird vor einem Koordinatenpaar das Sprachelement **STEP** angegeben, wird dieses Koordinatenpaar relativ zur zuletzt erfolgten Angabe errechnet und entsprechend die Verbindung gezeichnet.

3. Zeichnen von Rechtecken
   Wird in der Anweisung das Sprachelement **B** ("box") angegeben, bilden die Koordinatenpaare **(X1,Y1)** und **(Y1,Y2)** diagonal liegende Eckpunkte eines parallel zu den Achsen liegenden Rechtecks, wobei **(X1,Y1)** die linke untere Ecke bestimmt. Fehlen in der Anweisung die Schlüsselwörter **STEP**, wird das Rechteck nach Absolutkoordinaten, bezogen auf das gültige **SCALE**, gezeichnet. Ist vor einem Koordinatenpaar **STEP** angegeben, wird dieses Koordinatenpaar relativ zu den vorangegangenen Koordinaten interpretiert.

4. Zeichnen von ausgefüllten Rechtecken
   Wird der Parameter **BF** ("box fill") anstelle von **B** angegeben, wird das Rechteck mit der mit 'Farbindex' angegebenen Farbe ausgefüllt.

Farbindex

Dieser Parameter gibt an, in welcher Farbe die Strecke, das Rechteck oder das ausgefüllte Rechteck gezeichnet werden soll. Dabei gibt der Parameter 'Farbindex' für den Vier-Farb-Schirm die Position (Ø-3) der entsprechenden Farbe in der globalen **COLOR**=-Anweisung an. Für den Schwarz/Weiß- und den Acht-Farb-Schirm ist 'Farbindex' identisch mit 'Farbcode'. Fehlt dieser Parameter, wird in der aktuellen Vordergrundfarbe gezeichnet.

Operand

'Operand kann eines der Sprachelemente **AND, OR, XOR, NOT, PSET** oder **PRESET** sein.
Es wird auf dem für die Darstellung maßgebenden Bereich der Bitmap (vgl. Kapitel 4.3.18) die entsprechende logische Operation für jeden Elementarpunkt durchgeführt. Wird **PSET** angegeben, werden alle betroffenen Punkte gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Farbe des Hintergrundes gesetzt.

BEMERKUNGEN:

- Strecken werden so dargestellt, daß jeweils die Punkte in der angegebenen Farbe dargestellt werden, die dem bei der Interpolation errechneten Wert am nächsten liegen.
- Wird die Strecke bezogen auf den vorangehenden Endpunkt, kann der dann für die Strecke maßgebende Anfangspunkt z.B. durch eine zuvor ausgeführte **LINE**-, **DRAW**-, **PSET**- oder **PRESET**-Anweisung definiert werden.

- ACHTUNG:

  Wird eine Grenze der Skalierung - sei es aufgrund von Default-Werten oder aber einer **SCALE**-Anweisung - unter- bzw. überschritten, wird aus Anfangs- und Endpunkt ein Verhältnis entwickelt, das den Verlauf der Strecke bzw. die Form des Rechtecks bestimmt. Dieses Verhältnis ist von X- und Y-Koordinaten abhängig. Es empfiehlt sich deshalb, mit einer **SCALE**-Anweisung statt mit Default-Werten zu arbeiten, sowie die linke und rechte **SCALE**-Grenze nicht zu unter- bzw. überschreiten.

BEISPIELE:

```
0 CLEAR:SCALE 0,511,0,255
30 LINE(0,50)-(320,50)
40 LINE -(420,50)
50 LINE(100,50)-(180,160),,B
60 LINE(120,50)-(200,160),,BF
70 PSET(511,255):LINE STEP(-511,-255)-STEP(511,255),,B
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **COLOR=, COLOR, SCALE**
Funktion: **WINDOW**

ANWEISUNG: LINE INPUT

FUNKTION: Eingabe einer ganzen Kette von Zeichen ohne Trennzeichen über Tastatur und Zuweisung auf eine Stringvariable

FORMAT: **LINE INPUT** [;][Stringkonstante]{;|,} Stringvariable

WIRKUNG: Die 'Stringkonstante' und ggf. ein Fragezeichen werden angezeigt und das Programm wartet auf die Dateneingabe. Die Eingabe, die durch Eingabeabschlußtaste zu beenden ist, wird auf die Stringvariable zugewiesen.

BEMERKUNGEN:
- Es ist nicht möglich, bereits am Bildschirm vorhandene Eingaben durch Auslösen einer zulässigen Eingabeabschlußtaste zu übernehmen! Die Eingaben müssen erneut über Tastatur eingegeben werden. Nur die aktuelle Tastatureingabe wird übernommen!!
- Es ist nicht möglich, bei **LINE INPUT** Eingabefelder vor der aktuellen Eingabe zu begrenzen.
- Es ist nicht möglich, vor **LINE INPUT**-Anweisung die Länge des auf 'Stringvariable' zuzuweisenden Strings auf eine maximale oder minimale Anzahl zu begrenzen.
- Die Anzahl der eingegebenen Zeichen darf 254 nicht überschreiten.

- Die Eingabe kann ohne Anführungszeichen eingegeben werden. Führende oder am Ende stehende Blanks oder in der Eingabe enthaltene Kommas werden übernommen.
- Eingegebene Anführungszeichen werden über **LINE INPUT** zum Bestandteil der Stringeingabe gemacht.
- Der Strichpunkt unmittelbar nach dem Schlüsselwort **LINE INPUT** bewirkt das Unterbleiben des Zeilenvorschubs, der sonst stets nach dem Betätigen jeder zulässigen Eingabeabschlußtaste erfolgt. Dieser Strichpunkt muß auf jeden Fall gesetzt werden, falls die Eingabe in der letzten Bildschirm- bzw. Window-Zeile erfolgt. Nur so kann das Scrolling (Vorschub des Bildschirm- bzw. Window-Inhalts um eine Zeile nach oben) verhindert werden.
- Der Strichpunkt direkt vor der 'Stringvariablen' bewirkt, daß die Eingabe direkt hinter dem Fragezeichen und nicht in der nächsten Zeile erwartet wird.
- Das Komma direkt vor der 'Stringvariablen' bewirkt, daß das Fragezeichen nicht erscheint und die Eingabe hinter der 'Stringkonstanten' erwartet wird.

  ACHTUNG:

  Im Falle des Kommas muß vorher eine 'Stringkonstante' definiert sein, notfalls der Leerstring.
- Wird nichts eingegeben, erhält die 'Stringvariable' den Wert Leerstring zugewiesen.
- Die Eingabeabschlußtaste wird nicht zum Bestandteil der Eingabe.

- Ein **LINE INPUT** auf die Variablen **DATE$** und **TIME$** ist nicht möglich.

BEISPIELE:

```
10 CURSOR(10,16):PRINT "Ihre Eingabe: "
20 LINE INPUT ;"",E$
30 CURSOR(1,1):PRINT E$:STOP
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **INPUT, LINE INPUT#, CURSOR, WINDOW%**
Funktionen: **TAB, POS, INPUT$, INKEY$**
Assembler-Routine: **lt**

ANWEISUNG: LINE INPUT#

FUNKTION: Lesen einer Folge von Zeichen von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf eine Stringvariable

FORMAT: **LINE INPUT#** Filenr.,Stringvariable

Filenr.: num.Ausdr. (Ergebnis: 1-15)

WIRKUNG: Die 'Filenr.' wird berechnet und auf Ganzzahligkeit gerundet.

a) sequentielle Files:
   Aus dem unter 'Filenr.' zum Input geöffneten File (Zugriffsart: "I") wird ab der Position des Pointers gelesen.

b) Random-Files:
   Aus dem Random-File-Puffer, der dem unter 'Filenr.' geöffneten Random-File (Zugriffsart: "R") per **FIELD**-Anweisung zugeordnet wurde, wird von links gelesen.

Es werden solange Zeichen eingelesen, bis ein CR (=**CHR$(**13**)**) oder LF (=**CHR$(**10**)**) erkannt wird oder 255 Zeichen eingelesen werden. Die betreffende Zeichenfolge wird auf die 'Stringvariable' zugewiesen (ohne CR bzw. LF).

BEMERKUNGEN:
- Bei **LINE INPUT#** wird anders als bei **LINE INPUT** kein ? auf dem Bildschirm ausgegeben.

- Wird während der Ausführung der **LINE INPUT#**-Anweisung das File- bzw. Puffer-Ende erreicht, wird keine Fehlermeldung gegeben und auf Ende des einzulesenden Strings erkannt.
- Das Anführungszeichen (") kann mit **LINE INPUT#** zum Bestandteil des Strings gemacht werden.
- Führende Blanks oder eingeschlossene Kommas werden zum Bestandteil des Strings gemacht.
- Ein **LINE INPUT#** auf die speziellen Variablen **DATE$** und **TIME$** ist nicht möglich.

BEISPIELE:

```
0 ' entfernen       Texte hinter REM oder ' in File F$
10 OPEN "I",1,F$      'gespeichert mit SAVE ...,A
20 OPEN "O",2,"1:NPROG.asc" 'verkürzte Fassung
50 WHILE NOT EOF(1)
60 LINE INPUT#1,ZEILE$
80 P%=INSTR(ZEILE$," REM "):P1%=INSTR(ZEILE$,"'")
90 IF P%=0 AND P1%=0 THEN 110
100 IF P% THEN ZEILE$=LEFT$(ZEILE$,P%-1)
105 IF P1% THEN ZEILE$=LEFT$(ZEILE$,P1%-1)
110 PRINT#2,ZEILE$
120 WEND:CLOSE 1,2:END
```

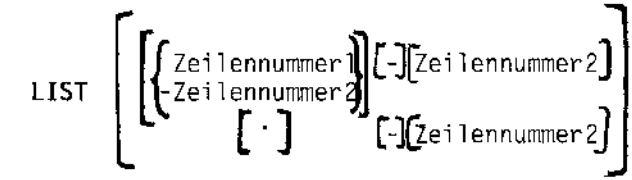
VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, INPUT#, PRINT#, PRINT# USING, WRITE#**

Fehler-Code: 54

FUNKTION: Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programms am Bildschirm.

FORMAT:



Zeilennumer1: ganze positive Zahl; gibt die Nummer der ersten auszugebenden Programmzeile an

Zeilennumer2: ganze positive Zahl; gibt die Nummer der letzten auszugebenden Programmzeile an

WIRKUNG: Sind keine Operanden angegeben, wird das im Arbeitsspeicher befindliche Programm am Bildschirm angezeigt. Die Ausgabe erfolgt beginnend mit der kleinsten Zeilennummer in aufsteigender Reihenfolge bis zur letzten Zeilennummer.
Ist nur der Operand 'Zeilennummer1' angegeben, erfolgt nur Ausgabe der Programmzeile, die durch 'Zeilennummer1' bestimmt ist. Der Operand '.' bestimmt, daß die zuletzt angesprochene Programmzeile ausgegeben werden soll. Der Operand '-' gibt an, daß ein Abschnitt des Programmes angezeigt werden soll. Der Abschnitt beginnt mit der ersten Programmzeile oder der im ersten Operanden angegebenen Programmzeile.

Ist der Operand 'Zeilennummer2' angegeben, erfolgt die Ausgabe bis zu der Programmzeile, die durch 'Zeilennummer2' bestimmt ist. Anderenfalls erfolgt die Ausgabe bis zur letzten Programmzeile. Nach **LIST** erfolgt der Übergang in den Direkt-Mode.

BEMERKUNGEN:

- **LIST** - hat die gleiche Funktion wie **LIST**.
- Die Ausführung des Befehls **LIST** bewirkt eine Umwandlung von Klein- in Großbuchstaben bei Schlüsselwörtern, Variablen- und Funktionsnamen. Außerdem wird das Zeichen **?** am Beginn einer Anweisung in die Anweisung **PRINT** umgewandelt.
- Geschützte Programme (Abspeicherung durch den Befehl **SAVE** mit Operand **P**) können nicht angezeigt werden.
- **C** bewirkt eine Unterbrechung der Ausgabe. Das System befindet sich anschließend im Direkt-Mode.
- Die Ausgabe erfolgt nur im aktiven Window.
- Bei der Ausgabe erfolgt ein Scrolling (automatisches Hochschieben der Zeichen) im aktiven Window, wenn der untere Window-Rand erreicht ist.
- **S** bewirkt eine Unterbrechung der Ausgabe (und damit des Scrollings), ohne daß das System in den Direkt-Mode übergeht. Die Ausgabe kann durch Drücken einer beliebigen Taste fortgesetzt werden.
- Mit der Anweisung **WIDTH** kann festgelegt werden, wieviel Zeichen pro Zeile ausgegeben werden sollen.

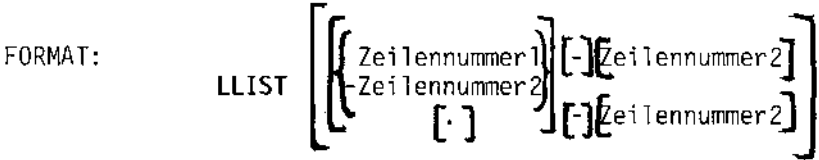
BEISPIELE:

```
LIST .
LIST -20500
```

VERWEISE: Kapitel 3
Befehl: **SAVE**
Anweisung: **WIDTH**
Funktion: **WINDOW**

FUNKTION: Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programms auf dem Drucker.

FORMAT:



Zeilennumer1: ganze positive Zahl; gibt die Nummer der ersten auszudruckenden Programmzeile an

Zeilennumer2: ganze positive Zahl; gibt die Nummer der letzten auszudruckenden Programmzeile an

WIRKUNG: Sind keine Operanden angegeben, wird das gesamte im Arbeitsspeicher befindliche Programm am Drucker ausgedruckt. Die Ausgabe erfolgt beginnend mit der kleinsten Zeilennumer in aufsteigender Reihenfolge bis zur letzten Zeilennummer.
Ist nur der Operand 'Zeilennummer1' angegeben, erfolgt nur Ausgabe der Programmzeile, die durch 'Zeilennummer1' bestimmt ist. Der Operand '.' bestimmt, daß die zuletzt angesprochene Programmzeile ausgegeben werden soll. Der Operand '-' gibt an, daß ein Abschnitt des Programms ausgedruckt werden soll. Der Abschnitt beginnt mit der ersten Programmzeile oder der im ersten Operanden angegebenen Programmzeile.

Ist der Operand 'Zeilennummer2' angegeben, erfolgt die Ausgabe bis zu der Programmzeile, die durch 'Zeilennummer2' bestimmt ist. Andernfalls erfolgt die Ausgabe bis zur letzten Programmzeile. Nach **LLIST** erfolgt der Übergang in den Direkt-Mode.

BEMERKUNGEN:

- **LLIST** - hat die gleiche Funktion wie **LLIST**.
- Die Ausführung des Befehls **LLIST** bewirkt eine Umwandlung von Klein- in Großbuchstaben bei Schlüsselwörtern, Variablen- und Funktionsnamen. Außerdem wird das Zeichen **?** am Beginn einer Anweisung in **PRINT** umgewandelt.
- Geschützte Programme (Abspeicherung durch den Befehl **SAVE** mit Operand **P**) können nicht ausgedruckt werden.
- **C** bewirkt eine Unterbrechung der Ausgabe. Das System befindet sich anschließend im Direkt-Mode.
- **S** bewirkt eine Unterbrechung der Ausgabe, ohne daß das System in den Direkt-Mode übergeht. Die Ausgabe kann durch Drücken einer beliebigen Taste fortgesetzt werden.
- Die Art des Drucks (Fettdruck, 10-Zolldruck, usw.) kann über vorherige **LPRINT**-Anweisungen mit Steuerzeichen vorgeschrieben werden (vgl. Kapitel 4.3.9.2 und 5.1).
- Mit dem PCOS-Befehl **sf** werden für den am System angeschlossenen Drucker Default-Werte gesetzt.
- Mit der Anweisung **WIDTH LPRINT** kann festgelegt werden, wieviel Zeichen pro Zeile ausgedruckt werden sollen.

BEISPIELE:

```
LLIST .
LLIST -20500
```

VERWEISE: Kapitel 4.3.9.2, 5.1 und Anhang II
Befehl: **SAVE**
Anweisung: **WIDTH**

BEFEHL: LOAD

FUNKTION: Laden eines Programms von Diskette in den Arbeitsspeicher

FORMAT: **LOAD** Filename [,R]

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Das mit 'Filename' bestimmte Programm bzw. das mit 'Filename' bestimmte ASCII-File wird von der Diskette in den Arbeitsspeicher geladen. Der bisherige Inhalt des Arbeitsspeichers wird dadurch gelöscht (Programmzeilen und Variablen). Enthält der Operand 'Filename' keinen Stationsnamen, wird das Programm auf der momentan aktiven (zuletzt angesprochenen) Station gesucht. Die Ausführung des Befehls **LOAD** ohne Operand **R** schließt alle offenen Datenfiles und bringt das Programm nicht zum Laufen.
Ist der Operand **R** angegeben, bleiben nach dem Laden des Programms alle bereits geöffneten Dateien offen und das Programm wird unmittelbar zum Laufen gebracht.

BEMERKUNGEN:
- Disketten- und File-Password, falls vorhanden, müssen angegeben werden.

Die Angabe des Disketten-Passwords kann entfallen, wenn es einmal angegeben wurde und seitdem die Diskette nicht gewechselt und das Betriebssystem nicht neu geladen wurde.

- Ist das Programm bzw. der Text in Form eines ASCII-Files auf Diskette gespeichert, bewirkt **LOAD** die Sortierung in Reihenfolge der Zeilennummern (z.B. bei über **ed %c** mit dem Full-Screen-Editor erstellten ASCII-Files). Danach ist das Programm lauffähig, sofern es aus syntaktisch richtigen Anweisungen besteht.
- Enthält das File 'Filename' eine Zeile ohne Zeilennummer, wird 'Direct statement in file' gemeldet.
- Nach **LOAD** ohne Parameter **R** erfolgt ein Übergang in den Direkt-Mode.
- Nach **LOAD...,R** bleiben zwar die Files des Vorprogramms offen, es gehen aber alle Werte und Vereinbarungen von Variablen - incl. von **FIELD**-Vereinbarungen - verloren.

BEISPIELE:

```
LOAD "1:TEST1.prg"
LOAD "PROGS:BRUTTORNG.prg"
```

VERWEISE: Kapitel 3.4

FUNKTION: LOC

ZWECK: gibt die Anzahl im Programm gelesener/beschriebener Sektoren eines sequentiellen Files oder die zuletzt im Programm angesprochene Record-Nr. eines Random-Files an

FORMAT: **LOC**(Filenr.)

Filenr.: num.Ausdruck (Ergebnis 1 - 15)

WIRKUNG: Die 'Filenr.' wird berechnet. Das Ergebnis der Funktion gibt an,

a) bei sequentiellen Files:
   wieviele 256-Byte-Sektoren von dem File bereits gelesen bzw. beschrieben wurden, seit das File unter 'Filenr.' eröffnet wurde;
b) bei Random-Files:
   welche Record-Nummer bei der letzten **GET**- bzw. **PUT**-Anweisung für das unter 'Filenr.' geöffnete Random-File verarbeitet wurde.

BEMERKUNGEN:
- Nach dem Öffnen eines Files liefert die Funktion in beiden Fällen den Wert Ø.

BEISPIELE:

```
10 OPEN "O",1,"1:ZAHLEN.seq"
20 FOR I%=1 TO 150
30 PRINT#1,USING "## ###### ";RND 'Zufallszahlen
40 NEXT
50 PRINT LOC(1);"256-Byte-Sektoren beschrieben"
60 CLOSE 1:END
```

VERWEISE: Kapitel 4.3.10.2
Anweisung: **OPEN**
Funktion: **LOF**

FUNKTION: LOF

ZWECK: liefert für sequentielle Files die Nummer des höchsten beschriebenen Bytes im File; bei Random-Files kann damit ermittelt werden, in welchem Sektor des Files der höchste beschriebene Record steht

FORMAT: **LOF**(Filenr.)

Filenr.: num.Ausruck (Ergebnis: 1 - 15)

WIRKUNG: Die 'Filenr.' wird berechnet und kaufmännisch auf Ganzzahligkeit gerundet.
Von dem unter dieser 'Filenr.' geöffneten File (alle Zugriffsarten möglich) wird ermittelt:

a) bei sequentiellem File (Zugriffsarten: "I", "O", oder "A"): die Position des Pointers (Byte-Nr.) nach der bisher jemals letzten Schreib-Operation mit dem File (wie **PRINT#, WRITE#**).
b) bei Random-File (Zugriffsart: "R"): ein Vielfaches von 256. Dieses berechnet sich aus der Nummer desjenigen 256-Byte-Sektors im File, in den jemals zuvor mit der bisher höchsten Record-Nr. geschrieben wurde (**PUT**-Anweisung), multipliziert mit 256.

BEMERKUNGEN: - Ist in ein File noch nicht geschrieben worden, liefert **LOF** nach **OPEN** den Wert Ø.

- Die Funktion kann auch negative Werte liefern. Der Wert wird im Integerformat intern dargestellt (vgl. dazu Kapitel 2.6.3, interne Abspeicherung von Integer-Daten); Minimum ist -32768, Maximum 32767 (bei Random-Files 32512).
- Liefert **LOF** Ø, muß dies nicht bedeuten, daß das File leer ist.
- Am Beispiel eines Random-Files sei erläutert, wie die Berechnung vorzunehmen ist:

LOF-Wert	LOF Ö256	höchster beschriebener Sektor
Ø	Ø	kein Sektor oder 256 oder 512 etc. (!)
256	1	1 oder 257 oder 513 etc.
512	2	2 oder 258 oder 514 etc.
⋮	⋮	⋮
32512	127	127 oder 383 oder 639 etc.
-32768	-128	128 oder 384 oder 640 etc.
-32512	-127	129 oder 385 oder 641 etc.
⋮	⋮	⋮
-256	-1	255 oder 511 oder 767 etc.
Ø	Ø	kein Sektor oder 256 oder 512 etc.
256	1	1 oder 257 oder 513 etc.
⋮	⋮	⋮
etc. wie ab oben		

BEISPIELE:

```
10 OPEN "R",1,"0:DATEN.rnd",256
20 FIELD 1,128 AS SEKTOR.1$,128 AS SEKTOR.2$
30 Z%=0:WHILE LOF(1) o 256 AND Z%<256
40 GET 1:Z%=Z%+1     :REM weiterverarbeiten Record
90 WEND:IF Z%(256 THEN PRINT "letzter Sektor!":GOTO 99
91 PRINT "zuviele Sektoren belegt, keine Aussage möglich"
99 CLOSE 1:END
```

VERWEISE: Kapitel 4.3.10.2
Anweisung: **OPEN**
Funktion: **LOC**

FUNKTION: **LOG**

ZWECK: liefert den natürlichen Logarithmus zur Basis e eines numerischen Werts

FORMAT: **LOG**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet. Von ihm wird der natürliche Logarithmus ermittelt.

BEMERKUNGEN:
- Das Argument muß größer als Ø sein.
- Das Ergebnis ist in der Genauigkeit des Arguments.
- Logarithmen zu einer anderen Basis als e können nach folgender Formel berechnet werden (B=Basis):

  $\log_B X$=**LOG**(X)/**LOG**(B)

BEISPIELE:

```
10 PRINT LOG(EXP(1));LOG(EXP(1))/LOG(10)
RUN
 1   .434294
```

VERWEISE: Kapitel 4.3.16

FUNKTION: **LPOS**
(line position)

ZWECK: liefert die aktuelle Position des Druckpointers innerhalb des Druckpuffers

FORMAT: **LPOS(num.Variable)**

num.Variable: Pseudo-Argument;
darf kein Array-Element sein

WIRKUNG: Als Ergebnis wird angegeben, das wievielte Element des Druckpuffers als nächstes ausgegeben würde.

BEMERKUNGEN:
- Der Wert von 'num.Variable' wird nicht verändert.
- Es wird nicht notwendigerweise die physische Spalten-Position des Druckkopfes angegeben.

BEISPIELE:

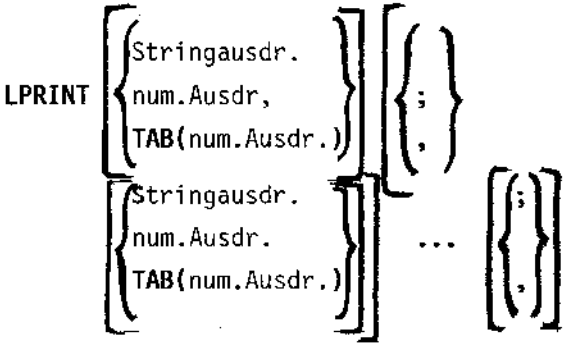
```
10 X%=100:LPRINT "123456789";
20 POS1%=LPOS(X%):LPRINT "ABCDEFGH"
30 POS2%=LPOS(X%):LPRINT POS1%,POS2%,X%
RUN
123456789ABCDEFGH
 10            1             100
```

VERWEISE: Kapitel 4.3.9.2
Anweisungen: **LPRINT, LPRINT USING, WIDTH**

ANWEISUNG: **LPRINT**
(line print)

FUNKTION: Ausgabe von Zahlen und/oder Strings auf dem Drucker

FORMAT:



WIRKUNG: Die Ergebnisse der Ausdrücke werden berechnet und von links nach rechts der Reihe nach auf dem Drucker ausgegeben. Numerische Werte werden im Standardformat ausgegeben.
Die Position der Zeichen in der Druckzeile kann mit den Anweisungselementen
- **TAB**(num.Ausdruck);
- , (Komma)
- ; (Strichpunkt

beliebig festgelegt werden.
LPRINT ohne Angabe von Ausgabeelementen bewirkt die Ausgabe einer Leerzeile.

Stellenkontrolle der Zeichen in der Druckzeile

Die **LPRINT**-Anweisung erzeugt eine der auszugebenden Folge von Ausdrücken entsprechende Folge von Zeichen in einem Puffer.

Ein Pointer weist auf die erste freie Stelle des Puffers. Erst wenn der Puffer gefüllt ist, wird sein Inhalt als eine Druckzeile ausgegeben und der Pointer auf die erste Stelle des Puffers zurückgesetzt.
Mit den Elementen ,, ; und **TAB**(num.Ausdr.) kann das Ausgabefomrat einer Druckzeile festgelegt werden.
Eine weitergehende Formatierung der Ausgabe kann durch **LPRINT USING** erreicht werden.

BEMERKUNGEN:

- Im Standardformat dargestellte Zahlen belegen bei Zahlen größer oder gleich Ø die erste Druckposition mit Blank, bei negativen Zahlen die erste Druckposition mit -. Außerdem wird am Ende bei allen Zahlen im Standardformat ein Blank gedruckt.
- Die Funktion **TAB** bewirkt die Ausgabe des nächsten Elements ab einer bestimmten Position.
- Durch Verwendung des Kommas (,) als Trennzeichen wird der Puffer des Druckers in Druckzonen zu je 14 Stellen aufgeteilt. Es wird ein Druck der Ergebnisse ab den Spalten 'aktuelle Ausgabeposition', 'akt.A.p.'+14, 'akt.A.p.'+28 etc. versucht. Sind diese belegt, wird ab der nächsten freien Druckzeile, ggf. in der nächsten Zeile, angedruckt.
- Durch Verwendung des Strichpunkt (;) als Trennzeichen wird erreicht, daß die nächste Ausgabe unmittelbar nach der letzten erfolgt.

- Die Ausgabe der Ausdrücke in der nächsten **LPRINT**-Anweisung im Programm erfolgt in derselben Zeile, wenn die Liste der Ausgabeelemente der vorangegangenen **PRINT**-Anweisung durch Komma oder Strichpunkt abgeschlossen wurde und aufgrund der aktuellen Position sich nicht zwingend eine Zeilenschaltung ergibt.
- Mit Hilfe der Anweisung **WIDTH LPRINT** kann der Default-Wert für die Druckzeilenlänge (132 Zeichen) im Programm verändert werden.
- Ist am Ende einer **LPRINT**-Anweisung kein Trennzeichen vorhanden, beginnt die Ausgabe in einer neuen Zeile und der Inhalt des Druckpuffers wird grundsätzlich ausgegeben.

BEISPIELE:

```
10 LPRINT "1. Eingabe";:LPRINT "2. Eingabe"
20 LPRINT TAB(5);"3. Eingabe";TAB(10);"4. Eingabe"
RUN
1. Eingabe2. Eingabe
     3. Eingabe
          4. Eingabe
```

VERWEISE: Kapitel 4.3.9.2 und 5.1
Anweisungen: **PRINT, LPRINT USING, WIDTH**
Funktionen: **TAB, SPC, LPOS**
Fehler-Code: 23

ANWEISUNG: **LPRINT USING**
(line print using)

FUNKTION: Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Drucker

FORMAT: **LPRINT** [**TAB**(num.Ausdruck)] [;]
USING Formatstring; Liste von Ausdrücken [;]

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: Die Daten der 'Liste von Ausdrücken' werden so formatiert wie durch den Anweisungsbestandteil **USING** definiert und am Drucker ausgegeben.

BEISPIELE: Siehe Anweisungsbestandteil **USING** und Anweisung **LPRINT**

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **LPRINT**
Kapitel 4.3.9.2

ANWEISUNG: LSET

(left set)

FUNKTION: Zuweisung eines linksbündigen, rechts eventuell mit Blanks ausgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable

FORMAT: **LSET** $\left\{\begin{matrix}\text{Feldvar.}\\ \text{Stringvar.}\end{matrix}\right\}$ =Stringausdr.

WIRKUNG: Der 'Stringausdruck' wird berechnet. Das Ergebnis wird linksbündig in das durch 'Feldvariable' definierte Feld (vgl. **FIELD**-Anweisung) eines Random-File-Puffers gebracht, wobei Feldreste automatisch mit Blanks gefüllt werden.
Bei einer **LSET**-Zuweisung auf eine normale Stringvariable wird deren alter Inhalt linksbündig durch das Ergebnis von 'Stringausdruck' ersetzt. Diejenigen Zeichen von 'Stringvariable', die bei der linksbündigen Ersetzung nicht betroffen werden, werden durch Blanks ersetzt.

BEMERKUNGEN:
- Hat das Ergebnis von 'Stringausdruck' mehr Zeichen als 'Feldvar.' bzw. 'Stringvar.', werden die überschüssigen Zeichen rechts abgeschnitten.
- Mit **MKI$**, **MKS$** oder **MKD$** gewonnene Werte sollten stets mit **RSET** auf die für sie vorgesehene Feldvariable zugewiesen werden.

ACHTUNG:
Ist die gewünschte Länge von 'Feldvar.' bzw. 'Stringvar.' nicht zuvor per **FIELD**-Anweisung oder über Zuweisung (z.B.
FELDVAR$=**SPACE$(**FELDLAENGE%**)**)
festgelegt worden, werden nur soviele Zeichen aus 'Stringausdruck' auf 'Feldvar.' bzw. 'Stringvar.' zugewiesen, wie aktuell in 'Feldvar.' bzw. 'Stringvar.' vorhanden sind!
Besonders zu beachten: Stringvariablen, die noch keinen Inhalt zugewiesen bekommen haben, haben den Default-Wert Leerstring ("") mit Ø Zeichen Länge!

BEISPIELE:

```
10 A$=SPACE$(20):B$=A$
20 LSET A$="olivetti M20":RSET B$="olivetti M20"
30 PRINT ")";A$;"(":PRINT ")";B$;"("
RUN
)olivetti M20        (
)        olivetti M20(
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **FIELD, PUT, RSET**

FUNKTION: Kombination eines sich im Arbeitsspeicher befindlichen Programms mit einem File, das im ASCII-Format (Parameter **A**) bei **SAVE** abgespeichert worden ist

FORMAT: **MERGE** Filename

Filename: Stringausdruck, dessen Ergebnis einen Begriff ergibt, der den Regeln zur Bildung von Filenamen gerecht wird (vgl. Kapitel 4.1.3)

WIRKUNG: Es wird das unter 'Filename' angegebene Programm, das ASCII-Format haben muß, in den Arbeitsspeicher zugeladen. Dort wird es mit dem vorhandenen Programm kombiniert. Anschließend erfolgt Übergang in den Direkt-Mode.

BEMERKUNGEN:
- Die Zeilen des Programms im Arbeitsspeicher werden mit den Zeilen des ASCII-Files verschachtelt.
- Sind Zeilennummern in beiden Programmen identisch, werden die Zeilen im Arbeitsspeicher durch die des ASCII-Files ersetzt.
- Sowohl das File im Arbeitsspeicher als auch das ASCII-File kann ein 'Text-File' sein, d.h. kann aus Zeilennummer plus beliebigem alphanumerischen Text bestehen.
- **MERGE** schließt offene Files und löscht alle Variablen.

- Disketten- und File-Passwords sind anzugeben, wenn vergeben. Die Angabe des Disketten-Passwords kann entfallen, wenn seit der letzten Angabe kein Diskettenwechsel und kein Neuladen des Betriebssystems erfolgt ist.
- Soll ein Programmmodul während eines Programmlaufs eingebunden und anschließend weitergearbeitet werden, muß die Anweisung **CHAIN** mit Parameter **MERGE** eingesetzt werden.

BEISPIELE:

```
SAVE "0:KOERPER.prg"
MERGE "0:T20500"
MERGE "0:MODULC.asc"
SAVE "0:GESPROG.prg"
```

VERWEISE: Kapitel 3.5.5 und 4.3.13
Befehl: **SAVE**
Anweisung: **CHAIN**

ANWEISUNG: **MID$...=**

FUNKTION: Ersetzen eines String-Bestandteils durch einen anderen String

FORMAT: **MID$(**Stringvar.,Position[,Anzahl Zeichen])=Stringausdruck

Position: num.Ausdruck
Anzahl Zeichen: num.Ausdruck

WIRKUNG: Der Stringausdruck wird berechnet.
'Position' und 'Anzahl Zeichen' werden berechnet und kaufmännisch gerundet. In 'Stringvar.' wird ab dem durch 'Position' definierten Zeichen (einschl.) die durch 'Stringausdruck' definierte Zeichenfolge eingesetzt. Dabei werden - von links kommend - nur soviel Zeichen aus 'Stringausdruck' zur Ersetzung herangezogen, wie durch 'Anzahl Zeichen' festgelegt. Ist 'Anzahl Zeichen' nicht angegeben, wird der gesamte 'Stringausdruck' zur Ersetzung herangezogen.

BEMERKUNGEN:
- Es handelt sich hier um eine wertändernde Anweisung, die allein stehen kann.
- 'Position' muß größer als Ø sein.
- Ist 'Anzahl Zeichen' gleich Ø, wird nichts ersetzt.
- Ist das Ergebnis von 'Stringausdruck' der Leerstring, wird nichts ersetzt.

- Der Inhalt von 'Stringvar.' muß mindestens so lang sein, daß genug Zeichen existieren, die zur Ersetzung herangezogen werden können.
- Die von der Ersetzung nicht betroffenen Zeichen des Inhalts von 'Stringvar.' bleiben unverändert.
- Wird 'Anzahl Zeichen' nicht angegeben, wird der volle 'Stringausdruck', aber nur bis zur bisherigen Länge von 'Stringvar.', übernommen.
- Würde die vorherige Länge von 'Stringvar.' überschritten, wird die Ersetzung nur bis zu dieser durchgeführt, auch wenn durch 'Anzahl Zeichen' explizit mehr verlangt wird.
- Die Länge des Strings, in dem ersetzt wird, kann nicht verändert (z.B. verkürzt) werden.

BEISPIELE:

```
10 A$=STR$(B!):P%=INSTR(A$,".")
20 IF P% THEN MID$(A$,P%,1)="," 'Ersetzen . durch ,
30 PRINT MID$(A$,2+(A!(0)) 'Abhacken Blank bei B!)=0
```

VERWEISE: Kapitel 4.3.2 und 4.3.4
Funktion: **MID$**

FUNKTION: **MID$**

ZWECK: Einem String wird ab einer bestimmten Position ein Teilstring entnommen

FORMAT: **MID$**(Ausgangsstring, Position [,Anzahl Zeichen])

Ausgangsstring: Stringausdruck
Position: num. Ausdruck, bestimmt Position, ab der der Teilstring entnommen wird
Anzahl Zeichen: num. Ausdruck, bestimmt die Anzahl zu entnehmender Zeichen

WIRKUNG: Die 'Position' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet.
'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet.
Aus dem Ausgangsstring wird ab 'Position' (inklusive) ein Teilstring in der Länge von 'Anzahl Zeichen' entnommen.
Ist 'Anzahl Zeichen' nicht angegeben, wird der gesamte Rest übernommen.

BEMERKUNGEN:
- 'Position' bzw. 'Anzahl Zeichen' müssen einen Wert zwischen 1 bzw. Ø und 255 aufweisen.
- Ist 'Position' größer als die Länge des Ausgangsstrings, ist der Teilstring der Leerstring.
- Ist die Differenz zwischen Länge des Ausgangsstrings und 'Position' kleiner als 'Anzahl Zeichen', wird der gesamte Rest übernommen.

BEISPIELE:

```
10 A$="Computer-System"
20 PRINT MID$(A$,10):PRINT MID$(A$,10,1)
RUN
System
S
```

VERWEISE: Kapitel 4.3.4
Anweisung: **MID$...=**
Funktionen: **LEFT$, RIGHT$**

OPERATOR: **MOD**

FUNKTION: Integerdivision von zwei Integerwerten; nur der Rest (Modulo) der Division wird ermittelt

FORMAT: Integer-Dividend **MOD** Integer-Divisor

WIRKUNG: Es wird ermittelt, wieviele Ganzzahleinheiten bei der Division durch 'Integer-Divisor' als Rest verbleiben.

BEMERKUNGEN:
- 14 **MOD** 4 liefert 2 und nicht Ø.5.
- Das (ganzzahlige) Ergebnis der Integerdivision (im obigen Fall 2) kann mit Hilfe des Operators **Ö** (Größbuchstabe) ermittelt werden (\ im US-ASCII-Zeichensatz).

BEISPIELE:

```
10 AX=14:BX=3:CX=-19
20 PRINT AX MOD BX;CX MOD BX;CXÖBX
RUN
 2 -1 -6
```

VERWEISE: Kapitel 2.6.1

FUNKTION: MKD$
(make double precision string)

ZWECK: Konvertierung eines doppelt genauen numerischen Werts in einen 8 Zeichen (Bytes) langen String

FORMAT: **MKD$(**num.Ausdruck**)**

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 8 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:
- Doppelt genaue numerische Werte müssen zum Schreiben in ein Random-File vorher über **MKD$** in einen String verwandelt und mit der Anweisung **RSET** in den Random-File-Puffer gebracht werden.
- **MKD$** ist die Umkehrfunktion von **CVD**.

BEISPIELE:

```
70 OPEN "R",1,F$,8  ' nur doppelt genaue Zahlen
75 FIELD 1,8 AS DZAHL$
80 INPUT "lfd. Nr.";N%:IF N%<=0 THEN CLOSE:END
81 INPUT "Wert ";Z#:RSET DZAHL$=MKD$(Z#)
85 PUT 1,N%:GOTO 80
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **RSET, FIELD**
Funktionen: **CVD, MKI$, MKS$**

FUNKTION: **MKI$**
(make integer string)

ZWECK: Konvertierung eines Integer-Werts in einen 2 Zeichen (Bytes) langen String

FORMAT: **MKI$(**num.Ausdruck**)**

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 2 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:
- Numerische Integer-Werte müssen zum Schreiben in ein Random-File vorher über **MKI$** in einen String verwandelt und mit der Anweisung **RSET** in den Random-File-Puffer gebracht werden.
- **MKI$** ist die Umkehrfunktion von **CVI**.

BEISPIELE:

```
70 OPEN "R",1,F$,2  ' nur Integer-Zahlen
75 FIELD 1,2 AS IZAHL$
80 INPUT "lfd. Nr.";N%:IF N%<=0 THEN CLOSE:END
81 INPUT "Wert ";Z%:RSET IZAHL$=MKI$(Z%)
85 PUT 1,N%:GOTO 80
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **RSET, FIELD**
Funktionen: **CVI, MKS$, MKD$**

FUNKTION: MKS$

(make single precision string)

ZWECK: Konvertierung eines einfach genauen numerischen Werts in einen 4 Zeichen (Bytes) langen String

FORMAT: **MKS$(**num.Ausdruck**)**

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 4 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:
- Einfach genaue numerische Werte müssen zum Schreiben in ein Random-File vorher über **MKS$** in einen String verwandelt und mit der Anweisung **RSET** in den Random-File-Puffer gebracht werden.
- **MKS$** ist die Umkehrfunktion von **CVS**.

BEISPIELE:

```
70 OPEN "R",1,F$,4  ' nur einfach genaue Zahlen
75 FIELD 1,4 AS EZAHL$
80 INPUT "lfd. Nr.";NX:IF NX<=0 THEN CLOSE:END
81 INPUT "Wert ";Z!:RSET EZAHL$=MKS$(Z!)
85 PUT 1,NX:GOTO 80
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **RSET, FIELD**
Funktionen: **CVS, MKI$, MKD$**

FUNKTION: Änderung des Namens eines Programm- oder Datenfiles auf Diskette

FORMAT: **NAME** Filename **AS** Filebezeichnung

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

Filebezeichnung: Stringausdruck; bestimmt die neue Filebezeichnung (vgl. Kapitel 4.1.3)

WIRKUNG: Dem mit Hilfe von 'Filename' spezifizierten File wird die mit 'Filebezeichnung' definierte neue Filebezeichnung zugeordnet. Der 'Filename' muß auf der angegebenen Diskette oder Station vorhanden sein. Es darf sich jedoch kein File mit dem Namen 'Filebezeichnung' auf der Diskette oder Station befinden. Nach der Ausführung des Befehls **NAME** befindet sich das angesprochene File auf derselben Diskette am gleichen Platz mit neuem Namen.

BEMERKUNGEN:
- Der PCOS-Befehl **fr** hat die gleiche Wirkung wie der BASIC-Befehl **NAME**.
- Die Diskette oder das unter 'Filename' spezifizierte File darf keinen Schreibschutz haben.
- File-Password und Disketten-Password, wenn vorhanden, werden nicht geändert.

- File-Password und/oder Disketten-Password, wenn vorhanden, müssen bei 'Filename' angegeben werden.
- Das File-Password kann mit **NAME** nicht geändert werden, sondern nur mit dem PCOS-Befehl **fp**.

BEISPIELE:

```
NAME "1:DATEN.rnd" AS "ALT.rnd"
```

VERWEISE: Kapitel 4.1.3

BEFEHL: **NEW**

FUNKTION: Löschen des Arbeitsspeicher-Inhalts; Vorbereitung zur Eingabe eines BASIC-Programms oder mit Zeilennr. versehenen Textes

FORMAT: **NEW**

WIRKUNG: Der Inhalt des Arbeitsspeichers wird gelöscht. der gesamte Datenbereich wird gelöscht. Alle offenen Datenfiles werden geschlossen. Wurde vorher der Befehl **TRON** gegeben, so wird dieser aufgehoben. Der Befehl **NEW** dient zur Vorbereitung des Systems auf die Neueingabe eines Programms oder mit Zeilennr. versehenen Textes über die Tastatur.

BEMERKUNGEN:
- Da der bestehende Inhalt des Arbeitsspeichers durch die Eingabe des Befehls **NEW** gelöscht wird, muß vor diesem Befehl der Befehl **SAVE** stehen, wenn der alte Inhalt gespeichert werden soll.
- Soll ein Programm mit den Befehlen **RUN** oder **LOAD** von der Diskette in den Arbeitsspeicher geladen werden, ist die Ausführung des Befehls **NEW** nicht notwendig.
- Wurden vorher Windows angelegt und aktiviert, bewirkt **NEW** nicht das Schließen aller Windows. Es sollte deshalb zusätzlich **CLEAR** (z.B. im Direkt-Mode) eingegeben weren.

- **NEW** bewirkt auch nicht die Wiederherstellung des mit **ss** gesetzten Default-Werts für die Anzahl Zeichen pro Bildschirmzeile bzw. der Default-Werte für den Drucker, falls diese mit der Anweisung **WIDTH** bzw. **WIDTH LPRINT** vorher geändert wurden. Eine neue **WIDTH**-Anweisung - ggf. im Direkt-Mode - ist nötig.

BEISPIELE:

```
NEW
```

VERWEISE: Kapitel 3
Anweisungen: **CLOSE WINDOW, WIDTH**

ANWEISUNG: **NEXT**

FUNKTION: Endanweisung einer oder mehrerer **FOR-NEXT** Schleifen

FORMAT: **NEXT** [Laufvar.1] [,Laufvar.2] ...

Laufvar.: numerische Variable; darf nicht Element eines Arrays und nicht doppelt genau sein

WIRKUNG: siehe Anweisung **FOR**

BEMERKUNGEN:
- Die am weitesten links angegebene Laufvariable bezieht sich auf die innerste Schleife.
- Wird die 'Laufvariable' weggelassen, wird diejenige 'Laufvariable' verarbeitet, die bisher am weitesten innen steht und deren Endwert noch nicht über/unterschritten ist.
- Fallen mehrere **NEXT**-Anweisungen an einer Programmstelle zusammen, kann dies durch die Angabe aller betroffenen 'Laufvariablen' hinter einem **NEXT**, getrennt durch Komma, erklärt werden. Diese Formulierung kann jedoch bei bestimmten Datenkonstellationen zu einer fehlerhaften Anzahl von Schleifendurchläufen führen.

BEISPIELE: siehe **FOR**

VERWEISE: Kapitel 4.3.6
Anweisungen: **FOR, CLEAR**
Fehler-Code: 1

OPERATOR: **NOT**

FUNKTION: Negation eines logischen Ausdrucks bzw. Bildung des Einerkomplements eines Integerwerts

FORMAT: **NOT** {log.Ausdruck / Integerwert}

Integerwert: num.Ausdruck

WIRKUNG: Ist ein logischer Ausdruck "wahr", liefert **NOT** -1, ist er "falsch" liefert **NOT** den Wert Ø.

Ist der Operand ein Integerwert, wird das Einerkomplement entwickelt (alle 16 Bits werden in ihrem Wert vertauscht).

BEISPIELE:

```
280 WHILE NOT EOF(1)
290 Z%=-NOT(Z%)   ' erhöhen positiven Wert um 1
300 REM  .....
390 WEND
```

VERWEISE: Kapitel 2.6.3
Operatoren: **AND, OR, XOR, EQV, IMP**

FUNKTION: verzögert die Ausführung von Anweisungen, die Ausgabeeinheiten betreffen

FORMAT: **NULL** num. Ausdruck

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit gerundet.
Die Ausführung von Anweisungen, die Ausgabe bewirken (z.B. **PRINT**) wird verzögert. Je größer der Wert von 'num.Ausdruck' ist, desto größer ist die Verzögerung.

BEMERKUNGEN:
- 'Num.Ausdruck' muß zwischen Ø und 255 liegen; sonst Meldung "Illegal function call".
- Die Anweisung kann nur durch eine neue **NULL**-Anweisung aufgehoben werden.
- Anwendungsmöglichkeiten:
  . Verzögerung von Ausgabezeilen am Bildschirm
  . Synchronisation der Übertragungsraten an periphere Einheiten (z.B. Lochstreifenstanzer oder Teletypekompatible Bildschirme)

BEISPIELE:

```
1 NULL 255:GOSUB 150
100 NULL 0:GOSUB 150
110 END
150 FOR I%=1 TO 150:PRINT I%:NEXT:RETURN
```

FUNKTION: **OCT$**

ZWECK: liefert den oktalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung

FORMAT: **OCT$(**num.Ausdruck**)**

WIRKUNG: Der numerische Ausdruck wird berechnet, kaufmännisch gerundet und das Ergebnis als Dezimalzahl interpretiert. Davon wird die oktale Darstellung ermittelt und in einen String verwandelt.

BEMERKUNGEN:
- Das Ergebnis von 'num.Ausdruck' muß zwischen Ø und 65536 liegen, sonst wird "Overflow" gegeben. Für negative Zahlen bis -65536 werden ebenfalls Umformungen vorgenommen; diese sind jedoch die oktalen Darstellung der 16-Bit-Komplemente der positiven Zahl.
- Die oktale Zahlendarstellung selbst wird in einen String verwandelt. Es wird kein führendes Blank und kein **&O**-Präfix vorangestellt.

BEISPIELE:

```
10 PRINT OCT$(9)
RUN
 11
```

VERWEISE: Kapitel 2.5.3.1
Funktionen: **HEX$, VAL**

ANWEISUNG: **ON ERROR GOTO**

FUNKTION: Verzweigung zu einer Fehlerbehandlungsroutine bei Auftreten eines BASIC-Fehlers

FORMAT: **ON ERROR GOTO** { Ø / Zeilennummer }

Zeilennummer: Zeilennummer des Programms

WIRKUNG: Tritt ein Fehler nach der Ausführung der **ON ERROR GOTO**-Anweisung auf, springt das System von sich aus sofort auf die angegebene Zeilennummer und führt dort das Programm weiter aus, bis eine **RESUME**-Anweisung für den Rücksprung auftritt. Eine aktivierte Fehlerbehandlungsroutine bleibt bis zur Desaktivierung durch **ON ERROR GOTO Ø** oder durch Aktivieren einer anderen Fehlerbehandlungsroutine aktiv.

BEMERKUNGEN:
- Die Fehlerbehandlungsroutine ist ein vom Anwender definiertes Unterprogramm, das mit einer oder mehreren **RESUME**-Anweisung(en) endet.
- Die Fehlerursache kann in der Fehlerbehandlungsroutine behandelt werden. Danach kann die BASIC-Anweisung wiederholt werden, die den Fehler verursachte, oder die nächste darauf folgende (auch bei Mehrfachanweisung) oder aber eine andere Programmzeile (vgl. **RESUME**-Anweisung).

- Durch eine neue **ON ERROR GOTO**-Anweisung kann eine neue Fehlerbehandlungsroutine definiert werden. Die alte Routine wird dann nicht mehr angesprungen, wenn ein Fehler auftritt.
- Wird der Parameter **Ø** oder keiner angegeben, wird die zuletzt definierte Fehlerbehandlungsroutine desaktiviert, ohne daß eine neue gilt.
- Tritt innerhalb der Fehlerbehandlungsroutine **ON ERROR GOTO Ø** auf, wird der Fehler gemeldet und in den Direkt-Mode verzweigt.
- Die Fehlerbehandlungsroutine bleibt über das Programmende hinaus und auch im Direkt-Mode (z.B. nach Unterbrechungen) aktiv, sofern sie nicht mit **ON ERROR GOTO Ø** abgeschaltet wird.

BEISPIELE:

```
0 'Flächenberechnung Dreieck
20 A$=" Seite a=###.## _Seite b=###.## _Seite c=###.##"+
"  Fläche=#######.##"
30 ON ERROR GOTO 100
40 INPUT "Seite a,Seite b,Seite c ";A#,B#,C#
50 IF A#*B#*C#=0 THEN ERROR 200 'mind. 1 Wert = 0
60 S#=(A#+B#+C#)/2
70 F2#=S#*(S#-A#)*(S#-B#)*(S#-C#)
80 IF F2#<=0 THEN ERROR 210
85 LPRINT USING A$;A#,B#,C#,SQR(F2#):GOTO 40
100 IF ERR=200 THEN PRINT "a oder b oder c < 0":RESUME 40
110 IF ERR=210 THEN PRINT "kein Dreieck!":RESUME 40
120 ON ERROR GOTO 0:STOP   'sonstiger Fehler
```

VERWEISE: Kapitel 4.3.11 und 4.3.12
Anweisungen: **RESUME, ERROR**
reservierte Variablen: **ERR, ERL**
Fehler-Code: 8, 19

ANWEISUNG: **ON...GOSUB**

FUNKTION: Sprung in ein Unterprogramm; das Sprungziel ist abhängig vom Wert eines num. Ausdrucks

FORMAT: **ON** num.Ausdr. **GOSUB** Zeilennr. [, Zeilennr.] ...

Zeilennumer: gibt das Sprungziel an

WIRKUNG: Der numerische Ausdruck wird berechnet und das Ergebnis gerundet.
Diese ganze Zahl gibt an, zur wievielten der Zeilennummern rechts von **ON...GOSUB** gesprungen werden soll. Jede Zeilennummer in der Anweisung gibt die erste Zeile des jeweiligen Unterprogramms an. Jedes Unterprogramm enthält die Anweisung **RETURN**, die den Rücksprung in die Anweisung nach **ON...GOSUB** bewirkt.

BEMERKUNGEN:
- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.
- In einem Unterprogramm können auch mehrere **RETURN**-Anweisungen vorkommen.
- Ergibt der numerische Ausdruck nach der Rundung eine Zahl, die Ø ist oder größer als die Anzahl Zeilennummern in der Anweisung **ON...GOSUB**, wird das Programm mit der nächsten Anweisung nach **ON...GOSUB** fortgesetzt.
- Ist der numerische Ausdruck im Ergebnis negativ oder größer als 255, wird "Illegal function call" gemeldet.

BEISPIELE:

```
10 PRINT "e(rfassen) ä(ndern) l(isten) "
20 LINE INPUT ;"",E$:IF E$="" then 10
30 A$=CHR$(ASC(LEFT$(E$,1)) AND 223) 'nur Großschreibung
40 ON INSTR("EÄL",A$) GOSUB 1001,2001,3001:END
45 GOTO 10 'weder E noch Ä noch L (INSTR liefert 0)
50 ' 1001 erfassen 2001 ändern 3001 listen
```

VERWEISE: Kapitel 4.3.5 und 4.3.7
Anweisungen: **GOSUB, RETURN, CLEAR**
Fehler-Code: 8

ANWEISUNG: **ON ... GOTO**

FUNKTION: Sprung in eine bestimmte Programmzeile; das Sprungziel ist abhängig vom Wert eines num. Ausdrucks

FORMAT: **ON** num. Ausdr. **GOTO** Zeilennr. [,Zeilennr.] ...
Zeilennummer: gibt das Sprungziel an

WIRKUNG: Der numerische Ausdruck wird berechnet und das Ergebnis gerundet. Diese ganze Zahl gibt an, zur wievielten der Zeilennummern rechts von **ON ... GOTO** gesprungen werden soll.

BEMERKUNGEN:
- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.
- Ergibt der numerische Ausdruck nach der Rundung eine Zahl, die Ø ist oder größer als die Anzahl Zeilennummern in der Anweisung **ON ... GOTO**, wird kein Sprung ausgeführt, sondern das Programm mit der nächsten Anweisung nach **ON ... GOTO** fortgesetzt.
- Ist der numerische Ausdruck im Ergebnis negativ oder größer als 255, wird "Illegal function call" gemeldet.

BEISPIELE:

```
10 INPUT "Wählen Sie 0 - 3 ";W%
20 IF W%=0 THEN END
25 IF W%>3 THEN 10
30 ON W% GOTO 101,201,301
101 PRINT 1:GOTO 10
201 PRINT 2:GOTO 10
301 PRINT 3:GOTO 10
```

VERWEISE: Kapitel 4.3.5
Anweisung: **GOTO**
Fehler-Code: 8

FUNKTION: Öffnen eines sequentiellen oder Random-Files für Zugriffe; Festlegung der Puffer-Länge für Zugriffe auf ein Random-File; (evtl. auch Neuanlage eines nicht vorhandenen Files bzw. Vorbereitung zur automatischen Verlängerung eines vorhandenen Files)

FORMAT: **OPEN** Zugriffsart,[#]Filenr.,Filename[,Record-Länge]

Zugriffsart: String-Ausdr. (Ergebnis: "I", "O", "A" oder "R")
Filenr.: num.Ausdr. (Ergebnis: 1 - 15)
Filename: String-Ausdr., muß als Ergebnis einen String liefern, der den Regeln zur Bildung von Filenamen genügt (vgl. Kapitel 4.1.3)
Record-Länge: num.Ausdr. (Ergebnis: 1 bis 4Ø96)

WIRKUNG: Die 'Filenr.' und die 'Record-Länge' werden errechnet und ggf. durch kaufmännische Rundung der ersten Nachkommastelle davon der ganzzahlige Wert ermittelt.
Ist das durch 'Filename' definierte File auf der angesprochenen Diskette noch nicht vorhanden und die Zugriffsart ist "O" oder "R", wird es auf Diskette angelegt.
Es erhält die ermittelte 'Filenr.' zugeordnet, unter der es von da an anzusprechen ist.

Liefert 'Zugriffsart' den Wert "I", wird das File als sequentielles File nur für Inputs geöffnet, bei "O" als sequentielles File nur für Outputs ab File-Beginn. Nach Zugriffsart "O" ist sofort der gesamte bisherige File-Inhalt verloren. Ist der Wert "A", wird das File als sequentielles File nur für Outputs ab dem bisherigen logischen File-Ende geöffnet ("Append"-Wirkung). Ist der Wert "R", wird es als Random-File sowohl für Inputs als auch für Outputs geöffnet.

Handelt es sich um 'Zugriffsart' "R", sollte 'Recordlänge' unbedingt angegeben sein; bei allen anderen 'Zugriffsarten' darf 'Recordlänge' nicht angegeben sein.

Durch 'Recordlänge' wird die logische Länge des Random-File-Puffers (in Bytes) vereinbart.

BEMERKUNGEN:

- Noch nicht vorhandene Datenfiles sollten stets mit dem PCOS-Befehl **fn** angelegt werden.
- Die Angabe von 'Record-Länge' führt bei sequentiellen Files zur Fehlermeldung "Illegal function call".
- Wird der Parameter 'Record-Länge' bei Random-Files weggelassen, wird als Default-Länge der im PCOS-Befehl **sb** gesetzte Wert genommen.
- Die 'Record-Länge' bei Random-Files muß zwischen 1 und 4096 liegen.

- Die Anzahl Filenummern, die mit **OPEN**-Anweisungen vergeben werden können (maximal 15) sowie die maximal einsetzbare Record-Länge wird mit dem PCOS-Befehl **sb** festgelegt. Normalerweise sind die Default-Werte 3 bzw. 128. Die Fehlermeldungen "Bad file number" oder "Illegal function call" können aus solchen zu niedrigen Default-Werten herrühren.
- Ist die 'Filenr.' im Programm bereits vergeben, wird die Fehlermeldung 'File already open" gegeben.
- Files auf Disketten mit Schreibschutz können geöffnet, aber später nicht durch **PUT, PRINT#, PRINT# USING** oder **WRITE#** beschrieben werden.
- Disketten-Passwords oder File-Passwords müssen beim 'Filenamen' enthalten sein.
- Das Disketten-Password muß nur dann angegeben werden, wenn es seit dem Einschalten bzw. dem letzten Diskettenwechsel noch nicht angegeben worden ist (z.B. bei einem anderen File).
- Wird der 'Filename' auf Diskette nicht gefunden, wird die Fehlermeldung "File not found" gegeben. Dies gilt nur bei Zugriffsart "I", angewandt auf alle Typen von Datenfiles. Für die anderen Zugriffsarten gilt: Ist das File nicht vorhanden, wird es angelegt. Reicht dafür der Platz aufgrund des PCOS-Befehls **ss** nicht aus, wird der Fehler 61 ("Disk full") gemeldet.
- Ist das Password des Files, falls vergeben, nicht korrekt angegeben, wird die Fehlermeldung "Invalid password" gegeben.

- Entspricht die 'Record-Länge' nicht der, die bei früherem Öffnen des Files angegeben wurde, wird keine Fehlermeldung gegeben
- Wird nach **OPEN** ein File automatisch angelegt oder vergrößert, werden im Inhaltsverzeichnis zwar die dabei betroffenen Sektoren als "belegt" vermerkt; die Sektoren des Files selbst werden hingegen nicht mit **CHR$(Ø)** vorbelegt!

BEISPIELE:

```
10 CLEAR:OPEN "I",1,"1:TEXTE.asc"
15 OPEN "O",2,"1:TEXTKOPIE.asc"
17 OPEN "A",3,"1:TEXTANH.seq"
20 OPEN "R",4,"1:TEXTE.rnd",1
22 FIELD 4,1 AS ZEICHEN$
90 WHILE NOT EOF(1)
100 LINE INPUT#1,T$:PRINT#2,T$:PRINT#3,T$
103 PRINT T$
105 WEND:CLOSE 1,2,3
106 PRINT "zeichenweise lesen, bei Code 0 bel. Taste"
110 GET 4:PRINT ZEICHEN$,"Code = ";ASC(ZEICHEN$)
120 K$=INKEY$:IF K$="" THEN 110 ELSE CLOSE 4:END
```

Bei der BASIC-Anweisung **OPEN** ist eine neue 'Zugriffsart' möglich, nämlich **"E"**.
Dies bedeutet: Zugriff wie bei **"R"** (also auch Angabe der 'Recordlänge' erforderlich). Es kann aber nur gelesen werden (**GET**-Anweisung).

Ist das File bei **OPEN** noch nicht vorhanden, wird es bei Zugriffsart **"E"** - wie bisher nur bei Zugriffsart **"I"** - nicht angelegt.

Unter LAN gelten folgende Regeln für Zugriffe auf eine gemeinsam genutztes File (auf der File-Server-HDU):

Ist ein File von einem Satellit mit einer Zugriffsart geöffnet, die Schreiben ermöglicht (**"O"**, **"A"** oder **"R"**), ist ein **OPEN** von anderen Satelliten in dieses File solange unmöglich ("locked status"), wie vom öffnenden Satellit her kein **CLOSE** für das File erfolgt (BASIC-Fehlermeldung).

Nach einem **OPEN** mit Zugriffsart **"I"** oder **"E"** können jedoch auch die anderen Satelliten noch öffnen.

Dieses Blatt betrifft das BASIC-Handbuch

VERWEISE: Kapitel 4.1.3 und 4.3.10

Anweisungen: **CLOSE, FIELD, GET, PUT, RSET, LSET, INPUT#, LINE INPUT#, PRINT# (USING), WRITE#**

Funktionen: **CVI, MKI$, CVS, MKS$, CVD, MKD$, EOF, LOC, LOF**

Fehler-Codes: 53, 54, 55, 61, 64, 67

ANWEISUNG: **OPTION BASE**

FUNKTION: Festlegung des kleinsten möglichen Index-Wertes aller Arrays

FORMAT: **OPTION BASE** {Ø | 1}

WIRKUNG: Es wird der kleinste mögliche Index-Wert für alle Arrays des Programms festgelegt. Der kleinstmögliche Index-Wert kann Ø oder 1 sein.

BEMERKUNGEN:
- Der Default-Wert ist Ø, d.h., daß alle Arrays mit dem Index Ø beginnen, falls keine andere festlegende **OPTION BASE**-Anweisung existiert.
- Es darf pro Programm nur eine **OPTION BASE**-Anweisung (vor allen **DIM**-Anweisungen) vorkommen.
- Wird ein Array mit höchstem Index X dimensioniert (**DIM** A(X)), enthält der Array
  bei **OPTION BASE** 1 X Elemente und
  bei **OPTION BASE** Ø X+1 Elemente.

BEISPIELE:

```
10 OPTION BASE 0
20 DIM A!(10,10) ' ergibt 11x11=121 Elemente

10 OPTION BASE 1
30 DIM A$(1000):A$(0)="" 'liefert
    Subscript out of range
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisung: **DIM**

OPERATOR: **OR**

(implicit or)

FUNKTION: Verknüpfung zweier logischer Ausdrücke mit Hilfe der implizierten ODER-Verknüpfung bzw. Bildung der logischen Summe zweier Integerwerte (Bit-für-Bit-Verknüpfung mit **OR**)

FORMAT: {log.Ausdruck / Integerwert} **OR** {log.Ausdruck / Integerwert}

WIRKUNG: Das Ergebnis der **OR**-Verknüpfung zweier Bedingungen ist

-1 (wahr), wenn eine der beiden oder auch beide Bedingungen erfüllt sind;

Ø (falsch), wenn keine der beiden Bedingungen erfüllt sind.

Bei der **OR**-Verknüpfung zweier Integerwerte wird die **OR**-Operation Bit für Bit durchgeführt und das Ergebnis berechnet (vgl. Kapitel 2.6.4).

BEISPIELE:

```
10 A%=5:B%=3:C!=2.47
15 C%=4% OR B%  '  C% ist jetzt 7
20 IF A%<B% OR C!>3 THEN GOSUB 1000 ELSE GOSUB 2000
```

VERWEISE: Kapitel 2.6.4

Operatoren: **NOT, AND, XOR, EQV, IMP**

FUNKTION: Ausfüllen eines ganzen oder eines Teils eines Windows

FORMAT: **PAINT** [%Window-Nr.,] (X,Y)[,[Farbindex1][,Farbindex2]]

Window-Nr.: numerischer Ausdruck (Ergebnis: 1-16)
X,Y: numerische Ausdrücke, Koordinaten des Ausgangspunktes
Farbindex1: numerische Ausdrücke
Farbindex2: (Ergebnis: beim Vier-Farb-Schirm: Ø bis 3, beim Acht-Farb-Schirm: Ø - 8)

WIRKUNG: Ausgehend vom Koordinatenpunkt (X,Y) wird ein zuvor gezeichneter geschlossener Bereich mit der durch 'Farbindex1' angegebenen Farbe ausgefüllt. Liegt der Punkt (X,Y) außerhalb des Bereichs, wird der Rest des Windows ausgefüllt. Durch 'Farbindex2' wird die Farbe bestimmt, mit der der Rand des Bereichs gezeichnet wird.

BEMERKUNGEN:
- Die Parameter 'Farbindex1' und 'Farbindex2' sind für den Vier-Farb-Schirm Zahlen zwischen Ø und 3, die der Position des Farbcodes in der globalen **COLOR**=-Anweisung entsprechen. Für den Schwarz/Weiß- und den Acht-Farb-Schirm sind sie identisch mit dem 'Farbcode'.
- Der Default-Wert für 'Farbindex1' und 'Farbindex2' ist die Vordergrundfarbe.
- Liegt der Punkt (X,Y) außerhalb des angesprochenen Windows, wird "Syntax error" gegeben.

- Ist der definierte Bereich nicht geschlossen (z.B. weil bei der Interpolation in einer **CIRCLE**-Anweisung nicht alle Elementarpunkte angesprochen wurden, die den Kreis bzw. die Ellipse völlig schließen würden), bricht das System beim Ausfüllen über die Begrenzung aus.

BEISPIELE:

```
10 CLEAR:SCALE 0,511,0,255
20 CIRCLE(200,100),80
30 PAINT(210,90),1
```

VERWEISE: Kapitel 4.3.14
Anweisung: **COLOR=**

FUNKTION: **POINT**

ZWECK: Ermittlung der Farbe in einem Koordinatenpunkt

FORMAT: **POINT(X,Y)**

X,Y: numerische Ausdrücke, die den zu betrachtenden Elementarpunkt bestimmen.

WIRKUNG: Für die Koordinatenwerte 'X' und 'Y' wird entsprechend dem für das aktive Window geltenden Maßstab der zugehörige Elementarpunkt ermittelt. Beim Vierfarb-Schirm wird der Farbindex innerhalb der globalen **COLOR**=-Anweisung ermittelt, der der Farbe entspricht, die sich in dem Elementarpunkt befindet. Beim Schwarz/Weiß- und Achtfarb-Schirm wird der Farbcode des Elementarpunkts ermittelt.

BEISPIELE:

```
10 CLEAR:COLOR=4,3,8,7
20 SCALE -50,50,-50 50
30 CIRCLE(10,10),40
40 PAINT(20,10),0,3
50 FINDEX%=POINT(10,10)'liefert 3
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **COLOR, COLOR=, SCALE, WINDOW %**
Funktion: **WINDOW**

FUNKTION: **POS**
(position)

ZWECK: liefert die aktuelle Spalten- oder Zeilen-Position des Bildschirm-Cursors

FORMAT: **POS(Position)**

Position: num.Ausdruck (Ergebnis: Ø oder 1)

WIRKUNG: 'Position' wird berechnet und kaufmännisch gerundet. Ist das Ergebnis Ø, liefert die Funktion die aktuelle Spalten-Position, an der sich der Bildschirm-Cursor befindet. Ist 'Position' nicht gleich Ø, liefert die Funktion die aktuelle Zeilenposition des Bildschirm-Cursors.

BEMERKUNGEN:
- Die Spalten- bzw. Zeichen-Position bezieht sich stets auf das aktuelle Window.

BEISPIELE:

```
5 CURSOR(60,7):PRINT "1. HALLO!";
10 A%=POS(0):B%=POS(1):STOP 'Fortsetzung mit CONT
20 CURSOR(A%,B%):PRINT "2. HALLO"+SPACE$(200)
```

VERWEISE: Kapitel 4.3.9.1
Anweisungen: **PRINT, PRINT USING, CURSOR, LINE INPUT, INPUT, WINDOW%, CLS**
Funktionen: **INPUT$, INKEY$, WINDOW**

FUNKTION: PRESET
(pixel reset)

FUNKTION: Zeichnen eines Elementarpunktes in der Hintergrundfarbe

FORMAT: **PRESET**[%Window-Nr.] (X,Y)

Window-Nr.: numerischer Ausdruck
(Ergebnis: 1 bis 16)

X, Y: numerische Ausdrücke

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, wird das entsprechende Window gewählt. Fehlt der Parameter, wird die Anweisung im aktiven Window ausgeführt. Das Koordinatenpaar (X,Y) bezieht sich auf die für das Window gültige **SCALE**-Anweisung. Der entsprechend dem Maßstab diesen Koordinaten zugeordnete Elementarpunkt wird auf die aktuelle Hintergrundfarbe gesetzt.

BEMERKUNGEN: siehe **PSET**

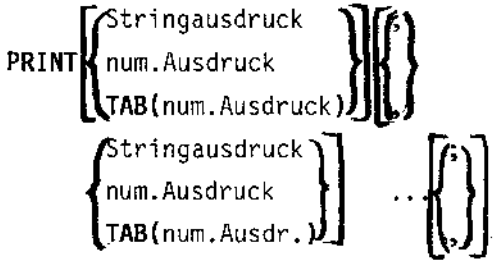
BEISPIELE:

```
0 CLEAR:SCALE 0,100,0,100
10 FOR I%=1 TO 100
20 PSET(I%/2,-I%):NEXT
30 K$=INPUT$(1)
40 FOR I%=-56 TO -50 STEP -1
50 PRESET(I%/2,I%):NEXT
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **COLOR=, COLOR, SCALE**

FUNKTION: Druck von Zahlen und/oder Strings im Standardformat auf dem Bildschirm

FORMAT:



, oder ; sind zur Trennung der Ausgabeelemente möglich

WIRKUNG: Die Ergebnisse der Ausdrücke werden berechnet und von links nach rechts der Reihe nach am Bildschirm ausgegeben. Numerische Werte werden im Standardformat ausgegeben.
Die Position der Zeichen in der ausgegebenen Zeile kann mit den Anweisungselementen
- **TAB** (num.Ausdruck);
- , (Komma)
- ; (Strichpunkt)

beliebig festgelegt werden.
PRINT ohne Angabe von Ausgabeelementen bewirkt die Ausgabe einer Leerzeile.
Mit den Elementen , , ; und **TAB** (num.Ausdruck) kann das Ausgabeformat einer Ausgabezeile festgelegt werden.
Eine weitergehende Formatierung der Ausgabe kann durch **PRINT USING** erreicht werden.

BEMERKUNGEN:

- Bei Verwendung des Kommas (,) als Zeichen zur Trennung der ausgegebenen Daten wird die Bildschirmzeile ab der aktuellen Ausgabeposition in Druckzonen zu je 14 Stellen aufgeteilt. Es wird ein Druck der Ergebnisse ab den Spalten 'aktuelle Ausgabeposition', 'akt.A.p.'+14, 'akt.A.p.'-+28 etc. versucht. Sind diese belegt, wird ab der nächsten freien Position, ggf. in der nächsten Zeile, ausgegeben.
- Im Standardformat dargestellte Zahlen belegen bei Zahlen größer oder gleich Ø die erste Position mit Blank, bei negativen Zahlen die erste Position mit -. Außerdem wird am Ende bei allen Zahlen im Standardformat ein Blank erzeugt.
- Die Funktion **TAB** bewirkt die Ausgabe des nächsten Elements ab einer bestimmten Position.
- Durch Verwendung des Strichpunkts (;) als Trennzeichen wird erreicht, daß die Ausgabe unmittelbar nach der letzten erfolgt.
- Die Ausgabe der nächsten **PRINT**-Anweisung im Programm erfolgt in der selben Zeile, wenn die Liste der Ausgabeelemente oder der vorangegangenen **PRINT**-Anweisung durch Komma oder Strichpunkt abgeschlossen wurde und aufgrund der aktuellen Position sich nicht zwingend eine Zeilenschaltung ergibt.
- Mit Hilfe der Anweisung **WIDTH** kann der Default-Wert für die Bildschirmzeilenlänge (64 oder 8Ø Zeichen) im Programm verändert werden.

- Ist am Ende einer **PRINT**-Anweisung kein Trennzeichen vorhanden, beginnt die Ausgabe einer neuen Zeile.
- Anstelle des Schlüsselworts **PRINT** kann ein ? (Fragezeichen) eingegeben werden. Dieses wird in Listings und beim Speichern des Programms durch **PRINT** ersetzt.
- Ist der Bildschirm-Zeichensatz über die PCOS-Befehle **rf**, **ed** und **wf** geändert worden, werden die dadurch definierten Zeichen angegeben.

BEISPIELE:

```
0 'Ausgabe des Zeichenvorrats am Schirm
10 PRINT "Zeichenvorrat (deutsche Tastatur):"
20 PRINT
30 FOR I%=33 TO 126:PRINT CHR$(I%),:NEXT:END
```

VERWEISE: Kapitel 4.3.9.1
Anweisungen: **LPRINT, PRINT USING, PRINT#, WIDTH**
Funktionen: **TAB, SPC**
PCOS-Befehle: **rf, ed, wf**

FUNKTION: Schreiben von Daten auf ein externes sequentielles File; Trennzeichen werden nicht automatisch auf das File geschrieben; am Ende wird ggf. ein CR und ein LF auf das File geschrieben

FORMAT: **PRINT#** Filenr.,Liste von Ausdrücken [;,]

Filenr.: num.Ausdruck (Ergebnis:1-15)
Liste von Ausdrücken: numerisch und/oder String; getrennt durch , oder ;

WIRKUNG: Die 'Filenr.' wird berechnet und gerundet. Auf das unter dieser Filenr. geöffnete sequentielle File (Zugriffsart: "O" oder "A") werden ab der Position des Pointers die Ergebnisse der (durch Trennzeichen getrennten) Ausdrücke geschrieben.
Als Zeichen zur Trennung der Ausdrücke in der 'Liste von Ausdrücken' gelten:
das Komma (,) und der Strichpunkt (;).
a) zwischen numerischen Ausdrücken:
- Es empfiehlt sich die Trennung durch Strichpunkt (;). Die numerischen Daten werden im Standardformat (vgl. **PRINT**-Anweisung) direkt hintereinander auf das File geschrieben.
- Das Komma (,) als Trennzeichen bewirkt, daß unter Verwendung von Blanks auf das File tabuliert wird (vgl. Funktion von , bei **PRINT**-Anweisungen).

b) zwischen Stringausdrücken:

- Auch hier empfiehlt sich die Trennung durch Strichpunkt (;). Dadurch werden allerdings die Daten direkt hintereinander, ohne Kennzeichnung durch Anführungszeichen, auf das File gebracht (vgl. Funktion von ; bei **PRINT**-Anweisungen), so daß eine Trennung bei späterem **INPUT#** nicht mehr möglich ist. Es müssen also zusätzlich solche Zeichen zwischen String-Daten auf das File geschrieben werden, die bei **INPUT#** bzw. **LINE INPUT#** als Trennzeichen erkannt werden (z.B. Komma (=**CHR$**(44)) oder Anführungszeichen (=**CHR$**(34)).
- Enthält ein String-Datum ein Komma oder sonstige bei **INPUT#** als Trennzeichen wirkende Zeichen, muß der String im **CHR$**(34) eingeschlossen auf das File geschrieben werden.

Am Ende einer auf das File geschriebenen 'Liste von Ausdrücken' werden ein CR (=**CHR$**(13)) und ein LF (=**CHR$**(10)) auf das File geschrieben. Diese Trennzeichen entfallen jedoch, wenn am Ende der 'Liste von Ausdrücken' ein ; steht.

BEMERKUNGEN:

- Numerische Werte werden im Standardformat auf das File gebracht (vgl. **PRINT**).
- Enthalten Stringvariablen führende oder am Ende stehende Blanks, werden diese auf das File geschrieben.

- **PRINT#** schreibt die Daten im Grunde so auf das File, wie sie durch **PRINT**-Anweisungen auf dem Bildschirm dargestellt würden, allerdings einschließlich nicht druckbarer Zeichen.
- Im Gegensatz zu **WRITE#** werden die Daten nicht automatisch durch ein Trennzeichen getrennt auf das File gebracht; die Strings werden nicht durch Anführungszeichen gekennzeichnet und das Ende der 'Liste von Ausdrücken' nur auf Wunsch mit CR und ggf. LF gekennzeichnet (Weglassen des ; am Ende der 'Liste von Ausdrücken').
- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu reservieren, wie durch den PCOS-Befehl **ss** spezifiziert wurde. Die kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.
- Mit Hilfe der Funktion **LOC** kann abgefragt werden, ob auf Diskette noch genügend Platz für die von **PRINT#** betroffenen Daten ist.
- Das Schreiben auf Diskette erfolgt nicht automatisch sofort nach **PRINT#**, sondern erst, wenn der File-Puffer voll ist. Erst nach **CLOSE** für das betreffende File ist der Rest des File-Puffers mit Sicherheit geschrieben.

BEISPIELE:

```
10 OPEN "O",1,"10:WERTE.seq"
20 A%=1:B!=2.7:C#=30180:D$="Ende"
30 PRINT#1,A%;B!;C#;D$
40 CLOSE 1:OPEN "I",1,"10:WERTE.seq"
60 LINE INPUT#1,L$
70 PRINT L$
80 CLOSE 1:END
RUN
 1  2.7  30+180 Ende
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, INPUT#, LINE INPUT#, WRITE#, PRINT**

Funktion: **LOC, LOF**

Fehler-Codes: 54, 61

ANWEISUNG: **PRINT USING**

FUNKTION: Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Bildschirm

FORMAT: **PRINT** [TAB(num.Ausdruck) {;}]
USING Formatstring; Liste von Ausdrücken [{;} {,}]

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma Getrennt sind

WIRKUNG: Die Daten der 'Liste von Ausdrücken' werden so formatiert, wie durch den Anweisungsbestandteil **USING** definiert, und am Bildschirm angegeben.

BEISPIELE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT**

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT**
Kapitel 4.3.9.2

FUNKTION: Schreiben von formatierten Daten (numerisch und/oder String) in ein sequentielles File

FORMAT: **PRINT #** Filenr., [**TAB**(num.Ausdruck)] [;]

USING Formatstring; Liste von Ausdrücken [{;,}]

'Filenr.: num.Ausdruck (Ergebnis:1-15)

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: Die Daten der 'Liste von Ausdrücken' werden so formatiert, wie durch den Anweisungsbestandteil **USING** definiert, und in das mit **OPEN** unter der 'Filenr.' zuvor geöffnete sequentielle File geschrieben.

BEISPIEL: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT#**

```
10 OPEN "0",1,"10:WERTE.seq"
20 A%=1:B!=2.7:C#=3D180:D$="Ende"
30 PRINT#1,USING "##  #.#^^^^  #.#^^^^ &";A%,B!,C#,D$
40 CLOSE 1:OPEN "I",1,"10:WERTE.seq"
60 LINE INPUT#1,U$
70 PRINT U$
80 CLOSE 1:END
RUN
 1  .27E+01  .3D+181 Ende
```

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT#**

Kapitel 4.3.9.2

ANWEISUNG: PSET
(pixel set)

FUNKTION: Zeichnen eines Punktes mit vorgewählter Farbe

FORMAT: **PSET** [%Window-Nr.] (X,Y) [,Farbindex]

Window-Nr.: numerischer Ausdruck; Wert 1 - 16
X, Y: numerische Ausdrücke, die die Koordinaten des gewählten Punktes bestimmen
Farbindex: numerischer Ausdruck, am Vier-Farb-Schirm: Wert:Ø bis 3; ansonsten identisch mit 'Farbcode'.

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, wird das dadurch bestimmte Window gewählt. Fehlt dieser Parameter, bezieht sich die Anweisung auf das aktive Window.
Gemäß der für das Window gültigen **SCALE**-Anweisung wird derjenige Elementarpunkt ermittelt, der dem Koordinatenpaar (X,Y) am nächsten liegt. Dieser Elementarpunkt erhält beim Vier-Farb-Schirm die Farbe, die der durch 'Farbindex' definierten Position in der geltenden globalen **COLOR**=-Anweisung entspricht. Beim Acht-Farb- und Schwarz/Weiß-Schirm ist 'Farbindex' identisch mit 'Farbcode'.

BEMERKUNGEN:
- Liegt der durch 'X' und 'Y' bestimmte Punkt außerhalb des Windows, hat **PSET** keine Wirkung.

BEISPIELE:

```
10 CLEAR:SCALE -1,1,3*-1+5,3*1+5
20 FOR I!=-1 TO 1 STEP .1
30 PSET(I!,3*I!+5):NEXT:END
```

VERWEISE: Kapitel 4.3.14

Anweisungen: **COLOR=, SCALE, PRESET**

FUNKTION: Übertragen eines logischen Records von einem Random-File-Puffer in das zugehörige Random-File ("Schreiben")

FORMAT: **PUT**[#]Filenr. [,Record-Nr.]

Filenr.: num.Ausdruck (Ergebnis: 1-15)
Record-Nr.: num.Ausdruck (Ergebnis: 1-32767)

WIRKUNG: Aus dem Random-File-Puffer wird der durch 'Record-Nr.' spezifizierte Record in das unter 'Filenr.' geöffnete Random-File gebracht ("Schreiben").

BEMERKUNGEN:
- **PUT** ist bei Disketten mit Schreibschutz nicht möglich und führt zum Fehler "Disk I/O-Error".
- 'Record-Nr.' muß positiv sein; Ø liefert die Fehlermeldung "Bad record number"; der maximale Wert ist 32767.
- Wird 'Record-Nr.' weggelassen, wird der Puffer-Inhalt unter derjenigen Record-Nr. in das mit 'File-Nr.' spezifizierte File gebracht, die auf den letzten durch **PUT** geschriebenen bzw. durch **GET** gelesenen Record folgt ("sequentielles Schreiben").

- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu reservieren, wie durch den PCOS-Befehl **ss** spezifiziert wurde. Dies kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.
- Mit der Funktion **LOC** kann die letzte angesprochene Record-Nr. abgefragt werden.
- Im Random-File-Puffer ist vor **PUT** durch **FIELD**-Anweisung(en) Platz für die Feldvariablen zu reservieren und anschließend entweder (evtl. unter Zuhilfenahme der Funktionen **MKI$, MKS$** oder **MKD$**) über **LSET** bzw. **RSET** Werte darauf zuzuweisen oder über **GET** Werte dorthin einzulesen.
- Eine andere Möglichkeit, den Random-File-Puffer zu füllen, besteht durch Einsetzen der Anweisungen **PRINT#, PRINT#...USING** oder **WRITE#**.
  Im Fall **WRITE#** wird der Puffer rechts mit Blanks aufgefüllt.
- Das Schreiben auf Diskette erfolgt nicht immer automatisch sofort nach **PUT**, sondern erst, wenn der File-Puffer voll ist. Erst nach **CLOSE** für das betreffende File ist der Rest des File-Puffers mit Sicherheit geschrieben.

BEISPIELE:

```
10 OPEN "R",1,"DFILE.rnd",1  'zeichenweise bearbeiten
20 FIELD 1,1 AS BYTE$
30 INPUT "ab Byte Nr. ";N%:Z%=N%
40 PRINT "Welche Zeichen "
45 Z$=INPUT$(1):PRINT Z$
50 LSET BYTE$=Z$:PUT 1,Z%
60 Z%=Z%+1:IF Z%)32767 THEN 70 ELSE  45
70 Z%=1:CLS:WHILE Z$()CHR$(0)
80 GET 1,Z%:PRINT BYTE$;
90 WEND     :CLOSE 1:   END
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, FIELD, PRINT#, PRINT#...USING, WRITE#**

Funktionen: **LOC, MKI$, MKS$, MKD$, LOF**

Fehler-Codes: 54, 61, 63

FUNKTION: Ein in einem Array gespeichertes Bild wird auf dem Bildschirm dargestellt

FORMAT: **PUT** [%Window-Nr. ,] (X1,Y1)-(X2,Y2),Arrayelement [,Operand]

Window-Nr.: numerischer Ausdruck; Wert: 1-16

X1,Y1: numerische Ausdrücke, die den Anfangspunkt des Bereiches definieren

X2,Y2: numerische Ausdrücke, die den Endpunkt des Bereiches definieren

Arrayelement: Bezeichnung des Elements eines eindimensionalen Integer-Arrays, in dem das gespeicherte Bild beginnt

Operand: kann sein: **AND, OR, XOR, NOT, PSET, PRESET**

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, wird das entsprechende Window gewählt. Fehlt dieser Parameter, bezieht sich die Anweisung auf das aktive Window. Die Koordinatenpaare (X1,Y1) und X2,Y2) definieren - bezogen auf die gültige **SCALE**-Anweisung - die diagonal liegenden Eckpunkte eines achsenparallelen Rechtsecks. Die in diesem Array gespeicherten Werte werden als Abbildung eines Bereichs der Bitmap (vgl. Kapitel 4.3.18) interpretiert und aus dem Array in den dem Rechteck entsprechenden Bereich der Bitmap übertragen.

Die Übertragung erfolgt in aufsteigender Reihenfolge der Arrayelemente in das Rechteck, beginnend von links oben nach rechts unten.
Der Parameter kann eines der Sprachelemente **AND, OR, XOR, NOT, PSET** oder **PRESET** sein.
Ist eines dieser Sprachelemente angegeben, wird auf dem durch **PUT %** angesprochenen Bereich der Bitmap für jeden Elementarpunkt die betreffende logische Operation durchgeführt.
Wird **PSET** angegeben, werden alle betroffenen Punkte gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Hintergrundfarbe rückgesetzt.

BEMERKUNGEN:

- Das angegebene Arrayelement muß so gewählt werden, daß es die Informationen über die Größe des im Array gespeicherten Bildes enthält. Es wird daher im allgemeinen dem im **GET %** angegebenen Arrayelement entsprechen.
- Ist der in **PUT %** angegebene Bereich größer als der im Array gespeicherte Bereich, wird zeilengerecht nur die Anzahl von Elementarpunkten übernommen, die ab 'Arrayelement' gespeichert sind.
- Ist der in **PUT %** angegebene Bereich kleiner als der im Array gespeicherte Bereich, werden die Grenzen des in **PUT %** angegebenen Bereiches eingehalten.
- Liegt das durch (X1,Y1) und (X2,Y2) definierte Rechteck nicht vollständig im angegebenen Window, wird nur der im Window liegende Bereich berücksichtigt.

- Die Anzahl der Arrayelemente, die für die Übertragung in das Window nötig sind, ergibt sich in etwa aus der Anzahl der im Rechteck liegenden Elementarpunkte, wobei in jedem Arrayelement 16 Bit gespeichert sind.

BEISPIELE:

```
10 CLEAR:SCALE 0,511,0,255:CURSOR POINT 0
20 OPTION BASE 0:DIM SPEICHER%(15000)
40 FOR I%=1 TO 7
50 CIRCLE(250,180-5*I%),40+5*I%,,1.2
60 CIRCLE(250,60+5*I%),40+5*I%
70 CIRCLE(295+5*I%,130),40+5*I%,,.8
80 CIRCLE(215-5*I%,130),40+5*I%
90 NEXT
100 GET (0,0)-(511,255),SPEICHER%(0)
110 FOR I%=1 TO 5000!:NEXT
120 CLS:COLOR 0,1:PRINT "Nach Abspeichern
im RAM durch GET% erfolgt jetzt PUT%"
130 PUT (0,0)-(511,255),SPEICHER%(0)
140 CALL "sp" 'nur graphikfähige Drucker
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **GET %, SCALE**
Funktion: **WINDOW**

FUNKTION: Festlegung bzw. Auswahl des Anfangswertes und der Folge von später aufzurufenden Zufallszahlen

FORMAT: **RANDOMIZE** [num.Ausdruck]

WIRKUNG: Der num.Ausdruck wird berechnet. Mit seiner Hilfe wird die erste später aufgerufene Zufallszahl **RND** eindeutig bestimmt, ebenso wie eine daraus resultierende Folge von Zufallszahlen.

BEMERKUNGEN:
- 'num.Ausdruck' muß einen Integer-Wert zwischen -32768 und 32767 ergeben, sonst erfolgt die Fehlermeldung "Overflow".
- Wird 'num.Ausdruck' weggelassen, fordert der M2Ø die Eingabe eines Wertes über Tastatur an. Meldung: "Random number seed (-32768 to 32767)?"
- Wird die **RANDOMIZE**-Anweisung nicht gegeben, wird bei jedem Aufruf des Programms der gleiche Anfangswert und die gleiche Folge von Zufallszahlen erzeugt, wenn die Funktion **RND** ohne Argument aufgerufen wird.
- Die **RANDOMIZE**-Anweisung muß in jedem Programm gegeben werden, das von der Standardfolge abweichende Zufallszahlen haben soll; allerdings kann dies auch durch Angabe eines Arguments beim Aufruf der **RND**-Funktion erreicht werden.

BEISPIELE:

```
10 RANDOMIZE VAL(RIGHT$(TIME$,2))
20 FOR I%=1 TO 25 ' 25 Zufallszahlen
30 PRINT RND : NEXT
```

VERWEISE: Kapitel 4.3.16
Funktion: **RND**

FUNKTION: Zuweisung von Werten aus dem internen File an Variablen; die Werte wurden vorher in einer internen Tabelle mit Hilfe von **DATA**-Anweisungen generiert

FORMAT: **READ** { num.Var. / Stringvar } [, { num.Var. / Stringvar }] ...

WIRKUNG: Die Werte der internen Tabelle werden der Reihe nach (von links nach rechts) den Variablen in der **READ**-Anweisung zugewiesen. Mit dem Lesen der Tabelle wird an der durch den Pointer bezeichneten Stelle begonnen (siehe Anweisungen **DATA** und **RESTORE**).
Nach jeder Zuweisung einer Konstanten an die entsprechende Variable wird der interne Pointer um eine Position (nächste Konstante) verschoben. Der Pointer kann mit der Anweisung **RESTORE** auf bestimmte Elemente der Tabelle zurückgesetzt werden.

BEMERKUNGEN:
- **READ** kann nur dann verwendet werden, wenn im Programm mindestens eine **DATA**-Anweisung vorkommt.
- In der **READ**-Anweisung werden die gelesenen Daten den Variablen in der Reihenfolge von links nach rechts zugewiesen. Das System verhält sich genau so, als stünde für jede Variable eine eigene **READ**-Anweisung.

- Stehen in der **DATA**-Anweisung mehr Konstante als mit der **READ**-Anweisung zu lesen sind, wird die erste nicht gelesene Konstante der **DATA**-Anweisung der ersten Variablen der nächsten **READ**-Anweisung zugewiesen.
- In einer **READ**-Anweisung dürfen nur soviele Werte verlangt werden, wie in der internen Tablle noch vorhanden sond. Ansonsten wird die Meldung "Out of DATA" gegeben.
- Die Variablen-Typen (String, numerisch) der **READ**-Anweisung müssen mit den Konstanten-Typen der **DATA**-Anweisung übereinstimmen. Anderenfalls wird die Meldung "Syntax error" gegeben. Allerdings können numerische **DATA**-Konstanten auch auf Stringvariablen zugelesen werden.
- Es können sich sowohl mehrere **READ**-Anweisungen auf eine **DATA**-Anweisung als auch eine **READ**-Anweisung auf mehrere **DATA**-Anweisungen beziehen.

BEISPIELE:

```
10 FOR I%=1 TO 10:READ T$(10):NEXT
620 RESTORE 9060
630 FOR I%=1 TO 2:READ PSEUDO$:NEXT:READ NWERT!
9050 DATA 2.555,57,frage1,3.45D27," Name,Vorn.  "
9060 DATA 345,1111,5.66,-1,text3
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **DATA, RESTORE**
Fehler-Code: 4

ANWEISUNG: **REM**
(remark)

FUNKTION: Einfügung von Erläuterungen in ein Programm

FORMAT: **REM** [Kommentar]
Kommentar: beliebiger Text

WIRKUNG: **REM** ist eine Anweisung, die nicht ausgeführt wird. Der Kommentar erscheint im Ausdruck (Listing), ist jedoch für die Programmausführung bedeutungslos.

BEMERKUNGEN:
- Es können auch Kommentare am Ende einer BASIC-Zeile angefügt werden. Dabei ist die Anweisung **REM** durch : von der letzten Anweisung zu trennen.
- Anstelle des Schlüsselwortes **REM** kann das Zeichen ' (Hochkomma) gesetzt werden. Dieses wird in Listings und beim Speichern des Programms nicht durch **REM** ersetzt, beeinflußt aber die Programmgeschwindigkeit negativ. Dabei kann sogar ein normalerweise nötiges Zeichen : vorher entfallen.
- Eine BASIC-Zeile mit einer **REM**-Anweisung kann als Sprungziel verwendet werden.
- Das Programm wird mit derjenigen BASIC-Zeile fortgesetzt, die der **REM**-Anweisung folgt.

- ACHTUNG:

  Befindet sich eine **REM**-Anweisung oder ein '(=Hochkomma) inmitten einer BASIC-Zeile mit durch : verbundenen Anweisungen, werden alle Zeichen, die **REM** oder ' in dieser Programmzeile folgen, als Kommentar interpretiert, also auch nachfolgende Anweisungen innerhalb der Zeile!

BEISPIELE:

```
0 DATE$="15.05.84"   :REM Programm erstellt am 15.5.84
1 PRINT DATE$   ' auch hier folgt eine Bemerkung
```

VERWEISE: Kapitel 4.3.1

FUNKTION: Umnumerierung der Zeilen eines im Arbeitsspeicher vorhandenen Programms

FORMAT: **RENUM** [neue Zeilennummer [,Schrittweite]], [alte Zeilennummer]

neue Zeilennr.: positive ganze Zahl; bestimmt die Zeilennummer, die der ersten umzunumerierenden Programmzeile zugeordnet wird

alte Zeilennr.: positive ganze Zahl, bestimmt die Zeilennummer, ab der umnumeriert wird

Schrittweite: positive ganze Zahl, welche die jeweilige Erhöhung der Numerierung bestimmt

WIRKUNG: Die im Arbeitsspeicher befindlichen Programmzeilen werden umnumeriert. Der ersten Programmzeile ab 'alte Zeilennr.' wird die mit 'neue Zeilennummer' definierte Zeilennummer zugeordnet. Jede folgende Zeile erhält eine, um 'Schrittweite' erhöhte, Zeilennr. Der Default-Wert für 'neue Zeilennummer' und 'Schrittweite' ist jeweils 10. Ist der Operand 'alte Zeilennummer' angegeben, beginnt die Umnumerierung nicht mit der ersten Programmzeile, sondern mit der durch 'alte Zeilennummer' definierten Zeile.

BEMERKUNGEN:

- Wird bei der Umnumerierung eine Zeilennummer größer als 65528 erzeugt, erfolgt die Meldung "Illegal function call".
- Ist 'neue Zeilennumer' kleiner als 'alte Zeilennumer' oder existieren Programmzeilen, die eine kleinere Zeilennumer als 'alte Zeilennumer' besitzen, erfolgt die Meldung "Illegal function call".
- Bei der Umnumerierung eines Programms modifiziert das System automatisch auch alle innerhalb von BASIC-Anweisungen vorkommenden Hinweise auf Zeilen (z.B. **GOTO, IF...THEN, GOSUB, ERL**).
  Ausnahmen: **IF** Zeilennr.=**ERL** (im Gegensatz zu IF **ERL**=Zeilennr.) und Parameter 'Startzeile' in der **CHAIN**-Anweisung mit Parameter **MERGE**.
- Es kann nur der gesamte restliche Programmteil ab 'alte Zeilennr.' bis zum physischen Programmende umnumeriert werden.
- Sollen nur Teile eines Programms umnumeriert werden, müssen die nicht benötigten Teile vorher mit **DELETE** gelöscht werden. (Vorher evtl. gesamtes Programm per **SAVE** auf Diskette sichern!) Das Programmteil (Modul) kann dann mit **SAVE** unter Verwendung des Parameters **A** als ASCII-File auf Diskette gespeichert, und später mit **MERGE** in andere Programme im Arbeitsspeicher eingebunden werden.

BEISPIELE:

```
RENUM 200,,2
RENUM 500,202,100
```

VERWEISE: Kapitel 3.3

ANWEISUNG: **RESTORE**

FUNKTION: Setzen des Pointers des internen Files auf das erste Element einer **DATA**-Anweisung

FORMAT: **RESTORE** [Zeilennummer]

WIRKUNG: Wird keine Zeilennummer angegeben, wird der Pointer des internen Files auf das erste Element der **DATA**-Anweisung mit der niedrigsten Zeilennummer gesetzt.
Wird eine Zeilennummer angegeben, wird der Pointer auf das erste Element derjenigen **DATA**-Anweisung gesetzt, die in der genannten Zeile steht.

BEMERKUNGEN: Siehe Anweisungen **DATA** und **READ**
- Der Pointer kann auch in eine **DATA**-Zeile positioniert werden, die noch nicht abgearbeitet ist.

BEISPIELE:

```
10 FOR I%=1 TO 10:READ T$(10):NEXT
620 RESTORE 9060
630 FOR I%=1 TO 2:READ PSEUDO$:NEXT:READ NWERT!
9050 DATA 2.555,57,frage1,3.45D27," Name,Vorn.  "
9060 DATA 345,1111,5.66,-1,text3
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **DATA, READ**

FUNKTION: Fortsetzung eines Programms nach einer Verzweigung in eine Fehlerbehandlungsroutine

FORMAT:

RESUME [{ **Ø** / **NEXT** / Zeilennr. }]

Zeilennr.: Zeilennr. einer Anweisung(sfolge); darf nicht in Form einer Variablen definiert sein

WIRKUNG: Stößt das System innerhalb einer Fehlerbehandlungsroutine (vgl. **ON ERROR GOTO**) auf die **RESUME**-Anweisung, wird aus der Fehlerbehandlungsroutine zurück zum Haputprogramm verzweigt.
Bei Angabe der **Ø** oder keines Parameters wird zu derjenigen Anweisung rückverzweigt, die den Sprung in die Fehlerbehandlungsroutine verursachte. Bei Angabe des Parameters **NEXT** wird zu derjenigen Anweisung verzweigt, die auf die Anweisung unmittelbar folgt, die den Sprung in die Fehlerbehandlungsroutine verursachte.
Bei Angabe von 'Zeilennr.' wird zu der mit 'Zeilennr.' bezeichneten Anweisungszeile verzweigt.

BEMERKUNGEN:
- Eine **RESUME**-Anweisung ohne vorherige Verzweigung in die Fehlerbehandlungsroutine führt zur Fehlermeldung "RESUME without error".

- Die reservierte Variable **ERR** enthält nach der **RESUME**-Anweisung den Wert Ø.
- Die reservierte Variable **ERL** bleibt von **RESUME** unberührt.

BEISPIELE:

```
0 'Flächenberechnung Dreieck
20 A$="_Seite a=###.##  _Seite b=###.##  _Seite c=###.##"+
"  Fläche=######.##"
30 ON ERROR GOTO 100
40 INPUT "Seite a,Seite b,Seite c ";A#,B#,C#
50 IF A#*B#*C#=0 THEN ERROR 200 'mind. 1 Wert = 0
60 S#=(A#+B#+C#)/2
70 F2#=S#*(S#-A#)*(S#-B#)*(S#-C#)
80 IF F2#<=0 THEN ERROR 210
85 LPRINT USING A$;A#,B#,C#,SQR(F2#):GOTO 40
100 IF ERR=200 THEN PRINT "a oder b oder c < 0":RESUME 40
110 IF ERR=210 THEN PRINT "kein Dreieck!":RESUME 40
120 ON ERROR GOTO 0:STOP   'sonstiger Fehler
```

VERWEISE: Kapitel 4.3.12
Anweisungen: **ON ERROR GOTO, ERROR**
reservierte Variable: **ERR, ERL**
Fehler-Code: 2Ø

FUNKTION: logisches Ende eines Unterprogramms, Rücksprung zu der unmittelbar auf das aufrufende **GOSUB** oder **ON...GOSUB** folgenden Anweisung

FORMAT: **RETURN**

WIRKUNG: Siehe Anweisungen **GOSUB** und **ON...GOSUB**

BEISPIELE:

```
40 U%=1:GOSUB 100 'Sprung ins 1. UP
45 GOSUB 300      'Sprung ins 3. UP
50 U%=2:GOSUB 500 'Sprung ins 2. UP
55 GOSUB 300      'Sprung ins 3. UP
90 END
100 PRINT "Sie befinden sich in UP";U%:RETURN
300 PRINT "zurück aus UP";U%:RETURN
500 PRINT "noch nicht":GOSUB 300:RETURN
```

VERWEISE: Kapitel 4.3.5 und 4.3.7
Anweisungen: **GOSUB, ON...GOSUB, CLEAR**
Fehler-Code: 3

FUNKTION: RIGHT$

ZWECK: Einem String wird von rechts kommend ein Teilstring entnommen

FORMAT: **RIGHT$(**Ausgangsstring, Anzahl Zeichen**)**

Ausgangsstring: Stringausdruck
Anzahl Zeichen, num. Ausdruck;
bestimmt die Anzahl Zeichen des Teilstrings

WIRKUNG: Die 'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet. Aus dem 'Ausgangsstring' wird von rechts kommend die berechnete Anzahl von Zeichen entnommen.

BEMERKUNGEN:
- Ist 'Anzahl Zeichen' größer als die Länge des 'Ausgangsstrings', wird der gesamte Ausgangsstring übergeben.
- Ist 'Anzahl Zeichen' = Ø, wird kein Zeichen entnommen und der Teilstring ist der Leerstring.
- 'Anzahl Zeichen' muß einen Wert zwischen Ø und 255 aufweisen.

BEISPIELE:
```
10 T$=TIME$:PRINT "Uhrzeit = ";T$;"***";
20 S$=RIGHT$(T$,2):PRINT "Sekunden: ";S$
30 PRINT VAL(S$)
RUN
Uhrzeit = 16:57:03***Sekunden: 03
 3
```

VERWEISE: Kapitel 4.3.4
Funktionen: **LEFT$, MID$**

FUNKTION: **RND**
(random number)

ZWECK: liefert eine Zufallszahl zwischen Ø und 1

FORMAT: **RND** [(num.Ausdr.)]

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit gerundet.
In Abhängigkeit von seinem Wert wird eine Zufallszahl zwischen Ø und 1 (einfach genau) als Ergebnis der Funktion entwickelt. Ist 'num.Ausdr.' größer als Ø, wird der nächste Wert einer Folge von Zufallszahlen bestimmt, deren erster Wert von 'num.Ausdr.' abhängt.
Ist 'num.Ausdr.' kleiner als Ø, wird immer wieder die gleiche vom 'numerischen Ausdruck' abhängige Zufallszahl erzeugt.
Ist 'num.Ausdr.'=Ø, wird stets Ø erzeugt.
Ist 'num.Ausdr.' nicht angegeben, wird die nächste Zahl der Standardfolge von Zufallszahlen erzeugt.

BEMERKUNGEN:
- Bei Weglassen von 'num.Ausdr.' kann die Standardfolge durch eine vorher erfolgte **RANDOMIZE**-Anweisung geändert werden.
- Die Zufallszahl ist von einfacher Genauigkeit.

BEISPIELE:

```
10 FOR I%=1 TO 2:FOR J%=1 TO 4
20 PRINT RND;:IF J%=4 THEN PRINT
30 NEXT I%:PRINT:NEXT J%
RUN
 .082254   .255843   .776169   .682012
 7.32367E-02   .377858   .956858   .372086
Ok
RUN
 .082254   .255843   .776169   .682012
 7.32367E-02   .377858   .956858   .372086
```

VERWEISE: Kapitel 4.3.16
Anweisung: **RANDOMIZE**

ANWEISUNG: RSET
(right set)

FUNKTION: Zuweisung eines rechtsbündigen, links eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable

FORMAT: **RSET** { Feldvar. / Stringvar. } =Stringausdr.

WIRKUNG: Der Stringausdruck wird berechnet. Das Ergebnis wird rechtsbündig in das durch 'Feldvariable' definierte Feld (vgl. **FIELD**-Anweisung) eines Random-File-Puffers gebracht, wobei Feldreste automatisch mit Blanks gefüllt werden.
Bei einer **RSET**-Zuweisung auf eine normale Stringvariable wird deren alter Inhalt rechtsbündig durch das Ergebnis von 'Stringausdruck' ersetzt. Diejenigen Zeichen von 'Stringvariable', die bei der rechtsbündigen Ersetzung nicht betroffen werden, werden durch Blanks ersetzt.

BEMERKUNGEN:
- Hat das Ergebnis von 'Stringausdruck' mehr Zeichen als 'Feldvar.' bzw. 'Stringvar.', werden die überschüssigen Zeichen rechts abgeschnitten.
- Bevor numerische Werte in den Random-File-Puffer gebracht werden, müssen sie - je nach Typ - mit Hilfe der Funktionen **MKI$**, **MKS$** oder **MKD$** in Strings konvertiert werden.

- Mit **MKI$, MKS$** oder **MKD$** gewonnene Werte sollten stets mit **RSET** auf die für sie vorgesehene Feldvariable zugewiesen werden.
- ACHTUNG: Ist die gewünschte Länge von 'Feldvar.' bzw. 'Stringvar.' nicht zuvor per **FIELD**-Anweisung oder über Zuweisung, z.B.
  Feldvar.**$**= **SPACE$(**Feldlänge**%)**
  festgelegt worden, werden nur soviele Zeichen aus 'Stringausdruck' auf 'Feldvar.' bzw. 'Stringvar.' zugewiesen, wie aktuell in 'Feldvar.' bzw. 'Stringvar.' vorhanden sind! Besonders beachten: Stringvariable, die noch keinen Inhalt zugewiesen bekommen haben, haben den Default-Wert Leerstring ("") mit Ø Zeichen Länge!

BEISPIELE:

```
70 OPEN "R",1,F$,2  ' nur Integer-Zahlen
75 FIELD 1,2 AS IZAHL$
80 INPUT "lfd. Nr.";N%:IF N%<=0 THEN CLOSE:END
81 INPUT "Wert ";Z%:RSET IZAHL$=MKI$(Z%)
85 PUT 1,N%:GOTO 80
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **FIELD, PUT, LSET**
Funktionen: **MKI$, MKS$, MKD$**

BEFEHL: **RUN**

FUNKTION: Start der Programmausführung

FORMAT: **RUN** [{Zeilennummer / Filename [,R]}]

Zeilennummer: Programmzeile, mit welcher die Verarbeitung beginnen soll; kann nicht Variable sein

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Sind keine Operanden angegeben, wird das sich im Arbeitsspeicher befindliche Programm, beginnend mit der ersten Programmzeile, abgearbeitet. Bei Angabe des Operanden 'Zeilennummer' beginnt die Ausführung des Programms mit der durch 'Zeilennummer' definierten Programmzeile. Die Ausführung des Befehls **RUN** löscht den gesamten Datenbereich und schließt alle offenen Datenfiles.
Ist der Operand 'Filename' angegeben, wird das Programm auf der entsprechenden Diskette gesucht, in den Arbeitsspeicher geladen und abgearbeitet. Der Operand **R** gibt an, daß alle bereits geöffneten Datenfiles offen bleiben sollen.

BEMERKUNGEN:

- Wird **RUN** mit 'Zeilennummer' angegeben, darf sich 'Zeilennumer' nicht auf eine Anweisung innerhalb einer **FOR-NEXT**-Schleife (Meldung: "NEXT without FOR"), innerhalb einer **WHILE-WEND**-Schleife (MELDUNG: "WEND without WHILE"), oder innerhalb eines Unterprogramms (Meldung: "RETURN without GOSUB") beziehen.

  ACHTUNG:

  bei **RUN** mit 'Zeilennumer' wird der datenbereich gelöscht; daraus folgt, daß das Programm ab 'Zeilennumer' für alle Variablen, die ohne vorherige Zuweisung aufgerufen werden, die Default-Werte ansetzt (num. Variablen: Ø; String-Variablen: "").

- Wird **RUN** mit 'Zeilennumer' angegeben und existiert die Zeilennumer nicht, erscheint die Fehler-Meldung "Undefined line number".
- **RUN** Zeilennumer hat die gleiche Funktion wie **GOTO** Zeilennummer, jedoch wird bei **GOTO** Zeilennummer der Datenbereich nicht gelöscht.
- **RUN** Filename,**R** hat die gleiche Bedeutung und Wirkung wie **LOAD** Filename,**R** (siehe auch dortige Bemerkung zu Variablen und Random-File-Puffer!)
- Die Programmausführung wird beendet, wenn

  1. die Anweisung **END** erkannt wird. Alle offenen Files werden geschlossen. Programmvariablen können angezeigt werden.

2. die Anweisung **STOP** erkannt wird, die Taste C gedrückt wird oder eine Fehlermeldung auftritt. Das System befindet sich dann im Direkt-Mode (Ausnahme: bei Syntax-Fehler). Offene Files bleiben offen, Programmvariablen können angezeigt und/oder geändert werden. Mit dem befehl **CONT** kann die Ausführung des Programms fortgesetzt werden.
3. die Taste S gedrückt wird, während Textausgaben am Bildschirm erfolgen. Das System befindet sich nicht im Direkt-Mode. Offene Files bleiben offen. Programmvariablen können nicht angezeigt werden. Durch Drücken irgendeiner Taste wird die Ausführung des Programms fortgesetzt.

BEISPIELE:

```
RUN
RUN "10:MODUL127.asc"
990 RUN "1:FOLGE1.prg",R
```

VERWEISE: Kapitel 3
Befehle: **LOAD, CONT**
Anweisungen: **END, GOTO, STOP, CHAIN**

FUNKTION: Abspeicherung eines BASIC-Programms oder eines mit Zeilennummern versehenen Textes (ASCII-Files) auf Diskette

FORMAT: **SAVE** Filename [, {A / P}]

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Das sich im Arbeitsspeicher befindliche Programm wird auf Diskette gespeichert. Das Programm wird in einer gepackten Binärform abgespeichert, wenn der Operand **A** nicht angegeben ist. Enthält der Operand 'Filename' keine Stationsbezeichnung, so wird das Programm auf die Diskette in der momentan aktiven (zuletzt angesprochenen) Station gespeichert. Bei der Abspeicherung kann dem Programm ein File-Password zugeordnet werden. Befindet sich auf der Diskette bereits ein File mit dem durch 'Filename' bestimmten Namen, wird dieses File durch das im Arbeitsspeicher befindliche Programm ersetzt, andernfalls wird automatisch Platz für das neue File auf Diskette angelegt. Ist einem bereits vorhandenen File ein File-Password zugeordnet, muß das File-Password bei **SAVE** mit angegeben werden.

Ist der Operand **A** angegeben, so wird das Programm im ASCII-Format auf der Diskette gespeichert. ASCII-Format benötigt mehr Platz auf der Diskette als die gepackte Binärform. Es gibt jedoch Befehle und Anweisungen, die ein File in ASCII-Format erfordern (z.B. **MERGE** und **CHAIN MERGE**). Der Operand **P** bestimmt, daß das Programm in gepackter Binärform abgespeichert und gegen folgende Befehle geschützt werden soll:

- **LIST**
- **EDIT**
- **SAVE** (ersetzen)

BEMERKUNGEN:

- Die Diskette darf nicht mit einem Schreibschutz versehen sein.
- Hat die Diskette ein Disketten-Password, ist dieses bei der Abspeicherung anzugeben. Dies kann entfallen, wenn das Disketten-Password schon einmal angegeben wurde und seitdem kein Diskettenwechsel und kein Neuladen des Betriebssystems erfolgt ist.
- Das System prüft, ob das abzuspeichernde Programm noch auf der Diskette Platz hat. Sollte dies nicht der Fall sein, so erscheint die Meldung "Disk full".
- Mit PCOS-Befehl **ss** wird festgelegt, wie groß die Anzahl zu reservierender Sektoren für ein abzuspeicherndes Programm sein soll. Ist diese Anzahl größer als die Anzahl freier Sektoren auf der Diskette, so erscheint bei dem Befehl **SAVE** ebenfalls die Meldung "Disk full".

- Die Operanden **A** und **P** können nicht gleichzeitig angegeben werden, d.h., ein ASCII-File kann nicht geschützt werden.

BEISPIELE:

```
SAVE "0:SACHKTEN.prg"
SAVE "0:SICH1.asc",A
SAVE "1:SACHKTEN.prg/K4711",P
```

VERWEISE: Kapitel 3 und 4.3.13

ZWECK: Festlegung des Koordinationssystems innerhalb eines Windows

FORMAT: **SCALE** [%Window-Nr.,] X-min, X-max, Y-min, Y-max

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 1 und 16 enthalten muß

X-min: numerische Ausdrücke, die Grenzen problembezogener Werte bilden

X-max:

Y-min:

Y-max:

WIRKUNG: Ist der Parameter % Window-Nr. angegeben, bezieht sich die Anweisung auf dieses Window. Fehlt der Parameter, bezieht sich die Anweisung auf das aktive Window. 'X-min' und 'X-max' stellen die Grenzwerte der Anwender-Koordinaten dar, die im angesprochenen Window gültig sind.
Dabei werden der linken unteren Ecke des Windows die Koordinaten (X-min/Y-min) zugewiesen. Die rechte obere Ecke des Windows erhält die Koordinaten (X-max/Y-max) zugeordnet:
Daraus ergibt sich:
X-min < X-max
Y-min < Y-max

Die positive Richtung der X-Achse ist somit der horizontalen Richtung von links nach rechts zugeordnet, die positive Richtung der Y-Achse entspricht der vertikalen Richtung von unten nach oben. Aus der Anzahl der Elementarpunkte, die in jeder Richtung des Windows enthalten sind, wird durch den gültigen Wertebereich der Abbildungsmaßstab für später folgende graphische Ausgaben bestimmt.

BEMERKUNGEN:

- Werden Koordinaten, die außerhalb des durch **SCALE** festgelegten Bereiches liegen, angesprochen, hat das für die meisten graphischen Anweisungen und Funktionen keine Wirkung. Für die imaginäre Darstellung außerhalb des Bereichs wird jedoch Rechenzeit benötigt (unnötigerweise).
- Ist X-max' kleiner oder gleich 'X-min' bzw. 'Y-max' kleiner oder gleich 'Y-min', wird "Division by zero" gemeldet und weitergearbeitet.
- **RUN**-Befehl oder **CHAIN**-Anweisung heben eine im Vorprogramm erfolgte **SCALE**-Anweisung nicht auf. Dies ist nur durch neue **SCALE**-Anweisung oder durch die **CLEAR**-Anweisung möglich.
- Wird ein Programm durch **CHAIN** angekettet, bleibt die **SCALE**-Definition des alten Programms für das neue gültig, es sei denn, dort wird eine neue **SCALE**-Anweisung verwendet.

- Die Default-Werte für **SCALE** betragen:
  bei 64x16-Zeichen-Format des Bildschirms (mit **ss** gesetzt!):
  X-min= Ø
  X-max= 511
  Y-min= Ø
  Y-max= 255

  bei 80x25-Zeichen-Format (mit **ss** gesetzt!):
  X-min: Ø
  X-max: 479
  Y-min: Ø
  Y-max: 255
- Für den Maßstab gilt:
  ((X-max - X-min)+1/ (Anzahl Elementarpunkte in X-Richtung)
  ergibt die Anzahl von problembezogenen Einheiten, die auf einen Elementarpunkt abgebildet werden; analog für die Y-Richtung.
- Da der Abstand der Elementarpunkte horizontal und vertikal nicht gleich ist, muß zur Erreichung eines gleichen Maßstabs in X und Y, bezogen auf den Gesamtbildschirm, gelten:
  dx : dy = 1.575 : 1 bei 64x16-Zeichen-Schirm
  dx : dy = 1.5Ø4 : 1 bei 80x25-Zeichen-Schirm.
  Diese Verhältnisse gelten nur, wenn das Bildschirmformat durch **ss** festgelegt wurde.

- Wird mit Hilfe der Funktion **WINDOW** der Gesamtschirm im Format geändert, wird zwar die dem Format entsprechende Zeichenfläche zugrundegelegt; die horizontale Aufteilung in Elementarpunkten (Anzahl) bleibt jedoch die in **ss** festgelegte. Deshalb empfiehlt sich, stets eine Skalierung vorzunehmen. Ist diese nicht in Anwenderkoordinaten, sollte das Format des Gesamtschirms über die **WINDOW**-Funktion definiert und danach folgende Skalierung gewählt werden:

Spaltenbreite	**SCALE**
8	Ø,511,Ø,255
6	Ø,479,Ø,255

BEISPIELE:

```
0 CLEAR:CALL "ss %n,,,,,1" '80x25-Format
10 SCALE 0,480,0,255
20 LINE(0,0)-(480,255),,B  ' Umrahmung
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **CLEAR**
Funktionen: **WINDOW, SCALEX, SCALEY**

ZWECK: gibt an, der wievielte Elementarpunkt des aktiven Windows in horizontaler Richtung durch eine anwenderbezogene X-Koordinate angesprochen wird

FORMAT: **SCALEX**(num.Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als X-Koordinate, bezogen auf das im aktiven Window gültige **SCALE**, interpretiert. Es wird ermittelt, der wievielte Elementarpunkt des aktiven Windows in X-Richtung, abzählend von Ø (=linker Window-Rand), dadurch angesprochen wird.

BEMERKUNGEN:
- Liegt das Argument außerhalb des durch **SCALE** festgelegten X-Intervalls, wird ein imaginärer Punkt ermittelt.
- 'num.Ausdruck' muß zwischen -32755 und 32755 liegen, sonst wird "Overflow" gemeldet.
- Eine X-Koordinate auf dem linken Window-Rand liefert den **SCALEX**-Wert Ø; der maximale Wert für den rechten Window-Rand ist abhängig vom Bildschirm-Format:

Bildschirm-Format	maximaler **SCALEX**-Wert (nur am Gesamtschirm!)
64 x 16	511
80 x 25	479

Die zuletzt gemachten Angaben beziehen sich auf den Gesamtschirm!

- Die Funktion ist besonders bei Aufruf des PCOS-Befehls **1a** in BASIC wichtig, da die Anfangsposition des auszugebenden Schriftzuges stets in Elementarpunkt-Positionen angzugeben ist.

BEISPIELE:

```
10 CLEAR:CALL "ss %n,,,,,0" '64x16-Format
20 SCALE 1960,1987,0,1000
30 PIXELX%=SCALEX(1965,500)
40 PIXELY%=SCALEY(1965,500)
50 CALL "1a"("1965/500",PIXELX%,PIXELY%,2,0)
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **SCALE**
Funktion: **SCALEY**
PCOS-Befehl: **1a**

FUNKTION: SCALEY

ZWECK: gibt an, der wievielte Elementarpunkt des aktiven Windows in vertikaler Richtung durch eine anwenderbezogene Y-Koordinate angesprochen wird

FORMAT: **SCALEY**(num.Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als Y-Koordinate, bezogen auf das im aktiven Window gültige **SCALE**, interpretiert. Es wird ermittelt, der wievielte Elementarpunkt des aktiven Windows in Y-Richtung, abzählend von Ø (= unterer Window-Rand), dadurch angesprochen wird.

BEMERKUNGEN:
- Liegt das Argument außerhalb des durch **SCALE** festgelegten Y-Intervalls, wird ein imaginärer Punkt ermittelt.
- 'num.Ausdruck' muß zwischen -32755 und 32755 liegen, sonst wird "Overflow" gemeldet.
- Eine Y-Koordinate auf dem unteren Window-Rand liefert den **SCALEY**-Wert Ø; der maximale Wert für den oberen Window-Rand ist, unabhängig vom Bildschirmformat, 255 (bezogen nur auf den Gesamtschirm!)
- Die Funktion ist besonders bei Aufruf des PCOS-Befehls **la** in BASIC wichtig, da die Anfangsposition des auszugebenen Schriftzuges stets in Elementarpunkt-Positionen anzugeben ist.

BEISPIELE:

```
10 CLEAR:CALL "ss %n,,,,,0" '64x16-Format
20 SCALE 1960,1987,0,1000
30 PIXELX%=SCALEX(1965,500)
40 PIXELY%=SCALEY(1965,500)
50 CALL "la"("1965/500",PIXELX%,PIXELY%,2,0)
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **SCALE**
Funktion: **SCALEX**
PCOS-Befehl: **la**

FUNKTION: **SGN**
(signum)

ZWECK: gibt Aufschluß über das Vorzeichen eines numerischen Wertes ("Signum-Funktion")

FORMAT: **SGN**(num.Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet.
Ist er positiv, liefert **SGN** als Ergebnis +1.
Ist er negativ, liefert **SGN** als Ergebnis -1.
Ist er genau Ø, liefert **SGN** als Ergebnis Ø.

BEISPIELE:

```
10 OPTION BASE 0:DIM A%(4)
20 A%(0)=-2:A%(1)=6:A%(3)=-4.7:A%(4)=0
30 FOR I%=0 TO 4:PRINT SGN(A%(I%));:NEXT
RUN
-1  1  0 -1  0
```

VERWEISE: Kapitel 4.3.16

FUNKTION: **SIN**

ZWECK: liefert den SInus eines Winkels im Bogenmaß

FORMAT: **SIN**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als Bogenmaß interpretiert und daraus wird der Sinus berechnet.

BEMERKUNGEN:
- Das Argument ist im Bogenmaß anzugeben. Umrechnungen müßten über Unterprogramm oder ein selbstdefinierte Funktion (vgl. **DEF FN**) erfolgen.
- Das Argument muß im Intervall $-65535.998 \leq x \leq 65535.998$ liegen, sonst erfolgt die Fehlermeldung "Overflow".
- Der Arcussinus läßt sich berechnen mit Hilfe der Formel:
  ARCSIN(X)=**ATN**(X/**SQR**(-X*X+1))

BEISPIELE:

```
1 PI!=3.14159
10 WIN!=40:MASK$="_SIN von ### Grad = #.######"
20 PRINT USING MASK$;WIN!,SIN(WIN!*PI!/180)
RUN
SIN von  40 Grad =  .642787
```

VERWEISE: Kapitel 4.3.16

FUNKTION: SPACE$

ZWECK: liefert einen String aus Blanks

FORMAT: **SPACE$(num.Ausdr.)**

WIRKUNG: Der numerische Ausdruck wird berechnet und zu einer Integer-Zahl kaufmännisch gerundet. Als Ergebnis liefert die Funktion einen String aus soviel Blanks wie durch die Integer-Zahl angegeben.

BEMERKUNGEN:
- Der 'numerische Ausdruck' muß zwischen Ø und 255 liegen.

BEISPIELE:

```
10 CLS:FL%=11
20 FELD$=SPACE$(FL%) 'aktuelle Länge jetzt 11
30 PRINT ")";FELD$;"(":PRINT "123456"
50 RSET  FELD$="123456":PRINT FELD$
RUN
)           (123456
)     123456(
```

VERWEISE: Kapitel 4.3.4 und 4.3.9
Funktion: **SPC**

FUNKTION: SPC

ZWECK: Ausgabe einer bestimmten Anzahl von Leerzeichen (Blanks) auf dem Bildschirm oder auf dem Drucker

FORMAT: **SPC**(num.Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet. Es werden ab der aktuellen Spalten-Position des Bildschirm-Cursors oder des Druckkopfes soviele Leerzeichen (Blanks) ausgegeben, wie der numerische Ausdruck angibt.

BEMERKUNGEN:
- Die **SPC**-Funktion darf nur innerhalb der Liste von Ausgabe-Elementen bei **PRINT**- oder **LPRINT**-Anweisungen eingesetzt werden.
- Das Argument muß zwischen Ø und 255 liegen.
- Es wird höchstens eine Bildschirmzeile mit Blanks erzeugt!

BEISPIELE:

```
10 A$="*":B$="*"
20 PRINT A$;SPC(5);B$
RUN
*     *
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2
Anweisungen: **PRINT, LPRINT**

FUNKTION: SQR
(square root)

ZWECK: liefert die positive Quadratwurzel eines numerischen Wertes

FORMAT: **SQR(**num.Ausdr.**)**

WIRKUNG: Der numerische Ausdruck wird berechnet. Daraus wird die positive Quadratwurzel gezogen.

BEMERKUNGEN:
- Das Argument muß größer oder gleich Ø sein, sonst wird die Fehlermeldung "Illegal function call" gegeben.
- Das Ergebnis hat die Genauigkeit des Arguments.

BEISPIELE:

```
10 PRINT SQR(23.465),23.465^(1/2),'allgem. Wurzel
RUN
 4.84407   4.84407
```

VERWEISE: Kapitel 4.3.16

ANWEISUNG: **STOP**

FUNKTION: Unterbrechen der Programmausführung und Übergang in den Direkt-Mode

FORMAT: **STOP**

WIRKUNG: Der Programmablauf wird unterbrochen. Die Meldung "Break in line nnnnn" wird ausgegeben, worin nnnnn die Nummer der die **STOP**-Anweisung enthaltenden Zeile angibt.
Das System geht in den Direkt-Mode (siehe Kapitel 3) über.

BEMERKUNGEN:
- Die **STOP**-Anweisung bewirkt kein Schließen der Files.
- Mit dem **CONT**-Befehl kann die Programmausführung fortgesetzt werden.
  ACHTUNG: Nach Übergang in den Direkt-Mode und Änderung oder Neuaufnahme einer Programmzeile ist der gesamte Datenbereich gelöscht!

BEISPIELE:

```
645 A%=81:PRINT "1.":STOP:PRINT "**"
RUN
1.
Break in line 645
Ok
PRINT A%;:CONT
 81 **
Ok
```

VERWEISE: Kapitel 3 und 4.3.5

FUNKTION: STR$

ZWECK: Bildung eines Strings aus einem numerischen Ausdruck

FORMAT: **STR$(**num.Ausdruck**)**

WIRKUNG: Der numerische Wert wird berechnet und in einen String umgewandelt. Im ersten Zeichen des Strings ist das Vorzeichen des numerischen Ausdrucks ausgedrückt Blank für positiv oder Ø "-" für negativ).

BEMERKUNGEN:
- Die Funktion **STR$** ist die logische Umkehrung der Funktion **VAL**.
- Bei Gleitkommastellung wird das Vorzeichen des Exponenten mit erzeugt. Bei + wird + erzeugt. Der Exponent wird mit zwei Stellen angegeben, wenn er eine oder zwei Stellen lang ist.
- ACHTUNG: Im Gegensatz zur Darstellung der numerischen Werte im Standardformat (vgl. **PRINT**-Anweisung und Kapitel 4.3.9) wird bei der Umwandlung eines numerischen Werts mit **STR$** nach der letzten signifikanten Stelle des numerischen Werts kein Blank erzeugt!
- **STR$(Ø.73ØØ1ØØ)** liefert den String " .73ØØ1".

BEISPIELE:

```
10 PRINT "*";STR$(123.45);"*";STR$(-123.45);"*"
RUN
* 123.45*-123.45*
```

VERWEISE: Kapitel 4.3.4.3
Funktion: **VAL**

FUNKTION: STRING$

ZWECK: Erzeugung eines Strings mit vorgegebener Länge aus einem vorgegebenem Zeichen

FORMAT: **STRING$**(Stringlänge, {dez. Wert / Stringausdr.})

Stringlänge: num. Ausdruck;
gibt die Länge des Ergebnisstrings an

dez. Wert: num. Ausdruck;
gibt den ISO-Code des Zeichens an

Stringausdruck; das erste Zeichen des Ergebnisses des Stringausdrucks wird übernommen

WIRKUNG: 'Stringlänge' und 'dez. Wert' werden berechnet und zur Ganzzahligkeit kaufmännisch gerundet. Je nach Angabe wird das betreffende Zeichen gemäß der ISO-Code-Tabelle ermittelt. (Bei Angabe von 'Stringausdruck' wird aus diesem das erste Zeichen ermittelt.) Der Ergebnisstring wird durch Aneinanderreihen des ermittelten Zeichens gebildet. Das Zeichen tritt in der durch 'Stringlänge' definierten Anzahl auf.

BEMERKUNGEN:
- 'Stringlänge' muß einen Wert zwischen Ø und 255 aufweisen.
- Ist 'Stringlänge' = Ø, ist der Ergebnisstring der Leerstring.

- 'dez. Wert' muß einen Wert zwischen Ø und 255 aufweisen.

BEISPIELE:

```
20 PRINT STRING$(8,"-")
35 PRINT STRING$(5,34)
RUN
--------
"""""
```

VERWEISE: Kapitel 4.3.4.2

FUNKTION: Vertauschen des Inhalts zweier Variablen vom gleichen Typ

FORMAT: **SWAP** {Stringvar.1 , Stringvar.2 / num.Var.1 , num.Var.2}

WIRKUNG: Der Inhalt von 'Stringvar.2', bzw. 'num.Var.2' wird auf 'Stringvar.1' bzw. 'num.Var.1' zugewiesen und der vorherige Inhalt von 'Stringvar.1' bzw. 'num.Var.1' auf 'Stringvar.2' bzw. 'num.Var.2'.

BEMERKUNGEN:
- Die beiden Variablen müssen vom gleichen Typ sein (Integer, einfache Genauigkeit, doppelte Genauigkeit oder String), sonst wird die Fehlermeldung "Type mismatch" gegeben.

BEISPIELE:

```
10 A1!=2:A2!=5:B$(1)=" Tag ":B$(2)=" Guten "
20 PRINT A1!;A2!;B$(1);B$(2)
30 SWAP A1!,A2!:SWAP B$(1),B$(2)
40 PRINT A1!;A2!;B$(1);B$(2)
RUN
 2  5  Tag  Guten
 5  2  Guten  Tag
```

VERWEISE: Kapitel 4.3.2

BEFEHL: **SYSTEM**

FUNKTION: Aufruf der PCOS-Ebene

FORMAT: **SYSTEM**

WIRKUNG: Das System geht von der BASIC-Ebene zur PCOS-Ebene über. Alle offenen Datenfiles werden geschlossen; der Inhalt des BASIC-Arbeitsspeichers wird gelöscht.

BEMERKUNGEN:
- Im Gegensatz zur BASIC-Ebene, wo PCOS-Befehle mit den Anweisungen **EXEC** bzw. **CALL** aufgerufen werden und in Form von Stringkonstanten in Anführungszeichen (") eingeschlossen anzugeben sind, sind die PCOS-Befehle in der PCOS-Ebene ohne " einzugeben.

BEISPIELE:

```
SYSTEM

990 PRINT "Taste drücken, dann   sb 3,,,100   eingeben;"
991 PRINT "danach   ba 1:MENU.prg   eingeben."
995 K$=INPUT$(1):SYSTEM
```

VERWEISE: Kapitel 3.1

FUNKTION: TAB
(tabulation)

ZWECK: Positionieren auf eine gewünschte Spalte am Bildschirm oder Drucker

FORMAT: **TAB**(Druckspalte)

Druckspalte: num. Ausdruck

WIRKUNG: Der numerische Ausdruck wird berechnet und kaufmännisch zu einer Integer-Zahl gerundet. Der Bildschirm-Cursor bzw. der Druckkopf wird an die durch ihn definierte Spaltenposition bewegt.

BEMERKUNGEN:
- **TAB** darf nur in **PRINT**- oder **LPRINT**-Anweisungen innerhalb der Liste von Ausgabeelementen eingesetzt werden.
- Bei **PRINT USING** bzw. **LPRINT USING**-Anweisungen kann **TAB** vor dem **USING** gesetzt werden.
- Die Position 'Druckspalte' wird auch festgehalten, wenn auf **TAB** kein ; folgt.
- 'Druckspalte' muß zwischen 1 und 255 liegen.
- Liegt die errechnete Spalten-Position links von der aktuellen Position des Bildschirm-Cursors bzw. des Druckkopfes, wird die Position in der nächsten Zeile angesteuert.

BEISPIELE:

```
10 FOR I!=.1 TO 1.2 STEP .05
20 PRINT TAB(TAN(I!)*20);"*";
30 NEXT
RUN
 ******** *** ** * * * * * *  *  *   *   *     *     *
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2

Anweisungen: **PRINT, LPRINT, WIDTH, PRINT USING, LPRINT USING**

FUNKTION: TAN

ZWECK: liefert den Tangens eines Winkels im Bogenmaß

FORMAT: **TAN**(num.Ausdr.)

FORMAT: Der numerische Ausdruck wird berechnet. Er wird als Bogenmaß interpretiert und daraus der Tangens ermittelt.

BEMERKUNGEN:
- Das Argument muß im Intervall $-65535.998 \leq x \leq 65534.428$ liegen, sonst erfolgt "Overflow".
- **TAN** ist die Umkehrfunktion von **ATN**.
- Das Argument ist im Bogenmaß anzugeben. Eine benötigte Umrechnung müßte in einem Unterprogramm oder einer selbstdefinierten Funktion (vgl. **DEF FN**) erfolgen.

BEISPIELE:

```
10 PI!=3.1415# NG!=50
20 T!=TAN(NG!*PI!/200)
30 PRINT "TANGENS von";NG!;"Neugrad =";T!
```

VERWEISE: Kapitel 4.3.16
Funktion: **ATN**

SPEZIELLE VARIABLE: **TIME$**

FUNKTION: von der Echtzeituhr verwaltete Uhrzeit

FORMAT: **TIME$**

WIRKUNG: Die Variable **TIME$** enthält die aktuelle Uhrzeit in der Form:

HH/MM/SS

wobei HH die Stunde, MM die Minuten und SS die Sekunden bedeutet.
Der Variablen **TIME$** können Werte zugewiesen werden, die eine gültige Uhrzeit darstellen.
Gültig ist:

ØØ≤HH≤23
ØØ≤MM≤59
ØØ≤SS≤59

und ein beliebiges druckbares Trennzeichen, das keine Ziffer sein darf.
Eine Zuweisung in diesem Format bewirkt eine Übernahme vom System als gültige Uhrzeit.
Eine Zuweisung eines Strings, der nicht diesem Format entspricht, bewirkt keine Veränderung der gültigen Uhrzeit.

BEMERKUNGEN:
- Die Variable **TIME$** ist auch ohne Wertzuweisung definiert, da der Wert vom PCOS-Befehl ss her ständig zur Verfügung steht.

- Die Uhrzeit wird korrekt mitgeführt, solange nicht PCOS über einen physischen RESET (z.B. Ausschalten) geladen wird. RESET (logischer RESET) hat keinen Einfluß auf **TIME$**.
- Bei Erreichen der Zeit ØØ/ØØ/ØØ wird die reservierte Variable **DATE$** entsprechend dem neuen Datum verändert.
- **TIME$** darf in allen Anweisungen verwendet werden, die keine Wertänderung bewirken. Die Wertänderung ist nur über die Zuweisungs-Anweisung **LET...=** möglich; das Schlüsselwort **LET** muß allerdings weggelassen werden.

BEISPIELE:

```
10 LINE INPUT "Datum ",D$
20 LINE INPUT "Zeit ";T$
30 C$="ss "+D$+","+T$:EXEC C$
40 ' oder:  DATE$=D$:TIME$=T$
60 PRINT DATE$,TIME$ ' nur bei zulässigem
       Format in D$ und T$ verändert!
```

VERWEISE: Kapitel 4.3.11
spezielle Variable: **DATE$**

BEFEHL: **TROFF**
(trace off)

FUNKTION: Aufhebung der Wirkung von **TRON**

FORMAT: **TROFF**

WIRKUNG: Die Nummern der ausgeführten Programmzeilen werden nicht mehr gedruckt. (Die Gültigkeit von **TRON** wird aufgehoben.)

BEISPIELE:

```
10 TRON
20 PRINT 20;"****"
30 PRINT 30;"****"
40 TROFF
```

VERWEISE: Befehl: **TRON**

BEFEHL: **TRON**
(trace on)

FUNKTION: Ausdruck der Zeilennummern während des Programmablaufs

FORMAT: **TRON**

WIRKUNG: Die Zeilennummer der jeweils ausgeführten Anweisung wird ausgedruckt.
Die Zeilennummern erscheinen in "Ä" und "Ü" eingeschlossen.

BEMERKUNGEN:
- Der **TRON**-Befehl wird durch den **TROFF**-Befehl, **TROFF** im Direkt-Mode, oder den Befehl **NEW** aufgehoben.

BEISPIELE:

```
10 TRON
20 PRINT 20;"****"
30 PRINT 30;"****"
40 TROFF
```

VERWEISE: Befehl: **TROFF**

FUNKTION: Bestimmung des Formats, in welchem bestimmte auszugebende Größen (Zahlen und/oder Strings) dargestellt werden

FORMAT: Sprachelement [TAB(num.Ausdruck) [;]]
USING Formatstring; Liste von Ausdrücken [{;,}]

Folgende Sprachelemente sind zulässig:
PRINT, PRINT#, LPRINT

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: 'Formatstring' wird berechnet. Er besteht aus einer Folge von Formatfeldern. Die einzelnen Ausdrücke von 'Liste von Ausdrücken' werden berechnet und in dem Format dargestellt, welches im 'Formatstring' für den entsprechenden Ausdruck angegeben ist. Jedem Ausdruck in 'Liste von Ausdrücken' wird ein Formatfeld (von links beginnend) zugeordnet. Zulässige Bestandteile von 'Formatstring' sind:

a) für numerische Daten:

1. #
2. .
3. +
4. -
5. ^^^
6. **
7. $$
8. **$
9. ,

b) für Strings:

10. Ö Ö (im deutschen Zeichensatz)
11. &
12. !

c) sonstige Zeichen:

13. _ (Unterstreichungszeichen)
14. Folge von Zeichen aller Art (außer 1-13)

Zu 1.:

Ein Formatfeld für ganzzahlige Größen besteht aus einer Folge von Symbolen (#### ...). Jedes #-Zeichen steht als Platzhalter für eine Ziffer. Für das Vorzeichen sollte ein #(zur Trennung) vorgesehen werden (kann bei positiven Werten entfallen). Die Zahl wird rechtsbündig ins Formatfeld übertragen. Bei nicht ganzzahligen Größen wird der kaufmännisch auf Ganzzahligkeit gerundete Wert dargestellt. Sind in einem Formatfeld mehr Symbole als darstellbare Zahlen vorhanden, wird das Formatfeld links mit Blanks aufgefüllt.

z.B.: **PRINT USING"### ";**12,-15,10.99

12 -15 11 (Das Blank wird am Ende mit ausgegeben!)

Zu 2.:

Ein Formatfeld für Dezimalzahlen besteht aus einer Folge von #-Zeichen und einem Dezimalpunkt. Der Dezimalpunkt kann an beliebiger Stelle des Formatfeldes stehen. Er bestimmt somit, mit wieviel Vor- und Nachkommastellen die Dezimalzahl dargestellt wird. Sind für Bruchteile der Zahl nicht genügend Symbole vorgesehen, wird die Zahl auf die letzte darstellbare Stelle kaufmännisch gerundet. Hat die Zahl weniger Nachkommastellen als im Formatfeld vorgesehen, wird bei den nicht benötigten #-Zeichen mit Ø aufgefüllt. Bei Dezimalzahlen ohne Vorkommastellen werden nur der Dezimalpunkt und die Nachkommastellen dargestellt (keine Ø vor dem Dezimalpunkt!),

z.B.: **PRINT USING"###.##";** 17.238

17.24

Zu 3.:

Ein +-Zeichen am Anfang des Formatfeldes bewirkt die Darstellung des Vorzeichens (positiv oder negativ) direkt vor der ersten signifikanten Ziffer. Ist das +-Zeichen am Ende des Formatfeldes, wird das Vorzeichen unmittelbar hinter der Zahl dargestellt.

z.B.: **PRINT USING"+###.## ";**12.56,-13.47

+12.56 -13.47

PRINT USING"###.##+ ";12.56,-13.47

12.56+ 13.47-

Zu 4.:

Ein --Zeichen am Ende des Formatfeldes bewirkt die Darstellung des negativen Vorzeichens unmittelbar hinter der Zahl. Positive Vorzeichen werden nicht dargestellt. Das --Zeichen kann nur am Ende des Formatfeldes stehen.

z.B.: **PRINT USING"##.##- ";-1.25,2.87**

1.25- 2.87

Zu 5.:

Bei numerischen Größen in einem Formatfeld für Zahlen in Exponentialdarstellung entspricht jeder Ziffer der Mantisse ein #-Zeichen (siehe 2.). Die vier ^^^^-Zeichen stehen am Ende des Formatfeldes. Sie sind Platzhalter für die Darstellung von E (Exponent) oder D, dem Vorzeichen und 2 bzw. 3 Ziffern, die den Exponenten zur Basis 10 bilden.
Der Dezimalpunkt kann an beliebiger Stelle des #-Feldes stehen. Die signifikanten Ziffern werden im numerischen Formatfeld rechtsbündig dargestellt. Aus dieser Darstellung und der festgelegten Position des Dezimalpunktes wird der entsprechende Exponent berechnet. Fehlt die Angabe eines der Zeichen + oder - (siehe 3. und 4.), wird ein #-Zeichen links vom Dezimalpunkt für das Vorzeichen verwendet (Blank oder -).

z.B.: **PRINT USING"###.##^^^^ ";234.45,7635.23,1.45**

23.44E+Ø1 76.35E+Ø2 14.5ØE-Ø1

Zu 6.:

Ist am Anfang des Formatfeldes ** angegeben, werden alle nicht benötigten #-Zeichen (siehe 1.) mit * dargestellt. ** wirkt ansonsten wie bei einer Formatierung mit zwei #-Zeichen.

z.B.: **PRINT USING"**#.# ";**12.39,-Ø.9,765.1

*12.4 **-.9765.1

Zu 7.:

Ist am Anfang des Formatfeldes **$$** angegeben, wird vor der ersten signifikanten Ziffer ein **$**-Zeichen dargestellt. Dieses kann die Funktion eines #-Zeichens übernehmen. Negative Zahlen in Verbindung mit **$$** können nur dann dargestellt werden, wenn das --Zeichen (siehe 4.) am Ende des Formatfeldes angegeben ist. Bei der Exponentialdarstellung kann für negative Zahlen kein **$$** angegeben werden.

z.B.: **PRINT USING"$$###.##";**456.78

$456.78

Zu 8.:

Steht am Anfang des Formatfeldes ****$**, bewirkt dies eine Verbindung der Punkte 7 und 8. Führende Blanks werden mit * aufgefüllt und vor der ersten signifikanten Ziffer steht ein **$**-Zeichen. ****$** kann die Funktion von zwei #-Zeichen übernehmen.

z.B.: **PRINT USING"**$#.# ";**123.5,1.2

$123.5 **$1.2

Zu 9.:

Befindet sich in einem Formatfeld für Dezimalzahlen irgendwo links vom Dezimalpunkt ein Komma (dies reicht bereits aus), werden die Vorkommastellen (von rechts kommend) zu Dreiergruppen zusammengefaßt dargestellt. Die Dreiergruppen werden durch Komma getrennt. Das erste Komma wirkt wie ein #-Zeichen. Für eventuelle weiter auftretende Kommas müssen # oder andere Platzhalter vorgesehen werden. Bei der Exponentialdarstellung hat das Komma keine Wirkung.

z.B.: **PRINT USING"#######,.##";**1234567.89
1,234,567.89

Zu 10.:

ÖÖ definiert ein Formatfeld für die Darstellung von Strings. (Im deutschen Zeichensatz steht Ö für ****). Jedes der beiden **Ö** steht als Platzhalter für ein Zeichen. Blanks zwischen den **Ö** dienen ebenso als Platzhalter. Anzahl Blanks +2 gibt an, wieviel Zeichen des Strings dargestellt werden. Der String wird linksbündig in das Formatfeld übertragen. Ist der String länger als das Formatfeld, werden die nicht zu übertragenden Zeichen abgeschnitten (keine Fehlermeldung). Ist der String kürzer, wird das Formatfeld rechts mit Blanks aufgefüllt.

z.B.: **PRINT USING"ÖÖ";**"AUTO"

AU

PRINT USING"Ö Ö"; "AUTO"

AUT

PRINT USING "Ö Ö";"AUTO"

AUTO

Zu 11.:

Das Zeichen **&** definiert ein Formatfeld von variabler Länge. Der String wird in seiner aktuellen Länge dargestellt.

z.B.: **PRINT USING "&";**"COMPUTER"

COMPUTER

PRINT USING "&";"OS"

OS

Zu 12.:

Das Zeichen ! definiert ein Formatfeld für ein einzelnes Zeichen. Nur das erste Zeichen des auszugebenden Strings wird dargestellt.

z.B.: **PRINT USING "!";**"OLIVETTI"

O

Zu 13.:

Das Unterstreichungszeichen (_) bewirkt eine Darstellung des ersten folgenden Format-Zeichens. Das nachfolgende Zeichen (eines der Zeichen zu 1-13) wird somit nicht als Bestandteil des Formats angesehen.

z.B.: **PRINT USING"_!##.##__";**12.49

!12.49_

PRINT USING"_##.##";2.76

#2.76

Zu 14.:

Eine beliebige Folge von Zeichen aller Art (außer 1-13) können dargestellt werden (z.B. als erläuternde Texte innerhalb von 'Formatstring'). Um solche Zeichen darstellen zu können, muß jedoch 'Formatstring' eines der unter 1.-12 definierten Formatfelder enthalten.

z.B.: **PRINT USING** "$$#.## DOLLAR";12.5
$12.5Ø DOLLAR

Steht am Ende der Anweisung ein ; erfolgt die nächste Ausgabe an die Folgeposition (vgl. Wirkung von ; bei **PRINT**). Steht am Ende der Anweisung ein ,, so wird das , nicht berücksichtigt, d.h. es wird nicht so tabuliert wie bei Verwendung von , in der Anweisung **PRINT**.

Zur Wirkung des Sprachelements **TAB** vgl. die Funktion **TAB**.

BEMERKUNGEN:

- Ist das Formatfeld vom Typ String und der zu formatierende Ausdruck numerisch, erfolgt die Meldung "Type mismatch".
- Ist das Formatfeld vom Typ numerisch und der zu formatierende Ausdruck String, wird keine Fehlermeldung gegeben, sondern die Adresse des Strings im BASIC-Arbeitsspeicher (s. Funktion **VARPTR**) ausgegeben.

- Reicht für einen numerischen Wert das vorgegebene Formatfeld nicht aus (Masken-Overflow), wird der Wert unformatiert mit einem führenden %-Zeichen ausgegeben. Die dahinter folgenden Werte werden davon nicht betroffen. Masken-Overflow tritt nur auf bei numerischen Werten, und zwar:
  a) wenn der ganzzahlige Anteil inklusive negativem Vorzeichen den dafür vorgesehenen Teil des Formatfeldes überschreitet
     oder
  b) wenn der Wert zur Beibehaltung des Signifikanzniveaus in Exponentialdarstellung ausgegeben werden müßte, jedoch diese Darstellung nicht vorgesehen ist.

  Bei Strings kann kein Masken-Overflow auftreten; es ist jedoch möglich, daß überschüssige Zeichen einfach rechts abgeschnitten werden (ohne Fehlermeldung).
- Da die Möglichkeit besteht, daß numerische Werte bei der Rundung der Nachkommastellen eine weitere Vorkommastelle erhalten, kann dabei ein Masken-Overflow auftreten.

  z.B.: **PRINT USING "#.##";**A!

  Hat A! den Inhalt 9.997, tritt Masken-Overflow auf.
- Für numerische Formatfelder können nicht mehr als 23 #-Zeichen benützt werden, sonst wird "Illegal function call" gemeldet.

- Eine Maskierung von numerischen Werten mit führenden Nullen ist über **USING** nicht möglich. Es muß auf die Stringverarbeitung zurückgegriffen werden (vgl. Kapitel 4.3.4).
- In einer Anweisung, die das Sprachelement **USING** verwendet, kann die Ausgabe von mehreren zu formatierenden Ausgabeelementen vorgeschrieben werden, ggf. auch mit einer geringeren Anzahl von Formatfeldern:
- Für die Anzahl der Formatfelder im Vergleich zur Anzahl der zu formatierenden Ausdrücke sind 3 Fälle zu unterscheiden:
  a) Die Anzahl der Formatfelder stimmt mit der Anzahl zu formatierender Ausdrücke überein.
  b) Die Anzahl der Formatfelder ist größer als die Anzahl der zu formatierenden Ausdrücke: Die am Ende nicht benötigten Formatfelder werden nicht berücksichtigt.
  c) Die Anzahl der Formatfelder ist geringer als die Anzahl der zu formatierenden Ausdrücke: Die Formatfelder werden zuerst von links nach rechts voll verwertet. Danach wird erneut ab dem ersten angegebenen Formatfeld weitergearbeitet.
- 'Formatstring' kann aus Stringkonstanten, Stringvariablen und/oder Stringoperationen gebildet werden.

- Ist an Bildschirm oder Drucker das Zeilenende erreicht (z.B. auch wegen einer **WIDTH**-Festlegung), die Liste der zu formatierenden Ausgabeelemente aber noch nicht zu Ende, wird in der nächsten Zeile weiter ausgegeben.
- Wird ein einfach genauer Wert mit **USING** maskiert, geschieht die Rundung bereits von der siebten auf die sechste signifikante Stelle.
- Das Ersetzen des Dezimalpunkts durch Komma (deutsche Zahlendarstellung) kann nur über Stringverarbeitung erreicht werden.
- Das Ergebnis einer Maskierung kann nur ausgegeben, nicht aber in einer Stringvariablen gespeichert werden.

BEISPIELE:

```
16 WIDTH LPRINT 64:Z$="1234567890":Z$=Z$+Z$+Z$+Z$+Z$+Z$+Z$+"1234"
20 A#=123.456:B#=-123.456:C#=.456:D#=1.575235D+22
30 A$="Alpha-Feld 1 ":B$="2. Alpha-Feld":LPRINT Z$
40 LPRINT USING "######.#### = A_# und ######.#### = B_#";A#;B#
60 LPRINT Z$:LPRINT USING "###.# = A_$ und ###.# = B_#";A#;B#
70 LPRINT Z$
80 LPRINT USING "####,###.##(D_#);######.######^^^^(D_#)";D#;D#;
100 LPRINT USING"+#.###### = C_# und ö          ö = B_$";C#;B$
101 INPUT "Wieviele * sollen eingefügt werden? ",NZ
102 IF NZ<0 OR NZ>35 THEN 101
103 LPRINT USING "-----&-----";MID$(Z$,6,54)
110 LPRINT USING "1234566          öÖ1&Das vorige Feld ist variabel lang -- hie
r: ## Zeichen";A$;STRING$(NZ,"*");NZ
```

```
1 PRINT "Artikelnummer (5 Zeichen), Bezeichnung (bis zu 25";
2 PRINT "Zeichen), Lagernummer (1-10)"
3 INPUT "",ART$,BEZ$,LNRZ
10 PRINT "Bitte Maske eingeben ":LINE INPUT MASKE$
20 REM Prüfung entfällt hier
30 LPRINT MASKE$
40 LPRINT ART$;"/";BEZ$;"/";LNRZ
50 LPRINT USING MASKE$;ART$;BEZ$;LNRZ
99 END
```

VERWEISE: Kapitel 4.3.9 (und 4.3.4 für komplizierte Druckaufbereitungen über Stringverarbeitung)
Anweisungen: **PRINT, PRINT#, LPRINT, WIDTH**
Funktion: **TAB**

FUNKTION: **VAL**
(value)

ZWECK: Ermittlung des numerischen Gehalts eines Strings

FORMAT: **VAL**(Stringausdruck)

WIRKUNG: Der Stringausdruck wird berechnet. Falls die ersten Zeichen dieses Strings einen num. Wert darstellen, wird dieser in den entsprechenden numerischen Wert umgewandelt. Die Umwandlung geschieht zeichenweise von links nach rechts.

BEMERKUNGEN:
- Die Funktion **VAL** ist die logische Umkehrung der Funktion **STR$**.
- Alle plausiblen Zahlendarstellungen werden korrekt umgewandelt.
  ACHTUNG: Overflow möglich!
- Wird keine plausible Zahlendarstellung am Anfang des 'Stringausdrucks' ermittelt, liefert die Funktion den Wert Ø.
- Die Umwandlung wird abgebrochen, sobald ein Zeichen erkannt wird, das für eine numerischen Wert unplausibel ist.

BEISPIELE:

```
10 A$=VAL("12345"):B$="F fm 5":C$="-2e5":D$="1D26"
20 PRINT VAL(A$);VAL(B$);VAL(C$);VAL(D$)
RUN
 12345  0  -2  1D+26
```

VERWEISE: Kapitel 4.3.4.3
Funktion: **STR$**

FUNKTION: **VARPTR**
(variable pointer)

ZWECK: liefert die Byte-Adresse einer Variablen im Arbeitsspeicher oder die Start-Adresse des I/O-Puffers, der der Filenr. eines sequentiellen Files zugeordnet ist bzw. die Adresse des Random-File-Puffers für das unter einer Filenr. geöffnete Random-File

FORMAT: **VARPTR** ({ Variablenname / #Filenr. })

Filenr.: numerische Konstante, Wert: 1-15

WIRKUNG:

a) Angabe eines Variablennamens:
   Die Funktion liefert die Byte-Adresse der angegebenen Variablen im Arbeitsspeicher. Das Ergebnis ist eine Integerzahl zwischen -32768 und 32767. Wird eine negative Zahl ausgegeben, so ist die aktuelle Adresse diese Zahl + 65536.
b) Angabe einer Filenr. in Form einer num. Konstanten:
   Die Funktion liefert die Start-Adresse des I/O-Puffers des unter dieser Filenr. geöffneten sequentiellen Files (Zugriffsarten: "I", "A" oder "O"). Wurde unter der Filenr. ein Random-File geöffnet (Zugriffsart "R"), so wird die Adresse des Random-File-Puffers geliefert.

BEMERKUNGEN:

- Es können nur Adressen von Variablen ermittelt werden, die vorher über eine Zuweisung echte Werte erhielten. Die Default-Werte (Ø bzw. Leerstring) werden nicht über Zuweisung gesetzt. Auch **DIM** und **ERASE** führen keine Zuweisung aus.
- Es kann die Adresse von bestimmten Elementen eines Arrays abgefragt werden.
- Das Array-Element mit dem niedrigsten Index (Ø oder 1) hat stets die niedrigste Adresse aller Array-Elemente im Arbeitsspeicher.
- Werden Adressen von Array-Elementen abgefragt, ist zu beachten:
  Jede Zuweisung auf eine neue Variable ändert die Adressen der Array-Elemente.
- Nach Stringmanipulationen können sich die Adressen aller Variablen ändern.
- Nach Verwendung der Funktion **FRE(**Str.Ausdr.**)** können sich die Adressen aller Variablen ändern.
- Es empfiehlt sich, vor Abfrage einer Adresse eine Zuweisung auf die betreffende Variable zu machen (z.B. A(2)=A(2)).

BEISPIELE:

```
NEW
10 A!=2.45466:B%=44:C!(5)=66
20 PRINT VARPTR(A!);VARPTR(B%);VARPTR(C!(0))
RUN
-29374 -29366 -29355
```

VERWEISE: Kapitel 4.3.16

ANWEISUNG: **WEND**

FUNKTION: Endanweisung einer **WHILE-WEND**-Schleife

FORMAT: **WEND**

WIRKUNG: siehe Anweisung **WHILE**

BEISPIELE:

```
10 I%=1
20 WHILE INT(RND*10)<>9
30 PRINT I%;:I%=I%+1
40 WEND:PRINT:PRINT "Ende nach";I%-1;"Durchläufen"
RUN
 1  2  3  4  5  6
Ende nach 6 Durchläufen
```

VERWEISE: Kapitel 4.3.6
Anweisung: **WHILE, CLEAR**
Fehler-Code: 30

FUNKTION: Start-Anweisung einer Schleife, die solange durchgeführt wird, wie eine Bedingung erfüllt ist

FORMAT:
```
WHILE log.Ausdruck
        .
    Anweisung(en)
        .
WEND
```

log.Ausdruck: Vergleichsausdruck, logischer Ausdruck oder Verknüpfung von Vergleichs- oder logischen Ausdrücken durch logische Operatoren; siehe Kapitel 2.7.3

WIRKUNG: Trifft das System auf eine **WHILE**-Anweisung, wird dieser die nächste freie **WEND**-Anweisung zugeordnet.
Der logische Ausdruck wird berechnet. Ist sein Wahrheitswert Ø, wird die zugeordnete **WEND**-Anweisung angesprungen, die dazwischen liegende(n) Anweisung(en) übergangen und die auf **WEND** folgende Anweisung abgearbeitet.
Ist der Wahrheitswert von 'log.Ausdruck' ungleich Ø, werden alle Anweisungen bis zum **WEND** ausgeführt und von dort wieder zur betreffenden **WHILE**-Anweisung zurückverzweigt.

BEMERKUNGEN:
- **WHILE**-Schleifen können verschachtelt werden.
- Jedes **WHILE** muß mit genau einem **WEND** gepaart sein.

- ACHTUNG: bei vorzeitigem Aussprung aus **WHILE-WEND**-Schleifen (z.B. mit **GOTO**) ohne späteren Rücksprung bleibt ein Stack-Speicher belegt. Dies kann bei wiederholtem Auftreten in einem Programm zu Speicher-Overflows führen (vgl. **FOR...NEXT**-Schleife).

BEISPIELE:

```
10 I%=1
20 WHILE INT(RND*10)<>9
30 PRINT I%;:I%=I%+1
40 WEND:PRINT:PRINT "Ende nach";I%-1;"Durchläufen"
RUN
 1  2  3  4  5  6
Ende nach 6 Durchläufen
```

VERWEISE: Kapitel 2.7.3, 4.3.5 und 4.3.6
Anweisung: **WEND, CLEAR**
Fehler-Code: 29

FUNKTION: Festlegung der Anzahl Zeichen pro Ausgabezeile für Drucker oder Bildschirm

FORMAT: **WIDTH** [**LPRINT**] Anzahl Zeichen

Anzahl Zeichen: num.Ausdruck (Ergebnis: 15-255)

WIRKUNG: 'Anzahl Zeichen' wird berechnet und auf Ganzzahligkeit gerundet. Der Wert legt fest, wieviele Zeichen pro Ausgabezeile ausgegeben werden sollen, bevor eine automatische Zeilenschaltung erfolgt. Ist der Parameter **LPRINT** nicht angegeben, gilt die Anweisung für den Bildschirm, anderenfalls für den Drucker.

BEMERKUNGEN:
- 'Anzahl Zeichen' muß zwischen 15 und 255 liegen.
- Der Default-Wert für 'Anzahl Zeichen' ist für den Bildschirm 64 oder 8Ø Zeichen (wird durch PCOS-Befehl **ss** festgelegt) und für den Drucker 132 Zeichen.
- Ist 'Anzahl Zeichen' gleich 255, ist die Zeichenanzahl pro Zeile "unendlich", d.h., es wird niemals eine Zeilenschaltung (CR) vom System angesetzt. Die Funktionen **POS** bzw. **LPOS** werden jedoch auf Ø gesetzt, sobald die 255. Position erreicht wurde.
- Für Tastatureingaben über **INPUT** oder **LINE INPUT** kann die Zeilenlänge am Bildschirm nicht mit **WIDTH** begrenzt werden.

BEISPIELE:

```
10 A$="12345678901234567890"
20 LPRINT A$
30 WIDTH LPRINT 10:LPRINT A$

1234567890
1234567890
1
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2
Anweisungen: **PRINT, WRITE, LPRINT**
Funktionen: **POS, LPOS**

ZWECK: Definition der Lage und Größe eines neuen Windows ("Anlegen" bzw. "Eröffnen") und/oder (Neu-)Festlegung der Anzahl Zeichen pro Zeile und/oder der Anzahl Zeilen in einem Window; ggf. Ermittlung der Window-Nr. des aktiven Windows

FORMAT: **WINDOW**(Quadrant,Position[,[Zeilenhöhe][,Spaltenbr.]])

Quadrant: num.Ausdruck (Ergebnis: Ø: oberer Teil, 1: unterer Teil, 2: linker Teil, 3: rechter Teil des aktiven Windows)

Position: num.Ausdruck, dessen Ergebnis eine ganze Zahl sein muß und die Größe des neuen Windows bestimmt (horizontale Teilung: max. 239; vertikale Teilung: max. 79)

Zeilenhöhe: num.Ausdruck; zulässige Werte: 1Ø, 11...16

Spaltenbreite: num.Ausdruck; zulässige Wert: 6 (8Ø Zeichen pro Gesamtbildschirm-Zeile) und
8 (64 Zeichen pro Gesamtbildschirm-Zeile)

WIRKUNG: 'Quadrant', 'Position', 'Zeilenhöhe' und 'Spaltenbreite' werden berechnet und kaufmännisch auf Ganzzahligkeit gerundet.

Achtung: Es handelt sich hier um eine **Funktion**. Ihr Ergebnis muß in einer Anweisung eingekleidet sein, z.B. in Form einer Zuweisung wie W%=**WINDOW**-(Ø,15Ø)

I Anlegen ("Eröffnen") eines neuen Windows durch Definition der Größe und ggf. Festlegung der Anzahl Zeichen pro Bildschirmzeile und/oder der Anzahl Zeilen im Bildschirm für dieses Window; als Ergebnis der Funktion wird die Nr. des neuen durch Teilung entstandenen Windows geliefert.

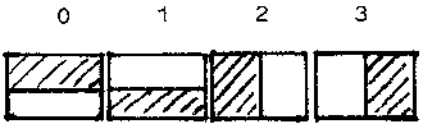
Vorbemerkung:

Ein neues Window kann nur durch waagerechte oder senkrechte Teilung eines bereits existierenden Windows angelegt werden. Das zu teilende Window muß aktiv sein (vgl. Anweisung **WINDOW%**).
Es kann zur gleichen Zeit immer nur ein Window aktiv, d.h. ansprechbar, sein. Zu Beginn ist der Gesamtschirm (Window 1) aktiv.
Durch Teilung des aktiven (zuletzt per **WINDOW%**-Anweisung angewählten oder nach einer Teilung verbliebenen) Windows wird ein neues Window angelegt und das aktive um diesen Bereich reduziert. Nach der Teilung bleibt das Ausgangswindow aktiv. Der Inhalt des neuen Windows (Text wie Graphik) wird gelöscht; der Inhalt des verbliebenen (aktiven) Windows bleibt erhalten.
Als Ergebnis der Funktion wird diejenige ganze Zahl zwischen 2 und 16 geliefert, die dem aktiven Window folgt und noch nicht vergeben ist, um ein Window zu spezifizieren.

Das neue Window ist in Zukunft unter dieser Zahl "eröffnet" und anzusprechen (ggf. allerdings vorher zu aktivieren!). Der volle Bildschirm ist zu Beginn grundsätzlich aktiv und hat die Window-Nr. 1.

Quadrant:

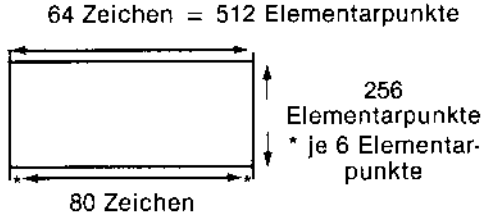
Durch den Parameter 'Quadrant' wird die Lage (und nur die Lage) des neuen Windows in Bezug auf das aktive Window bestimmt. Er kann die Werte Ø, 1, 2 oder 3 annehmen und legt damit fest, daß das neue Window im oberen, unteren, linken oder rechten Teil des aktiven Windows angelegt wird. Die nachfolgende Skizze soll dies veranschaulichen:



Das neue angelegte Window ist jeweils schraffiert dargestellt. Alle im neu angelegten Window vorhandenen Angaben (Text wie Graphik) werden gelöscht. Der unschraffierte Bereich ist das reduzierte Ausgangswindow.
Man beachte, daß durch den Parameter 'Quadrant' nur die Lage, nicht aber die Größe des neuen Windows definiert wird.

Position:

Mit 'Position' wird die Größe des neuen Windows festgelegt. Das aktive Window, aus dem das neue durch Teilung hervorgeht, wird um den entsprechenden Bereich reduziert. Die Größe des Parameters 'Position' wird bei horizontaler Teilung in Anzahl Elementarpunkten und bei vertikaler Teilung in Anzahl Textspalten angegeben.
Der Gesamtschirm besitzt vertikal 256 Elementarpunkte und horizontal entweder 64 oder 80 Textspalten (vergleiche dazu Kapitel 4.3.18).



Wurde für 'Quadrant' Ø oder 1 (horizontale Teilung) gesetzt, so ist für 'Position' die Anzahl Elementarpunkte vom oberen Rand des aktiven Windows anzugeben, gleichgültig ob das neue Window im oberen oder unteren Teil liegt. Das neue Window besitzt dann vertikal eine Ausdehnung, die gleich der angegebenen Anzahl Elementarpunkte ist. Das Ausgangswindow erstreckt sich danach vertikal über soviel Elementarpunkte, wie das Ausgangswindow vor der Teilung hatte, minus der für das neue Window definierten Anzahl Elementarpunkte ('Position').

Für die horizontale Teilung: eine Textspalte besteht - je nach Bildschirmformat (80 oder 64 Zeichen pro Zeile) - aus 6 oder 8 Elementarpunkten.
Der Wertebereich für den Parameter 'Position' ist demnach wie folgt definiert:

	Parameter 'Position'	
Horizontale Teilung (Quadrant Ø u. 1)	'Zeilenhöhe'+1='Position'=Anzahl Elementarpunkte des aktiven Windows (vertikal) minus 'Zeilenhöhe'	
vertikale Teilung (Quadrant 2 u.3)	1='Position'=	79 bei 80 Zeichen/Bildschirmzeile; 63 bei 64 Zeichen pro Bildschirmzeile

Tabelle 1: 'Position'

Zeilenhöhe:

Durch Angabe der 'Zeilenhöhe' - in Anzahl Elementarpunkten - kann der Zeilenabstand innerhalb des Windows festgelegt werden. Dabei setzt sich die 'Zeilenhöhe' zusammen aus der Zeichenhöhe (7 Elementarpunkten) plus 3 bis 9 Elementarpunkte für den Abstand zwischen den Zeilen.
In der nachfolgenden Tabelle wird Anzahl Zeilen pro Gesamtschirm in Abhängigkeit vom Parameter 'Zeilenhöhe' dargestellt:

'Zeilenhöhe' (Elementar-punkte)	Zeilenabstand (Elementar-punkte)	Anzahl Zeilen im Gesamt-bildschirm
10	3	25
11	4	23
12	5	21
13	6	19
14	7	18
15	8	17
16	9	16

Tabelle 2: 'Zeilenhöhe'

Ist dieser Parameter nicht angegeben, wird die durch den PCOS-Befehl **ss** festgelegte 'Zeilenhöhe' übernommen.

Spaltenbreite

Durch die Angabe der 'Spaltenbreite' (6 oder 8 Elementarpunkte) kann man den Spaltenabstand innerhalb des Windows festlegen. Dabei setzt sich die 'Spaltenbreite' zusammen aus der Zeichenbreite (5 Elementarpunkte) plus Spaltenabstand (3 oder 1 Elementarpunkt(e)). Bezogen auf den Gesamtschirm wird durch die Angabe der 'Spaltenbreite' die Anzahl Zeichen pro Zeile von 80 auf 64 und umgekehrt geschaltet.

'Spaltenbreite' (Elementarpunkte)	Spaltenabstand (Elementarpunkte)	Anzahl Zeichen im Gesamtbildschirm
6	1	80
8	3	64

Tabelle 3: 'Spaltenbreite'

Ist der Parameter 'Spaltenbreite' nicht angegeben, wird die durch den PCOS-Befehl **ss** festgelegte 'Spaltenbreite' übernommen.

II Neufestlegung der Anzahl Zeichen pro Zeile und/oder Anzahl Zeilen für ein bereits angelegtes Window; als Ergebnis der Funktion wird die Window-Nr. des aktiven Windows geliefert.

Für ein bereits existierendes Window können 'Zeilenhöhe' und 'Spaltenbreite' neu definiert werden. Dazu muß die Funktion **WINDOW** in dem Window aufgerufen werden, dessen Eigenschaften geändert werden sollen; das betreffende Window muß also aktiv sein. Aktiv ist immer das Window, das entweder nach einer Teilung mit Hilfe der **WINDOW**-Funktion verblieben ist oder durch eine **WINDOW%**-Anweisung aktiviert wurde.
Die Parameter 'Quadrant' und 'Position' müssen hierzu den Wert Ø besitzen. Im Falle der Neufestlegung wird der gesamte Inhalt des aktiven Windows gelöscht. Die Funktion liefert nun als Ergebnis die Window-Nr. des aktiven Windows.
Die Parameter 'Zeilenhöhe' und 'Spaltenbreite' sind anzugeben wie unter I beschrieben.

BEMERKUNGEN:

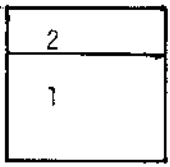
- Beim Anlegen von Windows liefert die Funktion eine ganze Zahl zwischen 2 und 16, die die Nr. des neuen Windows angibt, während sie bei Neufestlegung von 'Zeilenhöhe' und/oder 'Spaltenbreite' in einem existierenden aktiven Window als Ergebnis die Nr. dieses (aktiven) Windows liefert.
- Auf dem Bildschirm können gleichzeitig Windows mit z.B. 80x25-Format und 64x16-Format dargestellt werden. Dies sind die möglichen Default-Werte für das Format.
- Es können maximal 16 Windows gleichzeitig verwaltet werden. Im PCOS-Befehl **sb** ist die Anzahl Windows anzugeben, die maximal angelegt werden können.
- Im Window 1 (Gesamtschirm) wird für 'Zeilenhöhe' und 'Spaltenbreite' der im PCOS-Befehl **ss** angegebene Default-Wert angenommen.
- Wird beim Anlegen des Windows 'Zeilenhöhe' und/oder 'Spaltenbreite' nicht angegeben, so werden die durch **ss** festgelegten Werte angenommen.
- Sind beim Anlegen von Windows Parameter angegeben, die keine Teilung mehr möglich machen, wird die Meldung "Unable to create window" gegeben.
- In jedem Window kann alphanumerisch und/oder graphisch gearbeitet werden.
- In jedem Window kann mit dem Text-Cursor und/oder dem Graphik-Cursor gearbeitet werden. Dabei bezieht sich seine Position auf das Window (vgl. **CURSOR**).

- Die Anzahl Elementarpunkte im Parameter 'Position' muß kein Vielfaches des Parameters 'Zeilenhöhe' sein.
- Die Anzahl Textzeilen innerhalb eines Windows ergibt sich bei vorgegebener 'Zeilenhöhe' aus **INT('Position'/'Zeilenhöhe')**
- Der Ursprungspunkt des Text-Cursors ist die oberste linke Textposition jedes Windows.
- Der Default-Wert für den Koordinatensprung beim Arbeiten mit dem Graphik-Cursor ist der unterste linke Elementarpunkt des betreffenden Windows.
- Die letzte Position und die Eigenschaften des jeweiligen Cursors bleiben beim Aktivieren eines anderen Windows gespeichert, so daß jederzeit darauf zurückgegriffen werden kann.

BEISPIELE: 1. (Es ist noch kein Window angelegt worden, 64x16-Format ist gültig):

```
Z%=WINDOW(Ø,64)
```

Der Bildschirm wird in zwei Windows aufgeteilt, wobei das neue Window im oberen Teil entsteht und vertikal eine Ausdehnung über 64 Elementarpunkte hat. Der Variablen Z% wird vom System der Wert 2 zugewiesen. Das Window 1 wird auf die Größe von 192 Elementarpunkten reduziert.



64 Elementarpunkte

192 Elementarpunkte

'Zeilenhöhe' und 'Spaltenbreite' werden aus dem im PCOS-Befehl **ss** gesetzten Default-Wert für das Bildschirmformat abgeleitet. Sind z.B. aufgrund von **ss** eine 'Zeilenhöhe' von 16 und für 'Spaltenbreite' 8 gültig, wird Window 2 aus 4 und Window 1 aus 12 Zeilen bestehen.
Das Window 1 ist nach wie vor aktiv, d.h., falls ein neues Window angelegt wird, leitet es sich aus diesem Window her.

2. (Es ist noch kein neues Window angelegt worden):

```
R%=WINDOW(1,64)
```

wie Bespiel 1., jedoch liegt das neue Window unten. Die Variable R% hat den Wert 2.

3. (Es ist vorher kein Window angelegt worden):

```
W%=WINDOW(Ø,15)
WINDOW%2
A%=WINDOW(Ø,Ø,,6)
```

Im aktiven Window 2 (oben!) wird 'Spaltenbreite' zu 6 geändert, d.h. in diesem Window gilt enge Schrift. In der Variablen A% steht die Nr. des Windows, das gerade aktiv ist, also 2, in W% steht ebenfalls 2.

4.

```
10 CLEAR:CALL "ss %m,,,,,0" '64x16-Format
20 DEFINT W
30 W=WINDOW(3,32,,6):WINDOW%W
40 W=WINDOW(1,80,,8):WINDOW%W
50 FOR I%=1 TO 3:CLS %I%:NEXT
55 PRINT "Bitte Window-Nr. wählen (1-3)"
60 K$=INPUT$(1):IF INSTR("123",K$)=0 THEN 60
70 WINDOW %VAL(K$)
71 K1$="" WHILE K1$=""
75 K1$=INKEY$
80 PRINT K$;:WEND
100 PRINT "Dies ist Window Nr.";VAL(K$):GOTO 55
```

VERWEISE: Kapitel 4.3.9, 4.3.14 und 4.3.18
Anweisungen: **WINDOW%, CLOSE WINDOW, CLEAR, CLS, CURSOR, SCALE, COLOR, COLOR=**

FUNKTION: Anwählen ("Aktivieren") eines angelegten Windows

FORMAT: **WINDOW%** Window-Nr.

Window-Nr.: numerischer Ausdruck (Ergebnis: 1-16)

WIRKUNG: 'Window-Nr.' wird berechnet und auf Ganzzahligkeit kaufmännisch gerundet.
Das mit **WINDOW%** angewählte Window ist dann aktiv. Alle weiteren Anweisungen beziehen sich dann auf dieses Window (Ausnahmen: Anweisungen, in denen gezielt andere als das aktive Window angesprochen werden können). Es bleibt so lange aktiv, bis ein anderes Window aktiviert wird.

BEMERKUNGEN:
- Vor Beginn ist der gesamte Bildschirm unter der Window-Nr. 1 aktiviert.
- Das Anlegen von neuen Windows geschieht immer durch Teilung des aktiven Windows.
- Anweisungen wie **PRINT** oder **INPUT** beziehen sich immer auf das aktive Window.
- Soll ein neues Window aktiviert werden, ist eine neue **WINDOW%**-Anweisung nötig. Es kann nur ein Window aktiv sein.
- Von den letzten beiden Bemerkungen ausgenommen sind alle Funktionen und Anweisungen, in denen ein Window spezifiziert werden kann (erkenntlich am **%**-Zeichen als Bestandteil, z.B. **CLS %**, **GET %**.

- Ein Window kann nur aktiviert werden, wenn es vorher einmal unter Zuhilfenahme der **WINDOW**-Funktion angelegt ("eröffnet") wurde und das betreffende Window nicht durch eine **CLOSE WINDOW**-Anweisung aufgelöst wurde. In einem solchen Fall wird der Fehler "Window not open" gemeldet.

BEISPIELE:

```
5 CLEAR:CALL "ss %n,,,,,0" '64x15-Format
10 DEFINT J,W
20 W1=WINDOW(1,100,10,6)
30 W2=WINDOW(3,50,10,8)
40 FOR J=1 TO 3
50 WINDOW%J:FOR J1=1 TO 5-5*(J=1):PRINT "Window";J1
60 NEXT J1,J
70 CALL "ls 2"  'Hardcopy Text in Window 2
80 CALL "sp"    'ges. Hardcopy; nur graphikfähige Drucker
```

VERWEISE: Kapitel 4.3.9 und 4.3.14
Anweisungen: **CLS, CLOSE WINDOW, CURSOR, COLOR=, COLOR, CLEAR**
Funktion: **WINDOW**
Fehler-Code: 35

ANWEISUNG: **WRITE**

FUNKTION: Ausgabe von Daten am Bildschirm; als Trennzeichen werden automatisch Kommas ausgegeben; Strings werden automatisch in Anführungszeichen eingeschlossen ausgegeben

FORMAT: **WRITE** [Liste von Ausdrücken]

Liste von Ausdrücken: numerische und/oder String-Ausdrücke; getrennt durch Kommas

WIRKUNG: Die Ergebnisse der durch Komma getrennten Ausdrücke in der 'Liste von Ausdrücken' werden auf dem Bildschirm ausgegeben. Zwischen die Elemente der 'Liste von Ausdrücken' werden automatisch Kommas gesetzt. Strings werden automatisch in Anführungszeichen eingeschlossen ausgegeben. Nach dem letzten Element der 'Liste von Ausdrücken' wird automatisch eine Zeilenschaltung vorgenommen. **WRITE** ohne Parameter bewirkt eine Zeilenschaltung.

BEMERKUNGEN:
- Numerische Werte werden im Standardformat (vgl. **PRINT**-Anweisung) ausgegeben.
- Eine weitere Formatierung mit Hilfe von **USING** ist nicht möglich.
- Eine Tabulation mit **TAB** ist nicht möglich, wohl aber eine Ausgabe ab einer durch **CURSOR** festgelegten Position.

- Der Bildschirm-Cursor kann nicht durch ; an der letzten Position der Ausgabeliste festgehalten werden; es erfolgt zwangsweise eine Zeilenschaltung.

BEISPIELE:

```
10 CLS:A!=20:A$="M":B$="ist Spitze"
20 PRINT A$;A!;B$:WRITE A$;A!;B$
RUN
M 20 ist Spitze
"M",20,"ist Spitze"
```

VERWEISE: Kapitel 4.3.9.2
Anweisungen: **PRINT, CURSOR, WIDTH**

ANWEISUNG: **WRITE#**

FUNKTION: Schreiben von Daten auf ein sequentielles File; als Trennzeichen wird automatisch ein Komma auf das File geschrieben; Strings werden automatisch in Anführungszeichen eingeschlossen auf das File gebracht; am Ende wird ein CR und ein LF geschrieben

FORMAT: **WRITE#** Filenr.,Liste von Ausdrücken

Filenr.: num.Ausdr. (Ergebnis: 1-15)
Liste von Ausdrücken: numerische und/oder String-Ausdrücke; getrennt durch Kommas

WIRKUNG: Die 'Filenr.' wird berechnet und gerundet. Auf das unter dieser Filenr. geöffnete sequentielle File (Zugriffsart: "O" oder "A") werden ab der Position des Pointers die Ergebnisse der durch Komma getrennten Ausdrücke geschrieben. Zwischen die Elemente der 'Liste von Ausdrücken' wird ein Komma (=**CHR$(44)**) auf das File geschrieben. Strings werden in Anführungszeichen (**CHR$(34)**) eingeschlossen auf das File gebracht. An das Ende der 'Liste von Ausdrücken' wird automatisch ein CR (=**CHR$(13)**) und ein LF (=**CHR$(10)**) auf das File geschrieben.

BEMERKUNGEN:
- Numerische Werte werden im Standardformat (vgl. **PRINT**-Anweisung) auf das File gebracht.

- Enthalten Stringvariablen führende oder am Ende stehende Blanks, werden diese auf das File geschrieben.
- **WRITE#** schreibt die Daten im Grunde so auf das File, wie sie durch **WRITE**-Anweisung auf dem Bildschirm dargestellt würden, allerdings einschließlich nicht druckbarer Zeichen.
- Im Gegensatz zu **PRINT#** werden die Daten automatisch durch das Komma getrennt auf das File geschrieben; die Strings werden automatisch in Anführungszeichen eingeschlossen und das Ende der 'Liste von Ausdrücken' immer durch ein CR (=**CHR$**(13)) und ein LF (=**CHR$**(10)) gekennzeichnet.
- **WRITE#** ist bei Schreibschutz einer Diskette nicht möglich.
- Das Schreiben auf Diskette erfolgt nicht automatisch sofort nach **WRITE#**, sondern erst, wenn der File-Puffer voll ist. Erst nach **CLOSE** für das betreffende File ist der Rest des File-Puffers mit Sicherheit geschrieben.
- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu reservieren, wie durch den PCOS-Befehl ss spezifiziert wurde. Dies kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.

- Mit Hilfe der Funktion **LOC** kann abgefragt werden, ob auf Diskette noch genügend Platz für die von **WRITE#** betroffenen Daten ist.

BEISPIELE:

```
10 OPEN "O",1,"1:DATEN5.seq"
20 A$="TEXT":NR%=1
40 WRITE#1,A$,NR%
50 CLOSE 1:OPEN "I",1,"1:DATEN5.seq"
60 WHILE NOT EOF(1)   'zeichenweise lesen
70 K$=INPUT$(1 #1):PRINT K$,
80 WEND CLOSE
```

VERWEISE: Kapitel 4.3.10.2

Anweisungen: **OPEN, INPUT#, LINE INPUT#, PRINT#, WRITE**

Funktionen: **LOC, LOF**

Fehler-Code: 54,61

OPERATOR: **XOR**
(exclusive or)

FUNKTION: Verknüpfung zweier logischer Ausdrücke mit Hilfe der expliziten ODER-Verknüpfung bzw. Bildung der symetrischen Differenz zweier Integerwerte (Bit-für-Bit-Verknüpfung mit **XOR**)

FORMAT: {log.Ausdruck / Integerwert} **XOR** {log.Ausdruck / Integerwert}

WIRKUNG: Das Ergebnis der **XOR**-Verknüpfung zweier Bedingungen ist
-1 (wahr), wenn eine der beiden Bedingungen erfüllt ist, aber nicht beide;
Ø (falsch), wenn keine der Bedingungen oder alle beide erfüllt sind.
Bei der **XOR**-Verknüpfung zweier Integerwerte wird die **XOR**-Operation Bit für Bit durchgeführt und das Ergebnis berechnet (vgl. Kapitel 2.6.4).

BEISPIELE:

```
10 A%=5:B%=3:C!=2.47
15 C%=A% XOR B%   ' C% ist jetzt 6
20 IF A%<B% XOR C!>3 THEN GOSUB 1000 ELSE GOSUB 2000
```

VERWEISE: Kapitel 2.6.4
Operatoren: **NOT, AND, OR, EQV, IMP**

ANWEISUNG: **?**

FUNKTION: Kurzschreibweise des BASIC-Sprachelements **PRINT**

FORMAT: s. **PRINT (USING)** und **PRINT# (USING)**

BEMERKUNGEN:
- **?** wird vom BASIC-Interpreter bei **(L)LIST** und bei **SAVE** grundsätzlich in **PRINT** verwandelt.
- **?** kann nicht zur verkürzten Eingabe von **LPRINT** verwendet werden.

ANWEISUNG:

FUNKTION: Einleitung einer Bemerkung (Kommentar); wie **REM**-Anweisung

FORMAT: Kommentar

WIRKUNG: Siehe Anweisung **REM**

BEMERKUNGEN:
- benötigt weniger Speicherplatz als **REM**; bei interpretativem Abarbeiten des BASIC-Programms ist die Interpretation langsamer als bei Verwendung von **REM**.
- Bei einer **REM**-Anweisung am Ende einer Mehrfach-Anweisungszeile (als letzte Anweisung) kann der : vor ' entfallen, nicht aber vor **REM**.
- ' wird bei **SAVE** und (L)**LIST** nicht in **REM** verwandelt.
- Alles hinter ' in einer Anweisungszeile wird als Bemerkung interpretiert, es sei denn, ' ist in Anführungszeichen eingeschlossen.

BEISPIELE:

```
0 DATE$="15.05.84"    :REM Programm erstellt am 15.5.84
1 PRINT DATE$    ' auch hier folgt eine Bemerkung
```