Business Computer

MZ-3500

BASIC
PROGRAMMIERSPRACHENHANDBUCH



SHARP

Business Computer

MZ-3500

Programmier Sprachen Handbuch

© SHARP CORPORATION

VORWORT

Das vorliegende Handbuch bezieht sich auf die FDOS-Systemsoftware (FDOS Master File) der Bürocomputer der Modellserie 3500.

1. Alle Systemsoftware für die Bürocomputer der Modellserie 3500 wird in Form von Dateien auf Disketten oder anderen Datenträgern geliefert. Inhalt der Datenträger und dieses Handbuches können ohne Vorankündigung geändert werden. Daher sollten die Versionsnummern der Programme sorgfältig geprüft werden.
2. Dieses Handbuch wurde mit großer Sorgfalt erstellt. Sollten trotzdem Fragen oder Unklarheiten auftauchen, wird der zuständige SHARP-Händler gern Auskunft erteilen.
3. Alle Systemsoftware (FDOS Master File) der Modellserie 3500 ist Eigentum von SHARP und unterliegt dem Urheberrecht. Die Anfertigung von Kopien der Systemsoftware oder dieses Handbuches, als ganzes oder in Teilen, in welcher Form auch immer, ist ohne Einverständnis von SHARP nicht gestattet.

INHALTSVERZEICHNIS

I. EINFÜHRUNG 1
1. ARBEITSWEISE EINES COMPUTERS 1
(1) Starten von BASIC 1
(2) Manuelle Betriebsart 2
(3) Konstanten 9
(4) Variablen 11
(5) Numerische und Stringausdrücke 14
(6) Arithmetische Vorrangregeln 15
(7) Ausgabeformate für Zahlen 16
(8) Array-Variable 19
(9) Fehlermeldungen 22
2. FUNKTIONEN 23
(1) Stringfunktionen 23
(2) Numerische Funktionen 26
3. TEXT UND GRAPHIK 31
(1) Text- und Graphikdarstellung 31
(2) Koordinatensysteme und Kursor 33
(3) Farbwiedergabe 34
4. DATEIEN 35
(1) Was ist eine Datei ? 35
(2) Dateiarten 37
5. FDOS BETRIEBSSYSTEM 40
(1) Was ist FDOS ? 40
(2) Begriffe 41
(3) FDOS-Befehle 43
BASIC 44
INIT 45
CAT 47
BACKUP 48
COPY 50
SCOPY 52
KILLALL 54
DEBUG 55
6. FREI DEFINIERBARE FUNKTIONSTASTEN 58
(1) Belegen der freien Funktionstasten 58
(2) Anwendungsbeispiele 59
(3) Auflisten und Löschen der frei definierbaren Tasten 60

II. PROGRAMMIERUNG IN BASIC 61
1. ALLGEMEINES ZUR PROGRAMMIERUNG 61
(1) Aufbau eines BASIC-Programms 61
(2) Programmieren 62
2. KORRIGIEREN UND EDITIEREN 66
(1) Ändern von Programmen 66
(2) Der Korrekturvorgang 67
(3) Einfügen und Löschen von Zeilen 69
3. TESTHILFEN 71
(1) Einzelschrittest 71
(2) Ablauftest 73

III. BASIC-SCHLÜSSELWORTE 75
1. DEFINITION DER SYNTAX-SYMBOLE 75
2. BEFEHLE 77
RUN 78
VRUN 80
CONT 82
LIST 85
EDIT 90
NEW 92
DELETE 93
ERASE 95
AUTO 96
REN 98
TR 100
TR STOP 102
3. STANDARD-ANWEISUNGEN 103
LET 104
DISP 107
PRINT 110
IMAGE 113
INPUT 117
DATA 120
READ 121
RESTORE 123
REM 125
STOP 126
END 127
DIM 128
GO TO 131
GO SUB 134
RETURN 136
ON . . . GO TO 138

ON . . . GOSUB 141
IF . . . THEN 141
IF . . . : ELSE 144
FOR . . . TO . . . STEP/NEXT 149
4. SONDERANWEISUNGEN 156
CALL 157
SUB 160
SUBEND 162
KEY 163
KEYIN 165
POP 171
ON ERROR 172
OFF ERROR 175
ON KEY 176
OFF KEY 180
ON TIME 181
OFF TIME 184
RESUME 185
MOVE 187
MAXS 191
MINS 193
SEARCH 195
DEFFN 199
MUSIC 202
WAIT 204
DEFINT 206
DEFREAL 207
DEFSTR 208
DEG 210
RAD 211
RANDOM 212
ERROR MUSIC 213
5. BILDSCHIRM- UND DRUCKERANWEISUNGEN 214
DISP CLEAR 215
SET 217
RESET 219
POS 220
ENTER 221
CURSOR 223
MVCUR 225
LINE 226
BL 228
RV 229
RE 230

TABLE 232
SCROL 235
CHANGE DISP 237
CHANGE PRINT 238
COLOR 239
6. ANWEISUNGEN ZUR ERZEUGUNG VON GRAPHIK 240
SDISP 241
ODISP 243
GCOLOR 245
CLS 247
SCALE 249
PSET 252
PRESET 254
LINE 255
CIRCLE 259
PAINT 264
GENTER 269
GDISP 273
GPRINT 276
GCURSOR 277
GINPUT 279
GCHR 282
7. VERBINDUNG VON BASIC UND MASCHINENSPRACHE 286
NEW 287
ERASE 288
POKE(PEEK) 289
USR 290
8. STEUERUNG DER RS-232C SCHNITTSTELLE 293
CHANNEL 294
SEND 296
RCV 300
SENREV 303
OPCHNL 305
POLLING 306
RLIST 308
TERM 310

IV. ANWEISUNGEN FÜR DEN DISKETTENBETRIEB 312
1. ÜBERSICHT 313
2. DATEIVERWALTUNG 314
LOCK 315
UNLOCK 317
COPY 318
KILL 319

CAT 320
RNAME 323
LPRINT 324
LINPUT 326
BYE 328
3. DATEIVERARBEITUNG 329
MAXFILE 330
CREATE 332
OPEN 334
CLOSE 336
3.1. Sequentielle Dateien 337
BPRINT# 339
BINPUT# 341
PRINT# 342
INPUT# 345
SEP# 347
3.2. Direktzugriffsdateien 348
RFORMAT# 350
PUT# 352
GET# 353
4. PROGRAMMDATEIEN 354
SAVE 355
STORE 358
LOAD 360
CHAIN 362
MERGE 364
LOAD SUB 366
ALOAD 370
5. SONDERDATEIEN 372
MSAVE 373
MLOAD 374
VSAVE 375
VLOAD 376
GSAVE 377
GLOAD 379
KSAVE 380
KLOAD 381
STICHWORTVERZEICHNIS 382

I. EINFÜHRUNG

1. ARBEITSWEISE EINES COMPUTERS

Wenn ein Computer eine bestimmte Aufgabe übernehmen soll, reicht ein einfaches Einschalten nicht aus, sondern es sind noch einige weitere Vorbereitungen zu treffen.
Der Computer benötigt dazu Instruktionen in Form eines Programms (siehe II. 1. "Allgemeines zur Programmierung, Seite 61). Das Programm selbst wird in einer Programmiersprache abgefaßt; die in diesem Handbuch beschriebene Sprache BASIC (Abkürzung von "Beginner's All-Purpose Symbolic Instruction Code") ist eine von vielen Programmiersprachen.
Computer verstehen jedoch nur ihre prozessorspezifische Maschinensprache. Zur Ausführung eines BASIC-Programms übersetzt der Computer jede BASIC-Instruktion intern in die Maschinensprache. Die Umsetzung geschieht durch ein (selbst in Maschinensprache geschriebenes) Programm, den sog. BASIC-Interpreter, der in Zukunft kurz 'BASIC' genannt wird.
Der BASIC-Interpreter dieses Systems wird als Datei*1) auf einer Diskette mitgeliefert.

(1) Starten von BASIC

Um BASIC auf diesem System benutzen zu können, ist der BASIC-Interpreter über die nachfolgend beschriebenen Schritte zu starten (siehe IV. 2. "Einschalten", Seite 19 im Bedienerhandbuch):

① Mitgelieferte Diskette (FDOS Master Disk) in das rechte Laufwerk (Drive No. 0) einschieben und Rechner einschalten.

② Nach dem Einschalten erscheint folgende Meldung auf dem Bildschirm:

```
FDOS   22007A(V1.0A)
READY
```

③ An der Tastatur eingeben:
BASIC [ENTER]

④ Auf dem Bildschirm erscheint die Meldung:

```
BASIC   22007B(V1.0A)
READY
>
```

Auf diese Weise wurde BASIC von der Diskette in den Arbeitsspeicher*2) geladen und ist nun bereit zur Arbeit.

WICHTIGER HINWEIS

Zum Schutz der Original-Systemdiskette ist es sehr zu empfehlen, mindestens eine Arbeitskopie davon zu erstellen (siehe Seite 48)!

---

*1) Datei: Ein auf einer Diskette aufgezeichnetes Programm oder ein Satz von Daten.

*2) Arbeitsspeicher: Der Teil des Rechners, in dem Programme und Daten zur Ausführung bereitgestellt werden.

Bei der Ausführung von Schritt ② wird der IPL*[1] (Initial Program Loader) automatisch gestartet, der FDOS*[2] in den Arbeitsspeicher lädt und diesem die Systemsteuerung übergibt. Der Rechner befindet sich nun im Betriebssystemmodus und kann FDOS-Befehle*[3] von der Tastatur annehmen und ausführen.
Bei der Ausführung von Schritt ③ (BASIC ist einer der FDOS-Befehle) lädt FDOS den BASIC-Interpreter in den Arbeitsspeicher und übergibt ihm die weitere Steuerung. Der Rechner ist jetzt im BASIC-Operationsmodus und damit in der Lage, BASIC-Programme zu bearbeiten und manuelle BASIC-Instruktionen (siehe weiter unten) auszuführen. "READY" zeigt an, daß BASIC zur Annahme von Instruktionen bereit ist.
Das blinkende Rechteck unter "READY" heißt "Kursor", es zeigt die Ein-/Ausgabeposition auf dem Bildschirm an.

(2) Manuelle Betriebsart

Unter "manueller Betriebsart" versteht man die unmittelbare Ausführung von eingegebenen Instruktionen, ohne daß sie im Speicher als Programm festgehalten werden, ähnlich einem Tischrechner (Calculator). So erhält man nach der Eingabe der folgenden DISP-Anweisung*[4] sofort das Ergebnis von 240 x 1150.

```
DISP 240 * 1150 [ENTER]
```

Die DISP-Anweisung dient zur Darstellung bestimmter Daten auf dem Bildschirm, in diesem Beispiel also dem Zahlenwert 240 * 1150. Der Stern (*) steht dabei für das Multiplikationszeichen (x).
Gibt man eine Anweisung ein, wartet BASIC, bis die [ENTER] -Taste gedrückt wird, die gewissermaßen "Eingabe-Ende" anzeigt. Erst jetzt wird die Anweisung ausgeführt und das Ergebnis wie unten gezeigt auf dem Bildschirm ausgegeben:

```
DISP 240 * 1150
276000
```

Die Ausführung eines Programms kann zum Durchführen manueller Operationen unterbrochen werden.
Der Betriebsartschalter*[5] sollte vor manuellen Operationen in Stellung OP gebracht werden. Die meisten manuellen Operationen lassen sich zwar auch in Stellung PRO ausführen, aber in diesem Fall sind etwa im Speicher befindliche Programme nicht geschützt! Gibt man z.B. versehentlich statt eines numerischen Ausdrucks nur eine Zahl ein, so hat das in Stellung OP keine weiteren Folgen. In Stellung PRO jedoch würde eine Programmzeile*[6] gelöscht, wenn ihre Zeilennummer mit der eingegebenen Zahl übereinstimmt. (Siehe II2.3. "Einfügen und Löschen von Programmzeilen".)

---

*1) IPL † Urlader oder Bootstraplader: Programm zum Laden des Betriebssystems von der Diskette, nachdem der Rechner eingeschaltet wurde.
*2) FDOS siehe Abschnitt I.6. (1) "Das Betriebssystem FDOS", Seite 40.
*3) Befehle sind Instruktionen an den Rechner, die manuell an der Tastatur eingegeben werden.
*4) Anweisungen sind Instruktionen, die innerhalb von Programmen benutzt und automatisch ausgeführt werden.
*5) Betriebsartschalter siehe Seite 23 im Bedienerhandbuch.
*6) Programmzeile und Zeilennummer siehe Abschnitt II. 1.(1) "Programmierung in BASIC", Seite 62.

1) Eingabe numerischer Daten (Zahlen)

Folgende Tasten können benutzt werden:

Vorzeichen ..... + oder – (+ kann entfallen)

Ziffern ..... 0 – 9

Dezimalpunkt ..... .

Exponent ..... E

Hexadezimal ..... &

Beispiele:

Zahlenwert	
-123.45	-123.45
1.2×10^{-34}	1.2E-34
$3A71_{16}$	&3A71

2) Arithmetische Operationen

Zum Kennzeichnen arithmetischer Operationen stehen die Operatoren + (Addition), – (Subtraktion), * (Multiplikation), / (Division) und ∧ (Exponentiation) zur Verfügung.

BEISPIEL:

Tastenfolgen für (manuelle) arithmetische Operationen

Operation	Arithm. Ausdr.	Tastenfolge			Ergebnis
Addition	8+3=	DISP	8+3	ENTER	11
	−1.2+4=	DISP	-1.2+4	ENTER	2.8
Subtraktion	4−8=	DISP	4-8	ENTER	-4
	5.8−3.1=	DISP	5.8-3.1	ENTER	2.7
Multiplikation	3×5.1=	DISP	3*5.1	ENTER	15.3
	2.5×(−1.6)=	DISP	2.5*(-1.6)	ENTER	-4
Division	35÷4=	DISP	35/4	ENTER	8.75
	−7.8÷(−3)=	DISP	-7.8/(-3)	ENTER	2.6
Gemischt	(5+3−1)÷2=	DISP	(5+3-1)/2	ENTER	3.5
	3×2−21÷3=	DISP	3*2-21/3	ENTER	-1
Potenzierung	3^4=	DISP	3^4	ENTER	81
	2.8^3=	DISP	2.8^3	ENTER	21.952

Man beachte, daß bei der Multiplikation ein Stern (*) zu schreiben ist anstelle des sonst üblichen x, bei der Division ein Schrägstrich (/) anstelle von ÷ oder:.

Die DISP-Anweisung kann weggelassen werden, sie ist nur dann erforderlich, wenn der erste Operand in einem Ausdruck eine Zahl in Exponentenschreibweise ist. In Stellung OP des Betriebsartschalters kann DISP bei jeder arithmetischen Operation weggelassen werden. (Näheres siehe DISP-Anweisung auf Seite 107).

3) Vergleichsoperationen

Die folgende Tabelle führt die Relationsoperatoren für Vergleiche auf. Das Ergebnis einer Vergleichsoperation ist entweder 1 (wahr), wenn die Relation erfüllt ist, oder es ist 0 (falsch). Ein Vergleichsausdruck besteht aus zwei numerischen Ausdrücken, zwischen denen ein Vergleichsoperator steht. Zur Definition eines numerischen Ausdrucks siehe Seite 30.

Operator	Bedeutung – Ausdruck ist wahr wenn:
=	Linke Seite gleich der rechten Seite
>	Linke Seite größer als rechte Seite
<	Linke Seite kleiner als rechte Seite
> = = >	Linke Seite gleich oder größer als rechte Seite
< = = <	Linke Seite gleich oder kleiner als rechte Seite
< > > <	Linke Seite ungleich der rechten Seite

BEISPIEL:

Tastenfolgen für Vergleichsoperationen (hier im manuellen Betrieb)

Eingabe			Ergebnis
DISP	3>5	ENTER	0
DISP	3<5	ENTER	1
DISP	1.2>=2	ENTER	0
DISP	1.2<=2	ENTER	1
DISP	3<>5	ENTER	1
DISP	3<>3	ENTER	0

WICHTIGER HINWEIS

Bisher wurden nur numerische Vergleiche dargestellt, jedoch können auch Strings (Zeichenketten) miteinander verglichen werden. Der Vergleich geschieht hier Zeichen für Zeichen, beginnend beim ersten, auf der Grundlage des (äquivalenten) ASCII-Wertes.

BEISPIEL:

Vergleiche von Strings

Eingabe	Ergebnis	Erläuterung
DISP "ABC"> "ABD" ENTER	0	Bis zum zweiten Zeichen gleich. Der ASCII-Wert des "C" auf der linken Seite ist kleiner als der Wert von "D" rechts, daher ist die Relation nicht erfüllt und das Ergebnis gleich 0.
DISP "MEMORY" >"MEMORIAL" ENTER	1	Bis zum fünften Zeichen gleich. Der Codewert des sechsten Zeichens "Y" ist größer als der Wert von "I", also ist die Bedingung erfüllt und das Ergebnis gleich 1. (In letzterem Beispiel ist die Anzahl der Zeichen pro Operand unwichtig.)

ANMERKUNG:

In der Mathematik wird das Ergebnis einer Vergleichsoperation (Ergebnis = wahr oder falsch) nicht als Zahlenwert, sondern als Wert einer logischen oder booleschen Größe aufgefaßt. Da aber diesem BASIC logische Werte als Zahlenwerte 0 der 1 dargestellt werden und sich auch weiter wie numerische Daten behandeln lassen, soll in diesem Handbuch der Begriff "numerischer Ausdruck" sowohl arithmetische als auch Vergleichsausdrücke umfassen. Siehe auch Seite 30.

4) Logische Operationen

Für logische Operationen stehen die Operatoren AND (log. "und"), OR (log. "oder") und NOT (Negation) zur Verfügung. AND und OR verknüpfen je zwei numerische Ausdrücke, während NOT sich auf nur einen Ausdruck bezieht. (Wenn ein Vergleichsoperator Teil eines numerischen Ausdrucks ist, so muß der ganze Ausdruck in Klammern eingeschlossen werden.) Die logischen Operatoren verarbeiten immer 16-bit-Binärdaten, so daß die Werte der beteiligten numerischen Ausdrücke zwischen −32767 (= 1111111111111111 binär) und +32767 (= 0111111111111111 binär) je einschließlich liegen dürfen; das höchstwertige Bit zeigt das Vorzeichen an (0 für positiv, 1 für negativ). Nachkommastellen werden vor der Konvertierung der Operanden in das 16-bit-Format abgeschnitten.

a) Numerischer Ausdruck AND numerischer Ausdruck

Berechnung des logischen Produkts der beiden numerischen Ausdrücke.

Wenn beiden Operanden Vergleichsausdrücke sind, ist das Ergebnis nur dann 1, wenn beiden Operanden "wahr" sind.

BEISPIEL:

1. `DISP 12 AND 10` ENTER

In diesen Beispiel werden 12 und 10 in Binärdaten konvertiert und bitweise deren logisches Produkt gebildet, das Ergebnis erscheint (in dezimaler Schreibweise) auf dem Bildschirm. Die Berechnung läuft wie folgt:

```
0000000000001100 ..... 12 in 16-bit-Binärdarstellung
                 ××××
0000000000001010 ..... 10 in 16-bit-Binärdarstellung
                 ‖ ‖ ‖ ‖
0000000000001000 ..... logisches Produkt (= 8 dezimal)
Produkt = 0      dez.8
```

Die logische Multiplikation geschieht nach folgenden Regeln: 1 x 1 = 1, 1 x 0 = 0 x 1 = 0, 0 x 0 = 0; bitweise ohne Überträge.

2. `DISP (3>1) AND (6>=4)` ENTER

Beide Vergleichsausdrücke sind "wahr", als Ergebnis erscheint 1 auf dem Bildschirm.

ANMERKUNG:

Tritt der AND-Operator im Bedingungsteil einer IF-Anweisung auf (siehe Seite 141), kann anstelle von AND auch der Multiplikationsoperator * benutzt werden.

b) Numerischer Ausdruck OR numerischer Ausdruck

Berechnung der logischen Summe der beiden numerischen Ausdrücke. Sind beide Operanden Vergleichausdrücke, so ist das Ergebnis 1, wenn einer oder beide Operanden "wahr" sind.

BEISPIEL:

1. `DISP 6 OR 10` ENTER

In diesem Beispiel werden 6 und 10 in Binärdaten konvertiert, bitweise deren logische Summe gebildet und das Ergebnis auf dem Bildschirm in dezimaler Darstellung ausgegeben. Die Berechnung geschieht wie folgt:

```
0000000000000110 ..... 6 in 16-bit-Binärdarstellung
                 ++++
0000000000001010 ..... 10 in 16-bit-Binärdarstellung
                 || || ||
0000000000001110 ..... logische Summe (= 14 dezimal)
Summe = 0        dez.14
```

Die logische Summe wird bitweise nach folgenden Regeln bestimmt: 1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, 0 + 0 = 0; Überträge werden ignoriert.

2. `DISP (6>8) OR (7<>1)` ENTER

In diesem Beispiel ist der rechte der beiden Vergleichsausdrücke wahr, und als Ergebnis wird 1 aus dem Bildschirm ausgegeben.

ANMERKUNG:

Tritt der OR-Operator im Bedienungsteil einer IF-Anweisung auf (siehe Seite 141), kann anstelle von OR auch der Additions operator + benutzt werden.

c) NOT numerischer Ausdruck

Berechnet das logische Komplement des numerischen Ausdrucks.

BEISPIEL:

1. `DISP NOT (-1234)` ENTER

–1234 wird ins Binärsystem konvertiert, das Komplement gebildet (andere Sprechweise: der Operand wird negiert) und das Ergebnis auf dem Bildschirm angezeigt. Die Berechnung geschieht wie folgt:

```
1000010011010010 ..... –1234 in 16-bit-Binärdarstellung
0111101100101101 ..... das Komplement (= 31555 dezimal)
```

Die Berechnung des Komplements*) geschieht also einfach dadurch, daß alle Bits negiert bzw. invertiert werden, d.h. 0-Bits werden durch 1-Bits und umgekehrt ersetzt.

2. `DISP NOT (8>=7)` ENTER

In diesem Beispiel ist die Vergleichsbedingung erfüllt, der Wert des eingeklammerten Ausdrucks ist 1. Das Komplement von 1 ist nach obigen Regeln –32766 (= 1111111111111110 binär), also nicht 0 (!).

ANMERKUNG:

–0 (1000000000000000 binär) wird als 0 (0000000000000000 binär) aufgefaßt.

*) Einerkomplement: Der Wert, den man durch Subtraktion jedes Bits von 1 erhält; 1 – 0 = 1, 1 – 1 = 0

3. DISP NOT ((-3) AND (-4)) [ENTER]

In diesem Beispiel wird –32767 ausgegeben (nicht 32767), wie nachfolgend dargestellte Rechnung zeigt:

1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 ..... –3 in 16-bit-Binärdarstellung
1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ..... –4 in 16-bit-Binärdarstellung
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ..... das logische Produkt ist gleich –0

–0 wird in 0 konvertiert. Das Komplement von 0 wird berechnet, das Ergebnis ist –32767 (= 1111111111111111 binär).

5) Dieses System stellt Funktionen für eine Reihe besonderer Operationen zur Verfügung. Einzelheiten siehe 2. "Funktionen", Seite 23.

6) Berechnungen mit Hilfe von Variablen

Einzelheiten zu Variablen finden sich in (4) "Variablen", siehe Seite 11. Für die folgenden Beispiele genügt es, sich unter einer "Variablen" einfach den Speicherplatz vorzustellen, wo die Daten abgelegt werden. Mit Hilfe der Angabe Variable = numerischer Ausdruck (String) wird der Variablen ein Zahlenwert (eine Zeichenkette) zugewiesen. Man beachte, daß damit keine Gleichung (im mathematischen Sinn) der Art gemeint ist, daß die linke Seite gleich der rechten Seite sei. Vielmehr wird hiermit ein Transport von Daten, nämlich des Wertes des Ausdrucks auf der rechten Seite, auf die links stehende Variable, bewirkt!

BEISPIEL:

1. Wertzuweisungen

Zuweisung	Tastenfolge
A←1. 234	A=1.234 [ENTER]
A←A+1	A=A+1 [ENTER]
X←B	X=B [ENTER]
$C \leftarrow \sqrt{A^2+B^2}$	C=SQR (A^2+B^2) [ENTER]
B←A、C←A	B=A:C=A [ENTER]

2. Trigonometrische Funktionen mit Variablen (Einheit Grad)

Operation	Tastenfolge	Anzeige
X←30	X=30 [ENTER]	30
sinX	DISP SIN X [ENTER]	0.5
cosX	DISP COS X [ENTER]	0.8660254038
tanX	DISP TAN X [ENTER]	0.5773502692

7) Der Stapel (Stack)

Soll zum Beispiel 23 – 11 * SQR 4 berechnet werden, so wird zuerst SQR 4 bestimmt. Da aber der Ausdruck von links nach rechts abgearbeitet wird, muß 23 – 11 * irgendwie zwischengespeichert werden. Der dafür reservierte Speicherbereich heißt "Stapel" (Stack) oder "Keller". Es gibt zwei solcher Stapel, nämlich den Operandenstapel für Operanden wie z.B. 23 und 11, und den Operatorenstapel für Operatoren wie – und *. Der Operandenstapel hat 25 Plätze, der Operatorenstapel 48. Man beachte das Fassungsvermögen der Stapel beim Niederschreiben numerischer Ausdrücke!

BEISPIEL:

Datenbewegungen im Stapel bei der Berechnung von

10+3*SQR 7-LOG TAN(12+68)^2^3/EXP 2 [ENTER]

Operation	Operandenstapel				Operatorenstapel				
	Platz 1	Platz 2	Platz 3	Platz 4	Platz 1	Platz 2	Platz 3	Platz 4	Platz 5
10	10								
+	10				+				
3	3	10			+				
*	3	10			*	+			
SQR	3	10			SQR	*	+		
7	7	3	10		SQR	*	+		
−	1.793···E1				−				
LOG	1.793···E1				LOG	−			
TAN	1.793···E1				TAN	LOG	−		
(	1.793···E1				(	TAN	LOG	−	
12	12	1.793···E1			(	TAN	LOG	−	
+	12	1.793···E1			+	(	TAN	LOG	−
68	68	12	1.793···E1		+	(	TAN	LOG	−
)	80	1.793···E1			TAN	LOG	−		
^	7.536··E−1	1.793···E1			^	−			
2	2	7.536···E1	1.793···E1		^	−			
^	2	7.536··E−1	1.793···E1		^	^	−		
3	3	2	7.536··E−1	1.793···E1	^	^	−		
/	1.832··E−1	1.793··E−1			/	−			
EXP	1.832··E−1	1.793··E−1			EXP	/	−		
2	2	1.832··E−1	1.793···E1		EXP	/	−		
[ENTER]	1.792···E1								

8) Löschen und Korrigieren von Eingabedaten, Aufheben der Fehlersperre

Tippfehler bei manuellen Operationen lassen sich entweder durch Drücken der Löschtaste [CL] oder durch Bewegen des Kursors mit den Tasten [←] oder [→] beheben.

a) Löschen mit der [CL] -Taste

Durch Antippen der [CL] -Taste wird die gesamte Eingabe gelöscht, und man kann die Daten erneut eingeben. Die [CL] -Taste bewirkt außerdem das Aufheben des Fehlerstatus.

b) Korrekturen mit den Kursortasten [←] und [→]

Nach dem Drücken von [CL] muß man alle gelöschten Daten von Anfang an neu eintippen. Die Kursortasten [←] und [→] hingegen gestatten die gezielte Korrektur einzelner Fehler.

○ Ist ein Tipp- oder Bedienungsfehler die Ursache für einen Fehlerstopp, so wird eine Fehlersperre gesetzt, d.h. BASIC nimmt keine Eingaben mehr an, bevor der Bediener nicht auf die Fehlersituation reagiert hat. Die Fehlersperre ist durch Drücken von [CL] oder [↑] aufzuheben, dann kann mit der Korrektur begonnen werden.

c) Korrektur mit Hilfe der [↑] -Taste
Antippen der [↑] -Taste bewirkt eine Aufhebung der Fehlersperre. Gleichzeitig wird die fehlerhafte Zeile am Bildschirm ausgegeben und der Kursor an den Ort des Fehlers oder unmittelbar dahinter positioniert, so daß sofort korrigiert werden kann. Betätigt man die [↑] -Taste im normalen (Nicht-Fehler-) Status, wird die letzte Eingabezeile auf dem Bildschirm wieder ausgegeben und kann geändert werden.

(3) Konstanten

Unter "Konstanten" versteht man Datenelemente mit festem Inhalt.

1) Numerische Konstanten
Das System kennt vier Arten numerischer Konstanten.

a) Dezimalformat

BEISPIEL:
```
123
-12.34
.567 ..... die führende Null darf weggelassen werden (0.567)
2345.0
```

b) Exponentialformat
Der Exponententeil wird durch E eingeleitet.

BEISPIEL:
```
1.2E+13 ..... steht für 1.2 x 10^13 (+ im Exponenten kann entfallen)
4.56E-8 ..... steht für 4.56 x 10^-8
21.3E15 ..... steht für 21.3 x 10^15
```

c) Die "Kreiszahl" π (Verhältnis von Kreisumfang zu Kreisdurchmesser) steht als PI in diesem System zur Verfügung. Der Wert von PI = 3.14159265358 ist fest im Speicher abgelegt. (Man tippe `PI` [ENTER]).

BEISPIEL:
Der Umfang eines Kreises mit einem Radius von 5 cm kann wie folgt berechnet werden:
`2*PI*5` [ENTER]

d) Hexadezimale Schreibweise

Hexadezimale Zahlen werden mit einem kaufmännischen "und" (&) eingeleitet. In diesem System können Hexadezimalzahlen zwischen &0 und &FFFF je einschließlich benutzt werden.

Hexadezimal	Dezimal
&0	0
&1	1
&2	2
&3	3
⋮	⋮
&A	10
&B	11
&C	12
&D	13
&E	14
&F	15
&10	16
⋮	⋮
&1F	31
&20	32
⋮	⋮
&FFFF	65535

2) Stringkonstanten (Text- oder Zeichenkonstanten)

Eine Stringkonstante besteht aus einer Kette von einem oder mehreren Buchstaben, Ziffern oder anderen Zeichen, die durch Anführungszeichen (") oder Apostrophe(') eingeschlossen wird. Eine Stringkonstante wird genau so benutzt, wie sie hingeschrieben wurde.

BEISPIEL:

```
"GUTEN MORGEN"
"DM 1800"
'Endsumme ='
'2+5'.....hier werden nicht etwa 2 und 5 addiert, sondern '2+5' ist eine
          Stringkonstante! Zum besseren Verständnis:
```

```
>DISP 2+5
 7
>DISP '2+5'
2+5
```

WICHTIGER HINWEIS

Alle Zeichenfolgen, die sich über die Tastatur eingeben lassen, können als Stringkonstanten Verwendung finden. Es können jedoch keine Anführungszeichen in einer von Anführungszeichen abgegrenzten Stringkonstanten benutzt werden, ebensowenig dürfen Apostrophe in einer von Apostrophen eingeschlossenen Stringkonstanten auftauchen!

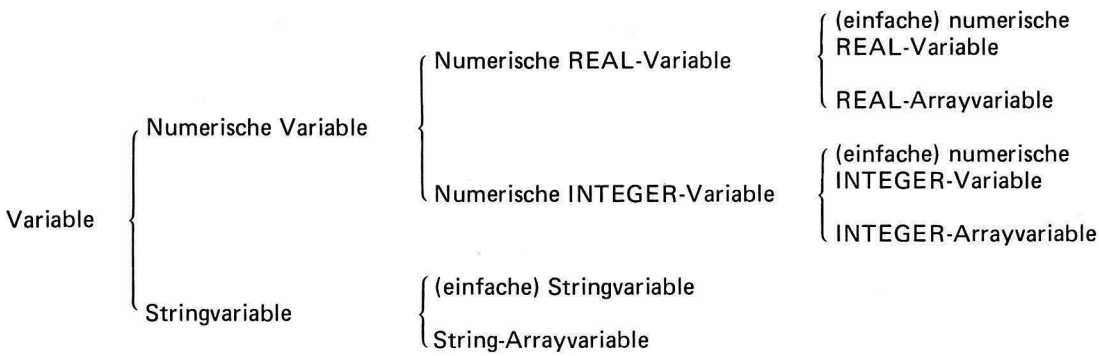
BEISPIEL:

```
"sie sagte "nein!""   ist falsch
'sie sagte "ja!"'     ist richtig
```

(4) Variablen

"Variablen" nennt man diejenigen Speicherplätze, in denen u.a. die im vorigen Abschnitt beschriebenen numerischen oder Stringkonstanten abgelegt (zugewiesen) werden. Variablen zum Speichern numerischer Werte heißen numerische oder arithmetische Variablen, solche zum Speichern von Strings Stringvariablen. Numerische Variablen lassen sich weiter unterteilen in Variablen von Typ "REAL" (Gleitpunktvariablen) und solche vom Typ "INTEGER" (ganzzahlige Variablen). Ist von einer numerischen Variablen die Rede, sind im allgemeinen REAL-Variablen damit gemeint. Andererseits unterscheidet man zwischen einfachen Variablen und Array-Variablen, siehe auch 1. (8) "Array-Variablen", Seite 19. "Variablen" bezeichnet im allgemeinen einfache Variablen; ein Brauch, der auch in diesem Handbuch geübt werden soll.

Folgendes Schema veranschaulicht die Organisation der Variablen:



Variablen werden über einen ihnen zugeordneten Namen, den "Variablennamen", angesprochen. Variablennamen müssen nach folgenden Regeln gebildet werden:

a) Ein Variablenname besteht aus 1 bis 8 Zeichen.
(Das 9. und nachfolgende Zeichen werden ignoriert.)

b) Das erste Zeichen eines Namens muß ein Buchstabe sein.

c) Das zweite und alle folgenden Zeichen müssen alphanumerisch sein, d.h. es sind nur Buchstaben und Ziffern zugelassen.

d) Namen von Funktionen (z.B. SIN), Befehle, Anweisungen und andere Schlüsselworte sind nicht als Variablennamen oder Bestandteile von Variablennamen zugelassen.

BEISPIEL:

(Richtig)
```
ABC
E1
antwort
Endsumme
AnachB1
```

(Falsch)
```
RADIUS ·····enthält das Schlüsselwort RAD
A* ····· einhält den Operator *.
INTEGRAL ···enthält den Namen der INT-Funktion
```

1) Numerische Variablen (einfache numerische Variablen)
Unter numerischen Variablen werden Variable vom Typ INTEGER und solche vom Typ REAL zusammengefaßt. Die Definition des Typs geschieht durch Anhängen des Typ-Definitionssymbols an den Variablennamen.

a) Numerische Variablen vom Typ REAL
Ein Variablenname wird durch Anhängen eines Doppelkreuzes (#) als Name einer REAL-Variablen definiert. Das Doppelkreuz kann normalerweise weggelassen werden (im Grundzustand nach dem Start von BASIC).
REAL-Variablen (wie man kurz sagt) können positive und negative Werte mit bis zu 12 signifikanten Stellen sowie den Wert 0 annehmen, der Exponent kann zwischen –99 und +99 je einschließlich liegen.

BEISPIEL:

```
A=-12.34 ..... Zuweisung von 12.34 an die REAL-Variable A.
Summe#=2300 ..... Zuweisung von 2300 an die REAL-Variable Summe #.
MITTEL#=-.031 ..... Zuweisung von 0.031 an die REAL-Variable MITTEL #.
                    (Zur Demonstration wurde das Doppelkreuz bei A weggelassen).
```

b) Numerische Variablen vom Typ INTEGER
Variablen werden durch Anhängen eines Prozentzeichens (%) an den Namen als vom Typ INTEGER definiert. INTEGER-Variablen (Kurzsprechweise) können ganzzahlige Werte zwischen –32767 und 32767 je einschließlich annehmen. Sie belegen weniger Speicher als REAL-Variablen, und Berechnungen mit ganzen Zahlen sind schneller als solche mit REAL-Zahlen. Daher wird die Benutzung von INTEGER-Variablen empfohlen, wann immer mit ganzen Zahlen im o.a. Bereich gearbeitet wird!

BEISPIEL:

```
ABC%=24 ..... Zuweisung von 24 an die INTEGER-Variable ABC%
I2%=-32700 ..... Zuweisung von –32700 an die INTEGER-Variable I 2%
ANSWER%=76.21 ..... Zuweisung von 76 an die INTEGER-Variable answer%
                    Da answer% vom Typ INTEGER ist, werden die Nachkommastellen abgeschnitten.
```

2) Stringvariablen (einfache Stringvariablen)
Durch Anhängen eines Dollar-Zeichens ($) oder eines "at"-Symbols (@, auch "Klammeraffe" genannt) wird ein Variablenname als Name einer Stringvariablen definiert.

BEISPIEL:

```
AB$="B.HANDBUCH" ..Zuweisung von "B.HANDBUCH" an die Stringvariable AB$.
DATUM@="01.09.83"..Zuweisung von "01.09.83" an die Stringvariable DATUM@.
```

WICHTIGER HINWEIS

Einer (einfachen) Stringvariablen kann ein String aus bis zu 16 Zeichen zugewiesen werden.

ANMERKUNG:

Die Unterschiede zwischen den mit $ gekennzeichneten Stringvariablen und denen mit @ sind wie folgt:

1. Das ASCII-Steuerzeichen "NULL" mit der Wertigkeit 0 (z.B. mit CHR$ 0 erzeugt, siehe CHR$-Funktion auf Seite 23) kann nur an @-Stringvariablen zugewiesen werden.

BEISPIEL:

```
A@=CHR$ 0 ..... NULL wird der Variablen A@ zugewiesen.
A$=CHR$ 0 ..... der Variablen A$ wird NULL nicht zugewiesen!
```

2. Normalerweise können sowohl den $- als auch den @-Variablen Strings aus bis zu 16 Zeichen zugewiesen werden. Darüber hinaus besteht auch bei einfachen @-Stringvariablen die Möglichkeit, die maximale Zahl von Zeichen mit Hilfe der DIM-Anwseisung (Seite 128) vorzugeben. (Mit der DIM-Anweisung läßt sich die maximale Anzahl von Zeichen bei String-Arrayvariablen beider Typen $ und @ festlegen.)

BEISPIEL:

```
DIM A@*10 ..... Deklaration der Stringvariablen A@ zusammen mit der
                Vorgabe von maximal 10 Zeichen Kapazität.
```

3. Die $-Stringvariablen belegen weniger Speicher als die vom @-Typ. Daher wird die Benutzung von $-Variablen empfohlen, wenn das NULL-Zeichen nicht benötigt wird.

4. Der Inhalt von $-Stringvariablen kann @-Variablen zugewiesen werden.

BEISPIEL:

```
A@=A$ ..... Der Variablen A@ wird der Inhalt von A$ zugewiesen.
```

3) Definition von Variablentypen

Die Typdefinition von numerischen REAL- und INTEGER-Variablen sowie der beiden Typen von Stringvariablen ($ und @) mit Hilfe der an den Namen angehängten Definitionssymbole wurde in den obigen Abschnitten 1) Numerische Variablen und 2) Stringvariablen beschrieben.

Eine andere Methode zur Definition von Variablentypen besteht in der Anwendung der Anweisungen DEFREAL, DEFINT und DEFSTR. Diese Anweisungen können sowohl manuell als auch in Programmen benutzt werden. Einzelheiten siehe Seite 206 (DEFINT), 207 (DEFREAL) und 208 (DEFSTR).

4) Numerische Sondervariablen (Pseudovariablen)

Es gibt die folgenden numerischen Sondervariablen (Pseudovariablen), die jedoch nicht als allgemeine Variablen zu verwenden sind.

a) TIME

Gibt die Uhrzeit an. Der ganzzahlige Teil enthält Monat, Tag und Stunde (24-Stunden-Zählung), die Nachkommastellen bedeuten Minuten und Sekunden (im Sechzigersystem). Zum Beispiel ist 40813.5035 zu lesen als 8. April (4. Monat), 13 Uhr 50 Minuten 35 Sekunden.

Mehrfaches Eingeben von `DISP TIME` [ENTER] . zeigt, daß der Wert von TIME sich ändert. Um TIME auf einen bestimmten Wert zu setzen, z.B. den 27. Januar, 16 Uhr 47 Minuten 37 Sekunden, lautet die Eingabe `TIME=50510.583` [ENTER] . Die neue Zählung der Zeit beginnt beim Betätigen der [ENTER] -Taste.

b) ERL (Error line number = Nummer der Fehlerzeile)
Gibt die Zeilennummer an, in der ein Fehler beim Ablauf eines Programms auftrat. Näheres siehe II. 1. (1) "Programme", Seite 61.

c) ERN (Error code number = Fehlernummer)
Gibt die Fehlernummer an, wenn ein Fehler auftrat. Die Fehlernummern sind in Abschnitt IV "Fehlernummern" (Seite 51) im Anhang zum BASIC-Handbuch verzeichnet.

BEISPIEL:

Man gebe an der Tastatur ein:

3/0 [ENTER]

Ein Fehler tritt auf, und in der unteren Bildschirmzeile wird ERROR 3 angezeigt. Nach dem Lösen der Fehlersperre mit der [CL] -Taste gebe man

ERN [ENTER]

ein. Die Anzeige lautet 3 (der Inhalt der Pseudovariablen ERN).
Fehlernummer 3 weist auf eine "Division durch Null" hin – siehe Liste der Fehlernummern. Auch die beim Auftreten des Fehlers in der unteren Bildschirmzeile angezeigte 3 ist die Fehlernummer.

(5) Numerische und Stringausdrücke

1) Numerische Ausdrücke
Folgende numerische Ausdrücke können in diesem System benutzt werden:

Typ	Anwendungsbeispiel	Seite
Numerische Konstante	A=-12.34	9
Numerische Variable	A=B	12
Numerische Funktion	A=SIN 30 B=ABS(COS X)	26
ASC-Funktion	A=ASC "J"	23
VAL-Funktion	A=VAL "1,234.5"	24
LEN-Funktion	A=LEN "ENGLAND"	24
FN Funktionsname (numer. Ausdruck) FN Funktionsname (Stringausdruck) (Ergebnisse numerischen Typs)	A=FN M(5,X-4) A=FN Q("VERSUCH")	199
Numer. Ausdrukck arithmet. Operator Numer. Ausdruck	A=25-LEN B$/2 B=5.6^3	3
Numer. Ausdruck Vergleichsoperator Numer. Ausdruck String Vergleichsoperator String	A=B<=C+6 X="NAME">"NAMEN"	3
Numer. Ausdruck AND numer. Ausdruck Numer. Ausdruck OR numer. Ausdruck NOT numer. Ausdruck	A=13 AND 11 B=6 OR 13 C=NOT (-1234)	5
Vorzeichen numer. Ausdruck	A=+SIN(A+B) B=-ATN 2.5	3

2) Strings

Die folgende Tabelle zeigt Beispiele für Stringausdrücke, wie sie auf diesem System benutzt werden können:

Typ	Anwendungsbeispiel	Seite
Stringkonstante	A$="VERKAUF"	10
Stringvariable	A$=B$	12
CHR$-Funktion	A$=CHR$ 37	23
STR$-Funktion	A$=STR$ (5+2*K)	24
STRING$-Funktion	A$=STRING$ (10,'=')	24
LEFT$-Funktion	A$=LEFT$(B$,4)	25
RIGHT$-Funktion	A$=RIGHT$(B$,6)	25
MID$-Funktion	A$=MID$(B$,5,2)	25
SPA-Funktion	A$=SPA(3*N)	25
FN Funktionsname (numer. Ausdruck) FN Funktionsname (Stringausdruck) (Ergebnisse vom Typ String)	A$=FN B$(5,3,Y$)+Z$ A$=FN C$("MORGEN"+Q$)	199
String + String	A$=B$+RIGHT$(C$,5)	104

Im folgenden wird ein Stringausdruck kurz als "String" bezeichnet.

(6) Arithmetische Vorrangregeln

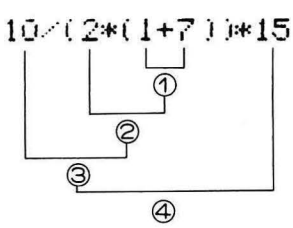
Arithmetische Ausdrücke werden in der Reihenfolge der unten aufgeführten Vorrangstufen abgearbeitet.

Höchste Priorität	Klammern (und)
↓	Funktionsaufrufe und NOT
↓	Exponentiation
↓	Vorzeichen + und –
↓	*, / und AND
↓	+, – und OR
Niedrigste Priorität	Vergleichsoperatoren =, <, >, <=, >= und < >

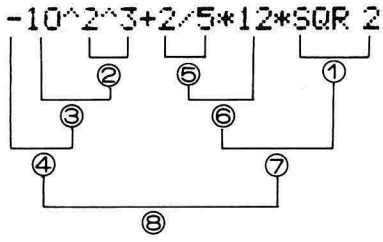
Operationen derselben Priorität werden von links nach rechts abgearbeitet. Folgen jedoch mehrere Exponentiationen (^) direkt aufeinander, so werden nur diese für sich von rechts nach links ausgeführt. Ausdrücke in Klammern werden zuerst ausgerechnet, so daß hier die normale Reihenfolge unterbrochen ist. Ist in Zweifelsfällen die Reihenfolge der Berechnung nicht klar, so sollte man Klammern setzen.

BEISPIEL:

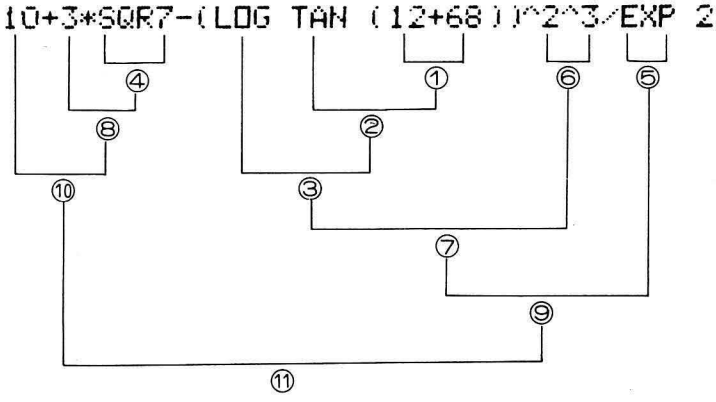
1. 10 ÷ (2 x (1 + 7)) x 15 wird wie folgt berechnet:



2. $-10^{2^3} + 2 \div 5 \times 12 \times \sqrt{2}$ wird wie folgt berechnet:



3. $10 + 3 \times \sqrt{7} - \{\log \tan (12 + 68)\}^{2^3} \div e^2$ wird so berechnet:



(7) Ausgabeformate für Zahlen

Für die Ausgabe von numerischen Werten auf Bildschirm und Drucker stehen drei verschiedene Formate zur Verfügung:

① Standardformat
② Festpunktformat
③ Fließpunktformat

Nach dem Einschalten des Systems oder der Ausführung einer NEW-Anweisung (siehe Seite 92) wird automatisch das Standardformat gewählt.

Die Zahlendarstellung läßt sich mit den Anweisungen FIXED auf Festpunktformat, mit FLOAT auf Gleitpunktformat und mit NORMAL auf Standardformat umschalten. Alle Formatwahl-Anweisungen können sowohl in Programmen als auch bei manuellen Operationen benutzt werden.

1) Standardformat

Im Standardformat wird je nach dem auszugebenden Wert automatisch ein Fest- oder Fließpunktformat so gewählt, daß die Darstellung möglichst leicht lesbar ist. Nicht signifikante Stellen werden abgeschnitten.

FORMAT: NORMAL

ERLÄUTERUNG:

1. Die Zahlen werden mit maximal 12 signifikanten Stellen ausgegeben. Die 13. und folgende Stellen werden abgeschnitten.
2. Führende Nullen und nicht signifikante Nullen hinter dem Dezimalteil werden unterdrückt.
3. Je nach dem Wert der Zahl wird ein passendes Fest- oder Fließpunktformat gemäß folgenden Regeln gewählt. Für die auszugebende Zahl steht x.

0 und alle Zahlen im Bereich $10^{-4} \leqq |X| < 10^{12}$
Zahlen im Bereich $10^{-11} \leqq |X| < 10^{-4}$, wenn die Anzahl signifikanter Dezimalen klein genug ist } Festpunktformat

Alle anderen Zahlen ..... Fließpunktformat

BEISPIEL:

Nach der Ausführung von NORMAL (bzw. NEW oder dem Einschalten)

Wert im Speicher	Ausgabeformat	
123456789012	123456789012	
-123456789012	-123456789012	
0.12345678901	0.12345678901	Festpunktformat
1.23E-9	0.00000000123	
12.34567890123	12.3456789012	11. Dezimale abgeschnitten
-1.00	-1	Nachlaufende Nullen unterdrückt.
0.000056789012	5.6789012E-05	Fließpunktformat.
-1234567890123	-1.23456789012E+12	

2) Festpunktformat

Im Festpunktformat bleibt die Anzahl der Dezimalen (Nachkommastellen) konstant. Die Anzahl der Dezimalen und die Rundung können vorgegeben werden. Mit der Anweisung FIXED wird das Festpunktformat angewählt.

FORMAT: FIXED m. n (FIXED numerischer Ausdruck)

m: Anzahl der Nachkommastellen (ganze Zahl von 0 – 11)

n : Rundungs-Vorgabe (ganze Zahl 0 – 9)

n	Rundung
0	abrunden
1	8 ab, 9 auf
2	7 ab, 8 auf
3	6 ab, 7 auf
4	5 ab, 6 auf
5	4 ab, 5 auf
6	3 ab, 4 auf
7	2 ab, 3 auf
8	1 ab, 2 auf
9	aufrunden

ERLÄUTERUNG:

1. Es wird die (m + 1) – te Dezimal gemäß der Vorgabe n gerundet. Die Zahl wird bis auf die weiter unten genannten Ausnahmen immer mit m Dezimalen ausgegeben.
2. Übersteigt der absolute Betrag der auszugebenden Zahl 10, so wird trotz des angewählten Festpunktformats (FIXED m.n) vorübergehend auf ein Fließpunktformat (FLOAT m.n) umgeschaltet.
3. Im Format FIXED 0.n wird der Nachkommaanteil unterdrückt.
4. Wenn (bei großen Zahlen) die spezifizierte Anzahl m (maximal 11) der Dezimalen plus der Anzahl der Stellen vor dem Dezimalpunkt größer als 12 ist, kann die Anzahl der ausgegebenen Stellen von der Formatspezifikation abweichen oder die Rundung unterbleiben.

BEISPIEL:

Die Anweisung FIXED 3.5 sei ausgeführt (also Rundung auf der 4. Dezimalen, Ausgabe der Zahlen mit 3 Dezimalen.)

Wert im Speicher	Ausgabeformat	
12	12.000	
-123.45678	-123.457	
0.1296	0.130	
1234567890123	1.235E+12	···· Absolutbetrag > 10^{12}, daher Fließpunktformat.
1.234E-4	0.000	
1234567890.5699	1234567890.56	···· Zuviele Stellen, überflüssige (mehr als 12) hier ohne Rundung abgeschnitten

3) Fließpunktformat

Das Fließpunktformat ist nützlich zur Behandlung sehr größer und sehr kleiner Zahlen. Es wird mit der Anweisung FLOAT angewählt.

FORMAT: FLOAT m.n. (FLOAT numerischer Ausdruck)

m: Anzahl der Nachkommastellen (ganze Zahl von 0 – 11)

n : Rundungs-Vorgabe (ganze Zahl 0 – 9)

Zur Bedeutung von n siehe Festpunktformat, Seite 17. Die Darstellung von Zahlen im Fließpunktformat hat folgendes Aussehen:

(–) d.d. . . . . dE ± dd

↓ ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾ ‾‾‾‾

Vorzeichen Mantisse Exponent

In der mathematisch üblichen Schreibweise bedeutet dies

(–) d.d. . . . . d x $10^{\pm dd}$

↓ ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾ ‾‾‾‾

Vorzeichen Mantisse Exponent

ERLÄUTERUNG:

1. Die erste von Null verschiedene Stelle der Zahl wird als erste Stelle der Mantisse ausgegeben. Ist die Zahl negativ, steht ein Minuszeichen vor der Mantisse, vor Null und positiven Zahlen steht ein Leerzeichen.
2. Im Format FLOAT 0.n wird der gesamte Nachkommaanteil einschließlich des Dezimalpunktes unterdrückt. In allen anderen Fällen folgt ein Dezimalpunkt auf die erste Mantissenstelle.
3. Auf den Dezimalpunkt folgen so viele Dezimalen wie in der Vorgabe m spezifiziert.
4. Der Exponent besteht aus einem Vorzeichen (+ oder –) und einer zweistelligen Zahl, die als Exponent von 10 aufzufassen ist. Ist der Exponent = 00, so wird er mit positivem Vorzeichen dargestellt.

BEISPIEL:

Das Format sei auf FLOAT 2.9 eingestellt, also Rundung auf der 3. Dezimale der Mantisse und Ausgabe der Zahlen mit 2 Dezimalen in der Mantisse.

Wert im Speicher	Ausgabeformat
123.45	1.24E+02
-3.2	-3.20E+00
0.00000123	1.23E-06
9876543210 1	9.88E+10
5E6	5.00E+06

(8) Array-Variable

Hat man es mit sehr vielen gleichartigen Daten zu tun, so kann die Verarbeitung recht unübersichtlich werden, wenn für jedes Datenelement eine eigene Variable benutzt wird. In solchen Fällen bieten sich Arrayvariablen als sehr nüzlich an.

1) Was ist eine Array-Variable ?

Unter einer Arrayvariablen oder Feldvariablen (kurz: Array oder Feld) versteht man eine Variable, die unter einem einzigen Namen zur selben Zeit mehrere Daten aufnehmen kann. Sie unterscheidet sich von einfachen Variablen durch eine an den Variablennamen angehängte, in Klammern eingeschlossene Zahl, den sog. "Index". Die der Arrayvariablen zugewiesenen Werte werden über den jeweiligen Index angesprochen. Will man z.B. dem Array A zehn Werte zuweisen, läßt man den Index von 1 bis 10 laufen und weist die einzelnen Daten A(1), A(2), . . . , A(10) zu. A(1), A(2) usw. nennt man "Array-elemente" oder "Feldelemente". Eine Arrayvariable setzt sich also aus einzelnen Arrayelementen zusammen, die je für sich genau einen Wert zur Zeit aufnehmen können.

Außer numerischen Konstanten können auch numerische Variablen, numerische Arraylemente, ja beliebig komplizierte numerische Ausdrücke als Index eines Arrayelements benutzt werden, was für die Anwendung von Arrays sehr wichtig ist!

BEISPIEL:

```
A(I)
A(3*I)
```

In den Anwendungsbeispielen zur DIM-Anweisung (Seite 128) finden sich weitere Beispiele für Indexausdrücke, auf die beim Studium der folgende Ausführungen Bezug genommen werden sollte.

WICHTIGER HINWEIS

Eine einfache Variable und eine Arrayvariable dürfen den gleichen Namen haben, sie bezeichnen trotzdem völlig verschiedene Speicherplätze! So können z.B. A(1) und A gleichzeitig verwendet werden, ohne daß es irgendwelche Konflikte geben kann.

2) Deklaration von Arrayvariablen

Bevor ein Array angesprochen werden kann, muß (am Programmanfang) die Arrayvariable mit Hilfe der DIM-Anweisung deklariert werden. In der DIM-Anweisung wird gleichzeitig der maximale Wert des Index (der Indizes) festgelegt und so der für das Array erforderlichen Platz im Speicher reserviert.

BEISPIEL:

```
DIM X(5)
```

deklariert das Array X mit fünf Elementen X(1) bis X(5).
In einer einzigen DIM-Anweisung können mehrere Arrayvariablen deklariert werden.

BEISPIEL:

```
DIM X(5),Y(10)
```

WICHTIGER HINWEIS

1. Es ist nicht zulässig, dieselbe Arrayvariable in mehreren DIM-Anweisungen mehrfach zu deklarieren. Wurde z.B. durch DIM C(10) das Array C deklariert, so würde DIM C(25) weder ein neues Array C mit 25 Elemente reservieren noch das vorherige Array C auf 25 Elemente vergrößern, sondern führt zum Programmabbruch (ERROR 20).
2. Ein Array kann maximal 32K Byte Speicher belegen, siehe auch Seite 130, Hinweis 1.
3. Der kleinste für den Indexausdruck zulässige Wert ist 1, d.h. die Zählung der Elemente eines Arrays geht immer von 1 bis zum in der DIM-Anweisung vorgegebenen Maximum.

3) Mehrdimensionale Arrayvariablen

Arrays mit einem einzigen Index wie in allen bisheringen Beispielen heißen "eindimensionale Arrays". Es können jedoch auch Arrays mit zwei oder noch mehr Indizes benutzt werden, sog. "zwei-" oder "mehrdimensionale Arrays". Die Indexausdrücke der einzelnen "Dimensionen" sind dabei durch Kommata zu trennen.

BEISPIEL:

```
A(3,5)·····zweidimensionales Array
ST(7,4,2)·····dreidimensionales Array
```

Genau wie eindimensionale Arrays müssen auch mehrdimensionale Arrays vor ihrer ersten Verwendung in einer Wertzuweisung mit Hilfe einer DIM-Anweisung deklariert werden.

BEISPIEL:

```
DIM Y(7,5)
```

deklariert das Array Y mit den folgenden 35 (= 7 x 5)

Y (1, 1)	Y (1, 2)	Y (1, 3)	Y (1, 4)	Y (1, 5)
Y (2, 1)	Y (2, 2)	Y (2, 3)	Y (2, 4)	Y (2, 5)
Y (3, 1)	Y (3, 2)	Y (3, 3)	Y (3, 4)	Y (3, 5)
Y (4, 1)	Y (4, 2)	Y (4, 3)	Y (4, 4)	Y (4, 5)
Y (5, 1)	Y (5, 2)	Y (5, 3)	Y (5, 4)	Y (5, 5)
Y (6, 1)	Y (6, 2)	Y (6, 3)	Y (6, 4)	Y (6, 5)
Y (7, 1)	Y (7, 2)	Y (7, 3)	Y (7, 4)	Y (7, 5)

4) Numerische INTEGER-Arrays und Stringarrays

Alles bisher über Arrays gesagte bezieht sich auf Arrayvariablen vom numerischen Typ REAL. Jedoch lassen sich auf ähnliche Art auch INTEGER- und Stringarrays verwenden.

BEISPIEL:

```
Hdd#(X,Y)
```
..... numerisches REAL-Array
(# kann normalerweise fortgelassen werden.)

```
C%(1,2,K)
```
..... numerisches INTEGER-Array

```
TEL$(A,B)
```
..... Stringarray vom Typ $

```
B@(M,N)
```
..... Stringarray vom Typ @

Die maximale Anzahl von Zeichen pro Element eines Stringarrays beträgt 16 (genau wie in einfachen Stringvariablen). In den DIM-Anweisung läßt sich diese Grenze jedoch ändern.

BEISPIEL:

```
DIM Name$(10)*30
```

Die der Indexklammer folgende Konstruktion *30 bewirkt, daß jedes Element des Arrays NAME$, also NAME$(1), NAME$(2), . . . , NAME$(10) je maximal 30 Zeichen aufnehmen kann. Es sind maximal 255 Zeichen pro Feldelement möglich (DIM Stringarray$(n)*255, siehe auch DIM-Anweisung auf Seite 128).

In manchen Anweisungen ist es möglich oder gar erforderlich, alle Elemente eines numerischen Arrays VEKTOR oder eines Stringarrays TEXT$ (diese Namen dienen nur zur Erläuterung) gleichzeitig anzusprechen. In diesem Fall schreibe man VEKTOR(*) bzw. TEXT$(*). Ein Stern anstelle eines Indexausdrucks bedeutet, daß nicht einzelne Elemente des betreffenden Arrays gemeint sind, sondern das gesamte Feld als solches.

Im Rahmen dieses Handbuches wird die Schreibweise Array(*) auch als "kollektive Arrayvariable" oder "kollektives Array" bezeichnet.

(9) Fehlermeldungen

Beim Auftreten eines Fehlers ertönt ein dreifaches akustisches Signal, und die Fehlernummer wird in der unteren Zeile des Bildschirms angezeigt. Die Fehlernummer gibt Aufschluß über die Art des Fehlers. Siehe auch Kapitel VI "Fehlernummernliste" auf Seite 51 des Appendix zum BASIC-Handbuch. Außerdem wird eine Fehlersperre gesetzt, d.h. das System nimmt keine Eingaben mehr an, bis der Benutzer auf die Fehlersituation reagiert hat.
Tritt ein Fehler während eines Programmlaufes auf, so wird die Ausführung des Programms abgebrochen und Fehlernummer sowie die Nummer der Programmzeile, in der der Fehler auftrat, wie oben beschrieben angezeigt.
So zeigt zum Beispiel

```
ERROR 4 IN LINE 100
      ↓          ↓
```

Fehlernummer Fehlerhafte Programmzeile

an, daß in Zeile 100 die Programmausführung unterbrochen wurde, weil hier ein Sprung zu einer Zeile mit einer nicht existierenden Nummer ausgeführt werden sollte.
Die akustische Fehlermeldung kann mit der ERROR MUSIC-Anweisung auf Wunsch aus- und eingeschaltet werden, siehe Seite 213. Das Lösen der nach dem Fehler gesetzten Sperre und die Korrekturmöglichkeiten sind auf Seite 8, Kapitel 1. (2) "Manuelle Betriebsart", beschrieben.

2. FUNKTIONEN

Der BASIC-Interpreter verfügt über ein reiches Repertoire von Funktionen zur Verarbeitung von Strings und numerischen Daten. Damit die Beschreibungen der Funktionen so knapp und klar wie möglich formuliert werden können, werden an dieser Stelle folgende Konventionen für die Argumente eingeführt. Unter "Argumenten" versteht man diejenigen Datenelemente, die einer Funktion beim Aufruf zur Verarbeitung übergeben werden.

Kurzzeichen	Steht für
x, y, z	beliebige numerische Ausdrücke vom Typ REAL.
m, n	beliebige numerische Ausdrücke, die jedoch vor der Verarbeitung innerhalb der Funktion in den Typ INTEGER konvertiert werden.
P$, Q$	beliebige Stringausdrücke.

Die als Argumente benutzten Ausdrücke dürfen im Allgemeinen beliebig kompliziert sein. Bei einzelnen Funktionen können jedoch Einschränkungen z.B. hinsichtlich des zulässigen Wertebereiches bestehen.
Unter dem "Wert" einer Funktion versteht man das Resultat, welches die Funktion nach ihrem Ablauf an das Programm zurückliefert.
Um den syntaktischen Regeln von BASIC zu genügen, wird in den Beispielen das Ergebnis des Funktionsaufrufes einheitlich einer einfachen Variablen A oder A$ zugewiesen, je nach dem Typ des Funktionswertes.

WICHTIGER HINWEIS

Wenn die Argumente von Funktionen Vorzeichen, arithmetische Operatoren und/oder Vergleichsoperatoren enthalten, so müssen sie in Klammern eingeschlossen werden!

(1) Stringfunktionen

Stringfunktionen verarbeiten Strings und numerische Ausdrücke, ihre Werte sind bis auf zwei Ausnahmen Strings.

a) ASC P$

Wandelt Strings in (numerische) Zeichenwerte um. Der Wert der ASC-Funktion ist der ASCII-Wert des ersten Zeichens des Stringausdrucks P$, mit dem diese Funktion aufgerufen wird. Die ASCII-Werte aller Zeichen finden sich in Anhang I "ASCII-Tabelle" im Bedienerhandbuch. Die ASC-Funktion ist die Umkehrung der CHR$-Funktion.

BEISPIEL:

```
A=ASC"SHARP"
```

Der ASCII-Wert des ersten Zeichens "S" ist 83.

b) CHR$ n

Wandelt numerische Werte in Zeichen um und ist damit die Umkehrfunktion der ASC-Funktion. Der Wert des Arguments n muß zwischen –32767 und +32767 liegen. Ist der Betrag des Arguments größer als 255, so wird es modulo 256 genommen, d.h der Divisionsrest von n/256 gebildet. Der so vorverarbeitete Wert wird in ein ASCII-Zeichen (String der Länge 1) umgewandelt.

BEISPIEL:

```
A$=CHR$ 78
```

Das Zeichen "N", dessen Wert 78 ist, wird A$ zugewiesen.

```
A$=CHR$ &5A
```

A$ wird das Zeichen "Z" (Wert 5A hexadezimal) zugewiesen.

c) LEN P$

Der Wert dieser Funktion ist Anzahl der Zeichen des String-Arguments P$ oder kurz gesagt, dessen Länge.

BEISPIEL:

```
A=LEN "MODEL-3500"
```

"MODELL 3500" ist 11 Zeichen lang, und A wird 11 zugewiesen.

d) VAL P$

Wandelt den Inhalt eines Strings, soweit möglich, in einen Zahlenwert um. VAL ist die Umkehrfunktion der STR$-Funktion.

BEISPIEL:

```
A=VAL "1.234.5"
```

Der in numerische Darstellung konvertierte Wert 1234.5 wird A zeugewiesen.

```
A=VAL "6.7*5"
```

Die Umwandlung des Strings in eine Zahl bricht beim * ab, denn * ist kein in Zahlen erlaubtes Zeichen. A erhält den Wert 6.7 (und nicht 33.5).

e) STR$ X

Die STR$-Funktion ist die Umkehrfunktion der VAL-Funktionen, sie wandelt numerische Daten in Strings um.

BEISPIEL:

```
A$=STR$ (.3*2+5)
```

Der numerische Ausdruck wird berechnet und das Ergebnis 5.6 in einen String konvertiert.

f) STRING$ (n, P$)

Liefert als Ergebnis einen String, in dem das erste Zeichen des Strings P$ n-fach wiederholt wird. n unterliegt den Regeln für die maximale Anzahl von Zeichen für Stringvariable, siehe Seite 13.

BEISPIEL:

```
A$=STRING$(10,"-")
```

A$ erhält den String "----------" zugewiesen.

g) STRING$ (m, n)

Konvertiert den absoluten Betrag des 2. Arguments n in eine Hexadezimalzahl und liefert als Ergebnis einen String aus so vielen Zeichen dieser Hexadezimalzahl, wie der Wert von m angibt. n muß im Bereich –65535 bis +65535 liegen. Ist der Wert von m kleiner als die Anzahl der benötigten Hexadezimalziffern, so wird der vordere Teil abgeschnitten; ist er größer, werden führende Nullen eingesetzt.

BEISPIEL:

```
A$=STRING$(4,4321)
```

A$ wird der String "10E1" (= hexadezimale Darstellung von 4321) zugewiesen.

```
A$=STRING$(6,4321)
```

Das Ergebnis ist der String "0010E1".

h) SPA m

Liefert als Ergebnis einen String aus m Leerzeichen.

BEISPIEL:

```
A$=SPA(2*B)
```

Sei B = 4, so erhält A$ 8 Leerzeichen zugewiesen.

i) LEFT$ (P$, m)

Greift die ersten m Zeichen, von links beginnend, aus P$ heraus.

BEISPIEL:

```
A$=LEFT$(B$,4)
```

Wenn B$ "COMPUTER" enthält, lautet der Funktionswert "COMP".

j) RIGHT$ (P$, m)

Greift die letzten m Zeichen aus P$ heraus.

BEISPIEL:

```
A$=RIGHT$(B$,4)
```

Sei B$ = "SYSTEM", dann lautet der Funktionswert "STEM".

k) MID$ (P$, m, n)

Greift aus P$ n Zeichen, heraus, beginnend beim m-ten Zeichen.

BEISPIEL:

```
A$=MID$(B$,3,4)
```

Wenn B$ = "MZ-3500" ist, lautet der Funktionswert "Z-3"

(2) Numerische Funktionen

Numerische Funktionen dienen zur Verarbeitung numerischer Daten. Argumente und Werte aller nachfolgend beschriebenen Funktionen sind numerisch.

a) INT x

Wandelt das Argument durch Abschneiden des Dezimalteils in eine ganze Zahl (INTEGER) um.

BEISPIEL:

```
A=INT (3.14)
```

Der ganzzahlige Teil von 3.14 ist 3.

b) ABS x

Berechnet den absoluten Betrag $|X|$.

BEISPIEL:

```
A=ABS (2*3-4^2/2)
```

Der Wert des Arguments ist –2, also lautet der Wert der Funktion 2.

c) SGN x

Liefert das Vorzeichen (signum) von x gemäß folgender Konvention:
SGN x = –1, wenn x negativ ist,
SGN x = 0, wenn x gleich Null ist,
SGN x = 1, wenn x positiv ist.

BEISPIEL:

```
A=SGN (16/4-6)
```

Das Argument hat den Wert –2, es ist also negativ, so daß der Funktionswert –1 lautet.

d) SQR x

Berechnet die Quadratwurzel (Squre root) $\sqrt{x}$. Wenn x negativ ist, wird eine Fehlermeldung ausgegeben.

BEISPIEL:

```
A=SQR (3^2+8*2)
```

Das Argument hat den Wert 25, $\sqrt{25}$ ist gleich 5.

e) SIN x

Berechnet den Sinuswert von x. Der Winkelmodus ist normalerweise (nach dem Einschalten) auf "Grad" eingestellt.

BEISPIEL:

```
A=SIN 30
```

Der Sinus von 30 Grad ist 0.5. (Winkelmodus Grad vorausgesetzt)

f) COS x

Berechnet den Cosinus von x.

BEISPIEL:

```
A=COS 30
```

(Winkelmodus Grad): cos 30° = 0.8660254038.

g) TAN x

Berechnet den Tangens des Winkels x.

BEISPIEL:

```
A=TAN 14.5
```

Der Winkelmodus sei auf "Bogenmaß" (radian) eingestellt, dann lautet das Ergebnis –2.634069132.

h) ASN x

Berechnet den Arcussinus von x (den zu x gehärigen Winkel) und stellt damit die Umkehrfunktion von SIN x dar. | X | muß kleiner oder gleich 1 sein.

BEISPIEL:

```
A=ASN (-.5)
```

Im Winkelmodus Grad ergibt sich 30 (Grad).

i) ACS x

Berechnet den Arcuscosinus von x. ACS ist die Umkehrfunktion der COS-Funktion. | X | muß $\leq$ 1 sein.

BEISPIEL:

```
A=ACS (.5-.2)
```

Das Ergebnis lautet im Winkelmodus Bogenmaß 1.266103673.

j) ATN x

Berechnet den Arcustangens von x, Umkehrfunktion der TAN-Funktion.

BEISPIEL:

```
A=ATN (-1)
```

Der Winkelmodus sei auf Grad eingestellt, dann lautet das Ergebnis –45.

---

*) Anmerkung zu den trigonometrischen und zyklometrischen Funktionen:
Im Grundzustand (nach dem Einschalten) ist die Einheit für Winkelberechnungen auf Grad (Kreisumfang = 360 Grad) eingestellt. Nach der Ausführung der Anweisung RAD (Seite 210) bildet jedoch das Bogenmaß (radian) die Winkelbasis (Kreisumfang = 2π = 6.28 . . . . .). Mit der DEG-Anweisung (Seite 209) kann man wieder auf Grad zurückschalten. Der Umrechnungsfaktor zwischen Grad- und Bogenmaß ist gleich 180/PI (PI siehe Seite 9.)

k) EXP x

Berechnet die Exponentialfunktion e^x

BEISPIEL:

```
A=EXP 2.5
```

$e^{2.5}$ wird berechnet, das Ergebnis ist 12.18249396.

l) LN x

Berechnet den natürlichen Logarithmus ln x (zur Basis e = 2.71828182846) und stellt die Umkehrfunktion von EXP x dar. x muß größer als 0 sein, sonst erfolgt eine Fehlermeldung.

BEISPIEL:

```
A=LN 2.5
```

ln 2.5 = 0.9162907319.

m) LOG x

Berechnet den gewöhnlichen (dekadischen oder Briggs-) Logarithmus log x. x muß größer als 0 sein, andernfalls erfolgt eine Fehlermeldung.

BEISPIEL:

```
A=LOG 2.5
```

log 2.5 = 0.397400087

n) RND x

Generiert (Pseudo-) Zufallszahlen zwischen 0 und x (kleiner als x). Die Zufallszahlen haben bis zu 12 signifikante Stellen. Vor dem ersten Aufruf von RND x wird die Ausführung einer RANDOM-Anweisung empfohlen (siehe Seite 212). Geschieht das nicht, so ist die erste Zufallszahl beim Aufruf von RND x (mit demselben Argument x) nach jedem Einschalten des Rechners oder jedem Löschen des Speichers dieselbe.

BEISPIEL:

```
A=RND 10+20
```

Der Aufruf von RND 10 liefert eine Zahl im Bereich von 0 bis 9.99999999999. Dazu wird 20 addiert, so daß A eine Zahl zwischen 20 und 29.9999999 zugewiesen erhält.

o) MODIFY x

Modifiziert den Wert von x je nach dem mit FIXED oder FLOAT eingestellten Zahlenformat.

BEISPIEL:

```
10 A=56.789
20 FIXED 2.0
30 B=MODIFY A
```

Die 3. und alle nachfolgenden Stellen von A werden gemäß der FIXED-Anweisung in Zeile 20 abgeschnitten, und in Zeile 30 erhält B den Wert 56.78 zugewiesen. Der Inhalt von A selbst bleibt unverändert.

p) CDMS x

Konvertiert x aus dem Dezimalsystem ins Sechzigersystem Grad (Degrees) – Minuten – Sekunden. Der Wert wird dabei so gerundet, daß sich eine ganze Zahl für die Sekunden ergibt.

BEISPIEL:

```
A=CDMS 12.3456
```

12.3456 sind im Sechzigersystem 12°20'44", der Wert der Funktion lautet 12.2044.

q) CDEG x

Konvertiert den Wert von x aus dem Sechzigersystem ins Dezimalsystem (decimal degrees = Grad dezimal). Umkehrfunktion von CDMS.

BEISPIEL:

```
A=CDEG 31.4545
```

31°45'45" sind in dezimaler Darstellung 31.7625.

r) STATUS m

STATUS ist eine Sonderfunktion zum Beschaffen von Informationen aus dem System. Sie liefert folgende Werte:

m = 2 ····· Anzahl unbenutzter Bytes im Anwenderbereich des Speichers.
m = 1 ····· Anzahl der im Programmspeicher belegten Bytes.
m = 0 ····· die gegenwärtige Zeilennummer.

Fragt man nach dem Starten von BASIC oder der Ausführung einer NEW-Anweisung den Wert von STATUS 2 ab, so erhält man die Gesamtzahl der im Anwenderbereich verfügbaren Bytes.

s) PEEK m

Liefert den Inhalt des durch m benannten ("adressierten") Bytes im Speicher als Zahl zwischen 0 und 255. Siehe auch Anhang I "ASCII-Tabelle" im Bedienerhandbuch. m muß im Bereich zwischen 38400 (&9600) und 53247 (&CFFF) je einschließlich liegen. PEEK ist im Zusammenhang mit der POKE-Anweisung (Seite 289) anzuwenden.

t) MOD (m, n)

MOD berechnet den Divisionsrest (Modulo-Funktion) von m/n. Der Rest hat dasselbe Vorzeichen wie m.

BEISPIEL:

```
A=MOD (19,3)
```

19/3 ergibt einen Rest von 1.

```
A=MOD (-35,6)
```

Der Rest von –35/6 ist nach obiger Konvention –5.

u) POINT (m, n)

Prüft, ob der durch die Koordinaten m (horizontal) und n (vertikal) beschriebene Punkt auf dem Bildschirm hell ist, und liefert bei einem hellen Punkt 1, bei einem dunklen Punkt 0 als Wert. Die Koordinaten m und n entsprechen denen der Anweisungen SET und RESET (Seite 219). Auf dem Graphik-Bildschirm bezeichnen die Koordinaten Graphik-Punkte. Siehe auch 3. (2) "Koordinatensysteme und Kursor" auf Seite 33.

Koordinaten auf dem Text-Bildschirm
$0 \leqq m < 160$
$0 \leqq n < 50$

Koordinaten auf dem Graphik-Bildschirm
$0 \leqq m < 640$
$0 \leqq n < 400$

ANMERKUNG:

1. Für alle Funktionen gilt im Gegensatz zu den später behandelten Unterprogramnen, daß die Argumente durch die Funktion niemals verändert werden. Anpassungen der übergebenen Daten an die Erfordernisse der Funktion werden nicht an den Originaldaten, sondern an einer temporären Kopie durchgeführt, die die Funktion intern anlegt. Zur Verdeutlichung ein kleines Beispiel:
   Es werde die SPA-Funktion mit einer Variablen X als Argument aufgerufen, z.B. QE$ = SPA X + CHR$(120–I%), und X habe in diesem Augenblick den Wert 14.5448. Da es nun aber keine "halben" Leerstellen gibt, muß die funktionsinterne Kopie von X vorher in eine ganze Zahl umgewandelt werden. X selbst bleibt davon unberührt, auch nach dem Aufruf der SPA-Funktion ist sein Wert unverändert 14.5448.

2. Unter einem "numerischen Ausdruck" (oder arithmetischen Ausdruck) versteht man eine Verknüpfung von numerischen Konstanten, Variablen (einfache und Arrayelemente) und Funktionen mit Klammern und arithmetischen Operatoren, kurz gesagt, eine mathematische Formel. In dieser Schrift werden auch Vergleichs- und logische Ausdrücke mit zu den numerischen Ausdrücken gezählt.
   Ein numerischer Ausdruck kann komplex sein, muß es aber nicht. Auch eine einfache Variable, ja sogar eine Konstante ist ein numerischer Ausdruck!
3. Im Gegensatz zu dem in der Mathematik üblichen Gebrauch verschiedener Klammerformen dürfen in BASIC ausschließlich runde Klammern benutzt werden!

3. TEXT UND GRAPHIK

(1) Text- und Graphikdarstellung

Alle Text- und graphischen Informationen, die auf dem Bildschirm angezeigt werden sollen, schreibt das System zu diesem Zweck automatisch in einen besonderen Speicherbereich, den sog. Bildwiederholspeicher (Video refresh memory, Video RAM). Die Video-Elektronik liest die Informationen aus diesem Speicher aus und erzeugt daraus das Bildsignal für den Monitor. Der Bildwiederholspeicher ist technisch in mehrere sog. "Ebenen" für Text und Graphik gegliedert, die im folgenden kurz als Text- oder Graphikebenen angesprochen werden. Im einzelnen sind das:

Monitor
- Textebene
- Graphikebenen
  - Graphikebene 1 (auf Farbmonitor blau)
  - Graphikebene 2 (auf Farbmonitor rot)
  - Graphikebene 3 (auf Farbmonitor grün)

1) Die Textebene

Auf dem Textbildschirm werden Zeichen dargestellt. Es können Blinken, inverse Darstellung, Markierungslinien und Farbe spezifiziert werden. Blinken wird mit der Anweisung BL (siehe Seite 228), inverse Darstellung (Hervorhebung durch Vertauschen der Farben von Zeichen und Hintergrund) mit der Anweisung RV (siehe Seite 229) gesteuert. Die TABLE-Anweisung (siehe Seite 232) bewirkt das Durchziehen von Markierungslinien zur Darstellung von Tabellen.

Wenn farbiger Text gewünscht wird, muß das System in den Farbmodus gebraucht werden; siehe Seite 34, "Farbwiedergabe".

Die kleinste in der Textebene darstellbare Einheit ist ein Zeichen.

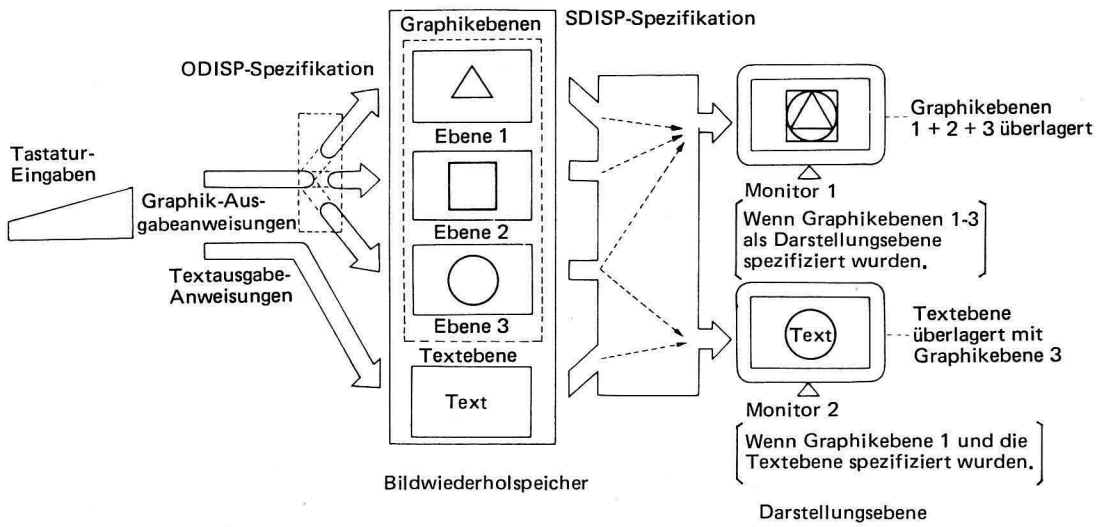
2) Die Graphikebenen

Auf dem Graphikbildschirm können Punkte, Linien (Punktmengen) und Figuren (Linienmengen) dargestellt werden. Alle diese Bildelemente werden zusammen kurz als "Graphik" bezeichnet.

Die Einheit der Graphikbene ist ein (Bild-) Punkt. Die Graphikebene ist insgesamt 640 Punkte breit x 400 Punkte hoch.

Der Graphikebene ist unterteilt in die Teilebenen 1, 2 und 3. Eine, zwei oder alle drei Teilebenen können gleichzeitig auf dem Monitor dargestellt werden. Wenn mehr als eine Ebene zur Anzeige kommt, sagt man, die Ebenen werden überlagert. Auch können Text und Graphik gemischt werden.

Stehen zwei Monitore zur Verfügung, so kann jeder Monitor nach Wahl mit einem eigenen Bild versorgt werden.



Für den Bildschirm bestimmte Informationen werden im Bildwiederholspeicher abgelegt. Mit der Anweisung ODISP (siehe Seite 243) kann der Benutzer angeben, in welche der drei Graphikebenen 1, 2 oder 3 "gezeichnet" werden soll. Jede Graphikebene kann für sich angesteuert, oder auch zwei oder alle drei Ebenen gleichzeitig mit denselben Graphikdaten gefüllt werden.

Mit der Anweisung SDISP (siehe Seite 241) wird gesteuert, welche Graphikebenen, evtl. auch mit der Textebene überlagert, auf welchem Monitor zur Darstellung gebracht werden. In obigem Beispiel enthält Graphikebene 1 ein Dreieck, Ebene 2 ein Quadrat und Ebene 3 einen Kreis. Auf Monitor 1 werden alle drei Graphikebenen überlagert dargestellt, während auf Monitor 2 die Textebene zusammen mit Graphikebene 1 zur Anzeige kommt.

Im Farbmodus werden die Graphikebene 1 bis 3 mit den Grundfarben blau, rot und grün belegt. Die Farbausgabe (acht Farben) kommt durch die Überlagerung aller drei Graphikebenen auf einem Farbmonitor zustande.

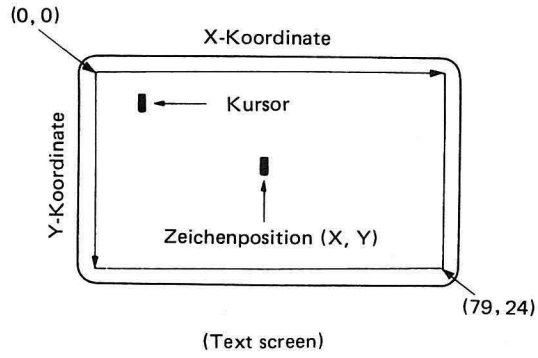
(2) Koordinatensysteme und Kursor

1) Kordinatensysteme

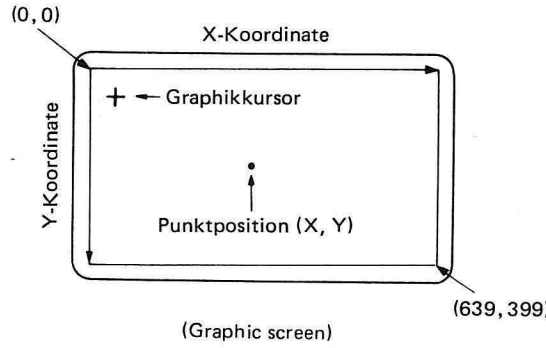
In diesem System finden zwei verschiedene Koordinatensysteme zur Angabe von Positionen auf dem Bildschirm Anwendung, eines für die Darstellung von Text, das andere für graphische Zwecke.

Das Text-Koordinatensystem bezieht sich auf die Positionen (X, Y) einzelner Zeichen der Textdarstellung. Hier stehen maximal 25 Zeilen mit je bis zu 80 Zeichen zur Verfügung.

Das Graphik-Koordinatensystem bezieht sich auf die Positionen (X, Y) einzelner Bildpunkte. Im folgenden Schema werden die Zusammenhänge zwischen den Werten der Koordinaten X, Y und den Positionen auf dem Bildschirm für beide Systeme dargestellt.



(a) Text-Koordinatensystem



(b) Graphik-Koordinatensystem

2) Steuerung des Kursors

Der Kursor zeigt die aktuelle Zeichen- oder Bildpunktposition für die Ausgabe an. Der Textkursor wird, wie oben angedeutet, durch ein Rechteck von der Größe eines Zeichenfeldes auf dem Bildschirm dargestellt. Man kann ihn mit den Tasten [→] und [←] auf ein bestimmtes Zeichen fahren und dieses dann mit Hilfe der Tastatur direkt korrigieren.

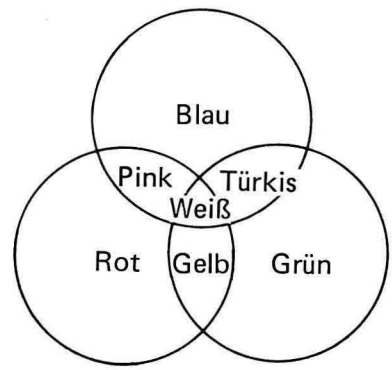
Der Graphikkursor hat die Form eines kleinen Kreuzes, dessen Mittelpunkt die Position eines Bildpunktes anzeigt. Er erscheint nur nach der Ausführung einer GCURSOR- oder GINPUT-Anweisung auf dem Bildschirm (siehe Seite 277 bzw. 279). Man kann ihn mit den Tasten [↑], [↓], [←], and [→] in alle vier Richtungen auf dem Bildschirm bewegen. Einfache Operationen auf der Tastatur bewirken keine Eingabe von Graphikpunkten. Wenn man mit der Tastatur Graphikeingaben machen möchte, so führe man die Anweisung GINPUT (siehe Seite 279) aus.

(3) Farbwiedergabe

Die Farbspezifikation ist für Text und Graphik verschieden. In beiden Fällen jedoch muß zuerst das System für Farbausgabe vorbereitet werden, indem man eine CHANGE DISP- (siehe Seite 237) oder eine SDISP-Anweisung (siehe Seite 241) ausführt. Man kann zwischen den Farben blau, rot, grün, pink, türkis, gelb, weiß und schwarz wählen, hat also insgesamt acht verschiedene Farben zur Verfügung.
Für die Textdarstellung ist die Farbe der Zeichen mit Hilfe der COLOR-Anweisung (siehe Seite 239), für Graphik und Hintergrund mit der GCOLOR-Anweisung (siehe Seite 245) vorzugeben. Die Graphikfarben sind dieselben wie für die Textausgabe. Die Auswahl und Ansteuerung der Graphikebenen erfolgt automatisch je nach der spezifizierten Farbe.

Graphik-farbe	Graphikebene		
	blau	rot	grün
Blau	O	X	X
Rot	X	O	X
Grün	X	X	O
Pink	O	O	X
Türkis	O	X	O
Gelb	X	O	O
Weiß	O	O	O
Schwarz	X	X	X

O : wird belegt
X : wird nicht belegt



Folgende Tabelle zeigt den Zusammenhang zwischen den Farben und den Graphikebenen. Rechts daneben ist die Überlagerung der drei Grundfarben blau, rot und grün zu den Mischfarben dargestellt.

WICHTIGER HINWEIS

Neben der Spezifikation der Farbe muß die Bildschirmsteuerung mit der SDISP-Anweisung (siehe Seite 241) richtig durchgeführt werden, wenn man eine farbige Ausgabe wünscht. Selbstverständlich ist auch ein farbtüchtiger Monitor erforderlich.

4. DATEIEN

(1) Was ist eine Datei?

1) Dateiname

Programme oder andere Daten, die auf einem externen Speichermedium abgelegt sind, werden als "Datei" bezeichnet. Enthält eine Datei ein Programm, so sagt man auch Programmdatei dazu, im Gegensatz zu anderen Dateien, die Daten allgemeiner Art enthalten. Der einer Datei zugeordnete Name ist der Dateiname.

Disketten ("Floppy Disks") und (Magnetband-) Kassetten sind Beispiele für externe Speichermedien. Disketten besitzen eine Reihe von Vorteilen gegenüber Bandkassetten:

① Höhere Schreib-/Lesegeschwindigkeit ② Größeres Fassungsvermögen ③ Direkter Zugriff auf bestimmte Daten

Auf Disketten werden Dateien über ihren Dateinamen verwaltet. Bei der Einrichtung einer Datei auf einer Diskette wird ihr sofort ein Name zugeordnet.

BEISPIEL:

Um ein Programm unter dem Dateinamen GERD abzuspeichern, lautet die Eingabe SAVE "GERD" [ENTER] .

Während der Aufzeichnung einer Datei werden Dateiname und der Ort auf der Diskette, wo die Daten abgelegt wurden, automatisch in einem Inhaltsverzeichnis (Directory, auf derselben Diskette) festgehalten. Mit Hilfe der Eintragungen im Inhaltsverzeichnis können die Daten dann beim späteren Lesen wieder lökalisiert werden.

2) Der Diskettenname (Volume name)

Zur selben Zeit können mehrere Disketten benutzt werden, und man kann während eines Programmlaufes Disketten auswechseln. Zur Unterscheidung läßt sich jeder Diskette ein eigener Name, der Diskettenname, zuordnen.

So trägt z.B. die mitgelieferte Diskette (FDOS master disk) den Namen "SYSTEM". Bevor eine neue Diskette verwendet werden kann, muß sie initialisiert werden (siehe Seite 45), und bei dieser Gelegenheit wird ihr auch der Diskettenname zugeordnet.

BEISPIEL:

Ein Programm soll unter dem Dateinamen "UWE4" auf der Diskette mit der Bezeichnung "DISK1" aufgezeichnet werden. Dann ist an der Tastatur einzugeben:

SAVE "UWE4, DISK1" [ENTER]

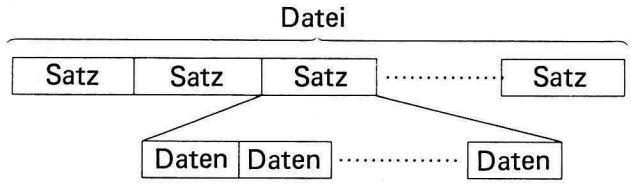
(In diesem Beispiel wurde der Diskettenname angegeben, nicht die Kanal-Laufwerk-Nummer.)

Diskettenname, Inhaltsverzeichnis und Dateinamen stehen in folgendem Zusammenhang:

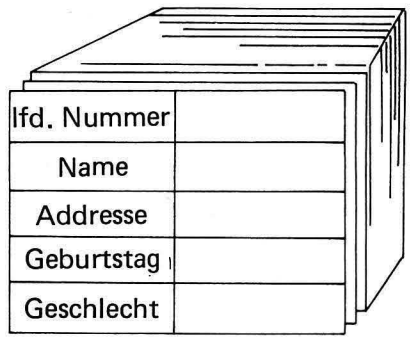
Diskettenname	← Bezeichnung der gesamten Diskette
Inhaltsverzeichnis	← Zur Verwaltung der Dateibereiche
Datei 1	
Datei 2	
Datei 3	
⋮	
Datei n	

3) Datensätze (Records)

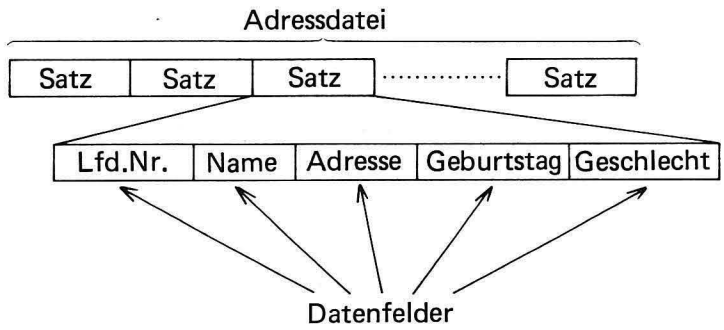
Daten werden in Einheiten von Datensätzen in Dateien geschrieben oder oder von ihnen gelesen. Ein Datensatz oder kurz "Satz" (engl. "record") ist eine zusammengehörige Gruppe von Daten. Eine Datei setzt sich also aus Datensätzen zusammen, die wiederum aus einzelnen Datenfeldern bestehen.



So läßt sich zum Beispiel eine Adresskartei in Form einer Datei abspeichern (Man beachte, wie ähnlich die Begrriffe klingen!):



Laufende Nummer, Name, Adresse, Geburtstag und Geschlecht jeder Person werden als ein Datensatz abgespeichert. Jeder Karteikarte entspricht also ein Satz in der Datei. Die einzelnen Eintragungen pro Karte bzw. Satz heißen "Datenfelder".



(2) Dateiarten

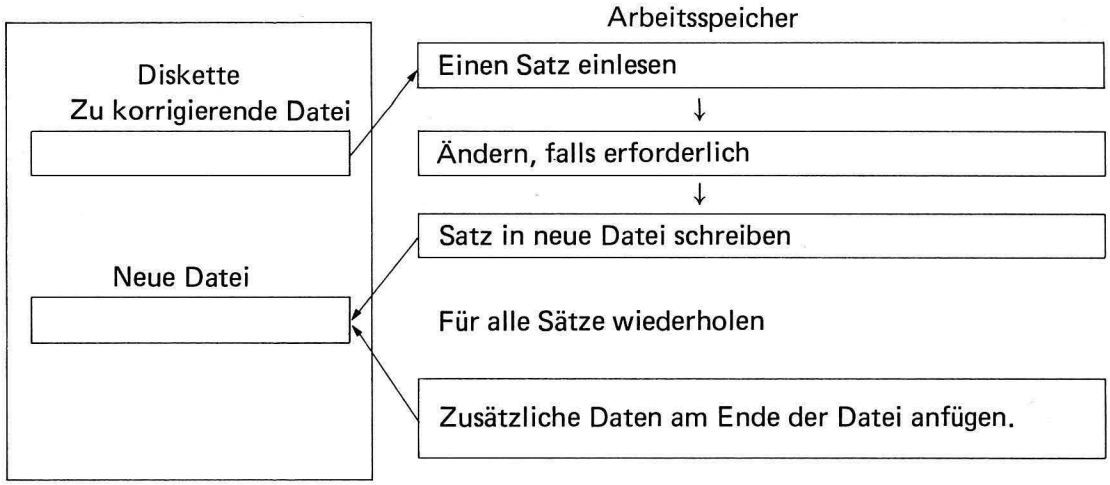
Dateien werden unterschieden in Programmdateien, Sonderdateien und (allgemeine) Dateien.

1) Allgemeine Dateien
   Hier unterteil man in sequentielle Dateien und Direktzugriffs- oder "Random"-Dateien. In sequentiellen Dateien werden die Datensätze in der Reihenfolge abgelegt, wie sie geschrieben werden. Es ist nicht möglich, mit dem Lesen oder Schreiben irgendwo in der Mitte der Datei zu beginnen. In Random-Dateien hingegen werden die Sätze intern numeriert, und man kann jeden beliebigen Satz frei nach Wahl lesen oder schreiben.

a) Sequentielle Dateien
   Die "Satzlänge" ist variabel je nach dem Umfang der Daten pro Satz. Mit variablen Satzlängen läßt sich der Platz auf der Diskette besonders effizient nutzen.
   Anzahl und Inhalt der Datenfelder werden bei jedem Schreiben eines Satzes durch die Satz-Schreibanweisung (PRINT#-Anweisung, siehe Seite 342) automatisch festgelegt, so daß von Satz zu Satz verschiedene Werte möglich sind.
   Man kann jedoch nicht mit dem Lesen oder Schreiben in der Mitte einer sequentiellen Datei beginnen. Will man auf einen bestimmten Satz-zugreifen, so muß man alle davor liegenden Sätze ebenfalls lesen.
   Zum Korrigieren oder Hinzufügen von Sätzen muß die gesamte Datei in den Arbeitsspeicher gelesen und hier die erforderlichen Änderungen oder Ergänzungen angebracht werden. Dancach werden alle Daten in dieselbe Datei zurückgeschrieben.



   Ist die zu ändernde Datei jedoch so umfangreich, daß sie nicht in den Arbeitsspeicher paßt, so geht man einen anderen Weg. Man legt eine neue Datei an und liest die alte Datei Satz für Satz durch. Sätze, dere Inhalt nicht geändert werden muß, werden sofort in die neue Datei geschrieben, andernfalls werden vor dem Schreiben die Korrekturen angebracht.
   Neue Sätze werden am Ende der neuen Datei hinzugefügt.



Gibt bei der Eröffnung der Datei den entsprechenden Zugriffschlüssel an, so kann man ohne weitere Maßnahmen eine Datei durch Hinzufügen von Sätzen am Ende vergrößern, siehe OPEN-Anweisung auf Seite 334.

b) Random-Dateien (Direktzugriffsdateien)

Eine Random-Datei ist mit Hilfe der CREATE-Anweisung (Seite 332), die den erforderlichen Platz auf der Diskette reserviert, auf einer Diskette einzurichten.
Die Größe der Datei muß daher im voraus bestimmt werden.
Die Satzlänge ist fest, und Datentransporte von der/zur Diskette spielen sich in Einheiten von Sätzen ab. Jeder Satz erhält eine Satznummer zur Identifikation, sodaß durch Angabe der Satznummer jeder beliebige Satz gelesen oder geschrieben werden kann.
Der Ort eines Satzes auf der Diskette (die Satzadresse) kann systemintern über die (feste) Satzlänge und die Satznummer berechnet werden:

Satzadresse = (Adresse des Dateianfangs) + Satzlänge x (Satznummer – 1)

Weil Random-Dateien beliebige Lese- und Schreibzugriffe auf alle Sätze gestatten, eignen sie sich gut für Zwecke der Datenhaltung und Korrektur. Der Platz auf den Disketten kann jedoch nicht so effizient genutzt werden, weil die Satzlänge entsprechend der maximal erforderlichen fest vorgegeben werden muß.

2) Programmdateien

BASIC-Programme können in zwei verschiedenen Formaten auf Disketten gespeichert werden: als ASCII-Dateien und als Zwischenkode-Dateien.

a) Zwischenkode-Dateien

Eine Zwischenkode-Datei enthält ein exaktes Abbild eines BASIC-Programms, wie es während der Ausführung im Speicher residiert. Der Zwischenkode ist eine spezielle, sehr kompakte Darstellungsart von Programmen, die weniger Platz beansprucht als ASCII-Programmdateien. Normalerweise werden Programme in diesem Format auf einer Diskette gespeichert.

b) ASCII-Programmdateien

Eine ASCII-Programmdatei enthält ein BASIC-Programm in Form eines Textes aus ASCII-Zeichen, genau so, wie man die Programme über die Tastatur eingibt oder auf Bildschirm oder Drucker auflisten kann. Da die syntaktischen Regeln dieses BASIC-Interpreters mehr oder weniger denen anderer Implementationen der BASIC-Sprache gleichen, gestattet dieses Format den Austausch von BASIC-Programmen mit fremden Computersystemen. Da die Aufzeichnungsformate vonDateien auf Disketten zwischen verschiedenen Systemen differieren, muß zur Anpassung ein entsprechendes Dienstprogramm (Utility) benutzt werden.

3) Sonderdateien

In diesem System gibt es außer allgemeinen und Programmdateien noch eine Reihe von Dateien mit einer besonderen Bedeutung:

Programmdateien mit Programmen in Maschinensprache
Kopien der Textebene des Bildwiederholspeichers
Kopien der Graphikebenen
Dateien mit Funktionstasten-Belegungen

Einzelheiten siehe Seite 372.

5. FDOS BETRIEBSSYSTEM

(1) Was ist FDOS?

FDOS (Floppy Disk Operating System) ist ein Systemprogramm, das aus mehreren Routinen*) besteht, in deren Mittelpunkt die Dateiverwaltung steht.
Um FDOS zu laden, schiebt man die FDOS-Masterdiskette in das rechte Laufwerk (Laufwerk Nr. 0) und schaltet das Gerät ein.
FDOS besteht aus folgenden Routinen:

① Floppy-Disk-Steuerungsroutine
② Dateiverwaltungs-Routine
③ Routinen zur Steuerung der FDOS-Befehle, zur Verwaltung des RAM-Speichers und zur Steuerung des Eingabe-/Ausgabesystems
④ FDOS-Befehlsausführungsroutine

zu 1: Diese Routine bestimmt Sektor und Spur auf der Diskette zum Lesen und Schreiben von Dateien. Sie bildet daher die Grundlage für die Steuerung der Floppy-Disk.

zu 2: Diese Routine verwaltet das Dateiwesen und erleichtert dem Anwender das Einrichten von Dateien, ohne daß er sich um die physikalische Aufzeichnung der Daten auf der Diskette in Sektoren oder auf Spuren kümmern muß.

zu 3: Ein Teil der Routinen für die Durchführung von FDOS-Befehlen wird ständig im Arbeitsspeicher gehalten (sie sind speicherresident), andere wiederum stehen als Datei auf der Systemdiskette zur Verfügung. Wird ein Befehl dieses zweiten (ausgelagerten) Typs über die Tastatur eingegeben, dann wird die entsprechende Routine von der Diskette in den Overlaybereich geladen.
Die Übertragung von Dateiinformationen zwischen der Diskette und dem Speicher erfolgt über einen Zwischenspeicher (Buffer); dieser Zwischenspeicherbereich wird ebenfalls von dieser Routine verwaltet.

zu 4: Die Routine zur Ausführung der FDOS-Befehle ist speicherresident.

Disk-BASIC operiert unter der Kontrolle der Routinen 1, 2 und 3. FDOS benutzt die

FDOS-Befehle
- BASIC
- INIT
- CAT
- BACKUP
- COPY
- SCOPY
- KILLALL
- DEBUG

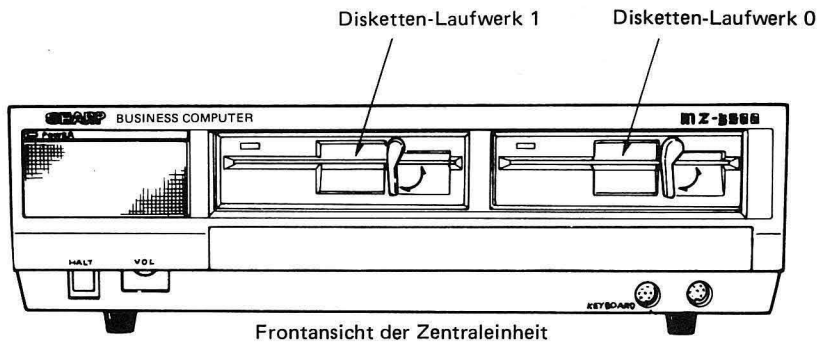
Alle Befehle bis auf CAT werden ausgeführt, nachdem die entsprechende Routine in den Overlaybereich geladen wurde. Unter FDOS können außer BASIC auch noch andere Programmiersprachen betrieben werden.

*) Unter Routine versteht man ein häufig benutztes Programm bzw. Teilprogramm.

(2) Bezeichnungen

i) Kanal-Laufwerk-Nummer

An das System können zusätzlich zu den eingebauten Floppy-Laufwerken weitere Laufwerke angeschlossen werden. Jedes Laufwerk besitzt eine Kanal-Laufwerk-Nummer. Die Kanal-Laufwerk-Nummer besteht aus dem Kanalkode und der Nummer des Laufwerkes. Zur Kennzeichnung des Kanalkodes dienen die Buchstaben A und B, als Laufwerks-Nummern die Ziffern 0,1,2 und 3. Die Laufwerke in der Zentraleinheit besitzen die Kanal-Laufwerk-Nummern A0 (rechtes Laufwerk) und A1 (linkes Laufwerk). Kanal B bezeichnet ein extern angeschlossenes Floppy-Laufwerk.



Frontansicht der Zentraleinheit

ii) Dateikennung

Eine Datei wird durch einen String (Zeichenkette) spezifiziert.
Dieser String besteht aus dem Dateinamen, einer Zusatzbezeichnung, der Diskettenbezeichnung und der Kanal-Laufwerk-Nummer.

Zwischen Dateinamen und Zusatzbezeichung muß ein Schrägstrich (/), vor der Diskettenbezeichnung ein Komma (,) und von der Kanal-Laufwerk-Nummer ein Doppelpunkt (:) stehen.

BEISPIEL:

```
"BEISPIEL/TXT,SHARP:A1"
```

↑ Kanal-Lauf-Nr.
↑ Diskettenbezeichnung
↑ Zusatzbezeichnung
↑ Dateiname

Dateiname ..... Der Dateiname besteht aus maximal acht alphanumerischen Zeichen, muß jedoch mit einem Buchstaben beginnen.

Zusatzbezeichung ..... Die Zusatzbezeichnung besteht aus maximal drei alphanumerischen Zeichen, muß jedoch ebenfalls mit einem Buchstaben beginnen.
Bei einigen FDOS-Befehlen hat die Zusatzbezeichung eine spezielle Bedeutung und kann als eine Art Dateinamen betrachtet werden.

Diskettenbezeichnung..... Die Diskettenbezeichnung beginnt ebenfalls mit einem Buchstaben und kann aus maximal sechs alphanumerischen Zeichen ohne Zwischenraum bestehen.

Die Diskettenbezeichnung und die Kanal-Laufwerk-Nummer sind optional. Ist kein Diskettenname aufgeführt, dann wird die Diskette im Laufwerk mit der angegebenen Kanal-Laufwerk-Nummer genommen. Umgekehrt wird die Diskette mit dem bezeichneten Diskettennamen gewählt, wenn keine Kanal-Laufwerk-Nummer angegeben ist. Fehlen beiden Angaben, der Diskettenname und die Kanal-Laufwerk-Nummer, dann wird automatisch die Diskette in Laufwerk A0 angesteuert.
Ist der Kanalkode in der Kanal-Laufwerk-Nummer weggelassen, wird Kanal A, fehlt die Laufwerk-Nummer, dann wird Laufwerk 0 angenommen.

HINWEIS

1. Im Rahmen von BASIC ist die Dateikennung als String in Anführungsstriche (" ") zu setzen. Nicht dagegen in einem FDOS-Befehl: hier wird sie als Zeichenkette ohne Anführungsstriche angegeben.
2. Die Dateikennung "SAMPLE/TXT" und "SAMPLE" kennzeichen unterschiedliche Dateien. Die erste Bezichnung enthält die Zusatzbezeichnung TXT.

(3) FDOS-Befehle

Befehle, die innerhalb der FDOS-Operationsebene benutzt werden können, werden als FDOS-Befehle bezeichnet, um sie von denjenigen Befehlen abzuheben, die in der BASIC-Operationsebene verwendet werden können.
Folgende FDOS-Befehle werden an dieser Stelle erläutert:

BASIC
INIT
CAT
BACKUP
COPY
SCOPY
KILLALL
DEBUG

BASIC-Befehl

Zum Laden des BASIC-Interpreters von der Diskette in den Arbeitsspeicher.

FORMAT: BAS I C

ERLÄUTERUNG:

Man schiebt die FDOS-Masterdiskette in Laufwerk A0 (rechtes Laufwerk in der Zentraleinheit) und schaltet den Strom ein (Zentraleinheit und Bildschirm). Nachdem FDOS geladen wurde, befindet sich das System in der FDOS-Operationsebene. Jetzt können FDOS-Befehle über die Tastatur eingegeben werden.

Wird der Befehl BASIC eingegeben, dann wird der BASIC-Interpreter von der Diskette gelesen und in den Arbeitsspeicher der Zentraleinheit geladen. Das System befindet sich jetzt in der BASIC-Operationsebene. Möchte man BASIC wieder verlassen und in die FDOS-Operationsebene zurückkehren, benutzt man die BASIC-Anweisung BYE (siehe Seite 328).

Nach dem Einlegen der FDOS-Masterdiskette in Laufwerk A0 und dem Einschalten des Systems wird folgende Meldung auf dem Bildschirm ausgegeben:

ANWENDUNGSBEISPIEL: Bildschirmausgabe

```
FDOS   22007A(V1.0A)
READY
```

Tippt man nun BASIC ein und drückt die CR-Taste [ENTER], wird folgende Meldung ausgegeben:

```
BASIC  22007B(V1.0A)
READY
>
```

Jetzt kann man mit BASIC arbeiten. In diesem Statium sind drei Zwischenspeicher (Puffer) für Ein- und Ausgabeoperationen zwischen der Diskette und dem Arbeitsspeicher reserviert. (Siehe hierzu MAXFILE-Anweisung auf Seite 330).

INIT-Befehl

Zur Initialisierung von Disketten.

〈VERWENDUNG〉
Um eine neue Diskette für das System benutzbar zu machen.

FORMAT: I N I T

ERLÄUTERUNG:

Vor dem erstmaligen Gebrauch muß jede neue Diskette über den Befehl INIT initialisiert werden. Dieses Programm prüft alle Spuren, ob sie fehlerfrei sind, und beschreibt die Diskette mit Spur- und Sektor-Nummern, Adressen, usw.

Im Rahmen der Initialisierung wird der Diskette zur späteren Identifikation auch ein Name (Diskettenbezeichung) zugeordnet. Durch INIT können Disketten über jeden der beiden Kanäle und in jedem beliebigen Laufwerk initialisiert werden.

ANWENDUNGSBEISPIEL:

Man schiebt die FDOS-Masterdiskette in Laufwerk A0 und gibt den INIT-Befehl ein:

INIT [ENTER] ..... Es werden nacheinader folgende Meldungen am Bildschirm ausgegeben:

```
FD INITIALIZE 22007C(V1.0A)

* CH.A(0) OR CH.B(1)?①
* DRIVE NUMBER?②
* VOLUME NAME?③
* PRESS ENTER KEY④

   TRACK : 00⑤


* END Y(0) OR N(1)?⑥
```

① Frage nach dem Laufwerk-Kanal.
Für Kanal A gibt man 0, für Kanal B eine 1 ein. 0 oder 1 [ENTER]

② Man gibt die Nummer des Laufwerks ein, in dem sich die zu initialisierende Diskette befindet (0 – 3). 0 – 3 [ENTER]

③ Wird nut die [ENTER]-Taste gedrückt, dann wird als Diskettenbezeichnung "SHARP" eingetragen. Möchte man die Diskette mit einem speziellen Namen versehen, dann kann man – beginnend mit einem Buchstaben – bis zu sechs alphanumerische Zeichen eingeben (ohne Zwischenraum). Man könnte z.B. "PROBE1" eintippen und dann die [ENTER] -Taste drücken.

④ Das Drücken der [ENTER] -Taste startet die Initialisierung. Bevor man die [ENTER] -Taste drückt, sollte man sich davon überzeugen, daß man wirklich die zu initialisierende Diskette eingelegt und die richtige Kanal-Laufwerk-Nummer eingegeben hat. Durch eine Initialisierung wird ein etwa vorhandener Inhalt der Diskette gelöscht.

⑤ Ausgabe der Spurnummern während der Initialisierung. Wird ein Fehler entdeckt, dann kann dieser über die ausgewiesene Spurnummer lokalisiert werden.

⑥ Diese Meldung erscheint nach Abschluß der Initialisierung. Möchte man eine weitere Diskette initialisieren, gibt man eine 1 ein und beginnt wieder mit dem ersten Schritt. Zum Beenden der Initialisierung gibt man eine 0 ein. 0 oder 1 [ENTER]

Wurde ein Fehler entdeckt, dann wird die entsprechende Fehlermeldung über den Bildschirm ausgegeben. Man drücke in diesem Fall die [CL] -Taste oder gleichzeitig die [CTRL] - und die [HOME] -Taste, um die Initialiserung noch einmal mit dem ersten Schritt (Eingabe des Laufwerk-Kanals) zu beginnen.
Merke: Gedrückthalten von [CTRL] und Eintippen von [HOME] führt zum ersten Schritt der Initialiserung, analoges Eingeben von [CTRL] und [A] zurück ins FDOS.

WICHTIGER HINWEIS

1. Bei Eingabe des INIT-Befehls muß sich die FDOS-Masterdiskette in Laufwerk A0 befinden.
2. Wenn eine Diskette initialisiert wird, auf der Programme und Daten abgespeichert wurden, dann werden alle Dateien gelöscht. Man versehe daher Disketten, die geschützt werden sollen, mit einem Schreibschutzaufkleber (Seite 27 des Beidienerhandbuchs).
3. Disketten mit dem Diskettennamen "SYSTEM" können nicht initialisiert werden.

CAT-Befehl

Zum Ausgeben eines Verzeichnisses mit den auf einer Diskette befindlichen Dateien.

〈VERWENDUNG〉
① Dieser Befehl gibt Auskunft über die auf der Diskette katalogisierten Dateien
② die Disketten- und Dateienspezifikation
③ den Umfang des freien Speichers auf der Diskette.

FORMAT: CAT␣CH m 〔*〕〔/〕

␣ : Zwischenraum (Leerzeichen). Ist bei der Darstellung eines FDOS-Befehls ein Zwischenraum ausgewiesen, dann ist dieser unbedingt mit einzugeben.
CH : Kanalkode (A oder B).
m : Laufwerk-Nummer (0, 1, 2, 3).
* : Ausgabe erfolgt über den Drucker.
/ : Ausgabe enthält die Angabe von Datum und Zeit.

ERLÄUTERUNG:

1. Der CAT-Befehl listet die auf der angegebenen Diskette befindlichen Dateien. Nähere Angaben zu der ausgegebenen Liste finden sich bei der Beschreibung der CAT-Anweisung (Seite 320).
2. Wird nach der Laufwerk-Nummer ein Stern eingegeben, dann wird das Verzeichnis über den Drucker ausgegeben, ist er ausgelassen, erfolgt die Ausgabe über den Bildschirm.
3. Ein Schrägstrich bei der Eingabe bewirkt die Angabe von Datum und Uhrzeit der Abspeicherung jeder Datei bzw. deren Rückschreibung.

ANWENDUNGSBEISPIEL:

1. CAT␣A1 [ENTER] ··· Listet die Dateien auf der Diskette in Laufwerk A1 am Bildschirm.
2. CAT␣A0*/ [ENTER] Druckt eine Liste aller Dateien auf der Diskette in Laufwerk A0 einschließlich der zeitlichen Angaben der Abspeicherung.

WICHTIGER HINWEIS

Für die Aufhebung der Fehlersperre benutze man die [CR] oder die [↑] Taste.

BACKUP-Befehl

Zum Kopieren einer ganzen Diskette.

〈VERWENDUNG〉
Für das Anfertigen von Sicherungs- bzw. Arbeitskopien der Originaldisketten oder Aktualisieren von Diskette-inhalten.

FORMAT: BACKUP

ERLÄUTERUNG:

Mit BACKUP kopiert man eine ganze Diskette, wenn man vom Diskettenoriginal ein Duplikat erstellen möchte. Von der FDOS-Masterdiskette sollte man auf ieden Fall eine Arbeitskopie herstellen, um das Original zu schützen.

Bei Eingabe des BACKUP-Befehls werden vom System zur Spezifizierung der betroffenen Laufwerke seitens des Benutzers Meldungen am Bildschirm ausgegeben.

ANWENDUNGSBEISPIEL:

Anfertigung einer Arbeitskopie von der FDOS-Masterdiskette. Hierzu schiebt man die FDOS-Masterdiskette in das Laufwerk A0. In Laufwerk A1 legen wir eine initialisierte, leere Diskette. Dann tippt man BACKUP ein und drückt die [ENTER] -Taste.

Auf dem Bildschirm erscheint nach Maßgabe der Eingaben folgende Anzeige:

```
FD BACKUP  22007D(V1.0A)

( SOURCE DISK )①      TRACK : 00 ⑧
* CH.A(0) OR CH.B(1)?②  } Informationen der zu kopierenden Diskette
* DRIVE NUMBER? ③       }

( DESTINATION DISK )④ TRACK : 00 ⑨
* CH.A(0) OR CH.B(1)?  ⑤  } Informationen der Diskette,
* DRIVE NUMBER?  ⑥        } auf die kopiert werden soll

* PRESS ENTER KEY  ⑦


* END Y(0) OR N(1)? ⑩
```

1. Die zu kopierende Diskette.
2. Kanalkode des Laufwerks, in dem sich die zu kopierende Diskette befindet: Da sich die Masterdiskette in Laufwerk A0 befindet, gibt man 0 ein. 0 oder 1 [ENTER]
3. Die entsprechende Laufwerk-Nummer: man gibt 0 ein. 0 – 3 [ENTER]
4. Disk, auf die kopiert werden soll.
5. Kanalkode des Laufwerks, in dem sich die Diskette befindet, auf die kopiert werden soll: [ENTER]
   Da sich unsere initialisierte Leerdiskette in Laufwerk A1 befindet, gibt man eine 0 ein. [ENTER]

6. Die entsprechende Laufwerk-Nummer: man gibt eine 1 ein.
7. Hat man die gemachten Eingaben noch einmal auf ihre Richtigkeit überprüft, drückt man die [ENTER] -Taste.
8. In Die Zeile(SOURCE DISK) zeigt an, welche Spur auf der zu kopierenden Diskette gerade gelesen wurde.
9. In Die Zeile (DESTINATION DISK) zeigt an, welche Spur gerade übertragen wurde.
10. Diese Anzeige erscheint nach Abschluß des Kopiervorgangs. Sollen weitere Kopien angefertigt werden, gibt man 1, ansonsten eine 0 ein. 0 oder 1 [ENTER]

Zusammenfassende Darstellung der Bildschirmanzeige für die BACKUP-Routine:

Hat man bei der Eingabe für die zu kopierende Diskette ("Quelle") und für die leere Diskette, auf die kopiert werden soll ("Zieldiskette"), das gleiche Laufwerk benannt – weil z.B. nur ein Laufwerk vorhanden ist – dann erscheint nach dem Starten des Kopiervorgangs auf dem Bildschirm folgende Anzeige:

```
* INSERT SOURCE DISK AND PRESS ENTER KEY
```

Man left jetzt die "Quelle" in das Laufwerk ein und drückt anschließend die [ENTER] Taste. Nachdem eine Spur gelesen wurde, erscheint die Meldung:

```
* INSERT DESTINATION DISK AND PRESS ENTER KEY
```

Jetzt entnimmt man dem Laufwerk die "Quelle" und legt die "Zieldiskette" ein.
Dieser Vorgang ist so oft zu wiederholen, bis auf dem Bildschirm die Meldung erscheint:

```
* END Y (0) OR N (1) ?
```

Wird ein Fehler festgestellt, dann liegt die Ursache bei der "Quelle", wenn die Spurnummer (TRACK) der "SOURCE DISK" größer ist als die der "DESTINATION DISK". Sind dagegen beide Spurnummern gleich, dann ist die Ursache für den Fehler bei der "Zieldiskette" zu suchen.
Möchte man mit der Eingabe noch einmal von vorne beginnen, drückt man die Tasten [CTRL] und [HOME] . Will man dagegen den Kopiervorgang abbrechen und zu FDOS zurückkehren, gibt man [CTRL] [A] ein.

WICHTIGER HINWEIS

1. Bei Eingabe des BACKUP-Befehls muß sich die FDOS-Masterdiskette in Laufwerk A0 befinden.
2. Enthält die zu kopierende Diskette ein als geheim deklariertes Programm, kann sie nicht kopiert werden.
3. Auf eine Diskette mit dem Diskettennamen "SYSTEM" kann keine Kopie mittels des BACKUP-Befehls erfolgen. (Soll trotzdem auf die Diskette kopiert werden, dann ist die Diskette mittels der LPRINT-Anweisung umzubenennen (Seite 324).

COPY-Befehl

Zum Kopieren einzelner Dateien.

〈VERWENDUNG〉
Für das Kopieren von Programm- und Daten-Dateien.

FORMAT: COPY␣C␣D

␣ : Zwischenraum
C : Spezifizierung der zu kopierenden Datei
D : Spezifizierung der Dateikopie

ERLÄUTERUNG:

1. Kopiert den Inhalt der Datei mit der Dateikennung C in eine Datei mit der Dateikennumg D.
2. Bei FDOS wird die Dateikennung nicht wie in BASIC in Anführungsstriche [" "] gesetzt.

EINFACHES BEISPIELE:

1. COPY␣CALC:A0␣DISK:A0 [ENTER]
   ..... Kopiert die Datei mit dem Namen CALC in eine Datei mit dem Namen DISK auf derselben Diskette in Laufwerk A0.
2. COPY␣CALC:A0␣TEST,SHARP [ENTER]
   ..... Kopiert die Datei mit dem Namen CALC von der Diskette in Laufwerk A0 in eine Datei des Namens TEST auf die Diskette mit der Bezeichnung SHARP.

ANWENDUNGSBEISPIEL:

Kopieren einer Datei mit dem Namen A von der Diskette in Laufwerk A0 in eine Datei des Namens TEST auf eine Diskette mit der Bezeichnung SHARP in Laufwerk A1.
Eingabe:

COPY␣A:A0␣TEST,SHARP:A1 [ENTER]

Bildschirmanzeige:

```
FILE COPY  22007E(V1.0A)

* INSERT DISK AND PRESS ENTER KEY ⬚ ← Kursor blinkt hier
```

Jetzt schiebt man die Diskette mit der zu kopierenden Datei A in Laufwerk A0 und die Diskette mit der Diskettenbezeichnung SHARP in Laufwerk A1' Dann drückt man die [ENTER] -Taste. Nach Abschluß des Kopiervorgangs erscheint auf dem Bildschirm die Meldung:

```
   END
READY
```

Die Datei A wurde durch den Kopiervorgang nicht verändert.

HINWEIS

1. Bei der Eingabe des COPY-Befehls muß die FDOS-Masterdiskette in Laufwerk A0 sein.
2. Dateien, die mittels einer BASIC-Anweisung als geheim deklariert wurden, können nicht kopiert werden.

SCOPY-Befehl

Zum Kopieren einzelner Dateien mit einem Laufwerk.

〈VERWENDUNG〉
Für das Kopieren von Dateien mit einem Laufwerk (Single Drive).

FORMAT: SCOPY␣C␣D

␣ : Zwischenraum
C : Spezifizierung der zu kopierenden Datei
D : Spezifizierung der Dateikopie

ERLÄUTERUNG:

1. Kopiert die Datei mit der Bezeichnung C in eine Datei mit der Bezeichnung D unter Benutzung verschiedener Disketten aber nur eines Laufwerks.
2. Werden im Rahmen der Dateikennung von C und D verschiedene Laufwerke angegeben, dann wird die bei C spezifizierte Kanal-Laufwerk-Nummer benutzt.

EINFACHES BEISPIELE:

SCOPY␣CALC:A0␣TEST2, SHARP [ENTER]
..... Kopiert die Datei mit dem Namen CALC von der Diskette in Laufwerk A0 in eine Datei des Namens TEST auf die Diskette mit der Bezeichnung SHARP.

ANWENDUNGSBEISPIEL:

Man kopiere eine Datei mit dem Namen A von der Diskette mit der Bezeichnung SYSTEM in eine Datei mit Namen TEST auf eine Diskette mit der Bezeichnung SHARP. Wir legen die FDOS-Masterdiskette in das Laufwerk ein und geben ein: [ENTER]

Bildschirmanzeige:

```
FILE COPY  22007F(V1.0A)
* INSERT SOURCE DISK OK  ← Kursor blinkt hier
```

Jetzt legt man die Diskette mit der zu kopierenden Datei A in das Laufwerk ein und drückt die [ENTER] -Taste. Es erfolgt die Meldung:

```
* INSERT DESTINATION DISK OK  ← Kursor blinkt hier
```

Dann wechselt man die Diskette mit der zu kopierenden Datei gegen die Diskette, auf die die Datei geschrieben werden soll, aus und drückt dann wiederum die [ENTER] -Taste. Diesen Vorgang wiederholt man so lange bis die folgende Meldung ausgegeben wird:

(Bildschirmausgabe)

```
    END
READY
```

Damit zeigt das System das Ende des Kopiervorgangs an und erwartet einen neuen Befehl.
Der Inhalt von Datei A wird durch den Kopiervorgang nicht verändert.

WICHTIGER HINWEIS

1. Bei Eingabe des SCOPY-Befehls muß sich die FDOS-Masterdiskette in Laufwerk A0 befinden.
2. Dateien, die mittels einer BASIC-Anweisung als geheim deklariert wurden, können nicht kopiert werden.

KILLALL-Befehl

Zum Löschen aller ungeschützten Dateien auf einer Diskette.

〈VERWENDUNG〉
Befinden sich auf einer Diskette überflüssige Dateien, kann man sie mittels des KILLALL-Befehls löschen.

FORMAT: K I L L A L L ␣ CH m

␣ : Zwischenraum
CH : Kanalcode (A oder B)
m : Laufwerk-Nummer (0, 1, 2, 3)

ERLÄUTERUNG:

Löscht alle Dateien auf einer Diskette mit Ausnahme der geschützten Dateien.

ANWENDUNGSBEISPIEL:

Alle nichtgeschützten Dateien auf einer Diskette in Laufwerk A1 sollen gelöscht werden. Man legt zuerst die FDOS-Masterdiskette in Laufwerk A0 ein und gibt dann den Befehl ein: KILLALL ␣ A1 [ENTER] ..... Eingabebefehl

Darauf erfolgt auf dem Bildschirm die Meldung:

```
* INSERT DISK AND PRESS ENTER KEY
```

Jetzt schiebt man die Diskette mit den zu löschenden Dateien in Laufwerk A1 und drückt anschließend die [ENTER] -Taste. Ist der Löschvorgang abgeschlossen, erfolgt die Meldung:

```
 *END
READY
```

WICHTIGER HINWEIS

1. Bei Eingabe des KILLALL-Befehls muß sich die FDOS-Masterdiskette in Laufwerk A0 befinden.
2. Vor dem Beginn des Löschvorgangs sollte man prüfen, ob man auch die richtige Diskette in das bezeichnete Laufwerk eingelegt hat.
3. Mit [CL] oder [↑] kann man die Fehlersperre aufheben.

DEBUG-Befehl

Zur Beseitigung von Fehlern in Maschinenprogrammen (Debuggen).

〈VERWENDUNG〉
Diese Routine dient der Ausgabe und Veränderung des Arbeitsspeicherinhalts.

FORMAT: DEBUG

ERLÄUTERUNG:

Durch die Eingabe des DEBUG-Befehls wird das System in den Debuggermodus versetzt. In diesem Modus können sogenannte Debuggerbefehle benutzt werden (siehe Funktionen 1 bis 4).

Nach der Eingabe von DEBUG und dem anschließenden Drücken der [ENTER] -Taste erscheint am Bildschirm folgende Anzeige:

```
DEBUG  22007H(V1.0A)

USER MICRO AREA ..... ①
     START 9600 ..... ②
     END   9FFF ..... ③

*
```

← Kursor blinkt hier

1. Der Speicherbereich für ein vom Benutzer in Maschinensprache geschriebenes Programm (User Micro Area) wird mittels der NEW-Anweisung (Seite 287) oder ERASE-Anweisung (Seite 288) im BASIC—Operationsmodus definiert.
2. Startadresse des benutzerdefinierten Speicherbereichs. (Hierzu siehe den Speicherbelegungsplan im Anhang zum BASIC-Handbuch auf Seite 50.)
3. Endadresse des im Benutzerbereich des Arbeitsspeichers befindlichen Maschinenprogramms.

Mit [A] [CTRL] kann man aus dem Debuggermodus in den FDOS-Modus zurückkehren.

Befehle, die im Debuggermodus durchgeführt werden können, werden im Rahmen der folgenden vier Funktionen behandelt:

● **FUNKTION 1:** Gibt den Inhalt des Speichers über den Bildschirm aus.

FORMAT: D␣AAAA␣BBBB

␣ : Zwischenraum
AAAA : Startadresse (hexadezimal)
BBBB : Endadresse (hexadezimal)

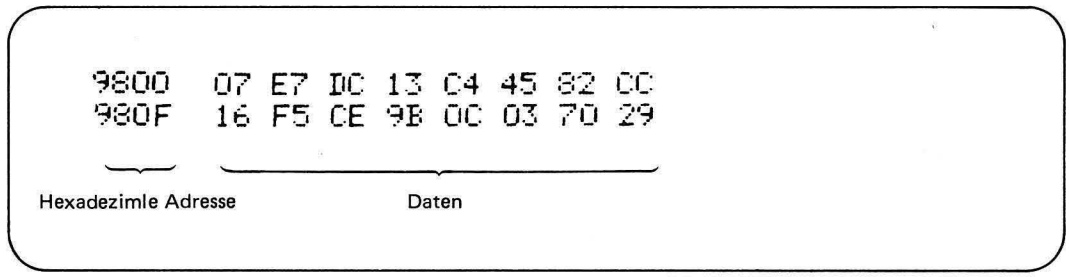
ERLÄUTERUNG:

1. Der D-Befehl gibt den Inhalt des Arbeitsspeichers von der Speicheradresse AAAA bis zur Speicheradresse EEEE (hexadezimal) am Bildschirm aus.
2. Die Ausgabe kann durch Drücken der Leerzeichentaste (SPACE) angehalten werden. Ein nochmaliges Drücken der Leerzeichentaste setzt die Ausgabe fort. Die Ausgabe kann (auch während der Unterbrechung) durch Drücken der [ENTER] -Taste abgebrochen werden. Das System erwartet dann einen neuen Debuggerbefehl.
3. Über den D-Befehl kann nur der Benutzerbereich des Speichers ausgegeben werden.

BEISPIEL:

D␣9800␣980F [ENTER] ..... gibt den Inhalt des Arbeitsspeichers von der Adresse &9800 bis zur Adresse &980F am Bildschirm aus.

Bildschirmausgabe:



● **FUNKTION 2:** Gibt den Inhalt des Speichers über den Drücker aus.

FORMAT: P␣AAAA␣BBBB

␣ : Zwischenraum
AAAA : Startadresse (hexadezimal)
BBBB : Endadresse (hexadezimal)

ERLÄUTERUNG:

1. Der P-Befehl gibt den Inhalt des Arbeitsspeichers von der Speicheradresse AAAA bis zur Speicheradresse EEEE (hexadezimal) über den Drucker aus.
2. Die Ausgabe kann durch Drücken der Leerzeichentaste (SPACE) angehalten werden. Ein nochmaliges Drücken der Leerzeichentaste setzt die Ausgabe fort. Die Ausgabe kann – auch während der Unterbrechung – durch das Drücken der [ENTER] -Taste abgebrochen werden. Das System erwartet dann einen neuen Debuggerbefehl.
3. Über den P-Befehl kann nur der Benutzerbereich des Speichers ausgegeben werden.

BEISPIEL:

P␣9800␣980F [ENTER] ..... gibt den Inhalt des Arbeitsspeichers von der Adresse &9800 bis zur Adresse &980F über den Drucker aus.

● **FUNKTION 3:** Ändert den Inhalt des Speichers.

FORMAT: M␣AAAA

␣ : Zwischenraum
AAAA : Speicheradresse (hexadezimal), deren Inhalt geändert werden soll

ERLÄUTERUNG:

1. Mittels des M-Befehls ist es möglich, den Inhalt einer Speicherzelle in dem für den Benutzer reservierten Speicherbereich zu verändern. Nach der Eingabe wird zunächst der Inhalt der durch AAAA angegebenen Adresse am Bildschirm angezeigt. Man gibt jetzt die neuen Daten in Form einer zweistelligen Hexadezimalzahl ein und drücket anschließend die [ENTER] -Taste. Der Inhalt der Speicherzelle ist jetzt geändert.

2. Soll keine Änderung erfolgen, dann kann man den Inhalt der folgenden Adresse durch Drücken der [↓] Taste bzw. den Inhalt der vorausgehenden Adresse durch Drücken der [↑] Taste anzeigen.
   Drückt man die [ENTER] -Taste ohne vorherige Dateneingabe, dann befindet man sich wieder im Debuggermodus und kann einen neuen Debuggerbefehl eingeben. Der Inhalt des Speichers hat sich in diesem Fall nicht verändert.

BEISPIEL:

M␣9A00 [ENTER] ..... gibt den Inhalt der Speicheradresse 9A00 aus und wartet auf die Eingabe des neuen Inhalts.

Bildschirmanzeige:

```
* M 9A00
   9A00   F3 -
```

● **FUNKTION 4:** Speichert den Inahlt des Speichers auf der Diskette ab.

FORMAT: S␣C␣AAAA␣BBBB

␣ : Zwischenraum
C : Dateikennung
AAAA : Anfangsadresse für die Abspeicherung (hexadezimal)
BBBB : Endadresse für die Abspeicherung (hexadezimal)

ERLÄUTERUNG:

Der S-Befehl speichert den Inhalt des Arbeitsspeichers von der Adresse AAAA bis zur Adresse EEEE auf einer Diskette unter der Dateikennung C ab.
Eine mittels des S-Befehls abgespeicherte Datei kann mit der BASIC-Anweisung MLOAD in den Arbeitsspeicher gelesen werden (Seite 374).

BEISPIEL:

S␣MICRO:A1␣9000␣9FFF [ENTER] ···speichert den Inhalt des Speichers von der Adresse 9000 bis 9FFF auf der Diskette in Laufwerk A1 unter dem Namen MICRO ab.

WICHTIGER HINWEIS

1. Durch Drücken der [CL] oder der [↑] Taste kann man den Fehlersperre aufheben.
2. Durch Eingabe von [CTRL] [A] kehrt man in die FDOS-Operationsebene zurück.

6. FREI DEFINIERBARE FUNKTIONSTASTEN

(1) Belegen der freien Funktionstasten

Mit der DFK-Anweisung werden die frei definierbaren Funktionstasten mit einem vom Anwender frei wählbaren Text belegt. Die Belegung der Tasten nach der Ausführung des FDOS-Befehls BASIC (siehe Seite 44) ist in der folgenden Tabelle aufgeführt. Diese Belegung bleibt bis zur Neudefinition durch eine DFK-Anweisung erhalten.

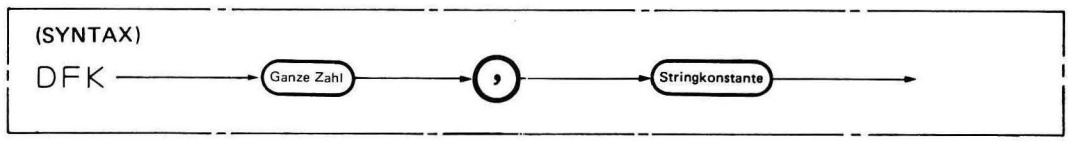
Taste Nr.	1	2	3	4	5	6	7	8	9	10
Zeichen	+	-	*	/	(	)	=	"	$	,

Nach der Ausführung einer DFK 0-Anweisung gilt wieder obige Tastenbelegung.

● **FUNKTION 1:** Definition eines beliebigen Textes.

FORMAT: DFK m, S

m : Nummer der freidefinierbaren Taste (1 – 20)
S : Definition



ERLÄUTERUNG:

Die Stringkonstante S wird als neuer Text für die angegebene Taste gespeichert. Das Zeichen @ im String steht für die [ENTER] -Taste, die auf diese Weise einer Taste zugeordnet werden kann.

Der String kann bis zu 12 Zeichen lang sein. Sollen längere Texte zugeordnet werden, so füge man ein + hinter dem Ende des Strings an. In diesem Fall wird die Definition der folgenden Taste mit hinzugenommen.

EINFACHES BEISPIELE:

Taste 1 wird der Text ABC····· XYZ, Taste 2 der Text LMN····· XYZ und Taste 3 der Text WXYZ zugeordnet.

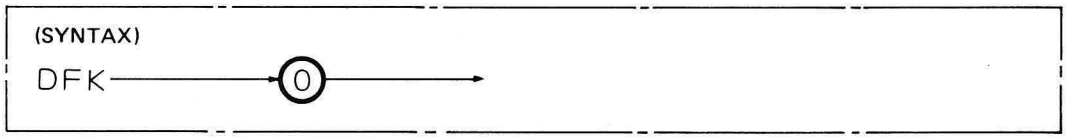
```
DFK 1,"ABCDEFGHIJK"+ [ENTER]
DFK 2,"LMNOPQRSTUV"+ [ENTER]
DFK 3,"WXYZ" [ENTER]
```

WICHTIGER HINWEIS

Zeichen hinter @ sind ungültig!

● **FUNKTION 2:** Definition der BASIC-Standardbelegung.

FORMAT: DFK 0



ERLÄUTERUNG:

Die Tasten [DEF1] bis [DEF10] werden automatisch mit den Zeichen +, –, *, /, (,) , = , " , $ und, belegt. Mit der Betätigung der Funktionstaste können dann die genannten Zeichen ohne Rücksicht auf den Tastaturmodus (z.B. Shift-Taste) eingegeben werden.

(2) Anwendungsbeispiele

Die gewöhnlichen Tasten auf der Tastatur geben bei jeder Betätigung nur ein einziges Zeichen ab. Anders jedoch die frei definierbaren Funktionstasten, die bei entsprechender Belegung die Eingabe beliebiger Zeichenfolgen oder Ausführung ganzer Befehle mit einem einzigen Tastendruck gestatten. Sie stellen damit eine wesentliche Erleichterung für den Umgang mit der Tastatur dar.
Wenn z.B. dieselbe Zeichenfolge mehrfach innerhalb eines Programms auftaucht, so ist die Zuordnung dieser Zeichenfolge zu einer frei definierbaren Taste erheblich bequemer und weniger anfällig gegen Tippfehler, als wenn man jedesmal aufs neue den gesamten Text von Hand eingeben wollte.

BEISPIEL:

```
10 DISP TAB 10;"LIST "
20 READ A,B
30 DISP TAB 10;A
40 DISP TAB 10;B
50 READ C$,D$
60 DISP TAB 10;C$
70 DISP TAB 10;D$
```

In diesem Programm kommt der Text DISP TAB 10; mehrfach vor. Zur Vereinfachung der Arbeit soll der Text einer frei definierbaren Taste, z.B. der Taste [DEF1] , zugeordnet werden. Der Befehl dazu lautet:

DFK 1,"DISP TAB 10;"[ENTER]

Gibt man nun das Programm ein, so kann Zeile 10 auf einfache Weise mit Hilfe der [DEF1] -Taste geschrieben werden: 10 [DEF1] "LIST" [ENTER]

Ähnlich verfährt man bei den Zeilen 30, 40, 60 und 70.
Ebenso kann man Befehle und Anweisungen, die man beim Programmieren häufig benötigt, frei definierbaren Tasten zuordnen.

BEISPIEL:

Die Befehle bzw. Anweisungen TR OFF, LIST und DISP sollen den Tasten DEF 2 , DEF 3 und DEF 4 zugeordnet werden. Die Eingabe lautet:

```
DFK 2,"TR OFF@"[ENTER]
DFK 3,"LIST@"[ENTER]
DFK 4,"DISP"[ENTER]
```

Das an die Definitionen der Tasten DEF 2 und DEF 3 angehängte @ wirkt wie die Betätigung der [ENTER] -Taste, so daß diesen Tasten der Vollständige Befehl zugeordnet wurde. Bei der DISP-Anweisung hätte das @ i. a. wenig Sinn. Zum Auflisten des Programms genügt nun das Antippen der Taste [DEF3] anstelle der manuellen Eingabe LIST [ENTER] .

(3) Auflisten und Löschen der frei definierbaren Tasten

a) Auflisten der Tastenzuordnungen
Um die Tastenzuordnungen auf dem Bildschirm anzuzeigen, gebe man ein (siehe auch Seite 88): LIST DFK [ENTER]

b) Löschen der Tastenzuordnungen
Mit Hilfe des Befehls
DELETE DFK [ENTER]
(siehe auch Seite 93) werden alle Tastenzuordnungen aufgehoben.

II. PROGRAMMIERUNG IN BASIC

1. ALLGEMEINES ZUR PROGRAMMIERUNG

(1) Aufbau eines BASIC-Programms

Unter einem Programm versteht man eine Beschreibung der durch den Computer auszuführenden Tätigkeiten in einer Form, die der Computer "Verstehen" kann. Ein Programm wird in einer Programmiersprache geschrieben. Dieses Handbuch erläutert die Programmiersprache BASIC. Ein in BASIC geschriebenes Programm besteht aus einer Reihe von Programmzeilen. Jede BASIC-Programmzeile beginnt mit einer Zeilennummer, gefolgt von einer oder mehreren BASIC-Anweisungen.

①Eine Programmzeile (im folgenden kurz "Zeile" genannt) hat zum Beispiel folgendes Aussehen:

```
10    DISP "START"
```

Zeilennummer: `10`
Anweisung: `DISP "START"`
Programmzeile: `10 DISP "START"`

②Schreibt man mehrere Anweisungen in dieselbe Zeile, so muß man die einzelnen Anweisungen durch Doppelpunkte (:) voneinander trennen.

BEISPIEL:

```
10 DISP "START": INPUT A
```

Anweisung: `DISP "START"`
Anweisung: `INPUT A`

③Beim Eintippen einer Zeile muß die Zeilennummer am Anfang stehen. Das zeigt dem Interpreter an, daß hier eine Programmzeile eingegeben wird, und bewirkt die Speicherung der Zeile im BASIC-Programmspeicher.

④Als Zeilennummern sind ganze Zahlen im Bereich von 0 bis 99999 je einschließlich zulässig.

⑤Eine Zeile kann bis zu ungefähr 250 Zeichen, die Zeilennummer mitgezählt, lang sein.

WICHTIGER HINWEIS

1. Leerzeichen (spaces) zählen bei der Zeilenlänge mit! Wenn viele Zeichen in eine Zeile geschrieben werden sollen, kann man die Leerzeichen hinter Schlüsselworten weglassen.
2. Die maximale Anzahl von Zeichen pro Zeile hängt ab von den dort benutzten Schlüsselworten, Daten usw.. Überschreitet man bei der Eingabe der Zeile diese Maximalzahl, erfolgt eine Fehlermeldung (ERROR 8).
3. Beim Listen oder Editieren einer langen Zeile werden bis zu 245 Zeichen (Zeilennummer nicht mitgezählt) ausgegeben. Das 246. und alle folgenden Zeichen werden durch einen senkrechten Strich (¦) symbolisiert.

Hier folgt ein einfaches Beispiel für ein BASIC-Programm, welches Umfang und Fläche von Kreisen nach Eingabe des Radius berechnet:

```
10 INPUT "RADIUS =";R ..... Ausgabe von RADIUS auf dem Bildschirm, Warten auf Eingabe, Zuweisen der eingegebenen Zahl an die Variable R.
20 IF R<=0 GO TO 80 ..... Sprung nach Zeile 80, wenn für R eine negative Zahl oder 0 eingegeben wurde.
30 P=2*PI*R ..... Berechnen des Ausdrucks 2 x π x R und Zuweisen des Ergebnisses an die Variable P.
40 DISP "UMFÄNG =";P ..... Bildschirmausgabe der Stringkonstanten UMFANG = und des Inhalts von P.
50 S=PI*R^2 ..... Berechnung von π x R² und Zuweisung an die Variable S.
60 DISP "FLÄCHE =";S ..... Bildschirmausgabe der Stringkonstanten FLÄCHE = und des Inhalts von S.
70 GO TO 10 ..... Sprung nach Zeile 10.
80 DISP "ENDE" ..... Ausgabe des Texts ENDE am Bildschirm.
90 END ..... Abschluß des Programmlaufs.
```

Nach der Eingabe des Radius berechnet das Programm Umfang und Fläche des Kreises und zeigt die Ergebnisse auf dem Bildschirm an. Dieser Vorgang wiederholt sich, bis man für den Radius 0 oder einen negativen Wert eingibt.
Die Bedeutungen der einzelnen Anweisungen sind in den Anmerkungen rechts neben dem Programm erläutert. Einzelheiten siehe III. 3. "Standard-Anweisungen".

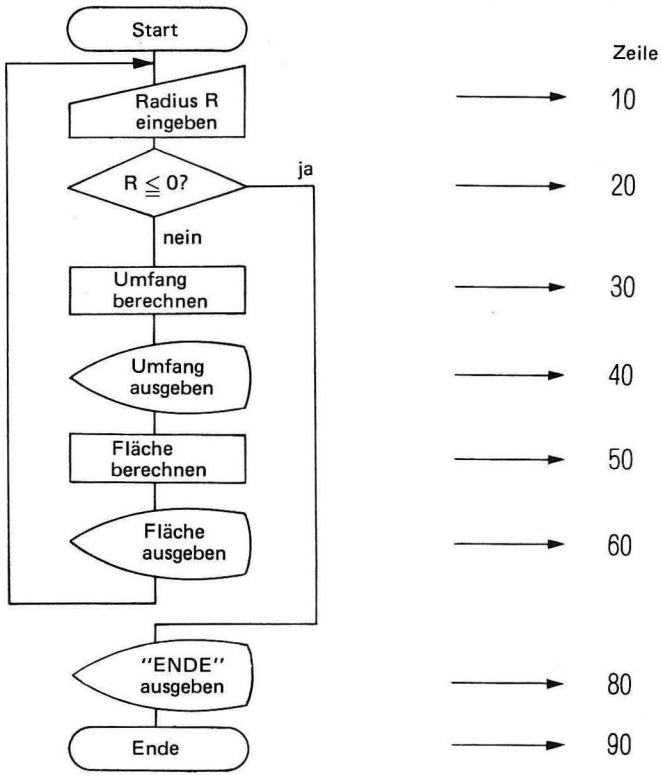
(2) Programmieren

Programmieren heißt, ein Programm zu erstellen. In der Regel erfordert das folgende Schritte:

①Sorgfältige und vollständige Analyse des Problems, bei dessen Lösung der Computer eingesetzt werden soll.
②Entscheidung über das Verfahren, nach dem der Computer die Aufgabe bearbeiten soll, den sog. "Algorithmus".
③Entwurf eines Ablaufdiagramms.
④Niederschrift des Programms in der gewählten Programmiersprache.
⑤Eingabe des Programms über die Tastatur, Speichern und probeweises Ausführen.
⑥Sorgfältige und vollständige Prüfung des Programms, Beseitigung aller Fehler.
⑦Dokumentation des Programms: Programmbeschreibung, Ablaufdiagramm, Listings, Ergebnisse von Tests, Beschreibung der Ein- und Ausgaben, Bedienungsanleitung usw.

In dem einfachen Beispiel oben war es möglich, die Programmentwicklung mit Schritt⑤ zu beginnen. Die Schritte ①, ② und ③ sind jedoch unabdingbar bei der Erstellung großer Programme oder ganzer Programmsysteme.

Zu 3, Ablaufdiagramm:
Ein Ablauf- oder Flußdiagramm ist eine graphische Darstellung, die den Ablauf eines Programms veranschaulicht. Das Ablaufdiagramm des Beispielprogramms sieht wie folgt aus:



Entwurf von Ablaufdiagrammen und Erläuterung der Symbole sind nicht Inhalt dieser Schrift.

Zu ④, Eingabe von Programmen:
BASIC-Programme sind auf diesem System wie folgt einzugeben:

① Betriebsartschalter in Stellung PRO bringen, nähere Einzelheiten im Bedienerhandbuch.
② Zuerst die Zeilennummer, dann die Anweisung(-en) der Zeile über die Tastatur eingeben. Nicht vergessen, am Ende einer Zeile die [ENTER] -Taste zu drücken.
③ Schritt ② für alle Programmzeilen wiederholen.

So hat man z.B. bei der Eingabe von Zeile 10 des Beispielprogramms die folgenden Tasten zu betätigen:

[1][0][ ][I][N][P][U][T]["][R][A][D][I][U][S][ ][=]["][;][R][ENTER]

[ ] bedeutet "Leertaste". Sie ändert nicht die Bedeutung der Programmzeile, sondern soll die Darstellung des Programms auf dem Bildschirm leichter lesbar machen. Der in Anführungszeichen (") stehende Teil ist eine Stringkonstante. Sie wird (in dieser Zeile) von der INPUT-Anweisung (siehe Seite 107) genau so auf dem Bildschirm dargestellt, wie sie im Programm niedergeschrieben wurde.

Nachdem das Programm fertig eingetippt ist, gebe man

LIST [ENTER]

ein. Der LIST-Befehl zeigt das zur Zeit im Speicher befindliche Programm auf dem Bildschirm in der Reihenfolge der Zeilennummern an. An Hand des "Listings" (wie man kurz dazu sagt) auf dem Bilschrim kann man dann prüfen, ob das Programm richtig eingegeben wurde.

WICHTIGER HINWEIS

1. Wenn man vergißt, am Ende einer Programmzeile [ENTER] zu drücken, wird die betreffende Zeile ausgelassen oder die nächste eingegebene Zeile mit der vorigen verschmolzen. Die [ENTER] -Taste zeigt dem Interpreter das Ende einer Zeile an.
2. Bei der Eingabe eines Programms ist die Reihenfolge der Zeilennummern ohne Bedeutung. Die Zeilen werden gemäß ihrer Nummer automatisch richtig in das Programm eingeordnet.

Zur Ausführung des Programms:
Ein Programm wird durch Betätigen der [RUN] -Taste oder durch die Eingabe von RUN [ENTER] auf der Tastatur gestartet. Im Beispielprogramm geschieht dann folgendes:

1. Die [RUN] -Taste werde betätigt. Auf dem Bildschirm erscheint die Meldung

```
RUN
RADIUS= ■
```

2. Gibt man nun z.B. 10 [ENTER] an der Tastatur ein, werden Umfang und Fläche berechnet und auf dem Bildschirm ausgegeben.

```
>RUN
RADIUS =10
UMFANG = 62.8318530716
FLäCHE = 314.159265358
RADIUS = ■
```

3. Für jede Zahl, die als Radius eingegeben wird, wiederholt sich das Spiel aufs neue, mit einer Ausnahme:
4. Gibt man 0 oder einen negativen Wert für den Radius ein, beendet das Programm seinen Lauf, nachdem es auf dem Bildschirm ausgegeben hat:

```
RADIUS =-1
ENDE
READY
> ■
```

(Die Meldung READY kommt nicht vom Programm, sondern vom BASIC-Interpreter, der damit seine Bereitschaft zur Annahme neuer Befehle signalisiert.)

Wurde das Programm fehlerhaft eingegeben oder enthält es selbst noch Fehler, erfolgt eine Fehlermeldung oder es werden unsinnige Werte ausgegeben. Wenn das der Fall ist, muß das Programm korrigiert werden.
Siehe dazu 2. "Korrigieren und Editieren" auf Seite 66.

Zu ⑦, Dokumentation

Während der Entwicklung des Programms hält man die Dokumentation stets auf dem aktuellen Stand und sieht sie nach der Fertigstellung noch einmal sorgfältig durch. Die Dokumentation muß alle notwendigen Informationen enthalten, damit das Programm auch von anderen Personen als dem Programmierer verstanden und richtig angewendet oder später leichter geprüft, korrigiert und geändert werden kann.

WICHTIGER HINWEIS

Jede Programmiersprache hat ihre besonderen Merkmale. Bei BASIC-Programmen ist es besonders wichtig, die Einsprungstellen zu dokumentieren, d.h. die Zeilennummern, die Ziel einer Sprunganweisung (GO TO, GO SUB) sind.

Wie schreibt man gute Programme ?

Ein einfaches Programm ist leichter zu prüfen, leichter zu korrigieren, leichter anzuwenden. Man beachte dazu unter anderem folgende Gesichtspunkte:

1. Man erwerbe sich Kenntnisse über Programmierung und klaren Aufbau von Programmen.
2. Man vermeide komplizierte logische Strukturen im Programm.
3. Man strukturiere das Programm, d.h. teile es in logisch zusammengehörige Blöcke auf, und füge ausreichend Kommentare mit Hilfe der REM-Anweisung (siehe Seite 125) ein.
4. Kommt derselbe Verarbeitungsgang mehrfach vor, schreibe man für diesen Teil ein Unterprogramm, das vom Hauptprogramm aufgerufen wird. Auf diese Weise kann man den Programmablauf leicht verfolgen, indem man nur das Hauptprogramm liest. Außerdem reduziert sich die Anzahl der Programmzeilen, wodurch das Programm leichter zu warten ist.
5. Ein gutes Programm führt den Bediener. Erwartet es eine Eingabe auf der Tastatur, informiert es stets darüber, welche Daten es nun erwartet und ggf. welches Format bei der Eingabe einzuhalten ist.
   Bestehen an einem bestimmten Punkt im Programmablauf mehrere Wahlmöglichkeiten über den weiteren Fortgang so hat sich die sog. "Menü-Technik" bewährt, d.h. das Programm führt alle in Frage kommenden Zweige auf dem Bildschirm auf und überläßt dem Bediener die Wahl durch Antippen einer Zahl oder eines Buchstabens.
6. Mit den vielfätigen Text- und graphischen Darstellungsmöglichkeiten dieses Systems ist es leicht, ein gut gegliedertes und damit leicht lesbares Schirmbild zu erzielen.

Das erste Ziel beim Schreiben eines Programms ist immer, ein korrektes Programm zu erstellen. Später kann man es optimieren, um Laufzeit, Speicherbedarf und Ein/Ausgaben zu verbessern. Dafür eignet sich BASIC sehr gut, weil man BASIC-Programme besonders leicht ändern kann.

Für das Aufspüren und Verbessern von Programmfehlern stellt dieser BASIC-Interpreter leistungsfähige Testhilfen zur Verfügung, die die Programmentwicklung sehr erleichtern.

SHARP wünscht Ihnen viel Erfolg beim Programmieren in BASIC!

2. KORRIGIEREN UND EDITIEREN

Tippfehler bei der Eingabe oder logische Fehler im Aufbau eines Programms bewirken, daß es nicht wie vorgesehen arbeitet. In solchen Fällen muß das Programm korrigiert werden. Unter "Editieren" versteht man allgemein das Ändern von Programmen. Folgende Möglichkeiten bieten sich an:

1. Korrigieren von früher eingegebenen Zeilen
2. Einfügen neuer Zeilen
3. Löschen überflüssiger Zeilen

WICHTIGER HINWEIS

Programmzeilen können nur in Stellung PRO des Betriebsartschalters editiert werden!

(1) Ändern von Programmen

Zur Korrektur einer Programmzeile muß diese zum Editieren aufgerufen werden, dann kann man weiter mit den Tasten [CL] , [←] oder [→] arbeiten.

a) Stößt der Interpreter bei der Ausführung eines Programms auf einen Fehler, so stoppt er und zeigt die Ursache des Fehlers an. Drückt man nun die [↑]-Taste, wird die Fehlersperre aufgehoben und die fragliche Zeile auf dem Bildschirm ausgegeben.
Sei zum Beispiel in Zeile 10 des Beispielprogramms (siehe Seite 61) aus Versehen INPUG statt INPUT eingegeben worden, dann wird die Aufführung des Programms in Zeile 10 abgebrochen und in der unteren Bildschirmzeile ausgegeben:

```
>RUN


  SYNTAX ERROR IN LINE 10
```

Nach dem Betätigen der [↑] -Taste wird die Fehlersperre gelöst, Zeile 10 angezeigt und der Kursor fährt an die Stelle unmittelbar hinter dem Fehler:

```
>RUN
>10  INPUG "RADIUS =";R
           ↑
        Kursor
```

Nun kann die fehlerhafte Zeile 10 korrigiert werden.

b) Gibt man eine Zeile neu ein und bemerkt unmittelbar nach dem Drücken der [ENTER] -Taste einen Fehler (bevor die [ENTER] -Taste ein zweites Mal gedrückt wurde), so betätige man die [↑] -Taste. Der Kursor springt an das Ende der Zeile zurück, und man kann mit der Korrektur beginnen.

c) Kann eine Zeile nicht mit der [↑] -Taste zum Editieren abgerufen werden, so benutze man die [EDIT] -Taste. Siehe auch EDIT-Befehl auf Seite 90.
Soll zum Beispiel Zeile 10 zum Editieren aufgerufen werden, so lautet die Tastenfolge.

[EDIT] [1] [0] [ENTER]

Der Kursor springt an den Anfang des ersten Nicht-Leerzeichens nach der Zeilennummer,

```
>10  INPUT "RADIUS =";R
     ↑
   Kursor
```

und die Zeile kann editiert werden. Soll die unmittelbar nachfolgende Zeile editiert werden, so läßt sich das auch ohne Eingabe einer Zeilennummer durch Betätigen der Tasten

[EDIT] [ENTER]

erreichen. Die unmittelbar davor liegende Zeile erreicht man, indem man statt der Zeilennummer einen beliebigen Buchstaben (A – Z, a – z) eingibt. Beispiel:

[EDIT] [D] [ENTER]

(2) Der Korrekturvorgang

a) Korrektur einer ganzen Zeile

① Drücken der [CL] -Taste zum Löschen der Zeile.
② Neueingabe der Zeile mitsamt Zeilennummer an der Tastatur.
③ Abschluß der Eingabe mit der [ENTER] -Taste.

b) Korrektur eines Teils einer Zeile

① Kursor mit den Tasten [←] oder [→] auf das zu korrigierende Zeichen fahren.
② Neues Zeichen an der Tastatur eingeben.
③ [ENTER] -Taste drücken.

c) Teil einer Zeile löschen

① Kursor auf das erste zu löschende Zeichen fahren.
② So oft die [DEL] -Taste drücken, wie Zeichen zu löschen sind. Jedes Drücken der [DEL] -Taste bewirkt das Löschen eines Zeichens.
③ [ENTER] -Taste drücken.

d) Zeichen in eine Zeile neu einfügen

① Kursor auf das Zeichen hinter der Stelle fahren, wo eingefügt werden soll.
② So oft die [INS] -Taste drücken, wie Zeichen zu löschen sind. Jedes Drücken der [INS] -Taste bewirkt das Löschen eines Zeichens.
③ Die einzufügenden Zeichen an der Tastatur eingeben.
④ [ENTER] -Taste drücken.

BEISPIEL:

Nehmen wir an, im Speicher steht folgendes Programm:

```
10 INPUT A,B,C
20 D=B*B-A*C
30 PRINT D
40 END
```

Zeile 20 und 30 sollen so geändert werden, daß das Programm danach wie folgt aussieht:

```
10 INPUT A,B,C
20 D=B*B-4*A*C
30 DISP D
40 END
```

Schritt	Tasten	Anzeige (nur geänderte Teile)
	Betriebsartschalter: PRO	
1	EDIT 2 0	>EDIT 20 □
2	ENTER	> 20 D =B*B-A*C (Kursor)
3	→ → → → → →	> 20 D =B*B-A*C
4	INS INS	> 20 D =B*B- □ A*C
5	4 *	> 20 D =B*B-4*A*C
6	ENTER	> □
7	EDIT	>EDIT □
8	ENTER	> 30 PRINT D
9	D I S P	> 30 DISP T D
10	DEL	> 30 DISP □ D
11	ENTER	> □

Um sich davon zu überzeugen, daß die Zeile nach der Änderung korrekt ist, kann man sie entweder mit der EDIT -Taste aufrufen oder mit einem LIST-Befehl den gesamten interessierenden Programmteil auf dem Bildschirm anzeigen.
In obigem Beispiel wurde bei Schritt 3 die → -Taste sechsmal nacheinander betätigt. Der Kursor kann aber auch durch Niederhalten der Taste bewegt werden – Tastenwiederholfunktion, siehe Seite 23 im Bedienerhandbuch.

(3) Einfügen und Löschen von Zeilen

a) Zeilen einfügen

Programme müssen nicht in der Reihenfolge aufsteigender Zeilennummern eingegeben werden. Daher ist es möglich, nachträglich Zeilen einzufügen.
Zum Einfügen einer neuen Zeile gibt man ihr eine Zeilennummer, die zwischen den Nummern der beiden Zeilen liegt, an der sie ins Programm eingesetzt werden soll. Der Betriebsartschalter muß dazu in Stellung PRO gebracht sein.
Zur Demonstration soll in das Programm auf Seite 62, welches Umfang und Fläche von Kreisen berechnet, eine Zeile zur Berechnung und Ausgabe des Druchmessers eingefügt werden. Die neue Zeile soll zwischen die Zeilen 20 und 30 zu stehen kommen, so daß sie eine Zeilennummer zwischen 21 und 29 je einschließlich bekommen kann, also z.B. 25. Die Eingabe lautet:

```
25 DISP "DURCHMESSER =";2*R ENTER
```

Danach hat das Programm folgende Gestalt:

```
10 INPUT "RADIUS =";R
20 IF R<=0 GO TO 80
25 DISP "DURCHMESSER =";2*R
30 P=2*PI*R
40 DISP "UMFANG =";P
50 S=PI*R^2
60 DISP "FLÄCHE =";S
70 GO TO 10
80 DISP "ENDE"
90 END
```

Folgen die Zeilennummern von und nach der Einfügungsstelle unmittelbar aufeinander, so daß keine Zeile mehr "dazwischen paßt", kann man mit dem REN-Befehl (siehe Seite 98) die Zeilen neu numerieren, um eine Lücke zu schaffen.

b) Löschen von Zeilen

Zum Löschen von Zeilen muß der Betriebsartschalter in Stellung PRO stehen.
Einzelne Zeilen werden gelöscht, indem man die Zeilennummer der zu löschenden Zeile eingibt und unmittelbar dahinter die [ENTER] -Taste drückt.
Um zum Beispiel Zeile 100 zu löschen, lautet die Eingabe:
100 [ENTER]
Zum Löschen mehrerer aufeinanderfolgender Zeilen ist es bequemer und sicherer, den DELETE-Befehl anzuwenden, siehe Seite 93.

Vom richtigen Einstezen oder Löschen von Zeilen kann man sich mit Hilfe eines LIST-Befehls leicht überzeugen.

3. TESTHILFEN

Der BASIC-Interpreter findet und meldet zwei Arten von Fehlern. Das sind zum einen Fehler wie Funktionsaufrufe mit unzulässigen Argumenten, so etwa der Versuch, die Wurzel aus einer negativen Zahl zu ziehen oder einen zu langen String einer Variablen zuzuweisen, und zum anderen Anweisungen, die nicht mit den in diesem Handbuch beschriebenen Regeln übereinstimmen, sog. "Syntaxfehler".
Es gibt aber noch weitere Möglichkeiten für Fehler innerhalb eines Programms, die sich z.B. in unsinnigen Ausgabedaten zeigen oder darin, daß das Programm in eine endlose Schleife läuft. Fehler dieser Art rühren in der Regel von einer fehlerhaften Programmkonstruktion, von beim Eingeben vergessenen Zeilen und ähnlichen Ursachen her und können unter Umständen sehr schwer zu finden sein.
Aber auch für solche Fälle bietet dieser BASIC-Interpreter wirkungsvolle Testhilfen an: Einzelschrittest und Ablauftest.

(1) Einzelschrittest

In diesem Modus wird das Programm Zeile für Zeile einzeln ausgeführt. Der Betriebartschalter muß dazu in Stellung OP gebracht werden.

a) Einzelschrittest ab der ersten Zeile

① CTRL -Taste niederhalten und DEB -Taste drücken.
② In der Folge wird bei jedem Betätigen der DEB -Taste genau eine Programmzeile ausgeführt.

Am Beispiel des Kreisberechnungsprogramms von Seite 62 soll der Vorgang verdeutlicht werden. Der Vollständigkeit halber wird es an dieser Stelle noch einmal aufgeführt:

```
10 INPUT "RADIUS =";R
20 IF R<=0 GO TO 80
30 P=2*PI*R
40 DISP "UMFANG =";P
50 S=PI*R^2
60 DISP "FLäCHE =";S
70 GO TO 10
80 DISP "ENDE"
90 END
```

Tätigkeit	Bildschirmanzeige	Ausgef. Zeile
Betriebsartschalter: OP		
CTRL gehalten, DEB	>RUN/	
(ab hier ohne CTRL) DEB	>CONT/	Führt Zeile 10 aus
	RADIUS =□	Führt Zeile 10 aus
10 ENTER	RADIUS =10	Führt Zeile 10 aus
DEB	>CONT/	Führt Zeile 20 aus
DEB	>CONT/	Führt Zeile 30 aus
DEB	>CONT/	Führt Zeile 40 aus
	UMFANG = 62.8318530716	
DEB	>CONT/	Führt Zeile 50 aus
DEB	>CONT/	Führt Zeile 60 aus
	FLACHE = 314.159265358	
DEB	>CONT/	Führt Zeile 70 aus
DEB	>CONT/	Führt Zeile 10 aus
	RADIUS =□	
9999 ENTER	RADIUS =-1	Führt Zeile 10 aus
DEB	>CONT/	Führt Zeile 20 aus
DEB	>CONT/	Führt Zeile 80 aus
	ENDE	
DEB	>CONT/	Führt Zeile 90 aus
	>READY	
	>	

In Spalte "Bildschirmanzeige" sind nur die Änderungen aufgeführt.

b) Einzelschrittest ab einer Zeile mitten im Programm

① Eingeben: RUN/n ENTER
n steht für die Zeilennummer, ab der die schrittweise Ausführung beginnen soll.

② In der Folge wird bei jedem Betätigen der DEB -Taste genau eine Programmzeile ausgeführt.

WICHTIGER HINWEIS

Nach der Ausführung jeder Zeile im Einzelschrittmodus erscheint die Meldung CONT/ auf dem Bildschirm. Dadurch werden jedoch Ausgaben des zu testenden Programms selbst zerrissen. Wenn das stört, dann betätige man vor Testbeginn in Stellung PRO des Betriebsartschalters die Tasten

(PRO-Stellung)
EDIT / ENTER

und fährt danach erst mit den oben beschriebenen Operationen fort. Damit werden die Ausgaben des Testhilfesystems auf die untere Bildschirmzeile umgeleitet und das Schirmbild bleibt weitgehend erhalten.

BEISPIEL:
```
10 C$="ABC"
20 FOR I=1 TO 5
30 CURSOR 10,I:DISP I,C$
40 NEXT I
50 END
```

Wenn dieses Programm im Einzelschrittest ausgeführt wird, ergibt sich folgendes Schirmbild:

```
>RUN ⏎
>CONT ⏎      1          ABC
>CONT ⏎      2          ABC
>CONT ⏎      3          ABC
>CONT ⏎      4          ABC
>CONT ⏎      5          ABC
>CONT ⏎
>CONT ⏎
READY
```

Betätigt man jedoch vor dem Start die Tasten EDIT / ENTER , so sieht die Bildschirmausgabe anders aus:

```
>EDIT⏎
             1          ABC
             2          ABC
             3          ABC
             4          ABC
             5          ABC
READY
```

(2) Ablauftest

Unter "Ablauftest" versteht man eine Art des Tests, bei der das Programm kontinuierlich ausgeführt und die gerade ausgeführten Zeilen sowie die Zwischenresultate (Wertzuweisungen an Variable) gleichzeitig auf Bildschirm oder Drucker angezeigt werden. Diese Art des Programmtests wird häufig auch mit dem englischen Begriff "Tracing" bezeichnet.
Einzelheiten siehe TR-Anweisung auf Seite 100.

a) Aktivieren des Ablauftests.
① Eingeben: TR ON ENTER
② Taste RUN drücken.

Das äußere Erscheinungsbild des Ablauftests soll am selben Programmbeispiel wie beim Einzelschrittest demonstriert werden. Auf dem Bildschirm erscheint, einschließlich der Befehle, folgende Ausgabe:

```
>TR ON
>RUN
     10 INPUT "RADIUS =";R
RADIUS =10
 10


     20 IF R<=0 GO TO 80
     30 P=2*PI*R
 62.8318530716
     40 DISP "UMFANG =";P
UMFANG = 62.8318530716
     50 S=PI*R^2
 314.159265358
     60 DISP "FLÄCHE =";S
FLÄCHE = 314.159265358
     70 GO TO 10
     10 INPUT "RADIUS =";R
RADIUS =-1
-1

     20 IF R<=0 GO TO 80
     80 DISP "ENDE"
ENDE
     90 END
READY
>
```

b) Abschalten des Ablauftests

TR OFF [ENTER]

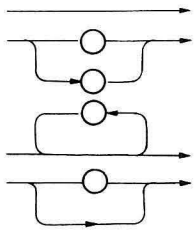
ANMERKUNG:

Der Ablauftest-Modus bleibt bis zum Abschalten mit der TR OFF-Anweisung aktiv. Hat man z.B. ein Programm auf diese Weise getestet und gibt nun ein weiteres Programm ein bzw. lädt es von der Diskette und führt es aus, so unterliegt auch das neue Programm der Überwachung durch den Ablauftest.

III. BASIC-SCHLÜSSELWORTE

1. DEFINITION DER SYNTAX-SYMBOLE

a) Zur Vereinfachung der syntaktischen Definitionen der BASIC-Befehle und -Anweisungen werden die in der Tabelle aufgeführten Symbole eingeführt. Die Symbole gelten ausschließlich für die unter "FORMAT" angegebenen Regeln, nicht etwa für die Programmbeispiele!

Symbol	Bedeutung
|	Trennzeichen zwischen verschiedenen Auswahlmöglichkeiten.
[ ]	Der in eckige Klammern [ ] eingeschlossenen Teil kann weggelassen werden; es wird lediglich die betroffene Teilfunktion unwirksam.
{ }	Der in geschweifte Klammern { } eingeschlossene Teil kann durch Komma (,) getrennt wiederholt hingeschrieben werden.
《 》	Der in Spitzklammern 《 》 eingeschlossene Teil kann durch Semikolons (;) getrennt wiederholt hingeschrieben werden.
{{ }}	Der in doppelte geschweifte Klammern {{ }} eingeschlossene Teil kann durch Kommata (,) oder Semikolons getrennt wiederholt hingeschrieben werden.
m, n	ganze Zahl, z.B. 10.
A, B	numerische Variable (auch Arrayvariable).
S, T	Stringkonstante, z.B. "GERD".
A$	Stringvariable vom $- oder @-Typ, auch Stringarrayelement.
M, N	Beliebige Variable (numerisch oder String, einfach oder Arrayelement)
X, Y, Z, U, V, W, R	Numerischer Ausdruck.
C, D	String (-ausdruck).
SF	Name einer Funktion oder eines Unterprogramms.
CH, DH	einfacher Buchstabe.
.....	Buchstaben, Ziffern oder Sonderzeichen, z.B. * beliebige
statement	Anweisung (Einschränkungen möglich).
	Fluß der Syntax. Bezeichnet eine Auswahl. Bezeichnet eine Wiederholmöglichkeit. Bezeichnet ein mögliches Weglassen.

b) Zeilenmerken:

Fast alle Programmzeilen dürfen auch mit einer Zeilenmarke (engl. "Label") gekennzeichnet werden, Ausnahmen siehe weiter unten. Eine Zeilenmarke hat die Form einer Stringkonstanten aus bis zu acht Zeichen. Sie muß unmittelbar hinter der Zeilennummer folgen und von weiteren Anweisungen in derselben Zeile wie üblich durch einen Doppelpunkt (:) getrennt werden.

c) Unterprogramm- und Funktionsnamen:

Der Name eines Unterprogramms oder einer (benutzerdefinierten) Funktion ist analog zu den Regeln für Variablennamen (siehe Seite 11) zu bilden. Er besteht aus bis zu acht Zeichen, deren erstes ein Buchstabe sein muß. Die folgenden Zeichen müssen alphanumerisch sein; bei Funktionen kann das letzte Zeichen auch eines der Zeichen %, # oder $ sein. Die Typdefinitionen für Variablen (siehe Seite 13) gelten sinngemäß auch für Funktiionen, während Unterprogramme keinen Typ besitzen.

WICHTIGER HINWEIS

1. Bei den syntaktischen Beschreibungen der Befehle wurde [ENTER] weggelassen. Bei der Anwendung der Befehle muß aber die [ENTER] -Taste zur Ausführung betätigt werden!
2. Einer Programmzeile kann eine Zeilenmarke zugeordnet werden. Bei folgenden Anweisungen ist jedoch Obacht geboten:
   SEP# und RFOMAT#
   Zeilen mit SEP#- oder RFORMAT#-Anweisungen dürfen keine Zeilenmarken enthalten. Diese beiden Anweisungen müssen unmittelbar hinter der Zeilennummer stehen. Hinter der RFORMAT#-Anweisung dürfen außerdem keine weiteren (mit: getrennten) Anweisungen in derselben Zeile folgen!
   DATA, SUB und DEFFN
   Vor diesen Anweisungen darf keine Zeilenmarke stehen!

2. BEFEHLE

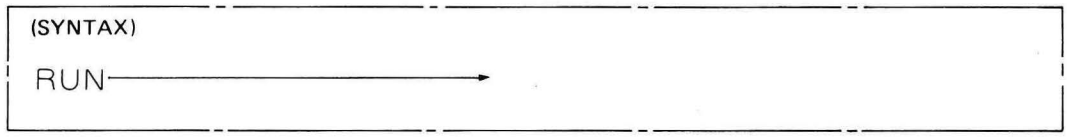
RUN	
VRUN	
CONT	
LIST	※
EDIT	
NEW	※
DELETE	
ERASE	※
AUTO	
REN	
TR	※
TR STOP	※

In diesem Kapitel sind die oben genannten Befehle beschrieben.
Die mit einem Stern ※ markierten Befehle können auch als Anweisungen innerhalb von Programmen verwendet werden.

RUN

Ausführen (Starten) eines BASIC-Programms

● **FUNKTION 1:** Normaler Start eines Programms.
FORMAT: RUN



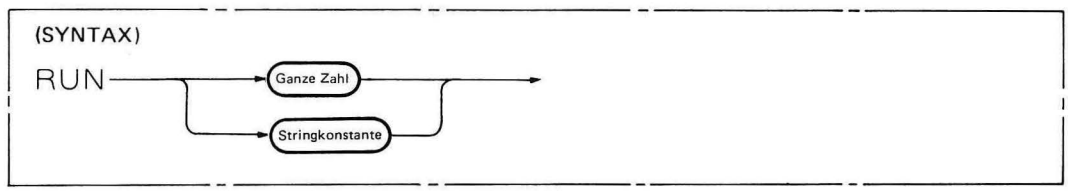
ERLÄUTERUNG:

1. Das im Speicher befindliche Programm wird ab der Zeile mit der kleinsten Zeilennummer ausgeführt.
2. Der Befehl läßt sich auch direkt mit der [RUN] -Taste eingeben.

● **FUNKTION 2:** Ausführen eines Programms ab einer bestimmten Zeilennummer oder Zeilenmarke.

FORMAT: RUN n | S

n: Zeilennummer
S: Zeilenmarke



ERLÄUTERUNG:

1. Das Programm wird beginnend ab einer bestimmten Zeilennummer oder Zeilenmarke ausgeführt.
2. Existiert die im Befehl angegebene Zeilennummer oder -marke nicht, erfolgt eine Fehlermeldung (ERROR 4). Dies gilt auch für Funktion 4.

EINFACHES BEISPIELE:

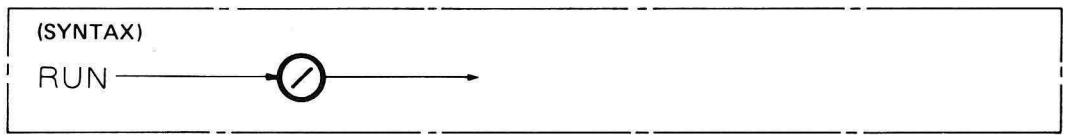
```
200 "A-1" :INPUT A$
300 PRINT A$
```

Um dieses Programm ab Zeile 200 auszuführen, gebe man ein:

```
RUN 200 [ENTER] ..... Vorgabe einer Zeilennummer
```
oder
```
RUN "A-1" [ENTER] ..... Vorgabe einer Zeilenmarke
```

● **FUNKTION 3:** Einstellen des internen Programmzeilen-Zeigers auf die Zeile mit der kleinsten Zeilennummer für einen nachfolgenden CONT-Befehl.

FORMAT: RUN /



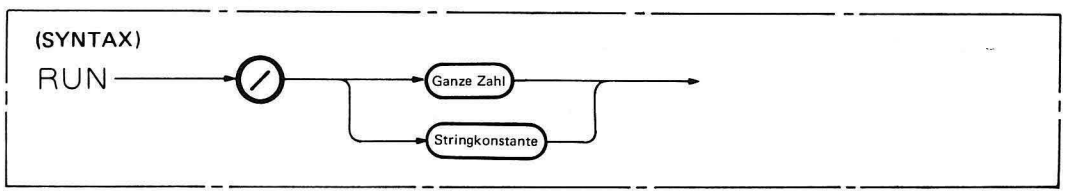
ERLÄUTERUNG:

1. Der interne Programmzeilenzeiger wird auf die Zeile mit der kleinsten Zeilennummer eingestellt.
2. Das Programm wird nach Eingabe dieses Befehls durch Drücken der DEB -Taste zeilenweise ausgeführt. (Die DEB -Taste erzeugt einen CONT/-Befehl.) Diese Form des RUN/-Befehls dient zur Fehlersuche mittels der Testhilfen, siehe auch Seite 71.
3. Antippen der DEB -Taste bei gedrückter CTRL -Taste bewirkt ebenfalls einen RUN/-Befehl.

● **FUNKTION 4:** Einstellen des internen Programmzeilenzeigers auf eine bestimmte Zeilennummer oder Zeilenmarke.

FORMAT: RUN / n | S

n: Zeilennummer
S: Zeilenmarke



ERLÄUTERUNG:

1. Der interne Programmzeilenzeiger wird auf die vorgegebene Zeilennummer oder Zeilenmarke eingestellt.
2. Diese Form des Befehls findet Anwendung, wenn die Fehlersuche an einer bestimmten Stelle im Programm beginnen soll. Einzelheiten siehe Seite 71.

WICHTIGER HINWEIS

Die Ausführung des RUN-Befehls bewirkt eine interne Initialisierung des Interpreters. Unter anderem werden vor dem eigentlichen Start alle Variablen gelöscht, alle Typ- und Funktionsdefinitionen unwirskam gemacht und evtl. offene Dateien geschlossen! Sollen die Variablen erhalten bleiben, so wende man den VRUN-Befehl (siehe Seite 80) an.

VRUN

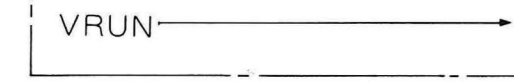
Ausführen (Starten) eines BASIC-Programms ohne Löschen der Variablen.

〈VERWENDUNG〉
Übernahme der Variableninhalte (Zwischenergebnisse) von einem Programm in ein danach ausgeführtes.

● **FUNKTION 1:** Normaler Start eines Programms, jedoch mit Variablenübernahme.

FORMAT: VRUN

(SYNTAX)



ERLÄUTERUNG:

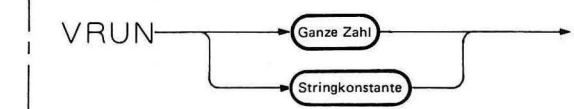
1. Der Befehl bewirkt die Ausführung eines Programms ab der Zeile mit der kleinsten Zeilennummer.
2. Der Befehl läßt sich in Kurzform mit folgenden Tasten eingeben: V RUN

● **FUNKTION 2:** Ausführen eines Programms ab einer bestimmten Zeilennummer oder Zeilenmarke.

FORMAT: VRUN n | S

n: Zeilennummer
S: Zeilenmarke

(SYNTAX)



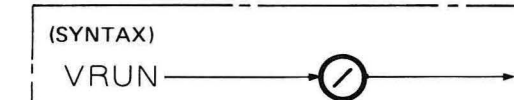
ERLÄUTERUNG:

1. Start des Programms beginnend ab einer bestimmten Zeilennummer oder Zeilenmarke.
2. Existiert die im Befehl angegebene Zeilennummer oder -marke nicht, erfolgt eine Fehlermeldung (ERROR 4). Dies gilt auch für Funktion 4.

● **FUNKTION 3:** Einstellen des internen Programmzeilenzeigers auf die Zeile mit der kleinsten Zeilennummer für einen nachfolgenden CONT-Befehl.

FORMAT: VRUN /

(SYNTAX)



ERLÄUTERUNG:

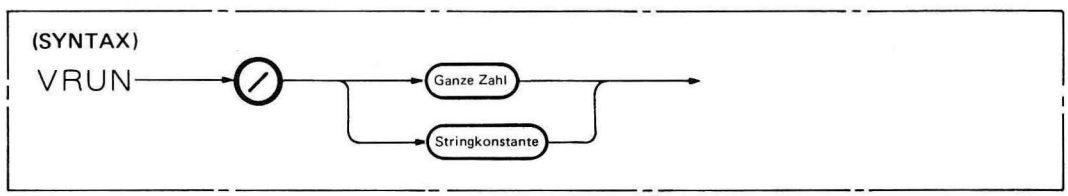
1. Einstellen des internen Programmzeilenzeigers auf die Zeile mit der kleinsten Zeilennummer.
2. Das Programm wird nach Eingabe dieses Befehls durch Drücken der DEB -Taste zeilenweise ausgeführt. (Die DEB -Taste erzeugt einen CONT/-Befehl.) Diese Form des VRUN/-Befehls dient zur Fehlersuche mittels der Testhilfen, siehe auch Seite 71.
3. Antippen der Taste [V] , alsdann der DEB -Taste bei gedrücker [CTRL] -Taste, sendet ebenfalls einen VRUN/-Befehl an den Rechner.

● **FUNKTION 4:** Einstellen des internen Programmzeilenzeigers auf eine bestimmte Zeilennummer oder Zeilenmarke.

FORMAT: VRUN / n | S

n: Zeilennummer

S: Zeilenmarke



ERLÄUTERUNG:

1. Stellt den internen Programmzeilenzeiger auf eine bestimmte Zeilennummer oder Zeilenmarke ein.
2. Diese Form des Befehls findet Anwendung, wenn die Fehlersuche an einer bestimmten Stelle im Programm beginnen soll. Einzelheiten siehe Seite 71.

WICHTIGER HINWEIS

Im Gegensatz zum RUN -Befehl (siehe Seite 78) werden die Variablen nicht gelöscht.

CONT

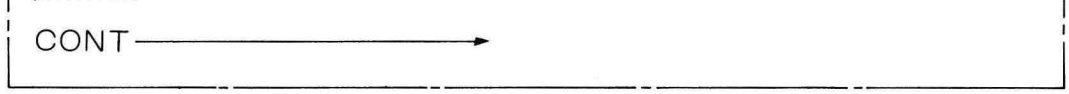
Fortsetzung eines unterbrochenen Programms. (Continue)

〈VERWENDUNG〉
① Fortsetzung eines Programms, das durch Drücken der BRK- oder HALT-Taste oder die Ausführung einer STOP- oder TR STOP-Anweisung angehalten wurde.
② Ausführung eines Programms nach einem RUN/- oder VRUN/-Befehl.

● **FUNKTION 1:** Fortsetzung der Programms ab der Zeile nach derjenigen, in der das Programm angehalten wurde.

FORMAT: CONT

(SYNTAX)



ERLÄUTERUNG:

1. Die Ausführung des Programms wird mit der Zeile fortgesetzt, die hinter der Zeile steht, in der der Programmstopp erfolgte.
2. Wurde das Programm jedoch durch Betätigen der HALT-Taste unterbrochen, so wird die Ausführung am Anfang der Zeile aufgenommen, die beim Drücken der HALT-Taste ausgeführt wurde.
3. Der Befehl kann über die CONT -Taste auch direkt eingegeben werden.

EINFACHES BEISPIELE:

```
400 STOP
450 INPUT A$
```

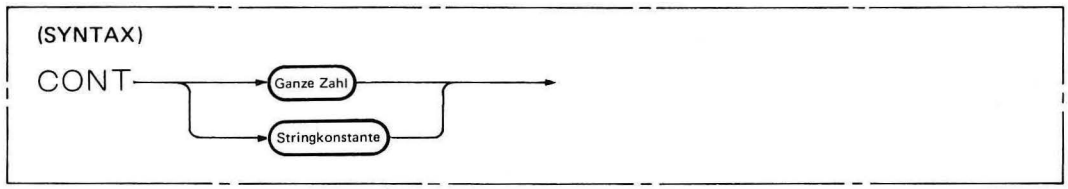
Nachdem das Programm auf die STOP-Anweisung in Zeile 400 lief, wird durch den CONT-Befehl die Ausführung beginnend ab Zeile 450 wieder aufgenommen.

● **FUNKTION 2:** Fortsetzung des Programms ab einer bestimmten Zeilennummer oder Zeilenmarke.

FORMAT: CONT n | S

n: Zeilennummer

S: Zeilenmarke

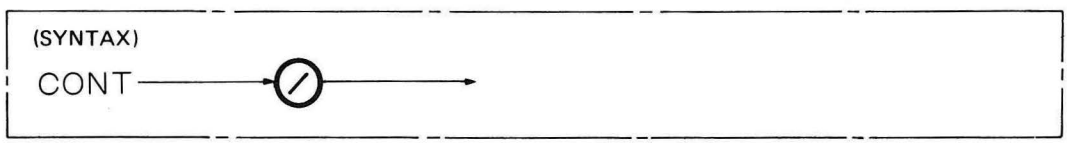


ERLÄUTERUNG:

1. Die Ausführung des Programms wird in der Zeile mit der vorgegebenen Zeilennummer oder Zeilenmarke wieder aufgenommen.
2. Existiert die Zeilennummer oder Zeilenmarke nicht in Programm, erfolgt eine Fehlermeldung (ERROR 4).

● **FUNKTION 3:** Ausführen nur einer einzigen Zeile hinter der Anweisung, bei deren Ausführung der Programmstopp erfolgte.

FORMAT: CONT /



ERLÄUTERUNG:

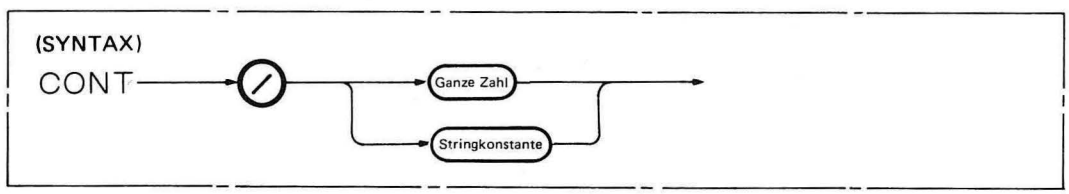
1. Es wird nur eine einzige Programmzeile nach der Stopzeile ausgeführt.
2. Nach dem Betätigen der HALT-Taste wird nur die Zeile ausgeführt, die zum Zeitpunkt des Tastendrucks bearbeitet wurde.
3. Diese Form des Befehls findet bei der Fehlersuche Anwendung. Einzelheiten siehe II.3.(1) "Einzelschrittest" auf Seite 71.
4. Mit der DEB -Taste kann der Befehl auch direkt eingegeben werden.

● **FUNKTION 4:** Ausführen einer einzigen Programmzeile mit einer bestimmten Zeilennummer oder Zeilenmarke.

FORMAT: CONT / n | S

n: Zeilennummer

S: Zeilenmarke



ERLÄUTERUNG:

1. Es wird nur die Zeile mit der angegebenen Zeilennummer oder Zeilenmarke ausgeführt.
2. Existiert die Zeilennummer oder Zeilenmarke nicht in Programm, erfolgt eine Fehlermeldung (ERROR 4).

LIST

Auflisten des gegenwärtig im Speicher befindlichen Programms.

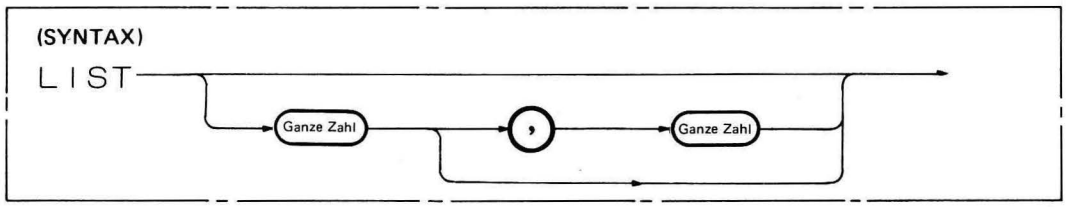
〈VERWENDUNG〉
① Auflisten eines Programms auf dem Bildschirm.
② Auflisten eines Programms auf dem Drucker.
③ Auflisten der aktuellen Belegung der Funktionstasten.
④ Prüfen, ob sich ein Programm im Speicher befindet.

● **FUNKTION 1:** Ausgabe einer Programmliste auf dem Bildschirm.

FORMAT: L I S T 〔m 〔, n〕〕

m : erste Zeilennummer

n : letzte Zeilennummer (n > m)



ERLÄUTERUNG:

1. Von dem in Speicher befindlichen Programm wird der Bereich zwischen den Zeilennummern m und n (je einschließlich) auf dem Bildschirm ausgegeben.
   Soll nur die eine Zeile mit der Nummer n aufgelistet werden, so muß diese zweimal in der Form LIST n,n angegeben werden.
2. Läßt man die Angabe ,n aus, so wird ab Zeile m der ganze Rest des Programms aufgelistet.
3. Werden beide Zeilennummern m und n weggelassen, so wird das gesamte Programm aufgelistet.
4. Wenn das Programmlisting größer als eine Bildschirmseite ist, läuft es durch ("Scrolling"). Man kann das Durchlaufen mit der Leertaste anhalten und wieder freigeben. Siehe auch Seite 320, CAT-Anweisung.
5. Existieren keine Zeilen mit den Nummern m und/oder n im Programm, so wird trotzdem der zwischen m und n liegende Bereich gelistet. Ist m > n, so erfolgt keine Fehlermeldung, es wird aber auch nichts ausgegeben.

EINFACHES BEISPIELE:

An der Tastatur werde folgendes Programm eingegeben, was hier in der Originalform der Eingabe wiedergegeben wird:

```
100A=1
110"A":B=5*A+13
120DISP"A=";A,"B=";B
130A=A+.5
140IFA>10THEN"Z"
150GOTO"A"
160"Z":END
```

Um das gesamte Programm auf dem Bildschirm zu listen, lautet die Eingabe:
LIST ENTER
Das Listing sieht dann wie folgt aus:

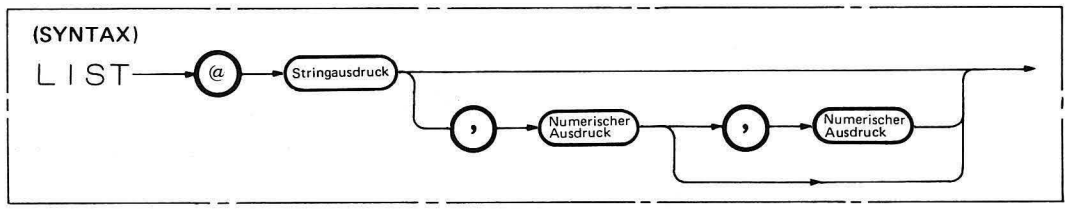
```
100 A=1
110*"A":B=5*A+13
120 DISP "A=";A,"B=";B
130 A=A+.5
140 IF A>10 THEN "Z"
150 GO TO "A"
160*"Z":END
```

Zeilen mit einer Zeilenmarke werden durch Anhängen eines Sterns (*) an die Zeilennummer hervorgehoben, in diesem Beispiel sind das die Zeilen 110 und 160.

● **FUNKTION 2:** Ausgabe eines passwortgeschützten Programms auf dem Bildschirm.

FORMAT: L I S T @C 〔, m 〔, n〕〕

@ : spezifiziert ein geschütztes Programm
C : Passwort
m : erste Zeilennummer
n : letzte Zeilennummer



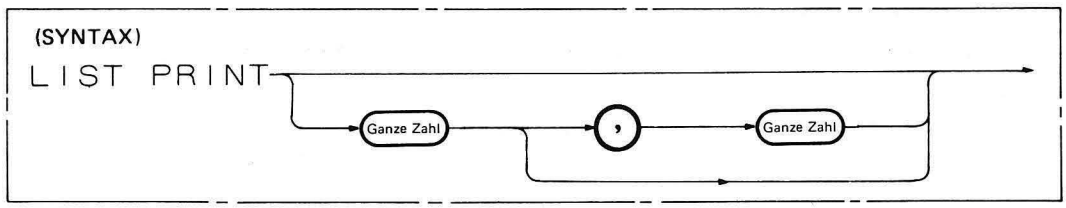
ERLÄUTERUNG:

1. Von dem im Speicher befindlichen, mit dem Passwort C geschützten Programm wird der Bereich zwischen den Zeilennummern m und n (je einschließlich) auf dem Bildschirm ausgegeben.
2. Alle weiteren Einzelheiten siehe Funktion 1.

● **FUNKTION 3:** Ausgabe einer Programmliste auf dem Drucker.

FORMAT: L I S T P R I N T 〔m 〔, n〕〕

m : erste Zeilennummer
n : letzte Zeilennummer



ERLÄUTERUNG:

1. Von dem im Speicher befindlichen Programm wird der Bereich zwischen den Zeilennummern m und n (je einschließlich) mit maximal 80 Zeichen pro Zeile auf dem Drucker ausgegeben.
2. Alle weiteren Einzelheiten siehe Funktion 1.

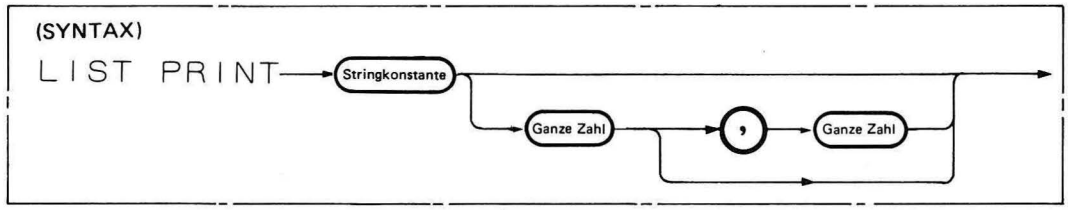
● **FUNKTION 4:** Ausgabe einer Programmliste auf dem Drucker mit Title, Seiteninteilung, Datum usw.

FORMAT: L I ST PR I NT S 〔, m 〔, n〕〕

S : Name für die Titelzeile

m : erste Zeilennummer

n : letzte Zeilennummer



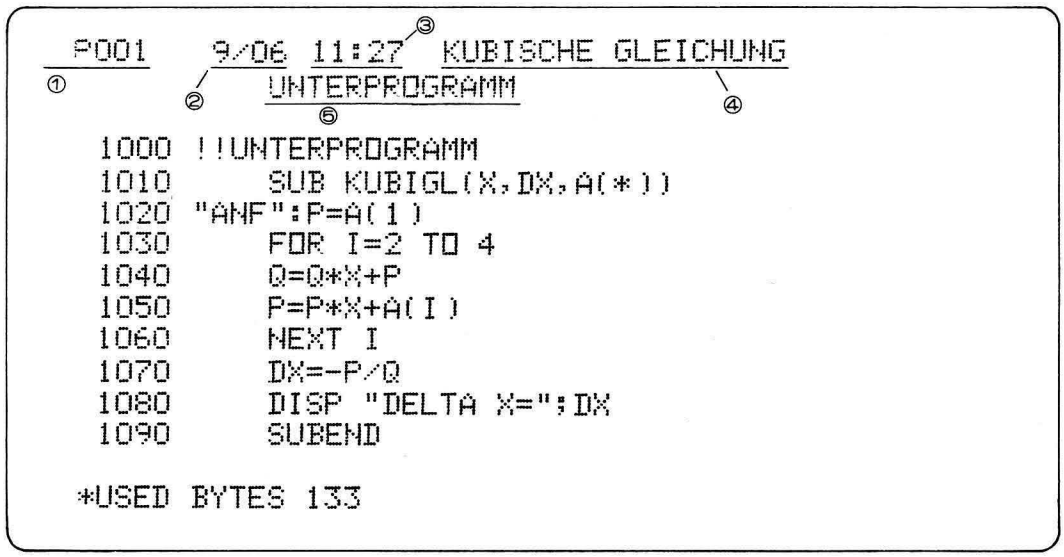
ERLÄUTERUNG:

1. Von dem im Speicher befindlichen Programm wird der Bereich zwischen den Zeilennummern m und n (je einschließlich) mit maximal 80 Zeichen pro Zeile auf dem Drucker ausgegeben. Jede Seite beginnt mit einer Titelzeile, die Seitennummer, Datum, Uhrzeit und den mit S vorgegebenen Namen enthält.
2. Weitere Einzelheiten siehe Funktion 1.
3. Eine Seite enthält einschließlich Titelzeile bis zu 59 Druckzeilen und 7 oder mehr Leerzeilen bis zum Anfang der nächsten Seite. Vor dem Drucken sollte man das Papier so einlegen, daß die Perforation in den Bereich der Leerzeilen zu liegen kommt. Mit diesem Befehl lassen sich Programme untergliedern:
4. Steht hinter der Zeilennummer ein !!, so wird ein Seitenvorschub eingefügt und ab dieser Zeile neu gedruckt, wobei der hinter !! stehende Text als Untertitel benutzt wird. Der Untertitel bleibt bis zum nächsten !! erhalten.
5. Untertitel können maximal 20 Zeichen lang sein, überschüssige Zeichen werden rechts abgeschnitten.
6. Am Ende des Listings wird die Länge des Programms in Bytes hinzugefügt (der Wert der STATUS 1-Funktion).
7. Drückt man während des Ausgabevorganges die HALT-Taste, so wird das Listing abgebrochen und die Länge des Programms ausgegeben. Danach erfolgt ein Seitenvorschub.

EINFACHES BEISPIELE:

```
1000 !!UNTERPROGRAMM
1010 SUB KUBIGL(X,DX,A(*))
1020 "ANF":P=A(1)
1030 FOR I=2 TO 4
1040 Q=Q*X+P
1050 P=P*X+A(I)
1060 NEXT I
1070 DX=-P/Q
1080 DISP "DELTA X=";DX
1090 SUBEND
```

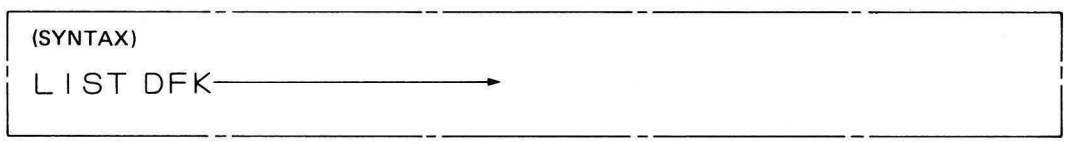
Gibt man für umseitiges Beispielprogramm
LIST PRINT"CUBIC EQUATION",1000 [ENTER]
ein, erhält man folgendes Druckbild:



① Seitennummer
② Datum, hier 6. September
③ Uhrzeit, hier 11 Uhr 27
④ Titel aus dem LIST PRINT-Befehl
⑤ Untertitel aus Zeile 1000

!! ist eine besondere Art von Kommentar (siehe auch REM-Anweisung, Seite 98) und hat keinerlei Einfluß auf den Ablauf des Programms.

● **FUNKTION 5:** Belegung der Funktionstasten anzeigen.
FORMAT: L I S T DFK



ERLÄUTERUNG:

1. Zeigt die derzeitige Belegung der frei definierbaren Funktiontasten auf dem Bildschirm an. Einzelheiten siehe Seite 58.
2. Für die Ausgabe werden zwei Spalten angelegt. Die Belegung von DEF 1 bis DEF 10 erscheint in der linken Spalte, die von DEF 11 bis DEF 20 in der rechten.
3. Ist einer Funktionstaste nichts zugeordnet, so wird sie ausgelassen. Im folgenden Beispiel sind das DEF 12 sowie DEF 15 bis DEF 20.
4. Wenn zum Text einer Taste der der folgenden hinzugefügt wird, erscheint das Symbol [↓] hinter der Definition.

BEISPIEL:

```
LIST DFK
   1 +                    11 RUN@
   2 -                    12 LIST
   3 *                    13 DISP
   4 /                    14 INPUT
   5 (
   6 )
   7 =
   8 "
   9 $
  10 ,
READY
```

EDIT

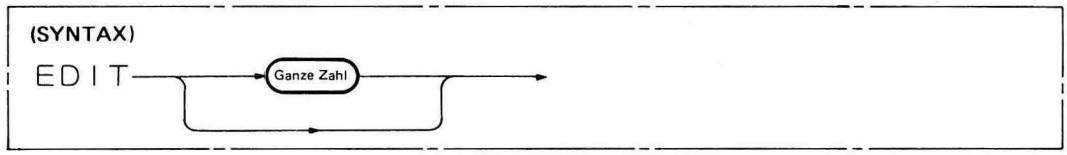
Aufruf einer Programmzeile zum Editieren.

〈VERWENDUNG〉
Bei der Korrektur eines Programms.

● **FUNKTION 1:** Aufruf einer Zeile über ihre Zeilennummer

FORMAT: E D I T 〔X〕

X: Zeilennummer



ERLÄUTERUNG:

1. Die Zeile mit der Nummer X (Dezimalen werden abgeschnitten) wird auf dem Bildschirm angezeigt und zur Änderung freigegeben.
2. Hat man eine Zeile mit der Nummer X editiert und führt dann den Befehl ohne Angabe einer Zeilennummer aus, so wird die der Zeile X unmittelbar folgende zum Editieren aufgerufen. Beim Editieren aufeinanderfolgender Zeilen braucht man also nur einmal eine Zeilennummer einzugeben.
3. Existiert keine Zeile mit der angegebenen Nummer im Programm, so wird die Zeile mit der nächstfolgenden Nummer aufgerufen.
4. Der Befehl läßt sich direkt mit der [EDIT] -Taste eingeben.

EINFACHES BEISPIELE:

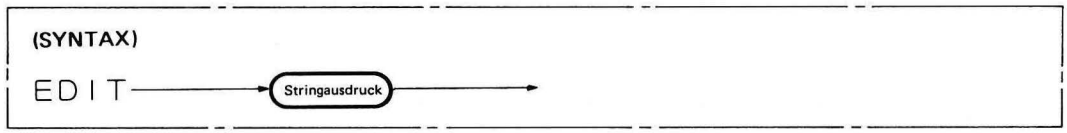
Ein Programm beginne mit der Zeilennummer 100, die Zeilen seien in Schritten von 100 numeriert. Zur Korrektur der Zeilen 300 und 400 kann man dann wie folgt verfahren:

① Betriebsartschalter in Stellung PRO bringen.
② [EDIT] 300 [ENTER] ····· Aufruf von Zeile 300.
③ Zeile 300 editieren, dann [ENTER] drücken.
④ [EDIT] [ENTER] ····· Aufruf der nächsten Zeile, hier also 400.
⑤ Zeile 400 editieren, mit [ENTER] abschließen.

● **FUNKTION 2:** Aufruf der Zeile von der zuletzt bearbeiteten.

FORMAT: E D I T CH

CH: Buchstabe



ERLÄUTERUNG:

1. Läßt man bei Funktion 1 die Zeilennummer aus, so wird die Folgezeile aufgerufen. In Funktion 2 wird statt der nachfolgenden die unmittelbar vorausgehende Zeile zum Editieren aufgerufen.
2. Für CH kann jeder beliebige Buchstabe eingesetzt werden.

EINFACHES BEISPIELE:

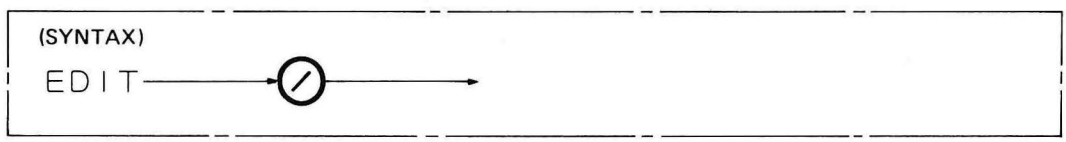
Ein Programm beginne mit der Zeilennummer 100, die Zeilen seien in Schritten von 100 numeriert. Stellt man während der Bearbeitung von Zeile 400 fest, daß auch in Zeile 300 eine Korrektur erforderlich ist, kann man wie folgt verfahren:
(Die Korrektur von Zeile 400 sei gerade abgeschlossen.)

① EDIT D ENTER ····· Aufruf der Zeile von 400, hier also 300.
(Statt D hätte auch jeder andere Buchstabe dieselbe Funktion erfüllt.)

② Zeile 300 editieren, mit ENTER abschließen.

● **FUNKTION 3:** Umleiten der Korrektur- und Testhilfen-Protokolle auf die untere Bildschirmzeile.

FORMAT: E D I T /



ERLÄUTERUNG:

1. Beim Einzelschrittest (Einzelheiten siehe Seite 71) werden von Hand eingegebene Befehle wie CONT/ normalerweise unter die anderen Bildschirmausgaben gemischt. Diese Funktion des EDIT-Befehls hilft das zu vermeiden.
2. Dieser BEfehl schützt die oberen 15 Bildschirmzeilen vor dem Überschreiben durch manuelle Eingaben.
3. Vor der Eingabe des Befehls muß die Betriebsart auf PRO geschaltet werden. Danach die EDIT -Taste und "/" drücken.
Abschalten des Einzelschrittests mit EDIT ENTER CL

WICHTIGER HINWEIS

1. Vor dem Editieren muß der Betriebsartschalter in Stellung PRO gebracht werden!
2. Einzelheiten zum Vorgehen bei der Korrektur von Programmen siehe II.2. "Korrigieren und Editieren" auf Seite 66.

NEW

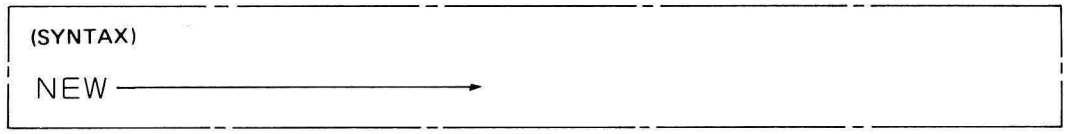
Löschen des gesamten Programms und aller Variablen.

〈VERWENDUNG〉

Bereinigung des Speicherinhalts vor der Eingabe eines neuen Programms.

FORMAT: NEW

(SYNTAX)



ERLÄUTERUNG:

1. Alle im Speicher vorhandenen Programmzeilen und alle Variablen werden gelöscht, alle Definitionen unwirksam gemacht.
   Alle noch offenen Dateien werden geschlossen.
   Die Belegung der Funktionstasten wird davon jedoch nicht berührt.
2. Wird dieser Befehl als Anweisung in ein Programm aufgenommen, so kann sich das Programm (nach getaner Arbeit) selbst löschen.

DELETE

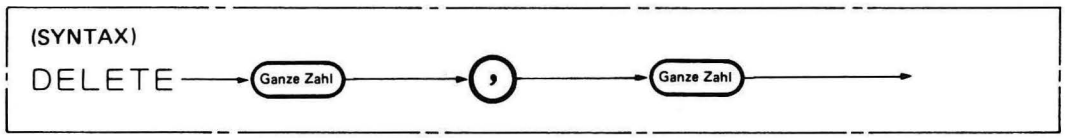
Löschen von Programmzeilen.

〈VERWENDUNG〉
① Löschen von Programmabschnitten.
② Löschen der Belegung von Funktionstasten vor der Definition eines neuen Textes.

● **FUNKTION 1:** Löschen von Programmzeilen.

FORMAT: DELETE m, n

m : Erste Zeilennummer des zu löschenden Bereichs.
n : Letzte Zeilennummer des zu löschenden Bereichs.



ERLÄUTERUNG:

1. Alle Zeilen zwischen m und n je einschließlich werden gelöscht.
2. Wenn nur eine einzige Zeile n gelöscht werden soll, so ist ihre Nummer zweimal in der Form DELETE n,n anzugeben.
   (Einfacher ist in diesem Fall, nur die Zeilennummer und danach sofort [ENTER] ohne den DELETE-Befehl einzugeben.)
2. Auch wenn keine Zeile mit Nummer m und/oder n exisiert, wird auf jeden Fall der zweimal in der Form DELETE n,n anzugeben.
   (Einfacher ist in diesem Fall, nur die Zeilennummer und danach sofort [ENTER] ohne den DELETE-Befehl einzugeben.)
3. Auch wenn kein Zeile mit Nummer m und/oder n existiert, wird auf jeden Fall der zwischen m und n liegende Bereich gelöscht.
4. Ist m > n oder gibt es im angegebenen Bereich keine Zeilen, erfolgt eine Fehlermeldung (ERROR 4).

EINFACHES BEISPIELE:

1. DELETE 100, 200 [ENTER] ..... löscht alle Zeilen von 100 bis 200.

2. DELETE 100, 100 [ENTER]
   oder
   100 [ENTER]
   ..... löscht die Zeile mit der Nummer 100.

WICHTIGER HINWEIS

Löschen einer Programmzeile ist nur wirksam, wenn der Betriebsartschalter auf PRO gestellt ist.

In Stellung OP erfolgt eine Fehlermeldung (ERROR 34).

● **FUNKTION 2:** Löschen der Belegungen frei definierbarer Funktionstasten.
FORMAT: DELETE DFK

(SYNTAX)

DELETE DFK⟶

ERLÄUTERUNG:

1. Bei allen Funktionstasten DEF 1 bis DEF 20 wird der zugeordnete Text gelöscht.
2. Nach der Ausführung des Befehls sind die Funktionstasten unwirksam und müssen bei Bedarf wieder neu belegt werden. Einzelheiten siehe I.7. "Frei definierbare Funktionstasten" auf Seite 58.

ERASE

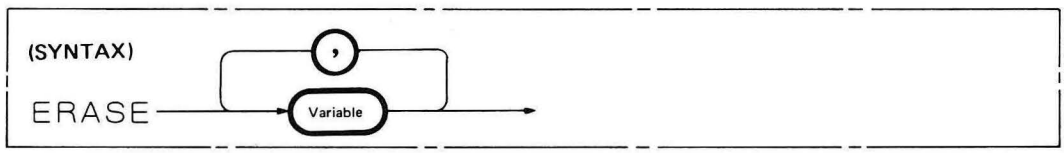
Löschen von Variablen

〈VERWENDUNG〉
Effiziente Ausnutzung des Speichers durch Löschen nicht mehr benötigter Variablen.

● **FUNKTION 1:** Löschen angegebener Variablen

FORMAT: ERASE 〔N〕

N : Name der zu löschenden Variablen.



ERLÄUTERUNG:

Die Variable N (mehrere Namen möglich) wird gelöscht.
Damit wird der von ihr belegte Speicher für andere Zwecke frei, was besonders bei Arrays sehr effizient ist.

EINFACHES BEISPIELE:

Ein mit der Anweisung DIM A(20,20) deklariertes Array wird nicht mehr benötigt. Mit dem Befehl

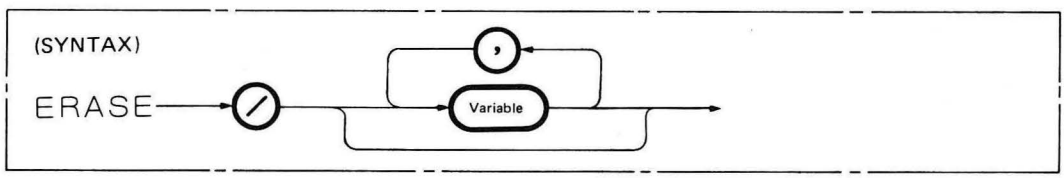
ERASE A(*) ENTER

wird es gelöscht. Man beachte die Schreibweise für den Index, da in diesem Befehl auf das ganze Array Bezug genommen wird, siehe auch Seite 21.

● **FUNKTION 2:** Löschen von Variablen bis auf die angegebenen.

FORMAT: ERASE / 〔〔N〕〕

N : Name der Variablen, die nicht gelöscht werden soll.



ERLÄUTERUNG:

1. Alle Variablen, außer den angegebenen, werden gelöscht.
2. Fehlt die Angabe von N, werden alle Variablen gelöscht.

WICHTIGER HINWEIS

1. "Variable" bedeutet in diesem Zusammenhang nicht nur den Wert (Inhalt) der Variablen, sondern bezieht sich auf die gesamte Variable. Beim Löschen eines Arrays wird also auch die DIM-Anweisung aufgehoben.
2. Der Befehl kann auch als Anweisung innerhalb eines Programms benutzt werden, darf jedoch nicht innerhalb eines Unterprogramms (siehe Seite 20) oder einer FOR/NEXT-Schleife auftreten.

AUTO

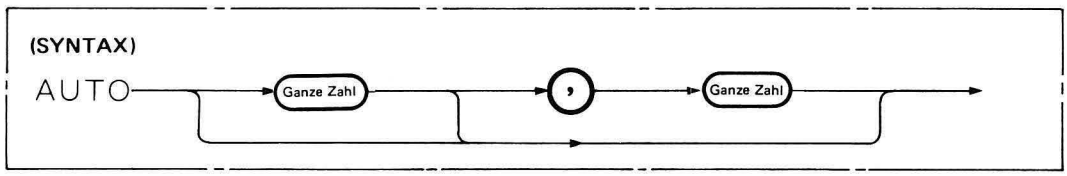
Automatisches Erzeugen von Zeilennummern.

〈VERWENDUNG〉
Erspart bei der Eingabe von Programmen die manuelle Eingabe der Zeilennummern.

FORMAT: AUTO 〔m 〔, n〕〕

m : Anfangs-Zeilennummer

n : Inkrement oder Abstand der Zeilennummern



ERLÄUTERUNG:

1. Die Zeilennummer m wird angezeigt, und man kann wie gewohnt den Inhalt der Zeile eingeben und mit [ENTER] abschließen. Die Zeile wird gespeichert, das Inkrement n zur Zeilennummer addiert und die neue Zeilennummer erscheint auf dem Bildschirm. Der Rechner ist bereit zur Annahme der nächsten Zeile.
   Bei jeder Betätigung der [ENTER] -Taste wird automatisch eine neue Zeilennummer erzeugt, sodaß der Programmierer von der manuellen Eingabe der Zeilennummern befreit wird.
2. Läßt man das Inkrement bei der Eingabe des Befehls weg, so wird automatisch 10 eingesetzt.
3. Will man nach der Eingabe der letzten Zeile die AUTO-Funktion wieder abschalten, verfahre man wie folgt:
   [CL] -Taste drücken oder Zeilennummer mit der [←] -Taste löschen.
   AUTO [ENTER] eingeben oder [CTRL] [HOME] betätigen.
   ([CTRL] [HOME] bewirkt gleichzeitig Löschen des Bildschirms.)

EINFACHES BEISPIELE:

1.

Befehl		Erzeugte Zeilennummern
AUTO 10	[ENTER]	10,20,30 .....
AUTO 5,5	[ENTER]	5,10,15 .....
AUTO 100	[ENTER]	100,110,120 ...
AUTO 100,20	[ENTER]	100,120,140 ...

2. AUTO 20,20 [ENTER]
   Nach Eingabe des Befehls wird die Zeilennummer 20 angezeigt und der Kursor springt auf die Position hinter der Zeilennummer.
   Nach der Eingabe der Zeile und dem Drücken der [ENTER] -Taste erscheint die nächste Zeilennummer 40, usw. Wenn man auf diese Weise das Programmende erreicht hat, die letzte Zeile des Programms sei z.B. 1000, erscheint die nächste Zeilennummer 1020. Der Prozeß wird dann durch Drücken der [CL] - oder der [←] - Taste und die Eingabe von AUTO [ENTER] bzw. Betätigen von [CTRL] [HOME] abgebrochen.

WICHTIGER HINWEIS

Macht man Fehler bei der Eingabe von Programmzeilen unter der Kontrolle der automatischen Zeilennumerierung, so kann sich ein Nebeneffekt einstellen, der sich am besten an einem Beispiel verdeutlichen läßt.

Folgendes kleines Programm soll nach dem Befehl AUTO 20,20 [ENTER] eingegeben werden:

```
20 A1=5:B1=10
40 C1=A1*B1*7
60 DISP A1,B1,C1
80 END
```

Nun hat man bei Zeile 40 aber aus Versehen die ersten beiden Zeichen der Anweisung vertauscht. Auf dem Bildschirm ist zu lesen:

```
40 1C=A1*B1*7
```

Die nächste Zeilennummer ist nicht wie erwartet 60, sondern 421. Der Interpreter zieht nämlich alle Ziffern am Anfang der Zeile ohne Rücksicht auf Leersteellen mit der Zeilennummer zusammen, so daß die fehlerhaft eingegebene Zeile als

```
40 1C=A1*B1*7
└┘
└─► aufgefaßt wurde. 401 + 20 = 421.
```

Um den Fehler zu beheben, löscht man die fehlerhafte Zeilennummer 421 mit der [CL] - oder der [←] -Taste. Dann gibt man die Zeile 40 erneut ein, aber diesmal vollständig mitsamt der Zeilennummer 40.

Wenn man es richtig gemacht hat, setzt die Zeilennummernautomatik daraufhin mit 60 fort, und man kann das Programm normal weiter eingeben.

Nicht vergessen: Die noch existierende Zeile 401 mit der Eingabe von 401 [ENTER] löschen!

Wenn man jedoch ein Programm "blind" ohne Kontrolle des Bildschirms eingibt und auf diese Weise schon eine größere Anzahl falsch numerierter Zeilen eingegeben hat, schalte man den AUTO-Befehl ab und kann versuchen, die fehlerhaften Zeilen mit dem REN-Befehl (Seite 98) richtig zu numerieren, bevor man mit dem normalen Eingeben weitermacht.

REN

Änderung von Zeilennummern (Renumber).

〈VERWENDUNG〉

① "Glätten" der Folge der Zeilennummern nach dem Löschen oder Neueinfügen von Programmzeilen.

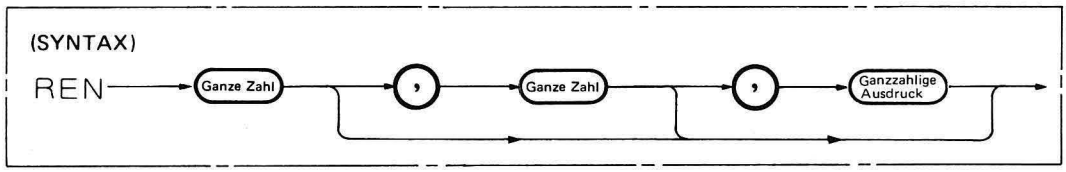
② Schaffen von Lücken zum Einfügen neuer Zeilen, wenn an der vorgesehenen Stelle die Zeilennummern im Abstand von 1 aufeinander folgen.

FORMAT: REN ℓ [, m [, n]]

ℓ : Erste Zeile des Abschnitts, der neu numeriert werden soll.

m : Erste neue Zeilennummer

n : Neues Inkrement



ERLÄUTERUNG:

1. Der mit der Zeile k beginnende Abschnitt des Programms erhält neue Zeilennummern, die mit m beginnen und in Schritten von n hochgezählt werden.
   k muß kleiner oder gleich m sein. Ist das nicht der Fall (k > m) und existieren Zeilen mit Nummern kleiner als m, erfolgt eine Fehlermeldung (ERROR 64).
   Auch wenn k die kleinste vorkommende Zeilennummer ist, muß k < m gelten).
   Existiert keine Zeile mit der Nummer k im Programm, erfolgt eine Fehlermeldung (ERROR 64).
2. Läßt man das Inkrement n weg, wird automatisch 10 als Inkrement angenommen.
3. Läßt man m und n weg, werden alle Zeilen ab der Zeile mit Nummer k in Zehnerschritten, beginnend mit k, neu numeriert.

EINFACHES BEISPIELE:

1.

```
10 A=5:B=10
15 C=A*B+7
18 DISP A,B,C
20 END
```

Nach der Ausführung des Befehls REN 10,100,20 hat das Programm folgende Gestalt:

```
100 A=5:B=10
120 C=A*B+7
140 DISP A,B,C
160 END
```

2.

```
10 A=1
15 DISP A^3,
20 A=A+1
21 IF A=2 THEN (10+10)
23 IF A<=20 THEN 15
25 END
```

Das Programm werde mit dem Befehl REN 10 [ENTER] bearbeitet und zeigt sich danach mit folgenden Zeilennummern:

```
10 A=1
20 DISP A^3,
30 A=A+1
40 IF A=2 THEN (10+10)          ①
50 IF A<=20 THEN 20             ②
60 END
```

① ···· Wurde nicht verändert
② ···· Wurde verändert

WICHTIGER HINWEIS

1. Der REN-Befehl paßt Sprungziele und andere Zeilennummernbezüge von Anweisungen wie GO TO, GO SUB, THEN, ELSE, ON ... GO TO, ON ... GO SUB, RESTORE und USING automatisch an die neuen Zeilennummern an.
   Wird der Bezug jedoch nicht als einfache Zahl, sondern in Form eines numerischen Ausdrucks gegeben, ist keine Anpassung möglich, siehe Beispiel 2.
2. Die Zeilennummern in den Befehlen TR, TR STOP und LIST werden nicht verändert.
3. Der Befehl kann nur in Stellung PRO des Betriebsartschalters ausgeführt werden. Versucht man in Stellung OP eine Neunumerierung, so erfolgt eine Fehlermeldung (ERROR 34).

ANMERKUNG:

Mit einem List-Befehl kann man sich leicht davon überzeugen, ob die Neunumerierung korrekt war. Bei umfangreichen Änderungen sollte man das Programm von Zeit zu Zeit auf Diskette sichern, siehe SAVE-Befehl auf Seite 355.

TR

Ablauftest eines Programms (Trace).

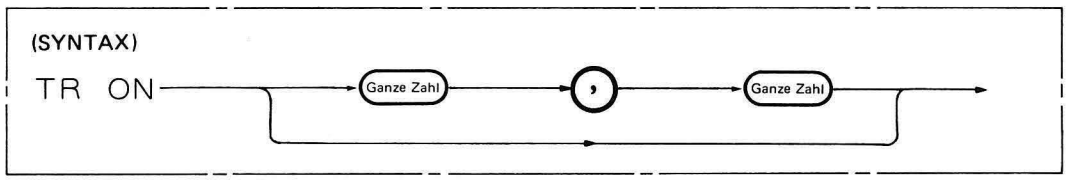
〈VERWENDUNG〉
Prüfen eines Programms an Hand des Bearbeitungsablaufes; Protokollierung der ausgeführten Zeilen und der Zwischenergebnisse.

● **FUNKTION 1:** Einschalten des Ablauftests (Tracings) mit Ausgabe auf dem Bildschirm.

FORMAT: TR ON 〔m, n〕

m : Erste Zeilennummer des zu überwachenden Bereichs

n : Letzte Zeilennummer des zu überwachenden Bereichs



ERLÄUTERUNG:

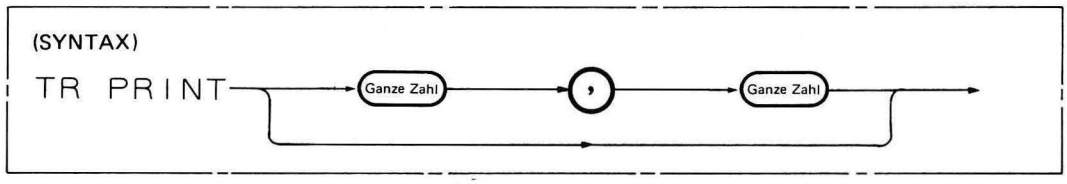
1. Die Tracefunktion des Interpreters wird für den Bereich der Zeilen mit den Nummern m bis n (m < n) je einschließlich aktiviert und für Bildschirmausgabe vorbereitet.
2. Wird nach dem TR-Befehl das Programm mit einem RUN-Befehl (siehe Seite 78) gestartet, so werden im angegebenen Bereich die gerade ausgeführten Zeilen sowie die wertzuweisungen an Variable auf dem Bildschirm protokolliert. Fehlt die Angabe von m und n, so erstreckt sich der Trace-Bereich über das geasmte Programm.
3. Zum Einzelschrittest unter Trace-Überwachung gibt man zunächst den TR-Befehl ein (TR ON m,n) und betätigt dann bei gedrückter [CTRL] -Taste die [DEB] -Taste (dadurch wird ein RUN/-Befehl eingegeben).
   Bei jedem Drücken der [DEB] -Taste (= Eingabe eines CONT/Befehls) wird nun eine Programmzeile im Trace-Modus ausgeführt.
4. Die Trace-Funktion wird auch dann aktiviert, wenn es im Programm keine Zeilen mit den Nummern m oder n gibt, sie wirkt dann auf den dazwischen liegenden Bereich. Hat z.B. ein Programm die Zeilennummern 10 bis 200 in Schritten von 10 und lautet der Trace-Befehl TR ON 25, 75, so werden die Zeilen 30 bis 70 überwacht.
5. Wurde m > n eingegeben, so ist der TR-Befehl ungültig.

● **FUNKTION 2:** Einschalten des Tracings mit Ausgabe auf dem Drucker.

FORMAT: TR PRINT [m, n]

m : Erste Zeilennummer des zu überwachenden Bereichs

n : Letzte Zeilennummer des zu überwachenden Bereichs

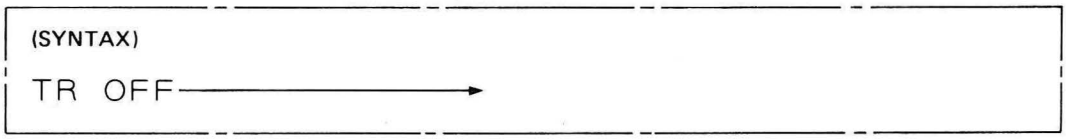


ERLÄUTERUNG:

1. Die Tracefunktion des Interpreters wird für den Bereich der Zeilen mit den Nummern m bis n (m < n) je einschließlich aktiviert und für Druckerausgabe vorbereitet.
2. Alle weiteren Einzelheiten siehe Funktion 1, Trace auf Bildschirm.

● **FUNKTION 3:** Abschalten des Tracings.

FORMAT: TR OFF



ERLÄUTERUNG:

1. Die Tracefunktion des Interpreters wird abgeschaltet.
2. Nach dem Einschalten bleibt die Tracefunktion dauernd aktiv, über beliebig viele Programmläufe hinweg. Spätestens vor der Ausführung des nächsten Programms muß die Tracefunktion mittels dieser Form des Befehls abgeschaltet werden, wenn man sie nicht mehr benötigt.

WICHTIGER HINWEIS

Es ist nicht möglich, das Testprotokoll gleichzeitig auf dem Bildschirm darzustellen und zu drucken. Gültig ist der jeweils letzte TR-Befehl.

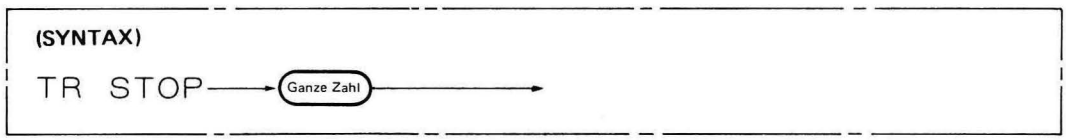
TR STOP

Zwischenstopp des Programms an einer bestimmten Zeilennummer.

〈VERWENDUNG〉
① Prüfen des Inhalts von Variablen während der Berechnung.
② Ändern von Variablen während der Berechnung.

FORMAT: TR STOP n

n : Zeilennummer



ERLÄUTERUNG:

1. Die Programmausführung wird unmittelbar vor der ersten Anweisung der Zeile n unter brochen. Mit dem CONT/-Befehl (siehe Seite 82) wird die Ausführung fortgesetzt.
2. Mit Funktion 3 des TR-Befehls (siehe Seite 101) wird auch der TR STOP-Befehl aufgehoben.

EINFACHES BEISPIELE:

```
    ⋮
250 A=A+1
    ⋮
```

Das Programm soll vor der Zeile 250 angehalten, der Inhalt von A geprüft und wenn nötig, verändert werden. Dazu spezifiziere man vor dem Start des Programms:

TR STOP 250 [ENTER]

Am Anfang von Zeile 250 wird das Programm unterbrochen. Nun kann man z.B.

If DISP A [ENTER]

eingeben, um den Inhalt von A zu inspizieren. Soll A von Hand geändert, beispielweise der Wert 100 zugewiesen werden, schreibe man

A=100 [ENTER]

Zur Fortsetzung des Programms mit Zeile 250 drückt man die [CONT] -Taste oder gibt ein: CONT [ENTER]

WICHTIGER HINWEIS

1. Der TR STOP-Befehl hat keinen Einfluß auf den Trace-Ablauftest. Die Tracefunktion wird durch den TR-Befehl, die TR STOP-Funktion durch den TR STOP-Befehl aktiviert.
   Zum Abschalten beider Funktionen dient der TR OFF-Befehl.
2. Wenn die angegebene Zeilennummer nicht existiert, ist der TR STOP-Befehl ungültig.
3. Im Gegensatz zur STOP-Anweisung (siehe Seite 126) gestattet der TR STOP-Befehl, das Programm gezielt "von Hand" and einer bestimmten Zeile anzuhalten, auch wenn die Tracefunktion nicht aktiv ist.

3. STANDARD-ANWEISUNGEN

Die Standard-Anweisungen bilden die Grundlage für das Programmieren in der Sprache BASIC.
Folgende Standard-Anweisungen werden in diesem Kapitel beschrieben:

LET ※
DISP ※
PRINT ※
IMAGE
INPUT ※
DATA
READ
RESTORE
REM ※
STOP
END ※
DIM ※
GO TO
GO SUB
RETURN
ON……GO TO
ON……GO SUB
IF……THEN
IF…… : ELSE
FOR……TO……STEP／NEXT

Mit einem Stern ※ markierte Anweisungen können auch im manuellen Betrieb benutzt werden.

LET

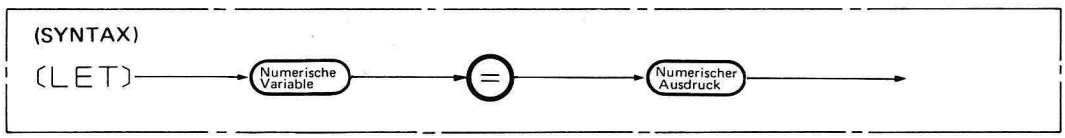
Zuweisung von Werten (numerische und Strings) an Variablen.

〈VERWENDUNG〉
① Abspeichern von Konstanten und Rechenergebnissen durch Zuweisung an numerische Variablen.
② Abspeichern von Strings durch Zuweisung an Stringvariablen.

● **FUNKTION 1:** Zuweisung numerischer Werte

FORMAT: 〔LET〕 A=X

A : numerische Variable oder Element eines numerischen Arrays
X : numerischer Ausdruck



ERLÄUTERUNG:

1. Der numerische Ausdruck X auf der rechten Seite des Gleichheitszeichens (=) wird berechnet und das Resultat der auf der linken Seite stehenden Variablen A zugewiesen.
   Das Gleichheitszeichen bedeutet in diesem Zusammenhang keine Gleichsetzung im mathematisch üblichen Sinn, sondern bewirkt vielmehr einen Transport von Daten! Auf der linken Seite steht immer der Name der einfachen Variablen oder des Arrayelements, dem ein Wert zugewiesen werden soll, numerische Ausdrücke sind an dieser Stelle nicht zulässig!
2. Das Schlüsselwort LET kann weggelassen werden, wie es in modernen Implementationen von BASIC üblich ist. Zum Beispiel haben die Anweisungen LET A = 10 völlig gleiche Wirkung.
3. Die LET-Anweisung kann auch manuell benutzt werden.

EINFACHES BEISPIELE:

1. `10 LET A=3.14` ..... Zuweisung der numerischen Konstante 3.14 an die numerische Variable A.
2. `10 B=B+1` ..... Zum bisherigen Inhalt von B wird 1 addiert und das Ergebnis wieder derselben Variablen B zugewieden.
3. `10 M=(X/Y-5)` ..... Berechnung des eingeklammerten numerischen Ausdrucks und Zuweisung des Ergebnisses an die Variable M.
4. `N=SIN 30` [ENTER] ..... Zuweisung des Sinus von 30 an die Variable N (Beispiel für manuelle Verwendung von LET.)

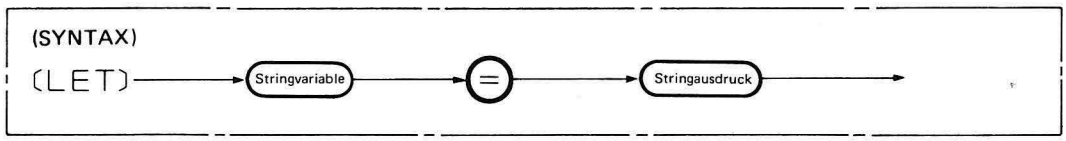
In den Beispielen 2, 3 und 4 wurde zur Demonstration auf das Schlüsselwort LET verzichtet.

● FUNKTION 2: Zuweisung von Strings

FORMAT:(LET) A$=C

A$: Stringvariable oder Element eines Stringarrays.

C : Stringausdruck



ERLÄUTERUNG:

Der Stringausdruck C wird ausgewertet und das Resultat der Stringvariablen A$ zugewiesen.

Genau wie in Funktion 1 zeigt das Gleichheitszeichen (=) einen Wertzuweisung und nicht eine Gleichsetzung im mathematisch üblichen Sinn an!

Weitere Einzelheiten siehe Funktion 1.

EINFACHES BEISPIELE:

1. `10 LET A$="BILDSCHIRM"` ..... Zuweisung der Stringkonstanten "BILDSCHIRM" an die Stringvariable A$.
2. `10 NAME$="BASIC"+"-Handbuch"`
   ..... die beiden Stringkonstanten werden aneinandergefügt und das Ergebnis "BASIC-Handbuch" der Variablen NAME$ zugewiesen.
3. `10 M$=CHR$ 65` ..... Das Zeichen "A" (dezimaler Kodwert = 65) wird der Variablen M$ zugewiesen.
4. `P$="MEYER"` [ENTER] ..... Zuweisung der Stringkonstanten "MEYER" an die Variable M$ (Beispiel für manuelle Verwendung von LET.)

In den Beispielen 2, 3 und 4 wurde zur Demonstration auf das Schlüsselwort LET verzichtet.

WICHTIGER HINWEIS

Der Typ der Variablen auf der linken Seite des Wertzuweisungszeichens (=) muß mit dem Typ des Ausdrucks rechts davon übereinstimmen. Numerische Werte können nur numerischen Variablen, Strings nur Stringvariablen zugewiesen werden!

ANWENDUNGSBEISPIELE:

Programm A, Addition und Subtraktion.

```
10 A=10 ..... Der numerischen Variablen A wird 10 zugewiesen.
20 B=20 ..... Der numerischen Variablen B wird 20 zugewiesen.
30 C=A+B ..... Der numerischen Variablen C wird die Summe von A+B (30 in diesem Fall)
               zugewiesen.
40 D=A-B ..... Der numerischen Variablen D wird die Differenz aus A-B (-10 in diesem
               Fall) zugewiesen.
50 DISP C ..... Ausgabe des Wertes der numerischen Variable C.
60 DISP D ..... Auggabe des Wertes der numerischen Variable D.
70 END ..... Enden des Programms.
```

Programm B, einfache Stringverarbeitung.

```
10 A$="Guten " ... Der Stringvariablen A$ wird "Ich bin" zugewiesen.
20 B$="Morgen!". Der Stringvariablen B$ wird "ein Junge." zugewiesen.
30 C$=A$+B$ ..... Der Stringvariablen C$ wird A$ + B$ ("Ich bin ein Junge.") zugewiesen.
40 DISP C$ ..... Ausgabe des Inhalts der Stringvariablen C$ auf dem Bildschirm.
50 END ..... Enden des Programms.
```

Strings können (neben der Anwendung von Stringfunktionen) nur mit dem Operator + verarbeitet werden, der bei Strings jedoch keine Addition, sondern eine Verkettung der Stringoperanden durch Aneinanderhängen ("Konkatentation") bewirkt.

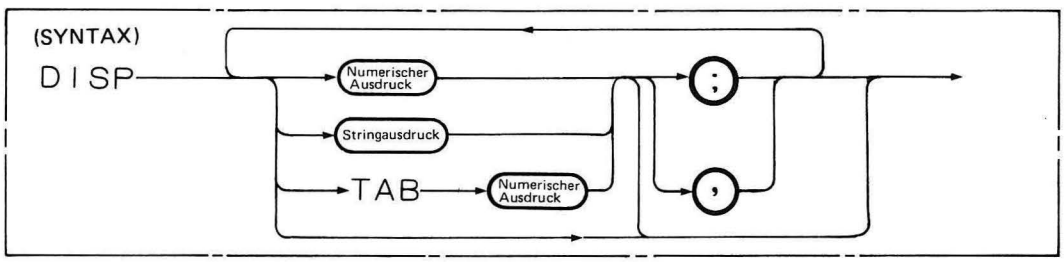
DISP

Ausgabe von numerischen Daten und Strings auf dem Bildschirm.

〈VERWENDUNG〉
Darstellung der Ergebnisse von Berechnungen und Stringoperationen auf dem Bildschirm.

● **FUNKTION 1:** Ausgabe von Werten und Steuerung des Kursors.
FORMAT: D I SP 〔 {{**X** | **C** | **TAB Y**}} 〔; | , 〕〕

X : numerischer Ausdruck
C : Stringausdruck
Y : Kursorposition (numerischer Ausdruck)
$0 \leqq Y \leqq 79$ bei 80-Spalten-Modus
$0 \leqq Y \leqq 39$ bei 40-Spalten-Modus
(Einzelheiten siehe DISP CLEAR, Seite 215)



ERLÄUTERUNG:

1. Der Wert des numerischen Ausdrucks X oder der Inhalt des Stringausdrucks C wird auf dem Bildschirm ausgegeben, oder der Kursor fährt auf die durch Y benannte Spalte. (Linker Rand = Spalte 0.)
   Steht ein Semikolon (;) zwischen zwei Argumenten, so werden sie ohne Einschieben von Leerstellen direkt hintereinander ausgegeben.
   Benutzt man ein Komma (,) als Trennzeichen, fährt der Kursor automatisch auf den nächsten Blockanfang (jede 20. Spalte), bevor der nächste Operand ausgegeben wird. Bei langen Operanden (z.B. Strings aus mehr als 19 Zeichen) werden ggf. mehrere Blöcke übersprungen.
   Negative Zahlen werden von einem Minuszeichen eingeleitet, vor Null und positiven Zahlen steht stattdessen ein Leerzeichen. Außerdem folgt ein Leerzeichen auf alle Zahlen.
   Strings werden wie spezifiert dargestellt.
2. Die TAB-Anweisung bewirkt, daß der Kursor auf die im Argument von TAB gegebene Spalte auf dem Bildschirm, gezählt vom linken Rand, fährt.
3. Folgen überhaupt keine Argumente hinter DISP (numerische Ausdrücke, Strings, TAB usw. alles ausgelassen),so erhält man eine Leerzeile auf dem Bildschirm.

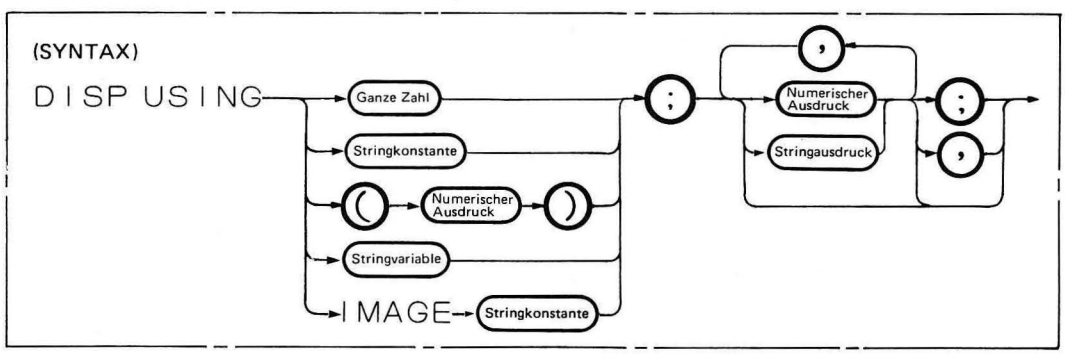
EINFACHES BEISPIELE:

1. `10 DISP A,B` ..... Der Inhalt der Variablen A wird dargestellt, der Kursor fährt auf den nächsten Block, es folgt der Wert der Variablen B, und der Kursor springt an den Anfang der nächsten Zeile.
2. `10 DISP "ANTWORT",` ..... Nach der Ausgabe der Stringkonstanten "ANTWORT" springt der Kursor wegen des Kommas am Ende der Anweisung an den Anfang des nächsten Blocks und verbleibt dort bis zur nächsten DISP-Anweisung.
3. `10 DISP TAB A*2;` ..... Die TAB-Anweisung fährt den Kursor auf die durch den Wert des Arguments von TAB bezeichnete Spalte, wo er wegen des abschließenden Semikolons bis zur nächsten DISP-Anweisung verbleibt. Sei z.B. A = 10, so steht der Kursor nun in Spalte 20 (von links gezählt).
4. `10 DISP A,"+",B;"=",C` ··Die Ausgabe beginnt mit dem Inhalt von A, am Anfang des nächsten Blocks folgt das Zeichen + , am übernächsten Block der Inhalt von B, direkt dahinter das Zeichen = und noch einen Block weiter schließt die Ausgabeziele mit dem Inhalt von C ab. Der Kursor steht danach am Anfang der nächsten Bildschirmzeile.

● **FUNKTION 2:** Ausgabe von Werten gemäß einem vorgegebenen Format.

FORMAT: DISP USING n | S | (Z) | A$ | IMAGE T ; [{X | C}] [; | ,]

- n : Zeilennummer einer IMAGE-Anweisung
- S : Zeilenmarke einer IMAGE-Anweisung
- Z : Zeilennummer (numerischer Ausdruck) einer IMAGE-Anweisung
- A$: Stringvariable, die die Formatierung enthält ("Maske")
- T : Stringkonstante mit der Ausgabemaske
- X : Numerischer Ausdruck
- C : Stringausdruck



ANMERKUNG:

Der Einfachheit halber wurde bei der Beschreibung der Wirkung der Trennzeichen (, und ü) sowie bei TAB von "Kursor" gesprochen. In Wirklichkeit ist der Kursor während der Programmausführung normalerweise unsichtbar. Die Erläuterungen beziehen sich genaugenommen auf die aktuelle Position des Kursors als Ausgabezeiger.

ERLÄUTERUNG:

1. Ausgabe von Zahlenwerten und Strings gemäß einer speziellen Formatierung oder "Maske". Einzelheiten siehe IMAGE-Anweisung auf Seite 113.
2. Bei dieser Form der DISP-Anweisung werden die auszugebenden Daten einer speziellen Formatierung unterworfen. Die Formatierungsvorschrift oder Maske kann auf vielfältige Wiese spezifiziert werden: durch die Zeilennummer n (statt einer Konstanten ist auch ein numerischer Ausdruck in Klammern möglich, Dezimalstellen werden in beiden Fällen abgeschnitten) oder die Zeilenmarke S einer IMAGE-Anweisung, durch den Inhalt einer Stringvariablen A$ oder durch eine in die DISP-Anweisung eingebettete IMAGE-Anweisung.
3. Zwischen der Maske (USING-Teil der Anweisung) und der Angabe der auszugebenden Werte muß ein Semikolon stehen.

WICHTIGER HINWEIS

Endet eine DISP-Anweisung mit einem Semikolon (;), so verbleibt der Kursor nach vollendeter Ausgabe an der gegenwärtigen Position auf dem Bildschirm, bis die nächste DISP-Anweisung aufgeführt wird oder das Programm durch eine END-Anweisung seinen Lauf beendet.

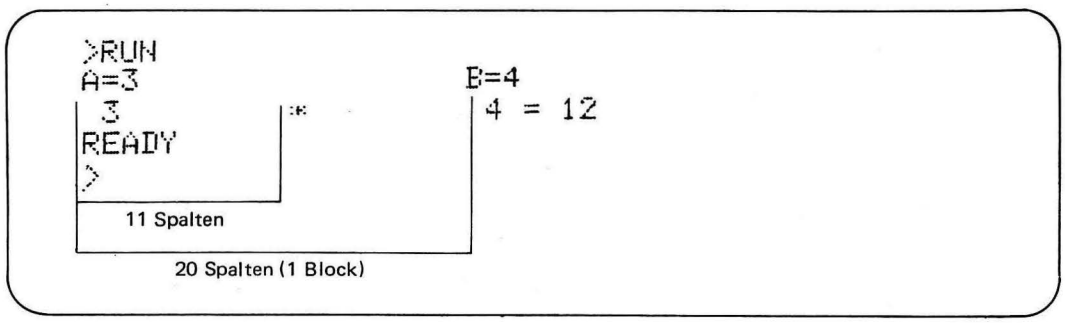
Endet eine DISP-Anweisung mit einem Komma (,), so geht der Kursor auf die erste Spalte des nächstliegenden 20-Spalten-Blocks nach rechts und verbleibt dort vorläufig.

ANWENDUNGSBEISPIEL:

Obacht bei der Benutzung von Kommata und Semikolons ist angezeigt!

```
10 INPUT "A=";A,"B=";B ··· Tastatureingabe von Daten für die numerischen Variablen
                           A und B.
20 C=A*B ····· Zuweisen des Produkts AxB an die numerische Variable C.
30 DISP A;TAB 11;"*",B;"=";C ····· Ausgabe des Ergebnisses.
40 END
```

Wir starten das Programm und geben für A einen Wert von 3, für B eine 4 ein. Nach der Beendigung des Programmlaufes steht auf dem Bildschirm:



ANMERKUNG:

Der Einfachheit halber wurde bei der Beschreibung der Wirkung der Trennzeichen (, und ;) sowie bei TAB von "Kursor" gesprochen. In Wirklichkeit ist der Kursor während der Programmausführung normalerweise unsichtbar. Die Erläuterungen beziehen sich genaugenommen auf die aktuelle Position des Kursors als Ausgabezeiger.

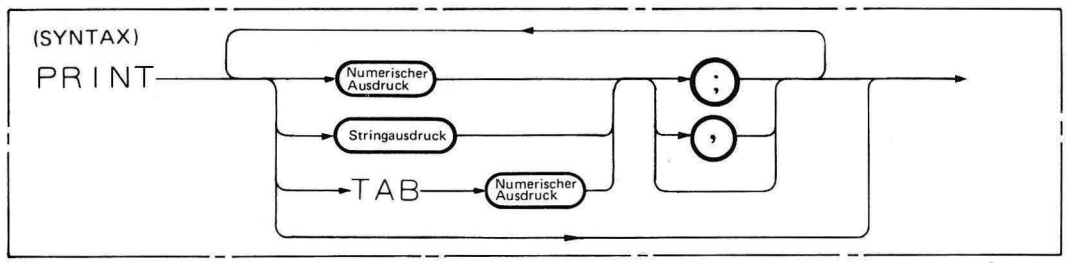
PRINT

Ausgabe von numerischen Daten und Strings auf dem Drucker.

〈VERWENDUNG〉
Darstellung der Ergebnisse von Berechnungen und Stringoperationen auf dem Drucker.

● **FUNKTION 1:** Drucken von Werten und Steuerung des Druckkopfes.
FORMAT: PRINT [{{**X** | **C** | **TAB Y**}} [; | ,]]

X: numerischer Ausdruck
C: Stringausdruck
Y: Druckkopfposition (numerischer Ausdruck)



ERLÄUTERUNG:

1. Der Wert des numerischen Ausdrucks X oder der Inhalt des Stringausdrucks C wird auf dem Drucker ausgegeben, oder der Druckkopf fährt auf die durch Y benannte Spalte. (Linker Rand = Spalte 0.)
2. Die Wirkung von Kommata oder Semikolons als Trennzeichen zwischen zwei Argumenten und das Fromat von Zahlen ist dieselbe wie bei der DISP-Anweisung, siehe Seite 107.
3. Folgen überhaupt keine Argumente hinter PRINT (numerische Ausdrücke, Strings, TAB usw. alles ausgelassen), so erhält man eine Leerzeile auf dem Drucker.

EINFACHES BEISPIELE:

1. `10 PRINT A,B` ..... Der Inhalt der Variablen A wird ausgedruckt, der Druckkopf fährt auf den nächsten Block, es folgt der Wert der Variablen B, und der Druckkopf springt an den Anfang der nächsten Zeile.
2. `10 PRINT "ANTWORT",` ..... Nach der Ausgabe der Stringkonstanten "ANTWORT" springt der Druckkopf wegen des Kommas am Ende der Anweisung an den Anfang des nächsten Blocks und verbleibt dort bis zur nächsten PRINT-Anweisung.
3. `10 PRINT TAB A*2;` ..... Die TAB-Anweisung fährt den Druckkopf auf die durch den Wert des Arguments von TAB bezeichnete Spalte, wo er wegen des abschließenden Semikolons bis zur nächsten PRINT-Anweisung verbleibt. Sei z.B. A = 5, so steht der Druckkopf nur in Spalte 10 von links.

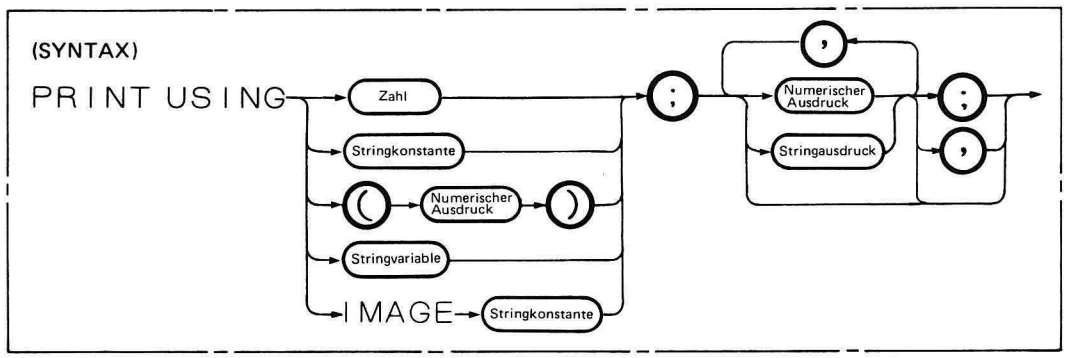
4.10 PRINT A,TAB 30;B$,C

..... Die Ausgabe beginnt mit dem Inhalt von A, ab Spalte 30 von links folgt der Inhalt von B$ und am Anfang des folgenden Blocks schließlich der Inhalt von C. Der Druckkopf steht danach am Anfang der nächsten Druckzeile.

● **FUNKTION 2:** Drucken von Werten gemäß einem vorgegebenen Format.

FORMAT:PRINT USING n | S | (Z) | A$ | IMAGE T ; [{X | C}] [; | ,]

n : Zeilennummer einer IMAGE-Anweisung
S : Zeilenmarke einer IMAGE-Anweisung
Z : Zeilennummer (numerischer Ausdruck) einer IMAGE-Anweisung
A$: Stringvariable, die die Formatierung enthält ("Maske")
T : Stringkonstante mit der Ausgabemaske
X : numerischer Ausdruck
C : Stringausdruck



ERLÄUTERUNG:

1. Ausgabe von Zahlenwerten und Strings gemäß einer speziellen Formatierung oder "Maske". Einzelheiten siehe IMAGE-Anweisung, Seite 113.
2. Bei dieser Form der PRINT-Anweisung werden die auszugebenden Daten einer speziellen Formatierung unterworfen. Alles bei der DISP-Anweisung auf Seite 108 Gesagte gilt auch hier.
3. Zwischen der Maske (USING-Teil der Anweisung) und der Angabe der auszugebenden Werte muß ein Semikolon stehen.

WICHTIGER HINWEIS

1. Endet eine PRINT-Anweisung innerhalb eines Programms mit einem Semikolon (;), so verbleibt der Druckkopf nach vollendeter Ausgabe an der gegenwärtigen Position auf dem Drucker, bis die nächste PRINT-Anweisung ausgeführt wird oder das Programm durch eine END-Anweisung seinen Lauf beendet.
   Endet eine PRINT-Anweisung mit einem Komma (,), so geht der Druckkopf auf die erste Spalte des nächstliegenden 20-Spalten-Blocks nach rechts und verbleibt dort vorläufig.
2. Wird eine PRINT-Anweisung ausgeführt, während kein Drucker an den Rechner angeschlossen oder kein Papier im Drucker ist, erfolgt eine Fehlermeldung (ERROR 25).

3. Die meisten Drucker speichern intern eine vom Rechner gesendete Zeile und drucken erst nach dem Eintreffen bestimmter Steuerzeichen vom Rechner. Von den oben beschriebenen Bewegungen des Druckkopfes wird man am Drucker selbst nichts sehen, das gewünschte Druckbild kommt jedoch richtig zustande.

ANWENDUNGSBEISPIEL:

Programm zur Aufteilung eines Geldbetrages in Noten- und Münzenwerte, kurz ein "Sortenprogramm".

```
10 PRINT "***** SORTENBERECHNUNG *****"
20 DIM S(12)
30 S(1)=100:S(2)=50:S(3)=20:S(4)=10
40 S(5)=5:S(6)=2:S(7)=1:S(8)=0.5
50 S(9)=0.1:S(10)=0.05:S(11)=0.02:S(12)=0.01
60 INPUT "BETRAG";B;"DM"
70 PRINT "***** BETRAG = ";B
80 FOR J=1 TO 12
90 A=INT (B/S(J))
100 PRINT S(J);"DM";TAB (13);A
110 B=B-S(J)*A
120 NEXT J
130 PRINT "********************************"
140 END
```

Wir starten das Programm und geben für B einen Betrag von 1638.28 DM ein. Nach der Beendigung steht auf dem Bildschirm:

```
>RUN
BETRAG =1638.28
READY
>
```

Auf dem Drucker erhalten wir folgende Aufstellung:

```
***** SORTENBERECHNUNG *****
***** BETRAG =  1638.28 DM
 100 DM         16
 50 DM          0
 20 DM          1
 10 DM          1
 5 DM           1
 2 DM           1
 1 DM           1
 0.5 DM         0
 0.1 DM         2
 0.05 DM        1
 0.02 DM        1
 0.01 DM        1
********************************
```

IMAGE

Spezifikation einer Ausgabe-Formatierung.

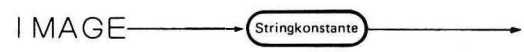
〈VERWENDUNG〉

Aufbereitung von Daten für die Ausgabe mit Hilfe einer Maske, um auf Bildschirm oder Drucker ein spezielles Bild zu erzielen.

FORMAT: I MAGE S

S: Stringkonstante, deren Inhalt zur Steuerung der Aufbereitung dient.

(SYNTAX)



ERLÄUTERUNG:

Bei der Ausgabe von Zahlen und Strings ermöglicht die IMAGE-Anweisung eine über die bisher beschriebenen Möglichkeiten hinausgehende Gestaltung des Ausgabebildes. Die in der (häufig "Maske" genannten) Stringkonstanten S stehenden Zeichen werden dazu von den Anweisungen DISP und PRINT auf bestimmte Weise interpretiert. Die Format symbole werden unterschieden nach variablenbezogenen und unabhängigen.

Variablenbezogene Formatsymbole

Typ	Symbol	Wieder-holung	Funktion
Strings	A	JA	steht für ein Zeichen eines Strings.
Zahlen	#	nein	Ziffernstelle mit Nullunterdrückung.*)
	Z	nein	Ziffernstelle ohne Nullunterdrückung.
	*	nein	Ziffernstelle, bei der führende Nullen durch Schutzsterne (*) ersetzt werden.
	.	nein	Dezimalpunkt.
	,	nein	Trennkomma für Dreiergruppen von Stellen vor dem Dezimalpunkt.
	E	nein	Beginn des Exponenten.
	+	nein	Vorzeichen: – für negative, + für positive Werte.
	–	nein	Vorzeichen: – für negative, Leerstelle für positive Werte.
	$	nein	Währungssymbol ($) vor die Zahl setzen.
Spezial	%	nein	zeigt an, daß dem % folgende Zeichen so oft zu wiederholen ist, wie der Wert der zugehörigen Variablen angibt.

*) Nullunterdrückung heißt, daß bedeutungslose Nullen vor der ersten signifikanten Stelle oder hinter dem Dezimalteil einer Zahl durch Leerstellen ersetzt werden, genau so, wie man Zahlen übicherweise schreibt.

Unabhängige Formatsymbole

Typ	Symbol	Wiederholung	Funktion
Leerstelle	X	JA	Leerstelle einfügen.
Fester String	"...." '....'	JA	Stringkonstante einfügen. Diese ist in "...." oder '...' eingeschlossen zu schreiben.
Neue Zeile	L	JA	Beginn einer neuen Zeile.
Sonderzeichen für Drucker etc.	N	JA	NULL (&00) : Nullzeichen
	BL	JA	BEL (&07) : akustisches Signal (Bell)
	BS	JA	BS (&08) : Rückschritt (Backspace)
	T	JA	TAB (&09) : Tabulation
	FF	JA	FF (&0C) : Seitenvorschub (Form feed)
	C	JA	CR (&0D) : Wagenrücklauf (Carriage return)
	SO	JA	SO (&0E) : Umschaltung ein (Shift-out)
	SI	JA	SI (&0F) : Umschaltung aus (Shift-in)
	CN	JA	CAN (&18) : Löschen (Cancel)
	ES	JA	ESC (&1B) : "Fluchtsymbol" (Escape)

WICHTIGER HINWEIS

Für die Wirkung der im letzten Teil der Tabelle aufgeführten Sonderzeichen ziehe man die Handbücher der entsprechenden Drucker oder anderen Peripheriegeräte zu Rate!

Näheres zur Formatierung

Die vielfältigen Möglichkeiten der Formatierung studiere man am besten an Hand der folgenden Beispiele.

a) Wiederholung

Alle in den Tabellen mit einem "JA" gekennzeichneten Formatsymbole können durch Vorstellen einer ganzen Zahl wiederholt werden. So haben z.B. "3A" und "AAA" die gleiche Bedeutung.

b) Stehen mehrere variablenbezogene Formatsymbole in derselben IMAGE-Anweisung, so sind sie durch mindestens eine Leerstelle voneinander zu trennen. Sinngemäß dasselbe gilt für den Inhalt von zur Formatierung benutzten Stringvariablen. Die Trennleerstellen dienen nur zum Aufschreiben der Formatsymbole, sie werden nicht mit ausgegeben (siehe Formatsymbol X).

EINFACHES BEISPIELE:

```
10 IMAGE "10A 2X $###,###"
```

Spezifiziert 10 Zeichen für das erste Argument der DISP- oder PRINT-Anweisung (muß vom Typ String sein), gefolgt von 2 Leerstellen. Das zweite Argument (numerisch) wird mit Unterdrückung führender Nullen, einem Dollarzeichen vor der Zahl und einem Komma zwischen der Zehntausender- und der Tausenderstellen ausgegeben.

WICHTIGER HINWEIS

1. Fehlt in einem Zahlenformat die Angabe eines Vorzeichens, so wird auch bei negativen Werten kein Vorzeichen ausgegeben! Wenn die Zahl nicht im Exponentenformat ausgegeben werden soll, so kann das Vorzeichen + oder – statt vor- auch nachgestellt werden.

```
10 A=-1234
20 PRINT USING IMAGE "##,###";A
30 PRINT USING IMAGE "-##,###";A
40 PRINT USING IMAGE "+##,###";A
50 PRINT USING IMAGE "ZZ,ZZZ+";A
60 PRINT USING IMAGE "ZZ,ZZZ-";A
70 END
```

Siehe auch Funktion 2 der PRINT-Anweisung. Druckbild:

```
 1,234
 -1,234
 -1,234
01,234-
01,234-
```

2. Wenn die Maske selbst in Anführungsstriche (") eingeschlossen ist, so müssen eingebettete Stringkonstanten in Apostrophe eingeschlossen werden, und umgekehrt. Es muß also z.B. heißen:
IMAGE "20A 'Adresse'" oder IMAGE '20A "Adresse"'.

ANWENDUNGSBEISPIELE:

Programm A

```
10 A$="SCHMIDT":B=158200
20 PRINT USING 30;A$,B:END
30 IMAGE '"NAME" 3X 7A 4X $#,###,###'
40 END
```

Ergebnis

```
NAME   SCHMIDT      $158,200
```

Programm B

```
 10 FOR A=1 TO 5
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "10X -##Z.##"
110 DATA 123,-1.256,0,-987.1,.05
120 END
```

Ergebnis

```
>RUN
             123.00
              -1.25 ..... dritte Dezimale abgeschnitten
               0.00
            -987.10
               0.05
READY
>
```

Programm C

```
 10 FOR A=1 TO 3
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "5X +#,***.**E02"
110 DATA 123456,2E4,-25
120 END
```

Ergebnis

```
>RUN
      +1,234.56E+02
        +200.00E+02
        -**0.25E+02
READY
>
```

Programm D

```
 10 FOR A=1 TO 3
 20 READ B
 30 DISP USING 100;B
 40 NEXT
 50 END
100 IMAGE "5X -#,###.## L"
110 DATA 123,6E3,-25.11
120 END
```

Ergebnis

```
>RUN
          123.00
                 ..... eine Leerzeile
        6,000.00
                 ..... eine Leerzeile
          -25.11
                 ..... eine Leerzeile
READY
>
```

INPUT

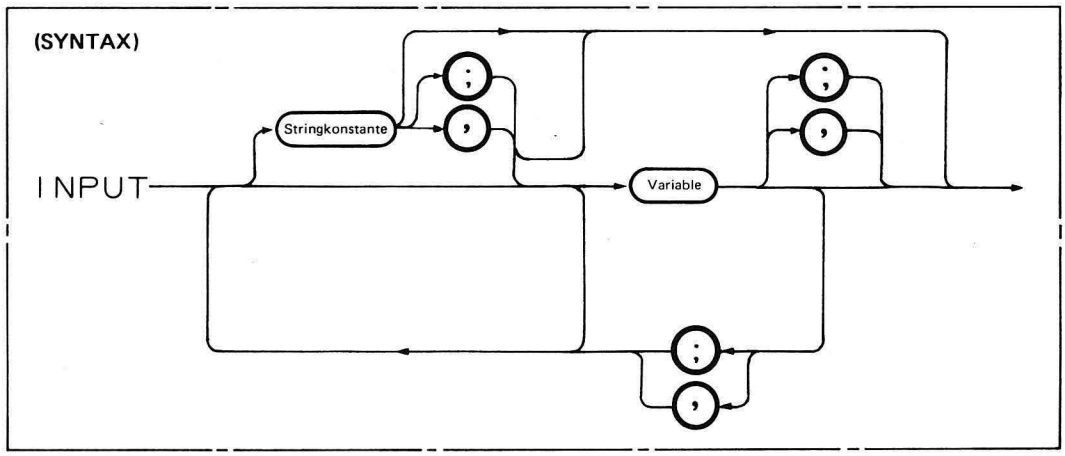
Eingabe von Zahlen oder Strings an der Tastatur.

〈VERWENDUNG〉
Eingabe von Daten an ein laufendes Programm.

FORMAT: INPUT [S ; | ,] M [; | , {[T | N}]] [; | ,]

S, T : Stringkonstante zur Erläuterung der Eingabe

M, N : Variable, denen Werte per Tastatur zugewiesen werden sollen.



ERLÄUTERUNG:

1. Die Stringkonstante S ("Prompt string") wird, wenn vorhanden, auf dem Bildschirm angezeigt. Sie soll dem Bediener anzeigen, welche Daten er einzugeben hat. Der Rechner wartet dann auf eine Eingabe durch den Bediener. Nach deren Abschluß werden die eingegebenen Daten der in INPUT spezifizierten Variablen M zugewiesen. Steht ein Semikolon als Trennzeichen zwischen der Stringkonstantren S und der Variablen, so wartet der Kursor hinter dem String, bei einem Komma rückt er auf den nächsten 20-Spalten-Block vor, gezählt vom ersten Zeichen des Strings.
2. Steht vor nachfolgenden Strings/Variablen ein Semikolon, werden die Daten ohne eingeschobene Leerstellen dargestellt, bei einem Komma erfolgt die Darstellung in Blockeinheiten.
3. Wird die INPUT-Anweisung in einem Programm benutzt und endet sie mit einem Komma oder Semikolon, gilt für die Kursorpositionierung das bei der DISP-Anweisung (siehe Seite 107) gesagte.

EINFACHES BEISPIELE:

1. `10 INPUT A` ..... Wartet auf eine Tastatureingabe und weist die eingegebene Zahl der Variablen A zu.
2. `10 INPUT "Wie ist Ihr Gewicht?";B`
   ..... Gibt "Wie ist Ihr Gewicht?" auf dem Bildschirm aus, erwartet eine Eingabe auf der Tastatur und weist die eingegebene Zahl B zu.
3. `10 INPUT "Wie ist Ihr Gewicht?";B;"kg"`
   ..... Ähnlich wie Beispiel 2, jedoch wird nach der Eingabe der Zahl noch kg angehängt. Hat z.B. 50 eingegeben, so steht auf dem Bildschirm: Wie ist Ihr Gewicht? 50 kg.
4. `10 INPUT "A=";A,"B=";B` ..... Ausgabe von A= auf dem Bildschirm, Warten auf eine Eingabe für die Variable A, Kursor springt auf den nächsten Blockanfang, Ausgabe von B= und Warten auf eine Eingabe für B.

WICHTIGER HINWEIS

1. Bei der Eingabe von Strings ist es nicht erforderlich, diese in Apostrophe oder Anführungszeichen (" oder ') einzuschließen, es sei denn, sie sind Bestandteile der einzugebenden Strings.
2. Hat man ohne die Eingabe von Daten sofort die [ENTER] -Taste gedrückt, so wird einer numerischen Variablen der Wert 0 und einer Stringvariablen ein Leerstring oder Nullstring zugewiesen (" " = leerer String, enthält überhaupt keine Zeichen, seine Länge ist 0). War einer Variablen jedoch vor der INPUT-Anweisung schon ein Wert zugewiesen worden, so bleibt dieser unverändert.
3. Die eingegebenen Daten müssen mit dem Typ der korrespondierenden Variablen übereinstimmen, sonst erfolgt eine Fehlermeldung (ERROR 21), bei Eingaben von Daten für numerische Variablen sind also die Regeln für numerische Konstanten zu beachten, siehe Seite 9. Eingabefehler können mit der [CL] - oder der [↑] -Taste korrigiert werden.

ANWENDUNGSBEISPIEL:

Programm A: Addition und Multiplikation mit per INPUT eingegebenen Daten.

```
10 INPUT "A=";A ..... Ausgabe von "A=" auf dem Bildschirm, Zuweisen des eingegebenen Wertes an die numerische Variable A.
20 INPUT "B=";B ..... Ausgabe von "B=" auf dem Bildschirm, Zuweisen des eingegebenen Wertes an die numerische Variable B.
30 C=A+B ..... Summe A+B an die numerische Variable C zuweisen.
40 D=A*B ..... Produkt A*B an die numerische Variable D zuweisen.
50 DISP "A+B=";C .... Ausgabe der Summe in der Form A+B=Wert von C auf dem Bildschirm.
60 DISP "A*B=";D .... Ausgabe des Produkts in der Form A*B=Wert von D auf dem Bildschirm.
70 END
```

Wir starten das Programm und geben bei den INPUT-Anweisunggen ein:

10 ENTER
20 ENTER

Die Ergebnisse lauten dann:

```
>RUN
A=10
B=20
A+B= 30
A*B= 200
READY
>
```

Programm B: Dieselben Berechnungen wie Programm A, aber andere Form der Ausgabe:

```
10 INPUT "A=";A,"B=";B,
20 C=A+B:D=A*B
30 DISP "A+B=";C,"A*B=";D
40 END
```

Nacd Eingabe eines Wertes für die numerische Variable B springt der Kursor an den Anfang des nächsten Blocks.

Ergebnisse für A=10 und B=20:

```
>RUN
A=10          B=20          A+B= 30          A*B= 200
READY
>
```

DATA

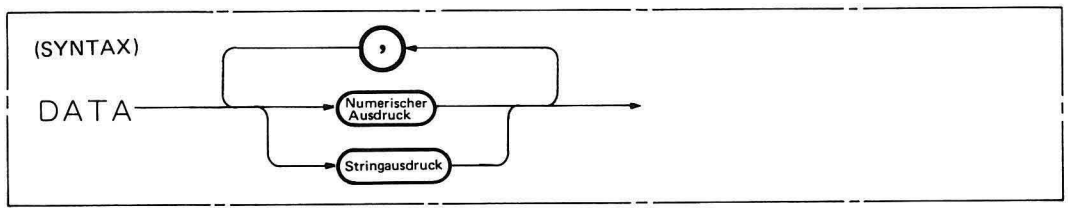
Bereitstellen von Daten im Programm für READ-Anweisungen.

〈VERWENDUNG〉
Aufnahme von festen Daten ins Programm

FORMAT: DATA 〔X | C〕

X: numerischer Ausdruck

C: Stringausdruck



ERLÄUTERUNG:

1. Die DATA-Anweisung ist im Verbund mit der auf der nächsten Seite beschriebenen READ-Anweisung zu verwenden, welche den aufgeführten Daten Variablen zuweist. Variablen zuweist.
2. In einem Programm können mehrere DATA-Anweisungen benutzt werden. Auch wenn sie nicht unmittelbar aufeinanderfolgen, gelten sie für die READ-Anweisung doch als kontinuierlich.
3. DATA-Anweisungen können an beliebigen Stellen im Programm stehen, ohne Rücksicht auf die Lage der READ-Anweisungen.

EINFACHES BEISPIELE:

1. `10 DATA 12,345,6.7` ..... die numerischen Konstanten 12, 345 und 6.7 werden als Daten spezifiziert.
2. `10 DATA "ABC","XYZ"` ..... Aufführen der Stringkonstanten "ABC" und "XYZ" als Daten.
3. `10 DATA 12,"SHARP",34.5` ..... Numerische Konstante 12, Stringkonstante "SHARP" und numerische Konstante 34.5.

ANWENDUNGSBEISPIEL:

```
 10 READ A$,B$
 20 DISP A$,B$
 30 END
100 DATA "KAFFEE","1,00 DM"
```

In diesem Beispiel wird der Stringvariablen A$ der String KAFFEE und der Variablen B$ der String 1.00 DM zugewiesen und beides auf dem Bildschirm dargestellt.
Weitere Einzelheiten siehe READ-Anweisung auf der nächsten Seite.

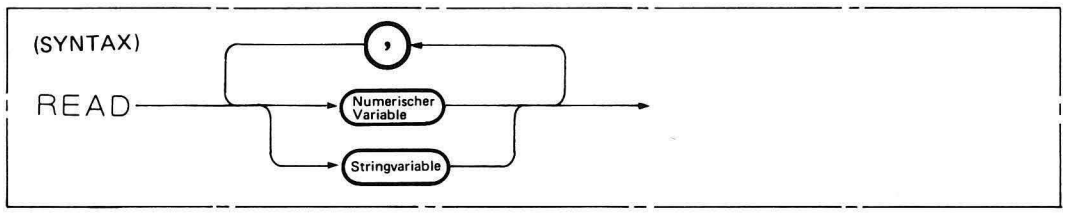
READ

Einlesen der durch DATA bereitgestellten Daten und Zuweisung der Werte an Variable.

FORMAT: READ {A | A$}

A : Numerische Variable

A$: Stringvariable



ERLÄUTERUNG:

1. Die READ-Anweisung wird in Verbindung mit der auf der vorigen Seite beschriebenen DATA-Anweisung verwendet.
   Bei der Ausführung der ersten READ-Anweisung im Programm beginnt das Lesen in der DATA-Anweisung mit der kleinsten Zeilennummer. Die Daten werden sequentiell gelesen und den in der READ-Anweisung aufgeführten Variablen zugewiesen; d.h. wenn die Daten aus der ersten DATA-Anweisung erschöpft sind, sucht READ nach der nächstfolgenden DATA-Anweisung und liest dort weiter.
   Durch die Angabe einer Zeilennummer in der RESTORE-Anweisung (siehe Seite 123) kann man READ jedoch veranlassen, jede beliebige DATA-Anweisung auszuwerten.
2. Nach vollendeter READ-Anweisung wird die Lage der zuletzt gelesenen Daten im Speicher intern vermerkt, so daß bei der nächsten READ-Anweisung auf die unmittelbar folgenden Daten zugegriffen wird, ohne daß BASIC jedesmal eine neue DATA-Anweisung sucht.

EINFACHES BEISPIELE:

1. `10 READ A,B,C` ..... Die in der DATA-Anweisung vorgefundenen Daten werden sequentiell den numerischen Variablen A, B und C zugewiesen.
2. `10 READ A$,B$,C` ..... Die Daten werden in der aufgeführten Reihenfolge den Stringvariablen A$ und B$, dann der numerischen Variablen C zugewiesen.

WICHTIGER HINWEIS

1. Die Daten in der DATA-Anweisung müssen im Typ mit den entsprechenden Variablen in der READ-Anweisung übereinstimmen.
2. Findet eine READ-Anweisung keine Daten mehr vor, erfolgt eine Fehlermeldung (ERROR 2). Umgekehrt aber darf man mehr Daten aufführen als durch READ gelesen werden, ohne daß ein Fehler gemeldet wird. Auch der Extremfall (DATA-Anweisungen ohne eine einzige READ-Anweisung) ist möglich.

ANWENDUNGSBEISPIEL:

Ausgabe der Getränkekarte am Bildschirm.

```
10 FOR I=1 TO 5 ···In der FOR/NEXT-Schleife wird die Variable I um 1 inkrementet.
20 READ A$,B$ ···Die Daten werden von DATA gelesen und den Variablen A$ und B$ zugeteilt.
30 DISP A$,B$;" DM" ·····Der Inhalt der Variablen A$ und B$ wird auf dem Bildschirm
40 NEXT I                ausgegeben.
50 END
60 DATA "GETRANK","PREIS","KAFFEE","2.50","MILCH",
   "2.00"
70 DATA "COLA","1.50","SAFT","3.00"
```

Ergebnisse:

```
>RUN
GETRANK          PREIS
KAFFEE           2.50 DM
MILCH            2.00 DM
COLA             1.50 DM
SAFT             3.00 DM
READY
>
```

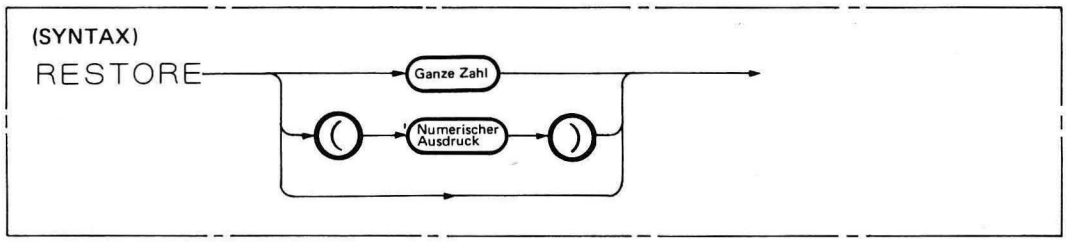
RESTORE

Vorgabe einer bestimmten DATA-Anweisung für die nächste READ-Anweisung.

〈VERWENDUNG〉
Setzt den internen Datenzeiger auf den Anfang einer bestimmten DATA-Anweisung, so daß READ beliebige DATA-Anweisungen verarbeiten kann.

FORMAT: RESTORE [n | (X)]

n: Zeilennummer
X: Numerischer Ausdruck für die Zeilennummer



ERLÄUTERUNG:

1. Die nächste READ-Anweisung beginnt mit dem Lesen am Anfang der in der RESTORE-Anweisung genannten Zeilennummer. Dezimalen werden abgeschnitten.
2. Läßt man die Zeilennummer fort, so setzt RESTORE automatisch die kleinste Zeilennummer einer DATA-Anweisung ein.

EINFACHES BEISPIELE:

1. `10 RESTORE 100` ..... Benennt die DATA-Anweisung in Zeile 100 für die folgende READ-Anweisung.
2. `10 RESTORE (500/3)` ..... Benennt die Zeile 166 (=500/3).
3. `10 RESTORE` ..... Benennt die DATA-Anweisung mit der kleinsten Zeilennummer.

ANWENDUNGSBEISPIEL:

Programm A

```
 10 READ A$,B,C$
 20 RESTORE 110
 30 READ D
 40 DISP A$,B,C$,D
 50 PRINT '10 INPUT A:! "ANZAHL"'
100 DATA "NOTEN"
110 DATA 200
120 DATA "BUCH"
130 END
```

Liest "NOTEN", 200 und "BUCH"

Liest 200

Ergebnisse:

```
>RUN
NOTEN          200            BUCH           200
READY
>
```

Programm B

```
 10 READ A,B,C
 20 DISP A,B,C
 30 READ X$,Y$
 40 DISP X$,Y$
 50 RESTORE 210
 60 READ V$
 70 DISP V$
 80 RESTORE
 90 READ D,E
100 DISP D,E
110 END
200 DATA 100,200,300
210 DATA "EINS","ZWEI"
```

Liest 100, 200 und 300

Liest "EINS" und "ZWEI"

Liest "EINS"

Liest 100 und 200

Ergebnisse

```
>RUN
 100            200            300
EINS           ZWEI
EINS
 100            200
READY
>
```

REM

Kommentar (Remark).

〈VERWENDUNG〉
Einfügen von Kommentaren in ein Programm, um das Listing leichter lesbar zu machen und Programmabschnitte zu dokumentieren.

FORMAT: REM beliebiger Text

(SYNTAX)



ERLÄUTERUNG:

1. Die REM-Anweisung dient zur Aufnahme von Kommentaren in ein Programm. Die Anzahl der REM-Anweisungen und ihre Lage im Programm unterliegt keiner Beschränkung; daher wird ein ausgiebiger Gebrauch von REM-Anweisungen empfohlen. (Im Gegensatz zu getrennten Beschreibungen können REM-Anweisungen verlorengehen!)
2. Die REM-Anweisung ändert den Ablauf eines Programms nicht; trifft der Interpreter eine an, so übergeht er sie und springt sofort auf die nächste Programmzeile. Alle dem REM folgenden Zeichen einschließlich des Doppelpunktes gehören zum Kommentar, daher ist es nicht möglich, hinter einer REM-Anweisung weitere Anweisungen in dieselbe Zeile zu schreiben, sie werden überlesen!
3. Unmittelbar hinter der Zeilennummer kann das Schlüsselwort REM auch durch ein Ansrufezeichen ersetzt werden. Ist der Kommentar jedoch nicht die erste Anweisung in einer Zeile, so muß man REM schreiben.

EINFACHES BEISPIELE:

```
10 REM***PROGRAMM LOHNABRECHNUNG***
```
..... Kommentar zur Anzeige, daß dieses Programm zur Lohnabrechnung dient.

```
oder 10 !***PROGRAMM
     LOHNABRECHNUNG***
```
Beide Formen haben dieselbe Bedeutung.

```
10 INPUT SP:REM StückPreis
```
..... Kommentar, daß hier der Stückpreis an der Tastatur eingegeben und der Variablen SP zugewiesen wird.

WICHTIGER HINWEIS

1. Folgende Konstruktion ist falsch:
   ```
   10 REM "ANZAHL":INPUT A
   ```
   Die Zeile ist zwar syntaktisch richtig, aber INPUT A wird als zum Kommentar gehörig betrachtet und daher nicht ausgeführt, was vermutlich nicht Absicht des Programmierers war!
2. ```
   10 INPUT A:! "ANZAHL"
   ```
   ist syntaktisch falsch (Fehlermeldung!), denn ein Ausrufezeichen zur Einleitung eines Kommentars darf nur am Zeilenanfang stehen.
   ```

STOP

Unterbrechen eines Programms.

〈VERWENDUNG〉
Zweischenstopp eines Programms, um z.B. Variableninhalte während der Ausführung zu prüfen und ggf. zu äandern.

FORMAT: STOP

(SYNTAX)

STOP ⟶

ERLÄUTERUNG:

1. Die Ausführung des Programms wird angehalten und der Interpreter geht in den Befehlsmodus, was er durch Anzeige der Eingabe-Aufforderung und blinkenden Kursor auf dem Bildschirm signalisiert.
   Durch den Befehl CONT (bzw. Drücken der CONT -Taste) wird die Programmausführung fortgesetzt. Einzelheiten dazu siehe CONT-Befehl auf Seite 82.
2. In diesem Status sind auch manuelle Operationen möglich wie z.B. Ausgabe von Variablen.
3. Bringt man jedoch Änderungen am Programm selbst an, z.B. Löschen, Editieren oder Einfügen von Zeilen, so kann es mit einem CONT-Befehl nicht mehr fortgesetzt werden!
4. Ein Programm darf beliebig viele STOP-Anweisungen enthalten.

ANWENDUNGSBEISPIEL:

```
10 INPUT A,B,C
20 X=5*A-B
30 Y=C/4-8
40 DISP Y
50 STOP
60 Z=X/Y
70 DISP Z
80 END
```

Wenn Y den Wert 0 hat, würde in das Programm in Zeile 60 auf einen Fehler laufen (Division durch 0). Daher wird das Programm davor angehalten, um den Inhalt der im Nenner stehenden Variablen Y zu überprüfen.

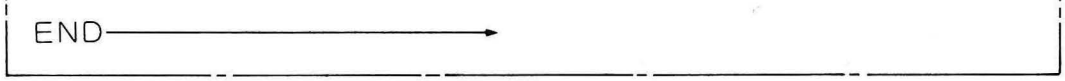
END

Beenden eines Programmlaufes.

〈VERWENDUNG〉
Beenden eines Programmlaufes.

FORMAT: END

(SYNTAX)



ERLÄUTERUNG:

Die Anweisung END beendet die Ausführung eines Programms. Der Interpreter geht danach in den Befehlsmodus.

Jedes Programm muß mit einer END-Anweisung abgeschlossen werden, andernfalls erfolgt eine Fehlermeldung (ERROR 54).

EINFACHES BEISPIEL:

```
100 END
```
..... Die Ausführung des Programms wird in Zeile 100 beendet.

ANWENDUNGSBEISPIEL:

Programm zur Berechnung der Summe aller ganzen Zahlen von 1 bis A.

```
10 INPUT A
20 B=0
30 FOR I=1 TO A
40 B=B+I
50 NEXT I
60 DISP B
70 END
```

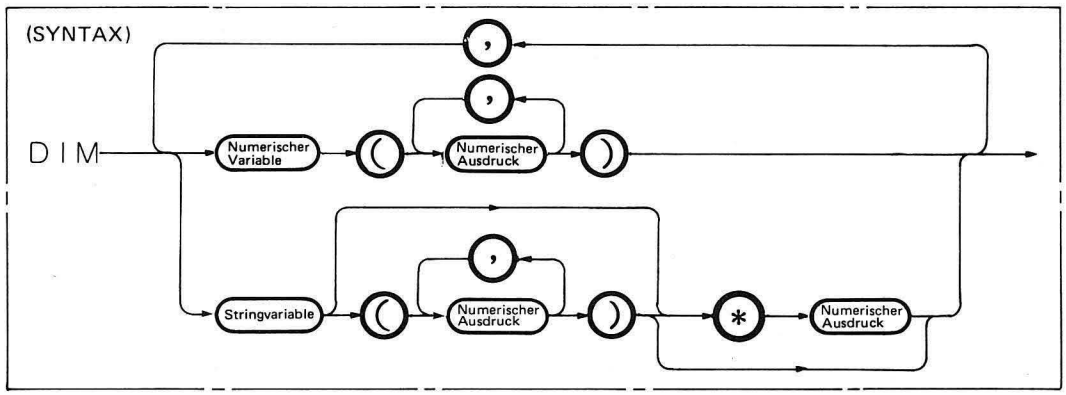
DIM

Deklaration von Array-Variablen und Reservierung des benötigten Speichers.

〈VERWENDUNG〉
Speicherung mehrerer Daten zur selben Zeit unter einem einzigen Variablennamen.

FORMAT: D I M 〔A (〔X〕) | A$ (〔Y〕) 〔*Z〕 〕

A : Name einer numerischen Array-Variablen
X : Anzahl der Elemente des Arrays
A$: Name einer String-Arrayvariablen
Y : Anzahl der Array-Elemente
Z : Größe der Array-Elemente (Anzahl der Zeichen bzw. Bytes pro Array-Element.



ERLÄUTERUNG:

1. Bei der Deklaration einer numerischen Arrayvariablen A(X) bedeutet A den Namen der Variablen und der ganzzahlige Teil des Wertes des numerischen Ausdrucks X (Dezimalen werden abgeschnitten) die Anzahl der Array-Elemente.
   Alle Elemente einer numerischen Arrayvariablen werden vor der DIM-Anweisung mit dem Wert 0 vorbesetzt.
2. Bei der Deklaration einer String-Arrayvariablen A$(Y)*Z steht A für den Namen der Arrayvariablen, der ganzzahlige Teil des Wertes des numerischen Ausdrucks X die Anzahl der Array-Elemente und der des Ausdrucks Z für die Größe eines Array-Elements.
   Ein Zeichen belegt ein Byte im Speicher. Fehlt die Angabe von Z, so werden automatisch 16 eingesetzt, sodaß ein Array-Element bis zu 16 Zeichen aufnehmen kann. Auch bei einfachen Stringvariablen des Typs @ kann die maximale Zahl von Zeichen mit einer DIM-Anweisung vorgegeben werden, siehe einfache Variable auf Seite 11.
   Alle Elemente einer String-Arrayvariablen werden von der DIM-Anweisung mit einem Leerstring (" ") vorbesetzt.

3. Eine Arrayvariable mit einem einzigen numerischen Ausdruck in der Indexklammer nennt man ein "eindimensionales Array". Zwei oder mehrere durch Kommata getrennte numerische Ausdrücke sind möglich, in diesem Fall spricht man von "mehrdimensionalen Arrays". So ist z.B. A(2,3) ein zweidimensionales Array, Q(5,5,4) ein dreidimensionales Array.
   Die Anzahl der Elemente eines mehrdimensionalen Arrays ergibt sich als Produkt aus den Werten der durch Kommata getrennten Indexausdrücke. Das Array A(2,3) hat z.B. 2*3=6, das Array Q(5,5,4) 5*5*4=100 Elemente.
4. Die Elemente eines Arrays werden in aufeinanderfolgenden Speicherplätzen abgelegt, beginnend mit dem Element mit dem kleinsten Index. Bei mehrdimensionalen Arrays läuft dabei der letze Index am schnellsten durch.
   Die Elemente eines mit DIM A(3) deklarierten Arrays werden z.B. in der Folge A(1), A(2), A(3), die des Arrays Z(2,3) in der Folge Z(1,1), Z(1,2), Z(2,1), Z(2,2), Z(3,1), Z(3,2) angeordnet. In genau dieser Reihenfolge werden sie auch von den Anweisungen MAX, MINS und SEARCH abgearbeitet.
5. Siehe auch I.1.(8) "Arrayvariable" auf Seite 19.

EINFACHES BEISPIELE:

1. `10 DIM A(5)` ..... Deklaration einer eindimensionalen numerischen Arrayvariablen mit dem Namen A und 5 Elementen:

A(1)	A(2)	A(3)	A(4)	A(5)

2. `10 DIM B$(6)*30` ..... Deklaration einer eindimensionalen String-Arrayvariablen mit dem Namen B$ und 6 Elementen, deren jedes einen String von bis zu 30 Zeichen Länge aufnehmen kann:

B$(1)	B$(2)	B$(3)	B$($)	B$(5)	B$(6)

3. `10 DIM C$(3,4)` ..... Deklaration einer zweidimensionalen String-Arrayvariablen mit dem Namen C$ und 3*4=12 Elementen mit einem Fassungsvermögen von maximal je 16 Zeichen. Die Elemente des Arrays C$ sind:

C$(1,1)	C$(1,2)	C$(1,3)	C$(1,4)
C$(2,1)	C$(2,2)	C$(2,3)	C$(2,4)
C$(3,1)	C$(3,2)	C$(3,3)	C$(3,4)

4. `10 DIM B$(6)*30,A(5)` ..... Deklaration der Arrays aus den Beispielen 1. und 2. innerhalb derselben DIM-Anweisung.
5. `10 DIM A@*20` ..... Deklaration der einfachen Stringvariablen A@ mit einem Fassungsvermögen von bis zu 20 Zeichen.

WICHTIGER HINWEIS

1. Bei numerischen Array-Variablen unterscheidet man zwischen den Typen INTEGER und REAL, deren Elemente 2 bzw. 8 Bytes im Speicher belegen. Einzelheiten siehe einfache numerische Variable, Seite 12.
2. Eine Arrayvariable darf bis zu 32K Byte Speicher belegen. Wurde z.B. die Elementgröße eines Stringarrays auf 32 Zeichen festgelegt, so darf es maximal 1023 Elemente haben. Der Speicherumfang eines Arrays errechnet sich aus Größe * Anzahl der Elemente + einige Bytes für den Namen etc.
3. Die Größe eines Elements kann auch 255 Byte übersteigen, solange die in Hinweis 2 genannte Bedingung erfüllt ist. Im allgemeinen wird man jedoch im Hinblick auf die Puffergrößen von INPUT und einigen weiteren internen Funktionen nicht über 255 Byte hinausgehen.
   Bei Elementgrößen von 256 Byte und mehr ist die MOVE-Anweisung (siehe Seite 187) zu verwenden.
4. Es ist nicht gestattet, ein per DIM deklariertes Array erneut zu deklarieren, in diesem Fall wird der Fehler ERROR 20 gemeldet. In Programmen mit vielen Arrays ist es besser, alle DIM-Anweisungen am Anfang des Programms zusammenzufassen, um diesen Fehler leichter vermeiden zu können.
5. Alle Elemente eines Arrays können gleichzeitig angesprochen werden, indem man einen Stern statt eines numerischen Ausdrucks in die Indexklammer schreibt, sog. "kollektiver Arrayname".

BEISPIEL: A$ (*)
Alle Elemente des Stringarrays A$ werden gleichzeitig angesprochen.

ANWENDUNGSBEISPIEL:

In vielen Fällen benutzt man Arrays und eine einfache Variable als Index zusammen mit der FOR/NEXT-Anweisung, siehe Seite 149.

Programm A: Quadratzahlen berechnen und in einem Array speichern.

```
10 DIM X(10)          ..... Deklaration der Arrayvariablen X
20 FOR K=1 TO 10      ..... In der FOR/NEXT–Schleife wird J von 1 bis 10 hochgezählt.
30 X(K)=K^2           ..... Zuweisen von K² an das K-te Element von X.
40 NEXT K
50 END
```

Programm B: Daten an der Tastatur eingeben und einem Array zuweisen.

```
10 INPUT "Anzahl der Daten?";I ..... Eingabe der Anzahl der Daten.
20 DIM X(I)            ..... Deklaration eines Arrays mit I Elementen.
30 FOR J=1 TO I        ..... J läuft von 1 bis I in Schritten von 1.
40 DISP "Wer Nummer";J;"?";
50 INPUT X(J)          ..... Eingabe eines Wertes für das J-te Element des Arrays X.
60 NEXT J
70 END
```

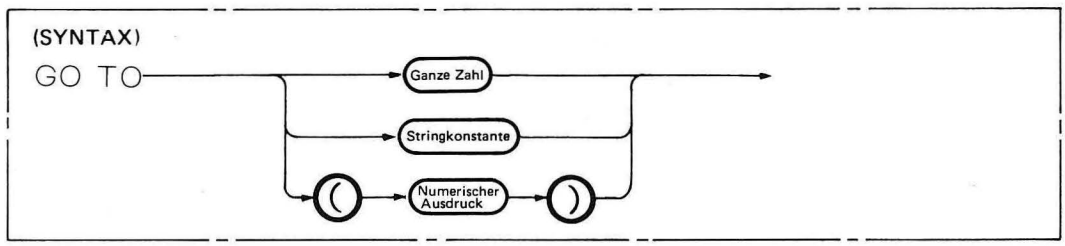
GO TO

Sprung zu einer bestimmten Zeile, an der die Programmausführung fortgesetzt wird.

〈VERWENDUNG〉
① Wiederholung des gleichen Verarbeitungsablaufes während der Programmausführung.
② Änderung des Programmablaufes (abhängig von Bedingungen).

FORMAT: GO TO n | S | (X)

n : Zeilennummer des Sprungziels
S : Zeilenmarke des Sprungziels
X : Zeilennummer des Sprungziels in Form eines numerischen Ausdrucks.



ERLÄUTERUNG:

1. Es erfolgt ein unbedingter Sprung zu der mit der Nummer n, der Zeilenmarke S oder der druch den Wert von X nach dem Abschneiden der Dezimalen bezeichneten Zeile.
2. Folgt die GO TO-Anweisung einer IF-Anweisung (siehe Seite 141), so spricht man von einem bedingten Sprung. Zu den bedingten Sprunganweisungen zählen außerdem noch die Anweisungen ON . . . GO TO, ON ERROR GOTO, ON KEY . . . GO TO und ON TIME . . . GO TO. (Einzelheiten siehe Beschreibungen der einzelnen Anweisungen).
3. Verwendet man STATUS 0 im numerischen Ausdruck für die Zeilennummer, so kann man auch relative Sprünge programmieren, d.h. Sprünge, deren Zeile in einer berechneten Distanz relativ zur Stelle des Absprungs liegt.

EINFACHES BEISPIELE:

```
100 GO TO 200 ..... Unbedingter Sprung zur Zeile mit der Zeilennummer 200.
100 GO TO "ABC" ..... Unbedingter Sprung zur Zeile mit der Zeilenmarke "ABC".
100 GO TO(A*5) ..... Unbedingter Sprung zu der Zeile, deren Nummer sich aus dem Wert des Ausdrucks A*5 errechnet. Sei z.B. Z=100, so erfolgt ein Sprung zur Zeile 500.
100 IF A>0 GO TO 50 ..... Wenn die Bedingung A > 0 erfüllt ist, erfolgt ein Sprung nach Zeile 50. Andernfalls (A ≦ 9) wird die Ausführung in der folgenden Zeile fortgesetzt.
```

ANWENDUNGSBEISPIELE:

Programm A: Berechnung der Summe der ganzen Zahlen von 1 bis 100.

```
10 S=0:N=1 ..... Variable S auf Null setzen, N den Wert 1 zuweisen.
20 S=S+N ..... Inhalt von S und N addieren, nach S zurückspeichern.
30 IF N=100 THEN 60 ...Sprung nach Zeile 60, wenn schon 100 Zahlen summiert sind.
40 N=N+1 ..... N um 1 erhöhen.
50 GO TO 20 .....Sprung zurück zur Summation in Zeile 20.
60 DISP "SUMME=";S .....Ausgabe des Ergebnisses auf dem Bildschirm.
70 END
```

Programm B: Entscheidung, ob eine eingegebene Zahl positiv, Null oder negativ ist.
IF . . . GO TO siehe Seite 141.

```
10 INPUT "GEBEN SIE EINE ZAHL EIN:";N
                ..... Eingabe eines Wertes für die Variable N.
20 IF N<0 GO TO 50 .....Sprung nach Zeile 50, wenn N negativ ist.
30 IF N=0 GO TO 60 .....Sprung nach Zeile 60, wenn N den Wert 0 hat.
40 DISP "POSITIV":END ...Bildschirmausgabe von POSITIV und Programmende.
50 DISP "NEGATIV":END ...Bildschirmausgabe von NEGATIV und Programmende.
60 DISP "NULL":END .....Bildschirmausgabe von NULL und Programmende.
```

Programm C: Auswahl und Ausführung der vier Grundrechenarten.
ON . . . GO TO siehe Seite 138.

```
10 DISP "+ : 1 eingeben"
20 DISP "- : 2 eingeben"
30 DISP "* : 3 eingeben"
40 DISP "/ : 4 eingeben"
50 INPUT N
```

..... Zunächst zeigt das Programm an, welche Schlüsselzahl für welche Rechenart einzugeben ist und wartet in Zeile 50 auf die Eingabe dieses Schlüssels, der der Variablen N zugewiesen wird. Für N=1 springt das Programm nach Zeile 70 für Addition, für N=2 nach Zeile 90 (Subtraktion), für N=3 nach Zeile 110 (Multiplikation) und für N=4 nach Zeile 130 (Division). Gibt man für N eine Zahl kleiner als 1 oder größer als 4 ein, setzt das Programm in der nächsten Zeile (70) fort. In vielen Fällen ist das unerwünscht; das Programm sollte hier durch Abfangen falscher Eingaben verbessert werden.

```
60 ON N GO TO 70,90,110,130
70 INPUT A,B
80 DISP "ERGEBNIS=";A+B:GO TO 150
90 INPUT A,B
100 DISP "ERGEBNIS=";A-B:GO TO 150
110 INPUT A,B
120 DISP "ERGEBNIS=";A*B:GO TO 150
130 INPUT A,B
140 DISP "ERGEBNIS=";A/B
150 END
```

Programm D: Berechnung der Quadratwurzel.
ON ERROR GO TO siehe Seite 172.

```
10 ON ERROR GO TO 100 ... Bei Auftreten eines Fehlers Sprung nach Zeile 100.
20 INPUT "X=";X ..... Bildschirmausgabe X=, Eingabe für die Variable X.
30 IF X<>0:ELSE END ..... Programmende, wenn X den Wert 0 hat.
40 DISP SQR X ..... Quadratwurzel von X berechnen und anzeigen. Bei
                    negativem X tritt eine Fehlerbedingung auf und die
                    Programmausführung wird in Zeile 100 fortgesetzt.
50 GO TO 20 ..... Sprung zurück nach Zeile 20.
100 DISP SQR (-X) ..... Fehlerbehandlung: Aus dem negierten (und damit
                    positiven) Wert von X wird die Wurzel gezogen und
                    angezeigt.
110 GO TO 20 ..... Sprung nach Zeile 20, normale Weiterverarbeitung.
```

Programm E: Gibt nacheinander Zahlen ab 1 aus, bis die Taste A gedrückt wird.
ON KEY . . . GO TO siehe Seite 176.

```
10 ON KEY "A" GO TO 50 ..... Zuordnen eines Sprunges nach Zeile 50 an die
                             Taste "A".
20 N=N+1 ..... Inhalt der numerischen Variable N wird um 1
               erhöht.
30 DISP N; ..... Anzeige von N auf dem Bildschirm.
40 GO TO 20 ..... Sprung nach Zeile 20, weiterzählen.
50 END
```

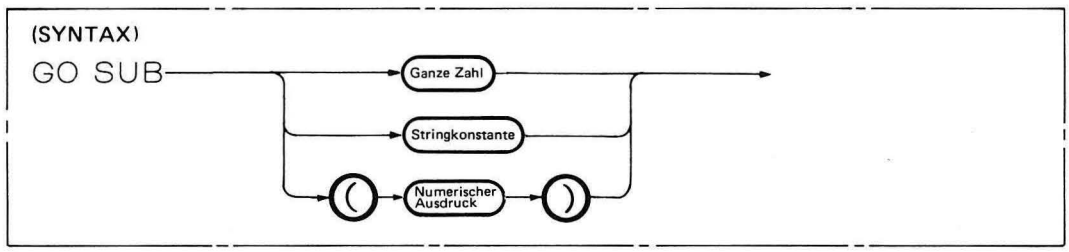
GO SUB

Sprung in ein Unterprogramm.

〈VERWENDUNG〉
Sprung zu einer beliebigen Zeile, die den Beginn eines Unterprogramms bildet.

FORMAT: GO SUB n | S | (X)

n : Zeilennummer des Unterprogramms
S : Zeilenmarke des Unterprogramms
X : Zeilennummer des Unterprogramms in Form eines numerischen Ausdrucks.



ERLÄUTERUNG:

1. Es erfolgt ein unbedingter Sprung in das Unterprogramm, dessen Beginn die mit der Nummer n, der Zeilenmarke S oder der durch den Wert von X nach dem Abschneiden der Dezimalen bezeichnete Zeile bildet.
2. Folgt die GO SUB-Anweisung einer IF-Anweisung (siehe Seite 141), so spricht man von einem bedingten Unterprogrammsprung. Zu den bedingten Unterprogrammsprüngen zählen außerdem noch die Anweisungen ON . . . GO SUB, ON ERROR GO SUB, ON KEY . . . GO SUB und ON TIME . . . GO SUB.
   (Einzelheiten siehe Beschreibungen der einzelnen Anweisungen).
3. Die GO SUB-Anweisung wird in Verbindung mit der RETURN-Anweisung verwendet. Einzelheiten siehe Seite 136.

EINFACHES BEISPIELE:

```
100 GO SUB 200
```
Unbedingter Unterprogrammsprung zur Zeile mit der Zeilennummer 200.

```
100 GO SUB "SUB-1"
```
Unbedingter Unterprogrammsprung zur Zeile mit der Zeilenmarke "SUB-1".

```
100 GO SUB(X*100)
```
Unbedingter Unterprogrammsprung zu der Zeile, deren Nummer sich aus dem Wert des Ausdrucks X*100 errechnet. Sei z.B. X=5, so erfolgt ein Unterprogrammsprung nach Zeile 500.

```
100 IF A=0 GO SUB 50
```
Wenn die Bedingung A = 0 erfüllt ist, erfolgt ein Unterprogrammsprung nach Zeile 50. Andernfalls (A≠0) wird die Ausführung mit der folgenden Zeile fortgesetzt. (Bedingter Unterprogrammaufruf.)

WICHTIGER HINWEIS

1. Die Ausführungsgeschwindigkeit eines Unterprogrammsprunges mittels GO SUB ist höher als bei Verwendung der CALL- und SUB-Anweisungen, die auf den Seiten 157 und 160 beschrieben werden.
2. Ein Unterprogramm kann beliebig oft aufgerufen werden, es darf selbst wieder andere Unterprogramme aufrufen.
3. Das Ende eines Unterprogramms bildet die Anweisung RETURN. Nach einem unbedingten Unterprogrammaufruf erfolgt der Rücksprung auf die unmittelbar hinter dem GOSUB stehende Anweisung. Bei bedingten Unterprogrammsprüngen hängt das Ziel des Rücksprunges jedoch von der Art des Unterprogrammsprunges ab. Nähere Auskunft gibt folgende Tabelle:

Anweisung	Rücksprungziel
IF . . . GO SUB	die auf IF . . . GO SUB folgende Anweisung
ON . . . GO SUB	die auf ON. . . GO SUB folgende Anweisung
ON ERROR GO SUB	Anfang der Zeile, in der der Fehler auftrat
ON KEY . . . GO SUB	die derjenigen Zeile folgende Zeile, die während des Tastendrucks ausgeführt wurde.
ON TIME . . . GO SUB	die derjenigen Zeile folgende Zeile, die zum angegebenen Zeitpunkt gerade ausgeführt wurde.

RETURN

Beenden eines Unterprogramms und Rücksprung an die Stelle hinter dem Aufruf des Unterprogramms.

〈VERWENDUNG〉
Aussprung aus einem Unterprogramm und Fortführen des normalen Verarbeitungsablaufes.

● **FUNKTION 1:** Rücksprung auf die dem GOSUB folgende Anweisung.

FORMAT: RETURN

(SYNTAX)

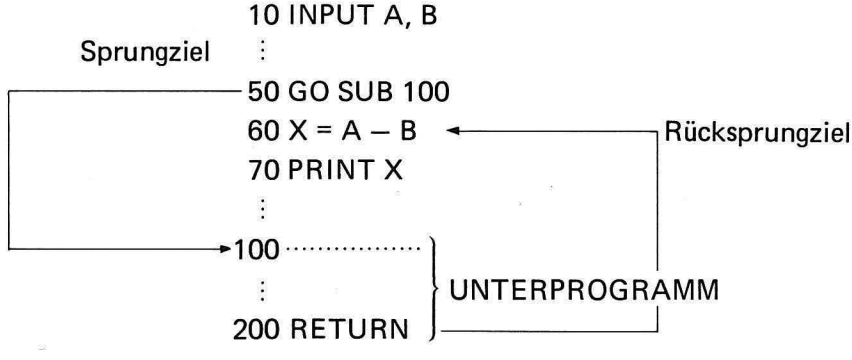
RETURN ——→

ERLÄUTERUNG:

Das aufgerufene Unterprogramm wird beendet und gibt die Programmsteuerung an die Anweisung weiter, die der entsprechenden GOSUB-Anweisung unmittelbar folgt. Diese Stelle ("Rücksprungadresse") wurde beim Aufruf des Unterprogramms automatisch vermerkt.

Eine Fehlermeldung (ERROR 10) erfolgt, wenn vor der RETURN-Anweisung keine GO SUB-Anweisung ausgeführt wurde. In diesem Fall kann der Interpreter keine zu RETURN passende Rücksprungadresse finden.

EINFACHES BEISPIELE:



● **FUNKTION 2:** Rücksprung an den Anfang der Zeile, die hinter der auf das entsprechende GOSUB folgenden Anweisung beginnt.

FORMAT: RETURN NEXT

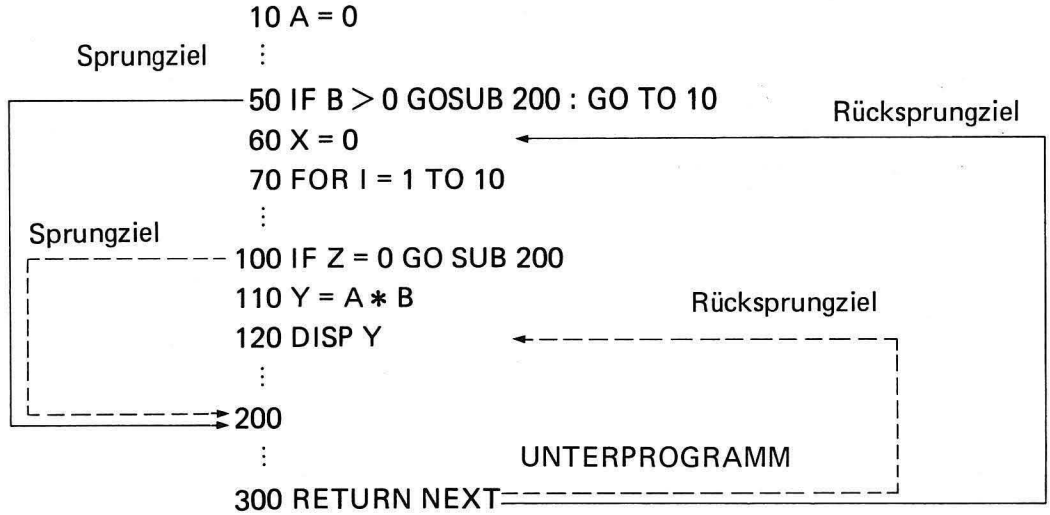
(SYNTAX)

RETURN NEXT ——→

ERLÄUTERUNG:

Das aufgerufene Unterprogramm wird beendet und gibt die Programmsteuerung an den Anfang der Zeile weiter, die hinter der Anweisung beginnt, auf die der Rücksprung in Format 1 erfolgen würde.

EINFACHES BEISPIELE:



Der erste Aufruf des in Zeile 200 beginnenden Unterprogramms erfolgt in Zeile 50, hinter dem GOSUB in Zeile 50 steht die Anweisung GOTO 10. Die nächste Zeile danach ist Zeile 60, so daß in diesem Fall der Fücksprung nach Zeile 60 stattfindet.

Nach dem Zweiten Aufruf des Unterprogramms in Zeile 200 (GOSUB in Zeile 100) führt der Rücksprung in Zeile 120. Der Aufruf steht am Ende der Zeile, die dem Aufruf unmittelbar folgende Anweisung ist Y=A*B in Zeile 110 (das wäre das Rücksprungziel in Format 1 der RETURN-Anweisung), die nächste neue Zeile danach also Zeile 120. Es ist unerheblich, ob die dem GOSUB unmittelbar folgende Anweisung selbst am Anfang einer Zeile steht.

WICHTIGER HINWEIS

Jeder mit GO SUB als Unterprogramm aufgerufene Programmteil sollte nur mittels RETURN und nicht mit GO TO verlassen werden, da andernfalls der Stapelspeicher für die Rückkehradressen überlaufen kann und ein Programmabbruch die Folge wäre (ERROR 10); dasselbe gilt für die Anweisungen CALL/SUBEND (siehe Seite 157 und 162). Im selben Speicherbereich werden außerdem die Schleifenparameter (siehe FOR/NEXT, Seite 149) abgelegt; auch Schleifen sollten wenn möglich nicht mittels GO TO verlassen werden. In diesem Zusammenhang sei auf die POP-Anweisung, Seite 171, verwiesen. Allgemein gilt:

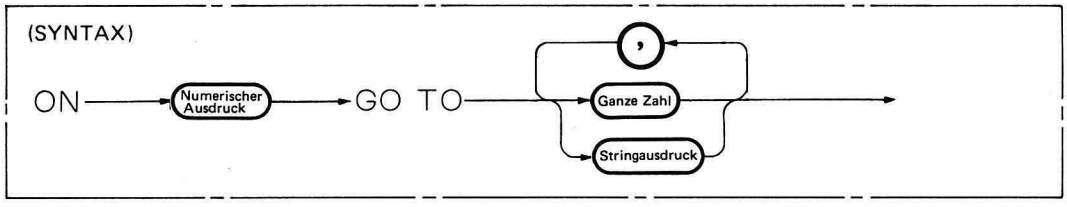
ON···GO TO

Indizierter Sprung oder Verteiler: Gemäß dem Wert eines Index (numerischer Ausdruck) wird das Sprungziel aus einer Liste von mehreren möglichen ausgewählt.

〈VERWENDUNG〉
Zur Programmierung einer Fallunterscheidung mit mehreren Zweigen, wenn die verschiedenen Fälle in je einem bestimmten Programmabschnitt abgehandelt werden.

FORMAT: ON X GO TO {n | S}

n : Zeilennummer eines Sprungziels
S : Zeilenmarke eines Sprungziels



ERLÄUTERUNG:

Hinter dem GO TO folgt eine Liste von Zeilennummern oder Zeilenmarken (durch Kommata getrennt), auf die ein Sprung führen soll. Daraus wird dasjenige Sprungziel ausgewählt, dessen Platz in der Liste durch den ganzzahligen Teil (Dezimalen werden abgeschnitten) des Wertes des numerischen Ausdrucks X bezeichnet wird. Ist X z.B. 1, erfolgt ein Sprung auf das erste der in der Liste aufgeführten Ziele (Zeilennummer oder Zeilenmarke); ist X = 2, wird das zweite Sprungziel angesteuert usw.. Allgemein wird, wenn X den Wert m hat, das m-te Element aus der Liste ausgewählt.

Wenn der Wert von X jedoch kleiner als 1 oder größer als die Anzahl der aufgeführten Sprungziele ist, wird die Ausführung des Programms in der folgenden Zeile fortgesetzt. Die ON . . . GO TO-Anweisung muß also, abgesehen von REM-Anweisungen, die letzte Anweisung in einer Zeile sein, weil dahinter folgende Anweisungen nie erreicht werden können.

EINFACHES BEISPIELE:

```
20 ON A GO TO 50,100,150
```

Wenn der Wert von A gleich 1 ist, erfolgt ein Sprung nach Zeile 50. Für A=2 wird nach Zeile 100, für A=3 nach Zeile 150 gesprungen. Ist der Wert von A nicht 1, 2 oder 3, wird das Programm in der folgenden Zeile weiter ausgeführt.

Ein Anwendungsbeispiel zur ON . . . GO TO-Anweisung ist auf Seite 131 zu finden.

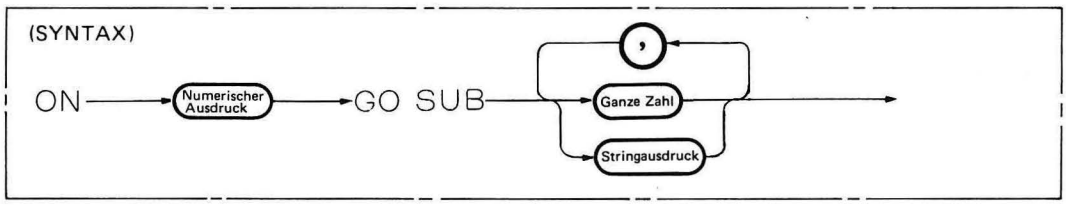
ON···GO SUB

Indizierter Unterprogrammsprung: Gemäß dem Wert eines Index (numerischer Ausdruck) wird ein Unterprogramm aus einer Liste von mehreren möglichen ausgewählt.

〈VERWENDUNG〉
Zur Programmierung einer Fallunterscheidung mit mehreren Zweigen, wenn die verschiedenen Fälle in je einem bestimmten Unterprogramm abgehandelt werden.

FORMAT: ON X GO SUB 〈n | S〉

n : Zeilennummer eines Unterprogramms
S : Zeilenmarke eines Unterprogramms



ERLÄUTERUNG:

1. Hinter dem GO SUB folgt eine Liste von Zeilennummern oder Zeilenmarken (durch Kommata getrennt), die jeweils den Anfang eines Unterprogramms markieren. Daraus wird dasjenige Unterprogramm ausgewählt, dessen Platz in der Listen durch den ganzzahligen Teil (Dezimalen werden abgeschnitten) des Wertes des numerichen Ausdrucks X bezeichnet wird. Ist X z.B. 1, so erfolgt ein Sprung in das erste der in der Liste aufgeführten Unterprogramme (Zeilennummer oder Zeilenmarke); ist X = 2, wird das zweite Unterprogram aufgerufen, usw.. Allgemein wird, wenn X den Wert m hat, das m-te Element aus der Liste ausgewählt.
2. Wenn der Wert von X jedoch kleiner als 1 oder größer als die Anzahl der aufgeführten Unterprogramme ist, wird die Ausführung des Programms in der folgenden Zeile fortgesetzt. Die Rücksprunganweisungen (RETURN) der Unterprogramme führen hingegen zur dem ON . . . GO SUB unmittelbar folgenden Anweisung, kehren also u.U. in dieselbe Zeile zurück.

EINFACHES BEISPIELE:

```
20 ON A GO SUB 50,100,"UP1"
```

..... Wenn der Wert von A gleich 1 ist, wird das in Zeile 50, für A = 2 das in Zeile 100 beginnende Unterprogramm aufgerufen. Hat A den Wert 3, wird das mit der Zeilenmarke "UP1" bezeichnete Unterprogramm aufgerufen. Ist der Wert von A nicht 1, 2 oder 3, wird das Programm in der folgenden Zeile weiter ausgeführt.

ANWENDUNGSBEISPIEL:

Das Programm entscheidet, ob eine eingegebene Zahl negativ, Null oder positiv ist.

```
10 INPUT "Anzahl der Daten?";I
20 DIM X(I)
30 FOR J=1 TO I
40 DISP "Wert Nummer";J;"?";
50 INPUT X(J)
60 NEXT J
70 END
```

Bei negativem A ist SGN(A) = −1, so daß der Wert des gesamten Ausdrucks zweischen ON und GO SUB gleich 1 ist und das erste Unterprogramm aus der Liste (Zeile 300) aufgerufen wird. Für A = 0 ist SGN (A) ebenfalls 0 und der Wert des Ausdrucks gleich 2, daher wird in diesem Fall das zweite Unterprogramm aus der Liste mit der Zeilennummer 100 aufgerufen. Bei positivem A schließlich ist SGN (A) = 1, der Wert des Ausdrucks 3 und es wird das dritte Unterprogramm (Zeile 200) aufgerufen. Näheres zur SGN-Funktion siehe Seite 26.

IF···THEN

Ausführung von Anweisungen oder eines Sprunges, wenn eine vorgegebene Bedingung erfüllt ist.

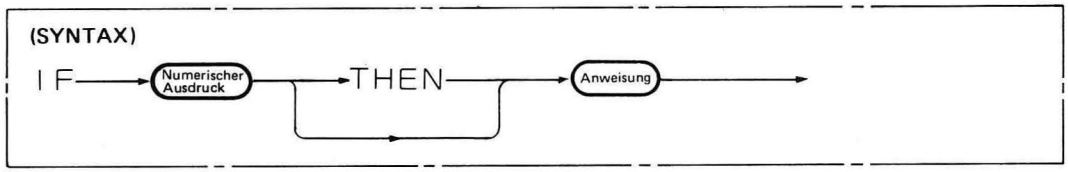
〈VERWENDUNG〉
Selbsttätige Auswertung einer Bedingung durch das Programm und Entscheidung über den weiteren Verarbeitungsablauf (Verzweigung).

● **FUNKTION 1:** Ausführung einer Anweisung.

FORMAT: I F X 〔THEN〕 Anweisung

X : Bedingung (numerischer Ausdruck)

Anweisung : Anweisung



ERLÄUTERUNG:

1. Zunächst wird die Bedingung X (numerischer Ausdruck) ausgewertet. Ist der logische Wert gleich 1 (wahr), so ist die Bedingung erfüllt und die hinter THEN folgende Anweisung wird ausgeführt. Statt einer sind auch mehrere durch Doppelpunkte (:) getrennte Anweisungen gestattet. In diesem Fall wirkt die Bedingung auf alle hinter THEN in derselben Zeile folgende Anweisungen.
2. Bei nicht erfüllter Bedingung wird die Ausführung des Programms in der nächsten Zeile fortgesetzt.
3. THEN ist optional. Folgt jedoch eine Wertzuweisung auf die Bedingung (LET-Anweisung), so muß entweder THEN oder LET hingeschrieben werden.
4. Hinter THEN ist jede beliebige Anweisung zulässig, z.B. auch eine weitere IF . . . THEN-Anweisung.

EINFACHES BEISPIELE:

```
1.50 IF A>100 GO TO 100
```
..... Wenn A größer als 100 ist, so ist die Bedingung erfüllt und die Folgeanweisung, hier GOTO 100, wird ausgeführt. Ist A jedoch gleich oder kleiner als 100, so wird die Ausführung in der folgenden Zeile fortgesetzt.

```
2.50 IF A*B<0 DISP A,B:GO TO 100
```
..... Ist das Produkt A*B kleiner als 0, werden die DISP- und die Sprunganweisung ausgeführt, andernfalls geht es in der nächsten Zeile weiter.

● **FUNKTION 2:** Bedingter Sprung

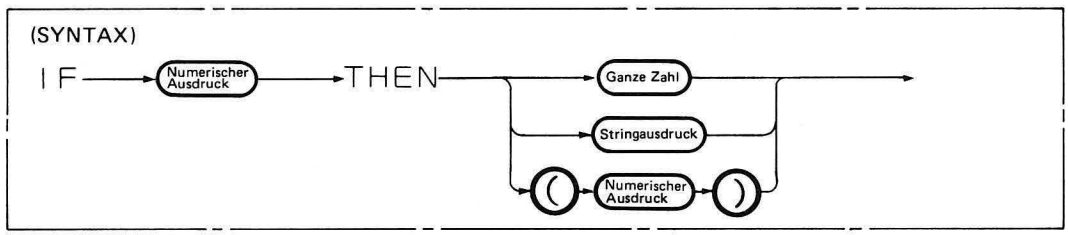
FORMAT: I F X THEN n S (Y)

X : Bedingung (numerischer Ausdruck)

n : Zeilennummer des Sprungziels

S : Zeilenmarke des Sprungziels

Y : numerischer Ausdruck, dessen Wert die Zeilennummer des Sprungziels angibt.



ERLÄUTERUNG:

Wenn die durch den numerischen Ausdruck X gegebene Bedinung erfüllt ist, so erfolgt ein Sprung zu der Zeile, die durch die Zeilenmarke S oder Zeilennummer n (bzw. den ganzzahligen Teil des Wertes von Y) bezeichnet wird.

EINFACHES BEISPIELE:

```
10 IF A>0 THEN 100
```
..... Wenn A größer als 0 ist, erfolgt ein Sprung nach Zeile 100.

```
10 IF A*B=0 THEN "ERROR"
```
..... Wenn A*B gleich 0 ist, erfolgt ein Sprung zu der mit dem Label "FEHLER" bezeichneten Zeile.

ANWENDUNGSBEISPIEL:

Ein Divisionsprogramm soll eine Fehlermeldung ausgeben, wenn der Nenner gleich 0 ist.

```
10 INPUT A,B
20 IF B<>0 THEN C=A/B:GO TO 40
30 DISP "FEHLER: Nenner=0":GO TO 50
40 DISP C
50 END
```

Ist in diesem Beispiel B von 0 verschieden, so wird die Anweisung C=A/B ausgeführt und nach Zeile 40 gesprungen. Ist B gleich 0, wird Zeile 30 ausgeführt.

WICHTIGER HINWEIS

Ist der numerische Ausdruck X zwischen IF und THEN kein Bedingungsausdruck, gilt die Bedingung als erfüllt, wenn der Wert des Ausdrucks von 0 verschieden ist. Für X=0 gilt die Bedingung als nicht erfüllt.

<HILFREICHER TIP>

Es gibt verschiedene Arten von Bedingungsausdrücken, wie Vergleiche oder logische Ausdrücke. In jedem Fall gilt die Bedingung als erfüllt, wenn der Wert des Ausdrucks von 0 verschieden ist, und als nicht erfüllt, wenn der Wert des Ausdruckes gleich 0 ist. Die folgende Aufstellung zeigt einige Beispiele.

BEISPIEL:

Bedingung erfüllt	Bedingung nicht erfüllt
10 > 8	10 < 8
5 = 5	6 = 5
13 AND 11	13 AND 0
6 OR 13	0 OR 0
8 – 2	0

IF···:ELSE

Ausführen einer Anweisung oder eines Sprungs, wenn eine vorgegebene Bedingung nicht erfüllt ist.

〈VERWENDUNG〉
Beschreibung des weiteren Verarbeitungsablaufes, wenn die von IF eingeleitete Bedingung nicht erfüllt ist.

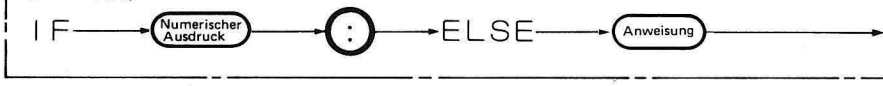
● **FUNKTION 1:** Ausführung einer Anweisung

FORMAT: I F X : E L S E Anweisung

X : Bedingung (numerischer Ausdruck)

Anweisung: Anweisung

(SYNTAX)



ERLÄUTERUNG:

Die durch den numerischen Ausdruck X gegebene Bedingung wird ausgewertet. Ist sie nicht erfüllt, so wird die dem ELSE folgende Anweisung sowie eventuelle weitere durch Doppelpunkte (:) getrennte Anweisungen ausgeführt.

EINFACHES BEISPIELE:

1. `100 IF A>100:ELSE DISP A` ····· Ist A gleich oder kleiner als 100, wird die DISP-Anweisung ausgeführt. Andernfalls wird das Programm in der darauffolgenden Zeile fortgesetzt.

2. `100 IF A$="FD1":ELSE`
   `DISP A$:GO TO 200` ····· Wenn die Stringvariable A$ nicht "FD1" enthält, so wird sie auf dem Bildschirm ausgegeben und nach Zeile 200 gesprungen. II allen anderen Fällen wird die Ausführung in der folgenden Zeile fortgesetzt.

WICHTIGER HINWEIS

Vor ELSE ist ein Doppelpunkt (:) obligatorisch!

● FUNKTION 2: Bedingter Sprung

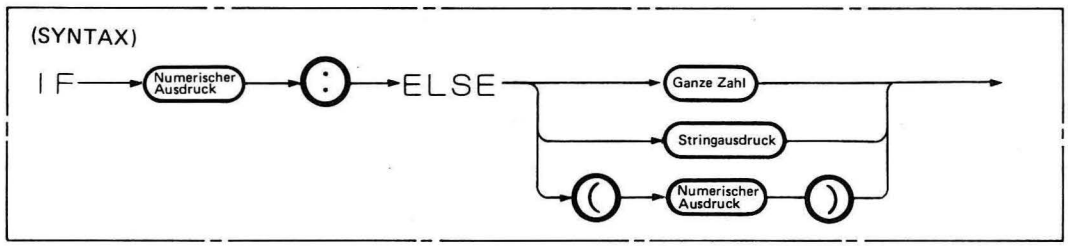
FORMAT: I F X : E L S E n | S | (Y)

X : Bedingung (numerischer Ausdruck)

n : Zeilennummer des Sprungziels

T : Zeilenmarke des Sprungziels

Z : numerischer Ausdruck, dessen Wert die Zeilennummer des Sprungziels angibt.



ERLÄUTERUNG:

Wenn die durch den numerischen Ausdruck X gegebene Bedingung nicht erfüllt ist, so erfolgt eine Sprung zu der Zeile, die durch die Zeilenmarke T oder Zeilennummer n (bzw. den ganzzahligen Teil des Wertes von Z) bezeichnet wird.

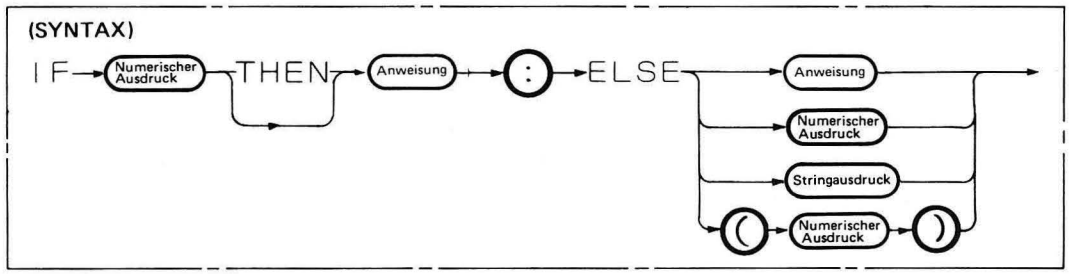
EINFACHES BEISPIELE:

1. `100 IF B<>100:ELSE 200` ..... Wenn B den Wert 100 hat, erfolgt ein Sprung nach Zeile 200.
2. `100 IF A$="ENDE":ELSE "SCHRITT1"` ..... Wenn die Variable A$ einen anderen Inhalt als "ENDE" hat, erfolgt ein Sprung zu der Zeile mit der Zeilenmarke "SCHRITT1".
3. `100 IF A-B=0:ELSE (Y*10)` ..... Ist A ungleich B (A-B ungleich 0), erfolgt ein Sprung zu der durch den Wert von Y*10 bezeichneten Zeile.

● **FUNKTION 3:** Für die erfüllte Bedingung wird eine Anweisung, für die nicht erfüllte Bedingung eine Anweisung oder ein Sprungziel vorgegeben (echte Alternative).

FORMAT: I F X 〔THEN〕 Anweisung : ELSE Anweisung | n | T | (Z)

X	: Bedingung (numerischer Ausdruck)
Anweisung hinter THEN	: Anweisung, die bei erfüllter Bedingung auszuführen ist.
Anweisung hinter ELSE	: Anweisung, die bei nicht erfüllter Bedingung auszuführen ist.
n	: Zeilennummer des Sprungziels
T	: Zeilenmarke des Sprungziels
Z	: numerischer Ausdruck, dessen Wert die Zeilennummer des Sprungziels angibt.



ERLÄUTERUNG:

Ist die durch den numerischen Ausdruck X bezeichnete Bedingung erfüllt, so wird die hinter THEN folgende Anweisung ausgeführt. Hier können mehrere durch Doppelpunkte (:) getrennte Anweisungen stehen, die alle bis zum darauffolgenden : ELSE ausgeführt werden.

Bei nicht erfüllter Bedingung wird der hinter :ELSE folgende Zweig der Anweisung ausgeführt, siehe dazu auch die Seiten 145 und 146.

EINFACHES BEISPIELE:

1. `50 IF A$="FERTIG" THEN END :ELSE 20`

   ..... Ist der Inhalt der Stringvariablen A$ "FERTIG", wird das Programm beendet, andernfalls wird die Verarbeitung in Zeile 20 fortgesetzt.

2. `50 IF A=999 LET Y$="AUS":ELSE "WEITER"`

   ..... Enthält die numerische Variable A den Wert 999, so wird der Stringvariablen Y$ der String "AUS" zugewiesen. Andernfalls erfolgt ein Sprung zu der Zeile mit der Zeilenmarke "WEITER".

WICHTIGER HINWEIS

Folgt auf den Bedingungsteil eine Wertzuweisung an eine Variable (siehe Seite 104), so muß entweder THEN oder LET hingeschrieben werden, ähnlich wie oben in Beispiel 2.

● **FUNKTION 4:** Für die erfüllte Bedingung wird ein Sprungziel, für die nicht erfüllte Bedingung eine Anweisung oder ein Sprungziel vorgegeben (echte Alternative).

FORMAT: I F X THEN m I S I (Y) : E L S E Anweisung I n I T I (Z)

X : Bedingung (numericher Ausdruck)

– Folgende Spezifikationen gelten für erfüllte Bedingung –

m : Zeilennummer des Sprungziels

S : Zeilenmarke des Sprungziels

Y : numerischer Ausdruck, dessen Wert die Zeilennummer des Sprungziels angibt.

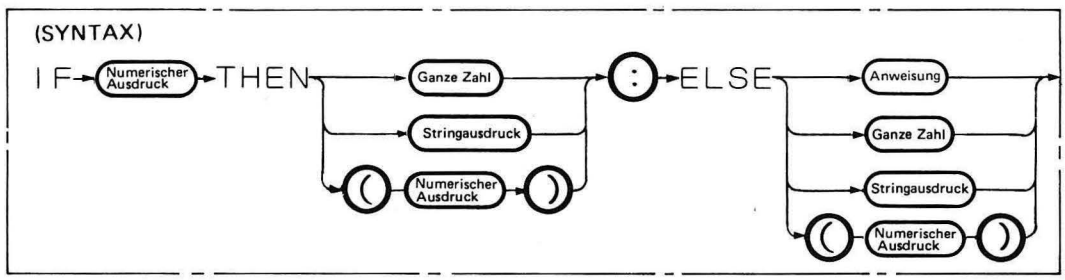
– Folgende Spezifikationen gelten für nicht erfüllte Bedingung –

Anweisung: Bei nicht erfüllter Bedingung auszuführende Anweisung

n : Zeilennummer des Sprungziels

T : Zeilenmake des Sprungziels

Z numerischer Ausdruck, dessen Wert die Zeilennummer des Sprungziels angibt.



ERLÄUTERUNG:

Ist die durch den numerischen Ausdruck X bezeichnete Bedingung erfüllt, erfolgt ein Sprung zu der hinter THEN über ihre Zeilenmarke bzw. durch eine Zahl oder den ganzzahligen Wert des Ausdrucks X gegebenen Zeilennummer bezeichneten Zeile.

Bei nicht erfüllter Bedingung wird der hinter :ELSE folgende Zweig der Anweisung ausgeführt, siehe dazu auch die Seiten 145 und 146.

EINFACHES BEISPIELE:

```
1.50 IF A<>10 THEN 100:ELSE DISP "FEHLER"
```

..... Ist der inhalt der numerischen Variablen A verschieden von 10, erfolgt ein Sprung nach Zeile 100, andernfalls wird die DISP-Anweisung ausgeführt.

ANWENDUNGSBEISPIEL:

```
    :
    :
50 IF Y<>0 THEN Y=0:ELSE 70
60 PRINT A$
70 INPUT M,N
    :
    :
```

In diesem Beispiel wird geprüft, ob die Variable Y den Wert 0 enthält. Ist das nicht der Fall, wird Y der Wert 0 zugewiesen und die Verarbeitung in der folgenden Zeile 60 fortgeführt. Enthielt Y zur Zeit der Abfrage den Wert 0, erfolgt ein Sprung nach Zeile 70.
(Nach der Beendigung dieses Programmzweiges hat Y also auf jeden Fall den Wert 0).

WICHTIGER HINWEIS

1. Ist der numerische Ausdruck X zwischen IF und THEN kein Bedingungsausdruck, gilt die Bedingung als erfüllt, wenn der Wert des Ausdrucks von 0 verschieden ist. Für X=0 gilt die Bedingung als nicht erfüllt. Siehe auch Seite 143.
2. Wenn zwei oder mehrere IF-Anweisungen mit ELSE-Zweig direkt aufeinander folgen, so ist nach den bisher gegebenen Regeln nicht eindeutig, zu welchem IF ein ELSE gehört. In solchen Fällen gilt, daß ein ELSE immer zu dem letzten "offenen" IF zu zählen ist. Ein Beispiel soll das verdeutlichen:

```
10 IF A=0 THEN IF B$="J" THEN DISP "JA":ELSE DISP "NEIN"
```

ELSE gehört hier zum zweiten IF. Sei A von 0 verschieden, so ist die erste Bedingung nicht erfüllt, und der gesamte zweite Teil der Anweisung wird nicht durchlaufen. Man erhält also auch keinerlei Ausgabe auf dem Bildschirm.
Hat aber die Variablen den Inhalt 0, dann ist die erste Bedingung erfüllt, und die zweite Bedingung wird ausgewertet. Je nach dem Inhalt der Stringvariablen B$ erhält man ein JA oder ein NEIN auf dem Bildschirm angezeigt.

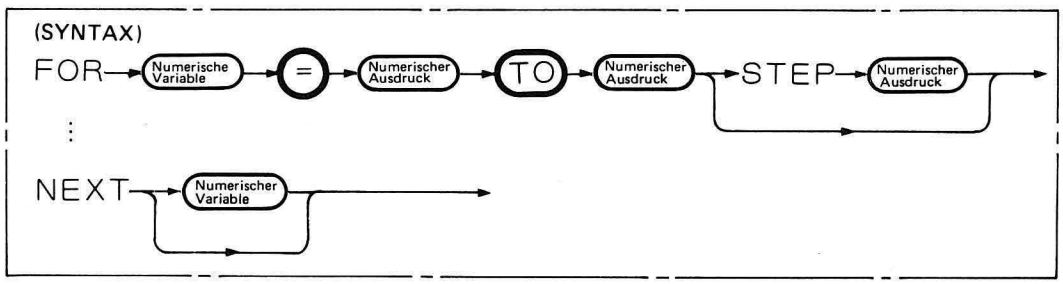
FOR···TO···STEP/NEXT

Schleife zur Wiederholung der zwischen FOR und NEXT stehenden Anweisungen unter vorgegebenen Bedingungen.

〈VERWENDUNG〉
① Mehrfaches Durchlaufen derselben Anweisungen.
② Verarbeitung von Arrayvariablen: Wertzuweisungen, Ein- und Ausgaben.

FORMAT: FOR A=X TO Y (STEP Z)
⋮
NEXT (A)

A : numerische Variable
X : Anfangswert
Y : Endwert numerische Ausdrücke
Z : Inkrement (Schrittgröße)



ERLÄUTERUNG:

1. Der Wert des numerischen Ausdrucks X wird der Variablen A zugewiesen und die zwischen FOR und NEXT stehenden Anweisungen durchlaufen (erster Schleifendurchlauf). Bei Erreichen der NEXT-Anweisung wird das Inkrement Z (numerischer Ausdruck) zum derzeitigen Wert von A addiert und das Ergebnis wieder nach A zurückgespeichert.
   Weiterhin wird der neue Inhalt von X mit dem durch den numerischen Ausdruck Y gegebenen Schleifenendwert verglichen. Solange X noch kleiner oder gleich Y ist, werden alle Anweisungen zwischen der FOR- und der NEXT-Anweisung wiederholt. Die Schleife wird beendet (sie "terminiert"), wenn der Inhalt von A den Endwert Y übersteigt.
   Die Variable A wird in ihrer besonderen Verwendung innerhalb der Schleife auch als "Schleifenvariable" bezeichnet.
2. An das Inkrement Z gilt nur die Forderung, daß es von Null verschieden sein muß; es sind auch nicht ganzzahlige und negative Wert gestattet. Bei positivem Inkrement muß außerdem $Y \geqq X$ erfüllt sein, die Schleife terminiert (wird beendet) bei $A > Y$. Ist Z negativ, so muß $Y \leqq X$ gelten, und die Schleife terminiert bei $A < Y$. Sind diese Bedingungen nicht erfüllt, so ergibt sich eine "unendliche Schleife" oder "Totschleife", die niemals verlassen wird!
3. STEP kann fortgelassen werden, in diesem Fall wird für das Inkrement automatisch 1 eingesetzt.

4. Hinter FOR und hinter NEXT muß die gleiche Variable stehen.
5. Schleifen können verschachtelt sein, d. h. in eine FOR/NEXT-Schleife dürfen weitere FOR/NEXT-Schleifen eingebettet werden.
6. Hinter NEXT darf der Variablenname weggelassen werden.
   Sind mehrere Schleifen ineinander verschachtelt und folgen die zugehörigen NEXT-Anweisungen unmittelbar aufeinander, so kann man auch eine einzige NEXT-Anweisung hinschreiben, die von den Namen der Schleifenvariablen, getrennt durch Kommata, gefolgt wird. Die Schleifenvariablen sind dann in der Reihenfolge der Verschachtelung von "innen" nach "außen" aufzuführen.

EINFACHES BEISPIELE:

1.
```
10 FOR J=1 TO 10 STEP 2
    ⋮
50 NEXT J
```
Die Anweisungen zwischen den Zeilen 10 und 50 werden wiederholt durchlaufen und dabei die Variable J ab 1 um je 2 hochgezählt, bis ihr Inhalt den Wert 10 übersteigt.

2.
```
10 FOR K=5 TO A*B STEP-1
    ⋮
50 NEXT
```
Wiederholung der Anweisungen zwischen den Zeilen 10 und 50. K wird dabei beginnend bei 5 um −1 inkrementiert, bis ihr Inhalt den (ursprünglichen) Wert des Produkts A*B unterschreitet.

3.
```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT K
80 NEXT J
90 NEXT I
```
Schleife 1 } Schleife 2 } Schleife 3

Beispiel für eine Verschachtelung ("Nesting") von Schleifen. Da STEP weggelassen wurde, wird 1 eingesetzt.

(Hier könnten die Variablen bei NEXT fehlen.)

4. BEISPIEL 3 läßt sich auch so schreiben:

```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT K,J,I
```
Schleife 1 } Schleife 2 } Schleife 3 ..... (Hier dürfen die Variablen bei NEXT nicht fehlen, siehe auch Punkt 6. der Erläuterungen !)

```
10 FOR I=1 TO 10
20 FOR J=1 TO 20
30 FOR K=1 TO 30
    ⋮
70 NEXT:NEXT:NEXT
```
Schleife 1 } Schleife 2 } Schleife 3

WICHTIGER HINWEIS

Die auch als "Schleifenparameter" bezeichneten Größen X, Y und Z werden nur ein einziges Mal vor Beginn der Schleife berechnet. Wenn z.B. in den numerischen Ausdrücken Variable verwendet und deren Werte innerhalb der Schleife verändert werden, so hat das keinen Einfluß mehr auf die Abarbeitung der Schleife.

ANWENDUNGSBEISPIEL:

Programm A: Ausdrucken der Werte der Schleifenvaraiablen J aus dem einfachen Beispiel 1.

```
10 FOR J=1 TO 10 STEP 2
20 PRINT J
30 NEXT J
40 PRINT J
50 END
```

J wird beginnend mit 1 in Schritten von 2 hochgezählt, bis der Inhalt von J größer als 10 ist.

Ergebnisse:

```
1
3
5
7
9
11
```

der Anfangswert von J, erster Schleifendurchlauf
J während des zweiten Schleifendurchlaufs
J während des dritten Schleifendurchlaufs
J während des vierten Schleifendurchlaufs
J während des fünften Schleifendurchlaufs
J nach Beendigung der Schleife.

Programm B: Eingabe von fünf Daten und Berechnung von Summe, Mittelwert und Maximum. Das Maximum könnte auch mit Hlife der MAX-Anweisung (siehe Seite 191) gefunden werden.

```
 10 DIM A(5)
 20 B=0:C=0
 30 FOR J=1 TO 5
 40 PRINT "Messwert ";J;" =";
 50 INPUT A(J)
 60 PRINT A(J)
 70 IF B<A(J) THEN B=A(J)
 80 C=C+A(J)
 90 NEXT J
100 D=C/5
110 PRINT "****************"
120 PRINT "Summe       =";C
130 PRINT "Maximum     =";B
140 PRINT "Mittelwert  =";D
150 END
```

Deklaration der Arrayvariablen A.
Maximum (B) und Summationsvariable (C) initialisieren.
Schleife
derzeitiges Maximum an Variable B zuweisen.
Bildung der Summe in der Variablen C.
Mittelwert berechnen und D zuweisen.

Ergebnisse:

```
Messwert  1 = 60
Messwert  2 = 75
Messwert  3 = 46
Messwert  4 = 67
Messwert  5 = 32
****************
Summe       = 280
Maximum     = 75
Mittelwert = 56
```

Programm C: Zweifache Schleifenverschachtelung

```
10 FOR I=1 TO 5
20 FOR J=1 TO 3
30 PRINT "I=";I,"J=";J
40 NEXT J,I
50 PRINT "I=";I,"J=";J
60 END
```

Schleife 1 (Zeilen 20–40) — Schleife 2 (Zeilen 10–40)

Ergebnisse:

```
I=  1          J=  1
I=  1          J=  2
I=  1          J=  3
I=  2          J=  1
I=  2          J=  2
I=  2          J=  3
I=  3          J=  1
I=  3          J=  2
I=  3          J=  3
I=  4          J=  1
I=  4          J=  2
I=  4          J=  3
I=  5          J=  1
I=  5          J=  2
I=  5          J=  3
I=  6          J=  4
```

J= 1, J= 2, J= 3 — Schleife 1

J= 1, J= 2, J= 3 — Schleife 1

J= 1, J= 2, J= 3 — Schleife 1

J= 1, J= 2, J= 3 — Schleife 1

Schleife 2

J= 4 ····· I und J nach der Schleife

Beide Schleifenvariablen beginnen mit dem Wert 1. Beim ersten Erreichen von Zeile 40 wird J um 1 auf zwei erhöht, und die innere Schleife wird wieder ab Zeile 20 begonnen. Der Inhalt von I bleibt vorläufig unberührt.
Nach dem dritten Durchlauf der inneren Schleife wird J in Zeile 40 schließlich auf 4 erhöht und ist größer als der Endwert der inneren Schleife, so daß diese damit beendet wird. Dann wird I um 1 erhöht und die äußere Schleife beginnt wieder ab Zeile 10.
Weil die J-Schleife in die I-Schleife verschachtelt ist, wird sie bei jedem Durchlauf durch die I-Schleife erneut vollständig abgearbeitet, denn jeder Durchauf der I-Schleife führt über die Eröffnung der J-Schleife in Zeile 20.
Beide Schleifen werden schließlich beendet, wenn die Werte von I und J 6 bzw. 4 und damit größer als die entsprechenden Endwerte (Schleifenparameter) sind.

Programm D: Drucken einer Multiplikationstabelle.

```
 10 DIM A(9,9) ..... Array deklarieren
 20 PRINT "MULTIPLIKATIONSTABELLE"
 30 FOR L=1 TO 9                          } Schleife 1
 40 PRINT TAB (4*L);L;" )";               }
 50 NEXT L
 60 PRINT ..... Zeilenvorschub            } Schleife 3
 70 FOR M=1 TO 9                          }
 80 PRINT M;" )";                         }
 90 FOR N=1 TO 9                          } Schleife 2
100 A(M,N)=M*N                            }
110 PRINT TAB (4*N);A(M,N)                }
120 NEXT N                                }
130 PRINT ..... Zeilenvorschub            }
140 NEXT M                                }
150 END
```

ANMERKUNG:

In allen bisherigen Beispielen zur Verwendung von FOR/NEXT-Schleifen wurden die Ergebnisse auf dem Drucker ausgedruckt. Selbstverständlich könnte man durch einfaches Ersetzen der PRINT- durch DISP-Anweisungen auch eine Bildschirmausgabe erzielen.

Ergebnisse:

```
MULTIPLICATION TABLE
     1  2  3  4  5  6  7  8  9
 1   1  2  3  4  5  6  7  8  9
 2   2  4  6  8  10 12 14 16 18
 3   3  6  9  12 15 18 21 24 27
 4   4  8  12 16 20 24 28 32 36
 5   5  10 15 20 25 30 35 40 45
 6   6  12 18 24 30 36 42 48 54
 7   7  14 21 28 35 42 49 56 63
 8   8  16 24 32 40 48 56 64 72
 9   9  18 27 36 45 54 63 72 81
```

WICHTIGER HINWEIS

1. Es ist zwar gestattet, mit einer GO TO-Anweisung aus einer Schleife herauszuspringen, aber man darf nicht von außen in eine Schleife hineinspringen! Der einzige legale Eintritt in eine Schleife führt über die FOR-Anweisung am Schleifenanfang! Man darf jedoch nach dem Aussprung aus einer Schleife (vor dem Durchlaufen der NEXT-Anweisung) wieder in dieselbe Schleife zurückspringen. Die Werte der Schleifenvariablen dürfen außerhalb der Schleifen nicht geändert werden. (Solange man sich nicht innerhalb einer Schleife befindet, d.h. keine FOR-Anweisung durchlaufen hat, stehen die in den Schleifen benutzten Variablen natürlich frei zur Verfügung!)
2. Aufrufe von Unterprogrammen mittels GOSUB zählen in dieser Weise nicht als Aussprung aus der Schleife, da die RETURN-Anweisung am Ende eines Unterprogramms automatisch wieder an die richtige Stelle zurückkehrt.
3. Die mögliche Schachteltiefe bei geschachtelten Schleifen hängt zusammen mit der Schachteltiefe aufgerufener Unterprogramme (siehe auch GOSUB, Seite 134). Steht weder ein Unterprogrammaufruf in der Schleife noch werden Schleifen innerhalb eines Unterprogramms ausgeführt, so können bis zu 14 Schleifen mit Schleifenvariablen vom Typ REAL geschachtelt werden. Sind die Schleifenvariablen vom Typ INTEGER, erhöht sich die zulässige Schachteltiefe bis auf etwa den doppelten Wert. Bei Überschreitung der Höchstgrenze erfolgt eine Fehlermeldung (ERROR 14).
4. Innerhalb von geschachtelten Schleifen kann eine Variable nur einmal als Schleifenvariable verwendet werden; jede Schleife braucht ihre individuelle Schleifenvariable! Außerdem dürfen sich Schleifen nicht überlappen. Innere Schleifen müssen vollständig in äußere Schleifen eingebettet sein!

Die folgenden Beispiele zeigen fehlerhaft aufgebaute Schleifen.

```
 10 FOR I=1 TO 10
  :                    }
  :                    }  Schleife 1
 50 FOR J=2 TO 5       }
  :                    }
  :                    }  Schleife 2
100 NEXT I            }
110 PRINT A           }
120 NEXT J            }
```

Dies ist eine Beispiel für falsch verschachtelte Schleifen.

```
 10 FOR I=5 TO 10
 20 A=A+I
 30 PRINT A
 40 NEXT
  :
  :
100 GO TO 20
```

Schleife

Ohne Durchlaufen der FOR-Anweisung darf nicht in die Schleife eingetreten werden, siehe Hinweis 1! Folgende Konstruktion ist hingegen erlaubt, jedoch auch ein Beispiel für schlechte Programmiertechnik:

```
 10 DIM A(5)
 20 Y=0
 30 FOR J=1 TO 5
 40 INPUT A(J)
 50 IF A(J)>100 GO TO 90
 60 Y=Y+A(J)
 70 NEXT J
 80 DISP Y:END
 90 DISP A(J);"ist zu gross!"
100 GO TO 40
```

Schleife

Vor der NEXT-Anweisung führt hier ein Sprung nach Zeile 90 außerhalb der Schleife (jederzeit möglich und auch korrekt). Der Sprung (Zeile 100) "von der Seite her" in die Schleife hinein ist deshalb gestattet, weil er wieder in dieselbe Schleife zurückführt und der Wert der Schleifenvariablen J nicht verändert wurde.
In diesem Fall läßt sich die Behandlung zu großer Zahlenwerte vollständig in die Schleife einbetten, eine viel klarere und bessere Lösung:

```
  :
40 INPUT A(J)
50 IF A(J)>100 THEN DISP A(J);"ist zu gross!":GO TO 40
```

Sprung verläßt die Schleife nicht!

Zeile 90 und 100 entfallen. Die Stärke dieser Lösung wird schnell klar, wenn man sich vorstellt, daß die Fehlerbehandlung statt in Zeile 90 und 100 am Ende eines langen Programms erfolgt und nun gerade dieser Teil geändert werden soll.

4. SONDERANWEISUNGEN

Die Sonderanweisungen stellen eine Erweiterung gegenüber dem Standardsprachumfang der Programmiersprache BASIC dar. Sie ermöglichen eine effektivere Nutzung dieses System.

CALL
SUB
SUBEND
KEY
KEY IN
POP
ON ERROR
OFF ERROR
ON KEY
OFF KEY
ON TIME
OFF TIME
RESUME
MOVE ※
MAXS ※
MINS ※
SEARCH ※
DEFFN
MUSIC ※
WAIT ※
DEFINT ※
DEFREAL ※
DEFSTR ※
DEG ※
RAD ※
RANDOM ※
ERROR MUSIC ※

Die mit ※ bezeichneten Anweisungen können auch in der manuellen Betriebsart benutzt werden.

CALL

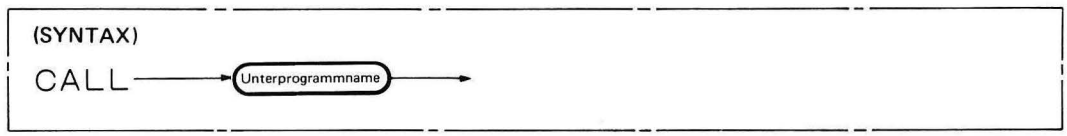
Ruft ein Unterprogramm*) auf.

〈VERWENDUNG〉
Zum Aufrufen eines Programmteils, der als in sich abgeschlossenes Programm funktioniert (Unterprogramm).

● **FUNKTION 1:** Aufruf eines Unterprogramms ohne Übergabe von Argumenten.

FORMAT: CALL SF

SF : Unterprogrammname



ERLÄUTERUNG:

Die CALL-Anweisung ruft das Unterprogramm SF auf.

EINFACHES BEISPIEL:

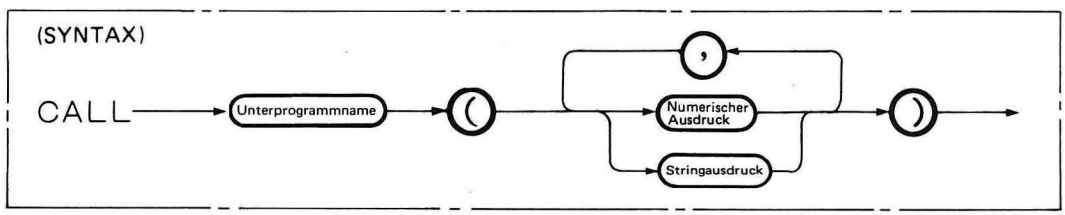
```
10 CALL PROGRAM
```
..... Ruft ein Unterprogramm namens PROGRAM auf.

● **FUNKTION 2:** Aufruf eines Unterprogramms mit Übergabe von Argumenten.

FORMAT: CALL SF (|X|C|)

SF : Name des Unterprogramms

X, C : Argument(e)



ERLÄUTERUNG:

1. Die CALL-Anweisung ruft das Unterprogramm mit dem Namen SF auf und übergibt ein oder mehrere Argumente X bzw. C. Die Argumente müssen zwischen runden Klammern stehen.
2. Handelt es sich bei Argument X bzw. C um eine Variable, deren Wert nach der Rückkehr aus dem Unterprogramm nicht verändert sein soll, dann muß diese bei der Übergabe in eine zusätzliche Klammer gesetzt werden: (X) oder (Y).
3. Ist X oder C eine Array-Variable, die kollektiv übergeben werden soll, muß dies durch X(*) bzw. C(*) gekennzeichnet sein.

EINFACHES BEISPIEL:

```
10 CALL KALK(A,B,C$)
```
..... Ruft das Unterprogramm KALK mit den Argumenten A, B, und C$ auf

*) Unterprogramm: Ein Programm, das in einem großen Programm als selbständige Einheit besteht. Das Programm, das ein Unterprogramm aufruft, heißt Hauptprogramm.

ANWENDUNGSBEISPIEL:

1. PROGRAMM A

```
 10 A=20
 20 B=32
 30 C=99
 40 D=99
 50 CALL KALK(A*2,B,C,D)
 60 DISP "SUMME C=";C
 70 DISP "DIFFERENZ D=";D
 80 END
 90 REM UNTERPROGRAMM
100 SUB KALK(X,Y,Z1,Z2)
110 Z1=X+Y
120 Z2=X-Y
130 SUBEND
```

Variablenwerte vor dem Unterprogrammaufrut

HAUPTPROGRAMM (Programm, welches das Unterprogramm aufruft.)

UNTERPROGRAMM

Einige ergänzende Erläuterungen zum Verständnis des Programmablaufs:

Wenn die Programmausführung Zeile 50 erreicht hat, kommt die CALL-Anweisung zur Ausführung. Sie ruft das Unterprogramm KALK auf, das in den Zeilen 100 bis 130 steht. Die Werte der Variablen des Hauptprogramms werden entsprechend der Spezifizierung im Unterprogrammaufruf in Zeile 50 an das Unterprogramm KALK übergeben:

```
50 CALL KALK(A*2,B,C,D)
```



```
100 SUB KALK(X,Y,Z1,Z2)
```

Ist das Unterprogramm abgearbeitet, dann werden die Werte von X,Y,Z1 und Z2 wieder an die entsprechenden Variablen A*2, B, C, und D, wie sie im Unterprogrammaufruf standen, (zurück) übergeben.

Die Werte von A*2,B,C und D, die vom Hauptprogramm an das Unterprogramm übergeben worden sind, sowie die Werte von X,Y, Z1 und Z2, die vom Unterprogramm an das Hauptprogramm zurückgegeben wurden, heißen Argumente. Argumente werden zum Informationsaustausch zwischen einem Hautprogramm und dem von ihm mittels CALL aufgerufenen Unterprogramm benutzt.

Anzeige nach der Ausführung von Programm A:

```
>RUN
SUMME C= 72
DIFFERENZ D= 8
READY
>
```

Die Werte der Variablen C und D werden geandert, wenn das Unterprogramm aufgerufen wird.

Soll der Wert der Variablen D nach der Rückkehr aus dem Unterprogramm unverändert 99 betragen, müßte man Zeile 50 wie folgt eingeben:

```
50 CALL KALK(A*2,B,C,(D))
```

In diesem Fall wäre die Anzeige nach der Programmausführung:

```
>RUN
SUMME C=72
DIFFERENZ D= 99
READY
```

} Der Wert der Variablen D ist gleich geblieben.

2. PROGRAMM B

Dieses Programm benutzt als Argumente numerische Konstanten.
Das Ergebnis ist jedoch gleich wie bei Programm A.

```
 30 C=99
 40 D=99
 50 CALL KALK(20*2,32,C,D)
 60 DISP "SUMME C=";C
 70 DISP "DIFFERENZ D=";D
 80 END
 90 REM UNTERPROGRAMM
100 SUB KALK(X,Y,Z1,Z2)
110 Z1=X+Y
120 Z2=X-Y
130 SUBEND
```

..... Als Argumente werden numerische Konstanten benutzt.

Anzeige nach der Ausführung von Programm B:

```
>RUN
SUMME C= 72
DIFFERENZ D= 8
READY
```

SUB

Kennzeichnet ein Unterprogramm.

〈VERWENDUNG〉
① Wenn gleiche Prozeduren in einem Programm mehrmals benötigt werden.
② Wenn ein besonders großes Programm zu Verbesserung der Übersichtlichkeit in mehrere, voneinander unabhängige Teilprogramme (Blöcke) zerlegt werden soll.

● **FUNKTION 1:** Unterprogramm ohne Übergabe von Argumenten.

FORMAT: SUB SF

SF : Unterprogrammname



ERLÄUTERUNG:

SUB SF zeigt an, daß in dieser Zeile ein Unterprogramm mit dem Namen SF beginnt.

EINFACHES BEISPIELE:

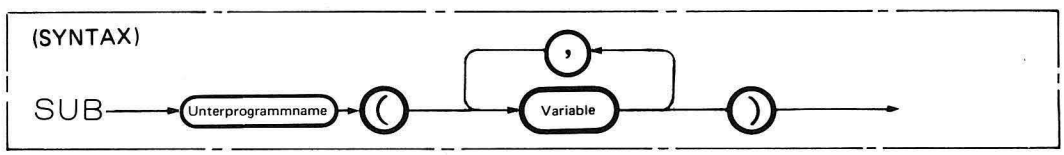
```
100 SUB TOTAL
```
..... Zeigt an, daß in Zeile 100 ein Unterprogramm namens TOTAL beginnt.

● **FUNKTION 2:** Unterprogramm mit Übergabe von Argumenten.

FORMAT: SUB SF (N)

SF : Unterprogrammname

N : Argument



ERLÄUTERUNG:

SUB SF (N) zeigt an, daß in dieser Zeile ein Unterprogramm mit dem Namen SF beginnt. Bei Aufruf dieses Unterprogramms ist für N ein Argument zu übergeben.

EINFACHES BEISPIELE:

```
100 SUB KALK(A,B(*),C$)
```
..... Zeigt an, daß KALK ein Unterprogramm mit folgenden Argumenten ist: einer numerischen Variablen A, einer numerischen Arrayvariablen B(*) und einer Stringvariablen C$.

Ein Anwendungsbeispiel findet sich auf Seite 158 bei der Behandlung der CALL-Anweisung.

WICHTIGER HINWEIS

1. Die BASIC-Befehle VRUN und ERASE können innerhalb eines Unterprogramms weder in der manuellen Betriebsart (während eines Unterprogrammstops) noch während des Programmablaufs benutzt werden. Zuvor müßte eine SUBEND-Anweisung ausgeführt worden sein.
2. Der Beginn eines Unterprogramms sollte mit einer Zeilennummer erfolgen, die größer ist als die größte Zeilennummer des Hauptprogramms.

<PRAKTISCHER TIP>

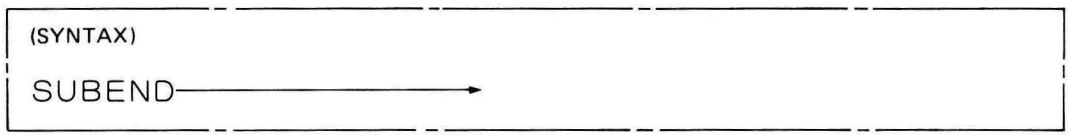
① In einem Unterprogramm, das durch CALL oder SUB spezifiziert ist, sind alle verwendeten Variablen unabhängig von den Variablen des Hauptprogramms oder eines anderen Unterprogramms, d.h. sie sind "lokal". Man kann daher ein Unterprogramm konzipieren, ohne hierbei auf die Variablen des Hauptprogramms Rücksicht nehmen zu müssen. Dies erleichtert ungemein die Erstellung einer Bibliothek von Unterprogrammen, die dann bei Bedarf in jedes beliebige Hauptprogramm übernommen werden können.
Die Variablendeklaration des Hauptprogramm hat für das Unterprogramm keine Gültigkeit; sie muß vielmehr entsprechend der Ausführungen auf Seite 13 innerhalb des Unterprogramms selbst, d.h. also gesondert, erfolgen.

② Die mittels der DEFFN-Anweisung im Hauptprogramm definierten Funktionen haben innerhalb eines Unterprogramms keine Gültigkeit; ebensowenig können die Funktionen des Unterprogramms im Hauptprogramm aufgerufen werden. Die Funktionen des Hauptprogramms und des Unterprogramms sind somit voneinander unabhängig.

SUBEND

Zeigt das Ende eines Unterprogramms an.

FORMAT: SUBEND



ERLÄUTERUNG:

Die Anweisung SUBEND schließt ein Unterprogramm ab und übergibt die Programmsteuerung vom Unterprogramm wieder an das Hauptprogramm. Als nächstes wird dann die dem Unterprogrammaufruf (CALL-Anweisung) folgende Anweisung abgearbeitet.

EINFACHES BEISPIEL:

```
 :
 50 CALL KALK(A,B,C)
 60 DISP "SUMME =";C
 :
100 SUB KALK(X,Y,Z)
110 Z=X+Y
120 SUBEND
```

HAUPTPROGRAMM

UNTERPROGRAMM

In diesem Beispiel ruft die CALL-Anweisung in Zeile 50 das Unterprogramm KALK auf. Darauf werden die Zeilen 100 bis 120 ausgeführt, dann wird die Kontrolle wieder an das Hauptprogramm zurückübergeben und Zeile 60 wird abgearbeitet.

KEY

Liest ein Zeichen von der Tastatur.

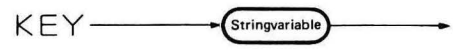
〈VERWENDUNG〉
Zur Eingabe eines Zeichens von der Tastatur während des Programmlaufs.

● **FUNKTION 1:** Ein über die Tastatur eingegebenes Zeichen wird einer Stringvariablen zugewiesen.

FORMAT: KEY A$

A$: Stringvariable, der die Tastatureingabe zugewiesen werden soll

(SYNTAX)



ERLÄUTERUNG:

1. Ein über die Tastatur eingegebenes Zeichen wird die Stringvariable A$ zugewiesen.
2. Im Gegensatz zur INPUT-Anweisung (Seite 117) wird die Programmausführung mit der nächsten Programmzeile fortgesetzt, wenn keine Eingabe stattgefunden hat. Auch wird ein eingegebenes Zeichen nicht auf dem Bildschirm angezeigt.

EINFACHES BEISPIEL:

```
10 KEY C$
```
..... Das über die Tastatur eingegebene Zeichen wird der Stringvariablen C$ zugewiesen.

● **FUNKTION 2:** Löscht alle bischerigen Tastatureingaben.

FORMAT: KEY

(SYNTAX)



ERLÄUTERUNG:

1. Löscht den Tastatureingabepuffer; bisherige Eingaben werden ungültig.
2. Wird zum Löschen bisheriger Eingaben verwendet, um den Puffer für neue, gültige Eingaben freizumachen.

WICHTIGER HINWEIS

1. Wird die Anweisung KEY A$ ausgeführt, wenn der Tastaturpuffer leer ist, bleibt der bisherige Inhalt der Variablen A$, z.B. die letzte Eingabe, unverändert.
2. Anweisungen, die nach KEY A$ in derselben Befehlszeile folgen, werden nur ausgeführt, wenn ein Zeichen von der Tastatur zugewiesen werden konnte.

ANWENDUNGSBEISPIEL:

Ein eingegebenes Zeichen soll an der Kursorposition (0,0) angezeigt werden. Drückt man die Taste ⟨ E ⟩, erscheint die Anzeige "ENDE" und das Programm wird beendet.

```
 10 DISP CLEAR ..... Löscht den Bildschirm
 20 KEY A1$:IF A1$="E" THEN 90:ELSE 40 ..... Beendet das Programm bei Eingabe von "E".
 30 CURSOR 0,0:DISP " ":GO TO 20 ..... Sprung zu Zeile 20, wenn keine Eingabe stattfand.
 40 KEY ..... Löscht den Eingabepuffer
 50 KEY A2$:IF A2$="E" THEN 90:ELSE 70 ..... Beendet das Programm bei Eingabe von "E".
 60 GO TO 20 ..... Sprung zu Zeile 20, wenn keine Eingabe stattfand.
 70 IF A1$=A2$:ELSE 20
 80 CURSOR 0,0:DISP A1$:GO TO 40 ..... Anzeigen der gedrückten Taste.
 90 CURSOR 0,0:DISP "ENDE"
100 END ..... Zeigt "ENDE" an.
```

KEYIN

Eingabe numerischer Daten oder Stringdaten über die Tastatur.

〈VERWENDUNG〉
Zur rechtsbündigen Eingabe von Zahlen und zur linksbündigen Eingabe von Stringdaten über die Tastatur. Die Länge der Eingabe ist festgelegt und verhindert Fehleingaben, z.B. bei Maskeneingaben.

● **FUNKTION 1:** Eingabe numerischer Daten

FORMAT: KEY I N A,X,Y,C

A: Numerische Variable

X: Gesamtzahl der einzugebenden Stellen

Y: Gesamtzahl der Dezimalstellen

C: Anzeigeformat

"N" ..... Führende Nullen werden angezeigt. Das Vorzeichen wird nicht angezeigt.

"L" ..... Ein negatives Vorzeichen wird vor der Zahl angezeigt.
Ein Plus-Zeichen wird durch das Leerzeichen ersetzt.

"R" ..... Ein negatives Vorzeichen wird nach der Zahl angezeigt.
Das positive Vorzeichen wird durch das Leerzeichen ersetzt.

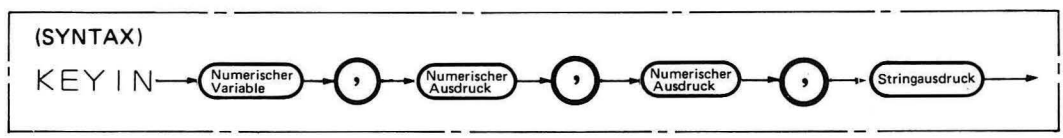
"F" ..... Das negative Vorzeichen wird vor der Zahl angezeigt (Gleitkommadarstellung). Ist die Zahl positiv, wird die Vorzeichenposition zur Zahlendarstellung verwendet.

"L," ..... Fügt nach jeweils drei Stellen ein Komma zum Positionieren ein. Die Darstellung des Vorzeichens entspricht "L".

"R," ..... Fügt nach jeweils drei Stellen ein Komma zum Positionieren ein. Die Darstellung des Vorzeichens entspricht "R".

"F," ..... Fügt nach jeweils drei Stellen ein Komma zum Positionieren ein. Die Darstellung des Vorzeichens entspricht "F".

(SYNTAX)



ERLÄUTERUNG:

1. Eine numerische Eingabe wird der Variablen A zugewiesen. Die Darstellung erfolgt mit dem durch die Stringvariable C festgelegten Format und einer Länge von X Zeichen links von der aktuellen Kursorposition. Zur Eingabe von ganzzahlingen Werten ist Y auf 0 zu setzen.

2. Folgende Tabelle enthält eine Übersicht aller verwendbaren Formate:

String-konstante C	Nullunter-drückung	Positionie-rung	Vorzeichen
"N"	nein	nein	nein
"L"	ja	nein	links
"R"	ja	nein	rechts
"F"	ja	nein	gleitend nur negativ
"L,"	ja	ja	links
"R,"	ja	ja	rechts
"F,"	ja	ja	gleitend nur negativ

An stelle des Kommazeichens zur Positionerung kann auch ein Punkt verwendet werden. ("L.", "R.", "F.")

3. Die Eingabe des Exponentialformats und hexadezimaler Zahlen ist nicht möglich.
4. Steht innerhalb des gewählten Eingabebereichs schon eine Zahl angezeigt, wird diese gelöscht, bevor eine Neueingabe angezeigt wird.
5. Die Eingabelänge X enthält alle Zeichen einschließlich Positionskommata, Vorzeichen (Minus-Zeichen bzw. Leerzeichen in den Formaten "L", "R", "L," und "R,") und Dezimalpunkt.
6. Wird [ENTER] gedrückt, ohne daß eine Zahl eingegeben wurde, wird der Variablen A der Wert 0 zugewiesen. Wurde eine Zahl vorgegeben, wird diese beibehalten. halten.

EINFACHES BEISPIELE:

1. `10 KEYIN A,7,2,"N"` ..... Eine Zahleneingabe wird der numerischen Variablen A zugewiesen. Die Eingabe darf bis zu 7 Stellen umfassen (davon 2 Dezimalstellen). Das Eingabeformat ist "N".

(Eingabedaten)		Anzeige
123	[ENTER]	0123.00
-12.34	[ENTER]	0012.34
1234.56	[ENTER]	1234.56
1234567	[ENTER]	Fehler
1234567	[−] [ENTER]	Fehler
1.234	[ENTER]	Fehler

2. `10 KEYIN A,7,2,"L"` ..... Eine Zahleneingabe wird der numerischen Variablen A zugewiesen. Die Eingabe darf bis zu 7 Stellen umfassen (davon 2 Dezimalstellen). Das Eingabeformat ist "L".

(Eingabedaten)		Anzeige
123	[ENTER]	123.00
-12.34	[ENTER]	- 12.34
-123.45	[ENTER]	-123.45
1234.56	[ENTER]	Fehler
1234567	[−] [ENTER]	Fehler
1.234	[ENTER]	Fehler

3. `10 KEYIN A,7,2,"R"` ..... Eine Zahleneingabe wird der numerischen Variablen A zugewiesen. Die Eingabe darf bis zu 7 Stellen umfassen (davon 2 Dezimalstellen). Das Eingabeformat ist "R".

(Eingabedaten)		Anzeige
123	ENTER	123.00
-12.34	ENTER	12.34-
-123.45	ENTER	123.45-
1234.56	ENTER	} Fehler
1234567	(—) ENTER	
1.234	ENTER	

4. `10 KEYIN A,7,2,"F"` ..... Eine Zahleneingabe wird der numerischen Variablen A zugewiesen. Die Eingabe darf bis zu 7 Stellen umfassen (davon 2 Dezimalstellen). Das Eingabeformat ist "F".

(Eingabedaten)		Anzeige
123	ENTER	123.00
-12.34	ENTER	-12.34
1234.56	ENTER	1234.56
-1234.56	ENTER	} Fehler
1234567	(—) ENTER	
1.234	ENTER	

5. `10 KEYIN A,9,2,"R"` ..... Eine Zahleneingabe wird der numerischen Variablen A zugewiesen. Die Eingabe darf bis zu 9 Stellen umfassen (davon 2 Dezimalstellen). Das Eingabeformat ist "R".

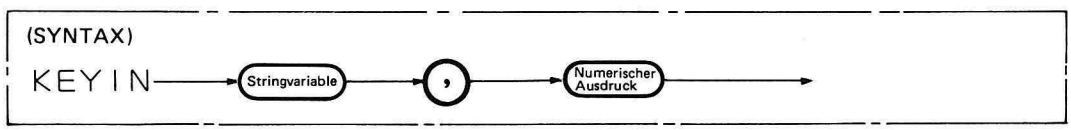
(Eingabedaten)		Anzeige
1234	ENTER	1,234.00
-123.45	ENTER	123.45-
1234.56	ENTER	1,234.56
-12345.6	ENTER	} Fehler
1234567	(—) ENTER	
123.456	ENTER	

● **FUNKTION 2:** Eingabe von Zeichen

FORMAT: KEY I N A$, X

A$: Stringvariable, der die Eingabe zugewiesen werden soll

X : Höchste zulässige Zeichenzahl (Stringlänge)



ERLÄUTERUNG:

1. Zeigt die der Stringvariablen A$ zuzuweisenden Zeichen linksbündig, beginnend mit der aktuellen Kursorposition an. Die maximal darstellbare Zeichenzahl wird durch den Wert X bestimmt.
2. Werden im Eingabefeld bereits Zeichen angezeigt, werden diese durch Neueingaben Zeichen für Zeichen überschrieben.
3. Drückt man die [ENTER]-Taste, solange der vorgegebene Eingabereich frei ist, wird der Variablen A$ ein Nullstring (" ") zugewiesen.

EINFACHES BEISPIEL:

KEYIN A$,5 ····· Stellt eine Eingabe mit bis zu 5 Zeichen auf dem Bildschirm dar und weist sie der Variablen A$ zu.

(Eingabedaten)	Anzeige
ABC [ENTER]	ABC
SHARP [ENTER]	SHARP
COMPUT ····· Fehler	

WICHTIGER HINWEIS

1. Mit dieser Anweisung werden Daten mit vorgegebener Länge vom Bildschirm gelesen, es findet also keine direkte Tastatureingabe statt. Der Variablen A$ werden daher bei Drücken der [ENTER]-Taste alle im zulässigen Eingabebereich angezeigten Zeichen zugewiesen, unabhängig vom Inhalt des Tastaturpuffers.
2. Wird versucht, mehr als die vorgegebene Zeichenzahl einzugeben, tritt ein Fehler auf (ERROR 31).

<HILFREICHER TIP>

① Eine durch Fehleingabe erzeugte Fehlerbedingung kann mit den Tasten [CL] oder [↑] aufgehoben werden.

② Zur Korrektur falscher Eingaben sind folgende Schritte nötig:

(A) Für numerische Eingaben:

Ein Druck auf die [CL] -Taste löscht den Eingabebereich. Zur Korrektur einzelner Zahlen kann die [→] - oder die [DEL] Taste verwendet werden. Beide Tasten haben hier dieselbe Funktion; das zuletzt eingegebene Zeichen wird gelöscht und die Anzeige wird um eine Position nach rechts geschoben.

(B) Für die Zeicheneingabe (Stringdaten):

Ein Druck auf die [CL] -Taste löscht den Eingabebereich. Zur Zeichenkorrektur können die [←] -, [→] -, [INS] - und [DEL] -Tasten verwendet werden. Der Kursor kann mit der [→] - oder der [←] -Taste nur innerhalb des Eingabefelds bewegt werden; Einfügungen können mit der [INS] -Taste vorgenommen werden, das letzte Zeichen des Eingabebereichs geht dabei verloren. Mit der [DEL] -Taste wird das an der Kursorposition befindliche Zeichen gelöscht.

WICHTIGER HINWEIS

1. Wird die Eingabe durch die [INS] -Funktion länger als der vorgegebene Bereich, gehen die darüber hinausgehenden Zeichen verloren. Eine Fehlerbedingung tritt nicht ein.
2. Steht der Kursor auf der ersten Stelle nach dem zulässigen Eingabebereich und wird eine andere als die [ENTER] -, [DEL] oder [←] -Taste gedrückt, wird Fehler 31 gemeldet.

(C) Eine bei einer Eingabe aufgetretene Fehlerbedingung kann durch Drücken der [CL] - oder der [↑] -Taste aufgehoben werden. Damit wird jedoch nur der Fehler beseitigt, die Programmaufführung wird damit nicht fortgesetzt.

ANWENDUNGSBEISPIEL:

Maskenaufbau

```
 10 DIM A$(4),A(4,4)
 20 DISP CLEAR :CHANGE DISP "80"
 30 TABLE /,2,7,12,11,9,16;5,3,3,3,3,3
 40 CURSOR 3,6:DISP "Nr."
 50 CURSOR 13,6:DISP "Name"
 60 CURSOR 23,5:DISP "Einzel-"
 70 CURSOR 23,6:DISP "Preis"
 80 CURSOR 32,6:DISP "Anzahl"
 90 CURSOR 42,6:DISP "GesamtPreis"
100 FOR I=1 TO 4
110 CURSOR 6,3*I+6:KEYIN A(1,I),4,0,"N"
120 CURSOR 11,3*I+6:KEYIN A$(I),9
130 CURSOR 29,3*I+6:KEYIN A(2,I),7,2,"F,"
140 CURSOR 37,3*I+6:KEYIN A(3,I),5,0,"F,"
150 A(4,I)=A(2,I)*A(3,I)
160 CURSOR 42,3*I+6:DISP USING 190;A(4,I)
170 NEXT I
180 CURSOR 0,21:END
190 IMAGE '"DM"###,###.#Z'
```

Eingegebene Daten

1. Äpfel, .25, 150
2. Birnen, .23, 250
3. Pfirsiche, 4.25, 180
4. Orangen, 1.23, 2500

Ergebnis:

```
>RUN
```

Nr.	Name	Einzel-Preis	Anzahl	GesamtPreis
0001	APfel	0.25	150	DM 37.50
0002	Birnen	0.23	250	DM 57.50
0003	Pfirsiche	4.25	180	DM 765.00
0004	Orangen	1.23	2,500	DM 3,075.00

```
READY
>
```

POP

Stellt den Unterprogrammzeiger*) um eine Position zurück.

〈VERWENDUNG〉
Rückkehr aus einem Unterprogramm, wobei die davorliegende Unterprogrammebene übersprungen wird.

FORMAT: POP

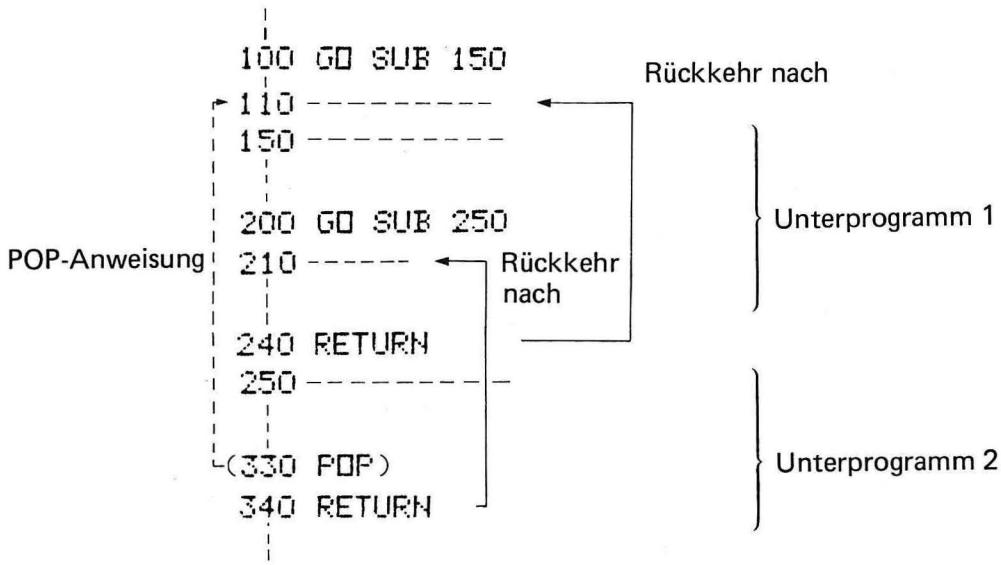
(SYNTAX)

POP⟶

ERLÄUTERUNG:

1. Entfernt die oberste Rückkehradresse (Aktuelle Zeilennummer zum Zeitpunkt des Unterprogrammaufrufs) vom Unterprogramm-Stapelspeicher.
2. Die Anweisung POP kann wiederholt, auch innerhalb einer Zeile, verwendet werden.

EINFACHES BEISPIEL:



Ohne die POP-Anweisung in Zeile 330 wird die Programmausführung mit Zeile 200 fortgesetzt. Wird dagegen die POP-Anweisung wie in obigem Beispiel benutzt, wird das Programm in Zeile 100 fortgesetzt.

WICHTIGER HINWEIS

Die POP-Anweisung wirkt sich auf die FOR-NEXT-Schleifen in gleicher Weise aus wie auf GO SUB-Anweisungen. In diesem Fall verläßt das Programm die aktuelle FOR-NEXT-Schleife und setzt das Programm mit der nächsthöheren Schleife fort.

*) Unterprogrammzeiger: Zeigt auf die aktuelle Rückkehradresse, an die bei der nächsten RETURN oder RESUME-Anweisung gesprungen werden soll.

ON ERROR

Verhindert beim Auftreten eines Fehlers einen Programmabbruch sowie die Ausgabe einer Fehlermeldung. Diese Funktion bleibt bis zur Ausführung der Anweisung OFF ERROR erhalten.

〈VERWENDUNG〉
Zur Verhinderung eines Programmabbruchs durch einen Fehler und Fortsetzung des Programms, ohne den Fehler zu beseitigen.

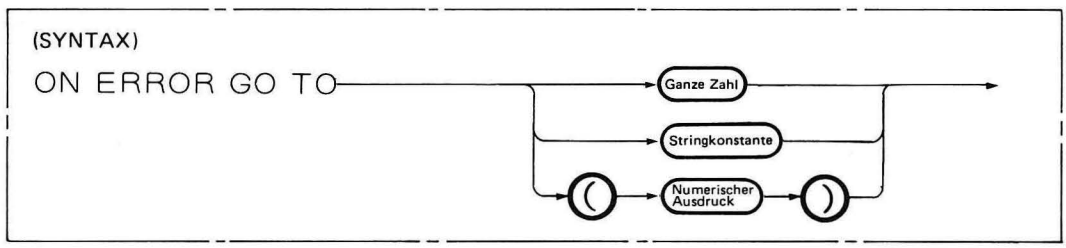
● **FUNKTION 1:** Sprung im Fehlerfall unter Verwendung von GO TO

FORMAT:ON ERROR GO TO n | S | (X)

n : Zeilenmmer, zu der gesprungen werden soll.

S : Marke der Programmzeile, zu der gesprungen werden soll.

X : Nummer (numerischer Ausdruck) der Programmzeile, zu der gesprungen werden soll.



ERLÄUTERUNG:

Tritt ein Fehler auf, wird das Programm mit der durch die Zeilennummer n, die Marke S oder den Wert des Ausdrucks X festgelegten Programmzeile fortgesetzt. Eventuell vorhandene Dezimalen von X werden abgeschnitten.

EINFACHES BEISPIELE:

1. `10 ON ERROR GO TO 100` ..... Spring zur Zeile 100, wenn ein Fehler auftritt.
2. `10 ON ERROR GO TO "ABC"` ..... Springt zur der mit der Marke "ABC" versehenen Zeile, wenn ein Fehler auftritt.
3. `10 ON ERROR GO TO (A+10)` ..... Springt zur der durch den Ausdruck A+10 festgelegten Zeile, wenn ein Fehler auftritt.

   Für A=20 wird das Programm z.B. mit Zeile 30 fortgesetzt.

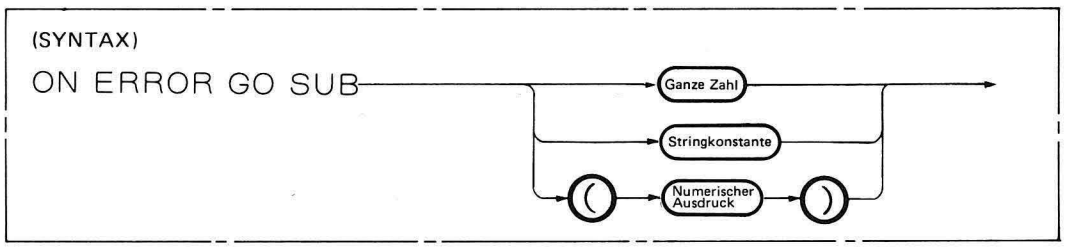
● **FUNKTION 2:** Aufruf eines Unterprogramms im Fehlerfall.

FORMAT:ON ERROR GO SUB n | S | (X)

n: Zeilennummer des Unterprogramms, das aufgerufen werden soll

S: Marke des Unterprogramms, das aufgerufen werden soll

X: Zeilennummer (Ausdruck, Variable) des Unterprogramms, das aufgerufen werden soll



ERLÄUTERUNG:

1. Tritt innerhalb des Programms ein Fehler auf, wird ein Unterprogramm aufgerufen, das durch die Zeilennummer n, die Marke S oder den Ausdruck (X) spezifiziert wird.
2. Nach Ausführung des Unterprogramms wird das Hauptprogramm am Anfang der Zeile fortgesetzt, in der der Fehler auftrat. Das Unterprogramm muß dazu mit einer RESUME-Anweisung (Seite 185) abgeschlossen werden.

EINFACHES BEISPIELE:

1. `10 ON ERROR GO SUB 200` ..... Ruft ein bei Zeile 200 beginnendes Unterprogramm auf, wenn ein Fehler auftritt.
2. `10 ON ERROR GO SUB "SUB1"` .. Ruft ein mit der Marke "SUB1" beginnendes Unterprogramm auf, wenn ein Fehler auftritt.
3. `10 ON ERROR GO SUB (1000-A)`
   ..... Ruft im Fehlerfall ein Unterprogramm auf, dessen Zeilennummer durch den Ausdruck 5*A festgelegt wird.
   Für A=20 wird z.B. ein Unterprogramm beginnend mit Zeilennummer 200 aufgerufen.

WICHTIGER HINWEIS

1. Die Anweisung ON ERROR arbeitet nur in Verbindung mit Fehlern, denen eine ungerade Fehlernummer zugewiesen ist. Eine Liste dieser Fehler ist im Appendix auf Seite 51 —— Fehlernummernliste —— enthalten. Die Anweisung ist daher z.B. nicht zur Bearbeitung eines "SYNTAX ERROR" geeignet.
2. Werden mehrere ON ERROR-Anweisungen innerhalb eines Programms ausgeführt, bleibt nur die zuletzt ausgeführte Anweisung gültig.

ANWENDUNGSBEISPIELE:

1. **BEISPIEL:** Fehlerbehandlung unter Verwendung von ON ERROR GO TO.

```
 10 ON ERROR GO TO 100 ..... Sprung zur Zeile 100, wenn ein Fehler auftritt.
 20 INPUT "X= ";X,
 30 IF X=0 THEN GO TO 200
 40 DISP "Wurzel X=";SQR (X) .. Ist X negativ, tritt ein Fehler auf. In diesem Fall
 50 GO TO 20                    wird das Programm mit Zeile 100 fortgesetzt.
100 DISP SQR (-X):GO TO 20
110 GO TO 20
200 END
```

Eingabedaten: 256, −81, 324, −169, 0

Ergebnis:

```
>RUN
X= 256                  Wurzel X= 16
X= -81                  Wurzel X= 9
X= 324                  Wurzel X= 18
X= -169                 Wurzel X= 13
X= 0
READY
>
```

2. **BEISPIEL:** Aufruf eines Unterprogramms, wenn ein Fehler auftritt.

```
 10 INPUT "X=";X,"Y=";Y
 20 ON ERROR GO SUB 100
 30 Z=X^Y ..... Für negatives X tritt ein Fehler auf.
                In diesem Fall wird das bei Zeile 100
                beginnenden Unterprogramm aufgerufen.
 40 OFF ERROR ..... Hebt die ON ERROR-Anweisung auf.
 50 DISP "X^Y=";Z
 60 GO TO 10 ..... Springt zur Zeile 10. Die ON ERROR
                   Anweisung wird erneut ausgeführt.
 70 END
100 REM UnterProgramm
110 IF Y<>INT Y THEN DISP "Fehler!":END
120 Y1=Y/2
130 IF Y1=INT Y1 THEN X=-X:RESUME
140 Z=-(-X)^Y
150 RESUME NEXT ..... Nach der Ausführung kehrt das Unter-
                      programm auf Zeile 40 zurück. Dies ist
                      die Zeile nach Zeile 30, in der der Fehler
                      auftrat.
```

Unterprogram

Eingabedaten: 3, 2, −5, 3, −4, 1,2

Ergebnis:

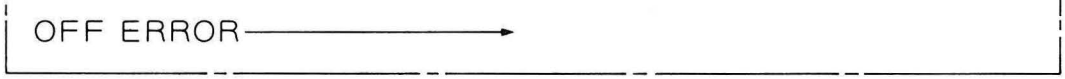
```
>RUN
X=3                     Y=2
X^Y= 9
X=-5                    Y=3
X^Y=-125
X=-4                    Y=1,2
Fehler!
READY
>
```

OFF ERROR

Hebt eine ON ERROR-Anweisung auf.

FORMAT: OFF ERROR

(SYNTAX)



ON KEY

Führt einen Sprung aus, wenn die spezifizierte Taste gedrückt wird. ON KEY bleibt bis zur Ausführung der Anweisung OFF KEY gültig.

〈VERWENDUNG〉
Um den Ablauf eines Programms durch Drücken einer bestimmten Taste zu ändern.

● **FUNKTION 1:** Sprung unter Verwendung von GO TO

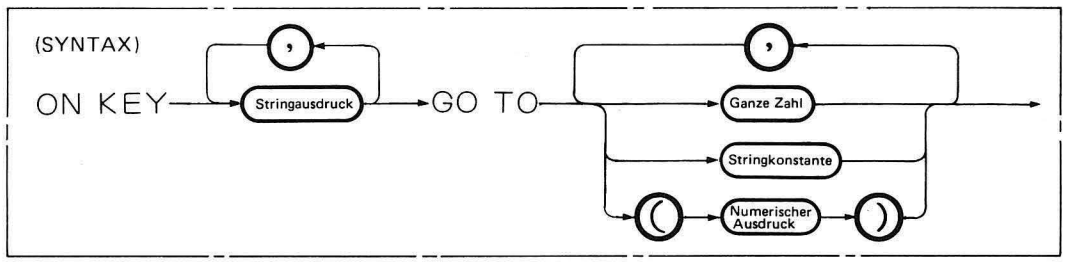
FORMAT:ON KEY {C} GO TO {n | S | (X)}

C: Gewünschte Taste

n: Zeilennummer, zu der gesprungen werden soll

S: Marke der Zeile, zu der gesprungen werden soll

X: Zeilennummer (numerischer Ausdruck), zu der gesprungen werden soll



ERLÄUTERUNG:

1. Die Programmausführung wird mit der durch Zeilennummer n, Marke S oder den numerischen Ausdruck X (Dezimalen werden abgeschnitten) festgelegten Zeile fortgesetzt, wenn die durch C definierte Taste gedrückt wurde.
2. Der Sprung zu der in 1. beschriebenen Programmzeile wird erst ausgeführt, wenn die Zeile, in der die mit C definierte Taste gedrückt wurde, abgearbeitet worden ist.

BEISPIEL:

```
10 ON KEY "A","B","C" GO TO 100,"ABC",(150+50)
```

Das Programm wird mit der Zeile 100 fortgesetzt, wenn "A" eingegeben wird, mit der durch "ABC" gekennzeichneten Zeile bei Eingabe von "B" und mit der Zeile (150+Y) bei Eingabe von "C".

● **FUNKTION 2:** Aufruf eines Unterprogramms.

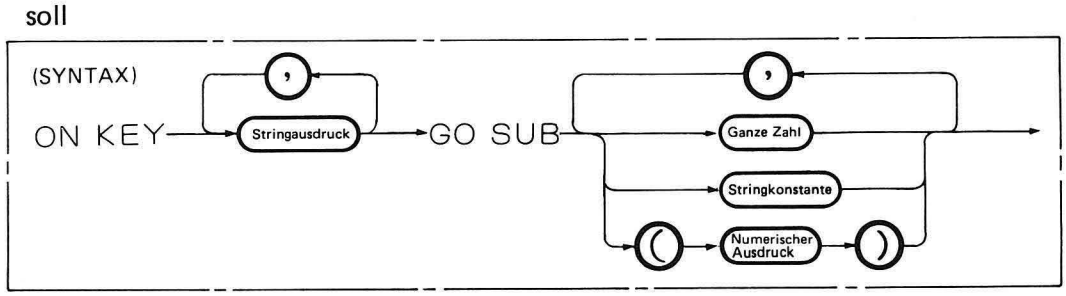
FORMAT:ON KEY {C} GO SUB {n | S | (X)}

C: Festgelegte Taste

n: Zeilennummer des Unterprogramms, das aufgerufen werden soll

S: Marke des Unterprogramms, das aufgerufen werden soll

X: Zeilennummer (Ausdruck, Variable) der Unterprogramms, das aufgerufen werden soll



ERLÄUTERUNG:

1. Ein mit Zeilennummer n, Marke S oder dem numerischem Austruck X festgelegtes Unterprogramm wird aufgerufen, wenn die durch C festgelegte Taste gedrückt wurde. Bei der Verwendung numerischer Ausdrücke ist nur der ganzzahlige Teil maßgeblich.
2. Das Unterprogramm wird erst nach Abarbeitung der Programmzeile aufgerufen, in der die mit C definierte Taste gedrückt wurde. Nach Abarbeitung des Unterprogramms wird die Programmausführung mit der Zeile fortgesetzt, die der aufrufenden folgt. Das Unterprogramm muß dazu mit RESUME abgeschlossen sein.

BEISPIEL:

```
10 ON KEY "A","B","C" GO SUB (X+50),"ABC",100
```

Ruft ein Unterprogramm auf, dessen Zeilennummer durch den Ausdruck (X+50) festgelegt ist, wenn die Taste "A" gedrückt wurde. Analog dazu wird bei Eingabe von "B" und "C" das Unterprogramm "ABC" bzw. ein mit Zeile 100 Beginnendes aufgerufen.

WICHTIGER HINWEIS

1. Mit einer ON KEY-Anweisung können bis zu 28 Stringvariablen deklariert werden, d.h. bis zu 28 Tasten können als Funktionstasten definiert werden.
2. Nur das erste Zeichen eines mit C festgelegten Strings ist für die Tastenerkennung maßgeblich.
3. Um eine programmierbare Funktionstaste oder eine Steuertaste mit der Stringvariablen C zu definieren, müssen die entsprechende Tastenkodes verwendet werden:

Taste	Tastenkode
DEF 1	CHR$ &01 (oder CHR$ 1)
DEF 2	CHR$ &02 (oder CHR$ 2)
|	| |
DEF 9	CHR$ &09 (oder CHR$ 9)
DEF10	CHR$ &0A (oder CHR$ 10)
DEF11	CHR$ &11 (oder CHR$ 17)
DEF12	CHR$ &12 (oder CHR$ 18)
|	| |
DEF19	CHR$ &19 (oder CHR$ 25)
DEF20	CHR$ &1A (oder CHR$ 26)
INS	CHR$ &0B (oder CHR$ 11)
DEL	CHR$ &7F (oder CHR$ 127)
ENTER	CHR$ &0D (oder CHR$ 13)
ENTER	CHR$ &0E (oder CHR$ 14)
CL	CHR$ &0F (oder CHR$ 15)
HOME	CHR$ &1B (oder CHR$ 27)
↑	CHR$ &1C (oder CHR$ 28)
↓	CHR$ &1D (oder CHR$ 29)
→	CHR$ &1E (oder CHR$ 30)
←	CHR$ &1F (oder CHR$ 31)

BEISPIEL:

```
10 ON KEY CHR$&01,CHR$&0D GO TO 200,300
 :
90 OFF KEY
```

In obigem Beispiel wird das Programm mit Zeile 200 fortgesetzt, wenn die Taste DEF1 gedrückt wurde, oder mit Zeile 300, wenn die [ENTER] -Taste gedrückt wurde.

4. Eine Neudefinition hebt die letzte ON KEY-Anweisungen auf. Die Neudefinition kann an einer beliebigen Stelle im Programm vorgenommen werden.
5. Für die ON KEY-Anweisung ist es unerheblich, an welcher Stelle die erforderliche Eingabe gemacht wird. Erfolgt die Eingabe innerhalb einer INPUT- oder KEYIN-Anweisung (Seite 117 bzw. 165), wird die Sprunganweisung erst nach deren Abschluß ausgeführt. Es ist nicht von Bedeutung, an welcher Stelle innerhalb der Anweisung die Eingabe zur Ausführung von ON KEY gemacht wird.

BEISPIEL:

Ausführung der ON KEY-Anweisung, solange sich INPUT nicht im Eingabemodus befindet:

```
 10 ON KEY "A" GO TO 1000
  :
100 INPUT X
  :
1000.....
```

Gibt man beispielsweise 123.45 [ENTER] A ein, solange die Zeilen zwischen 10 und 100 abgearbeitet werden, wird die Eingabe 123.45 aus dem Eingabepuffer gelöscht und die Programmausführung wird mit Zeile 1000 fortgesetzt.

BEISPIEL:

Ausführung der ON KEY-Anweisung, wenn sich INPUT im Eingabemodus befindet:

```
 10...........
  :
100 ON KEY "A" GO TO 1000:INPUT X:OFF KEY
110 GO TO 100
  :
1000.........
```

Gibt man 123.45 [ENTER] A, solange die Zeilen 10 bis 100 abgearbeitet werden, wird die Zahl 123.45 der numerischen Variablen X zugewiesen und dann zur Zeile 1000 gesprungen.

6. Nach Ausführung eines Sprungs durch die ON ERROR-Anweisung (Seite 172) oder die ON KEY-Anweisung wird automatisch eine OFF KEY-Anweisung (Seite 180) ausgeführt.
   Wurde die mit ON ERROR angesprungene Routine mit RESUME (Seite 185) abgeschlossen, tritt nach Rückkehr ins Hauptprogramm die ON KEY-Anweisung wieder in Kraft. Wird dagegen RETURN (Seite 136) verwendet, bleibt die ON KEY-Anweisung ungültig.

ANWENDUNGSBEISPIEL:

Programm A: Wir lassen einen Würfel rollen und erraten die Zahl. Glauben wir, sie sei kleiner oder gleich drei, drücken wir die Taste A, nehmen wir an, liegt sie über 3, drücken wir B. Zur Beendigung des Spiels kann E gedrückt werden.

```
 10 ON KEY "A","B","E" GO TO 40,50,100
```
Springt bei Eingabe von A zur Zeile 40, bei B zur Zeile 50 und bei E zur Zeile 100.

```
 20 X=INT (RND 6)+1
```
Ermittelt unter Verwendung von Zufallszahlen die gefallene Würfelseite.

```
 30 GO TO 20
 40 IF X<4 THEN 60:ELSE 80
```
Ergebnisprüfung und -anzeige, wenn A gedrückt wurde.

```
 50 IF X>3 THEN 60:ELSE 80
```
Ergebnisprüfung und -anzeige, wenn B gedrückt wurde.

```
 60 DISP "Richtig!"
 70 GO TO 10
```
Definiert ON KEY neu.

```
 80 DISP "Daneben!"
 90 GO TO 10
```
Definiert ON KEY neu.

```
100 DISP "SPielende":END
```
Programmausführung wird nach Eingabe von E beendet.

Programm B: Gleiche Funktionen wie A, jedoch unter Verwendung von Unter-Programmen:

```
 10 ON KEY "A","B","E"
    GO SUB 40,70,100
```
Ruft Unterprogramme, beginnend mit den Zeilen 40, 50 oder 100 aus, wenn die entsprechenden Tasten A, B oder E gedrückt werden.

```
 20 X=INT (RND 6)+1
 30 GO TO 20
```
Ermittelt unter Verwendung von Zufallszahlen die gefallene Würfel-Seite.

```
 40 REM UnterProGramm 1
 50 IF X<4 THEN DISP "Richtig!":ELSE DISP "Daneben!"
 60 RESUME
```
Nach Rückkehr vom Unterprogramm ist ON KEY wieder aktiv.

```
 70 REM UnterProGramm 2
 80 IF X>3 THEN DISP "Richtig!":ELSE DISP "Daneben!"
 90 RESUME
```
Nach Rückkehr vom Unterprogramm ist ON KEY wieder aktiv.

```
100 DISP "SPielende":END
```

OFF KEY

Hebt die ON KEY-Anweisung auf.

FORMAT: OFF KEY

(SYNTAX)



ON TIME

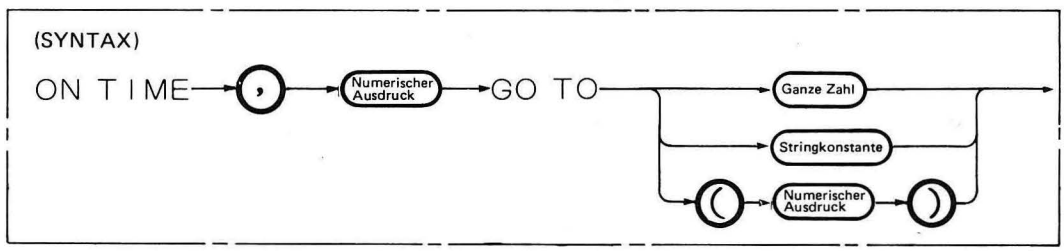
Veranlaßt zum angegebenen Zeitpunkt einen Sprung zu einer vorgegebenen Zeile, bis eine OFF TIME-Anweisung ausgeführt wird.

〈VERWENDUNG〉
Zum Abbruch eines Programmlaufs oder zum Starten eines anderen Programms zu einem festgelegten Zeitpunkt.

● **FUNKTION 1:** Sprung mittels der GO TO-Anweisung, wenn ein vorgegebener Zeitpunkt eingetreten ist.

FORMAT: ON TIME, X GO TO n | S | (Y)

X : Vorgegebener Zeitpunkt (Stunden. Minuten Sekunden)
n : Nummer der Zeile, zu der gesprungen werden soll
S : Zeilenmarke, zu der gesprungen werden soll
Y : Numerischer Ausdruck der Zeilennummer, zu der gesprungen werden soll



ERLÄUTERUNG:

1. Die Anweisung ON TIME veranlaßt zu einem vorgegebenen Zeitpunkt X einen Sprung zu der Zeile n oder zu der Zeilenmarke S oder zu der Zeile mit der Nummer, die dem Wert des numerischen Ausdruck Y entspricht (ist Y keine ganze Zahl, werden die Dezimalstellen abgeschnitten).
   Der ganzzahlige Teil von X zeigt die Stunden (24-Stunden-System), der Dezimalteil die Minuten und Sekunden an.
2. Der Sprung zu der in der ON TIME-Anweisung angegebene Sprungaddresse erfolgt nach Abarbeitung der Zeile, in der sich die Ausführung beim Erreichen des vorgegebenen Zeitpunkts gerade befindet.

EINFACHES BEISPIELE:

1. `10 ON TIME,15.10 GO TO 100`
   ..... Diese Anweisung bewirkt um 15 h 10 m einen Sprung zur Zeile 100 (nach Abarbeitung der in Ausführung befindlichen Zeile).

2. `20 ON TIME,A+0.3 GO TO "ABC"`
   ..... Diese Anweisung veranlaßt einen Sprung zur Zeilenmarke ABC zum Zeitpunkt A+0.3. (Hat numerische Variable z.B. den Wert 12, dann erfolgt der Sprung um 12 h 30 m.)

● **FUNKTION 2:** Sprung in ein Unterprogramm zum vorgegebenen Zeitpunkt.

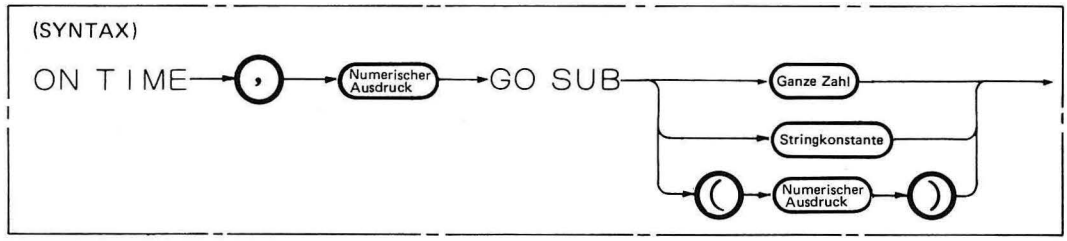
FORMAT:ON TIME, X GO SUB n | S | (Y)

X : Vorgegebener Zeitpunkt (Stunden. Minuten Sekunden)

n : Zeilennummer des Unterprogramms, zu dem gesprungen werden soll

S : Marke des Unterprogramms, zu dem gesprungen werden soll

Y : Numerischer Ausdruck der Zeilennummer des Unterprogramms, zu dem gesprungen werden soll



ERLÄUTERUNG:

1. Diese Anweisung veranlaßt zu einem vorgegebenen Zeitpunkt X einen Sprung in ein Unterprogramm, das in der Zeile n, mit der Zeilenmarke S oder der Zeilenummer, die dem Wert des numerischen Ausdrucks Y entspricht, beginnt (ist Y keine ganze Zahl, werden die Dezimalstellen abgeschnitten).
2. Der Sprung zu dem in der ON TIME-Anweisung angegebenen Unterprogramm erfolgt nach Abarbeitung der Zeile, in der sich die Ausführung beim Erreichen des vorgegebenen Zeitpunkts gerade befindet.
   Nach Abarbeitung des Unterprogramms kehrt die Ausführung zu derjenigen Zeile zurück, die der Zeile folgt, von der aus der Sprung in das Unterprogramm erfolgt ist. Das Unterprogramm kann mittels der Anweisung RESUME (Seite 185) verlassen werden.

EINFACHES BEISPIELE:

1. `10 ON TIME,11.5 GO SUB 100`
   ..... Diese Anweisung bewirkt um 11 h 50 m einen Sprung in ein Unterprogramm, das in Zeile 100 beginnt.
2. `20 ON TIME,A*2 GO SUB "ABC"`
   ..... Diese Anweisung veranlaßt zum Zeitpunkt A*2 einen Sprung zu einem Unterprogramm, das in der Zeile mit der Marke ABC beginnt. (Hat die numerische Variable z.B. den Wert 7, dann erfolgt der Sprung um 14 h 00 m.)

WICHTIGER HINWEIS

Bei mehrmaligem Durchlaufen einer ON TIME-Anweisung ist nur die jeweils letzte gültig.

ANWENDUNGSBEISPIEL:

Man prüfe, ob man die Zahlen 1 bis 10000 innerhalb eines vorgegebenen Zeitintervalls am Bildschirm ausgeben kann.

```
 10 INPUT "ZEIT=";X ..... Eingabe der Vorgabezeit.
 20 TIME=0
 30 ON TIME,TIME+X GO TO 80 ..... Sprung zu Zeile 80, wenn die Vorgabezeit
 40 FOR I=1 TO 10000          abgelaufen ist.
 50 DISP I; ..... Ausgabe der Zahlen.
 60 NEXT I
 70 GO TO 100
 80 DISP ..... Zeilenwechsel Zeitende-Meldung
 90 DISP "ZEIT IST ABGELAUFEN!"
100 END
```

Nach dem Starten des Programms gebe man z.B. 0.01 ein (dies entspricht 1 Minute).
Bildschirmanzeige nach dem Starten des Programms und erfolgter Eingabe:

```
>RUN
ZEIT=0.01
 1  2  3  4  5 .....
```

Sollte die Zeit nicht ausgereicht haben für die Ausgabe der Zahlen 1 – 10000, erscheint am Bildschirm folgende Meldung:

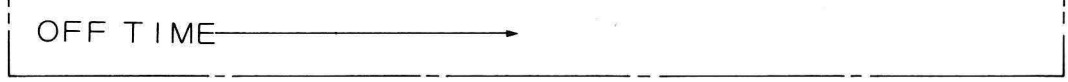
```
ZEIT IST ABGELAUFEN!
READY
>
```

OFF TIME

Hebt die ON TIME-Anweisung auf.

FORMAT: OFF TIME

(SYNTAX)



RESUME

Beendet ein Unterprogramm, zu dem mittels einer Anweisung gesprungen wurde, die mit ON*) beginnt und setzt das Programm fort.

〈VERWENDUNG〉
Zum Beenden eines Unterprogramms, zu dem mittels einer Anweisung gesprungen wurde, die mit ON beginnt und Fortsetzen des Ursprungprogramms.

● **FUNKTION 1:** Rückkehr zu der Zeile, die ver dem Sprung abgearbeitet nurde.

FORMAT:RESUME

(SYNTAX)

RESUME ⟶

EINFACHES BEISPIELE:

```
10 ON KEY "A" GO SUB 100
:
30 A=B+C  <-------- Nehmen wir an, die Taste A wurde während
40 DISP A           der Ausführung dieser Zeile gedrückt.
:
100 ..........  }
:               } UNTERPROGRAMM
200 RESUME      }
```

Rückkehr zu

Die Taste A soll während der Ausführung von Zeile 30 gedrückt worden sein. Darauf erfolgte ein Sprung zu Zeile 100, in der das Unterprogramm beginnt. Das Unterprogramm wird abgearbeitet. Am Schluß des Unterprogramms steht die RESUME-Anweisung und bewirkt einen Sprung zu Zeile 30.

● **FUNKTION 2:** Rückkehr zu der Zeile, die der Zeile folgt, zu der in FUNKTION 1 zurückgesprungen wurde.

FORMAT:RESUME NEXT

(SYNTAX)

RESUME NEXT ⟶

ERLÄUTERUNG:

In diesem Fall beginnt die Fortsetzung der Programmausführung in der Zeile, die der Zeile folgt, zu der in FUNKTION 1 zurückgesprungen wurde.

*) z.B. die Anweisungen ON . . . GO SUB, ON ERROR, ON KEY, ON TIME

EINFACHES BEISPIELE:

```
10 ON ERROR GO SUB 100
 :
30 A=SQR(X) ..... Nehmen Sie an, bei der Abarbeitung dieser Zeile tritt
40 B=SQR(-X)      ein Fehler auf; das System veranlaßt mit der ON ERROR
50 DISP A,B       – Anweisung einen Sprung in das Unterprogramm.
 :                                          Rückkehr zu
100 ..............
 :                 } UNTERPROGRAMM
200 RESUME NEXT
```

Hat die Variable X einen positiven Wert, dann tritt während der Abarbeitung von Zeile 30 ein Fehler auf. Das System veranlaßt daher einen Sprung zu dem in Zeile 100 beginnenden Unterprogramm. Nach Abarbeitung des Unterprogramms bewirkt die RESUME NEXT-Anweisung einen Sprung zu Zeile 40.

WICHTIGER HINWEIS

In einem Unterprogramm, das mittels einer mit ON beginnenden Anweisung angesprungen wurde, könnte anstatt der RESUME-Anweisung zur Rückkehr in das aufrufende Programm auch die RETURN-Anweisung verwendet werden. Trotzdem wird in diesem System der Gebrauch der RESUME-Anweisung empfohlen. Der Unterschied in der Verwendung dieser beiden Anweisungen wird im Absatz "PRAKTISCHER TIP" erläutert.

<PRAKTISCHER TIP>

Benutzt man in einem Programm die ON KEY-Anweisung, dann bleibt die Vereinbarung der ON KEY-Anweisung nur dann gültig, ween zur Rückkehr aus dem Unterprogramm, in das mittels der Anweisung ON KEY oder ON ERROR gesprungen wurde, eine RESUME-Anweisung verwendet worden ist. Wurde dagegen eine RETURN-Anweisung benutzt, verliert die ON KEY-Anweisung von diesem Moment an ihre Gültigkeit. (Seite 180).

MOVE

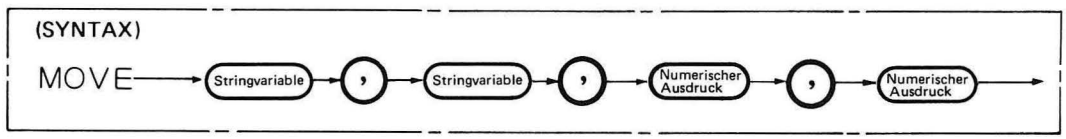
Partielle Übertragung eines Strings auf eine Stringvariable, Verknüpfung eines Strings mit Teilen eines anderen Strings oder Kopieren von Zeilen eines BASIC-Programms.

〈VERWENDUNG〉
① Zue Übertragung von Teilen eines Strings auf eine Stringvariable.
② Zur Verknüpfung eines Strings mit Teilen eines anderen Strings.
③ Zum Kopieren von BASIC-Programmzeilen innerhalb eines BASIC-Programmtextes, z.B. wenn man den gleichen oder einen ähnlichen Programmtext mit anderen Zeilennummern benötigt.

● **FUNKTION 1:** Partielle Übertragung dines Strings von einer Stringvariablen auf eine andere Stringvariable.

FORMAT:MOVE A$, B$, X, Y

A$: Stringvariable, der Zeichen zugeordnet werden sollen
B$: Stringvariable, von der Zeichen übertragen werden sollen
X : Position des ersten der zu übertragenden Zeichen in String B$
Y : Anzahl der zu übertragenden Zeichen



ERLÄUTERUNG:

1. Die MOVE-Anweisung überträgt aus dem String B$ ab dem X-ten Zeichen Y Zeichen in fortlaufender Folge auf die Stringvariable A$.
2. Die in dieser Anweisung enthaltenen Stringvariablen können bis zu 256 Zeichen enthalten. Stringvariablen mit mehr als 16 Zeichen müssen mittels der DIM-Anweisung als String-Array-Variable deklariert worden sein (Seite 128).

EINFACHES BEISPIEL:

```
10 B$="SCHORNSTEINFEGER"
20 MOVE A$,B$,7,5 ..... Anweisung zum Übertragen von 5 Zeichen beginnend mit dem 7, Zeichen von B$ auf A$.
30 DISP A$
40 END
```

Bildschirmanzeige nach dem Starten des Programms:

```
>RUN
STEIN ..... übertragener String
READY
>
```

● **FUNKTION 2:** Verkettung einer Stringvariablen mit dem Teil einer anderen Stringvariablen.

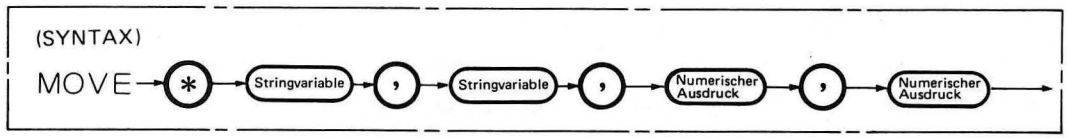
FORMAT: MOVE *A$, B$, X, Y

A$: die zu verkettende Stringvariable

B$: Stringvariable, die Zeichen für die Verkettung liefert

X : Position des ersten der zu verkettenden Zeichen

Y : Zahl der Zeichen, die mit A$ verkettet werden sollen



ERLÄUTERUNG:

1. Die MOVE-Anweisung verkettet Y Zeichen beginnend mit dem X. Zeichen aus dem String B$ mit dem String A$.
2. Die in dieser Anweisung enthaltenen Stringvariablen können bis zu 256 Zeichen enthalten. Stringvariablen mit mehr als 16 Zeichen müssen mittels der DIM-Anweisung als String-Array-Variable deklariert worden sein.

EINFACHES BEISPIELE:

```
10 A$="SCHUH"
20 B$="HUTMACHER"
30 MOVE *A$,B$,4,6
40 DISP A$
50 END
```

Bildschirmausgabe nach dem Starten des Programms:

```
>RUN
SCHUHMACHER
READY
>
```

● **FUNKTION 3:** Kopieren von Zeilen eines BASIC-Programmtextes.

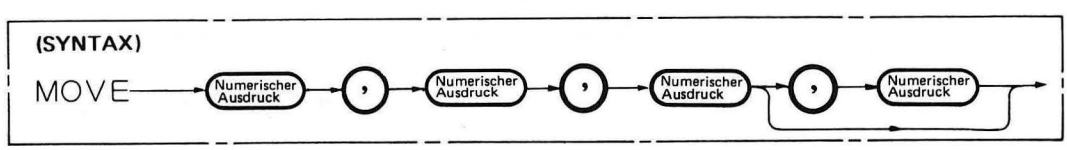
FORMAT: MOVE U, V, X [, Y]

U : Nummer der Zeile, ab der kopiert werden soll

V : Nummer der Zeile, bis zu der kopiert werden soll

X : Nummer der ersten Zeile der Kopie

Y : Inkrement der Zeilennummern



ERLÄUTERUNG:

1. Kopiert die Programmzeilen von Zeile U bis Zeile V und beginnt die Numerierung der kopierten Zeilen bei X mit dem Inkrement Y. Die Inhalte der Zeilen U bis V bleiben unverändert.
2. Unterbleibt in der MOVE-Anweisung eine Angabe zur Inkrementierung, wird automatisch jeweils um 10 erhöht.
3. Dieser Befehl kann nur in der manuellen Betriebsart und nicht innerhalb eines Programms verwendet werden, vorausgesetzt, der Betriebsartschalter steht auf PRO. Seine Verwendung im OP-Modus verursacht eine Fehlermeldung (ERROR 34).

EINFACHES BEISPIEL:

```
MOVE 30,100,150,10 [ENTER]
```
····Fertigt eine Kopie der BASIC-Programmzeilen 30 bis 100, die mit der Programmzeile 150 beginnt. Das Inkrement beträgt 20.

WICHTIGER HINWEIS

In den kopierten Programmzeilen werden die Sprungadressen nicht angepaßt, sie bleiben unverändert.

ANWENDUNGSBEISPIELE:

Programm A:

Man entnehme dem String

"SHARP PERSONAL COMPUTER < MODELL-3500 SERIE >"

die Worte "SHARP" und "MODEL-3500" und gebe sie am Bildschirm aus.

```
10 DIM A$(1)*100
20 A$(1)="SHARP PERSONAL COMPUTER <MODEL-3500 SERIES>"
30 MOVE B$,A$(1),1,6
40 MOVE C$,A$(1),26,11
50 MOVE *B$,C$,1,10
60 DISP B$
70 END
```

30 MOVE ····Ordnet der Stringvariablen B$ 6 Zeichen zu, beginnend mit dem ersten Zeichen.

40 MOVE ····Ordnet der Stringvariablen C$ 11 Zeichen zu, beginnend mit dem 26. Zeichen.

50 MOVE ····Verkettet den Inahlt der Stringvariablen C$ mit der Stringvariablen B$.

Bildschirmausgabe nach dem Starten des Programms:

```
>RUN
SHARP MODEL-3500
READY
>
```

Programm B

```
 10 READ A
 20 X=SIN A
 30 Y=COS A
 40 Z=TAN A
 50 DISP "sin=";X,"cos=";Y,"tan=";Z
 60 END
100 DATA 15,30,45,60,75
110 REM **********************
```

Man gebe in der manuellen Betriebsart folgende Anweisung ein:

```
MOVE 10,60,200,100 ENTER
```

Anschließend gebe man das Programm mit LIST am Bildschirm aus:

```
 10 READ A
 20 X=SIN A
 30 Y=COS A
 40 Z=TAN A
 50 DISP "sin=";X,"cos=";Y,"tan=";Z
 60 END
100 DATA 15,30,45,60,75
110 REM **********************
200 READ A
300 X=SIN A
400 Y=COS A
500 Z=TAN A
600 DISP "sin=";X,"cos=";Y,"tan=";Z
700 END
```

Die Zeilen 10 bis 60 wurden kopiert und neu numeriert.

MAXS

Bestimmen des größten Arrayelements in einer numerischen Arrayvariablen.

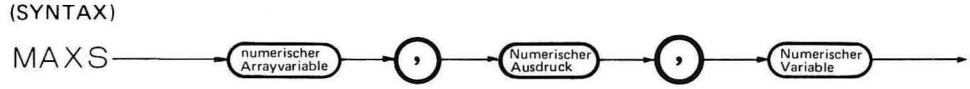
〈VERWENDUNG〉

Wenn man herausfinden möchte, welches Arrayelement innerhalb eines numerischen Arrays den größten Wert besitzt.

FORMAT: MAXS A (*), X, B

A (*) : Kollektive numerische Arrayvariable
X : Position des Arrayelements, bei dem die Suche beginnen soll
B : Variable, der die Position des Größten Elements zugeordnet werden soll

(SYNTAX)



ERLÄUTERUNG:

1. Mittels der Anweisung MAXS kann man das größte Element in dem numerischen Array A – dargestellt durch die kollektive numerische Arrayvariable A(*) – herausfinden.
   Mit X bezeichnet man die Position desjenigen Elements, ab dem die Suche beginnen soll; d.h. man kann bei der Suche nach dem größten Arrayelement eine bestimmte Anzahl vorderer Elemente ausnehmen.
   Die Anweisung weist der Variablen B die Position des Arrayelements mit dem größten Wert zu.
2. Die angegebenen Position bezeichnet die Ordinalzahl des gesuchten Arrayelements, d.h. die Nummer in der Reihenfolge der Anordnung der Elemente innerhab des Arrays. (Siehe DIM-Anweisung Seite 128).
3. Eine Fehlermeldung erfolgt, wenn der Wert des numerischen Ausdrucks X kleiner oder gleich 0 oder aber größer als die Zahl der Elemente in dem Array A(*) ist.
4. Kommt der größte Wert mehr als einem Element zu, dann liefert B die Position des ersten dieser gleichgroßen Elemente.

EINFACHES BEISPIELE:

```
50 MAXS M(*),3,X1
```
Ordnet die Position des größten Elements innerhalb des Array M(*) der Variablen X1 zu und beginnt die Suche beim dritten Arrayelement. Besitzt z.B. M(10) den größten Wert, dann hat die Variable X1 nach Ausführung der MAX-Anweisung den Wert 10.

ANWENDUNGSBEISPIEL:

Bei diesem Beispiel werden nacheinander 10 Zahlen eingegeben. Das Programm soll nun ermitteln, die wievielte der eingegebenen Zahlen am größten war.

```
10 DIM N(10)
20 FOR J=1 TO 10
30 INPUT N(J)
40 PRINT "NR";J;"=";N(J)
50 NEXT J
60 MAXS N(*),1,A
70 PRINT "NR";A;"IST AM GRöSSTEN UND HAT DEN WERT";
   N(A)
80 END
```

Nehmen wir einmal an, wir hätten nacheinander die Zahlen 27, 4, 30, 90, –14, 0, 257, –111, 56, 3 eingegeben. Wir erhalten dann folgenden Ausdruck:

```
NR 1 = 27
NR 2 = 4
NR 3 = 30
NR 4 = 90
NR 5 =-14
NR 6 = 0
NR 7 = 257
NR 8 =-111
NR 9 = 56
NR 10 = 3
NR 7 IST AM GRöSSTEN UND HAT DEN WERT 257
```

Wenn wir in diesem Programm die PRINT- durch eine DISP-Anweisungen ersetzen, erhalten wir die gleiche Ausgabe über den Bildschirm.

MINS

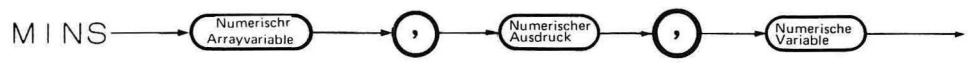
Bestimmen des kleinsten Arrayelements in einer numerischen Arrayvariablen.

〈VERWENDUNG〉
Wenn man herausfinden möchte, welches Arrayelement innerhalb eines numerischen Arrays den kleinsten Wert besitzt.

FORMAT: M I NS A (*), X, B

A (*) : Kollektive numerische Arrayvariable
X : Position des Arrayelements, bei dem die Suche beginnen soll
B : Variable, der die Position des kleinsten Elements zugeordnet werden soll

(SYNTAX)



ERLÄUTERUNG:

1. Mittels der Anweisung MINS kann man das kleinste Element in dem numerischen Array A – dargestellt durch die kollektive numerische Arrayvariable A(*) – herausfinden.
   Mit X bezeichnet man die Position desjenigen Elements, ab dem die Suche beginnen soll; d.h. man kann bei der Suche nach dem kleinsten Arrayelement eine bestimmte Anzahl vorderer Elemente ausnehmen.
   Die Anweisung weist der Variablen B die Position der Arrayelements mit dem kleinsten Wert zu.
2. Die angegebene Position bezeichnet die Ordinalzahl des gesuchten Arrayelements, d.h. die Nummer in der Reihenfolge der Anordnung der Elemente innerhalb des Arrays. (Siehe DIM-Anweisung Seite 128).
3. Eine Fehlermeldung erfolgt, wenn der Wert des numerischen Ausdrucks X kleiner oder gleich 0 oder aber größer als die Zahl der Elemente in dem Array A(*) ist.
4. Kommt der kleinste Wert mehr als einem Element zu, dann liefert B die Position des ersten der gleichgroßen Elemente.

EINFACHES BEISPIEL:

```
50 MINS D(*),2,E
```
Ordnet die Position des kleinsten Elements innerhalb des Array D(*) der Variablen Y1 zu und beginnt die Suche beim fünften Arrayelement. Besitzt z.B. D(12) den kleinsten Wert, dann hat die Variable Y1 nach Ausführung der MINS-Anweisung den Wert 12.

ANWENDUNGSBEISPIEL:

Das Programm soll so formuliert werden, daß es aus einer Folge von Zahlen, die mittels der DATA-Anweisung eingelesen werden, die kleinste herausfindet und hierbei mit der zweiten Zahl beginnt. Das Ergebnis soll über den Bildschirm ausgegeben werden.

```
 10 DIM N(10)
 20 FOR J=1 TO 10
 30 READ N(J)
 40 NEXT J
 50 MINS N(*),2,A
 60 DISP N(A)
 70 END
100 DATA -8,-2,1,3,-7,6,10,13,4,2
```

Nach dem Starten des Programms erhalten wir folgende Ausgabe am Bildschirm:

```
>RUN
-7
READY
>
```

Läßt man die erste Zahl außer acht, ist von den eingelesenen Zahlen die fünfte am kleinsten.

SEARCH

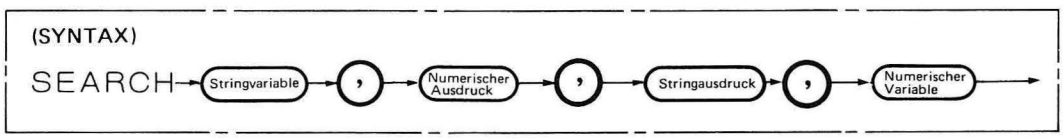
Liefert die Position des ersten Auftretens eines Strings in einem anderen String.

〈VERWENDUNG〉
① Wenn man die Position eines Strings in einem anderen, längeren String bestimmen möchte.
② When man in einem Stringarray ein Arrayelement mit einem ganz bestimmten String (z.B. einen Namen) sucht.

● **FUNKTION 1:** Sucht einen bestimmten String in einem anderen String.

FORMAT: SEARCH A$, X, C, A

A$: Zu durchsuchende Stringvariable (auch einelementige Stringarrayvariable) oder Stringarrayelement
X : Position des Zeichens, ab dem zu suchen ist
C : Der aufzufindende String
A : Numerische Variable, der die Beginnposition von String C in String A$ zugewiesen wird



ERLÄUTERUNG:

Die Anweisung SEARCH sucht nach dem erstmaligen Auftreten des Strings C in String A$ und beginnt ihre Suche bei dem X. Zeichen von String A$. Wurde String C in String A$ gefunden, dann ordnet sie der numerischen Variablen A die Stellenzahl des Zeichens zu, an dem String C beginnt. (Würde z.B. der String "LEBEN" in dem String "HUNDELEBEN" gescut, dann bekäme A den Wert 6 zugewiesen).

Wenn String C in String A$ nicht gefunden wurde, String A$ kürzer als String C war oder String A$ vor der Suche ein Nullstring (" ") zugewiesen wurde, dann wird der numerischen Variablen A der Wert 0 zugeordnet. Ist dagegen String C ein Nullstring, wird eine Fehlermeldung ausgegeben.

Die Stringvariable A$ kann eine einfache Stringvariable (auch einelementige, eindimensionale Stringarrayvariable für Strings mit mehr als 16 Zeichen) oder das Element einer kollektiven Stringarrayvariablen sein.

EINFACHES BEISPIEL:

```
10 A$="SHARP COMPUTER"
20 SEARCH A$,2,"COMPUTER",A
30 DISP A
40 END
```

Hier wird in String A$ beginnend nach dem 2. Zeichen nach dem String COMPUTER gesucht.

Bildschirmanzeige nach dem Starten des Programms:

```
>RUN
 7 ..... Der String COMPUTER beginnt in String
READY    A$ bei dem 7. Zeichen.
>
```

● **FUNKTION 2:** Ermittelt den Index des Arrayelements mit dem gesuchten String.

FORMAT: SEARCH A$ (*), X, /C, A

A$(*) : Kollektive Variable des Stringarrays, das durchsucht werden soll

X : Index des Arrayelements, bei dem die Suche beginnen soll

C : Der aufzufindende String

A : Numerische Variable, weldcher der Index des Arrayelements mit dem gesuchten String zugewiesen wird

ERLÄUTERUNG:

Die Anweisung SEARCH sucht nach dem ersten Stringarrayelement, dessen Inhalt mit dem gesuchten String C identisch ist, und beginnt die Suche bei dem X-ten Element. Ist es gefunden, dann ordnet sie der numerischen Variablen A den Index dieses Elements zu. Die Suche erfolgt in der Reihenfolge der Elemente in dem Array. (Siehe DIM-Anweisung Seite 128).

Wurde String C nicht gefunden, wird der Variablen A der Wert 0 zugewiesen.

EINFACHES BEISPIEL:

```
10 DIM A$(4)
20 A$(1)="HANS"
30 A$(2)="MAX"
40 A$(3)="PAUL"
50 A$(4)="HANS"
60 SEARCH A$(*),2,"HANS",X
70 DISP "DAS";X;"-TE ELEMENT IST ";A$(X)
80 END
```

Wir haben zunächst ein Stringarray erzeugt, dessen Elemente vier Namen sind. Darauf wenden wir die SEARCH-Anweisung an und suchen, beginnend mit dem Zweiten Arrayelement, das Element, das den Namen HANS enthält. Wir starten dieses Programm und erhalten dann die Bildschirmanzeige:

```
>RUN
DAS 4 -TE ELEMENT IST HANS
READY
>
```

Da die Suche bei dem zweiten Element begann, bleibt das erste Element mit HANS unberücksichtigt.

● **FUNKTION 3:** Sucht in einem Stringarray nach einem String, der bestimmte Zeichen an einer bestimmten Stelle enthält.

FORMAT: SEARCH A$ (*), X, /C, A

A$(*) : Kollektive Variable des Stringarrays, das durchsucht werden soll

X : Index des Arrayelements, bei dem die Suche beginnen soll

C : Spezifiziert den zu suchenden String durch Angabe der Zeichen und ihrer relativen Position innerhalb des Strings. Wir geben für C einige Beispiele mit ihrer Auswirkung auf die SEARCH-Anweisung:

"A–" : Gesucht wird ein String, der mit A beginnt

"–B" : Gesucht wird ein String, der mit B aufhört

"–C–": Gesucht wird ein String, der den Buchstaben C enthält

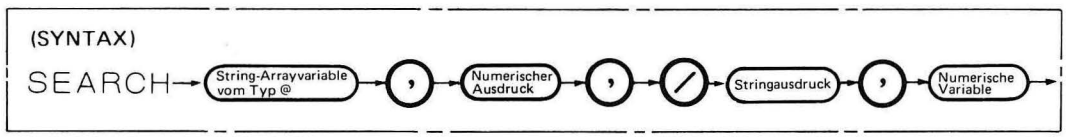
"*E*": Gesucht wird ein String aus 3 Zeichen mit dem Buchstaben E in der Mitte

"*F– ": Gesucht wird ein String, dessen zweites Zeichen ein F ist

"–K**": Gesucht wird ein String, dessen drittletzter Buchstabe ein K ist.

"***W": Gesucht wird ein String, der aus 4 Zeichen besteht und an letzter Stelle ein W enthält

Wie man sieht, steht "–" für einen beliebigen String (könnte auch ein Nullstring sein) und * für ein beliebiges Zeichen.



ERLÄUTERUNG:

Die SEARCH-Anweisung durchsucht die kollektive Stringarrayvariable A$(*) ab dem X-ten Element und ordnet der numerischen Variablen A den Index desjenigen Arrayelements zu, das als erstes der vorgegebenen Stringspezifikation entspricht.

EINFACHES BEISPIELE:

Nehmen wir an, wir hätten ein Stringarray mit den 5 Elementen: A$(1) = "ADAM", A$(2) = "PETER", A$(3) = "HANS", A$(4) = "ULRICH", A$(5) = "SIEGFRIED".

```
20 SEARCH A$(*),1,/"U–",A
```
Die Anweisung sucht ein Arrayelement, das mit U beginnt. Der numerischen Variablen A würde daher der Wert 4 zuwiesen.

```
30 SEARCH A$(*),1,/"–R",A
```
Die Anweising sucht ein Arrayelement, das mit R endet. Der Variablen A würde daher der Wert 2 zugewiesen.

```
50 SEARCH A$(*),1,/"*A–",A
```
Die Anweisung sucht ein Arrayelement, an dessen zweiter Stelle ein A steht. Der Variablen A würde daher der Wert 3 zugewiesen.

```
40 SEARCH A$(*),1,/"A*F*",A
```
Die Anweisung sucht ein Arrayelement, das aus vier Zeichen besteht und dessen erstes und drittes Zeichen ein A ist. Der Variablen A würde somit der Wert 1 zugewiesen.

```
70 SEARCH A$(*),1,/"B-",A
```
..... Die Anweisung sucht ein Arrayelement, an dessen erster Stellen ein B steht. Da kein solches Element in A$(*) existiert, wird der Variablen A der Wert 0 zugewiesen.

ANWENDUNGSBEISPIELE:

1. Das erste Programm wandelt eine Hexadezimalzahl in eine Dezimalzahl um:

```
 10 DIM A$(1)*16
 20 A$(1)="123456789ABCDEF0"
 30 INPUT X$
 40 X=LEN X$
 50 IF X>4 THEN GO TO 30 ..... Maximale Eingabe von 4 Ziffern.
 60 DISP X$;
 70 N=0
 80 FOR I=1 TO X
 90 MOVE C$,X$,I,1 ..... Ordnet C$ ein Zeichen des Eingabestrings zu.
100 SEARCH A$(1),1,C$,A ..... Ordnet A die Positionsziffer zu, wo C$ in A$ gefunden wurde.
110 IF A=0 THEN DISP "FEHLER":END ... Wurde C$ in A$ nicht gefunden, endet das Programm mit einer Fehlermeldung.
120 IF A=16 THEN A=0
130 N=N+A*16^(X-I)
140 NEXT I
150 DISP N
160 END
```

Die Eingabe von 1A4E (hexadezimal) verursacht folgende Bildschirmausgabe:

```
>RUN
1A4E
!A4E 6734 ..... Ausgabe des Resultats der Hexadezimal-Dezimal-Umwandlung.
READY
>
```

2. Das zweite Programm liefert den Index des Arrayelements, das ein gesuchtes Zeichen "E" enthält.

```
10 DIM X$(2,3)
20 X$(1,1)="A":X$(1,2)="B":X$(1,3)="C"
30 X$(2,1)="D":X$(2,2)="E":X$(2,3)="F"
40 SEARCH X$(*),1,"E",X
50 DISP "X=";X
60 END
```

Bildschirmanzeige nach dem Starten des Programms:

```
>RUN
X= 5
READY
>
```

DEFFN

Definition einer Funktion durch das Benutzerprogramm.

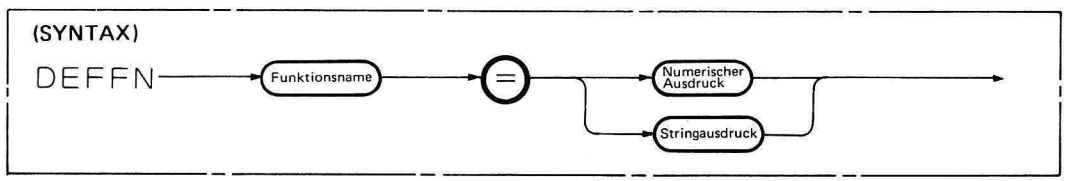
〈VERWENDUNG〉
Um eine Verknüpfung in Form numerischer Ausdrücke oder Strings als eine Funktion zu deklarieren und diese Funktion an beliebiger Stelle des Programms aufrufen zu können.

● **FUNKTION 1:** Funktion ohne Argument.

FORMAT: DEFFN SF = X | C

SF : Name der Funktion

X or C : Definition der Funktion



ERLÄUTERUNG:

Die DEF FN-Anweisung definiert den Inhalt des numerischen Ausdrucks X oder Strings C als eine Funktion mit dem Namen SF. Funktionen, die am Anfang eines Programms definiert worden sind, können auf dieselbe Art Verwendung finden wie Stringfunktionen (Seite 23) und arithmetische Funktionen (Seite 26). Die Funktion SF wird im Programm mit FN SF aufgerufen.

EINFACHES BEISPIELE:

1. `10 DEFFN A=8^10` ..... Definiert den numerischen Ausdruck 8^{10} als Funktion A.
2. `10 DEFFN B$="SHARP"` ..... Definiert den String SHARP als Funktion B$

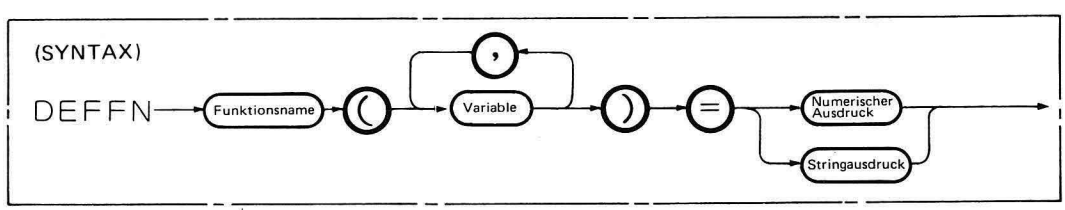
● **FUNKTION 2:** Funktion mit Argument(en).

FORMAT: DEFFN SF ((N)) = X | C

SF : Name der Funktion

N : Argument(e)

X or C : Definition der Funktion



ERLÄUTERUNG:

Die Anweisung DEF FN definiert den Inhalt des numerischen Ausdrucks X oder des Strings C als Funktion SF mit Argument, wenn in dem numerischen Ausdruck oder dem String ein oder mehrere Argumente (N) enthalten sind.

EINFACHES BEISPIELE:

```
10 DEFFN A(X,Y)=X^Y
```
Definiert den numerischen Ausdruck X^Y als Funktion mit den Argumenten X und Y.

```
20 DEFFN B$(C$)="GUTEN "+C$
```
Definiert den String "GUTEN+C$" als Funktion B$ mit dem Argument C$.

WICHTIGER HINWEIS

1. Diese Anweisung kann nicht mit einer Marke versehen werden.
2. Das Format des Arguments in der Definition der Funktion muß mit dem Format des Arguments in der FN-Anweisung (Aufruf der Funktion) übereinstimmen.
3. In einem Programm können seitens des Benutzers bis zu 42 Funktionen mittels der DEFFN-Anweisung oder Unterprogramme mittels der SUB-Anweisung (Seite 160) definiert werden.
4. Die Variablen in der Funktion FN sind unabhängig von den Variablen des Programms.
5. Funktionsnamen können analog Variablennamen gebildet werden.

ANWENDUNGSPROGRAMME:

Programm A:

Definition einer Funktion durch einen numerischen Ausdruck:

```
10 DEFFN A(X,Y)=X^2+2*X*Y-Y^2+1
20 INPUT Z1,Z2
30 M=FN A(Z1,Z2)
40 DISP "Z1=";Z1;
50 DISP "Z1=";Z2,
60 DISP "Z1^2+2*Z1*Z2-Z2^2+1=";M
70 END
```

Wir geben die Zahlen 5 und 3 ein.

Bildschirmanzeige:

```
>RUN
5                    3
X= 3 Y= 5            Z1^2+2*Z1*Z2-Z2^2+1= 15
READY
>
```

Das gleiche Ergebnis erhält man, wenn man schreibt:

```
20 Z1=3:Z2=5     anstat   20 INPUT Z1,Z2
                   oder
30 M=FN A(3,5)   anstat   30 M=FN A(Z1,Z2)
```

Programm B:
Definition einer Stringfunktion.

```
 10 DIM A$(3)
 20 A$(1)="MORGEN"
 30 A$(2)="TAG"
 40 A$(3)="ABEND"
 50 DEFFN B$(X$)="GUTEN "+X$
 60 INPUT N
 70 IF N<1 THEN GO TO 200
 80 IF N>3 THEN GO TO 200
 90 X$=A$(N)
100 C$=FN B$(X$)
110 DISP C$
120 GO TO 60
200 END
```

Eingabe: 1, 2, 3
Bildschirmausgabe:

```
>RUN
1
GUTEN MORGEN
2
GUTEN TAG
3
GUTEN ABEND
0
READY
>
```

MUSIC

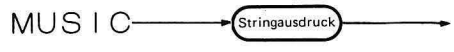
Ausgabe einer in Form eines Strings dargestellten Melodie oder Toneffekts über den eingebauten Lautsprecher.

〈VERWENDUNG〉
① Zum Gebrauch von Toneffekten oder -signalen in einem Programm.
② Zum Abspielen einer Melodie.

FORMAT: MUS I C C

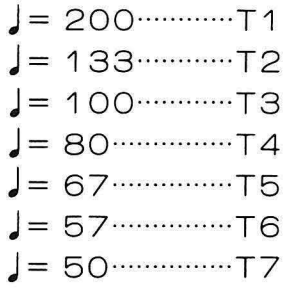
C : String

(SYNTAX)

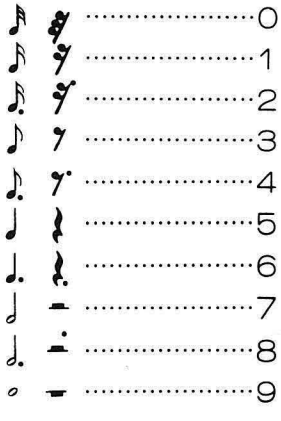


ERLÄUTERUNG:

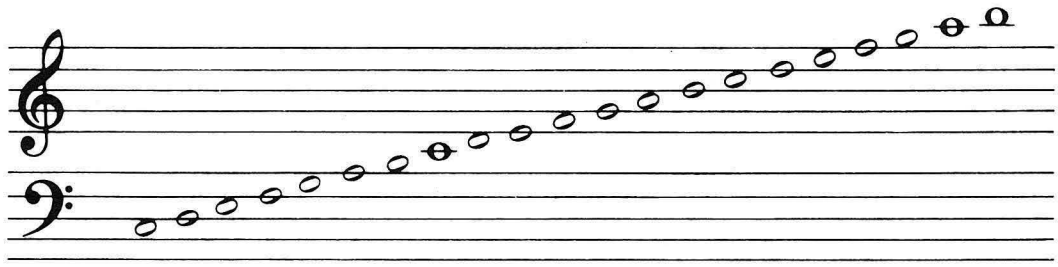
1. Die Anweisung MUSIC veranlaßt die Ausgabe einer Tonfolge (Melodie) in der durch den String C dargestellten Weise über den Lautsprecher.
   Die Darstellung in String C bezieht sich auf das Tempo für das Abspielen der Melodie, auf die Tonhöhe, auf die Dauer des jeweiligen Tons und auf Pausen zwischen den einzelnen Tönen, sowie deren Dauer.
   a. Am Anfang des Strings sollte das Tempo für das Abspielen der Melodie spezifiziert werden. (Fehlt eine Spezifikation, wird automatisch Tempo T4 eingestellt.)



   b. Vor dem Symbol für die Tonhöhe (=Note) muß das Zeichen für die Tondauer stehen (gilt ebenso für die Kennzeichnung der Dauer einer Pause). Folgen mehrer Töne mit gleicher Tondauer aufeinander, dann genügt es, wenn man den ersten Ton dieser Reihe mit dem entsprechenden Tondauersymbol versieht.



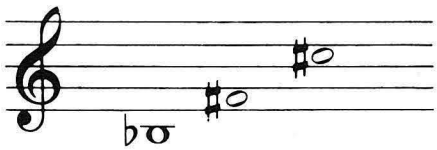
c. Die Tonhöhe erstreckt sich über drei Oktaven. Die Symbole für die einzelnen Töne ergeben sich aus der folgenden Abbildung.



LD LF LA C E G B UD UF UA

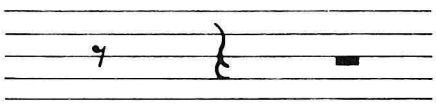
LC LE LG LB D F A UC UE UG UB

d. Für die Darstellung von Halbtönen werden die Zeichen # und $ benutzt. # bedeutet die Erhöhung des angegebenen Tones um einen Halbton, $ seine entprechende Verminderung.



$LB #F #UC

e. Soll ein Tonfolge durch eine Pause unterbrochen werden, wird dies durch das Zeichen R angezeigt. Die Spezifikation ihrer Dauer erfolgt analog der Darstellung der Tondauer.



3R 5R 9R

2. Da ein String nur bis zu 255 Zeichen enthalten kann, sollte man zur Darstellung längerer Melodien mehrere Anweisungen benutzen.
3. Die Lautstärke kann am Lautstärkeregler auf der Frontseite des Geräts unterhalb des Lautsprechers eingestellt werden.

ANWENDUNGSBEISPIEL:

Das folgende Programm spielt die ersten Takte aus Beethovens Klavierkomposition "Für Elise":

```
10 MUSIC "T22UE#UDUE#UDUEBUDUC4A2CEA4B2E#GB4UC2EUE#
   UDUE#UDUEBUDUC4A2CEA4B2EUCB4A"
20 MUSIC "T22BUCUD5UE2GUFUE5UD2FUEUD5UC2EUDUC5B1R2E
   4UE2UE4UE3R"
30 MUSIC "T22UE#UDUE#UDUEBUDUC4A2CEA4B2E#GB4UC2EUE#
   UDUE#UDUEBUDUC4A2CEA4B2EUCB6A"
40 END
```

WICHTIGER HINWEIS

Bei der Eingabe der MUSIC-Anweisung können bis zu 250 Zeichen in eine Zeile geschrieben werden; längere Kompositionen muß man auf mehrere Zeilen verteilen.

WAIT

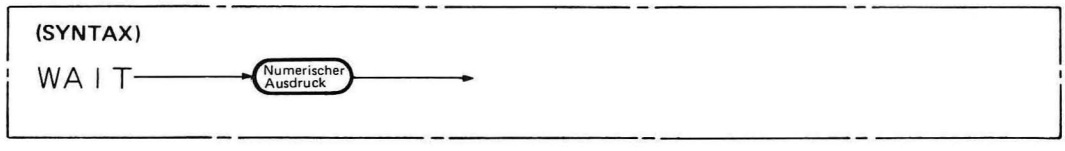
Unterbricht die Programmausführung oder wartet mit der Programmausführung bis zu einer vorgegebenen Zeit.

〈VERWENDUNG〉
1. Zur Unterbrechung der Programmausführung für die zeitlich befristete Ausgabe von Zwischenresultaten.
2. Zum Starten einer Programmausführung zu einem genauen Zeitpunkt.

● **FUNKTION 1:** Unterbricht ein Programm für einen genau definierten Zeitraum.

FORMAT: WA I T X

X: Unterbrechungszeit (in Zehntelsekunden d.h. 1 = 0.1 Sek)



ERLÄUTERUNG:

Unterbricht die Programmausführung für eine Zehntelsekunde. (WAIT 1 bewirkt eine Unterbrechung von 0.1 Sekunden).

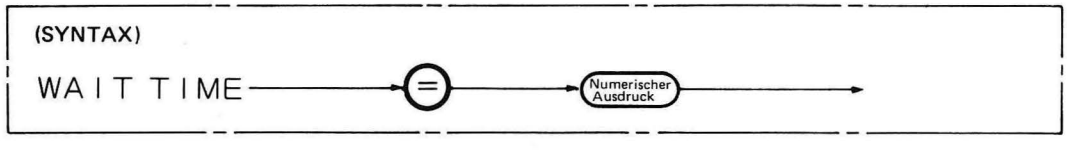
EINFACHES BEISPIEL:

```
10 WAIT 30
```
..... Diese Anweisung unterbricht die Programmausführung für 3 Sekunden.

● **FUNKTION 2:** Wartet mit der Programmausführung bis zum angegebenen Zeitpunkt.

FORMAT: WA I T T I ME = X

X: Zeitpunkt für den Ausführungsbeginn (Stunden. Minuten Sekunden)



ERLÄUTERUNG:

Wartet mit der Programmausführung bis zu der durch X angegebenen Zeit. X ist eine Festkommazahl, deren ganzzahliger Teil die Stunde angibt. Die ersten beiden Nachkommastellen geben die Minuten, die dritte und vierte Nachkommastelle die Sekunden an. (14.1030 bedeutet z.B. 14 Uhr 10 Minuten und 30 Sekunden).

Steht die WAIT-Anweisung am Anfang eines Programms, dann wird die Programmausführung zum vorgegebenen Zeitpunkt aufgenommen (vorausgesetzt, das Programm wurde überhaupt gestartet). Steht sie innerhalb eines Programms, dann wird die Programmausführung ab der nächsten Zeile bis zu dem vorgegebenen Zeitpunkt unterbrochen. Dine Verzögerung bei der Programmausführung findet nicht statt, wenn der angegebene Zeitpunkt bereits vergangen ist.

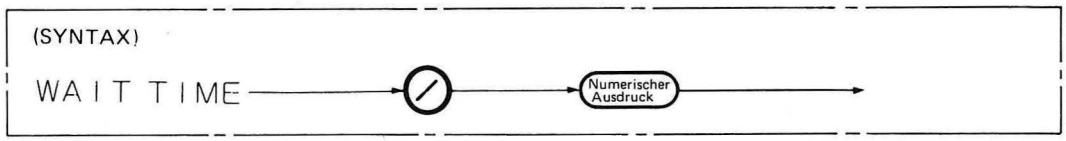
EINFACHE BEISPIEL:

```
10 WAIT TIME=17.30
```
..... Wartet mit der Programmausführung bis 17 Uhr 30.

● **FUNKTION 3:** Wie FUNKTION 2 jedoch mit Zeitausgabe.

FORMAT: WA I T T I ME / X

X : Zeitpunkt für den Ausführungsbeginn (Stunden. Minuten Sekunden)



ERLÄUTERUNG:

Gibt während der Wartezeit bis zur Programmausführung in der augenblickliche Kursorposition die laufende Uhrzeit in AM/PM-Notation aus.

EINFACHES BEISPIEL:

```
10 WAIT TIME/10.0935
```
..... Gibt bis zum Beginn der Programmausführung um 10 h 9 m 35 s die laufende Uhrzeit am Bildschirm aus.

WICHTIGER HINWEIS

Diese Anweisung ist ungültig, wenn der angegebene Zeitpunkt schon vergangen ist.

<PRAKTISCHER TIP>

Das Programm kann beendet und damit die Ausführung der WAIT-Anweisung unterbrochen werden, wenn man den HALT-Knopf drückt.

ANWENDUNGSBEISPIEL:

Ausgabe der Zahlen 1 bis 10 im Abstand von einer Sekunde.

```
 10 DISP CLEAR ..... Löschen des Bildschirms.
 20 N=0
 30 N=N+1
 40 IF N>10 GO TO 100
 50 CURSOR 20,20 ..... Positioniert den Kursor in den Koordinaten (20,20).
 60 DISP N ..... Ausgabe der Zahl.
 70 WAIT 10 ..... Wartet eine Sekunde.
 80 GO TO 30
100 END
```

DEFINT

Zur Deklaration von Variablen als INTEGER-Variablen.

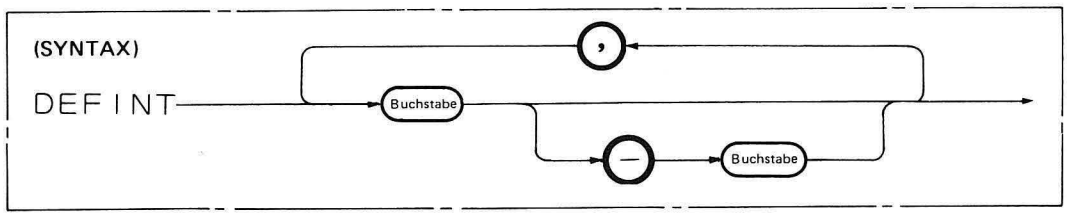
〈VERWENDUNG〉

① Zum Verwenden von Variablen als Variablen vom Typ INTEGER, ohne sie mit dem zusätzlichen Symbol % versehen zu müssen.

② Zur globalen Verwendung von Variablen, die mit einem bestimmten Buchstaben beginnen, als Variablen vom Typ INTEGER.

FORMAT: DEFINT [CH [-DH]]

CH (and DH): Buchstaben, mit denen ein Variablenname beginnt



ERLÄUTERUNG:

1. Erklärt alle Variablen (einfache numerische und Arrayvariablen) als INTEGER-Variablen, die mit dem Buchstaben CH oder mit einem Buchstaben, der in der alphabetischen Reihenfolge zwischen CH und DH liegt (beide eingeschlossen), beginnen. Dies gilt für Großbuchstaben.
2. Setzt man für CH einen kleinen Buchstaben, z.B. den Buchstaben a, dann werden automatisch alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als INTER-Variablen angesehen.

EINFACHES BEISPIELE:

1. `10 DEFINT A` ..... Deklariert alle Variablen, deren Name mit A beginnt, als INTEGER-Variablen.
2. `10 DEFINT I-N` ..... Deklariert alle Variablen, deren Name mit I, J, K, L, M beginnt, als INTEGER-Variablen.
3. `10 DEFINT a` ..... Deklariert alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als INTEGER-Variablen.
4. `10 DEFINT I,L-N,a` ..... Deklariert alle Variablen, deren Name mit I, L, M, N oder einem kleinen Buchstaben beginnt, als INTEGER-Variablen.

WICHTIGER HINWEIS

1. Auf die Eigenschaft einer DEFINT-Anweisung, im Falle der Variablendeklaration bezüglich eines Kleinbuchstabens, alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als Variablen des Typs INTEGER zu behandeln, ist sorgfältig zu achten.
2. Alle Variablen, die mittels der DEFINT-Anweisung deklariert wurden, werden ohne Anhängen des %-Symbols als INTEGER-Variablen behandelt.
3. Die Typdeklaration mit #, $ und § besitzt Priorität vor der Variablendeklaration mit der DEFINT-Anweisung.
   Auch wenn am Anfang eines Programms die Deklaration DEFINT I-N erfolgte, so werden trotzdem I# als reelle Variable, J$ und M§ als Stringvariable behandelt.

DEFREAL

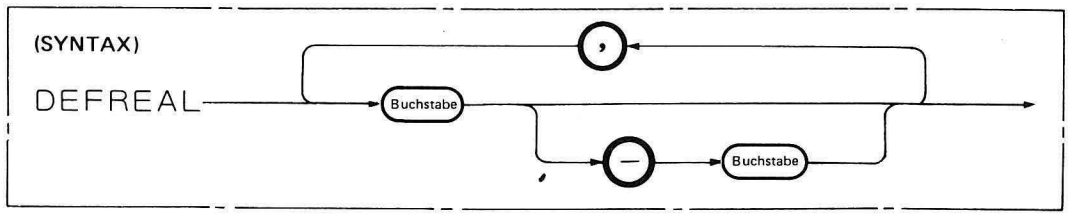
Zur Deklaration von Variablen als REAL-Variablen.

〈VERWENDUNG〉

1. Zum Verwenden von Variablen als REAL-Variablen, ohne sie mit dem zusätzlichen Symbol # versehen zu müssen.
2. Zur globalen Verwendung von Variablen, die mit einem bestimmten Buchstaben beginnen, als REAL-Variablen.

FORMAT: DEFREAL [CH [-DH]]

CH [und DH] : Buchstaben, mit denen ein Variablenname beginnt



ERLÄUTERUNG:

1. Erklärt alle Variablen (einfache numerische und Arrayvariablen) als REAL-Variablen, die mit dem Buchstaben CH oder mit einem Buchstaben beginnen, der in der alphabetischen Reihenfolge zwischen CH und DH (beide eingeschlossen) liegt. Dies gilt für Großbuchstaben.
2. Setzt man für CH einen kleinen Buchstaben, z.B. den Buchstaben a, dann werden automatisch alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als REAL-Variablen angesehen.

EINFACHES BEISPIELE:

1. `10 DEFREAL A-H` ..... Deklariert alle Variablen, deren Name mit A, B, C, D, E, F, G, H beginnt, als REAL-Variablen.
2. `10 DEFREAL a` ..... Deklariert alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als REAL-Variablen.
3. `10 DEFREAL A-C,Q,X-Z` ..... Deklariert alle Variablen, deren Name mit A, B, C, Q, X, Y, Z beginnt, als reelle Variablen.

HINWEIS

1. Auf die Eigenschaft der DEFREAL-Anweisung, im Falle der Variablendeklaration bezüglich eines Kleinbuchstabens, alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als REAL-Variablen zu behandeln, ist sorgfältig zu achten.
2. Alle Variablen, die mittels der DEFREAL-Anweisung deklariert wurden, werden ohne Anhängen des #-Symbols als REAL-Variablen behandelt
3. Die Typdeklaration mit %, $ und § besitzt Priorität vor der Variablendeklaration mit der DEFREAL-Anweisung.
   Auch wenn am Anfang eines Programms die Deklaration DEFREAL I–N erfolgte, so werden trotzdem I% als INTEGER-Variable, J$ und M§ als Stringvariablen behandelt.

DEFSTR

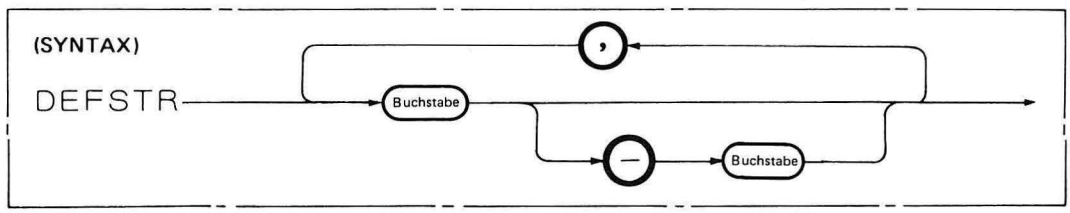
Zur Deklaration von Variablen als Stringvariablen.

〈VERWENDUNG〉
1. Zum Verwenden von Variablen als Stringvariablen, ohne sie mit dem zusätzlichen Symbol $ oder § versehen zu müssen.
2. Zur globalen Verwendung von Variablen, die mit einem bestimmten Buchstaben beginnen, als Stringvariablen.

● **FUNKTION 1:** Deklaration von Stringvariablen des $-Typs.

FORMAT:DEFSTR 〔CH 〔−DH〕〕

CH (und DH): Buchstaben, mit denen ein Variablenname beginnt



ERLÄUTERUNG:

1. Erklärt alle Variablen (einfache und Arrayvariablen) zu Stringvariablen des $-Typs, die mit dem Buchstaben CH oder mit einem Buchstaben beginnen, der in der alphabetischen Reihenfolge zwischen CH und DH (beide eingeschlossen) liegt. Dies gilt für Großbuchstaben.
2. Setzt man für CH einen kleinen Buchstaben, z.B. den Buchstaben a, dann werden automatisch alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als Stringvariablen angesehen.

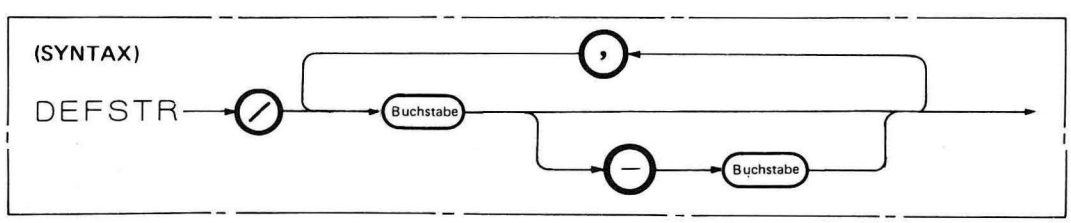
EINFACHES BEISPIELE:

1. `10 DEFSTR C` ..... Deklariert alle Variablen, deren Name mit dem Buchstaben C beginnt, als Stringvariablen des $-Typs.
2. `10 DEFSTR A-Z` ..... Deklariert alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als Stringvariablen des $-Typs.

● **FUNKTION 2:** Deklaration von Stringvariablen des @-Typs.

FORMAT:DEFSTR / 〔CH 〔−DH〕〕

CH (und DH): Buchstaben, mit denen ein Variablenname beginnt



ERLÄUTERUNG:

Erklärt alle Variablen (einfache und Arrayvariablen) zu Stringvariablen des @-Typs, die mit demBuchstaben CH beginnen oder alle Variablen, die mit einem Buchstaben beginnen, der in der alphabetischen Reihenfolge zwischen CH und DH (beide eingeschlossen) liegt. Näheres siehe FUNKTION 1.

EINFACHES BEISPIELE:

1. `10 DEFSTR/S` ..... Deklariert alle Variablen, deren Name mit dem Buchstaben C beginnt, als Stringvariablen des @-Typs.
2. `10 DEFSTR/u` ..... Deklariert alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als Stringvariablen des @-Typs.

HINWEIS

1. Auf die Eigenschaft der DEFSTR-Anweisung, im Falle der Variablendeklaration bezüglich eines Kleinbuchstabens, alle Variablen, deren Name mit einem kleinen Buchstaben beginnt, als Variablen des Typs String zu behandeln, ist sorgfältig zu achten.
2. Alle Variablen, die mittels der DEFSTR-Anweisung deklariert wurden, werden ohne Anhängen des $ oder @-Symbols als Stringvariablen behandelt.
3. Die Typdeklaration mit % oder # besitzt Priorität vor der Variablendeklaration mit der DEFSTR-Anweisung.

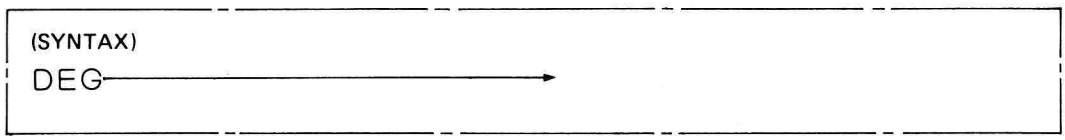
DEG

Wählt GRAD als Einheit der Winkelmessung in trigonometrischen Funktionen.

〈VERWENDUNG〉
Zur Angabe der Winkel in trigonometrischen Funktionen im Gradmaß.

FORMAT: DEG

(SYNTAX)



ERLÄUTERUNG:

1. Die DEG-Anweisung veranlaßt das System, bei Winkeln, die im Zusammenhang mit trigonometrischen Funktionen angegeben werden, Gradmaß anzunehmen. DEG hebt die Wirkung vorausgegangener RAD-Anweisungen wieder auf (Seite 211).
2. Im Normalmodus, nach Einschalten des Geräts, arbeitet das System im Gradmaß.

ANWENDUNGSBEISPIEL:

Ausgabe des Sinuswertes für einen Winkel von 30 Grad.

```
10 DEG ..... Wählt als Winkeleinheit GRAD.
20 A=SIN 30 ..... Berechnet den Sinus von 30 Grad.
30 DISP "sin 30 Grad =";A
40 END
```

Bildschirmausgabe:

```
>RUN
sin 30 Grad = 0.5
READY
>
```

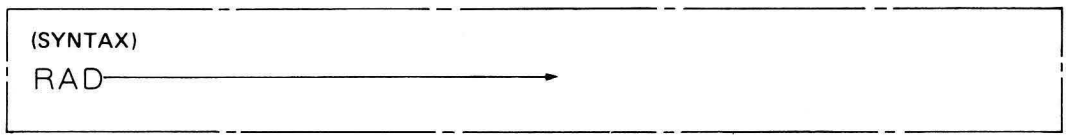
RAD

Wählt Radian als Einheit der Winkelmessung in trigonometrischen Funktionen.

〈VERWENDUNG〉
Zur der Angabe der Winkel in trigonometrischen Funktionen im Bogemaß (Radian).

FORMAT: RAD

(SYNTAX)



ERLÄUTERUNG:

Die DEG-Anweisung veranlaßt das System, bei Winkeln, die im Zusammenhang mit trigonometrischen Funktionen angegeben werden, Bogenmaß anzunehmen. Da das System nach dem Einschalten des Geräts im Gradmaß operiert, ist die Eingabe der RAD-Anweisung erforderlich, wenn im Bogenmaß gerechnet werden soll.

ANWENDUNGSBEISPIEL:

Ausgabe des Sinauswertes für einen Winkel von PI/6 in RAD.

```
10 RAD ..... Wählt als Winkeleinheit RAD.
20 A=SIN (PI/6) ..... Berechnet den Sinus von PI/6 in RAD.
30 DISP "sin(π/6)=";A
40 END
```

Bildschirmausgabe:

```
>RUN
sin(π/6)= 0.5
READY
>
```

RANDOM

Erzeugt eine Zufallszahl als Anfangswert bei der Initialisierung des Zufallszahlengenerators.

〈VERWENDUNG〉
Wird vor der Verwendung der Funktion RND (Seite 29), die der Erzeugung von Zufallszahlen dient, empfohlen. (Siehe Erläuterung.)

FORMAT: RANDOM

(SYNTAX)
RANDOM→

ERLÄUTERUNG:

1. Erzeugt eine Zuhallszahl als Anfangswert für den Zufallszahlengenerator. Diese Anweisung wird von der ersten Verwendung der RND-Funktion empfohlen. So wird vermieden, daß die RND-Funktion jedes mal nach dem Laden des BASIC-Interpreters den gleichen Anfangswert bei dem Generieren von Zufallszahlen verwendet und daher auch die gleiche Folge von Zufallszahlen liefert.
2. Diese Anweisung kann in der manuellen Betriebsart und innerhalb eines Programms benutzt werden.

ANWENDUNGSBEISPIELE:

Programm A:

Ausgabe einer Folge von 10 ganzzahligen Zufallszahlen aus dem Intervall 0 bis 9.

```
10 RANDOM ..... Erzeugt eine Zufallszahl als Anfangswert für den Zufallszahlengenerator.
20 FOR I=1 TO 10 ..... Erzeugt Zufallszahlen unter 10 und ordnet der Variablen
30 A=INT (RND 10)      A den ganzzahligen Teil zu.
40 DISP A ..... Zeigt Werte der Variablen A an.
50 NEXT I
60 END
```

Dies Programm liefert 10 Zufallszahlen. Wird die Anweisung RANDOM in Zeile 10 entfernt, dann würde dieses Programm jedesmal nach dem Laden von BASIC die gleiche Zahlenfolge liefern.

Programm B:

Dieses Programm schaltet in den invertierenden Bildmodus um und füllt auf dem Bildschirm ein Rechteck mit zufällig gesetzten Punkten.

```
10 DISP CLEAR
20 RV 0,0,2000
30 X=INT (RND 80)+40
40 Y=INT (RND 25)+12
50 SET X,Y
60 GO TO 30
```

ERROR MUSIC

Zum Ein- und Ausschalten eines akustischen Signals beim Auftreten eines Fehlers.

〈VERWENDUNG〉
Tritt ein Fehler auf, ertönt ein akustisches Signal und am Bildschirm wird eine Fehlermeldung ausgegeben.
Mit dieser Anweisung kann man das Tonsignal ein- und ausschalten.

● **FUNKTION 1:** Schaltet das Tonsignal bei Auftreten eines Fehlers aus.

FORMAT:ERROR MUS I C OFF

(SYNTAX)
ERROR MUS I C OFF⟶

ERLÄUTERUNG:

Nach dem Einschalten des Geräts wird bei Auftreten eines Fehlers ein akustisches Fehlersignal ausgegeben. Es unterbleibt, wenn diese Anweisung eingegeben worden ist.

● **FUNKTION 2:** Zum Wiedereinschalten des akustichen Signals beim Auftreten eines Fehlers.

FORMAT:ERROR MUS I C ON

(SYNTAX)
ERROR MUS I C ON⟶

ERLÄUTERUNG:

Wurde das akustiche Fehlersignal mittels der ERROR MUSIC OFF-Anweisung ausgeschaltet, dann kann es durch diese Anweisung wieder eingeschaltet werden. (Diese Anweisung erzeugt keine Fehlermeldung, wenn das Signal nicht ausgeschaltet war.)

5. BILDSCHIRM- UND DRUCKERANWEISUNGEN

DISP CLEAR
SET
RESET
POS
ENTER
CURSOR
MVCUR
LINE
BL
RV
RE
TABLE
SCROL
CHANGE DISP
CHANGE PRINT
COLOR

Diese Anweisungen sollen im folgenden Kapitel beschrieben werden.
Alle diese Anweisungen können auch in der manuellen Betriebsart des Systems verwendet werden.
Bis auf die Anweisungen DISP CLEAR, CURSOR (in Funktion 1) und MVCUR verändern diese Anweisungen die aktuelle Kursorposition nicht.

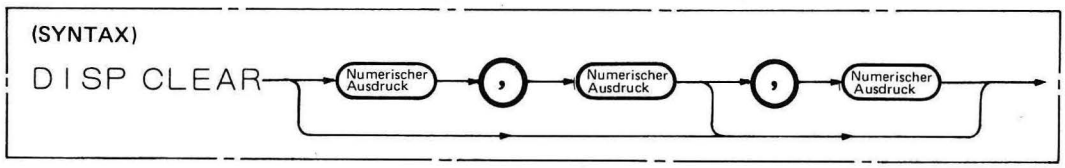
DISP CLEAR

Löscht den Bildschirm.

〈VERWENDUNG〉
Zur besseren Gestaltung der Bildschirmausgabe und zum Löschen überflüssiger Anzeigen.

FORMAT:D I SP CLEAR 〔X, Y 〔, Z〕 〕

X : X-Koordinate des ersten der zu löschenden Zeichen
Y : Y-Koordinate des ersten der zu löschenden Zeichen
Z : Anzahl der zu löschenden Zeichen

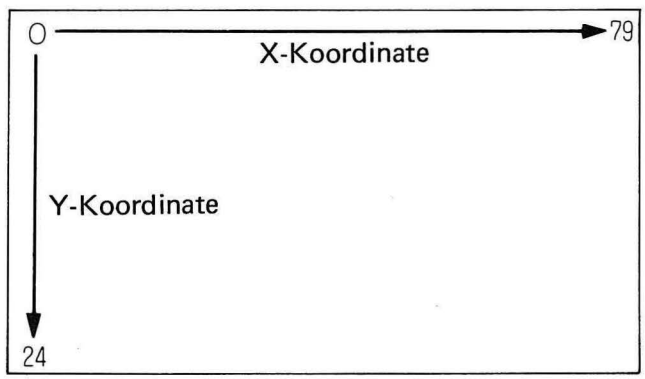


ERLÄUTERUNG:

Die DISP CLEAR-Anweisung löscht insgesamt Z Zeichen beginnend bei der durch die Textkoordinaten (X, Y) bezeichneten Position. Nach dem Löschen befindet sich der Kursor an der Position des ersten der gelöschten Zeichen. Der Wertebereich für die numerischen Ausdrücke X und Y hängt von dem für die Ausgabe gewählten Zeichen-Modus ab. Er beträgt:

- im 80 x 25 Zeichen-Modus
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 24$ (Y: 0 – 24)
- im 80 x 20 Zeichen-Modus
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 19$ (Y: 0 – 19)
- im 40 x 25 Zeichen-Modus
  $0 \leqq X \leqq 39$ (X: 0 – 39)
  $0 \leqq Y \leqq 24$ (Y: 0 – 24)
- im 40 x 20 Zeichen-Modus
  $0 \leqq X \leqq 39$ (X: 0 – 39)
  $0 \leqq Y \leqq 19$ (Y: 0 – 19)

Das System der Textkoordinaten verdeutlicht die folgende Abbildung am Beispiel des 80 x 25 Zeichen-Modus:



2. Fehlt in der DISP CLEAR-Anweisung das Argument für Z, dann wird automatisch der ganze Bildschirm ab dem Textkoordinatenpunkt (X, Y) gelöscht. Der Kursor befindet sich anschließend im Koordinatenpunkt (X, Y).
3. Fehlen neben dem Argument für Z auch Angaben zu den Textkoordinaten X und Y, dann wird der ganze Bildschirm gelöscht. Der Kursor befindet sich anschließend in der HOME-Position (0, 0).

EINFACHES BEISPIELE:

1. `10 DISP CLEAR 0,4,80` ···· Löscht 80 Zeichen vom Textkoordinatenpunkt (0, 4), m. a. W. löscht die 5. Zeile.
2. `10 DISP CLEAR 0,10` ····· Löscht den Bildschirm ab dem Textkoordinatenpunkt (0, 10).
3. `10 DISP CLEAR` ····· Löscht den ganzen Bildschirm.

WICHTIGER HINWEIS

Diese Anweisung löscht nicht graphische Anzeigen (Seite 31), Markierungslinien (Seite 226), blinkende (Seite 228), invertierte (Seite 229) und farbige Anzeigen (Seite 239). Für diese Anzeigen bestehen eigene Löschbefehle.

Markierungslinien, blinkende, invertierte und farbige Anzeigen können aber durch die Tasten CTRL und HOME gelöscht werden (Drücken und Halten der CTRL -Taste und dann Drücken der HOME -Taste).

Die graphische Anzeige kann mittels der CLS-Anweisung (Seite 247) gelöscht werden.

SET

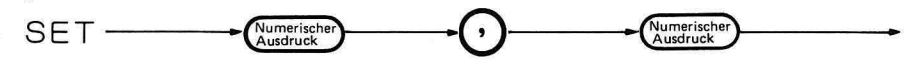
Setzt einen Punkt in der angegebenen Koordinatenposition des Bildschirms.

〈VERWENDUNG〉
Zum Erzeugen von Punkten und Linien in der Textebene ohne Benutzung der Graphikanzeige.

FORMAT: SET X, Y

X : X-Koordinate
Y : Y-Koordinate

(SYNTAX)

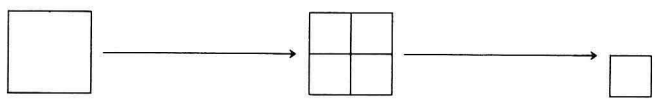


ERLÄUTERUNG:

Die SET-Anweisung setzt einen Punkt an den Bildschirmkoordinaten (X, Y).
Ergeben die numerischen Ausdrücke für X und Y keine ganzen Zahlen, werden die Dezimalteile der sich ergebenden Werte abgeschnitten.
Der Wertebereich für X und Y hängt von dem für die Ausgabe gewählten Zeichen-Modus ab. Er beträgt:

- im 80 x 25 Zeichen-Modus
  $0 \leqq X \leqq 159$ (X: 0 – 159)
  $0 \leqq Y \leqq 49$ (Y: 0 – 49)
- im 80 x 20 Zeichen-Modus
  $0 \leqq X \leqq 159$ (X: 0 – 159)
  $0 \leqq Y \leqq 39$ (Y: 0 – 39)
- im 40 x 25 Zeichen-Modus
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 49$ (Y: 0 – 49)
- im 40 x 20 Zeichen-Modus
  $0 \leqq X \leqq 79$ (X: 0 – 79)
  $0 \leqq Y \leqq 39$ (Y: 0 – 39)

Die Zahl der Koordinatenpunkte bei der SET-Anweisung verdoppelt sich gegenüber dem Umfang der Koordinatenpunkte im normalen Textmodus.



Koordinateneinheit im Textmodus (1 Zeichen)

Koordinateneinheit bei der SET-Anweisung

EINFACHES BEISPIEL:

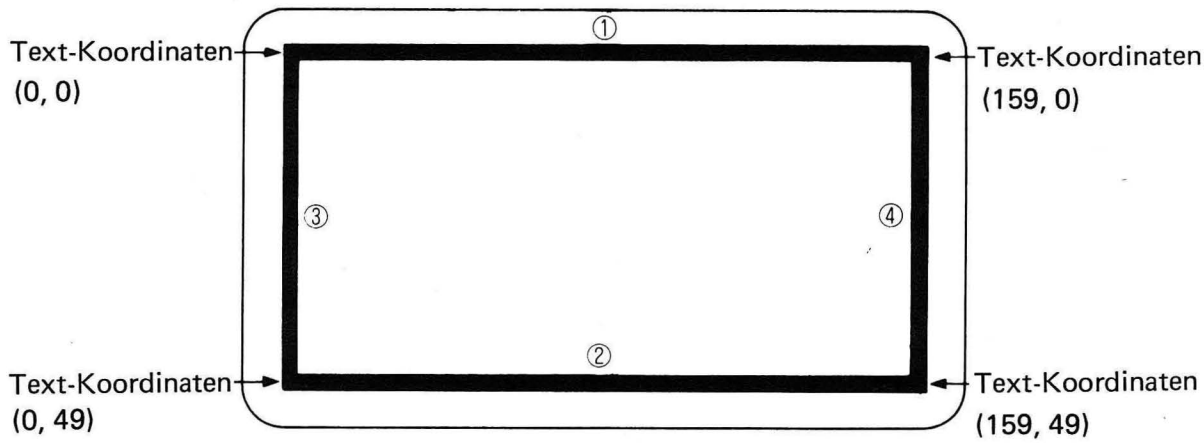
```
10 SET 4,6
```
..... Setzt einen Punkt im Koordinatenpunkt (4, 6).

ANWENDUNGSBEISPIEL:

Dieses Programm zeichnet ein Rechteck entlang der Kanten des Bildschirms.

```
10 FOR X=0 TO 159
20 SET X,0 ..... Zeichnet die obere Kante des Rechtecks.
30 SET X,49 ..... Zeichnet die untere Kante des Rechtecks.
40 NEXT X
50 FOR Y=0 TO 49
60 SET 0,Y ..... Zeichnet die linke Seite des Rechtecks.
70 SET 159,Y ..... Zeichnet die rechte Seite des Rechtecks.
80 NEXT Y
90 END
```

Bildschirmanzeige:



(Die Linien werden in der Reihenfolge ①, ②, ③, ④ gezeichnet.)

RESET

Löscht einen Punkt in der angegebenen Koordinatenposition des Bildschirms.

〈VERWENDUNG〉

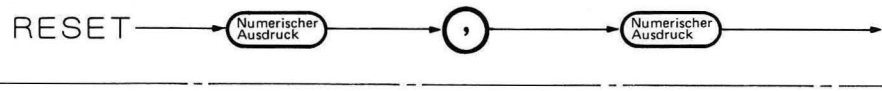
Zum Löschen von Punkten und Linien, die mit der SET-Anweisung gesetzt worden sind.

FORMAT: RESET X, Y

X: X-Koordinate

Y: Y-Koordinate

(SYNTAX)



ERLÄUTERUNG:

Die RESET-Anweisung löscht den Punkt (setzt ihn zurück) in der Koordinatenposition (X, Y), der durch die SET-Anweisung gesetzt worden ist. Die zu der SET-Anweisungen (Seite 217) gemachten Ausführungen gelten hier analog.

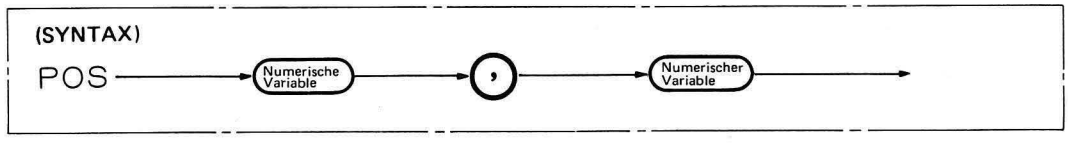
POS

Weist die Koordinatenwerte der augenblicklichen Kursorposition den spezifizierten Variablen zu.

〈VERWENDUNG〉
Für die übersichtliche Gestaltung der Bildschirmausgabe im Zusammenhang mit der Verwendung der CURSOR-Anweisung.

FORMAT: POS A, B

A : Numerische Variable für die Zuordnung des Wertes der X-Koordinate
B : Numerische Variable für die Zuordnung des Wertes der Y-Koordinate



ERLÄUTERUNG:

Die POS-Anweisung bewirkt die Zuweisung des augenblicklichen Wertes der X-(Text) Koordinate an die Variable A und die entsprechende Zuweisung des Wertes der Y-Koordinate an die Variable B.

EINFACHES BEISPIEL:

```
10 POS M,N
```

Machen wir die Annahme, der Kursor befinde sich im Moment der Ausführung dieser Anweisung im Koodinatenpunkt (20, 5). Nach Ausführung dieser Anweisung hat dann die Variable M den Wert 20 und die Variable N den Wert 5.

ANWENDUNGSBEISPIEL:

Der am Bildschirm ausgegebenen String "SHARP CORPORATION" soll unterstrichen werden.

```
10 POS X,Y:DISP "SHARP CORPORATION"
20 CURSOR X,Y+1:DISP "                    "
30 END
```

ENTER

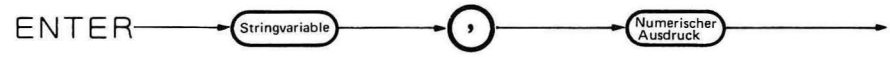
Ordnet am Bildschirm ausgegebene Zeichen einer spezifizierten Stringvariablen zu.

〈VERWENDUNG〉
Für die Verarbeitung von am Bildschirm ausgegebenen Zeichen innerhalbe des Programms oder ihre Ausgabe über den Drucker.

FORMAT:ENTER A$, X

A$: Stringvariable
X : Zahl der zuzuordnenden Zeichen

(SYNTAX)



ERLÄUTERUNG:

1. Die ENTER-Anweisung ordnet der Stringvariablen A$, beginnend von der augenblicklichen Kursorposition, insgesamt X Zeichen zu.
2. Die Größe der Stringvariablen A$ bestimmt sich entsprechend der Relation: 1 ASCII-Zeichen = 1 Byte.

EINFACHES BEISPIEL:

```
10 CURSOR 30,10
20 DISP "SONNTAGABEND"
30 CURSOR 30,10
40 ENTER T$,7
50 END
```

Schreibt ab der Kursorposition (30, 10) das Wort "SONNT GABEND" auf den Bildschirm.

Weist die ersten 7 Zeichen ab der Kursorposition (den String "SONNTAG") der Variablen T$ zu.

ANWENDUNGSBEISPIEL:

Ausgabe aller am Bilschirm angezeigten Daten über den Drucker:

```
10 DIM A$(1)*2000 ·····Definition der Arrayvariablen, welcher der Bildschirminhalt zugewiesen werden soll.
   :
   :
1000 CURSOR 0,0 ·····Kursor zu Position HOME (0, 0).
1010 ENTER A$(1),2000 ····Ordnet alle am BS ausgegebenen Daten der Variablen A$ zu.
1020 PRINT A$(1) ·····Gibt den Inhalt der Variablen A$ über den Drucker aus.
1030 END
```

NINWEIS

1. Die Zuordnung durch die [ENTER] Anweisung umfaßt auch eine mittels der SET-Anweisung erzeugte Pseudographik.
2. Die Zuordnung durch [ENTER] umfaßt nicht die mit der Anweisung LINE erstellten Markierungslinien.
3. Ebenso wenig kann man mittels [ENTER] graphische Daten, die über den Bilschirm ausgegeben werden, lesen. Für die Zuordnung graphischer Ausgabedaten zu einer kollektiven Arrayvariablen gibt es die Anweisung GENTER (Seite 221).

CURSOR

Positioniert den Kursor auf dem Bildschirm und schaltet die Kursoranzeige ein/aus.

〈VERWENDUNG〉
Dient der Ausgabe von Daten an einer vorbestimmten Stelle des Bildschirms.

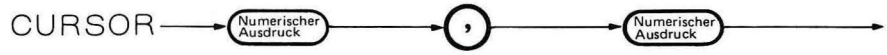
● **FUNKTION 1:** Positionierung des Kursors.

FORMAT: CURSOR X, Y

X: Wert der X-(Text) Koordinate

Y: Wert der Y-(Text) Koordinate

(SYNTAX)



ERLÄUTERUNG:

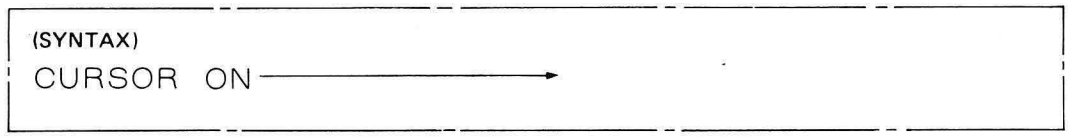
Die CURSOR-Anweisung bewegt den Kursor an die Stelle (X, Y) des Text-Koordinatensystems. Nähere Ausführungen über den Wertebereich für X und Y sowie über den Aufbau des Text-Koordinatensystems finden sich auf Seite 215 (DISP CLEAR).

ANWENDUNGSBEISPIEL:

Man gebe den Text "SHARP COMPUTER MZ-3500 BASIC PROGRAMMIERSPRACHENHANDBUCH" in der Mitte des Bildschirms aus.

```
10 DISP CLEAR ..... Löschen des Bildschirms.
20 X=26 ..... Zuteilen von 26 an die numerische Variable X.
30 CURSOR X,9 DISP "SHARP COMPUTER" ... Ausgeben von "SHARP COMPUTER", beginnend in der Position der Textkoordinaten (26,9).
40 CURSOR X,11 DISP "MZ-3500" ..... Ausgeben von "MZ-3500", beginnend in der Position der Textkoordinaten (26,11).
50 CURSOR X,13 DISP "BASIC" ..... Ausgeben von "BASIC", beginnend in der Position der Textkoordinaten (26,13).
60 CURSOR X,15 DISP "PROGRAMMIERSPRACHENHANDBUCH"
                ..... Ausgeben von "PROGRAMMIERSPRACHENHANDBUCH", beginnend in der Position der Textkoordinaten (26,15).
70 END ..... Beenden der Programmausführung.
```

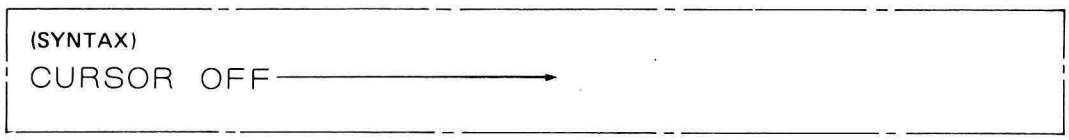
● **FUNKTION 2:** Schaltet die Kursoranzeige während der Programmausführung ein.
FORMAT: CURSOR ON



ERLÄUTERUNG:

Die CURSOR ON-Anweisung schaltet die Anzeige des Kursors während einer Programmausführung ein.

● **FUNKTION 3:** Schaltet die Kursoranzeige während der Programmausführung aus.
FORMAT:CURSOR OFF



ERLÄUTERUNG:

Die CURSOR OFF-Anweisung schaltet die Anzeige des Kursors während einer Programmausführung aus.

Nach der Initialisierung des Systems ist CURSOR OFF geschaltet.

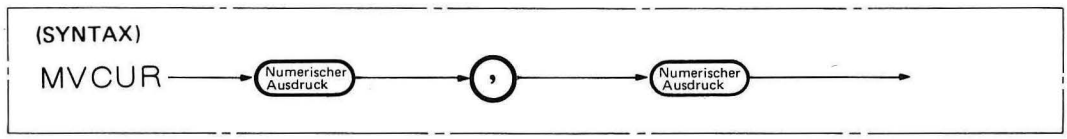
MVCUR

Zur Kursorbewegung.

FORMAT: MVCUR X, Y

X: Zahl der Spalten für die Kursorbewegung nach rechts

Y: Zahl der Zeilen für die Kursorbewegung nach unten



ERLÄUTERUNG:

Bewegt den Kursor, ausgehend von der augenblicklichen Kursorposition um X Spalten (1 Spalte = 1 Zeichen) nach rechts und um Y Zeilen (1 Zeile = 1 Zeichen) nach unten. Hat X bzw. Y einen negativen Wert, vollzieht sich die Bewegung um I X I Spalten nach links bzw. um I Y I Zeilen nach oben.

EINFACHES BEISPIEL:

```
10 MVCUR 15,-7
```
..... Bewegt den Kursor, ausgehend von der derzeitigen Kursorposition, um 15 Spalten nach rechts und 7 Zeilen nach oben. (Ist die augenblickliche Kursorposition (0, 20), bewegt sich der Kursor nach (15, 13)).

ANWENDUNGSBEISPIEL:

Ausgabe eines Textblocks an einer bestimmten Stellen des Bildschirms.

```
  10 DISP CLEAR ..................................... Löscht den Bildschirm.
  20 X=30:Y=10:GO SUB 1000
 999 END
1000 CURSOR X,Y:DISP "**************";..... Gibt untereinander 4
1010 MVCUR -14,1:DISP "SHARP COMPUTER";  Strings aus.
1020 MVCUR -14,1:DISP "MODELL MZ-3500";
1030 MVCUR -14,1:DISP "**************";
1040 RETURN
```

LINE

Zeichnet eine waagrechte oder senkrechte Markierungslinie.

〈VERWENDUNG〉
Zur Verbesserung der Möglichkeiten der Bildschirmausgaben.

FORMAT: L I NE – / , X, Y, Z

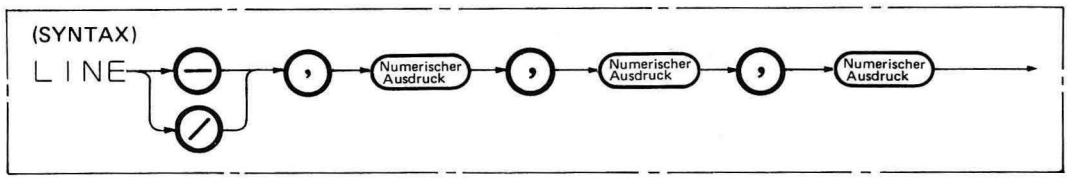
–: Zeichnet eine horizontale Linie (von links nach rechts)

/ : Zeichnet eine vertikale Line (von oben nach unten)

X: Position der X-(Text) Koordinate für den Beginn der Linie

Y: Position der Y-Koordinate für den Beginn der Linie

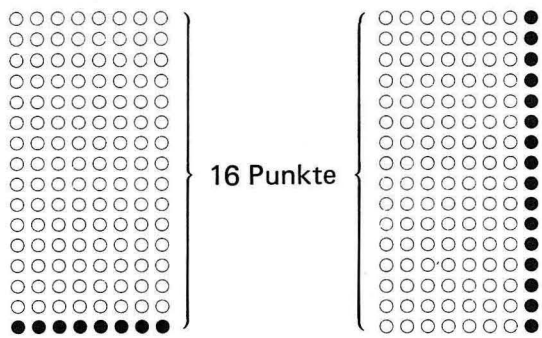
Z: Anzahl der Zeichen in einem Intervall, in dem die Markierungslinie gezogen wird.



ERLÄUTERUNG:

Die LINE-Anweisung zeichnet eine Markierungslinie auf dem Bildschirm. Die Linie beginnt in der Position (X, Y) der Textkoordinaten. Die Länge der Linie beträgt Z Zeichen. Die Richtung wird durch das erste Argument der LINE-Anweisung bestimmt: – bedeutet von links nach rechts, / von oben nach unten.

Für horizontale Markierungsinien wird die unterste Reihe, für vertikale Markierungslinien die äußerste rechte Spalte der Textzeichen-Punktmatrix benutzt (siehe Abbildung unten).



EINFACHES BEISPIEL:

```
10 LINE -,0,10,40
```
..... Zeichnet eine Markierunglinie, die im Text-Koordinatenpunkt (0, 10) beginnt und sich über 40 Zeichen von links nach rechts erstreckt.

WICHTIGER HINWEIS

1. Zum Löschen einer Markierungslinie benutze man die RE-Anweisung (Seite 229). Nähere Angaben finden sich bei der Darstellung der RE-Anweisung unter Funktion 1. Die Markierungslinie kann auch zusammen mit der Textanzeige durch Drücken der Tasten CTRL und HOME gelöscht werden.
2. Wenn der numerische Ausdruck für Z beim Zeichnen einer vertikalen Markierungslinie zu einem Überschreiten der untersten Bildschirmzeile (Textzeile) führt, wird eine Fehlermeldung (ERROR 1) ausgegeben.
3. Wenn der numerische Ausdruck für Z beim Zeichenen einer horizontalen Markierungslinie zum Überschreiten des rechten Bildschirmrands führt, wird die Markierungslinie in der Folgezeile fortgesetzt. Überschreitet er jedoch den rechten Rand der untersten Zeile, dann wird ebenfalls eine Fehlermeldung ausgegeben (ERROR 1).
4. Die Markierungslinie der LINE-Anweisung unterscheidet sich von einer Linie, die durch Verwendung der pseudographischen Symbole gezeichnet wird.

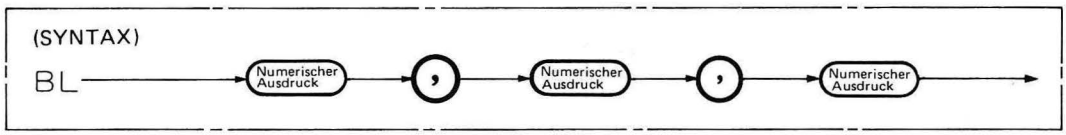
BL

Blinkende Ausgabe von Zeichen am Bildschirm.

〈VERWENDUNG〉
Zum Hervorheben von Zeichen durch Blinken.

FORMAT:BL X, Y, Z

X: X-Koordinate des ersten Zeichens für die blinkende Ausgabe
Y: Y-Koordinate des ersten Zeichnes für die blinkende Ausgabe
Z: Anzahl der Zeichen, die blinkend ausgegeben werden sollen



ERLÄUTERUNG:

Die BL-Anweisung veranlaßt, beginnend in der (Text) Koordinatenposition (X, Y), die blinkende Darstellung von insgesamt Z Zeichen.
Invertiert oder farbig ausgegebenen Zeichen können ebenfalls blinkend dargestellt werden.

EINFACHES BEISPIEL:

```
10 BL 0,2,80
```
.....Die blinkende Ausgabe ersteckt sich über 80 Zeichen beginnend ab der Kursorposition (0, 2). (1 Zwischenraum zählt ebenfalls als 1 Zeichen).

WICHTIGER HINWEIS

1. Das Blinken kann mittels der RE-Anweisung (Seite 230) aufgehoben werden. Nähere Angaben finden sich bei der Darstellung der RE-Anweisung unter Funktion 2. Die Blinkfunktion wird ebenso durch Drücken der Tasten CTRL und HOME (Löschen der Textausgabe des Bildschirms) aufgehoben.
2. Eine blinkende Ausgabe graphischer Muster durch die BL-Anweisung ist nicht möglich. Pseudographische Symbole dagegen zählen zur Textausgabe. Sie können blinkend dargestellt werden.

RV

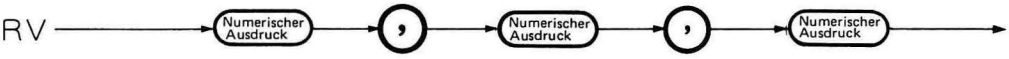
Invertierte Zeichenausgabe am Bildschirm.

〈VERWENDUNG〉
Zum Hervorheben von Zeichen durch invertierte Darstellung.

FORMAT: RV X, Y, Z

X: X-Koordinate des ersten Zeichens für die invertierte Darstellung
Y: Y-Koordinate des ersten Zeichnes für die invertierte Darstellung
Z: Anzahl der Zeichen, die invertiert dargestellt werden sollen

(SYNTAX)



ERLÄUTERUNG:

Die RV-Anweisung veranlaßt, beginnend in der Bildschirmkoordinatenposition (X, Y), die invertierte Darstellung der folgenden Z Zeichen. Blinkende Zeichen können ebenfalls invertiert ausgegeben werden, nicht dagegen farbige.

EINFACHES BEISPIEL:

```
10 RV 5,5,20
```
.....Beginnt bei der Bildschirmposition (5, 5) mit der invertierten Darstellung von 20 Zeichen.

WICHTIGER HINWEIS

1. Zum Aufhebung der invertierten Darstellungsweise benutzt man die RE-Anweisung (Funktion 3). Nähere Angaben hierzu finden sich auf Seite 229. Die Invertierung kann auch über die Tasten CTRL und HOME (Löschen des Bildschirms) aufgehoben werden.
2. Eine invertierte Ausgabe graphischer Muster durch die RV-Anweisung ist nicht möglich. Pseudographische Symbole dagegen zählen zur Textausgabe. Sie können invertiert dargestellt werden.

RE

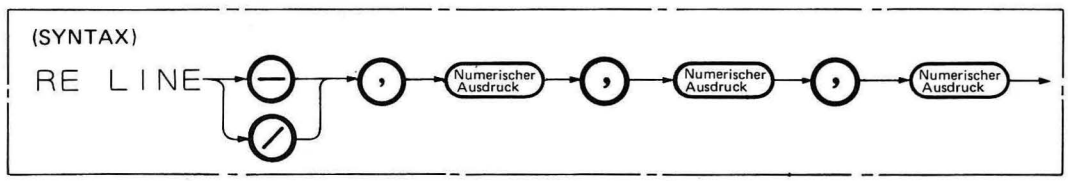
Löscht Markierungslinien, hebt die blinkende oder invertierte Darstellung auf.

〈VERWENDUNG〉
Zum Löschen von Markierungslinien, Aufheben von blinkender oder invertierter Darstellung der Zeichen am Bildschirm.

● **FUNKTION 1:** Löscht Markierungslinien.

FORMAT: RE LINE – /, X, Y, Z

– : Löscht in horizontaler Richtung
/ : Löscht in vertikaler Richtung
X : X-(Text) Koordinate des ersten zu löschenden Zeichens
Y : Y-Koordinate des ersten zu löschenden Zeichens
Z : Umfang des zu löschenden Intervalls



ERLÄUTERUNG:

Die RE LINE-Anweisung löscht die Markierungslinie über ein Intervall von Z Zeichen. Sie beginnt bei den Bildschirm-Koordinaten (X, Y). Die Löschrichtung hängt von dem ersten Argument der Anweisung ab: – löscht in horizontaler, / in vertikaler Richtung.
Führt der numerische Ausdruck für Z beim Löschen einer vertikalen Markierungslinie zum Überschreiten der untersten Bildschirmzeile (Textzeile), dann wird eine Fehlermeldung (ERROR 1) ausgegeben.

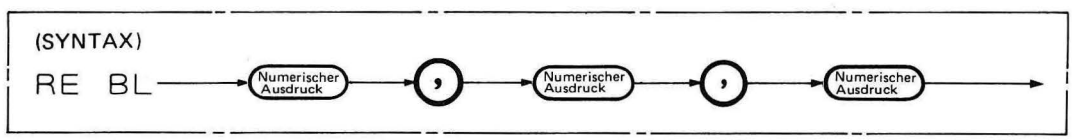
EINFACHES BEISPIEL:

```
100 RE LINE /,0,0,25
```
.....Löscht die Markierungslinie von oben nach unten in einem Intervall, das 25 Zeichen entspricht, und beginnt im (Text) Koordinatenpunkt (0,0).

● **FUNKTION 2:** Hebt das Blinken auf.

FORMAT: RE BL X, Y, Z

X: X-(Text) Koordinate für den Startpunkt
Y: Y-Koordinate für den Startpunkt
Z: Anzahl von Zeichen, über die der Blinkmodus aufgehoben wird



ERLÄUTERUNG:

Die RE BL-Anweisung hebt die blinkende Darstellung in den auf den Text-Koordinatenpunkt (X, Y) folgenden Z Zeichen auf.

EINFACHES BEISPIELE:

```
100 RE BL 0,2,80
```
..... Hebt die blinkende Darstellung ab dem Text-Koordinatenpunkt (0, 2) über ein Intervall von 80 Zeichen auf.

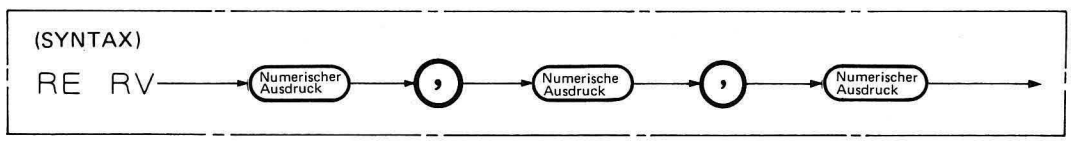
● **FUNKTION 3:** Hebt die invertierte Darstellungsweise auf.

FORMAT: RE RV X, Y, Z

X: X-(Text) Koordinate für den Startpunkt

Y: Y-Koordinate für den Startpunkt

Z: Zeichenzahl, über die die invertierte Darstellungsweise aufgehoben wird



ERLÄUTERUNG:

Die RE RV-Anweisung hebt die invertierte Darstellungsweise in den auf den Text-Koordinatenpunkt (X, Y) folgenden Z Zeichen auf.

EINFACHES BEISPIEL:

```
100 RE RV 5,5,20
```
..... Hebt die invertierte Darstellung ab dem Text-Koordinatenpunkt (5, 5) über ein Interval von 20 Zeichen auf.

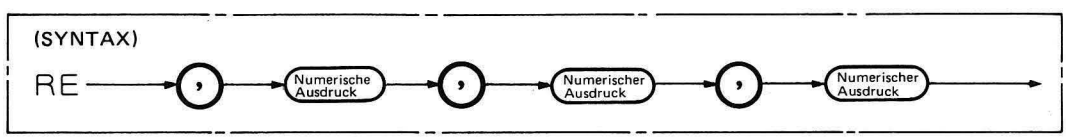
● **FUNKTION 4:** Führt die Funktionen 1 – 30 gleichzeitig aus.

FORMAT: RE, X, Y, Z

X: X-(Text) Koordinate für den Startpunkt

Y: Y-Koordinate für den Startpunkt

Z: Zeichenzahl, über die die Markierungslinie gelöscht, die blinkende und/oder die invertierte Darstellungsweise aufgehoben wird.



ERLÄUTERUNG:

Die RE-Anweisung löscht die Markierungslinie und hebt sowohl die blinkende als auch invertierte Darstellungsweise in den auf den Text-Koordinatenpunkt (X, Y) folgenden Z Zeichen auf.

Näheres hierzu siehe bei den Funktionen 1 – 3.

TABLE

Erzeugt ein Liniengitter für eine Tabelle.

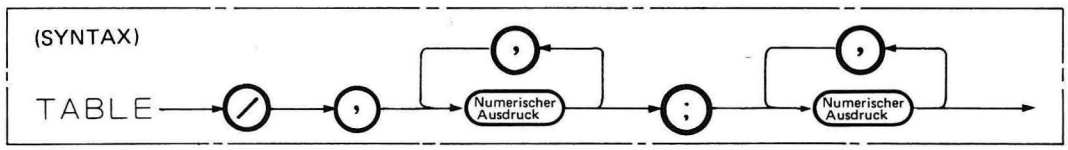
〈VERWENDUNG〉
Zum Erstellen eines Rahmens für eine Tabelle.

● **FUNKTION 1:** Erzeugen des Tabellengitters durch Verwendung der Pseudo-Graphiksymbole

FORMAT: **TABLE / , {X} ; {Y}**

X: Z-Koordinaten der Gitterschnittpunkte (X ≧ 1)

Y: Y-Koordinaten der Gitterschnittpunkte (Y ≧ 1)



ERLÄUTERUNG:

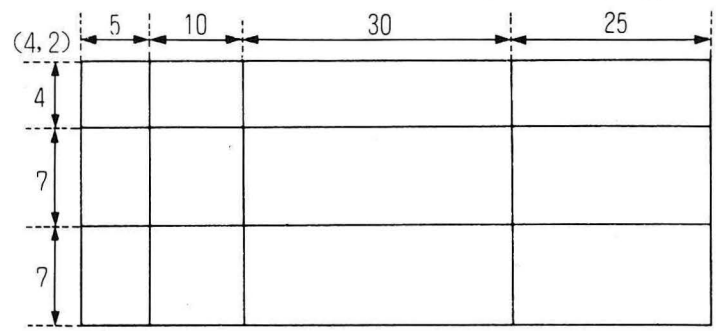
Die TABLE /-Anweisung erzeugt ein Tabellengitter durch die Verwendung der Pseudo-Graphiksymbole ├ ┴ ┤ ┬ ┌ └ ┘ │ ─ ┼ . Der linke. obere Punkt des Gitters liegt in dem Text-Koordinatenpunkt ((X-1, Y-1). Er wird durch den jeweils ersten numerischen Ausdruck für X und Y bestimmt.

Der jeweils zweite und die folgenden numerischen Ausdrücke für X und Y geben den Zwischenraum zwischen den einzelnen Tabellenlinien an, die X-Werte die Zwischenräume in waagrechter, die Y-Werte in senkrechter Richtung. Der Zwischenraum wird in Spalten bzw. Zeilen ausgedrückt. Ein Zwischenraum von X macht die Unterbringung von X-1 Zeichen in einem Tabellenfeld (in waagrechter Richtung) möglich, ein Zwischenraum von Y entsprechend die Unterbringung von Y-1 Zeichen (in senkrechter Richtung).

EINFACHES BEISPIEL:

```
10 TABLE /,5,5,10,30,25;3,4,7,7
```

.....Erzeugt ein Tabellengitter entsprechend der Abbildung.



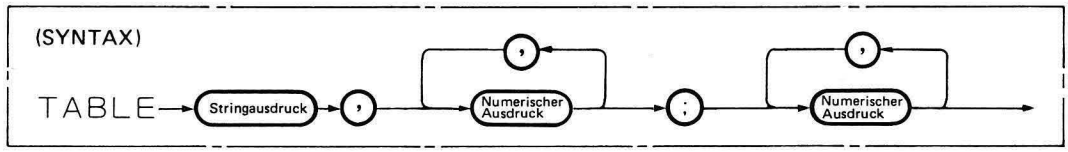
● **FUNKTION 2:** Erzeugen eines Tabellengitters mit einem beliebigen Zeichen.

FORMAT: **TABLE C, {X} ; {Y}**

C: String für die Zusammensetzung der Gitterstäbe

X: X-Koordinaten der Gitterschnittpunkte (X ≧ 1)

Y: Y-Koordinaten der Gitterschnittpunkte (Y ≧ 1)



ERLÄUTERUNG:

Die TABLE C-Anweisung erzeugt ein Tabellengitter, dessen Linien sich durch wiederholtes Aneinanderfügen des ersten Zeichens des Strings C ergibt. Im übrigen gelten die zu Funktion 1 gemachten Ausführungen.

EINFACHES BEISPIELE:

```
10 TABLE "*",5,5,10,30,25;3,4,7,7
```

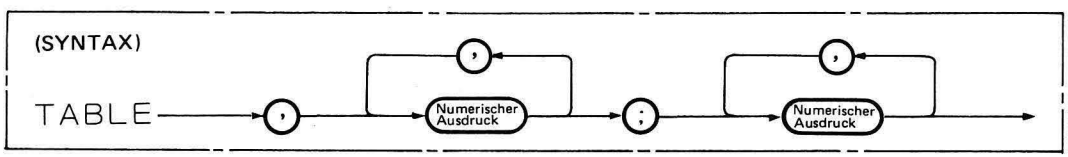
.....Erzeugt ein Tabellengitter entsprechend der Abbildung

```
**********************************************************************
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
**********************************************************************
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
**********************************************************************
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
*    *         *                   *                   *
**********************************************************************
```

● **FUNKTION 3:** Erzeugen eines Tabellengitters mit den Markierungslinien.
FORMAT: TABLE, (X) ; (Y)

X: X-Koordinaten der Gitterschnittpunkte (X ≧ 1)

Y: Y-Koordinaten der Gitterschnittpunkte (Y ≧ 1)



ERLÄUTERUNG:

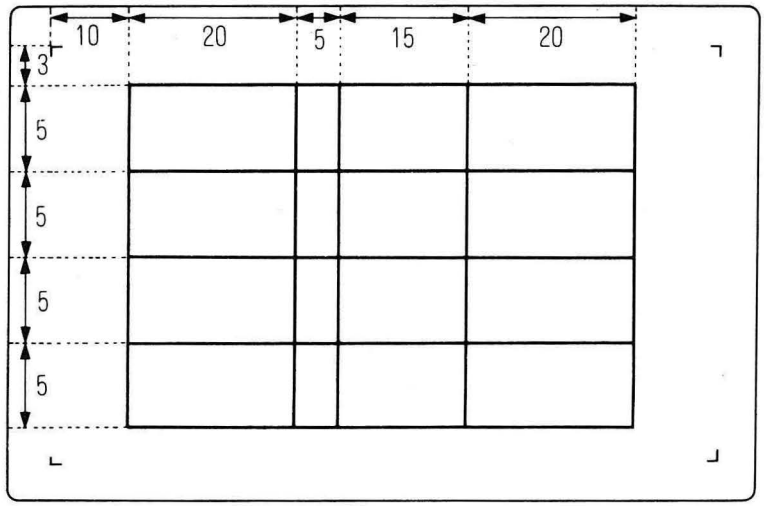
Die TABLE,-Anweisung erzeugt ein Tabellengitter aus Markierungslinien.

Die zu Funktion 1 gemachten Ausführungen gelten analog, jedoch mit dem Unterschied, daß die für die Bemessung der Zwischenräume angegebenen Werte X und Y auch der Anzahl der Zeichen entspricht, die in den Tabellenfeldern in waagrechter (X) bzw. senkrechter (Y) Richtung untergebracht werden können.

EINFACHES BEISPIEL:

```
10 TABLE ,10,20,5,15,20:3,5,5,5,5
```

..... Erzeugt ein Tabellengitter entsprechend der Abbildung.



SCROL

Verschiebt ein eindeutig definiertes, rechteckiges Feld auf dem Bildschirm in die gewünschte Richtung.

〈VERWENDUNG〉
Zur Verbesserung der Ausgabemöglichkeiten am Bildschirm.

FORMAT: SCROL (@) Z, U, V, X, Y

@ : Zur Spezifikation der Behandlung von Markierungslinien, Blinkausgabe, invertierter Ausgabe und Farbausgabe

Z : Verschiebungsrichtung (0, 1, 2 oder 3)

Wert für Z	Verschiebungsrichtung
0	↑
1	↓
2	←
3	→

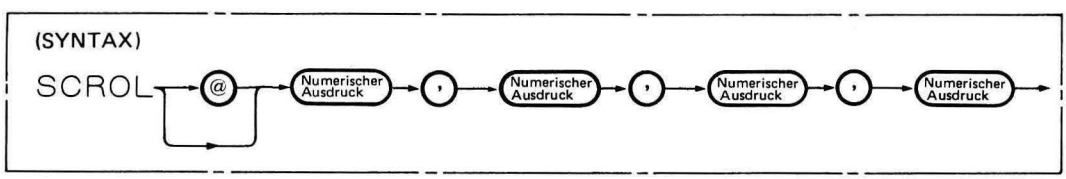
Die folgenden numerischen Ausdrücke definieren das zu verschiebende Feld auf dem Bildschirm:

U : X-Koordinate der linken oberen Ecke

V : Y-Koordinate der linken oberen Ecke

X : X-Koordinate der rechten unteren Ecke

Y : Y-Koordinate der rechten unteren Ecke



ERLÄUTERUNG:

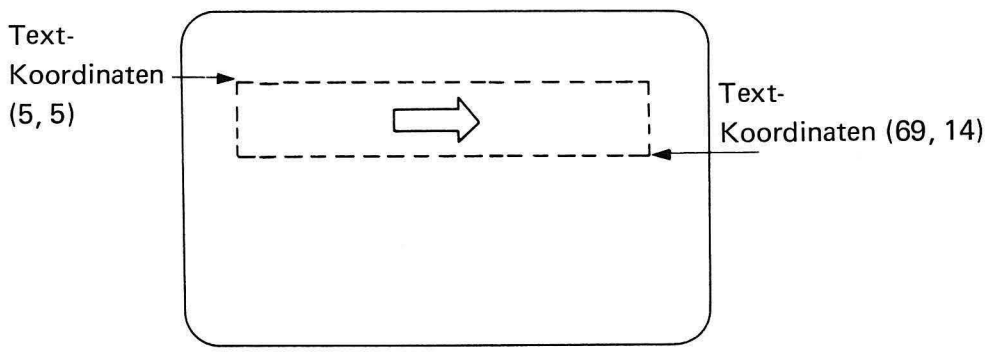
1. Die SCROL-Anweisung verschiebt ein rechteckiges Feld, das durch die Text-Koordinaten seiner linken oberen (U, V) und rechten unterren (X, Y) Ecke eindeutig definiert ist, in die durch Z spezifizierte Richtung um ein Zeichen.
2. Enthält die SCROL-Anweisung das Argument @, dann werden die Markierungslinien und der gesamte restliche Ausgabestatus (blinkende) Anzeige, invertierte oder farbige Ausgabe der Zeichen) mit verschoben. Fehlt @, dann werden nur die Zeichen, nicht aber ein besonderer Ausgabestatus verschoben.

EINFACHES BEISPIELE:

1. `100 SCROL @3,5,5,60,14` .....Mit dieser Anweisung wird der angezeigte Bildschirminhalt innerhalb des Feldes, dessen linke obere Ecke im Text-Koordinatenpunkt (5, 5) und dessen rechte untere Ecke im Punkt (60, 14) liegt einschließlich der Makierungslinien, der blinkenden, invertierten oder farbigen Darstellung um ein Zeichen nach rechts verschoben.

2. `100 SCROL 0,5,5,69,14` .....Mit dieser Anweisung wird der angezeigte Bildschirminhalt innerhalb des Feldes, dessen linke obere Ecke im Text-Koordinatenpunkt (5, 5) und dessen rechte untere Ecke im Punkt (69, 14) liegt, ohne Markierungslinien, blinkende, invertierte oder farbige Darstellung um ein Zeichen nach oben verschoben.

Darstellung des zu verschiebenden Bildschirmfeldes einschließlich der Verschiebungsrichtung.



CHANGE DISP

Wahl des Bildschirmausgabemodus (Zeichenmodus und Farbmodus).

〈VERWENDUNG〉
Zum Einschalten des gewünschten Zeichenmodus (Spalten- und Zeilenzahl) und Farbmodus (monochrom oder farbig).

FORMAT: CHANGE DISP S

S: Stringkonstante zur Charakterisierung des gewünschten Bildschirmmodus

S	Bildschirmmodus	S	Bildschirmmodus
"80"	80 x 25 monochrom	"80C"	80 x 25 farbig
"80/"	80 x 20 monochrom	"80/C"	80 x 20 farbig
"40"	40 x 25 monochrom	"40C"	40 x 25 farbig
"40/"	40 x 20 monochrom	"40/C"	40 x 25 farbig

(SYNTAX)



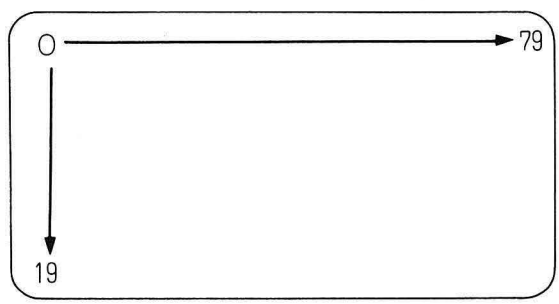
ERLÄUTERUNG:

1. Mit der CHANGE DISP-Anweisung kann man entsprechend der Wahl der Stringkonstante S den Zeichenausgabemodus (Zahl der Spalten und Zeilen auf dem Bildschirm) und den Farbmodus (monochrom oder farbig) ändern.
2. Nach dem Einschalten des Geräts ist die Bildschirmausgabe auf 80 x 25 monochrom eingestellt.

EINFACHES BEISPIEL:

```
10 CHANGE DISP "80/C"
```

..... Stellt den Bildschirm auf einen Zeichenumfang von 80 x20 (80 Zeichen pro Zeile und 20 Zeichen pro Spalte) und auf farbige Anzeige ein. (s. Abbildung)



HINWEIS

Zur Festlegung des graphischen Ausgabemodus siehe SDISP- und ODISP-Anweisungen (Seite 241 und 243).

CHANGE PRINT

Einstellung des Papiervorschubs beim Drucker.

〈VERWENDUNG〉
① Zum Ausdrucken von Tabellen aus Pseudo-Graphiksymbolen.
② Rückstellen des Papiervorschubs von der Graphik- zur normalen Textausgabe.

FORMAT: CHANGE PRINT S

S : Bestimmung des Zeilenvorschubwegs
"N": Normalmodus
"G": Graphikmodus

(SYNTAX)



ERLÄUTERUNG:

1. Die CHANGE PRINT-Anweisung stellt den Papiervorschub bei dem Drucker entsprechend der Zeichenkonstante S ein. Im Normalmodus beträgt der Zeilenvorschub 1/6". Im Graphikmodus differiert die Einstellung von Durcker zu Drucker.
2. Werden Pseudo-Graphiksymbole benutzt, sollte in den Graphikmodus umgeschaltet werden. Dadurch werden die Symbole nahtlos aneinandergefügt.
3. Nach dem Einschalten des Geräts ist der Normalmodus eingestellt.

EINFACHES BEISPIEL:

```
10 CHANGE PRINT "G"
```
..... Der Zeilenvorschub wir für den Graphikmodus eingestellt.

COLOR

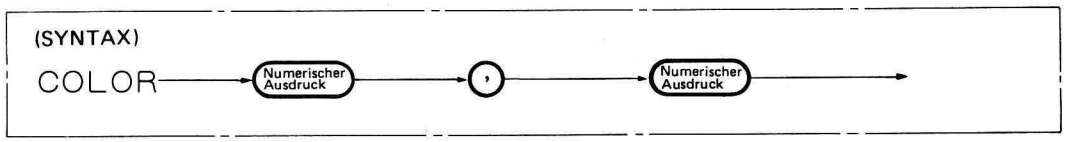
Bestimmt die Farbe für die Ausgabe der Zeichen (einschließlich der Pseudo-Graphiksymbole).

FORMAT: COLOR X, Y

X: Farbe

X	0	1	2	3	4	5	6	7
FARBE	weiß	gelb	türkis	grün	pink	rot	blau	schwarz

Y: Anzahl der Zeichen



ERLÄUTERUNG:

1. Die COLOR-Anweisung bewirkt ab der aktuellen Kursorposition die Darstellung von Y Zeichen in der durch X bestimmten Farbe.
2. Wird bei eingeschaltetem Farbmodus keine Farbe durch die Anweisung COLOR bestimmt, erfolgt die Ausgabe automatisch in weiß. (Die Hintergrundfarbe ist schwarz (7).)

EINFACHES BEISPIEL:

```
10 COLOR 5,10
```
..... Veranlaßt die Ausgabe der nächsten 10 Zeichen ab der aktuellen Kursorposition in Rot.

HINWEIS

1. Diese Anweisung kann nur wirksam werden, wenn zuvor durch die CHANGE DISP-Anweisung (Seite 237) oder die SDISP-Anweisung (Seite 241) der farbige Ausgabemodus eingeschaltet worden ist.
2. Die Gültigkeit dieser Anweisung setzt natürlich den Anschluß eines Farbmonitors voraus.
3. Farbige Zeichen können blinkend, nicht jedoch in invertierter Darstellung ausgegeben werden.

ANWENDUNGSBEISPIEL:

Man gebe das Wort FARBE im Zentrum des Bildschirms aus.

```
10 CHANGE DISP "80C" ..... Stellt auf den 80 x 25-Zeichen-Farb-Modus ein.
20 CURSOR 38,12:COLOR 5,5 ..... Stellt Kursorposition und Farbe ein.
30 DISP "FARBE" ..... Gibt das Wort FARBE aus.
40 END
```

6. ANWEISUNGEN ZUR ERZEUGUNG VON GRAPHIK

Folgende Anweisungen stehen für die Nutzung der umfangreichen graphischen Möglichkeiten dieses Systems zu Verfügung:

SDISP
ODISP
GCOLOR
CLS
SCALE
PSET
PRESET
LINE
CIRCLE
PAINT
GENTER
GDISP
GPRINT
GCURSOR
GINPUT
GCHR

Jede dieser Anweisungen kann auch in der manuelle Betriebsart eingegeben werden.
Die Benutzung der Graphik-Anweisungen setzt das Vorhandensein einer Graphikkarte und eines Graphikspeichers voraus (Optionen). (Näheres hierzu findet sich im Bedienerhandbuch).

SDISP

Ein- und Ausschalten des Ausgabemodus für den Bildschirm.

〈VERWENDUNG〉
① Für die Ausgabe von Graphik über den Bildschirm.
② Für die kombinierte Ausgabe von Text und Graphik über den Bildschirm.

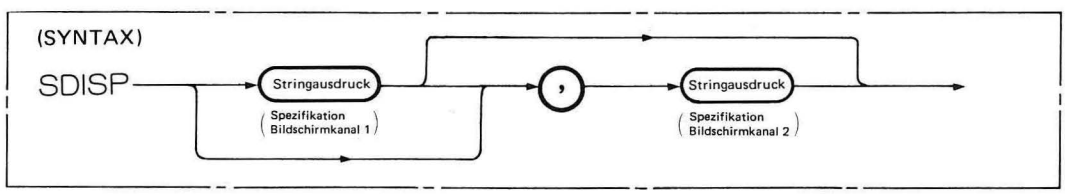
FORMAT: SDISP C〔, D〕 | 〔C〕, D

C : Spezifizierung der Ausgabeebene für Bildschirmkanal 1 (CRT-1)
D : Spezifizierung der Ausgabeenene für Bildschirmkanal 2 (CRT-2)
Die Spezifizierung erfolgt in der Reihenfolge:
"Ausgabeebene Farbe Besonderheiten"
① ② ③

① Ausgabeebene:
0 ..... Textebene
1 ..... Graphikebene 1 (blau)
2 ..... Graphikebene 2 (rot)
3 ..... Graphikebene 3 (grün)
xy ..... Überlagerung von 2 Ebenen
xyz ..... Überlagerung von 3 Ebenen
xyzw ..... Überlagerung von 4 Ebenen
(x, y, z, w: 0, 1, 2, 3)

② Farbe:
C ..... Farbmodus eingeschaltet

③ Besonderheiten:
A ..... Darstellungsweise (blinkende oder invertierte Darstellung, Ausgabe von Markierungslinien) kann modifiziert werden. A kann nur in String D stehen, da die der Anzeige für Kanal 1 (CRT-1) jederzeit modifiziert werden kann.



ERLÄUTERUNG:

1. Die SDISP-Anweisung bestimmt, welche Speicherebenen über die Bildschirmkanäle 1 und 2 ausgegeben werden. String C legt die Ausgabe für Bildschirmkanal 1 und String D die Ausgabe für Kanal 2 fest.
   Diese Spezifikation beinhaltet die Nummern der auszugebenden Ebenen, den Farbmodus und den Vorbehalt zur Ausgabemodifikation.
2. Im ersten Teil der Spezifikation werden die auszugebenden Bereiche festgelegt: die Textebene und drei Graphikebenen (1, 2 und 3). Sie können einzeln oder beliebig miteinander kombiniert (überlagert) ausgegeben werden. Sollen z.B. die Textebene, die Graphikebene 1 und die Graphikebene 3 zusammen über den Bildschirmkanal 1 ausgegeben werden, dann müßte für C der String "013" stehen.
3. Steht in einem der beiden Strings das Argument C, dann erfolgt die Ausgabe über den angesprochenen Bildschirmkanal im Farbmodus.
   Wurde der Farbmodus bereits durch die CHANGE DISP-Anweisung (siehe Seite 237) eingestellt, erfolgt die farbige Ausgabe auch ohne eine erneute Spezifikation in der SDISP-Anweisung.
4. Die Angabe von Argument A zum Vorbehalt einer Veränderung des Ausgabemodus für Textzeichen und Cursoranzeige kann nur im String D erfolgen. (Bei Kanal 1 (CRT-1) wird der Cursor angezeigt und Zeichenmodifikation ist stets möglich.)
5. Fehlt in der SDISP-Anweisung eine der Spezifikationen C oder D, dann bleibt der Ausgabemodus für den betreffenden Kanal unverändert.
6. Bei der Initialisierung werden Kanal 1, Kanal 2 und die Textebenen, Kanal 1 und 2 (CRT-1, CRT-2) so eingestellt, daß sie in monochromer Betriebsart verändert werden können, und der Cursor angezeigt wird (nicht in Farbe).

EINFACHE BEISPIELE:

1. `10 SDISP"1","0A"` ..... Graphikebene 1 (blau) wird über Kanal 1 ausgegeben. Die Textebene wird über Kanal 2 so ausgegeben, daß Zeichen geändert werden können, und der Cursor angezeigt wird.
2. `10 SDISP"01C","123"` ..... Dr Ausgabemodus wird wie folgt eingestellt: über Kanal 1 wird die Textebene und Graphikebene 1 farbig ausgegeben, über Kanal 2 alle drei Graphikebenen in Farbe.

WICHTIGER HINWEIS

1. Zur Ausgabe einer 8-farbigen Graphik müssen alle drei Graphikebenen gleichzeitig eingeschaltet sein. Für eine farbige Ausgabe der Textebene ist keine Überlagerung notwendig.
2. Es ist nicht möglich, Bildschirmkanal 1 farbig und zugleich Bildschirmkanal 2 monochrom einzustellen (oder umgekehrt).
3. Werden gleichzeitig ein farbiger sowie ein grüner Bildschirm im Farbmodus betrieben, dann muß der grüne Bildschirm am Kanal 2 (CTR-2) angeschlossen sein. In diesem Fall kann Option A in der SDISP-Anweisung nicht durchgeführt werden.

ODISP

Schaltet den Eingabemodus der Graphikebenen ein.

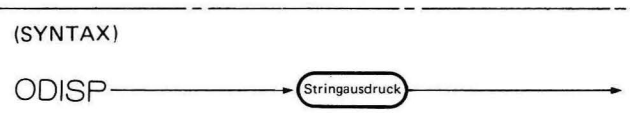
〈VERWENDUNG〉
Zum Zeichnen von Graphik.

FORMAT: ODISP C

C : Auswahl der Graphikebene(n)

1 ..... Graphikebene 1
2 ..... Graphikebene 2
3 ..... Graphikebene 3
xy ..... Zwei Graphikebenen
xyz ..... Drei Graphikebenen
(x, y, z : 1, 2, 3)

(SYNTAX)



ERLÄUTERUNG:

1. Eine Graphik wird in denjenigen Bereich des Bildwiederholspeichers gezeichnet (geplottet), welcher der durch die ODISP-Anweisung spezifizierten Graphikebene entspricht. (Der Bildwiederholspeicher ist ein Teil des Speichers, in dem die Daten der Graphikebenen und der Textebene getrennt abgelegt sind.)
   Die ODISP-Anweisung steuert die Eingabe von Daten in den Bildwiederholspeicher: Durch String C wird bestimmt, für welche der drei Graphikebenen der Eingabemodus eingeschaltet werden soll.
   Eingaben für die Textebene sind jederzeit möglich.
2. Die mit den Graphikanweisungen PSET, LINE, CIRCLE, PAINT, GDISP oder GCHR erzeugten graphischen Muster werden in die durch die ODISP-Anweisung bestimmte Graphikebene(n) abgespeichert.
3. Ein in einer Graphikebene abgespeichertes, graphisches Muster wird über den Bildschirm ausgegeben, indem man für diese Graphikebene mittels der SDISP-Anweisung (Seite 241) den graphischen Ausgabemodus einschaltet.
4. Bei der Initialisierung des Systems wird für die Graphikebene 1 der Eingabemodus (nicht aber zugleich auch der Ausgabemodus!) eingeschaltet.

EINFACHES BEISPIELE:

```
10 ODISP "23"
```
..... Schaltet den Eingabemodus für die Graphikebenen 2 (Rot) und 3 (Grün) ein.

WICHTIGER HINWEIS

Für das Plotten von 8-farbiger Graphik muß der Eingabemodus für alle drei Graphikebenen gleichzeitig eingeschaltet sein. Die Ausgabe von farbiger Graphik über den Bildschirm setzt eine entsprechende Spezifizierung mittels der SDISP-Anweisung voraus.

ANWENDUNGSBEISPIEL:

Ein Anwendungsbeispiel zur Demonstration der Wirkungsweise der beiden Anweisungen SDISP und ODISP befindet sich auf Seite 243.

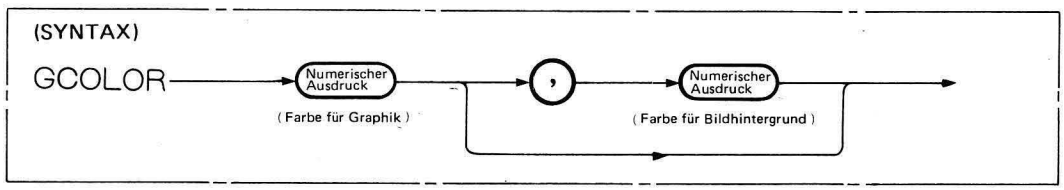
GCOLOR

Bestimmt die Farbe des graphischen Musters*) und des Bildhintergrunds für die im Eingabemodus befindliche Graphikebene.

〈VERWENDUNG〉
① Zur Festlegung der Farben von graphischen Darstellungen und Bildhintergrund.
② Um bei gleichbleibendem Farbwunsch auf eine ständige Wiederholung der Farbangaben in den Anweisungen zum Erzeugen farbiger Graphik (z.B. PSET, LINE, CIRCLE u.a.) verzichten zu können.

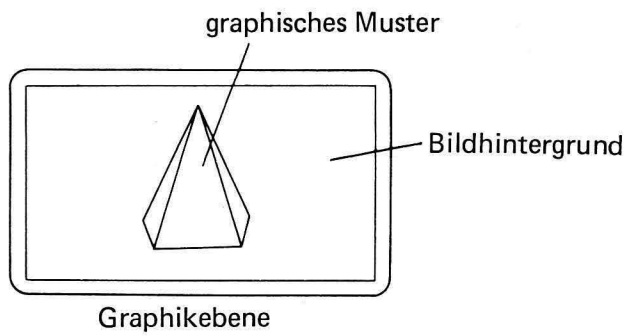
FORMAT: GCOLOR X 〔, Y〕

X : Farbe des graphischen Musters (0 – 7)
Y : Farbe des Bildhintergrundes (0 – 7)
Farbkodes siehe Anweisung COLOR (Seite 239).



ERLÄUTERUNG:

1. Die GCOLOR-Anweisung bewirkt, daß graphische Muster in der durch den numerischen Ausdruck X und der Bildhintergrund in der durch Y bestimmten Farbe dargestellt werden.



2. Fehlt die Spezifizierung der Bildhintergrundfarbe, wird die in der letzten GCOLOR-Anweisung angegebenee Farbe beibehalten.
3. Bei der Initialisierung des Systems wird Weiß als Farbe für die Graphik und Schwarz als Farbe für den Bildhintergrund eingestellt.

*) Graphisches Muster: Eine Anzahl von Punkten, die auf dem Bildschirm dargestellt werden.

EINFACHES BEISPIELE:

1. `10 GCOLOR 5,3` ..... Stellt für das Graphikmuster die Farbe Rot (der Farbkode für Rot ist 5) und für den Bildhintergrund die Farbe Grün (Farbkode = 3) ein.
2. `10 GCOLOR 5,3` ..... Stellt für das Graphikmuster die Farbe Rot und für den Bildhintergrund die Farbe Grün ein.

   ⋮

   `50 GCOLOR 1` ..... Verändert die Farbe für das Graphikmuster nach Gelb (Farbkode = 1) und läßt die Farbe für den Bildhintergrund unverändert (Grün).

   ⋮

WICHTIGER HINWEIS

1. Die GCOLOR-Anweisung ist nur dann gültig, wenn zuvor mittels der CHANGE DISP-Anweisung (Seite 237) oder der SDISP-Anweisung (Seite 241) der farbige Ausgabemodus ("C") eingestellt worden ist. Sie ist im monochromen Ausgabemodus unwirksam.
2. Ein graphisches Muster wird grundsätzlich in der Farbe ausgegeben, die in der jeweiligen Graphik-Anweisung spezifiziert wurde. Fehlt eine Farbspezifikation, wird das graphische Muster in der durch die GCOLOR-Anweisung bestimmten Farbe ausgegeben.
3. Unterscheidet sich die derzeitige Bildhintergrundfarbe von der Spezifikation in der GCOLOR-Anweisung, wird sie unmittelbar nach Ausführung der GCOLOR Anweisung geändert. Die Spezifikation für die Farbe der graphischen Muster wirkt sich jedoch erst bei solchen Darstellungen aus, die nach der Ausführung der GCOLOR-Anweisung erzeugt werden. Sie besitzt keine Wirksamkeit für Graphikmuster, die bereits erstellt waren.
4. Bei der Überlagerung von farbigen Mustern besitzt das Graphikmuster der zuletzt ausgegebenen Graphikebene Priorität. (Überschneiden sich zwei Kurven, z.B. eine rote und eine früher erzeugte grüne Kurve, dann wird der Punkt, in dem sich die beiden Kurven schneiden, in der Farbe der zuletzt gezeichneten Kurve gesetzt, d.h. er wäre in diesem Fall rot. Die grüne Kurve wäre somit im Schnittpunkt der beiden Kurven "unterbrochen".)
5. Für die Ausgabe einer 8-farbigen Graphik werden eine Graphikkarte und zwei zusätzliche Graphikspeicher benötigt (Option). Nähere Angaben sind dem Bedienerhandbuch zu entnehmen.

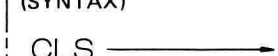
CLS

Löscht graphische Ebenen, die sich im Eingabemodus (ODISP) befinden.

〈VERWENDUNG〉
① Zum vollständigen Löschen einer Graphikebene.
② Zum Löschen eines genau definierten, rechteckigen Feldes in einer Graphikebene.

● **FUNKTION 1:** Vollständiges Löschen einer Graphikebene.
FORMAT: CLS

(SYNTAX)



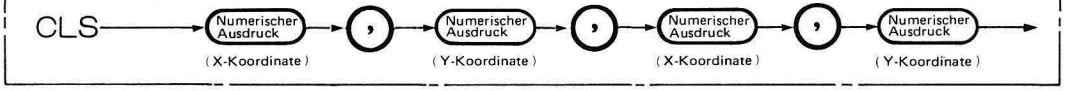
ERLÄUTERUNG:

Die Anweisung CLS löscht graphische Ebenen, die sich im Eingabemodus befinden, ganz.

● **FUNKTION 2:** Partielles Löschen einer Graphikebene.
FORMAT: CLS X, Y, U, V

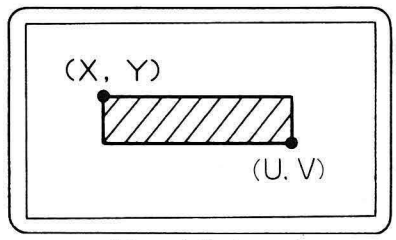
X : X-Koordinate der linken oberen Ecke des zu löschenden, rechteckigen Feldes (siehe Abbildung)
Y : Y-Koordinate der linken oberen Ecke des zu löschenden, rechteckigen Feldes
U : X-Koordinate der rechten unteren Ecke des zu löschenden, rechteckigen Feldes
V : Y-Koordinate der rechten unteren Ecke des zu löschenden, rechteckigen Feldes

(SYNTAX)



ERLÄUTERUNG:

Löscht in einer Graphikebene, die sich im Eingabemodus befindet, ein rechteckiges Feld, das durch seinen linken oberen (X, Y) und rechten unteren (U, V) Eckpunkt in der CLS-Anweisung definiert ist.



(Graphikebene)

EINFACHES BEISPIEL:

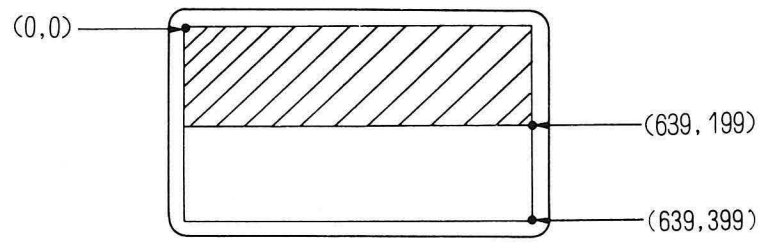
```
10 CLS 10,10,100,100
```
Löscht in einer Graphikebene, die sich im Eingabemodus befindet, ein rechteckiges Feld, dessen linker oberer Eckpunkt die Koordinaten (10, 10), und dessen rechter unterer Eckpunkt die Koordinaten (100, 100) hat.

ANWENDUNGSBEISPIELE:

Programm zum Löschen der oberen Hälfte der, Graphikebene (entspricht der oberen Bildschirmhälfte):

```
10 ODISP "123"      Schaltet den Eingabemodus drei Graphikebenen ein.
20 SCALE OFF        Setzt den Ursprung für die Graphik-Koordinaten in (0, 0).
30 CLS 0,0,639,199  Löscht die obere Hälfte der drei Graphikebenen.
40 END
```

Ergebnis:
(Das schraffierte Feld wurde gelöscht)



(Graphikebene)

Anwendungsbeipiel zur Demonstration der Wirkungsweise der Anweisungen SDISP und ODISP.

```
10 SDISP "0"          Ausgabemodus für Textebene ein-, für Graphikebenen ausgeschaltet.
20 ODISP "123":CLS    Eingabemodus für alle Graphikebenen. einschalten, Löschen der Graphikebenen.
30 SCALE 320,200      Bildschirmmitte = Koordinatennullpunkt.
40 ODISP "1":LINE 0,-100;87,50;-87,50,,F
                      Eingabemodus einschalten für Graphikebene 1, plotten Dreieck.
50 ODISP "2":LINE 100,100;-100,100;-100,-100;100,-100,,F
                      Eingabemodus einschalten für Graphikebene 2, plotten Quadrat.
60 ODISP "3":CIRCLE 0,0,100   Eingabemodus einschalten für Graphikebene 3, plotten Kreis.
70 DISP CLEAR         Löschen der Textebene.
80 INPUT "AUSZUGEBENDE EBENE(N)?";A$   Frage nach String für die Wahl des Ausgabemodus.
90 SDISP A$           Einschalten des Ausgabemodus entsprechend A$.
100 GO TO 80          Schleife zum Wiederholen der Ausgabespezifi-
```

Durch Eingabe z.B. von 1, 2,, 3, 12, 13, 123,C, 0123, . . . können die Graphikebenen einzeln und in beliebiger Überlagerung am Bildschirm ausgegeben werden.

SCALE

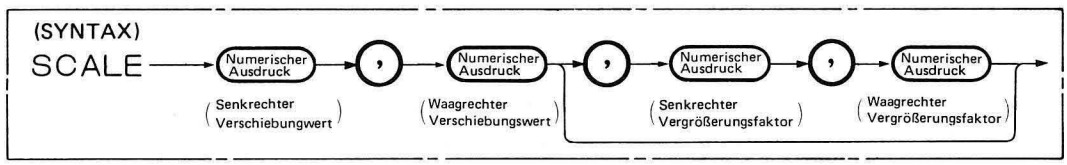
Verändert den Ursprung (Nullpunkt) der Graphik-Koordinaten und deren Skalierung oder schaltet zurück in den Initialisierungszustand.

〈VERWENDUNG〉
① Zum Verschieben des Kordinatenursprungs an eine beliebige Stelle des Graphik-Bildschirms.
② Zur Veränderung der Skalierungseinheiten auf den Achsen.

● **FUNKTION 1:** Bestimmt den Koordinatenursprung und die Skalierung.

FORMAT: SCALE X, Y 〔, U, V〕

X : Veschiebung der X-Achse
Y : Verschiebung der Y-Achse
U : Skalierung der X-Achse
V : Skalierung der Y-Achse



ERLÄUTERUNG:

1. Die SCALE-Anweisung verschiebt den Koordinatenursprung um den Wert des numerischen Ausdrucks für X bzw. Y: nach rechts bei positivem X und nach links bei negativem X, nach unten bei positivem Y und nach oben bei positivem Y. Der Wert von U bestimmt die Skalierungseinheit auf der X-Achse, der Wert von V die Skalierungseinheit auf der Y-Achse. (Beträgt z.B. die Skalierungseinheit für die X-Achse bisher 10 (je graphischer Punkt) und U = 5, dann gilt als neue Skalierungseinheit nach Ausführung der SCALE-Anweisung 50).
2. Fehlen Angaben zur Skalierung, dann wird die bestehende beibehalten. (Beim Einschalten des Geräts wird die Skalierungseinheit = 1 gesetzt.)
3. Nach der Initialisierung liegt der Koordinatenursprung (0, 0) in der linken oberen Bildschirmecke. Der Punkt (639, 399) bezeichnet die rechte untere Bildschirmecke. (Siehe Abbildung.)



(Graphik-Bildschirm)

EINFACHES BEISPIEL:

```
10 SCALE 100,100,10,10
```
.....Bewegt den gegenwärtigen Koordinatenursprung um 100 nach rechts und 100 nach unten und setzt als Skalierungseinheit für beide Achsen den Wert 10.

● **FUNKTION 2:** Zurücksetzen des Koordinatenursprungs und der Skalierungseinheit in den Initialiserungszustand.

FORMAT: SCALE OFF

(SYNTAX)

SCALE OFF ⟶

ERLÄUTERUNG:

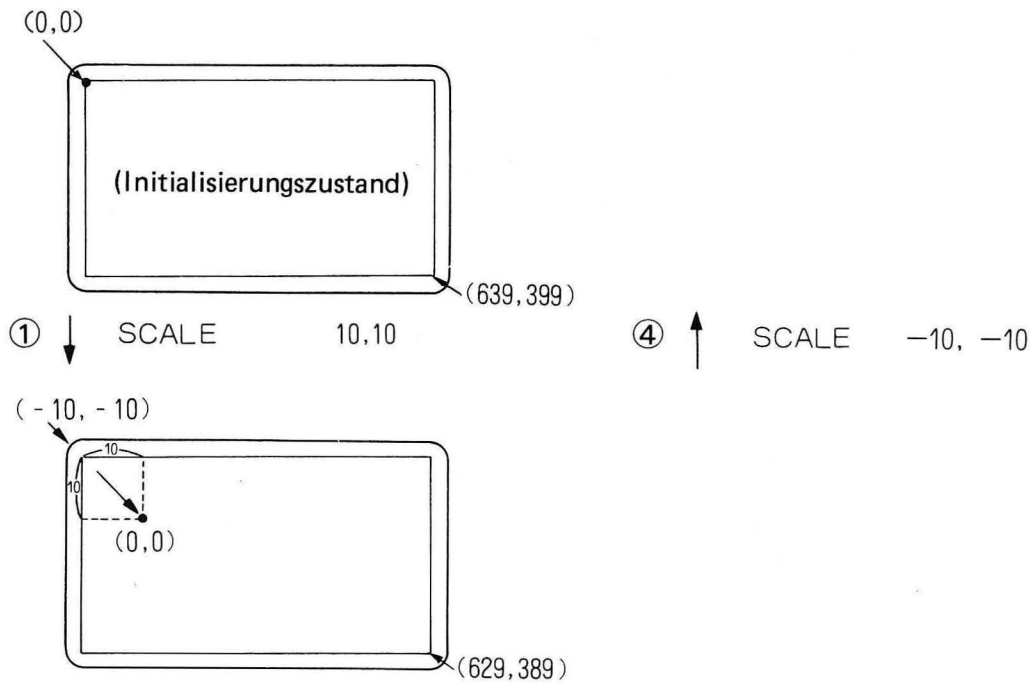
Stellt den Initialisierungszustand für den Koordinatenursprung und die Skalierung der beiden Achsen wieder her. In diesem Zustand liegt der Koordinatenursprung (0, 0) in der linken oberen Ecke des Graphik-Bildschirms. Die rechte untere Ecke entspricht der Position (639, 399).

ANWENDUNGSBEISPIEL:

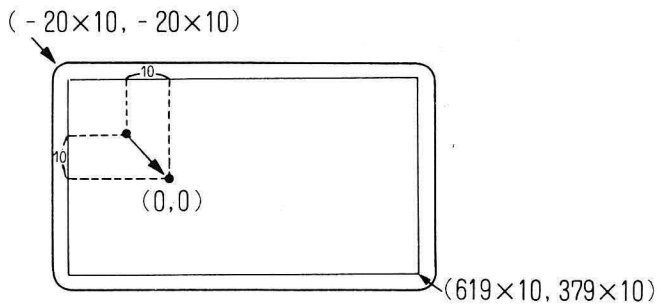
Stelle mittels der SCALE-Zuweisung den Koordinatenursprung und die Skalierung so ein, daß der linke obere Punkt des Graphik-Bildschirms die Koordinaten (–200, –200) und der rechte untere Punkt die Koordinaten (6190, 3790) hat.

Stelle anschließend den Initialisierungszustand wieder her.

Die Schritte zur Lösung dieses Problems werden durch nachfolgende Darstellungsfolge verdeutlicht.



② ↓ SCALE 10,10,10,10 ③ ↑ SCALE −100, −100, 0.1, 0.1



Einfacher wäre es natürlich, zur Rückkehr in den Initialisierungszustand die Anweisung SCALE OFF zu benutzen.

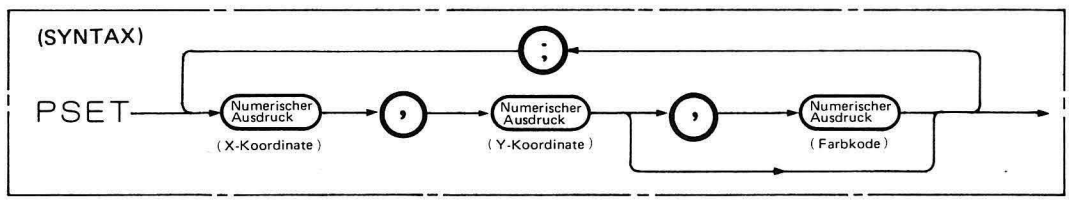
PSET

Setzt einen Punkt in einer im Eingabemodus befindlichen Graphikebene.

〈VERWENDUNG〉
Zum Erzeugen graphischer Muster durch einzelne Punkte.

FORMAT: PSET 《X, Y 〔, Z〕》

X : X-Koordinate
Y : Y-Koordinate
Z : Farbkode (0 – 7)
Farbkodes siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.



ERLÄUTERUNG:

1. Die PSET-Anweisung setzt einen Punkt in den Koordinaten (X, Y) in einer Graphikebene, die sich im Eingabemodus (ODISP) befindet. Die Farbe des Punktes bestimmt sich nach Z.
2. Fehlt eine Farbangabe (Z), dann wird die Farbe entsprechend der letzten GCOLOR-Anweisung bestimmt (Seite 245). Werden mehrere Punkte durch eine PSET-Anweisung gesetzt, dann bestimmt die Farbe für (vorausgehende) Punkte ohne Farbangabe der nächstfolgende Punkt mit Farbangabe. (Siehe Beispiele).

EINFACHES BEISPIELE:

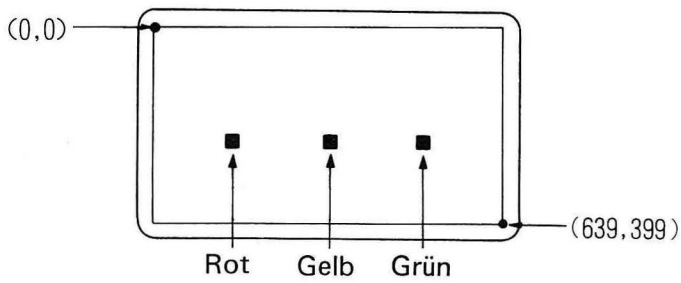
1. `10 PSET 100,100,1` ..... Setzt einen gelben Punkt in der Koordinatenposition (100, 100).
(Farbkode für gelb = 1)
2. `10 PSET 10,10:20,20:30,30,5` .... Setzt drei rote Punkte in den Koordinatenpositionen (10, 10), (20,20) und (30, 30).
(Farbkode für rot = 5)
3. `10 PSET 10,10:10,15,1:10,20` .... Setzt zwei gelbe Punkte in den Koordinationpositionen (10, 10) und (10, 15) und einen Punkt in (10, 20), dessen Farbe durch die letzte GCOLOR-Anweisung bestimmt wird.

ANWENDUNGSBEISPIEL:

Ein Programm für die Ausgabe von drei Quadraten in rot, gelb und grün.

```
10 SDISP "123C":ODISP "123":CLS ····· Einschalten des Eingabemodus und Ausgabemodus für die drei Graphikebenen.
20 FOR I=1 TO 10
30 FOR N=1 TO 10
40 PSET N+69,I+194,5:N+313,I+194,1:N+559,I+194,3
50 NEXT N
60 NEXT I
70 END
```

Anzeige nach Programmausführung:



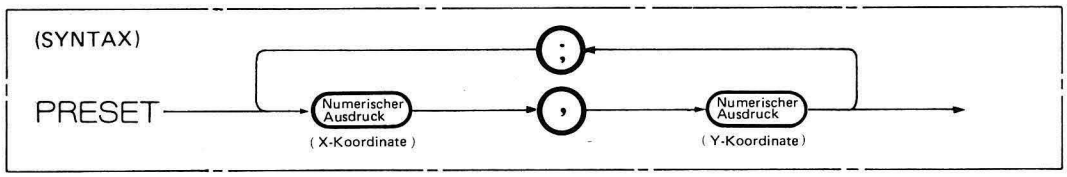
PRESET

Löscht einen Punkt in einer im Eingabemodus befindlichen Graphikebene.

〈VERWENDUNG〉
Zum Löschen eines gesetzten Punktes.

FORMAT: PRESET 《X, Y》

X : X-Koordinate
Y : Y-Koordinate



ERLÄUTERUNG:

Löscht den Punkt (X, Y) in einer sich im Eingabemodus (ODISP) befindlichen Graphikebene (= setzt ihn zurück). War der Punkt nicht gesetzt, bewirkt diese Anweisung nichts.

EINFACHES BEISPIEL:

```
10 PRESET 10,10:100,100
```
..... Löscht die beinden Punkte mit den Koordinaten (10, 10) und (100, 100).

WICHTIGER HINWEIS

Das Löschen eines Punktes mittels der Anweisung PRESET funktioniert bei monochromer Arbeitsweise (z.B. mit einem grünen Monitor) nur dann, wenn über die Anweisung CHANGE DISP (Seite 237) oder SDISP (Seite 241) der Farbmodus eingeschaltet worden ist.

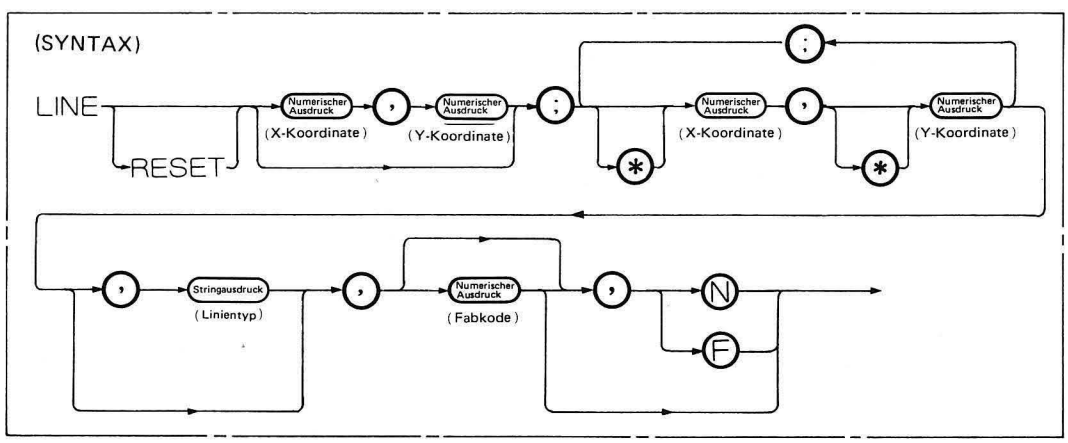
LINE

Zeichnet oder löscht eine Strecke in einer sich im Eingabemodus befindlichen Graphikebene.

〈VERWENDUNG〉
Zum Erzeugen oder Löschen einer Strecke oder eines Streckenzugs (Polygon).

FORMAT: L I NE 〔RESET〕 〔X, Y〕 ; ≪〔*〕U, 〔*〕V≫ 〔, C〕〔, Z | 〔Z〕, CH〕

RESET : Einschalten des Löschmodus
X : X-Koordinate des ersten Punktes
Y : Y-Koordinate des ersten Punktes
* : Angabe relativer Koordinaten
U : X-Koordinate der Folgepunkte
V : Y-Koordinaten der Folgepunkte
C : Linientyp
C ist wie folgt anzugeben:
A : Gepunktete Linie ----------
B : Einpunkt-Linie ——·——·——
C : Zweipunkt-Linie ——··——··——
(C ausgelessen) : Durchgezogene Linie ——————
Z : Farbkode (0 – 7)
Farbkodes siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.
CH : Verbindungsmodus
N : Verbindet die angegebenen Koordinatenpunkte miteinander.
Normalmodus. (Der erste Punkt wird nicht mit dem latzten Punkt verbunden.)
F : Wie Verbindungsmodus N, aber verbindet zusätzlich den letzten mit dem ersten Punkt.



ERLÄUTERUNG:

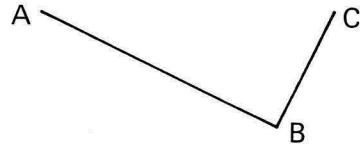
1. Die LINE-Anweisung verbindet in einer sich im Eingabemodus befindlichen Graphikebene die Koordinatenpunkte (U, V), beginnend bei dem Koordinatenpaar (X, Y), durch Strecken in dem durch CH gekennzeichneten Verbindungsmodus.
   Der Parameter C bezeichnet den Linientyp und Z die Farbe, in der die Strecke dargestellt werden soll. Wird RESET mit ausgegeben, wird eine Gerade des bezeichneten Typs gelöscht.
2. Das Symbol * kennzeichnet relative Koordinaten (relativ in Beziehung zum Anfangspunkt der Strecke). So könnte man z. B. die Anweisung

   ```
   LINE 100,100;101,99
   ```

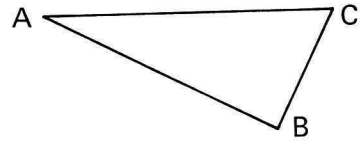
   unter Verwendung relativer Koordinaten wie folgt ausdrücken:

   ```
   LINE 100,100;*1,*-1
   ```

3. Der Verbindungsmodus zeigt an, wie die Koordinatenpaare miteinander zu verbinden sind. Enthält die LINE-Anweisung z.B. drei Koordinatenpaare zur Kennzeichnung der Punkte A, B und C, dann wird.
   (1) im N-Modus jeder Punkt mit seinem Nachfolger verbunden:



   (2) im F-Modus jeder Punkt mit seinem Nachfolger und zusätzlich der letzte mit dem ersten Punkt verbunden:

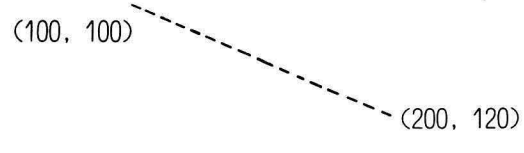


4. Fehlt die Angabe eines ersten Punktes (X, Y), so wird der Endpunkt der vorausgehenden LINE-Anweisung als Startpunkt genommen.
   Geht keine LINE-Anweisung voraus, wird der Koordinatenursprung (0, 0) als Startpunkt genommen.
5. Ist kein Linientyp C angegeben,erzeugt das System eine durchgezogene Linie.
6. Fehlt die Farbspezifikation Z, bestimmt die letzte GCOLOR-Anweisung die Farbe.
7. Bei fehlendem Verbindungsmodus CH wird der N-Modus gewählt.
8. Soll eine Strecke gelöscht werden, ist dies über den Parameter RESET in der LINE-Anweisung möglich. In diesem Fall ist eine durchgehende Linie zu spezifizieren und auf die Angabe eines Farbkodes zu verzichten. Der gelöschte Teil erhält die Hintergrundfarbe, die durch die GCOLOR-Anweisung spezifiziert wurde.

EINFACHES BEISPIELE:

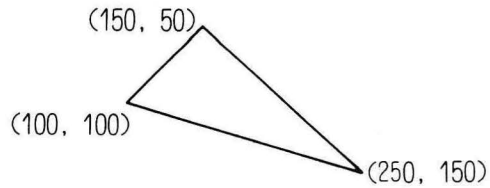
1.
```
10 LINE 100,100;200,120,"A",3,N
```

..... Verbindet den Startpunkt (100, 100) mit dem Endpunkt (200, 120) im Verbindungsmodus N mit einer gepunkteten, grünen Strecke (Farbkode 3).



2. `10 LINE 100,100:*50,*-50:250,150,,F`

..... Verbindet nacheinander die Punkte (100, 100), (150, 50) und (250, 150) im Verbindungsmodus F. Die Farben bestimmen sich nach der GCOLOR-Anweisung.



3. `10 LINE RESET 100,100:200,120`

..... Löscht die Strecke des ersten Beispiels.

4. `10 LINE RESET 100,100:*50,*-50:250,150,,F`

..... Löscht die Strecke des zweiten Beispiels.

WICHTIGER HINWEIS

1. Wird in der LINE-Anweisung durch den Parameter RESET der Löschmodus aktiviert, so werden die angegebenen Strecken unabhängig von der ausgewiesenen Farbe gelöscht. Daher kann man im Löschmodus auf eine Farbangabe verzichten.
2. Fehlt in der LINE-Anweisung die Angabe eines Startpunktes (X, Y), wird der Endpunkt der vorausgehenden LINE-Anweisung als Startpunkt genommen. In der vorausgehenden LINE-Anweisung bestimmt der Verbindungsmodus CH den Endpunkt; im Normalmodus (N) bildet das letzte angegebene Koordinatenpaar (U, V), im Verbindungsmodus F jedoch der erste Punkt (X, Y) den Endpunkt.
3. Der Löschmodus mittels RESET funktioniert bei einfarbiger Arbeitsweise (z.B. mit einem grünen Monitor) nur dann, wenn über die Anweisung CHANGE DISP (Seite 237) oder SDISP (Seite 241) der Farbmodus eingeschaltet worden ist.

ANWENDUNGSBEISPIEL:

Dieses Programm ermöglicht die graphische Darstellung der individuell erbrachten, individuellen Leistungen eines Stundenten z.B. in Fächern eines Grundstudiums. Die erzielten Punkte eines Kandidaten werden über die Tastatur eingegeben. Die maximal erzielbare Punktzahl in einem Fach sei 100.

Den Fächern werden folgende Farben zugeordnet: Englisch=Gelb, Mathematik=Türkis, Linguistik=Grün, Naturwissenschaften=Pink, Sozialkunde=Rot.

```
10 SDISP "0123C":ODISP "123"
20 DIM A%(5)
30 INPUT "ENGLISCH";A%(1)
40 INPUT "MATHEMATIK";A%(2)
50 INPUT "LINGUISTIK";A%(3)
60 INPUT "NATURWISSENSCHAFTEN";A%(4)
70 INPUT "AOZIALKUNDE";A%(5)
80 DISP CLEAR
90 CLS
100 LINE 0,300:639,300,0
110 FOR I=1 TO 5
120 A%(I)=A%(I)*2
130 LINE 100*I,300:*0,*-A%(I):*40,*0:300,I,N
140 NEXT
150 END
```

Einschalten des farbigen Ein- und Ausgabemodus für alle drei Graphikebenen.

Eingabe der erzielten Punkte in den 5 Fächern.

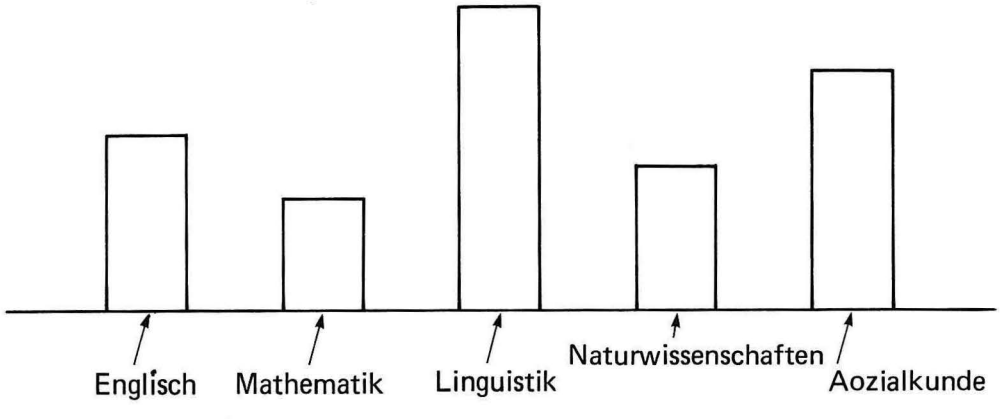
Löscht den Bildschirm.

Zeichnet eine Grundlinie in weiß.

Gibt man nach dem Starten des Programms z.B. folgende Punktzahlen ein:

(Englisch)	60
(Mathematik)	35
(Linguistik)	80
(Naturwissenschaften)	45
(Aozialkunde)	70

dann erhält man folgende Bildschirmausgabe



CRT 1

CIRCLE

Zeichnet oder löscht einen Kreis, eine Ellipse bzw. Kreis oder Ellipsensektor.

〈VERWENDUNG〉
Zum Zeichnen von Kreisen, Ellipsen oder Sektoren.

● **FUNKTION 1:** Zeichnen oder Löschen eines Kreises oder einer Ellipse.

FORMAT: CIRCLE [RESET] X, Y, R [, W] [; ; Z | [Z], CH]

RESET : Löschmodus

X : X-Koordinate des Kreis(Ellipsen-)mittelpunktes

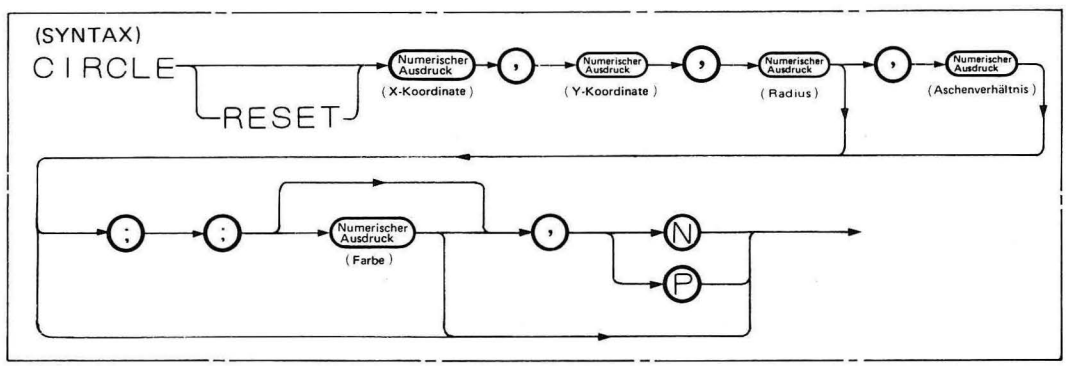
Y : Y-Koordinate des Kreis(Ellipsen-)mittelpunktes

R : Kreisradius oder große Halbachse der Ellipse

W : Relation der Nebenachse zur Hauptachse (Achsenverhältnis)
Achsenverhältnis W < 1 : horizontale Ellipsenhauptachse
Achsenverhältnis W ≧ 1 : vertikale Ellipsenhauptachse

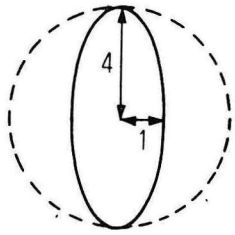
Z : Farbkode (0 – 7)
Siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.

CH : Darstellungsmodus
N ····· Zeichnet eine Kreis- oder Ellipsenlinie (Normalmodus).
P ····· Zeichnet eine Kreis- oder Ellipsenfläche (Malmodus).

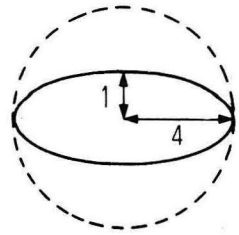


ERLÄUTERUNG:

1. Fehlt die Angabe des Achsenverhältnisses W, wird ein Kreis mit dem Durchmesser R gezeichnet, dessen Zentrum im Punkt (X, Y) des Graphikkoordinatensystems liegt. Seine Farbe bestimmt der Parameter Z. Soll er ausgemalt werden, wählt man als Darstellungsmodus P, sonst N.
2. Ist ein Wert für das Achsenverhältnis W angegeben, wird eine Ellipse gezeichnet. Ist W größer als 1, verläuft die Ellipsenhauptachse in senkrechter, ist W kleiner als 1, in waagrechter Richtung (siehe dazu die folgende Abbildung).



W = 4 (> 1)



W = 0.25 (< 1)

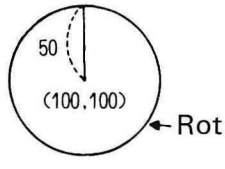
Eine Umkehrung des Achsenverhältnisses W (1/W) dreht die Ellipse um 90 Grad.

3. Fehlt die Angabe des Farbkodes, erfolgt die Ausgabe in der durch die GCOLOR-Anweisung bestimmten Farbe.
4. Fehlt eine Angabe zum Darstellungsmodus CH, wird der Normalmodus N genommen. (Siehe Beispiele).
5. Ist in der CIRCLE-Anweisung RESET angegeben, wird der spezifizierte Kreis (bzw. die Ellipse) gelöscht. (Bei farbiger Anzeige erhalten die gelöschten Teile die Farbe des Graphikhintergrunds.)

EINFACHES BEISPIELE:

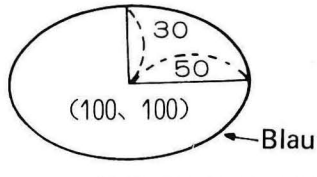
1. `10 CIRCLE 100,100,50;;5,N`

..... Zeichnet einen Kreis mit Radius 50 in Rot im Normalmodus. Der Kreismittelpunkt ist (100, 100).



2. `10 CIRCLE 100,100,50,0.6;;6,P`

..... Zeichnet eine blaue Ellipse mit einer großen Halbachse von 50 und einem Achsenverhältnis von 0.6. Der Ellipsenmittelpunkt ist (100, 100). Die Fläche der Ellipse wird ausgemalt (Darstellungsmodus P).



3. `20 CIRCLE RESET 100,100,50`

..... Löscht den im ersten Beispiel dargestellten Kreis.

4. `20 CIRCLE RESET 100,100,50,0.6;;,P`

..... Löscht die im zweiten Beispiel dargestellte Ellipse.

● **FUNKTION 2:** Zeichnet oder löscht einen Kreis- oder Ellipsensektor.

FORMAT: C I RCLE 〔RESET〕 X, Y, R 〔, W〕 ; U | 〔U〕, V 〔; Z | 〔Z〕〔, CH〕〕

RESET : Löschmodus

X : X-Doordinate des Kreis(Ellipsen-)mittelpunktes

Y : Y-Koordinate des Kreis(Ellipsen-)miitelpunktes

R : Kreisradius oder große Halbachse der Ellipse

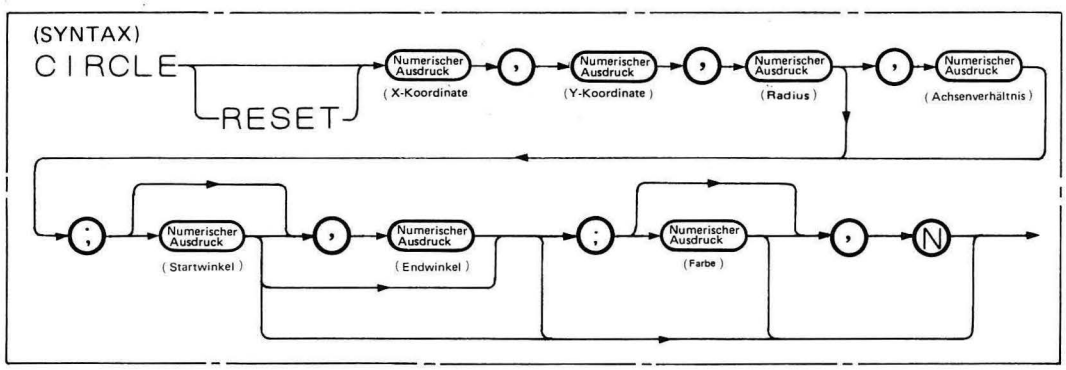
W : Relation der Nebenachse zur Hauptachse (Achsenverhältnis)
Achsenverhältnis W < 1 : horizontale Ellipsenhauptachse
Achsenverhältnis W ≧ 1 : vertikale Ellipsenhauptachse

U : Beginn des Sektorwinkels (0 – 360)

V : Ende des Sektorwinkels (0 – 360)

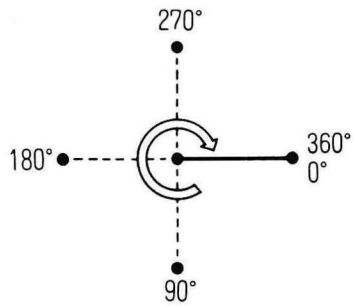
Z : Farbkode (0 – 7)
Farbkodetabelle siehe Anweisung COLOR (Seite 239).

CH : Darstellungsmodus
N ····· Zeichnet einen Sektor durch Verbinden des Kreismittelpunktes mit den beiden Endpunkten des Kreisbogens (analog bei der Ellipse).



ERLÄUTERUNG:

1. Fehlt in der CIRCLE-Anweisung das Achsenverhältnis, wird ein Kreisbogen in der Farbe Z gezeichnet. Der Bogenwinkel beginnt bei U Grad (Startwinkel) und endet bei V Grad (Endwinkel). Der Kreismittelpunkt hat die Koordinatenposition (X, Y).
2. Ist das Achsenverhältnis W angegeben, wird analog zum Kreisbogen ein Teil der Ellipsenlinie gezeichnet.
3. Ist in dem Darstellungsmodus CH als N spezifiziert (die Angabe von P ist nicht möglich), wird der Kreis- bzw. Ellipsenmittelpunkt mit den Endpunkten des gezeichneten Bogens verbunden (Sektor). Fehlt die Spezifikation des Darstellungsmodus, wird nur der Bogen gezeichnet.
4. Der Zählmodus von Start- und Endwinkel U und V ergibt sich aus der folgenden Darstellung. Beide Winkel sind ganzzahlig und innerhalb eines Intervalls von 0 – 360 Grad zu spezifizieren. U kann größer als V sein.

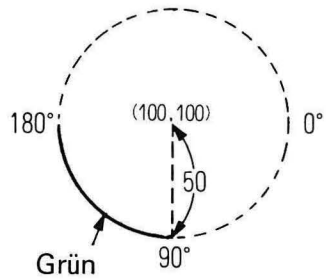


5. Fehlt der Startwinkel U, wird ein Winkel von 0 Grad angenommen; fehlt der Endwinkel V, wird dafür ein Winkel von 360 Grad gesetzt.
6. Fehlt die Angabe des Farbkodes Z, wird entsprechend der letzten GCOLOR-Anweisung (Seite 245) verfahren.
7. Bei Angabe von RESET wird der entsprechende Kreis- bzw. Ellipsensektor gelöscht. Der Farbkode kann in diesem Fall entfallen. (Der gelöschte Teil erhält die durch die GCOLOR-Anweisung bestimmte Hintergrundfarbe.)

EINFACHES BEISPIELE:

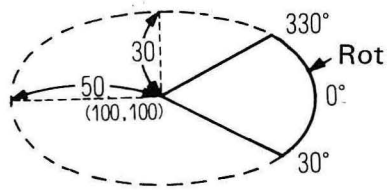
1. `10 CIRCLE 100,100,50;90,180,3`

..... Zeichnet einen grünen Kreisbogen mit Radius 50 vom Startwinkel 90 zum Endwinkel 180. Der Kreismittelpunkt liegt in (100, 100).



2. `10 CIRCLE 100,100,50,0.6;330,30,5,N`

..... Zeichnet einen Ellipsensektor mit einer großen Halbachse von 50 und einem Achsenverhältnis von 0.6 vom Startwinkel 330 bis zum Endwinkel 30 in Blau. Der Ellipsenmittelpunkt liegt in (100, 100).



3. `20 CIRCLE RESET 100,100,50;90,180`

..... Löscht den Kreisbogen aus dem ersten Beispiel.

4. `20 CIRCLE RESET 100,100,50,0.6;330,30`

..... Löscht den Ellipsensektor aus dem zweiten Beispiel.

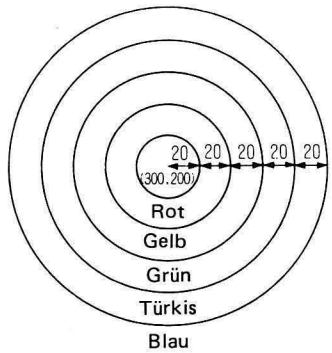
WICHTIGER HINWEIS

1. Bei der Angabe des Löschmodus RESET in der CIRCLE-Anweisung ist zu beachten: Hat man eine Kreis- oder Ellipsenfläche im Ausmalmodus (P) erzeugt und verbindet nun die Löschanweisung für diesen Kreis oder diese Ellipse mit dem Darstellungsmodus N (Normalmodus), dann wird nur die Kreis- oder Ellipsenlinie gelöscht. Die Fläche dagegen bleibt ungelöscht. (Es erfolgt keine Fehlermeldung.)
   Hat man dagegen eine Kreislinie, eine Ellipsenlinie oder einen Sektor im Linienmodus (N) erzeugt und verbindet die Löschanweisung hierfür mit dem Darstellungsmodus P (Malmodus), dann werden durch den Löschvorgang auch eventuell vorhandene graphische Darstellungen innerhalb des Kreises, der Ellipse oder des Sektors mitgelöscht.
2. Der Löschmodus mittels RESET funktioniert bei monochromer Arbeitsweise (z.B. mit einem grünen Monitor) nur dann, wenn über die Anweisung CHANGE DISP (Seite 237) oder SDIPS (Seite 241) der Farbmodus eingeschaltet worden ist.

ANWENDUNGSBEISPIELE:

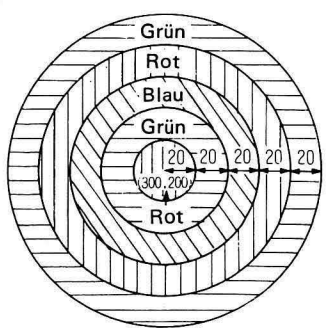
Es sind fünf konzentrische Kreise in den Farben (beginnend von innen) rot, gelb, grün, fürkis und blau zu zeichnen.

```
10 GDISP "123C":ODISP "123":CLS   Initialisieren und Leeren der Graphikebenen.
20 X=300:Y=200:R=20               Kreiszentrum (300, 200). Ringbreite 20.
30 CIRCLE X,Y,R;;5                Kreis(linie) mit Radius 20 in Rot.
40 CIRCLE X,Y,R*2;;1              Kreis(linie) mit Radius 40 in Gelb.
50 CIRCLE X,Y,R*3;;3              Kreis(linie) mit Radius 60 in Grün.
60 CIRCLE X,Y,R*4;;2              Kreis(linie) mit Radius 80 in Türkis.
70 CIRCLE X,Y,R*5;;6              Kreis(linie) mit Radius 100 in Blau.
80 END
```



Im nächsten Programm sollen die Kreise im Malmodus (P) ausgegeben werden. Dadurch werden die Flächen zwischen den Kreislinien (Ringe) farbig gefüllt. Die Ringe sollen zuletzt folgende Farben haben (von innen): rot, grün, blau, rot, grün. Zwischen der Ausgabe der Kreise soll eine Pause von 5 Sekunden liegen.

```
10 GDISP "123C":ODISP "123":CLS   Initialisieren und Leeren der Graphikebenen.
20 X=300:Y=200:R=20               Kreiszentrum (300, 200). Ringbreite 20.
30 CIRCLE X,Y,R*5;;3,P:WAIT 50    Kreis mit Radius 20 in Grün.
40 CIRCLE X,Y,R*4;;5,P:WAIT 50    Kreis mit Radius 40 in Rot.
50 CIRCLE X,Y,R*3;;6,P:WAIT 50    Kreis mit Radius 60 in Blau.
60 CIRCLE X,Y,R*2;;3,P:WAIT 50    Kreis mit Radius 80 in Grün.
70 CIRCLE X,Y,R;;5,P:WAIT 50      Kreis mit Radius 100 in Rot.
80 END
```



PAINT

Ausmalen einer Figur.

〈VERWENDUNG〉
Zum (farbigen) Ausmalen einer Fläche, die innerhalb einer geschlossenen Kurve liegt.

FORMAT: PAINT X, Y, Z 〔, 〔U〕〕〔; V〕

X : X-Koordinate für den PAINT-Bezugspunkt

Y : Y-Koordinate für den PAINT-Bezugspunkt

Mit Bezugspunkt bezeichnen wir einen beliebigen Punkt innerhalb der auszumalenden Figur. Durch seine Angabe ist somit die auszumalende Figur eindeutig bezeichnet.

Z : Farbkode (0 – 7)

Farbkodetabelle siehe Anweisung COLOR (Seite 239).

Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.

U : Farbkode der Begrenzungslinie (0 – 7)

V : Farbdichte

0 ·····100% Dichte

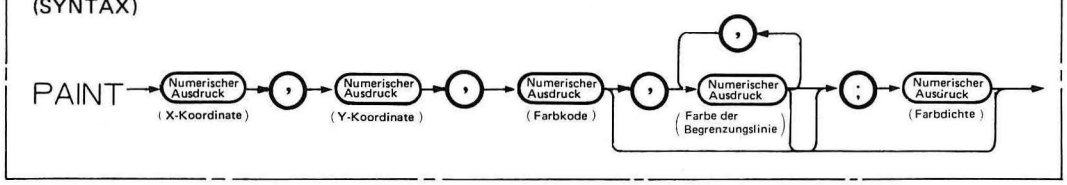
1 ····· 75% Dichte

2 ····· 50% Dichte

3 ····· 25% Dichte

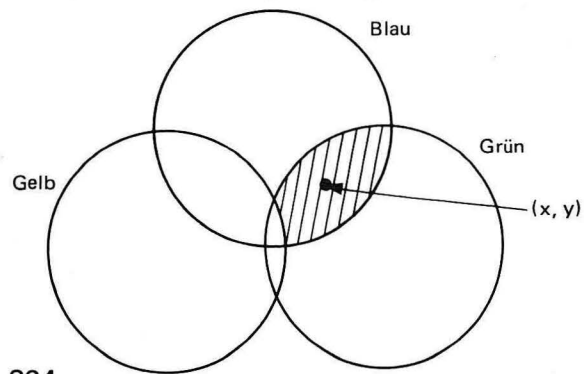
Diese Spezifikation wirkt sich nicht beim Füllen einer Fläche mit Schwarz aus.

(SYNTAX)



ERLÄUTERUNG:

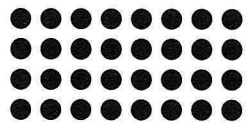
1. Die PAINT-Anweisung beginnt mit dem Ausmalen einer Fläche in dem durch die Koordinaten (X, Y) definierten Bezugspunkt und füllt sei in der Farbe Z bis zu den durch U spezifizierten farbigen Begrenzungslinien. Die Farbdichte bestimmt der Parameter V. Die Begrenzungslinie selbst wird nicht übermalt.
2. Für die Begrenzungslinie(n) kann eine beliebige Anzahl von Farben benannt werden. Die kleinste Figur, die von den durch U spezifizierten Begrenzungslinien umschlossen wird, wird ausgemalt.
(Siehe Abbildung).
Der Bezugspunkt sei (x, y) und die Begrenzungslinien durch die Farben Gelb, Grün und Blau bestimmt. Dann würde in nebenstehender Darstellung die schraffierte Figur "ausgemalt".



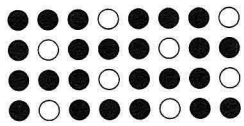
3. Mit Farbdichte bezeichnen wir das Verhältnis der farbig angezeigten zu den gesamten Punkten einer Fläche. Die Farbdichte wird durch den Parameter V bestimmt. Den Werten von V entsprechen folgende Relationen:

   Punkt
   farbig angezeigter Punkt

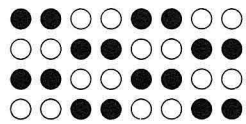
   V = 0 : 100% der Punkte werden farbig angezeigt (gefüllt)



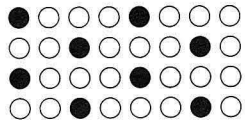
   V = 1 : 75% der Punkte werden farbig angezeigt



   V = 2 : 50% der Punkte werden farbig angezeigt



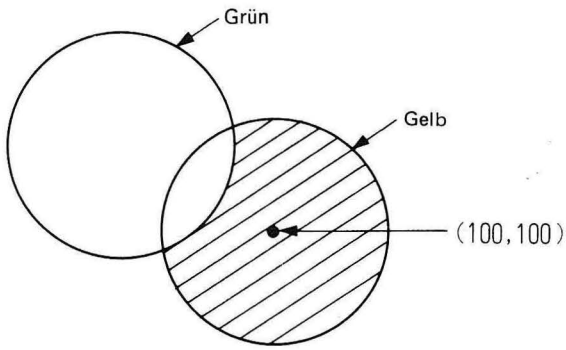
   V = 3 : 25% der Punkte werden farbig angezeigt



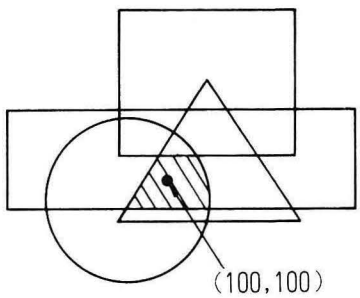
4. Ist für den Parameter U (Farbe der Begrenzungslinien) kein Wert angegeben, wird jede Linie, gleich welcher Farbe, als Begrenzung angenommen.
5. Ist keine Angabe zur Farbdichte (V) gemacht, wird V = 0 gesetzt (100% Dichte).

EINFACHES BEISPIELE:

1. `10 PAINT 100,100,5,1,3:0` ..... Beginnt im Bezugspunkt (100, 100) und füllt die Figur bis zu den grünen und gelben Begrenzungslinien mit Rot. (In der Abbildung wurde die rote Figur schraffiert dargestellt.)



2. `10 PAINT 100,100,1:2` ..... Beginnt im Bezugspunkt (100, 100) und füllt die Figur innerhalb der nächstliegenden Begrenzungslinien mit 50% Dichte in Gelb. (Da in diesem Beispiel keine Angabe zur Farbe der Begrenzungslinien gemacht wurde, füllt die Anweisung nur die kleinste, durch die Linien, gleich welcher Farbe, begrenzte Fläche aus.)

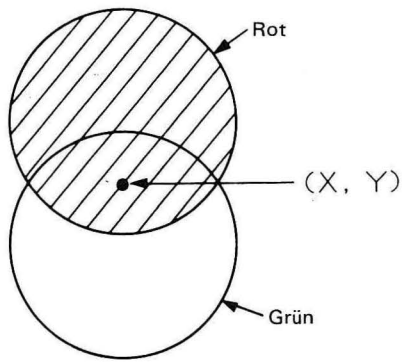


WICHTIGER HINWEIS

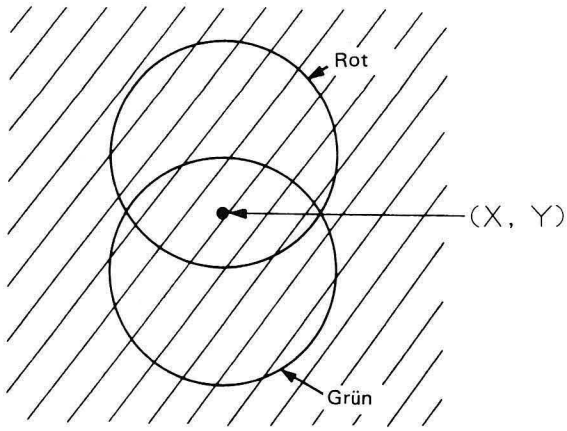
Wenn sich Kurven überlagern, hängt die Ausmalung von der Überlagerungsfolge und von der Farbe der Kreuzungspunkte ab. (Siehe hierzu auch bei WICHTIGER HINWEIS, Punkt 4, auf Seite 264.) Der Zusammenhang soll anhand der folgenden Darstellungen verdeutlicht werden.

Wir machen folgende Annahme: der Bezugspunkt sei (X, Y) und Rot als Farbe der Begrenzungslinie spezifiziert.

1. Überlagert die rote die grüne Kurve, dann entspricht die ausgemalte der schraffierten Fläche:



2. Überlagert dagegen die grüne Kurve die rote, "läuft" die Ausmalung "über":

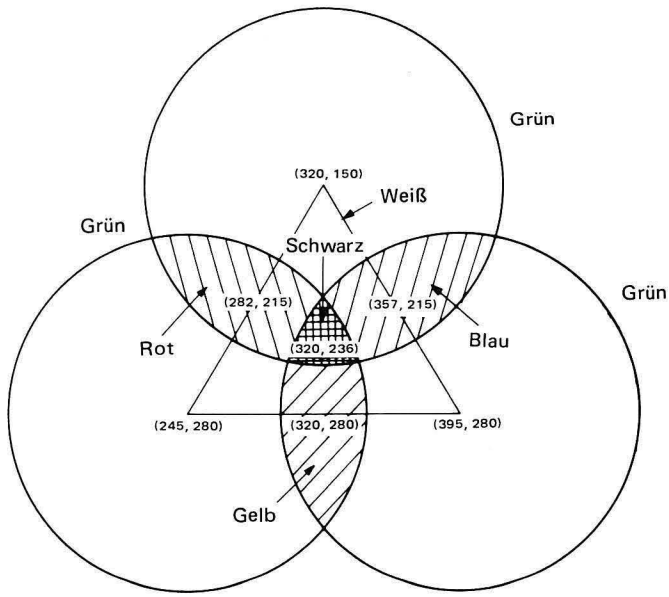


Ganz allgemein läßt sich sagen: existiert in der angegebenen Begrenzungslinie ein andersfarbiger Punkt, dann ist die Kurve in diesem Punkt geöffnet und hat ihre Wirkung als Begrenzungslinie für das Ausmalen einer Fläche verloren.

ANWENDUNGSBEISPIEL:

Man erstelle ein Programm, das die folgende Abbildung als Farbgraphik über den Bildschirm ausgibt. (Drei grüne Kreise, deren Mittelpunkte durch weiße Linien verbunden sind. Die Linien bilden ein gleichseitiges Dreieck. Die gemeinsamen Schnittflächen der Kreise sind auszumalen, indem wir für jede Fläche einen Bezugspunkt (beliebiger Punkt innerhalb der Fläche), die Farbe und Farbdichte angeben.)

Fläche mit dem Bezugspunkt (282, 215) in Rot mit 50% Farbdichte.
Fläche mit dem Bezugspunkt (320, 280) in Gelb mit 50% Farbdichte.
Fläche mit dem Bezugspunkt (357, 215) in Blau mit 50% Farbdichte.
Fläche mit dem Bezugspunkt (320, 236) in Weiß mit 100% Farbdichte.



```
10 SDISP "123C":ODISP "123":CLS
20 LINE 320,150;245,280;395,280,0,F
30 CIRCLE 320,150,100;;3
40 CIRCLE 245,280,100;;3
50 CIRCLE 395,280,100;;3
60 PAINT 320,236,0,3;0
70 PAINT 282,215,5,3;2
80 PAINT 320,280,1,3;2
90 PAINT 357,215,6,3;2
100 END
```

Initialisierung der Graphikebenen.

Zeichnet ein weißes Dreieck durch Verbinden der Kreismittelpunkte mit Linien.

Zeichnet einen grünen Kreis mit Radius 100, dessen Mittelpunkt in (320, 150) liegt.

Zeichnet einen grünen Kreis mit Radius 100, dessen Mittelpunkt in (245, 280) liegt.

Zeichnet einen grünen Kreis mit Radius 100, dessen Mittelpunkt in (395, 280) liegt.

Malt die Fläche um den Bezugspunkt (320, 236) innerhalb der grünen Begrenzunglinien in Weiß mit der Dichte 0 (100%) aus.

Malt die Fläche um den Bezugspunkt (282, 215) innerhalb der grünen Begrenzungslinien in Rot mit der Dichte 2 (50%) aus.

Malt die Fläche um den Bezugspunkt (320, 280) innerhalb der grünen Begrenzungslinien in Gelb mit der Dichte 2 (50%) aus.

Malt die Fläche um den Bezugspunkt (357, 215) innerhalb der grünen Begrenzungslinien in Blau mit der Dichte 2 (50%) aus.

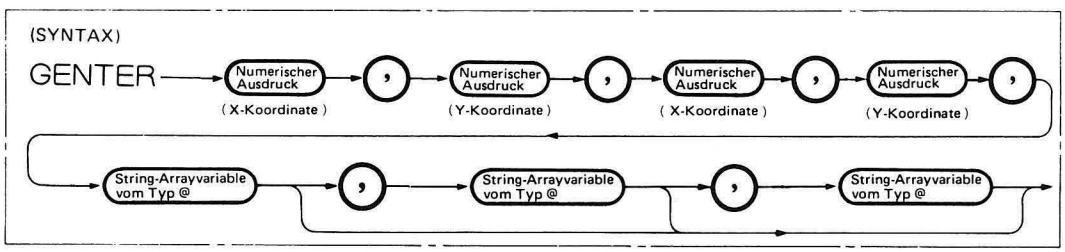
GENTER

Weist einer Arrayvariablen die Graphikdaten eines genau definierten Bereichs einer Graphikebene zu.

〈VERWENDUNG〉
Speicher von graphischen Daten aus einem genau definierten, rechteckigen Bereich einer Graphikebene in einer Stringarrayvariablen Ausgabe über Bildschirm oder Drucker.

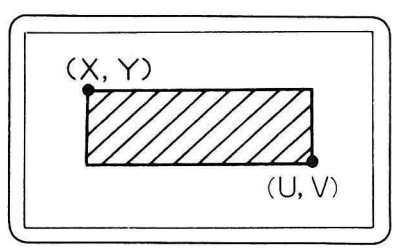
FORMAT: GENTER X, Y, U, V, A@(*) [, B@(*) [, C@(*)]]

X : X-Koordinate der linken oberen Ecke des zuzuordnenden Bereichs
Y : Y-Koordinate der linken oberen Ecke des zuzuordnenden Bereichs
U : X-Koordinate der rechten unteren Ecke des zuzuordnenden Bereichs
V : Y-Koordinate der rechten unteren Ecke des zuzuordnenden Bereichs
A @ (*) : kollektive Stringarrayvariable des @-Typs
B @ (*) : kollektive Stringarrayvariable des @-Typs
C @ (*) : kollektive Stringarrayvariable des @-Typs



ERLÄUTERUNG:

1. Liest die graphischen Daten eines rechteckigen Bereichs (schraffierter Teil in der Abbildung), dessen linke obere Ecke durch die Koordinaten (X, Y) und dessen linke untere Ecke durch die Koordinaten (U, V) bestimmt sind, und weist sie der kollektiven String-Array-Variablen zu.



[Graphikebene]

2. Sollen die graphischen Daten dreier (überlagerter) Graphikebenen gleichzeitig gelesen werden, muß entsprechende Zuordnung zu drei kollektiven @-Typ-Stringarrayvariablen A@(*), B@(*) und C@(*) erfolgen.

3. Die Größe einer kollektiven @-Typ-Stringarrayvariablen A@(*), wie sie in der DIM-Anweisung zu deklarieren ist – und analog die Größe der Variablen B@(*) und C@(*) – wird wie folgt bestimmt:

   monochromer Modus: A@(V – Y + 1) *m (1-dimensional)
   Farbmodus : A@(3, V – Y + 1) *m (2-dimensional)

   mit m = (U–X+1)/8 (m ist aufzurunden!)

4. Das Graphikmuster, das den Variablen zugewiesen worden ist, kann mit der GDISP-Anweisung über den Bildschirm oder mit der GPRINT-Anweisung über den Drucker ausgegeben werden.

EINFACHES BEISPIELE:

1. `20 GENTER 10,20,100,120,A@(*)`
   ..... Liest graphische Daten aus dem Bereich einer Graphikebene, dessen linker oberer Eckpunkt die Koordinaten (10, 20) und dessen rechter unterer Eckpunkt die Koordinaten (100, 120) hat, und weist sie der kollektiven @-Typ-Stringarrayvariablen A@(*) zu.

   Bevor diesen Anweisung durchgeführt werden kann, ist die Größe von A@(*) in einer DIM-Anweisung zu deklarieren (Seite 128):

   a) monochromer Modus:
      `DIM A@(101)*12` ..... Deklaration der @-Typ-Stringarrayvariablen A@.
      Zahl der Elemente = 120–20+1 = 101
      Größe eines Elements = m
      m = (100–10+1)/8 = 11.375
      Aufrunden : m = 12

   b) Farbmodus:
      `DIM A@(3,101)*12` ..... Deklaration einer zweidimensionalen @-Typ-Stringarrayvariablen.

2. 20 GENTER 10, 10, 450, 280, A@(*), B@(*), C@(*)

..... Liest nacheinander aus einem Bereich aller drei Graphikebenen, dessen linke obere Ecke im Punkt (10, 10) und dessen rechte unterer Ecke im Punkt (450, 280) liegt, graphische Daten. Die Zuweisung an die kollektiven Stringarrayvariablen erfolgt wie folgt:
Die graphischen Daten der Graphikebene 1 werden der Variablen A@(*), die graphischen Daten der Graphikebene 2 der Variablen B@(*) und die graphischen Daten der Graphikebene der Variablen C@(*) zugewiesen.

WICHTIGER HINWEIS

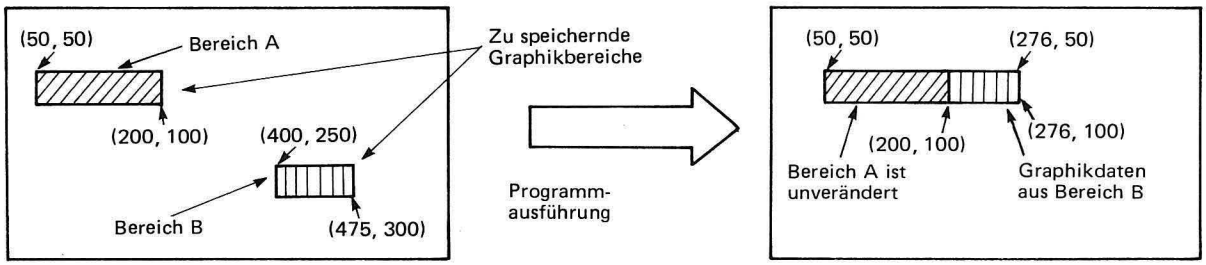
1. Wurde die Größe von A@(*) in der DIM-Anweisung zu klein angegeben, um alle graphische Daten aus einem spezifizierten Bereich aufnehmen zu können, dann werden ab Beginn des spezifizierten Feldes nur soviele Graphikdaten in die Variable A@(*) eingelesen, bis sie voll ist.
   Befinden sich in diesem Bereich Textzeichen, dann werden sie ignoriert.
2. Wurden die Variablen A@(*), B@(*) und C@(*) im Farbmodus als eindimensionale Stringarrayvariablen deklariert, dann werden die graphischen Daten des spezifizierten Bereichs den Variablen jeweils einzeln wie folgt zugewiesen:

   Daten der blauen Graphikebene zu A@(*)
   Daten der roten Graphikebene zu B@(*)
   Daten der grünen Graphikebene zu C@(*)

3. Wurden in der GENTER-Anweisung drei eindimensionale Stringarrayvariablen aufgeführt, dann werden, falls sich alle drei Graphikebenen im Eingabemodus (ODISP) befinden, die Graphikdaten diesen Variablen in folgender Reihenfolge zugewiesen: Graphikebene 1 der ersten, Graphikebene 2 der zweiten und Graphikebene 3 der dritten Variablen.
   Wurden nur zwei Variablen aufgeführt, werden nacheinander die Daten der Graphikebene 1 und der Graphikebene 2 zugewiesen. Wurde nur eine Variable aufgeführt, werden ihr nur die Graphikdaten der Graphikebene 1 zugewiesen.
4. Durch eine entsprechende Wahl des graphischen Eingabemodus (ODISP) kann man bewirken, daß durch GENTER die Graphikdaten der Graphikebenen (GE) 1, 2 und 3 den Stringarrayvariablen wie folgt zugewiesen werden:
   a. GENTER enthält 2 Stringarrayvariablen A@(*) und B@(*):
      GE1 zu A@(*) und GE2 zu B@(*) durch ODISP"12" oder ODISP"123"
      GE2 zu A@(*) und GE3 zu B@(*) durch ODISP"23"
      Die gleichzeitige Zuordnung von G1 zu A@(*) und G3 zu B@(*) ist nicht möglich.
   b. GENTER enthält 1 Stringarrayvariable A@(*):
      GE1 zu A@(*) durch ODISP"1", ODISP"12", ODISP"123", ODISP"13"
      GE2 zu A@(*) durch ODISP"2" oder ODISP"23"
      GE3 kann A@(*) nicht zugewiesen werden.

ANWENDUNGSBEISPIEL:

Man weise die farbigen Graphikdaten der beiden Anzeige-Bereiche (siehe Abbildung 1) zwei zweidimensionalen Stringarrayvariablen A@(*) und B@(*) zu und gebe sie gemeinsam am Bildschirm aus (siehe Abbildung 2).




```
10 DIM A@(3,511)*19 ..... Deklariert 2-dim, Stringarrayvariable (Farbmodus) für Bereich A.
20 DIM B@(3,51)*10 ..... Deklariert 2-dim. Stringarrayvariable (Farbmodus) für Bereich B.
30 GENTER 50,50,200,100,A@(*) ..... Weist die Graphikdaten aus Bereich A der Variablen A@(*) zu.
40 GENTER 400,250,475,300,B@(*) . Weist die Graphikdaten aus Bereich B der Variablen B@(*) zu.
50 CLS ..... Löscht den Bildschirm.
60 GDISP 50,50,200,100,A@(*) ..... Ausgabe der Daten aus Bereich A.
70 GDISP 201,101,276,100,B@(*) ..... Ausgabe der Daten aus Bereich B in einem Bereich, der an Bereich A anschließt.
80 END
```

Siehe auch Anwendungsbeispiel zu der Anweisung GDISP auf Seite 269.

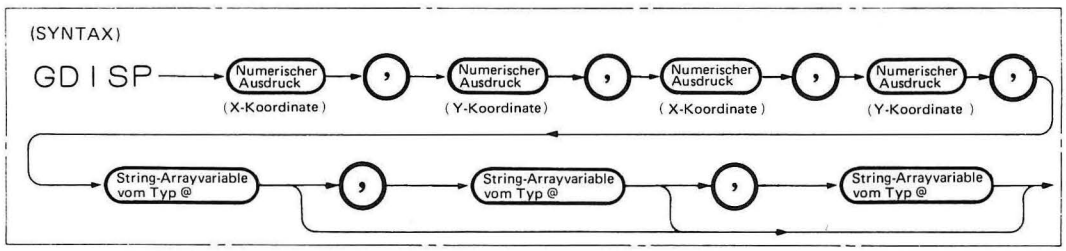
GDISP

Ausgabe gespeicherter graphischer Muster am Bildschirm.

〈VERWENDUNG〉
① Zur Ausgabe von Graphikdaten, die in einer Stringarrayvariablen gespeichert sind.
② Zum Verschieben graphischer Muster auf dem Bildschirm in gemeinsamer Verwendung mit der Anweisung GENTER.

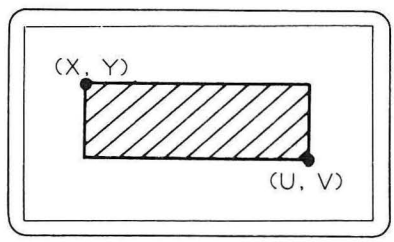
FORMAT: GDISP X, Y, U, V, A@(*) [, B@(*) [, C@(*)]]

X : X-Koordinate der linken oberen Ecke des Ausgabebereichs
Y : Y-Koordinate der linken oberen Ecke des Ausgabebereichs
U : X-Koordinate der rechten unteren Ecke des Ausgabebereichs
V : Y-Koordinate der rechten unteren Ecke des Ausgabebereichs
A@(*) : kollektive Stringarrayvariable des @-Typs
B@(*) : kollektive Stringarrayvariable des @-Typs
C@(*) : kollektive Stringarrayvariable des @-Typs



ERLÄUTERUNG:

1. Die Anweisung GDISP gibt die in einer kollektiven @-Typ Stringarrayvabiablen gespeicherten Graphikdaten in dem durch die Koordinaten (X, Y) und (U, V) bestimmten Bereich des Bildschirms (Ausgabebereich) aus. (Siehe folgende Abbildung.)



(Graphik-Bildschirm)

Die in der Variablen A@(*) gespeicherten Daten entstammen einer Graphikebene und wurden mittels der Anweisung GENTER gelesen (Seite 269).

2. Sind in der GDISP-Anweisung drei Stringarrayvariablen aufführt, werden die Graphikdaten dieser drei Variablen in Überlagerung ausgegeben.

EINFACHES BEISPIEL:

```
30 GDISP 10,10,100,100,A@(*)
```

..... Gibt die in der Variablen A@(*) gespeicherten graphischen Daten, in einem Bildschirmbereich aus, der durch seine linke obere Ecke (10, 10) und seine rechte untere Ecke (100, 100) definiert ist.

WICHTIGER HINWEIS

Ist A@(*) größer als der durch die Koordinatenpunkte (X, Y) und (U, V) spezifizierte Bildschirmbereich, dann werden die Informationen, die im Bereich, bestimmt vom Anfang von A@(*) an, gespeichert werden können, am Schirm angezeigt. (Keine Fehlermeldung) Ist dagegen A@(*) in vertikaler Richtung (Y) kleiner als der spezifizierte Bereich, erfolgt eine Fehlermeldung. Ist er kleiner in horizontaler Richtung (X), erfolgt Auffüllung mit Leerzeichen.

ANWENDUNGSBEISPIEL:

Das folgende Programm soll noch einmal die Wirkungsweise der Anweisungen GENTER und GDISP verdeutlichen.

Es erzeugt jeweils im linken oberen Bereich der Graphikebene 1 ein Dreieck (Zeile 3), in Graphikebene 2 einen Kreis (Zeile 40) und in Graphikebene 3 ein Quadrat (Zeile 50) und gibt diese drei Graphiken gemeinsam (d.h. in Überlagerung aller drei Ebenen) am Bildschirm aus.

Mittels der Anweisung GENTER werden die Graphiken der drei Ebenen in drei Stringarrayvariablen gespeichert, und zwar

das Dreieck aus der Ebene 1 in der Variablen A@(*)

der Kreis aus der Ebene 2 in der Variablen B@ (*)

das Quadrat aus der Ebene 3 in der Variablen C@(*).

Anschließend werden die Inhalte der drei Stringarrayvariablen durch die Anweisung GDISP nacheinander in jeweils diagonal nach unten versetzten Ausgabebereichen angezeigt.

```
 10 DIM A@(101)*13:DIM B@(101)*13:DIM C@(101)*13 ····(1)
 20 SDISP "123":ODISP "123":CLS ····(2)
 30 ODISP "1":LINE 50,0;7,75;93,75,,F
 40 ODISP "2":CIRCLE 50,50,50
 50 ODISP "3":LINE 0,0;0,100;100,100;100,0,,F
 60 ODISP "123" ····(3)
 70 GENTER 0,0,100,100,A@(*),B@(*),C@(*)
 80 GDISP 100,100,200,200,A@(*)
 90 GDISP 200,175,300,275,B@(*)
100 GDISP 300,275,400,375,C@(*)
110 END
```

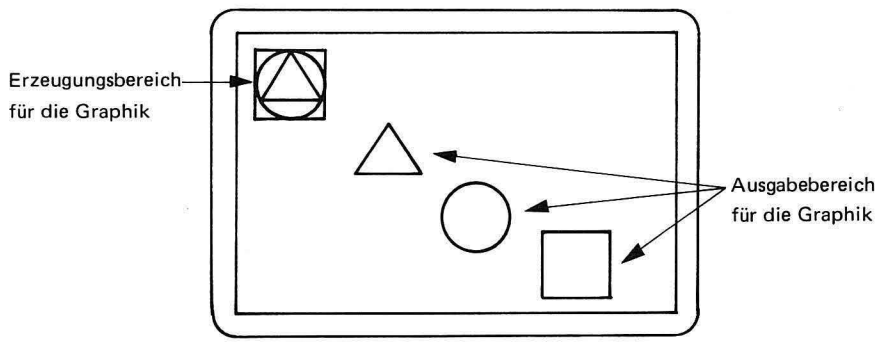
zu (1): Die Dimensionierung der Stringarrayvariablen ergibt sich entsprechend der Formeln von Seite 265:

A@(100–0+1)*m

mit m = (100–0+1):8 = 12.65 = 13 (da m aufzurunden ist)

zu (2): Initialisierung der drei Graphikebenen.

zu (3): Eine Zuweisung von Graphikdaten mittels GENTER kann nur erfolgen, wenn der Eingabemodus der entsprechenden Graphikebene(n) eingeschaltet ist.



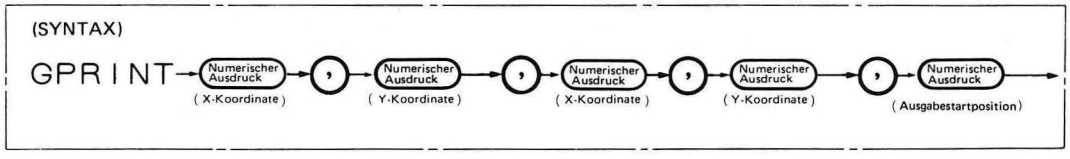
GPRINT

Ausgabe von Graphik über den Drucker.

〈VERWENDUNG〉
Zum Ausgeben eines genau definierten Bereichs der graphischen Bildschirmanzeige (Ausgabebereich) über den Drucker.

FORMAT: GPRINT X, Y, U, V, W

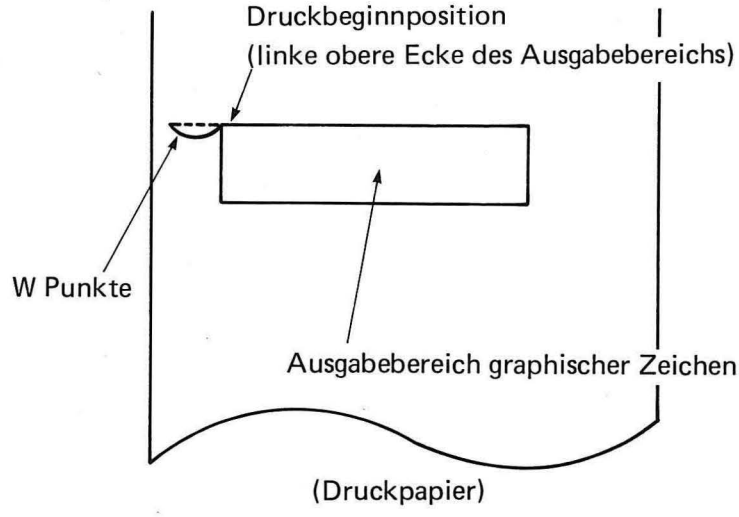
X: X-Koordinate der linken oberen Ecke des Ausgabebereichs
Y: Y-Koordinate der linken oberen Ecke des Ausgabebereichs
U: X-Koordinate der rechten unteren Ecke des Ausgabebereichs
V: Y-Koordinate der rechten unteren Ecke des Ausgabebereichs
W: Druckbeginnposition auf dem Druckpapier



ERLÄUTERUNG:

Die GPRINT-Anweisung gibt den durch die Koordinatenpunkte (X, Y) und (U, V) bestimmten Ausgabebereich ab der Druckbeginnposition W über den Drucker aus (Hardcopy).

Die Positionierung des Ausgabebereichs durch den Drucker auf dem Papier verdeutlicht folgende Abbildung:



EINFACHES BEISPIEL:

```
20 GPRINT 100,100,500,200,10
```

..... Gibt das graphische Muster des durch die linke obere Ecke (100, 100) und die rechte untere Ecke (500, 200) bestimmten Ausgabebereichs vom 10. Punkt (Druckbeginnposition) and über den Drucker aus.

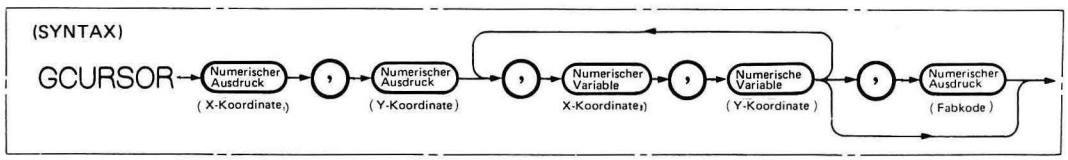
GCURSOR

Anzeigen des Graphikkursors*) und Lesen von Graphik-koordinaten.

〈VERWENDUNG〉
Zum Einlesen graphischer Koordinaten von einer beliebigen Stelle des Graphik-Bildschirms, wohin der Graphikkursor mittels der Kursortaste bewegt wurde.

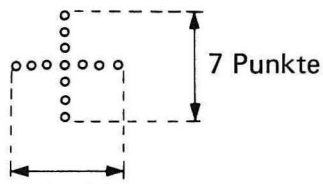
FORMAT: GCURSOR X, Y, {A, B} [, Z]

X: X-Koordinate zur Anzeige des Graphikkursors
Y: Y-Koordinate zur Anzeige des Graphikkursors
A: numerische Variable zur Speicherung der X-Koordinatenposition
B: numerische Variable zur Speicherung der Y-Koordinatenposition
Z: Farbkode für den Graphikkursor
Farbkodetabelle siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.



ERLÄUTERUNG:

1. Die GCURSOR-Anweisung macht den Graphikkursor in einer durch den Parameter Z bestimmten Farbe in der Position (X, Y) auf dem Graphikbildschirm sichtbar. Mit den Kursortasten kann der Graphikkursor jetzt zu einer beliebigen Stelle des Bildschirms bewegt werden. Durch Drücken der [ENTER]-Taste werden die Graphkkkoordinaten der aktuellen Kursorposition gelesen und den Variablen A und B zugewiesen.
2. Das Ausgabeformat des Graphikkursors ist ein Kreuz, dessen waagrechte und senkrechte Linie aus jeweils 7 Punkten besteht. Der Kreuzungspunkt der beiden Linien bestimmt die Kursorposition.



Graphikkursor

3. Die Anzahl der Variablenpaare (A, B) in der GCURSOR-Anweisung bestimmt die Wiederholungszahl des INPUT-Status. Im INPUT-Status bringt man den Graphik-kursor an die jeweils gewünschte Stelle auf dem Bildschirm und drückt anschließend die [ENTER]-Taste für die Zuweisung der Koordinatenwerte.

*) Graphikkursor: Der lursor, der auf dem Graphikbildschirm benutzt wird.

4. Fehlt die Spezifizierung des Farbkodes für den Graphikkursor, dann wird er in der durch die GCOLOR-Anweisung bestimmten Farbe angezeigt.
5. Diese Anweisung ist nur für eine über Bildschirmkanal 1 (CRT-1) ausgegebene Graphikebene gültig.

EINFACHES BEISPIELE:

```
10 GCURSOR 50,50,A,B,3
```
..... Bewegt den Graphikkursor zu den Graphikkoordinaten (50, 50) und zeigt ihn grün an (Farbkode=3). Der Graphikkursor kann jetzt zu der gewünschten Position gebracht werden. Durch Drücken der [ENTER]-Taste werden die Koordinaten dieser Position den Variablen A und B zugeordnet.

```
20 GCURSOR 100,100,A1,B1,A2,B2,1
```
..... Bewegt den Graphikkursor zu den Graphikkordinaten (100, 100) und zeigt ihn gelb an (Farbkode = 1). Er kann jetzt zur gewünschten Position auf dem Bildschirm gebracht werden. Durch Drücken der [ENTER] Taste werden die Koordinaten dieser Position den Variablen A1 und B1 zugewiesen. Jetzt kann man den Graphikkursor zu einer weiteren Position bewegen und durch abermaliges Drücken der ENTER-Taste die Koordinaten dieser position den Variablen A2 und B2 zuweisen.

ANWENDUNGSBEISPIEL:

Durch beliebige Positionierung des Graphikkursors auf dem Bildschirm werden 3 Punkte eingegeben. Die eingegebenen Punkte werden durch Geraden verbunden und als Dreieck ausgegeben.

```
10 SDISP "1":ODISP "1":CLS ..... Initialisierung der Graphik.
20 GCURSOR 300,200,A1,B1,A2,B2,A3,B3
                                ..... Anzeigen des Graphikkursors in der
                                      Position (300, 200). Dreimalige
                                      Wiederholung des INPUT-Status zum
                                      Eingeben der jeweils gewünschten
                                      Position.
30 LINE A1,B1,A2,B2,A3,B3,,F ..... Verbinden der abgespeicherten Koor-
                                   dinatenpaare durch Strecken im
                                   Verbindungsmodus F.
40 END
```

WICHTIGER HINWEIS

Durch gleichzeitiges Drücken von zwei Kursortasten kann der Graphikkursor auch in diagonaler Richtung bewegt werden. (z.B. bewirkt das gleichzeitige Drücken der beiden Tasten "Dursor nach rechts" und "Kursor nach unten" eine diagonale Kursorbewegung nach rechts unten.)

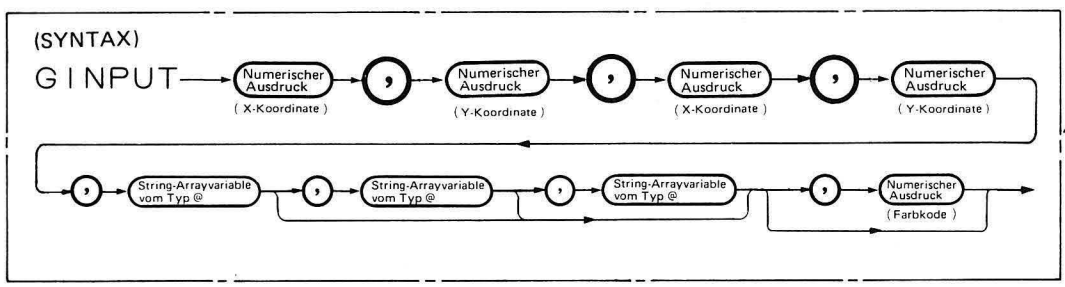
GINPUT

Eingabe von Graphik über die Tastatur und Speicherung in einer Variablen.

〈VERWENDUNG〉
Zum Eingeben eines graphischen Musters über die Tastatur und zum Abspeichern dieses Musters in einer bestimmten Stringarrayvariablen.

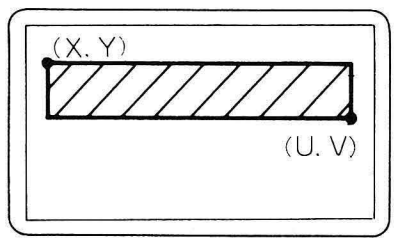
FORMAT: G I NPUT X, Y, U, V, A@(*) [, B@(*) [, C@(*)]] [, Z]

X : X-Koordinate der linken oberen Ecke des Eingabebereichs
Y : Y-Koordinate der linken oberen Ecke des Eingabebereichs
U : X-Koordinate der rechten unteren Ecke des Eingabebereichs
V : Y-Koordinate der rechten unteren Ecke des Eingabebereichs
A@(*): kollektive Stringarrayvariable des @-Typs
B@(*): kollektive Stringarrayvariable des @-Typs
C@(*): kollektive Stringarrayvariable des @-Typs
Z: Farbkode für die einzugebenden, graphischen Muster (0 – 7)
Farbkodetaballe siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.



ERLÄUTERUNG:

1. Die Anweisung GINPUT ermöglicht die Eingabe graphischer Muster über die Tastatur in eine Graphikebene, die sich im Eingabemodus befindet (ODISP). Der graphische Eingabebereich wird durch seine linke obere Ecke (X, Y) und seine rechte untere Ecke vordefiniert. Ist die Eingabe abgeschlossen, wird der Inhalt des Eingabebereichs in eine kollektive Stringarrayvariable eingelesen. Die Stringarrayvariable ist mittels einer DIM-Anweisung analog dem zu der Anweisung GENTER auf Seite 269 beschriebenen Verfahren zu deklarieren.



(Graphikebene)

2. Wenn die GINPUT-Anweisung zur Ausführung kommt, wird der Graphikkursor in der linken oberen Ecke des Eingabebereichs (X, Y) angezeigt und die Möglichkeit zu Graphikeingabe eröffnet.
   Hierfür sind folgende Tasten zu benutzen:
   [S] Taste : Graphikeingabe einschalten
   [R] Taste : Löschmodus einschalten
   [M] Taste : Bewegung des Graphikkursors ohne Graphikeingabe
   Kursortasten : Bewegung des Kursors in waagrechter, senkrechter und diagonaler Richtung (durch gleichzeitiges Drücken von zwei Kursortasten)
   [ENTER] -Taste : Abschließen der Graphikeingabe und Speicher aller Graphikdaten des Eingabebereichs in der kollektiven Stringarrayvariablen
   Tasten [0] – [7] : Drücken einer Zifferntaste (0 – 7) bestimmt die Farbe der anschließend eingegebenen Graphik. (Farbkodetabelle siehe Anweisung COLOR (Seite 239). Eine unmittelbare Anzeige graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.
   (Näheres zur Graphikeingabe siehe "Praktischer Tip.)
3. Fehlt in der GINPUT-Anweisung der Parameter Z (Angabe des Farbkodes), dann bestimmt die letzte GCOLOR-Anweisung die Farbe der eingegebenen Graphik.

EINFACHES BEISPIELE:

1. `20 GINPUT 10,10,50,50,A@(*),5`
   ..... Ermöglicht die Eingabe graphischer Daten in eine Graphikebene in einem Bereich von links oben (10, 10) bis rechts unten (50, 50) in Rot. Die Farbe der eingegebenen Graphikzeichen kann während der Eingabe geändert werden. Nach Beenden der Eingabe wird das graphische Muster des Eingabebereichs der Variablen A@(*) zugewiesen.
2. `20 GINPUT 100,100,110,105,B@(*)`
   ..... Ermöglicht die Eingabe graphischer Daten in eine Graphikebene in einem Bereich von links oben (100, 100) bis rechts unten (110, 105). Die Farbe der eingegebenen Graphikzeichen wird durch die letzte GCOLOR-Anweisung bestimmt. Nach Beenden der Eingabe wird das graphische Muster des Eingabebereichs der Variablen B@(*) zugewiesen.

WICHTIGER HINWEIS

1. Die GINPUT-Anweisung bewirkt, daß alle Graphikdaten des spezifizierten Eingabebereichs einer kollektiven Stringarrayvariablen zugewiesen werden, auch solche, die schon vor der Eingabe über die Tastatur exisierten.
2. Der Löschmodus mittels der R-Taste funktioniert bei einfarbiger Arbeitsweise (z.B. mit einem grünen Monitor) nur dann, wenn über die Anweisung CHANGE DISP (Seite 237) oder SDISP (Seite 241) der Farbmodus eingeschaltet worden ist.

<PRAKTISCHER TIP>

Wir wollen das Arbeiten mit der GINPUT-Anweisung an einem kleinen Beispiel verdeutlichen. Die Aufgabe bestehe darin, das graphische Muster in Abbildung 1 so zu verändern, daß das graphische Muster von Abbildung 2 entsteht.



Abbildung 1
(alle markierten Punkte sind rot)

Abbildung 2
(die markierten Punkte der ersten Zeile sind rot; Punkt (3, 2) ist blau, Punkt (7, 2) grün)

Die für die Änderung der Punkte erforderliche Anweisung lautet

```
GINPUT 1,1,10,3,A@(*),5
```

Bei Ausführung dieser Anweisung in einem Programm wird zunächst der Graphikkursor in der linken oberen Ecke (1, 1) des spezifizierten Eingabebereichs angezeigt. Durch Drücken der [S] -Taste schalten wir den Eingabemodus ein und bewegen den Graphikkursor durch Drücken der [→] -Taste um einen Punkt nach rechts. Punkt (1, 1) ist jetzt rot gesetzt.

Jetzt drücken wir die [M] -Taste und bewegen den Graphikkursor (ohne zu zeichnen) mit der entsprechenden Kursortaste um eins nach unten und um eins nach rechts. Wir befinden uns jetzt über dem Punkt (3, 2), der blau zu markieren ist. Daher schalten wir zunächst durch Drücken der [S] -Taste wieder in den Eingabemodus ein und wählen dann durch Drücken der Zifferntaste 6 die Farbe Blau. Jetzt drücken wir einmal die [→] -Taste. Punkt (3, 2) ist jetzt in Blau gesetzt.

Nach Drücken der [M] -Taste bringen wir den Kursor durch dreimaliges Drücken der [→] -Taste in die Position (7, 2). Wir drücken wieder die [S] -Taste und dann die Zifferentaste 3 (Farbkode für Grün). Durch Drücken der [→] -Taste setzen wir den grünen Punkt.

Jetzt sind alle farbigen Punkte entsprechend Abbildung 2 gezeichnet. Bleibt noch das Löschen von Zeile 3.

Wir drücken zunächst die [M] -Taste und bewegen den Graphikkursor zweimal nach rechts und einmal nach unten. Er befindet sich jetzt in der Position (10, 3).

Durch Drücken der [R] -Taste schalten wir den Löschmodus ein und bewegen den Kursor neunmal nach links (Position (1, 3)) und dann wieder einmal nach rechts. Damit wurde die ganze untere Zeile gelöscht.

Jetzt bleibt uns nur noch das Drücken der [ENTER] -Taste, um das veränderte Graphikmuster der kollektiven Stringarrayvariablen A@(*) zuzuweisen.

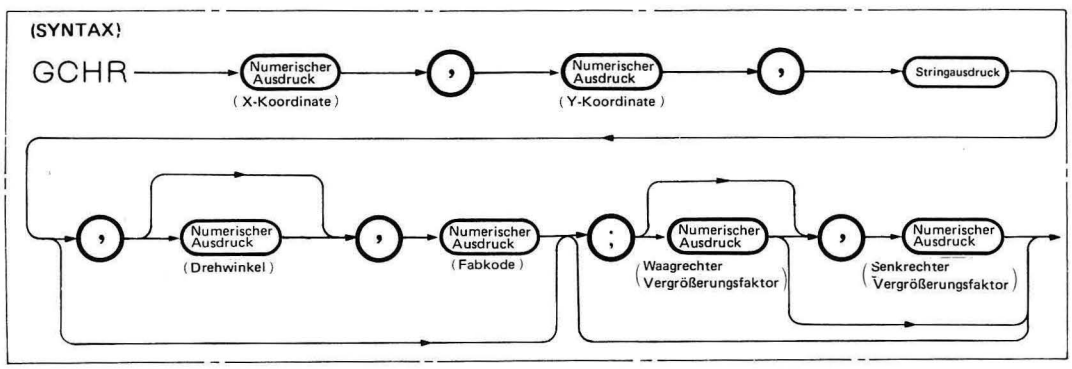
GCHR

Zur Ausgabe von ASCII-Zeichen über den Graphikbildschirm.

〈VERWENDUNG〉
① Zur Ausgabe eines Strings über den Graphikbildschirm.
② Zum Vergrößern und Drehen (Rotieren) der Zeichen in der Anzeige.

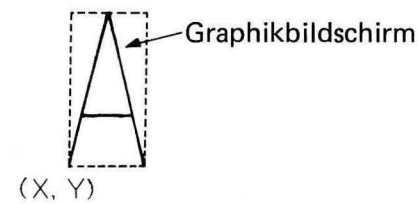
FORMAT: GCHR X, Y, C [, W [, Z]] [W], Z] [; U [, V] | [U], V]

X : X-Koordinate der linken unteren Ecke des Strings
Y : Y-Koordinate der linken unteren Ecke des Strings
C : der auszugebende String
W : Drehungswinkel
0 ..... 0 Grad (keine Drehung)
1 ..... 90 Grad Drehung
2 ..... 180 Grad Drehung
3 ..... 270 Grad Drehung
Z : Farbkode für den auszugebenden String (0 – 7)
Farbkodetabelle siehe Anweisung COLOR (Seite 239).
Die unmittelbare Ausgabe graphischer Muster in Schwarz auf dem Hintergrund ist nicht möglich.
U : Vergrößerungsfaktor für die vertikale Richtung
V : Vergrößerungsfaktor für die horizontale Richtung



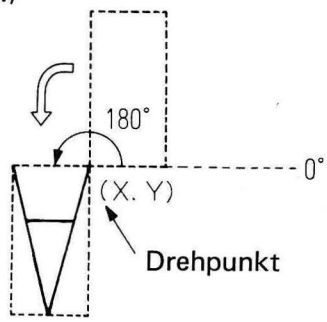
ERLÄUTERUNG:

1. Die GCHR-Anweisung gibt auf dem Graphikbildschirm einen String in der Farbe Z aus, dessen linke untere Ecke die Koordinaten (X, Y) hat (siehe Abbildung). Parameter W gibt den Winkel an, um den der String um den Punkt (X, Y) zu drehen ist (Drehwinkel). Parameter U bezeichnet die Vergrößerung des Strings C in vertikaler, V in horizontaler Richtung.



2. Wird der String gedreht, dann gegen den Uhrzeigersinn um seinen linken unteren Punkt (X, Y). (Siehe Abbildung.)

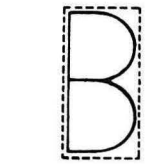
Annahme:
W = 2



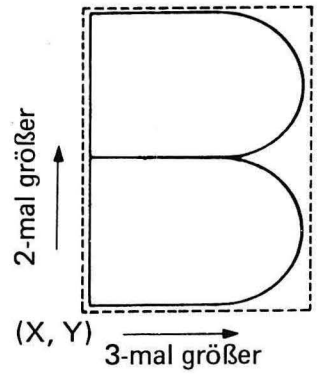
Nach der Drehung

3. Die Vergrößerung eines Strings erfolgt ausgehend von dem Punkt (X, Y) vertikal nach oben und horizontal nach rechts. (Siehe folgende Abbildung.)

Annahme:
U = 2
V = 3



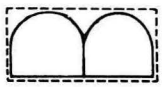
Vor der Vergrößerung



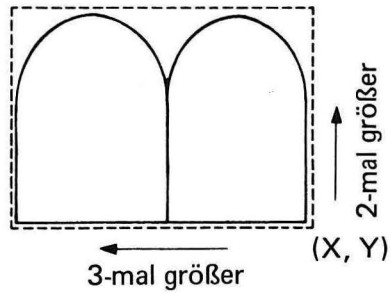
Nach der Vergrößerung

Eine Vergrößerung bei gleichzeitiger Drehung macht die folgende Darstellung deutlich:

Annahme:
U = 3
V = 2



Vor der Vergrößerung



Nach der Vergrößerung

Punkt (X, Y) beschreibt den linken unteren Punkt des auszugebenden Strings vor der Drehung.

4. Fehlt der Parameter W, wird ein Drehwinkel von 0 Grad (keine Drehung) angenommen.
5. Fehlt die Angabe eines Farbkodes, wird die Farbe entsprechend der letzten GCOLOR-Anweisung (Seite 245) bestimmt.
6. Fehlen die Parameter U und V, findet keine Vergrößerung statt.

EINFACHES BEISPIELE:

1. `10 GCHR 100,100,"A",2,5`

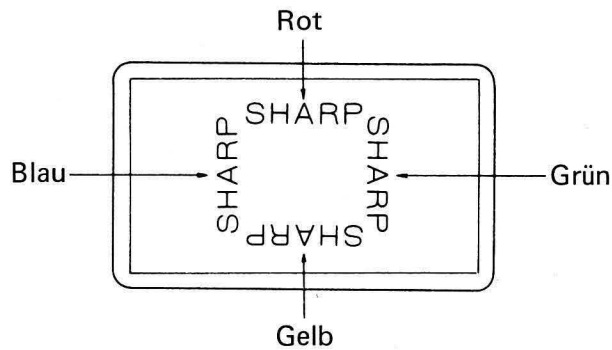
   ..... Gibt den Buchstaben A in der Position (100, 100) um 180 Grad gedreht in Rot (Farbkode 5) aus.
   (Entspricht der Darstellung auf Seite 282.)

2. `10 GCHR 100,100,"B",0,1;2,3`

   ..... Vergrößert den Buchstaben B in vertikaler Richtung um den Faktor 2 und in horizontaler Richtung um den Faktor 3 und gibt ihn mit der linken unteren Ecke in der Position (100, 100) in Gelb aus.
   (Entspricht der Darstellung auf Seite 282.)

ANWENDUNGSBEISPIEL:

Man gebe den String "SHARP" entsprechend der folgenden Darstellung auf dem graphischen Bildschirm aus:



Graphikbildschirm

```
10 SDISP "123C":ODISP "123":CLS
20 SCALE OFF
30 GCHR 199,79,"SHARP",0,5;2,6
40 GCHR 199,319,"SHARP",1,6;6,2
50 GCHR 439,319,"SHARP",2,1;2,6
60 GCHR 439,79,"SHARP",3,3;6,2
70 END
```

10 SDISP "123C":ODISP "123":CLS ····· Initialisierung der Graphik.

30 GCHR 199,79,"SHARP",0,5;2,6 ····· Gibt den String "SHARP" ohne Drehung in Rot bei zweimaliger Vergrößerung in vertikaler Richtung und sechsmaliger Vergrößerung in horizontaler Richtung am Graphikbildschirm aus. Die linke untere Ecke des Strings hat die Koordinatenposition (199, 79).

40 GCHR 199,319,"SHARP",1,6;6,2 ····· Gibt den String "SHARP" mit einem Drehwinkel von 90 Grad in Blau bei sechsmaliger Vergrößerung in vertikaler Richtung und zweimaliger Vergrößerung in horizontaler Richtung am Graphikbildschirm aus. Die linke untere Ecke des Strings hat die Koordinatenposition (199, 319).

50 GCHR 439,319,"SHARP",2,1;2,6 ····· Gibt den String "SHARP" mit einem Drehwinkel von 180 Grad in Gelb bei zweimaliger Vergrößerung in vertikaler Richtung und sechsmaliger Vergrößerung in horizontaler Richtung am Graphikbildschirm aus. Die linke untere Ecke des Strings hat die Koordinatenposition (439, 319).

60 GCHR 439,79,"SHARP",3,3;6,2 ····· Gibt den String "SHARP" mit einem Drehwinkel von 270 Grad in Grün bei sechsmaliger Vergrößerung in vertikaler Richtung und zweimaliger Vergrößerung in horizontaler Richtung am Graphikbildschirm aus. Die linke untere Ecke des Strings hat die Koordinatenposition (439, 79).

70 END

WICHTIGER HINWEIS

Diese Anweisung kann ASCII-Zeichen von Kode &20 bis &7A ausgeben. (Siehe ASCII-Kode-Tabelle im Bedienerhandbuch – Anhang 1). Eine Ausgabe deutscher Sonderzeichen (Umlaute, ß) mittels der Anweisung GCH ist nicht möglich.

7. VERBINDUNG VON BASIC UND MASCHINENSPRACHE

Dieser Abschnitt erklärt Anweisungen, die sich auf die Verwendung von Maschinensprache in BASIC-Programmen beziehen. Genaue Kenntnis der Maschinensprache ist bei Verwendung dieser Anweisungen dringend erforderlich.

NEW ※
ERASE ※
POKE(PEEK) ※
USR

Die mit ※ bezeichneten Anweisungen können auch in der manuellen Betriebsart benutzt werden.

NEW

Löscht alle Programmzeilen und Variablen und reserviert einen Arbeitsbereich für Maschinenprogramme.

〈VERWENDUNG〉
Reserviert den Anwendermikrobereich*) zur Verwendung von Maschinenprogrammen.

FORMAT: NEW X

X: Höchste Adresse des Anwendermikrobereichs (&9600 <= x <= &BFFF)

(SYNTAX)



ERLÄUTERUNG:

1. Diese Anweisung löscht alle Programmzeilen und Variablen und reserviert einen durch die Adressen &9600 und X begrenzten Speicherbereich als Anwendermikrobereich. BASIC-Programme und Daten können nicht in diesen Bereich geschrieben werden. X muß im Bereich zwischen &9600 und &BFFF liegen.
2. Soll der Anwendermikrobereich wieder als gewöhnlicher Programm- und Datenspeicher Verwendung finden, muß die Anweisung mit dem Argument 0 ausgeführt werden.

EINFACHES BEISPIELE:

1. NEW &ACFF [ENTER] ..... Legt den Adressbereich von &9600 bis &ACFF als Anwendermikrobereich fest. Gleichzeitig werden alle Programmzeilen und Variablen gelöscht.



2. NEW 0 [ENTER] ..... Gibt den Speicherbereich über der Adresse &9600 zur Verwendung als Programm- und Datenspeicher frei. Alle Programmzeilen und Variablen werden gelöscht.

WICHTIGER HINWEIS

1. Wird diese Anweisung ausgeführt, werden alle Programmzeilen gelöscht. Zur programmgesteuerten Festlegung eines Anwendermikrobereichs muß die Anweisung ERASE (siehe nächste Seite) verwendet werden.
2. Die Anfangsadresse des Anwendermikrobereichs kann sich durch Systemänderung oder durch Erweiterung des Graphikspeichers verschieben.
   Diese Adresse kann mit Hilfe des FDOS-Befehls DEBUG geprüft werden.

*) Anwendermikrobereich: Speicherbereich für Maschinenprogramme

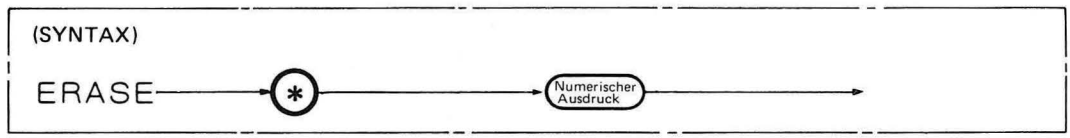
ERASE

Löscht alle Variablen und reserviert einen Speicherbereich für Maschinenprogramme.

〈VERWENDUNG〉
Programmgesteuerte Festlegung eines Anwendermikrobereichs vor Verwendung eines Maschinenprogramms.

FORMAT: ERASE * X

X: Höchste Adresse des Anwendermikrobereichs (&9600 <= X <= &BFFF)



ERLÄUTERUNG:

1. ERASE löscht alle Variablen und reserviert den Adressbereich von &9600 bis X als Anwendermikrobereich.
   BASIC-Programme und Daten können nicht mehr in diesen Bereich geschrieben werden. Programme werden durch diese Anweisung nicht gelöscht.
2. Zur Freigabe des Anwendermikrobereichs als Programm- und Datenspeicher muß ERASE mit dem Argument 0 ausgeführt werden.

EINFACHES BEISPIELE:

1. `10 ERASE *&ACFF` ..... Reserviert den Speicherbereich zwischen &9600 und &ACFF als Anwendermikrobereich und löscht alle Variablen.
2. `20 ERASE *0` ..... Gibt den Speicher nach der Adresse &9600 wieder zur Programm- und Datenspeicherung frei und löscht alle Variablen.

WICHTIGER HINWEIS

1. Diese Anweisung kann nicht in einem Unterprogramm verwendet werden (siehe auch Seite 157).
2. Die Anfangsadresse des Anwendermikrobereichs kann sich durch Systemänderungen oder durch Erweiterung des Graphikspeichers verschieben. Sie kann unter Verwendung des FDOS-Befehls DEBUG geprüft werden (siehe auch Seite 55).

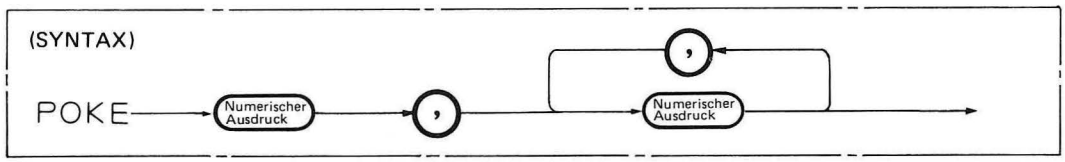
POKE (PEEK)

Speichert Daten an einer gewünschten Speicheradresse.

FORMAT: POKE X, {Y}

X: Speicheradresse (zwischen &9600 und &CFFF)

Y: Zu speichernder Wert, zwischen &00 (0) und &FF (255)



ERLÄUTERUNG:

1. POKE speichert den Wert der numerischen Variablen Y an der durch X festgelegten Adresse. Über die Funktion der verwendbaren Speicheradressen gibt der Speicherplan im Anhang dieses BASIC-Handbuchs Auskunft. Da die POKE-Anweisung nur im Adressbereich zwischen &9600 und &CFFF ausführbar ist, darf X nur Adressen innerhalb dieses Bereichs enthalten. Der Wert des numerischen Ausdrucks Y muß zwischen &00 (0) und &FF (255) liegen.
2. Spezifiziert man mehrere Ausdrücke Y, werden deren Werte sequentiell in den Addressen X, X+1, X+2, . . (Adreßinkrement 1) gespeichert. Überschreitet X den Wert &CFFF, werden weitere Daten ignoriert.
3. Das Gegenteil zu POKE stellt die PEEK-Anweisung dar. Sie liefert den Inhalt einer bestimmten Speicherstelle.

EINFACHES BEISPIEL:

```
10 POKE &A000,&C1
```
..... Schreibt den Wert &C1 (193) in die Speicherstelle &A000 ein.

WICHTIGER HINWEIS

1. Die höchste Adresse, auf die über die POKE-Anweisung zugegriffen werden kann, wird durch die Anweisungen NEW (Seite 287) und ERASE (Seite 288) festgelegt (&9600 <= X <= &CFFF).
2. Der Adreßbereich zwischen &C000 und &CFFF stellt den Bildwiederholspeicher dar. Werden Daten in diesen Bereich "gepoked", erscheint das dem Wert entsprechende ASCII-Zeichen auf dem Bildschirm.

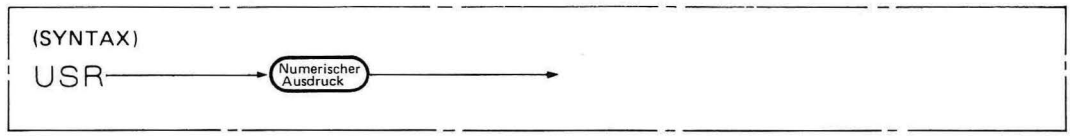
USR

Ruft ein Maschinenunterprogramm auf.

● **FUNKTION 1:** Der Aufruf der Maschinenroutine erfolgt ohne Argumente.

FORMAT: USR X

X: Startadresse des Maschinenprogramms



ERLÄUTERUNG:

Übergibt die Steuerung an ein mit der Adresse X beginnendes Maschinenunterprogramm.

EINFACHES BEISPIEL:

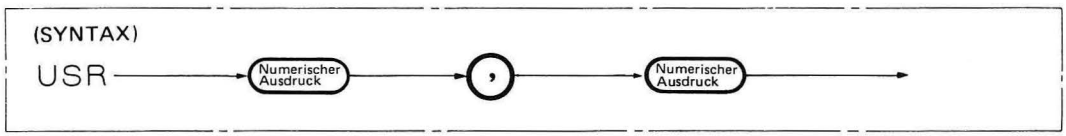
```
10 USR &9800
```
..... Übergibt die Kontrolle an ein mit der Adresse &9800 beginnendes Maschinenunterprogramm.

● **FUNKTION 2:** Der Aufruf an die Maschinenroutine enthält ein Argument.

FORMAT: USR X, A

X: Startadresse des Maschinenunterprogramms

A: Argument, wird dem Maschinenunterprogramm übergeben



ERLÄUTERUNG:

Übergibt die Programmkontrolle an ein Maschinenunterprogramm mit der Startadresse X. Das Argument A wird in das Binärformat übersetzt und im HL-Register gespeichert. Das höchste Bit des HL-Registers bestimmt dabei das Vorzeichen, wobei 0 für Plus und 1 für Minus steht.

1. Die Rückkehr vom Maschinen- zum BASIC-Programm wird durch den Befehl RET (&C9) auf Maschinenebenen erreicht.
2. Soll ein Argument (zurück-) übergeben werden, muß dessen Wert im HL-Register gespeichert und das Carry-Flag zurückgesetzt sein.

```
AND A
RET
```

3. Enthält z.B. das HL-Register den binären Wert 0020H, beträgt der Wert des Arguments 32. Soll der Wert des Arguments durch das Maschinenprogramm nicht verändert werden, muß bei der Rückkehr das Carry-Flag gesetzt sein.

```
SCF
RET
```

EINFACHES BEISPIELE:

USR &A000,B ..... Übergibt die Programmkontrolle an ein Maschinenprogramm mit der Startadresse &A000 und dem Argument B. Enthält das Argument B z.B. den Wert 90, wird das HL-Register mit 005AH geladen und das Maschinenprogramm an der Adresse A000H gestartet. Enthält B den Wert –90 wird im HL-Register die Zahl 805AH übergeben.

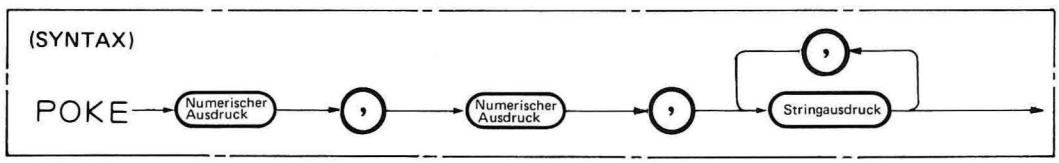
● **FUNKTION 3:** Dem Maschinenprogramm sollen mehrere Argumente übergeben werden.

FORMAT: USR X, Y, {M}

X : Startadresse des Maschinenprogramms

Y : Erste Adresse des Speicherbereichs, der die Argumente enthält

M : Weitere Argumente

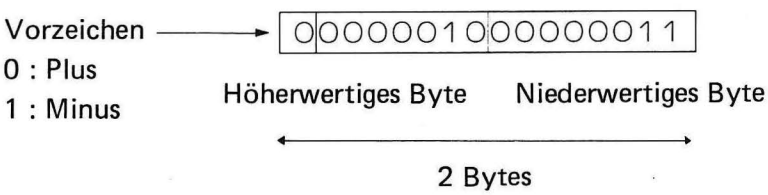


ERLÄUTERUNG:

Die Argumente werden beginnend mit der Adresse & gespeichert. Danach wird die Programmkontrolle an das Machinenprogramm mit der Startadresse X übergeben. Argumente werden in folgenden Formaten gespeichert:

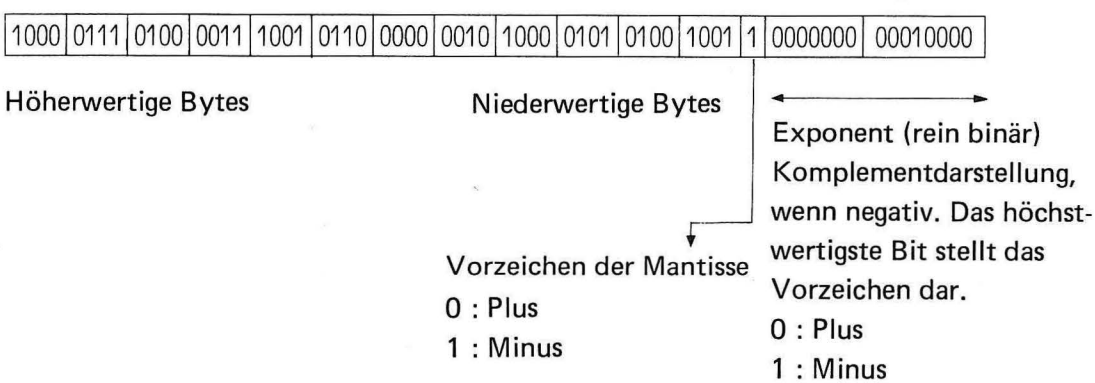
1. Variablen des Typs INTEGER

BEISPIEL: A% = 515



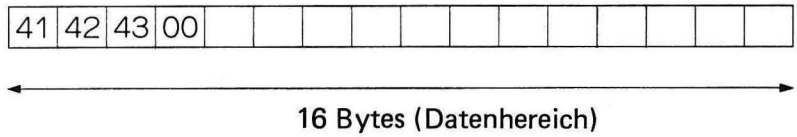
2. Variablen des Typs REAL

BEISPIEL: A# = –8.74396028549 x 10E16



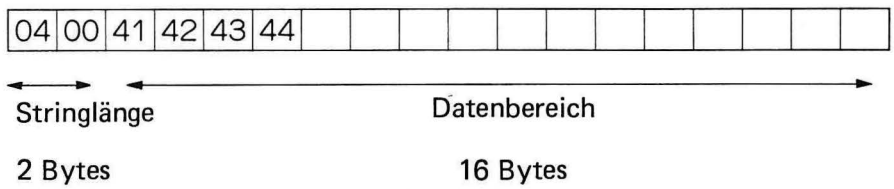
3. Stringvariablen des Typs $

BEISPIEL: A$ = "ABC"



4. Stringvariablen des Typs @

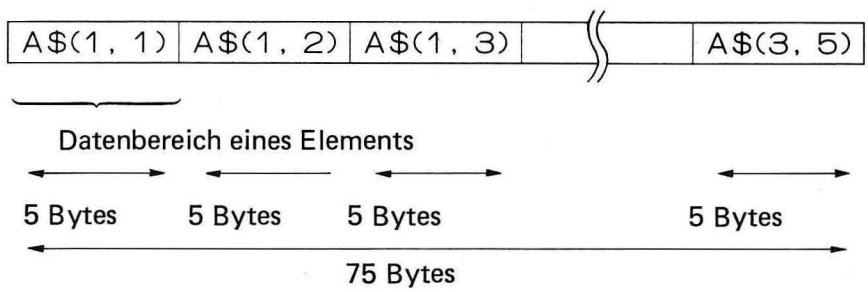
BEISPIEL: A@ = "ABCD"



Wurde die Variable vordefiniert, ist die Länge durch DIM festgelegt.

5. Array-Variablen

BEISPIEL: Ein Variable des Typs A$(*), definiert durch DIM A$(3, 5) * 5



Bei Variablen des @-Typs ist die Zeichenzahl in den ersten beiden Bytes vor jedem Element enthalten.

8. STEUERUNG DER RS-232C SCHNITTSTELLE

Dieser Abschnitt enthält Anweisungen zur Steuerung der RS-232C-Schnittstelle.

CHANNEL
SEND
RCV
SENREV
OPCHNL
ROLLING
RLIST
TERM

Diese Anweisungen können auch in der manuellen Betriebsart verwendet werden. Die RS-232C-Schnittstelle selbst wird im Abschnitt — RS-232C-Schnittstelle — des Bedienerhandbuchs beschrieben.

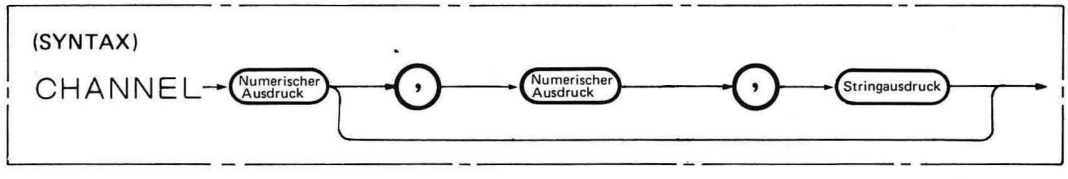
CHANNEL

Bestimmt den zu verwenden Kanal und das Ein- und Ausgabeformat.

FORMAT: CHANNEL X [, Y, C]

X : Kanalnummer (0, 1 oder 2)
Y : Übertragungsrate (110, 200, 300, 600, 1200, 2400, 4800, 9600 Baud)
C : Übertragungsformat
Das Übertragungsformat wird wie folgt festgelegt:
" Wortlänge, Paritätsformat, Stopbitlänge"

1. Wortlänge
   7 ..... Wortlänge 7 Bit
   8 ..... Wortlänge 8 Bit
2. Paritätsformat
   E ..... gerade Parität
   O ..... ungerade Parität
   N ..... keine Parität
3. Stopbitlänge
   1 ..... 1 Bit
   1.5 ..... 1,5 Bit
   2 ..... 2 Bit



ERLÄUTERUNG:

1. CHANNEL bestimmt den zu verwendenden Kanal und Das Übertragungsformat der Ein- und Ausgabe. Dieses Format enthält die Übertragungsrate (Ausdruck Y) und das Datenformat (Stingkonstante C). Nach Ausführung dieses Befehls werden alle SEND-, RCV- und SENREV-Anweisungen auf diese Werte bezogen. Das Übertragungsformat wird in Form einer Stringkonstanten angegeben. (Beispiel: "7E1.5")
2. Werden Y und C weggelassen, bleiben Übertragungsrate und Datenformat der letzten CHANNEL-Anweisung erhalten. Wurde noch keine CHANNEL-Anweisung ausgeführt oder wurde der NEW-Befehl gegeben, ist die Übertragungsrate auf 300 Baud und das Datenformat auf 7-Bit Wortlänge, gerade Parität und 1 Stopbit festgelegt.
3. Nach einer Systeminitialisierung (z.B. Einschalten der Stromversorgung) ist Kanalnummer 0 gewählt.
   (Die eingebaute Schnittstelle ist für Kanal 0 vorgesehen.)

EINFACHES BEISPIELE:

1. `10 CHANNEL 0,600,"8N1"` ..... Bestimmt die Verwendung von Kanal 0, eine Übertragungsrate von 600 Baud, eine Wortlänge von 8 Bit, keine Parität und eine Stopbitlänge von 1.
2. `10 CHANNEL 0;` ..... Bestimmt die Verwendung von Kanal 0 und behält das Übertragungsformat der letzten Anweisung bei.

WICHTIGER HINWEIS

1. Werden die Argumente Y und C bei Verwendung einer Wortlänge von 7 Bit angegeben, ist der "shift in/shift out" (SI/SO)-Status auf SI festgelegt. Werden Y und C weggelassen, wird der SI/SO-Status nicht verändert.
2. Die eingebaute Schnittstelle ist für Kanal 0 vorgesehen. Die Kanäle 1 und 2 können nach Einbau der seriellen Schnittstelle MZ-1E01 in den Erweiterungsrahmen verwendet werden. Genauere Angaben über Kanalnummern und Einbau sind dem Handbuch der Schnittstelle zu entnehmen.

SHIFT IN / SHIFT OUT

Mit Hilfe des SI/SO-Status kann ein 7-Bit-Kode genauso wie 8-Bit-Kode verwendet werden.

Der SO-Kode (&0E) schaltet vom SI- in den SO-Status, der SI-Kode (&0F) vom SO- in den SI-Status. Diese Kodes sind sowohl im SI- als auch im SO-Zeichensatz enthalten. SI- und SO-Zeichen sind deshalb in jedem ASCII-7-Bit-Datensatz enthalten. In Programmen mit begrenztem Speicherplatz für Daten ist dieser Umstand zu beachter

SI- und SO-Status werden bei jedem Kanal für Ein- oder Ausgabe getrennt verarbeitet. Beim Einschalten des Geräts (Initialisierung) wird automatisch der SI-Status gewählt, ebenfalls bei Ausführung einer CHANNEL- oder OPCHNL-Anweisung.

Bei einer OPCHNL-Anweisung wird jedoch nur der Eingabestatus festgelegt.

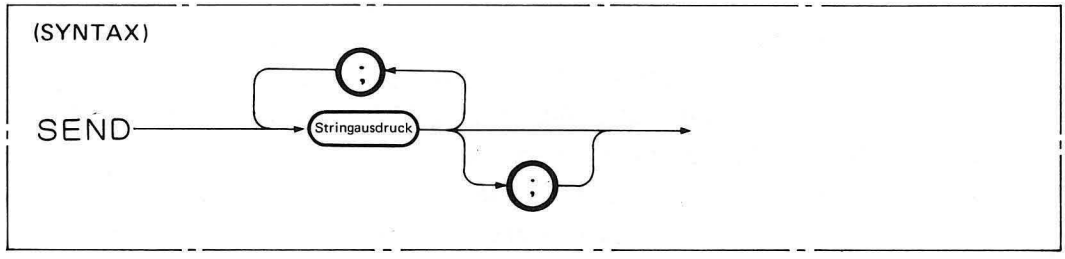
SEND

Gibt Daten mit dem durch CHANNEL festgelegten Format aus.

● **FUNKTION 1:** Ausgabe von Stringdaten.

FORMAT: SEND 《C》〔;〕

C: String, dessen Inhalt übertragen werden soll



ERLÄUTERUNG:

1. Wird das Semikolon (;) nach der letzten spezifizierten Stringvariablen weggelassen, werden die in C gespeicherten Daten ausgegeben, danach wird automatisch ein [CR] (Kode &0D) und ein [LF] (Kode &0A) gesendet.
2. Wird nach der letzten spezifizierten Stringvariablen ein Semikolon (;) gesetzt, werden nur die in C gespeicherten Daten gesendet; [CR] und [LF] werden nicht angefügt.
3. Arrays können durch Angabe ihres Namesns (z.B. A$(*)) übertragen werden. Jedes Element wird dabei mit der in DIM definierten Stringlänge gesendet; kürzere Elemente werden durch Anfügen von [NULL] -Zeichen (&00) angeglichen.
4. Das Ausgabeformat bei der Übertragung mehrerer Stringvariablen C ist in FUNKTION 1 der Anweisung PRINT (Seite 110) beschrieben.

EINFACHES BEISPIELE:

1. `10 SEND "ABC"` ..... Gibt die Zeichenkonstante "ABC", gefolgt von [CR] und [LF], über die Schnittstelle aus.
2. `10 SEND "ABC";` ..... Es wird nur die Zeichenkonstante "ABC" übertragen.
3. `10 SEND A$(*)` ..... Das Feldes A$(*) enthält folgende Elemente:

AA	BBBB	CCC
D	EEE	FF
GGG	H	I I
JJJJ	KK	LLL
M	NNNN	

(Das Array wurde in der Form DIM A$(3,5)*4)definiert.)

Eine Datenausgabe im 8-Bit-Kode hat dann folgendes Format:

[A][A][NL][NL][B][B][B][B][C][C][C][NL][D][NL][NL][NL][E][E][E][NL]
[F][F][NL][NL][G][G][G][NL][H][NL][NL][NL][I][I][NL][NL][J][J][J][J]
[K][K][NL][NL][L][L][L][NL][M][NL][NL][NL][N][N][N][N][NL][NL][NL][NL]
[CR][LF]

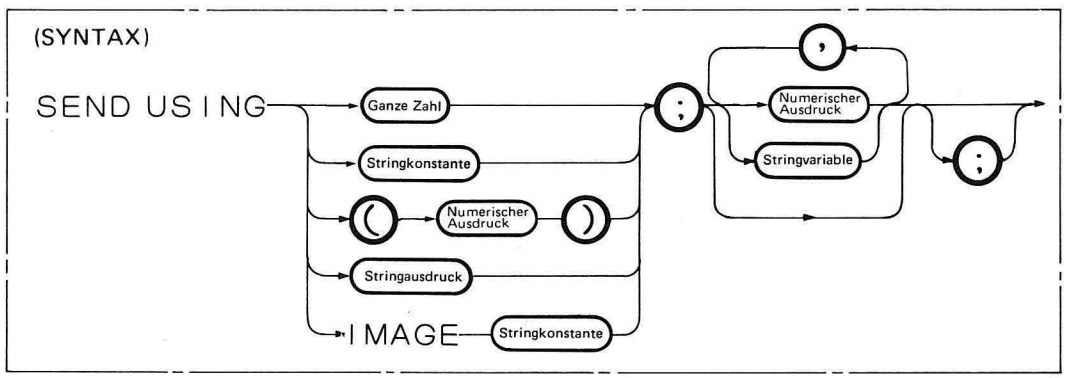
4.10 SEND "ABC";"D";"EF"; .....Gibt die Daten in folgendem Format aus:

[A][B][C][D][E][F]

● **FUNKTION 2:** Gibt numerische Daten und Strings formatiert aus.

FORMAT: SEND USING n | S | (X) | A$ | IMAGE T;{{Y | C}}{;}

n : Zeilennummer (Die Zeile enthält die Formatanweisung)
S : Zeilenmarke (Die Zeile enthält die Formatanweisung)
X : Zeilennummer (numerischenr Ausdruck, sonst wie oben)
A$: Stringvariable, enthält die Formatanweisung
T : Format für die IMAGE-Anweisung
Y : Auszugebender numerischer Ausdruck
C : Auszugebender String



ERLÄUTERUNG:

1. Die durch die numerische Variable Y und den String C festgelegten Daten werden mit dem durch IMAGE definierten Format ausgegeben.
2. Die Formatierung entspricht FUNKTION 2 der PRINT-Anweisung (Seite 110) bzw. der IMAGE-Anweisung (Seite 113).
3. Wird das Semikolon nach dem letzten zu übertragenden Ausdruck weggelassen, wird automatisch ein [LF]-Zeichen angefügt.([CR] wird nicht gesendet.)Ein Weglassen von Y und C verhindert jegliche Ausgabe. Wird ein Semikolon angefügt, werden nur Daten gesendet. Sollen dennoch [CR] und [LF] gesendet werden, können sie über die IMAGE-Anweisung spezifiziert werden.

EINFACHES BEISPIELE:

1.
```
10 H$="Transistoren":K=1320
20 SEND USING 100;H$,K
30 END
100 IMAGE '11A 5X "Preis" 4X $#.### C'
```

Die Daten werden nun in folgendem Format ausgegeben:

[T][R][A][N][S][I][S][T][O][R][E][N][SP][SP][SP][P][R][E][I][S][SP][SP][SP][SP]
[$][1][.][3][2][0][CR][LF]

2.
```
10 H$="Stift":K=100
20 SEND USING "$";H$,K;
100+"$":IMAGE '5A 5X "Preis" 4X $#,### C'
```

Die Daten werden nun in folgendem Format ausgegeben:

S t i f t SP SP SP SP SP P r e i s SP SP SP SP SP SP $ 1
0 0 CR

3.
```
10 DIM P$(1)*25:P$(1)=
       '5A 5X "Preis" 4X $#,### L'
20 H$="Stift":K=100
30 SEND USING P$(1);H$,K
40 END
```

Die Daten werden nun in folgendem Format ausgegeben:

S t i f t SP SP SP SP SP P r e i s SP SP SP SP SP SP
$ 1 0 0 LF LF

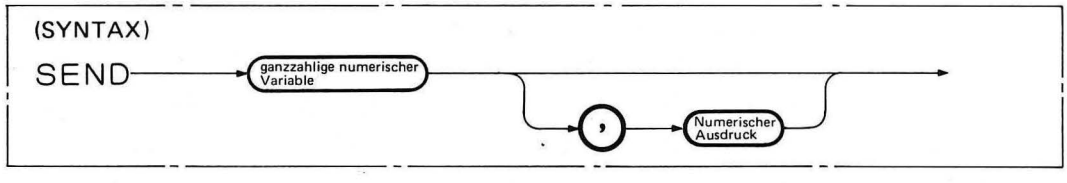
Man beachte die unterschiedlichen Formate der obigen Beispiele. Weitergehende Informationen können den Bescheibungen der Anweisungen PRINT (Seite 110) und IMAGE (Seite 113) sowie der Symboltabelle zur IMAGE-Anweisung (Seite 113) entnommen werden.

● **FUNKTION 3:** Ausgabe numerischer Werte im Binärformat. Die Werte müssen in Form von Integer-Variablen vorliegen.

FORMAT: SEND A% (, X)

A%: Integer-Variable (Der Inhalt wird ausgegeben)

X : Anzahl der auszugebenden Bytes



ERLÄUTERUNG:

1. SEND uberträgt nur das niederwertige Byte der Integervariablen A%. Das höherwertige Byte wird nicht berücksichtigt.
2. Es können auch INTEGER-Arrayvariablen) durch Angabe ihres Namens (z.B. B%(*)) übertragen werden. Auch in diesem Fall werden nur die niederwertigen Bytes der einzelnen Elemente übertragen.
3. Die Anzahl der auszugebenden Elemente einer kollektiven Array-Variablen kann mit der Variablen X festgelegt werden.
4. Bei dieser Übertrangungsart werden CR und LF nicht ausgegeben.

EINFACHES BEISPIEL:

```
10 SEND B%(*),10
```
..... Gibt die niederwertigen Bytes der ersten 10 Elementen des Arrays B%(*) im Binärformat aus.

WICHTIGER HINWEIS

Zur Verwendung von Funktion 3 muß eine Wortlänge von 8 Bit eingestellt werden. Der [NULL]-Kode (&00) kann auch übertragen werden. (Beispiel: SEND 0%)

[NL] steht für NULL-Kode (&00)
[SP] steht für Leerzeichen (&20)
[CR] steht für Wagenrücklauf (Carriage return &0D)
[LF] steht für Zeilenvorschub (Line feed &0A)

RCV

Empfängt Daten entsprechend dem mit CHANNEL festgelegten Format.

● **FUNKTION 1:** Empfang von Stringdaten

FORMAT: RCV [/] A$, X, C [, | ; D]

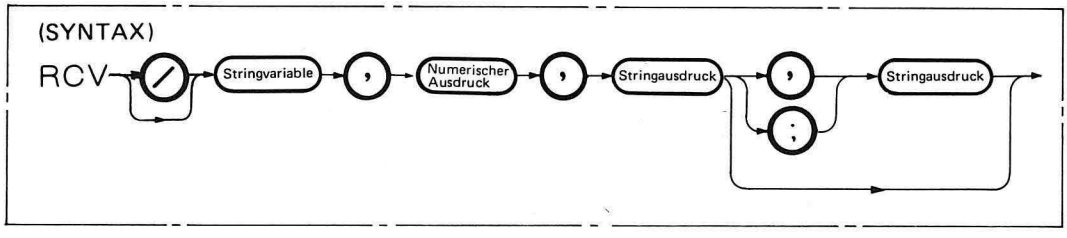
/ : Wahl der Echofunktion

A$: Variable zur Speicherung der empfangenen Daten

X : Wartezeit (0 – 255), Einheit 0.1 Sekunden

C : 1. Kode für Übertragungsende

D : 2. Kode für Übertragungsende



ERLÄUTERUNG:

1. Alle empfangenen Daten, einschließlich der in C oder D festgelegten Endzeichen, werden in der Variablen A$ gespeichert. Wird vor Empfang eines Endzeichens der zulässige Speicherplatz der Variablen A$ überschritten, wird ein Fehler gemeldet (ERROR 109). Mit Stringvariablen des @-Typs kann auch das NULL-Zeichen (&00) empfangen werden. An Stelle von A$ kann auch eine kollektive Arrayvariable treten (A$(*)). Als Endzeichen ist jeweils nur das erste Zeichen der Stringvariablen C und D wirksam.
2. Wählt man die [/] -Option, wird die Echofunktion*) des Empfängers in Betrieb genommen. Bei Verwendung von 7-Bit-Kode muß der SI-SO-Status von Sender und Empfänger beachtet wird.
3. Der numerische Ausdruck X bestimmt die Wartezeit in Einheiten von 0.1 Sekunden. Stellt z.B. X den Wert 150 dar, beträgt die Wartezeit 15 Sekunden. Werden innerhalb dieser Wartezeit keine Daten empfangen, wird Fehler 103 gemeldet. X darf Werte zwischen 0 und 255 annehmen. Für X = 0 wird unbegrenzt gewartet (Wartezeit unendlich).
   In allen Fällen kann der Wartezyklus durch Drücken der HALT-Taste abgebrochen werden.
4. Wird D weggelassen, werden alle Daten bis zum Empfang des durch C definierten Endzeichens empfangen.

*) Echofunktion: Ein empfangenes Zeichen wird durch diese Funktion sofort wieder zum Sender übertragen. Diese Funktion ist bei Geräten mit niedriger Übertragungsrate (weniger als 1200 Bit pro Sekunde) geeignet.

5. Gibt man sowohl C als auch D getrennt durch ein Komma an, werden solange Daten empfangen, bis eines der beiden durch C und D festgelegten Endzeichen empfangen wurde.
   Sind C und D durch ein Semikolon getrennt, werden Daten empfangen, bis beide Endzeichen in der mit C und D bestimmten Reihenfolge empfangen wurden.
   Lautet die Eingabe z.B. "A", "B", werden Daten bis zum Empfang von "A" oder "B" aufgenommen. Wird das Format "A"; "B" verwendet, werden Daten empfangen, bis die Endzeichen in der Reihenfolge "A" "B" aufgenomen worden sind.

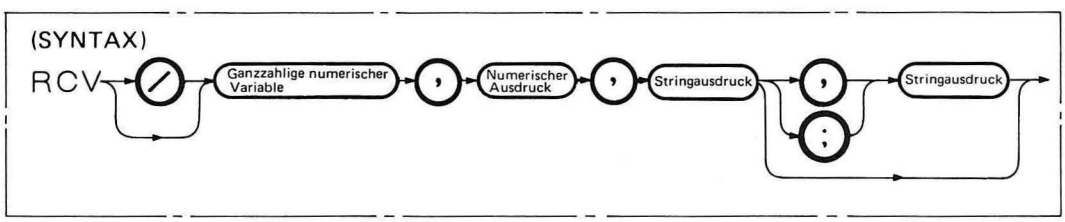
EINFACHES BEISPIELE:

1. `10 RCV A$,0,"#"` ..... Empfängt Daten und speichert sie in der Stringvariablen A$, bis das Endzeichen "#" empfangen wird. Die Wartezeit ist unendlich.
2. `10 RCV A$,150,"#","A"` ..... Empfängt Daten und speichert sie in der Stringvariablen A$, bis das Endzeichen [#] oder [A] empfangen wird. Die Wartezeit beträgt 150 x 0.1 = 15 Sekunden.
3. `10 RCV A$,150,"#";"A"` ..... Empfängt Daten und speichert sie in der Stringvariablen A$, bis die Endzeichen in der Reihenfolge [#] [A] empfangen werden.
4. `10 RCV/A$,150,"#";"A"` ..... Funktion wie im vorigen Beispiel, jedoch mit Echofunktion.

● **FUNKTION 2:** Eingabe von Daten und Zuweisung an eine Integer-Variable.

FORMAT: RCV [/] A%, X, C [, | ; D]

/ : Wählen der Echofunktion
A% : Integer-Variable, der Daten zugewiesen werden sollen
X : Wartezeit (0 – 255), Einheit 0.1 Sekunde
C : 1. Endzeichen
D : 2. Endzeichen



ERLÄUTERUNG:

1. In dieser Form können numerischer Daten mit einem Byte Länge (binäre Daten) empfangen werden. Dieses Byte wird dem niederwertigen Teil der Integervariablen A% (2 Byte Länge) zugewiesen. Das höherwertige Byte wird 0. Es können somit Werte zwischen 0 und 255 empfangen werden. An Stelle der Integervariablen A% können auch kollektive Arrayvariablen der Form C%(*) verwendet werden.
2. Alle anderen Eigenschaften entsprechen Funktion 1.

EINFACHES BEISPIEL:

```
10 RCV B%(*),100,"A"
```

Empfängt jeweils 1 Byte binärer Daten und speichert sie sequentiell im Feld B%(*), bis das Endzeichen "A" (&41) erkannt wird. Die Wartezeit beträgt 0.1 x 100 = 10 Sekunden.

WICHTIGER HINWEIS

1. Daten können nur im Wartezustand einer RCV0 oder eine SENREV-Anweisung (siehe nächste Seite) von einem Peripheriegerät empfangen werden.
2. Liegt das Endzeichen im Bereich von CHR$ &01 bis CHR$ &1F oder von CHR$ &81 bis CHR$ &9F, kann es bei der Verwendung von 7-Bit-Kode ohne Beachtung des SI/SO-Status erkannt werden. Liegt das Endzeichen im Bereich von CHR$ &20 bis CHR$ &7F oder CHR$ &A0 bis CHR$ &FF, wird der SI/SO-Status zu Erkennung notwendig.
3. Funktion 2 kann nur in Verbindung mit 8-Bit-Kode verwendet werden.

SENREV

Sendet Daten mit dem in der CHANNEL-Anweisung festgelegten Format und schaltet dann sofort in den Empfangsmodus um.

● **FUNKTION 1:** Sendet und empfängt Stringdaten

FORMAT: SENREV 《C》, A$, X, D 〔, | ; E〕

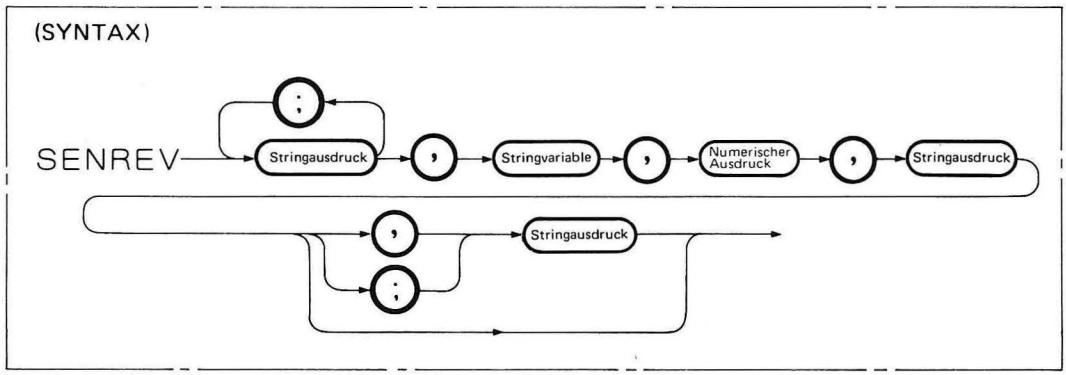
C : String, dessen Inhalt gesendet werden soll

A$: Stringvariable, in der empfangene Daten gespeichert werden

X : Wartezeit (0 – 255), Einheit 0.1 Sekunden

D : 1. Endzeichen

E : 2. Endzeichen



ERLÄUTERUNG:

1. Diese Anweisung sendet zunächst den Inhalt des Strings C und empfängt sofort danach Daten bis zur Erkennung der Endzeichen D bzw E. Diese Daten werden der Stringvariablen A$ zugewiesen.
2. Weitere Details können Funktion 1 der SEND- und der RCV-Anweisung entnommen werden. [CR] - und [LF] -Zeichen (&0D und &0A) werden am Ende der Datenüberagung nicht gesendet. Eine Echo-Funktion beim Datenempfang ist nicht möglich.

EINFACHES BEISPIEL:

```
10 SENREV "READY",D$,0,CHR$ &0D
```

..... Sendet "READY" und empfängt danach Daten bis zur Erkennung des [CR] -Zeichens. Diese Daten werden in der Stringvariablen D$ gespeichert. Die Wartezeit ist unendlich.

WICHTIGER HINWEIS

Soll der Inhalt eines Arrays ausgegeben werden, muß ein Endzeichen am Ende dieser Daten spezifiziert werden. Soll zum Beispiel das [CR] -Zeichen das letzte zu übertragende Zeichen sein, ist folgende Befehlzeile nötig:

```
SENREV A$(*);CHR$ &0D,A$,X,CHR$ &0A
```

● **FUNKTION 2:** Gibt Stringdaten aus und empfängt danach Daten zur Speicherung in einer numerischen Variablen des Typs Integer.

FORMAT: SENREV 《C》, A%, X, D 〔, | ; E〕

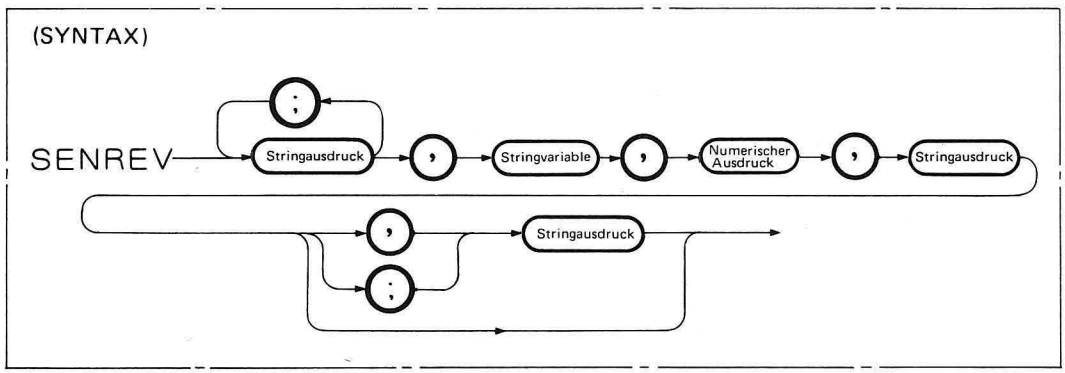
C : String, dessen Inhalt gesendet werden soll.

A%: Integervariable, in der die empfangenen Daten gespeichert werden sollen

X : Wartezeit (0 – 255), Einheit 0.1 Sekunden

D : 1. Endzeichen

E : 2. Endzeichen



ERLÄUTERUNG:

1. Der Inhalt des Strings C wird gesendet. Sofort danach werden Byte für Byte binäre Daten empfangen, bis die Endzeichen D bzw. E erkannt wurden.
   Die empfangenen Daten werden in der Integervariablen A% gespeichert.
2. Weitere Details sind den Anweisungen SEND, Funktion 1, und RCV, Funktion 2, zu entnehmen.

EINFACHES BEISPIEL:

```
10 SENREV "ABC",C%(*),100,CHR$ &0D
```

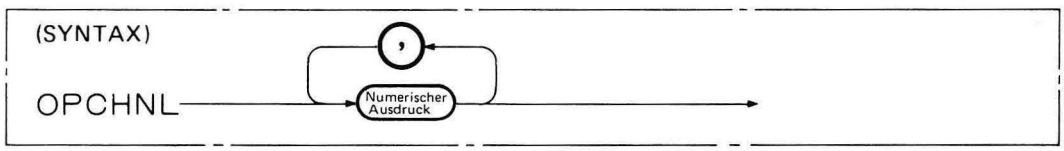
..... Sendet die Zeichen "ABC" und empfängt danach byteweise binäre Daten, die bis zum Empfang des Endzeichens &0D im Array C%(*) gespeichert werden. Die Wartezeit für jedes Element beträgt 10 Sekunden.

OPCHNL

Öffnet einen Zwischenspeicher (1 Byte) für einen spezifierten Kanal

〈VERWENDUNG〉
Zur Erkennung einer Kanalanforderung durch ein Peripherigerät und Verwendung des erkannten Kanals in Verbindung mit der POLLING-Anweisung.

● **FUNKTION 1:** Öffnet einen Zwischenspeicher für den angegebenen Kanal.
FORMAT: OPCHNL 〔X〕
X: Kanalnummer



ERLÄUTERUNG:
OPCHNL öffnet einen Zwichenspeicher (Puffer) für den durch X festgelegten Kanal und löscht dessen Inhalt. Der Zustand eines anderen Kanals wird dadurch nicht geändert. Diese Anweisung bezieht sich nur auf POLLING. SEND, RCV und SENREV werden nicht beeinflußt. Der Puffer hat eine Länge von einem Byte; werden 2 oder mehr Bytes eingegeben, bleibt nur das zuletzt empfangene Zeichen erhalten.
Der CS-Anschluß (Pin 5) des gewählten Kanals nimmt durch die Öffnung des Puffers HIGH-Potential an. Dadurch kann das Peripheriegerät die Empfangsbereischaft des Systems erkennen.

EINFACHES BEISPIEL:
```
10 OPCHNL 0,1,2 ..... Öffnet Pufferspeicher für die Kanäle 1, 2 und 3.
```

● **FUNKTION 2:** Schießt alle geöffneten Zwischenspeicher.
FORMAT: OPCHNL



ERLÄUTERUNG:
Schließt alle geöffneten Kanäle.

WICHTIGER HINWEIS

1. Bei Verwendung des 7-Bit-Codes befindet sich der Kanal im SI-Status (nur Eingabe).
2. Ein geöffneter Kanal wird auch durch SEND, RCV oder SENREV geschlossen.

POLLING

Stellt den Inhalt des Eingabepuffers (ein Byte) eines durch OPCHNL geöffneten Kanals bereit.

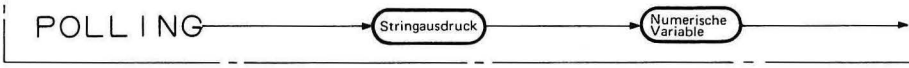
〈VERWENDUNG〉
Feststellung einer Kanalanforderung durch ein Peripheriegerät und deren Weiterverwendung in Verbindung mit der OPCHNL-Anweisung.

FORMAT: POLLING C, A

C : Erkennungszeichen (Vergleichskode)

A : Numerische Variable für die Zuweisung der Kanalnummer

(SYNTAX)



ERLÄUTERUNG:

1. Mit dieser Anweisung kann die Kanalanforderung eines Peripheriegeräts erkannt werden. Dazu wird der Inhalt des Puffers eines mit OPCHNL eröffneten Kanals mit dem Erkennungszeichen C verglichen.
   In Falle der Übereinstimmung wird der Variablen A die entsprechende Kanalnummer zugeordnet (Man beachte die Anweisung CHANNEL auf Seite 294.) Damit ist auch die Verwendung dieses Kanals für nachfolgende Sende- und Empfangsoperationen festgelegt.
   Die Prüfung erfolgt sequentiell beginnend mit der niedrigsten Kanalnummer. Stellt sich eine Übereinstimmung mit dem Erkennungszeichen heraus, wird die Überprüfung der anderen Kanäle sofort abgebrochen. Zum Vergleich wird jedoch nur das erste Zeichen des Strings C herangezogen, alle weiteren Zeichen werden vernachlässigt.
2. Wird keine Übereinstimmung entdeckt, wird der Variablen A der Wert −1 zugewiesen. In diesem Fall wird die bisherige Kanalzuordnung für Ein- und Ausgabeoperationen nicht geändert.

EINFACHES BEISPIEL:

```
100 POLLING CHR$ &05,A
```

Stimmt der Inhalt des Pufferspeichers mit dem Vergleichscode CHR$ &05 überein, wird die entsprechende Kanalnummer in A gespeichert und die Verwendung dieses Kanals festgelegt.

ANWENDUNGSBEISPIEL:

```
100 CHANNEL 0,300,"7E1"
110 CHANNEL 1,1200,"7E2"
120 CHANNEL 2,110,"8N1"
130 OPCHNL 0,1,2
140 "POL":POLLING CHR$ &05,A
150 IF A<0 THEN "POL"
160 SENREV "READY",D$,0,CHR$ &0D
    :
    :
500 GO TO "POL"
```

In diesem Beispiel sind drei Peripheriegeräte mit dem System verbunden. Das System soll auf Anforderung Daten zu diesen Geräten übertragen. (Der Anforderungskode END lautet &05.)
Der anfordernde Kanal wird in Zeile 140 erkannt, in Zeile 160 werden dann zu dem so festgelegten Kanal Daten übertragen. In diesem Beispiel kann ein einmal festgelegter Kanal nicht mehr für weitere Zuweisungen verwendet werden.

WICHTIGER HINWEIS

Jeder einmal zugewiesene Kanal wird automatisch geschlossen.

<PRAKTISCHER TIP>

Die Befehle OPCHNL und POLLING können zum Beispiel dazu verwendet werden, alle angeschlossenen Peripheriegeräte darauf zu prüfen, ob von ihnen eine Anforderung zur Datenübertragung vorliegt. Die Peripheriegeräte müssen dazu nur das in der POOLING-Anweisung definierte Erkennungszeichen senden.

RLIST

Gibt das Programmlisting an einen Drucker aus.

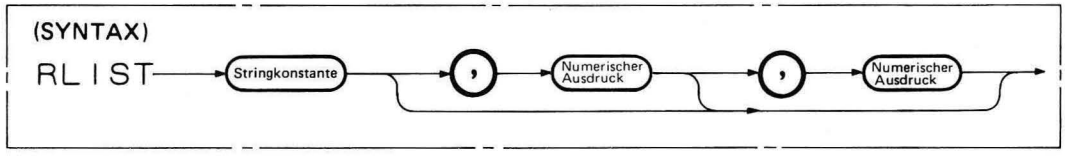
〈VERWENDUNG〉
Ausgabe des Programmlistings an einen Drucker mit RS-232C-Schnittstelle.

FORMAT: RLIST S [, X [, Y]]

S : Titel des Ausdrucks (Programmname)

X : Erste Zeilennummer des auszugebenden Programmteils

T : Letzte Zeilennummer des auszugebenden Programmteils



ERLÄUTERUNG:

Mit dieser Anweisung kann ein Programmlisting ausgegeben werden, das mit der Zeilennummer X beginnt und mit der Zeilennummer Y endet. Gleichzeitig werden Seitennummer sowie Monat, Tag, Stunde und Minute der Bearbeitung ausgegeben. Jede Seite wird außerdem mit der durch S festgelegten Überschrift versehen.

Man beachte dazu auch Funktion 3 des LIST-Befehls.

EINFACHES BEISPIEL:

```
RLIST "CUBIC EQUATION",1000 [ENTER]
```

Es sei zum Beispiel der 20. Oktober, 14:30 Uhr. Das Programm auf der linken Seite würde ausgedruckt das Bild auf der rechten Seite ergeben:

```
1000 !!Gleichung
1010 SUB EQUA
     (X,DX,A(*))
1020*"INIT":P=A(1)
1030 FOR I=2 TO 4
1040 Q=Q*X+P
1050 P=P*X+A(I)
1060 NEXT I
1070 DX=-P/Q
1080 DISP
     "DELTA X=";DX
1090 SUBEND
```

```
P001    10/20 14:30  Kubische
        Gleichung

 1000 !!Gleichung
 1010    SUB EQUA (X,DX,A(*))
 1020 "INIT":P=A(1)
 1030      FOR I=2 TO 4
 1040      Q=Q*X+P
 1050      P=P*X+A(I)
 1060      NEXT I
 1070      DX=-P/Q
 1080      DISP "DELTA X=";DX
 1090      SUBEND

*USED BYTES 133*
```

WICHTIGER HINWEIS

1. Im Programm enthaltene Sonderzeichen (CHR$ &80 bis CHR$ &9F sowie CHR$ &E0 bis &FF) werden in das Zeichen CHR$ &5F konvertiert.
2. Kleinbuchstaben werden direkt ausgegeben. Sollte der Drucker über diese Zeichen nicht verfügen, sind die Programmlistings nicht vollständig darstellbar. Sie könnten bei manchen Druckern auch zu Fehlfunktionen führen.
3. Die Ausführung dieser Anweisung beginnt nicht mit einem Seitenvorschub. Es werden nur (CR) (&OD) und (LF) (&OA) gesendet.
4. Die Ausgabe kann maximal 80 Zeichen pro Zeile umfassen.

TERM

Bestimmt das Ein- und Ausgabeformat bei Verwendung des Systems als Terminal.

〈VERWENDUNG〉
Benutzung des Systems als Terminal

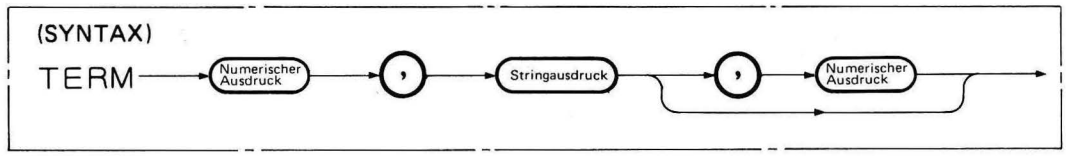
FORMAT: TERM X, C [, Y]

X : Übertragungsrate (110, 200, 300, 600, 1200, 2400, 4800, 9600 Baud)

C : Datenformat

Y : Funktion des (CR) – Zeichens

Ausdruck Y	Funktion
0	Kein Zeilenvorschub nach Empfang des (CR) – Zeichens
1	Zeilenvorschub nach Empfang eines (CR) – Zeichens



ERLÄUTERUNG:

1. Nach Ausführung dieser Anweisung kann das System als Terminal (Eingabe/Ausgabe-Einheit) versendet werden. Man kann dazu die Übertragungsbedingungen wie folgt einstellen:
   Der numerische Ausdruck X definiert die Übertragungsrate (Einheit Bit pro Sekunde), C definiert das Übertragungsformat (siehe dazu Anweisung CHANNEL auf Seite 294) und die Variable Y bestimmt das Verhalten beim Empfang des Zeichens (CR) (&OD).
2. Ist Y gleich 0 oder gar nicht vorhanden, wird kein Zeilenvorschub beim Empfang des (CR)–Zeichens durchgeführt.

EINFACHES BEISPIELE:

1. `TERM 1200,"7E1",1` [ENTER] ..... Verwendet das System als Terminal, setzt die Übertragungsrate mit 1200 Baud fest, zur Datenübertragung wird ein 7-Bit-Kode verwendet. Es wird auf gerade Parität geprüft, die Stopbitlänge beträgt 1. Beim Empfang des CR-Kodes wird ein Zeilenvorschub ausgeführt.

2. TERM 2400,"8N1" [ENTER] ..... Verwendet das System als Terminal, die Übertragungsrate beträgt 2400 Baud, ein 8-Bit-Kode wird verwendet, die Stopbitlänge beträgt 1, eine Paritätsprüfung findet nicht statt.. Beim Empfang eines [CR]-Kodes wird kein Zeilenvorschub ausgeführt.

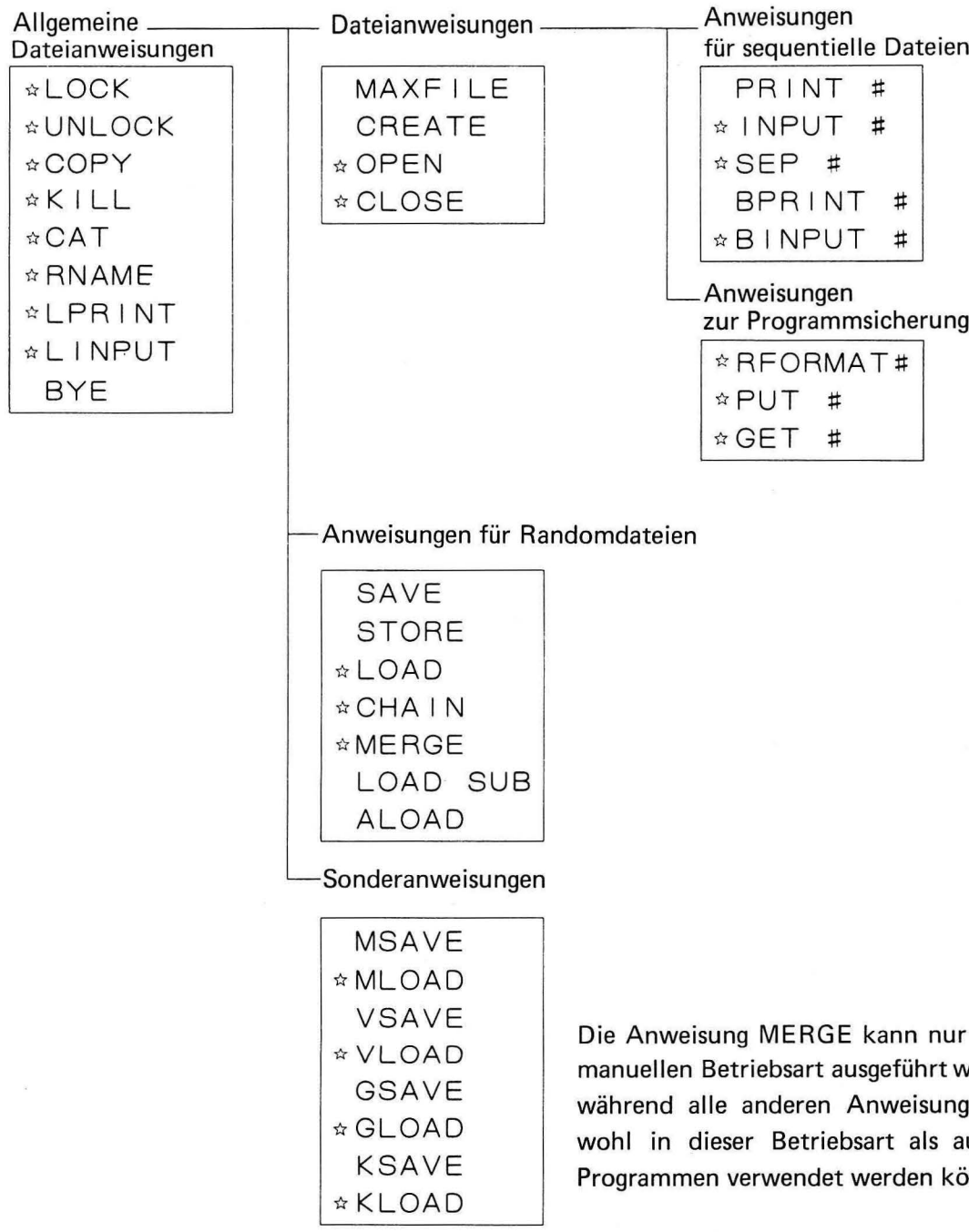
WICHTIGER HINWEIS

1. Die FDOS-Masterdiskette muß bei Ausführung dieses Befehls in Laufwerk A0 eingelegt sein.
2. Dieser Befehl ist nur bei Kanal 0 (eingebaut) wirksam.

IV. ANWEISUNGEN FÜR DEN DISKETTENBETRIEB

DISK-BASIC stelle eine Erweiterung der Sprache BASIC um Befehle zur Behandlung von Diskettendateien dar.
Dieser Abschnitt behandelt detailliert alle Befehle und Anweisungen, die zur Bearbeitungen solcher Diskettendateien notwendig sind. Die verwendeten Dateien können grob in Daten-, Programm–oder Textdateien eingeteilt werden; die Anweisungen von DISK-BASIC sind entsprechend in drei Hauptgruppen einteilbar. Dazu gibt es eine Reihe von Anweisungen die auf alle Dateitype anwendbar sind. Sie dienen der bestmöglichen Verwendung des zur Verfügung stehenden Diskettenspeicherplatzes.
Zusätzliche Informationen enthält der Abschnitt 1.5 — Dateien — auf Seite 35 dieses Buchs.

1. ÜBERSICHT



Die Anweisung MERGE kann nur in der manuellen Betriebsart ausgeführt werden, während alle anderen Anweisungen sowohl in dieser Betriebsart als auch in Programmen verwendet werden können.

Alle mit ☆ markierten Anweisungen können auch auf Dateien angewendet werden, die mit Computern der Serie 3000 erzeugt wurden, jedoch nur unter folgenden Bedingungen:

○ COPY ..... Es können nur Dateien kopiert werden, wenn die verwendeten Diskette mit einem Computer der Serie 3000 initialisiert wurden.

○ OPEN ..... Nur die Zugriffsspezifikationen "I" und "R" sind zulässig.

Werden Dateien von einem Computer der Serie 3000 auf Disketten der Serie 3500 kopiert (Seite 50) oder gesichert (Seite 48), können alle Anweisungen verwendet werden.

2. DATEIVERWALTUNG

Dieser Abschnitt beschreibt alle Befehle, die zur Verwaltung vorhandener Diskettendateien beliebiger Art vorgesehen sind:

LOCK
UNLOCK
COPY
KILL
CAT
RNAME
LPRINT
LINPUT
BYE

Alle Befehle können auch im Programm verwendet werden.

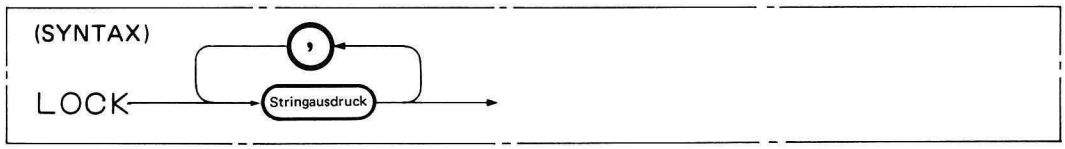
LOCK

Sichert eine Diskettendatei

〈VERWENDUNG〉
Schützt eine Datei vor unbeadsichtigtem Löschen, Überschreiben, oder Verändern des Dateinamens.

FORMAT: LOCK [C]

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")



ERLÄUTERUNG:

1. Sichert die bezeichnete Datei vor Überschreiben und Löschen sowie der Änderung des Dateinamens. Es können mehrere Dateien mit einer LOCK-Anweisung gesichert werden.
2. Ist eine Datei gesichert, wird dies durch Ausgabe eines Sterns (*) nach dem Dateinamen im Inhaltsverzeichnis der Diskette angezeigt. Zur Ausgabe dieses Verzeichnisses dient die Anweisung CAT (Seite 320).
3. Es wird empfohlen, Systemdateien und andere wichtige Datien auf diese Art zu sichern.
4. Eine mit LOCK gesicherte Datei kann mit der Anweisung UNLOCK wieder entsichert werden.

EINFACHES BEISPIELE:

1. LOCK "BASIC/SYS" [ENTER] ..... Sichert die Systemdatei mit dem Namen BASIC/SYS auf Laufwerk A0.

Beispiel eines Disketteninhaltsverzeichnisses (Ausschnitt):

```
BASIC/SYS    *    SYS  21
             ↑
           Sterns
```

2. LOCK "PROBE1","PROBE2" [ENTER] ..... Die beiden mit PROBE1 und PROBE2 bezeichneten Dateien werden gesichert.

<PRAKTISCHER TIP>

Wird die Schreibschutzkerbe mit Hilfe eines Aufklebers verschlossen, kann der gesamte Disketteninhalt geschützt werden (siehe auch Bedienerhandbuch, Seite 27). Da die Diskette nun nicht beschrieben werden kann, ist die Veränderung von Dateiinhalten, Sicherungszuständen oder Namen ausgeschlossen.

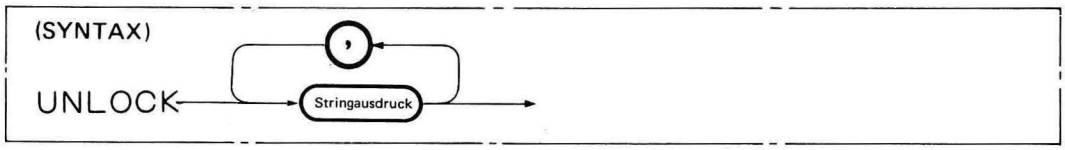
UNLOCK

Entsichert die angegebene Datei

〈VERWENDUNG〉
Ermöglicht das Beschreiben, Löschen oder Umbenennen einer Datei.

FORMAT: UNLOCK 〔C〕

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")



ERLÄUTERUNG:

Entsichert eine geschützte Datei und erlaubt damit das Beschreiben, Löschen oder Umbenennen dieser Datei. Es können mehrere Dateien gleichzeitig entsichert werden.

EINFACHES BEISPIELE:

1. UNLOCK "BASIC/SYS" ENTER
   ..... Entsichert die geschützte Datei BASIC/SYS.
2. UNLOCK "PROBE1,SHARP","PROBE2,SHARP:A1" ENTER
   ..... Entsichert die beiden Dateien mit der Bezeichnung PROBE1, SHARP und PROBE2 SHARP: A1.

<PRAKTISCHER TIP>
Eine entsicherte Datei verliert bei der Ausgabe des Inhaltsverzeichnisses durch die Anweisung CAT (Seite 320) die * – Kennzeichnung.

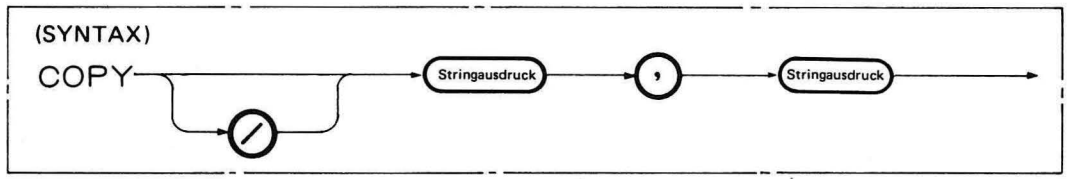
COPY

Kopiert Dateien

〈VERWENDUNG〉
Kopieren von Inhalten beliebiger Dateien zu Speicherzwecken.

FORMAT: COPY [/] C, D

/ : Schützt Variableninhalte
C : Dateikennung (Quelldatei)
D : Dateikennung (Zieldatei)



ERLÄUTERUNG:

Kopiert den Inhalt der Datei C in eine neue Datei mit der Bezeichnung D, ohne den Inhalt der Quelldatei zu verändern. Wird auf die / – Option vor dem Dateinamen C verzichtet, werden beim Kopiervorgang alle Variablen im Speicher gelöscht.

Mit der/-Option können die Variableninhalte jedoch nur dann geschützt werden, wenn der freie Benutzerspeicher bis zu Adresse &COOO mindestens 84 Bytes ist, andernfalls wird Fehler 216 gemeldet.

EINFACHES BEISPIELE:

1. `COPY "ORIGINAL","KOPIE:A1"` ENTER
   ..... Kopiert den Inhalt der Datei ORIGINAL in Laufwerk A0 in die Datei KOPIE in Laufwerk A1.

2. `COPY "EINS","ZWEI"` ENTER
   ..... Kopiert den Inhalt der Datei EINS in Laufwerk A0 in eine neue Datei mit dem Namen ZWEI im selben Laufwerk.

WICHTIGER HINWEIS

1. Es ist entsicherte den Inhalt einer Datei in eine Datei gleichen Namens zu kopieren, wenn sich beide Dateien auf der selben Diskette befinden. Es ist ebenfalls unmöglich, geheime Dateien zu kopieren.
2. Dieser Befehl löscht automatisch Puffer 1 (siehe Seite 330).
3. Die Speicherbelegung hängt vom Systemplan ab. Ferner ist die Größe (in Bytes) des freien Benutzerbereiches vom Variablentyp abhängig. Bei Fehlermeldung 216 Variablen löschen, falls möglich.

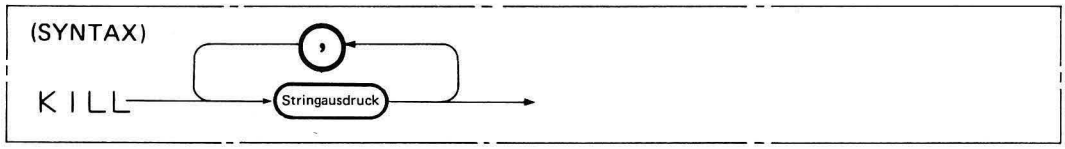
KILL

Löscht eine Datei.

〈VERWENDUNG〉
Löschen einer nicht mehr benötigten Datei.

FORMAT: K I L L 〈C〉

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")



ERLÄUTERUNG:

KILL löscht eine mit C festgelegte Datei und gibt den von ihr belegten Speicherplatz wieder frei. Es können mehrere Dateien mit einer KILL-Anweisung gelöscht werden.

EINFACHES BEISPIEL:

KILL "PROBE,SHARP" ENTER ..... Löscht die Datei mit der Dateikennung PROBE, SHARP.

WICHTIGER HINWEIS

Dateien, die mit der LOCK-Anweisung (Seite 315) gesichert wurden, können nicht gelöscht werden. Der Versuch, die KILL-Anweisung dennoch auszuführen, führt zur Meldung "ERROR 226".

CAT

Gibt das Inhaltsverzeichnis einer Diskette aus

〈VERWENDUNG〉
① Feststellung aller auf einer Diskette gespeicherten Dateien.
② Feststellung aller Kenngrößen von Diskette und Dateien.
③ Feststellung des verbleibenden Speicherplatzes auf der Diskette.

– Inhaltsverzeichnis einer Diskette –

Das Inhaltsverzeichnis enthält Informationen über Diskettenname, den Zustand der automatischen Lade-Funktion (siehe auch Seite 370), die Namen der gespeicherten Dateien, den Sicherungszustand der Dateien und ihre Größen (Blockzahl). Bei Random-Dateien wird zusätzlich Anzahl und Länge der Records ausgegeben.

Auf Wunsch kann Datum und Stunde der Aufzeichnung jeder Datei ausgegeben werden.

Am Ende des Inhaltsverzeichnisses wird der verbleibende Speicherplatz der Diskette in Blöcken ausgegeben. Jeder Block umfaßt 2 KByte.

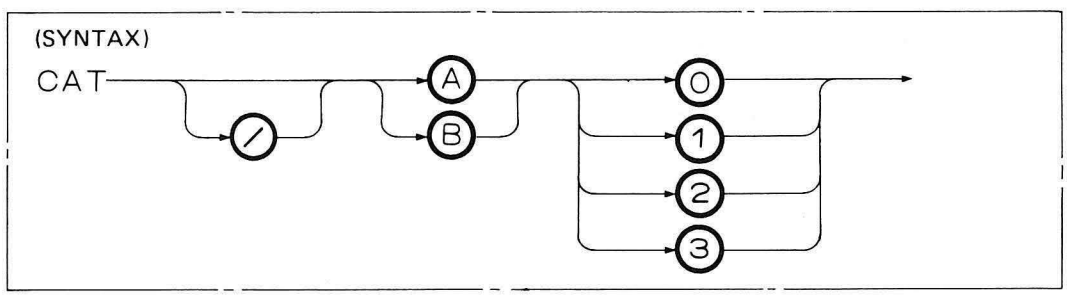
● **FUNKTION 1:** Ausgabe des Inhaltsverzeichnisses auf dem Bildschirm.

FORMAT: CAT 〔/〕 CH m

/ : Veranlaßt die Ausgabe von Datum und Stunde

CH : Kanalnummer (A oder B)

m : Laufwerknummern(0, 1, 2 oder 3)



ERLÄUTERUNG:

1. CAT gibt das Inhaltsverzeichnis der Diskette im durch Kanal-Laufwerk-Nummer spezifizierten Laufwerk am Bildschirm aus.
2. Die Angabe der (/)-Option vor der Kanal-Laufwerk-Nummer bewirkt die Ausgabe der Dateinamen zusammen mit Datum und Stunde des letzten Zugriffs aud diese Datei. Die Datumsangabe hat das Format "Monat. Tag. Stunde". Diese Zusatzdaten beziehen sich auf die TIME-Funktion und werden beim Ausführen der Anweisungen CLOSE, SAVE, STORE, MSAVE, VSAVE, KSAVE und GSAVE erneuert.

3. Gesicherte Dateien werden bei der Ausgabe mit einem Stern (*) auf der rechten Seite des Dateinamens gekennzeichnet. Geheime Dateien werden mit dem Zeichen S (secret) an gleicher Stelle versehen. Der Dateityp wird mit folgenden Abkürzungen dargestellt:

Typ	Bedeutung
(leer)	Sequentielle oder Random-Datei
BAS	BASIC-Zwischenkode-Datei (Programmdatei)
TXT	Textdatei in ASCII-Format (Programmdatei)
OBJ	Maschinenkode-Datei
DSP	Bildschirmdatei (Textebene)
GRA	Bildschirmdatei (Graphikebene)
KEY	Belegung der Funktionstasten
SYS	Systemdatei (z.B. BASIC)

EINFACHES BEISPIELE:

1. CAT A1 [ENTER] ..... Gibt das Inhaltsverzeichnis einer in Laufwerk A1 eingelegten Diskette aus.
2. CAT /A1 [ENTER] ..... Gibt zusätzlich zu obigem Beispiel Datum und Uhrzeit der letzten Dateibearbeitung an.

ANWENDUNGSBEISPIEL:

CAT /A0 [ENTER] ..... Befehlseingabe

Bildschirmanzeige:

```
CAT   A0                                  V1.0A
VOLUME NAME-SHARP
ALOAD       -TEST/SR:A0,  100
-------------------------------------------
FILE NAME        TYPE SIZE     DATE

SUBCPU/SYS    *   SYS   4
DOS/SYS       *   SYS   8
BASIC/SYS     *   SYS  21
DEMO1             OBJ   1    05.21.10
DEMO2          S  BAS   2    05.22.09
TEST/SR       *S  TXT   5    06.03.15
DATA.0                 18    06.11.14
  REC= 1100 LEN=   28
DATA.1        *        3     05.27.11
DATA.2                 8     06.16.13
K10-20            KEY   1    07.18.09
CRT.12            DSP   2    06.25.14
-------------------------------------------
FREE    79
```

Beschreibung der automatischen Ladefunktion

Diese Anzeige wird unterdückt, wenn der / weggelassen wird.

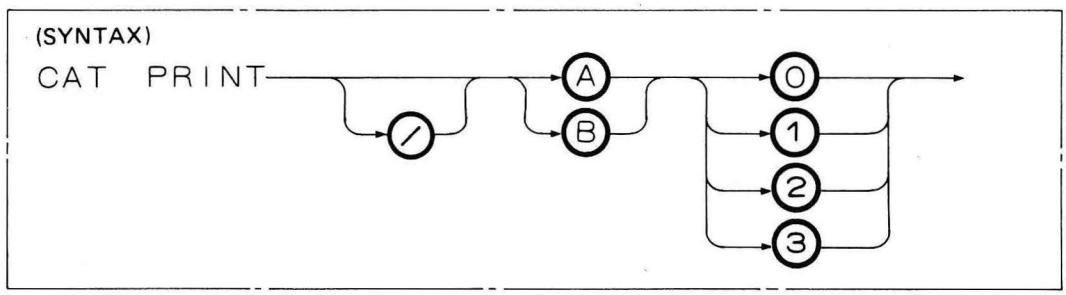
● **FUNKTION 2:** Ausgabe des Inhaltsverzeichnisses auf einen Drucker

FORMAT: CAT PRINT 〔/〕 CH m

/ : Veranlaßt die Ausgabe von Datum und Stunde

CH: Kanalnummer (A oder B)

m : Laufwerknummer (1, 2, 3 oder 4)



ERLÄUTERUNG:

CAT PRINT gibt das Inhaltsverzeichnis einer Diskette auf einem Drucker aus. Die Diskette muß sich in dem durch Kanal- und Laufwerknummer spezifizierten Laufwerk befinden. Die Anweisung entspricht in allen anderen Eigenschaften Funktion 1.

EINFACHES BEISPIEL:

CAT PRINT /A1 [ENTER] ..... Gibt das Inhaltsverzeichnis der in Laufwerk A1 befindlichen Diskette auf einem Drucker aus. Die Dateinamen werden zusammen mit Datum und Stunde des letzten Dateizugriffs ausgedruckt.

WICHTIGER HINWEIS

Die Ausgabe kann mit Hilfe der (HALT)- oder der Leertaste unterbrochen werden. Die (HALT)-Taste bricht die Ausführung vollständig ab, während die Ausgabe im anderen Fall durch einen erneuten Druck auf die Leertaste fortgesetzt werden kann. Wird nach einer Unterbrechung durch die Leertaste die [ENTER]-Taste gedrückt, wird nur noch der verbliebene Diskettenspeicherplatz angegeben; die Ausführung der Anweisung ist dann abgeschlossen.

Nach einer Unterbrechung mit der Leertaste werden nur die Leer- und die [ENTER]-Taste akzeptiert, bei laufender Ausgabe nur Leer- und (HALT)-Taste.

RNAME

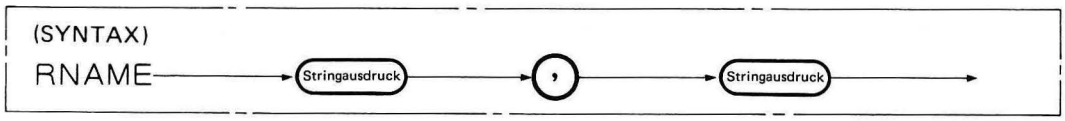
Ändert einen Dateinamen

〈VERWENDUNG〉
Umbenennen einer Datei

FORMAT: RNAME C, D

C : Vorhandener Dateiname ("Name, Diskettenname: Kanal-Laufwerknummer")
D : Neuer Dateiname (ohne Dateikennung)

(SYNTAX)



ERLÄUTERUNG:

1. Der Name der Datei C wird durch den neuen Namen C ersetzt.
2. Die Position der Datei im Inhaltsverzeichnis bleibt auch nach Ausführung der RNAME-Anweisung erhalten.

EINFACHES BEISPIELE:

1. `RNAME "PROBE:A1","PROBE2"` ENTER
   ..... Ändert den Namen der Datei PROBE auf der Diskette in Laufwerk A1 in PROBE2 ab.
2. `RNAME "EINS","ZWEI"` ENTER
   ..... Ändert den Namen der Datei EINS auf der Diskette in Laufwerk A0 in ZWEI ab.

WICHTIGER HINWEIS

1. Der Versuch, RNAME auf eine gesicherte Datei anzuwenden (siehe auch Seite 315), führt zu einem Fehler (Fehler 226). Wird der abzuändernde Dateiname auf der spezifizierten Diskette nicht gefunden, tritt ebenfalls ein Fehler auf (Fehler 210).
2. Diskettenname und Kanal-Laufwerknummer müssen nur für den alten Dateinamen spezifiziert werden, eine eventuell vorhandene Angabe beim neuen Dateinamen wird ignoriert.

LPRINT

Ändert den Diskettennamen.

〈VERWENDUNG〉
Zur Änderung des bei der Initialisierung vergebenen Diskettennamens.

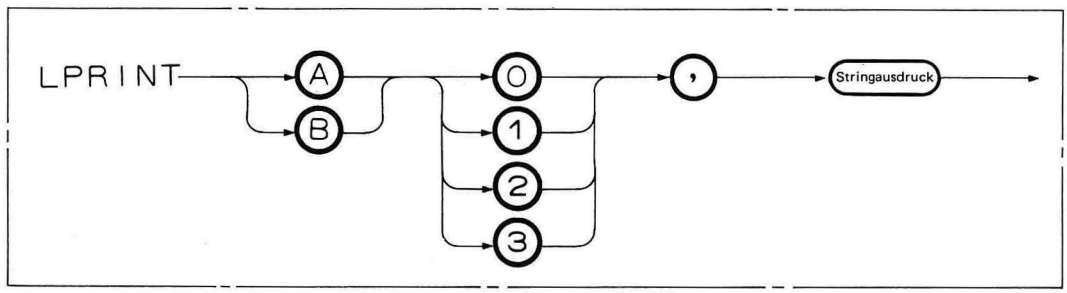
● **FUNKTION 1:** Festlegung von Kanal- und Laufwerknummer durch Konstanten.

FORMAT: LPRINT CH m, C

CH : Kanalnummer (A oder B)

m : Laufwerknummer (1, 2, 3 oder 4)

C : Neuer Diskettenname



ERLÄUTERUNG:

Einer Diskette, die sich in dem durch Kanal CH und Laufwerknummer m festgelegten Laufwerk befindet, wird ein neuer, durch den String C definierter Name zugeordnet. Der neue Name kann aus maximal 6 Großbuchstaben und Ziffern bestehen, Leerzeichen dürfen jedoch nicht darunter sein.

EINFACHES BEISPIEL:

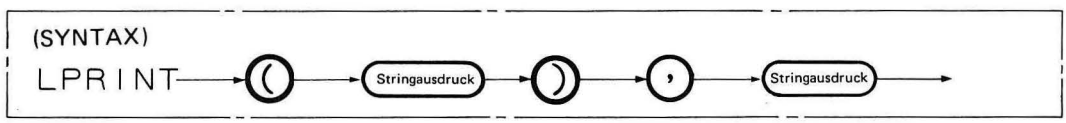
LPRINT A0,"AUGUST" ENTER ..... Ändert den Namen der in Laufwerk A0 befindlichen Diskette in AUGUST ab.

● **FUNKTION 2:** Festlegung der Kanal-Laufwerknummer durch eine Stringvariable.

FORMAT: LPRINT (D), C

D : Kanal-Laufwerknummer (Stringvariable)

C : Neuer Diskettenname



ERLÄUTERUNG:

Definiert die Kanal-Laufwerknummer durch die Stringvariable D. Sonstige Funktion wie 1.

EINFACHES BEISPIEL:

```
LPRINT (C$), "PROBE" ENTER
```

..... Ändert den Namen der Diskette, die sich im durch C$ festgelegten Laufwerk befindet, in den Namen PROBE ab.

<HILFREICHER TIP>

Die Änderung des Diskettennamens kann durch Ausgabe des Inhaltsverzeichnisses (Befehl CAT, Seite 320) geprüft werden.

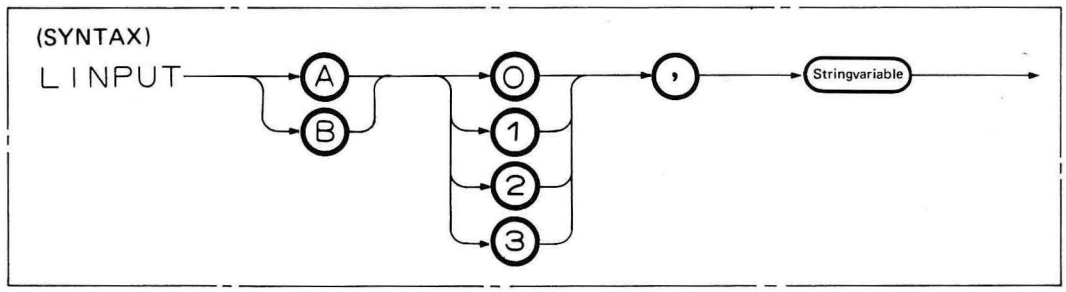
LINPUT

Der Diskettenname wird einer Variablen zugewiesen.

〈VERWENDUNG〉
Feststellen, ob die richtige Diskette verwendet wird.

● **FUNKTION 1:** Darstellung der Kanal-Laufwerknummer als Konstante.
FORMAT: L I NPUT CH m, A$

CH: Kanalnummer (A oder B)
m : Laufwerknummer (1, 2, 3 oder 4)
A$: Stringvariable, ihr wird der Diskettenname zugewiesen



ERLÄUTERUNG:

Der Variablen A$ wird der Name einer Diskette zugewiesen, die sich in dem durch die Konstante CH m bestimmten Laufwerk befindet.

EINFACHES BEISPIEL:

```
10 LINPUT A1,X$
```
..... Der Variablen A$ wird der Name der Diskette zugewiesen, die sich im Laufwerk A1 befindet.

ANWENDUNGSBEISPIEL: Es ist zu prüfen, ob die richtige Diskette eingelegt wurde.

```
10 LINPUT A0,X$
```
..... Liefert den Diskettennamen der in Laufwerk A0 befindlichen Diskette. X$ enthält diesen Namen.

```
20 IF P$<>"SHARP" DISP "Falsche Diskette!":END
```
..... Beendet das Programm, wenn die geprüfte Diskette nicht den Namen "SHARP" trägt.

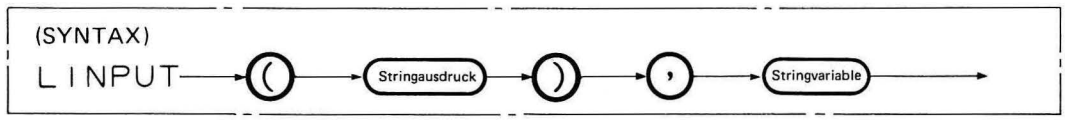
```
30 END
```

● **FUNKTION 2:** Festlegen der Kanal-Laufwerknummer mit einer Stringvariablen

FORMAT: L I NPUT (C), A$

C : Kanal-Laufwerknummer

A$: Stringvariable, der der Diskettenname zugeordnet wird.



ERLÄUTERUNG:

Die Kanal-Laufwerknummer wird mit Hilfe der Stringvariablen C festgelegt. Entspricht sonst Funktion 1.

EINFACHES BEISPIEL:

```
20 LINPUT (C$),X$
```

Weist den Namen der Diskette, die im durch C$ festgelegten Laufwerk eingelegt ist, der Variablen X$ zu.

ANWENDUNGSBEISPIEL: Man prüfe, ob in Laufwerk A0 und A1 die richtigen Disketten eingelegt sind.

```
100 READ B$
110 LINPUT (B$),X$
120 IF B$="A1" THEN 150
130 IF X$<>"EINS" DISP "Falsche Diskette!":END
140 GO TO 100
150 IF X$<>"ZWEI" DISP "Falsche Diskette!":END
160 DATA "A0","A1"
170 END
```

Der Diskettenname der im durch B$ festgelegten Laufwerk befindlichen Diskette wird der Variablen X$ zugewiesen.

Über die READ-DATA-Anweisung wird LINPUT das zu prüfende Laufwerk zugewiesen. Stimmen die Namen nicht überein, wird das Programm beendet.

BYE

Übergibt die Systemkontrolle von DISK-BASIC an FDOS

〈VERWENDUNG〉
Ausführung von FDOS-Befehlen

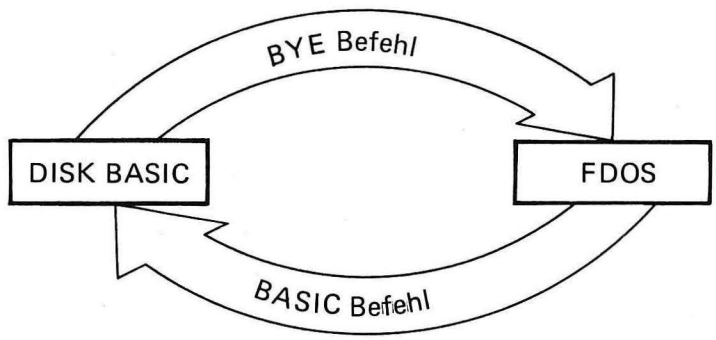
FORMAT: BYE

(SYNTAX)



ERLÄUTERUNG:

Der BYE-Befehl übergibt die Programmkontrolle von BASIC and FDOS. Dadurch können FDOS-Befehle ausgeführt werden. Um von FDOS zu BASIC zurückzukehren, muß der Befehl BASIC (siehe auch Seite 44) verwendet werden. Nachfolgende Graphik stellt den Zusammenhang zwischen den Befehlen BYE und BASIC dar:



HINWEIS

Bei Ausführung des Befehls BYE werden alle Dateien geschlossen (siehe auch Seite 336).

3. DATEIVERARBEITUNG

Dieser Abschnitt behandelt Anweisungen, die allgemein auf alle Dateitypen anwendbar sind:

MAXFILE
CREATE
OPEN
CLOSE

Diese Anweisungen können auch in der manuellen Betriebsart angewendet werden.

MAXFILE

Legt die Zahl der Zwischenspeicher fest oder ermittelt ihre Anzahl.

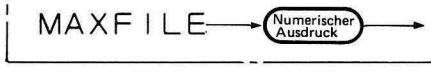
〈VERWENDUNG〉
① Aufbau von Zwischenspeichern.
② Feststellung der verwendeten Pufferzahl.

● **FUNKTION 1:** Festlegung der gewünschten Pufferzahl.

FORMAT: MAXFILE X

X : Anzahl der Puffer (1–7)

(SYNTAX)



ERLÄUTERUNG:

1. MAXFILE öffnet eine durch X bestimmte Zahl von Zwischenspeicher. Diese Anweisung legt dadurch auch die Zahl der gleichzeitig bearbeitbaren Dateien fest. Es können maximal 7 Puffer mit einer Länge von 64 Bytes definiert werden.
2. Bei Ausführung dieser Anweisung werden alle Variablen gelöscht.
3. Im Einschaltmoment wird die Zahl der Pufferspeicher auf 3 eingestellt.
4. Die Anzahl der Speicher bleibt bis zu einer Neudefinition erhalten.

EINFACHES BEISPIEL:

```
10 MAXFILE 4
```
..... Definiert 4 Puffer und löscht alle Variablen. Der Pufferspeicher hat dadurch einen Umfang von (64x4) Bytes.

HINWEIS

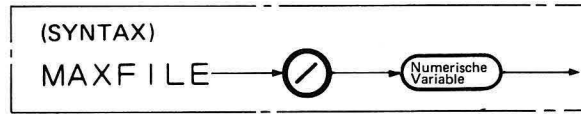
Da durch diese Anweisung alle Variablen gelöscht werden, sollte sie nur im Initialisierungsteil eines Programms verwendet werden.

● **FUNKTION 2:** Stellt die Anzahl der verwendeten Puffer fest.

FORMAT: MAXFILE /N

N : Numerische Variable, ihr wird die aktuelle Pufferzahl zugewiesen.

(SYNTAX)



ERLÄUTERUNG:

Der numerischen Variablen N wird die aktuelle Speicherzahl zugewiesen. Dadurch läßt sich die verwendete Pufferzahl feststellen. Nach Einschalten des Geräts sind 3 Zwischenspeicher definiert (siehe auch Funktion 1).

EINFACHES BEISPIEL:

```
10 MAXFILE #A
```
..... A wird die Anzahl der verwendeten Puffer zugewiesen.

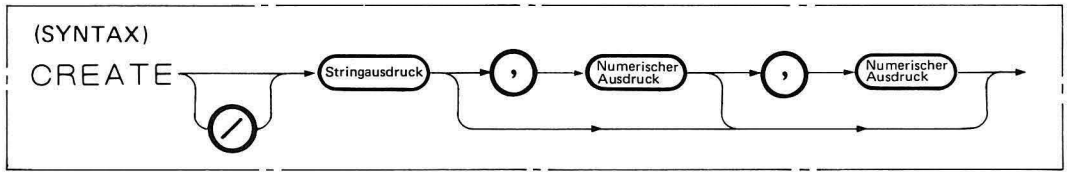
CREATE

Reserviert Speicherplatz auf einer Diskette

〈VERWENDUNG〉
Aufbau einer Datei (Insbesondere einer Random-Datei).

FORMAT: CREATE [/] C [, X [, Y]]

/ : Löschanweisung
C : Dateikennung
X : Anzahl der Datensätze (1—32767)
Y : Länge der Datensätze (2—65535)



ERLÄUTERUNG:

1. Reserviert den für X Datensätze der Länge Y notwendigen Speicherplatz auf der Diskette. Die Zahl der Datensätze darf zwischen 1 und 32767 liegen, ihre Länge darf bis zu 65535 Bytes betragen.
2. Wird die Satzlänge Y nicht spezifiziert, wird sie vom System auf 256 Bytes festgelegt.
3. Werden Satzlänge Y und Anzahl X nicht spezifiziert, entspricht der reservierte Speicherplatz einem Block, d.h. 8 mal 256 Bytes (8 Datensätze mit der Länge 256).
4. Wird vor der Dateikennung ein (/) gesetzt, erhalten alle reservierten Diskettenspeicherplätze den Wert 0. (Der Bereich wird gelöscht.)

EINFACHES BEISPIELE:

1. `20 CREATE "PROBE:A0",100,20`
   ..... Reserviert den für 100 Records der Länge 20 notwendigen Diskettenspeicherplatz, also 2000 Bytes. Damit wird ein ganzer Block für die Datei PROBE: A0 belegt (2000 ≦ 2048).
2. `20 CREATE /"PROBE:A0",50`
   ..... Reserviert 50 Records mit der Standardlänge 256 für die Datei PROBE: A0 und löscht sie. Belegt werden dadurch zwei Blöcke (3840 ≦ 4096).
3. `CREATE /"PROBE:A0"` ..... Reserviert den Standardbereich, also einen Block (2KBytes) entsprechend 8 Records mit 256 Bytes Länge und löscht diesen Bereich.

WICHTIGER HINWEIS

1. Vor Aufbau einer Randomdatei muß der benötigte Speicherplatz auf der gewünschten Diskette mit dieser Anweisung reserviert werden.
2. Diese Anweisung kann nicht mehrfach mit derselben Dateikennung ausgeführt werden.

<PRAKTISCHER TIP>

Zum Aufbau einer sequentiellen Datei ist diese Anweisung nicht notwendig. Sie kann jedoch dazu verwendet werden, zu prüfen, ob auf der Diskette noch ausreichend Speicherplatz vorhanden ist.

OPEN

Gibt eine Datei zur Bearbeitung frei.

〈VERWENDUNG〉
Öffnen einer Datei, um Daten lesen oder schreiben zu können.

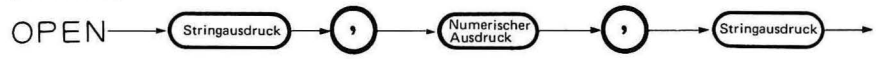
FORMAT: OPEN C, X, D

C : Zugriffsart
- "I" ..... Lesen sequentieller Daten (Eingabe/Input)
- "O" ..... Schreiben sequentieller Daten (Ausgabe/Output)
- "A" ..... Anfügen von Daten an eine sequentielle Datei (Append)
- "R" ..... Bearbeiten einer Randomdatei (Lesen und schreiben)

X : Nummer des verwendeten Puffers

D : Dateikennung (" Dateiname, Diskettenname: Kanal-Laufwerknummer")

(SYNTAX)



ERLÄUTERUNG:

Gibt eine durch die Dateikennung D festgelegt Datei zur Bearbeitung frei. Dieser Vorgang wird auch als Öffnen einer Datei bezeichnet. Die Art der Bearbeitung wird durch die Stringkonstante C bestimmt. Die Übertragung der Daten erfolgt über den durch X spezifizierten Puffer; X muß im Bereich der durch MAXFILE (Seite 330) definierten Anzahl liegen. Mit der Anweisung CLOSE (siehe nächste Anweisung) wird die Dateibearbeitung beendet.

EINFACHES BEISPIELE:

1. `10 OPEN "I",1,"ABCD:A1"`
   ..... Bereitet die Datei ABCD: A1 zum sequentiellen Lesen der Daten vor und verwendet dazu Puffer 1. Das Beschreiben der Datei ist in diesem Zustand nicht möglich.
2. `10 OPEN "A",2,"XYZ,SHARP:A0"`
   ..... Bereitet die Datei XYZ, SHARP: A0 zum Anfügen weiterer Daten vor. Die Datenübertragung erfolgt über Puffer 2.

WICHTIGER HINWEIS

1. Ist die Bearbeitung der durch diese Anweisung geöffneten Datei beendet, muß sie durch die Anweisung CLOSE (Seite 336) geschlossen werden.
2. Vor der Bearbeitung einer Randomdatei muß der benötigte Speicherbereich durch die CREATE-Anweisung reserviert worden sein (Seite 332).
3. Wird eine Anweisung ausgeführt, die der Zugriffsart widerspricht, wird ein Fehler gemeldet (Fehler 206). Dies ist z.B. der Fall, wenn bei der Zugriffsart "I" eine PRINT# – Anweisung ausgeführt werden soll.
4. Wird versucht, mit einem schon zugewiesenen Puffer eine weitere Datei zu öffnen, tritt Fehler 202 auf. Der Puffer Kann erst nach Ausführung der CLOSE-Anweisung neu verwendet werden.
   Es ist jedoch möglich, eine Datei durch Zuweisung eines neuen Puffers nochmals zu öffnen. (wenn die sequentielle Datei zum Lesen bestimmt ist.)
5. Geöffnete Dateien können bis zur vollständigen Schließung durch die CLOSE-Anweisung nicht gelöscht werden (Seite 319).

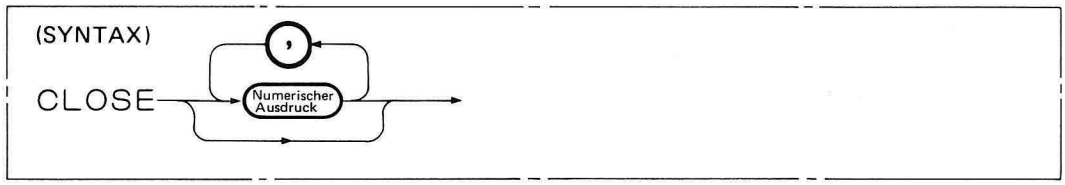
CLOSE

Beendet die Bearbeitung einer Datei.

〈VERWENDUNG〉
Schließen einer Datei nach Lese- oder Schreiboperationen

FORMAT: CLOSE [[X]]

X : Puffernummer



ERLÄUTERUNG:

1. Beendet die Bearbeitung der Datei, der der Puffer X zugewiesen wurde, und löscht ihn. Die Angabe mehrerer Puffer ist möglich.
2. Wird keine Puffernummer spezifiziert, werden alle geöffneten Dateien geschlossen.

EINFACHES BEISPIELE:

1. `100 CLOSE 2,3` ..... Schließt die Dateien, denen die Puffer 2 und 3 zugewiesen sind und löscht die Dateien 2 und 3.
2. `200 CLOSE` ..... Schließt alle offenen Dateien und löscht die Puffer.

WICHTIGER HINWEIS

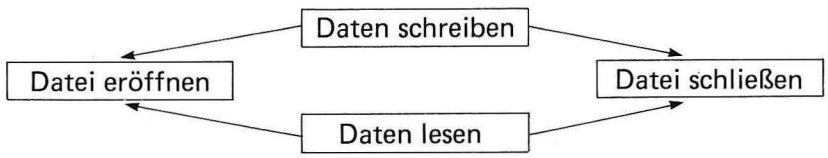
1. Alle Disketten, die geöffnete Dateien enthalten, dürfen erst dann aus den Laufwerken entfernt werden, wenn eine CLOSE- Anweisung für diese Dateien ausgeführt wurde. Ebenso darf das System erst ausgeschaltet werden, wenn alle Dateien geschlossen sind. Andernfalls gehen alle im Puffer befindlichen Daten verloren und das Schreiben weiterer Daten oder ein gezielter Zugriff zum Lesen eines Werts kann zu erheblichen Störungen führen.
2. Die Ausführung dieser Anweisung übergeht die Anweisungen SEP# (Seite 347) und RFORMAT# (Seite 350).
3. Diese Anweisung kann nicht auf Puffer angewendet werden, die noch nicht keine Zuweisung erhalten haben.
4. Die Ausführung der Befehle LOAD (Seite 360), RUN (Seite 78), MERGE (Seite 364) und NEW sowie Teile des CHAIN-Befehls haben eine automatische Ausführung der CLOSE-Anweisung zur Folge.

3-1. SEQUENTIELLE DATEIEN

Dieser Abschnitt beschreibt Anweisungen zum Lesen und Schreiben sequentieller Dateien:

BPRINT #
BINPUT #
PRINT #
INPUT #
SEP #

Diese Anweisungen können auch in der manuellen Betriebsart verwendet werden. Um Daten sequentiell lesen oder schreiben zu können, muß folgende Prozedur eingehalten werden:



Die folgenden einfachen Beispiele sollen dazu dienen, die im einzelnen notwendigen Schritte zu verdeutlichen. Dazu wollen wir die Programme Zeile für Zeile durchgehen und versuchen, die Aufgabe der verwendeten Anweisungen zu verstehen.

a) Schreiben einer sequentiellen Datei:

```
 10 MAXFILE 1 ..... Pufferzahl definieren
 20 OPEN "O",1,"TEL:A0" ..... Datei öffnen
 30 INPUT "Personenzahl?";N
 40 PRINT #1,N; ..... Ausführen eines Schreibbefehls
 50 FOR I=1 TO N
 60 INPUT "Name?";N$
 70 INPUT "Telefon?";T$
 80 PRINT #1,N$;CHR$ &0D;T$ ..... Ausführen eines Schreibbefehls
 90 NEXT
100 CLOSE 1 ..... Datei schließen
110 END
```

In diesem Beispiel wurden alle Daten in ASCII aufgezeichnet. Zum Schreiben binärer Daten kann die Anweisung BPRINT # (Seite 339) verwendet werden.

b) Lesen einer sequentiellen Datei

```
 10 MAXFILE 1 ..... Definition der Pufferzahl
 20 OPEN "I",1,"TEL:A0" ..... Eröffnen der Datei
 30 INPUT #1,N ..... Ausführen eines Lesebefehls
 40 DISP "Personenzahl";N
 50 DISP "Name","Telefon"
 60 FOR I=1 TO N
 70 INPUT #1,N$,T$ ..... Ausführen eines Lesebefehls
 80 DISP N$,T$
 90 NEXT
100 CLOSE 1 ..... Schließen der Datei
110 END
```

In diesem Beispiel müssen alle zu lesenden Daten im ASCII-Format vorliegen. Zum Lesen binärer Daten kann die Anweisung BINPUT # (Seite 341) verwendet werden.

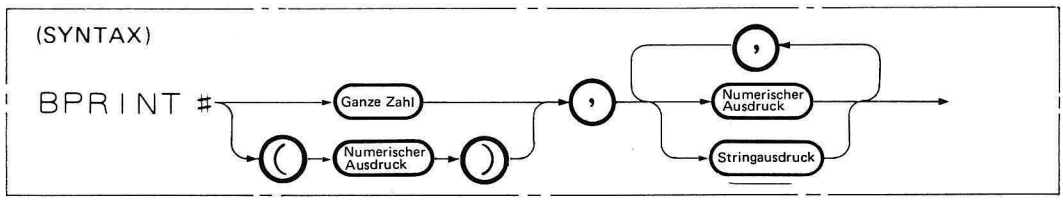
BPRINT

Schreibt binäre Daten in eine sequentielle Datei.

〈VERWENDUNG〉
① Aufbau einer sequentiellen Datei für binäre Daten.
② Ergänzen und Überschreiben einer sequentiellen Datei mit binären Daten.

FORMAT: BPRINT # n | (X), {Y | C}

n : Nummer des Puffers
X : Nummer des Puffers in Form eines math. Ausdrucks
Y : Numerische Daten, die aufgezeichnet werden sollen
C : Stringdaten, die aufgezeichnet werden sollen



ERLÄUTERUNG:

1. BPRINT# schreibt sequentiell alle durch die numerischen Variablen Y oder die Stringvariablen C bestimmten Binärwerte auf die Diskette. Dazu wird der durch n spezifizierte Puffer verwendet.
2. Wurde mit der OPEN-Anweisung die Funktion "0" gewählt (Seite 334), beginnt die Neuaufzeichnung stets beim ersten Element der Datei, d.h. vorhandene Daten werden überschrieben. Bei Verwendung der Funktion "A" werden die neuen Daten jedoch an die schon vorhandene Datei angefügt.
3. Da bei dieser Aufzeichnungsart alle Variablen und Konstanten binär kodiert aufgezeichnet werden, benötigen sie einen wie in nachfolgender Tabelle angegebenen Speicherplatz.

Datentyp	Speicherbedarf
Numerische Variablen und Konstanten des Typs Integer (z.B. 3%, A%)	2 Bytes
Numerische Variablen und Konstanten des Typs Real (z.B. 4.56, X)	8 Bytes
Stringkonstanten (z.B. "ABCD")	Zeichenlänge + 2 Bytes
Stringvariablen des $-Typs (z.B. A$)	18 Bytes
Stringvariablen des @-Typs (z.B. E@) (Ohne DIM-Deklaration 16+4 Byte)	Durch DIM festgelegte Zeichenlänge + 4 Bytes
Stringarrayvariablen des $-Typs	(Durch DIM festgelegte Zeichenlänge * Anzahl der Elemente) + 2 Bytes
Stringarrayvariablen des @-Typs	((Durch DIM festgelegte Zeichenlänge + 2) * Anzahl der Elemente + 2) Bytes.
Numerischer Ausdruck (z.B. 5+C1)	2 oder 8 Bytes
Strings (z.B. P$+Q$)	Zeichenlänge + 2 Bytes

EINFACHES BEISPIELE:

```
100 BPRINT #1,A$,B
```
..... Schreibt den Inhalt der Variablen A$ und B in dieser Reihenfolge über Puffer 1 auf die Diskette. Der benötigte Speicherplatz beträgt (18 + 8) Bytes.

WICHTIGER HINWEIS

1. Vor Ausführung eines Schreibbefehls muß die gewünschte Datei mit der Anweisung OPEN eröffnet worden sein (Seite 334).
2. Bei der Aufzeichnung binärer Daten müssen keine Trennzeichen vorgesehen werden.

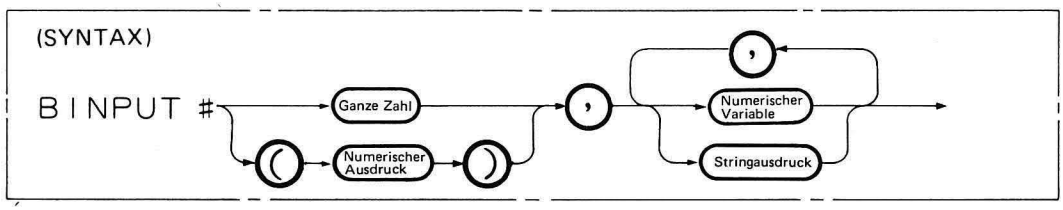
BINPUT

Liest Binärdaten aus einer sequentiellen Datei.

〈VERWENDUNG〉
① Verwendung binär aufgezeichneter Daten in einem Programm.
② Änderung von Binärdaten in einer sequentiellen Datei in Verbindung mit BPRINT#.

FORMAT: B I NPUT # n I (X), {N I A$}

n : Puffernummer (Integer—Variable oder Konstante)
X : Puffernummer (Numerischer Ausdruck)
N : Numerische Variable, der gelesene Daten zugewiesen werden sollen.
A$: Stringvariable, der gelesene Daten zugewiesen werden sollen.



ERLÄUTERUNG:

BINPUT # liest über den mit n festgelegten Puffer Daten aus einer sequentiellen Datei und weist sie der numerische Variablen N oder der Stringvariablen A$ zu. Die aufgezeichneten Daten müssen im Binärformat vorliegen.

EINFACHES BEISPIEL:

```
100 BINPUT #2,X$,Y
```
..... Liest über Puffer 2 binäre Daten von der Diskette und weist sie den Variablen X$ (Stringvariable) und Y (numerische Variable) zu.

WICHTIGER HINWEIS

1. Vor der Ausführung eines Lesebefehls muß die gewünschte Datei mit der Anweisung OPEN (Seite 334) eröffnet worden sein.
2. Aufgezeichnete binäre Daten einer kollektiven Array-Variablen (z.B. AP (*)) müssen beim Lesen einem Array gleichen Typs zugewiesen werden.

PRINT

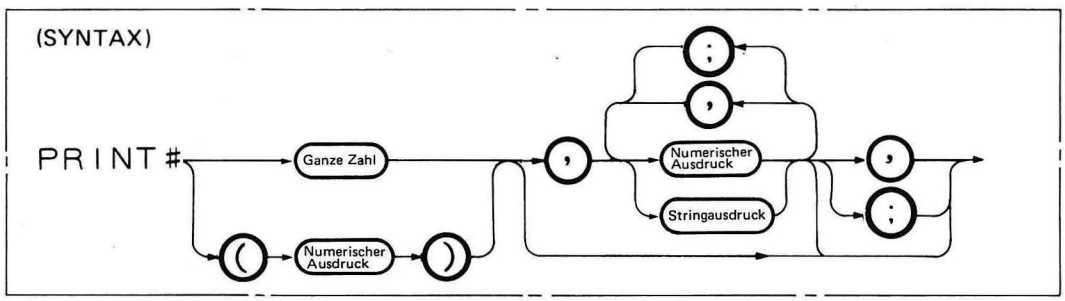
Schreibt Daten im ASCII-Format in eine sequentielle Datei

〈VERWENDUNG〉
① Aufbau einer sequentiellen Datei im ASCII-Format.
② Erneuern oder Ergänzen einer sequentiellen ASCII-Datei.

● **FUNKTION 1:** Schreibt Daten im ASCII-Format in eine sequentielle Datei.

FORMAT: PRINT # n | (X) [, {{Y | C}} [, | ;]]

n : Puffernummer (integerkonstante oder – variable)
X : Puffernummer (numerischer Ausdruck)
Y : Numerische Daten, die aufgezeichnet werden sollen
C : Stringdaten, die aufgezeichnet werden sollen



ERLÄUTERUNG:

1. PRINT # schreibt die durch den numerischen Ausdruck Y oder den String C festgelegen Daten über den mit n oder X spezifizierten Puffer in ASCII-Form auf die Diskette. Das Aufzeichnungsformat entspricht der PRINT-Anweisung (Seite 110).
2. Wurde mit der OPEN-Anweisung die Zugriffsfunktion "O" festgelegt, werden die Daten vom Dateianfang beginnend geschrieben. Schon vorhandene Daten werden dabei überschrieben. Wurde die Funktion "A" gewählt, beginnt der Schreibvorgang nach dem letzten vorhandenen Datum, d.h. neue Daten werden angefügt.

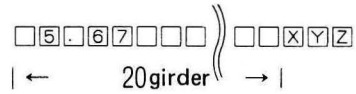
EINFACHES BEISPIEL:

1. `20 PRINT #1,12.34;"ABC"` ..... Die angegebenen Daten werden über Puffer 1 in folgendem Format aufgezeichnet:

*1) *2)
□1 2 . 3 4 □ A B C CR

*1) □ : Leerzeichen
*2) CR : Wagenrücklauf CR (&0D)

2. `20 PRINT #(2+3),5.67,"XYZ";` ·····Die angegebenen Daten werden über Puffer 5 in folgendem Format auf der Diskette aufgezeichnet:

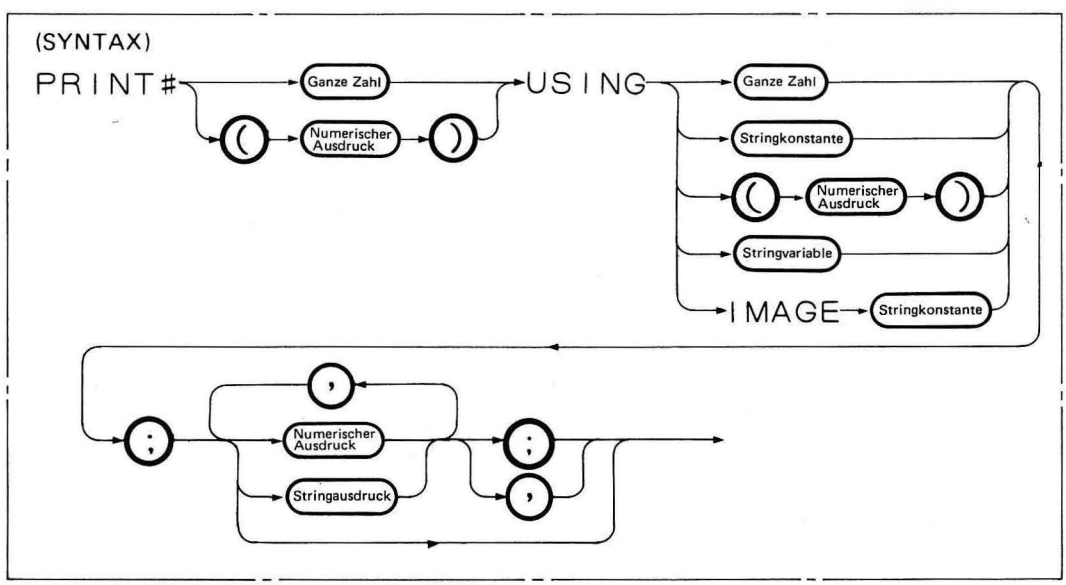


Man beachte die Funktion von Komma (,) und Semikolon (;).

● **FUNKTION 2:** Datenaufzeichnung mit Hilfe der USING-Anweisung (Seite 108).

FORMAT: PRINT # n | (X) USING m | S | (Z) | A$ | IMAGE T ; [{Y | C} [; | ,]]

n : Puffernummer
X : Puffernummer (numerischer Ausdruck)
m : Zeilennummer, die Zeile enthält die IMAGE-Anweisung
X : Zeilenmarke, die Zeile enthält die IMAGE-Anweisung
Z : Zeilennummer (numerischer Ausdruck)
A$: Formatanweisung (Seite 113)
T : Formatanweisung
Y : Numerische Daten, die aufgezeichnet werden sollen
C : Stringdaten, die aufgezeichnet werden sollen



ERLÄUTERUNG:

PRINT # schreibt spezifizierte numerische Daten Y oder Stringdaten C mit dem durch USING bestimmten Format auf die Diskette. Der zur Übertragung verwendete Puffer wird durch die Integerkonstante n oder den numerischen Ausdruck X (Dezimalstellen werden abgeschnitten) bestimmt. Die Ausgabe wird in der durch die der PRINT-Anweisung festgelegten Reihenfolge ausgeführt. Zu beachten ist dabei auch die IMAGE-Anweisung (Seite 113).

EINFACHES BEISPIELE:

1. `20 PRINT #1 USING 300;Q$` ..... Die durch die IMAGE-Anweisung in Zeile 300 formatierte Stringvariable Q$ wird über Puffer 1 auf die Diskette geschrieben.

2. `20 PRINT #(3+4) USING P$;Q$,K`
..... Die Inhalte der Stringvariablen Q$ und der numerischen Variablen K werden über Puffer (3+2) auf die Diskette geschrieben. Die Formatierung erfolgt mit Hilfe der Variablen P$, die Daten entsprechend der IMAGE-Anweisung enthält.

WICHTIGER HINWEIS

1. Vor Ausführung einer PRINT # – Anweisung muß die gewünschte Datei eröffnet worden sein (Seite 334).
2. Da die Aufzeichnung in ASCII-Kode erfolgt, muß beim Lesen einer solchen Datei auf Trennzeichen Rücksicht genommen werden. Eine genaue Erläuterung von Trennzeichen und ihren Eigenschaften erfolgt bei der Beschreibung der INPUT-Anweisung (Seite 117).

ANWENDUNGSBEISPIEL:

Programm A

```
 10 MAXFILE 1
 20 OPEN "O",1,"AZ:A0"
 30 A$="ABC"
 40 PRINT #1 USING 500;A$
500 IMAGE "5A"
600 END
```

Die Daten werden über Puffer 1 in die Datei AZ: A0 geschrieben. Sie haben folgendes Format:

A B C □ □ CR

Programm B

```
 10 MAXFILE 2
 20 OPEN "O",2,"AZ:A1"
 30 A$="5A 2X $#,###"
 40 B$="ABC":K=1320
 50 PRINT #2 USING A$;B$;K
100 END
```

Die Daten werden über Puffer 2 in die Datei AZ: A1 geschrieben. Sie haben folgendes Format:

A B C □ □ □ □ $ 1 . 3 2 0 CR

Weitere Beispiele befinden sich auf Seite 336.

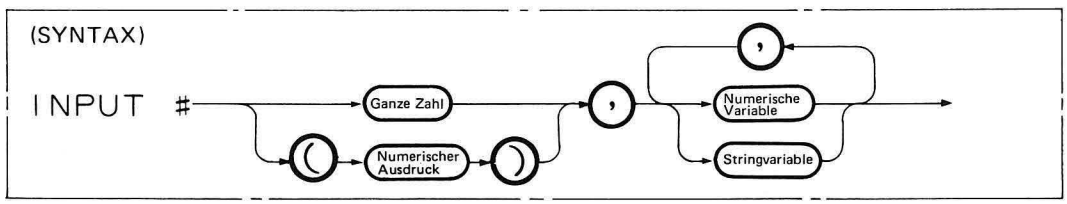
INPUT

Liest Daten im ASCII-Format aus einer sequentiellen Datei.

〈VERSENDUNG〉
① Um die Daten einer sequentiellen Datei in einem Programm verwenden zu können.
② Um Daten einer sequentiellen Datei im ASCII-Format zu ändern.

FORMAT: INPUT # n | (X), {N | A$}

n : Puffernummern(Integerkonstante)
X : Puffernummer (numerischer Ausdruck, Dezimalstellen werden abgeschnitten)
N : Numerische Variable, der die gelesenen Daten zugewiesen werden sollen.
A$: Stringvariable, der die gelesenen Daten zugewiesen werden sollen.



ERLÄUTERUNG:

1. INPUT # liest Daten einer sequentiellen Datei über einen durch n oder X spezifiziert Puffer und weist sie der numerischen Variablen N oder der Stringvariablen A$ zu. Die Datei muß Daten in ASCII-Format erhalten.
2. Zur Unterscheidung und Trennung der aufgezeichneten Daten werden Trennzeichen (engl. separator, delimiter) verwendet. Werden beim Lesevorgang solche Trennzeichen erkannt, endet die Zuweisung zur aktuellen Variablen. Als Trennzeichen werden [CR] [LF] *1) oder EOD*2) angesehen. Bei numerischen Konstanten werden auch Leerzeichen als Trennzeichen angesehen. Bei Ausführung der PRINT-Anweisung ist deshalb auf richtige Verwendung der Trennzeichen zu achten.

EINFACHES BEISPIEL:

```
10 INPUT #1,A$,B
```

Werden Daten wie in folgendem Bild über Puffer 1 von der Diskette gelesen, erhält die Variable A$ den String SHARP zugewiesen, die Variable B die Zahl −78.9.

[S][H][A][R][P][CR]−[7][8].[9][ ][CR]

↑ Trennzeichen ↑ Trennzeichen (Leerzeichen)

*1) [LF] : Zeilenvorschub (&0A)
*2) EOD : Zeigt das Ende einer Datenaufzeichnung an. Wird EOD gelesen, tritt Fehler 209 auf. Zur Verwendung von EOD als Trennzeichen kann die Fehlerbedingung mit ON ERROR erkannt und nach Test auf Fehler 209 beseitigt werden.

WICHTIGER HINWEIS

1. Vor einem Dateizugriff mit der INPUT #-Anweisung muß die gewünschte Datei eröffnet worden sein (Seite 345).
2. Normalerweise werden [CR] , [LF] und EOD als Trennzeichen für Stringkonstanten verwendet, [ ] , [CR] , [LF] und EOD für numerische Konstanten. Zur Trennung von Stringkonstanten kann auch die SEP #-Anweisung verwendet werden (Seite 347).
3. Anwendungsprogramme befinden sich auf Seite 336.

SEP

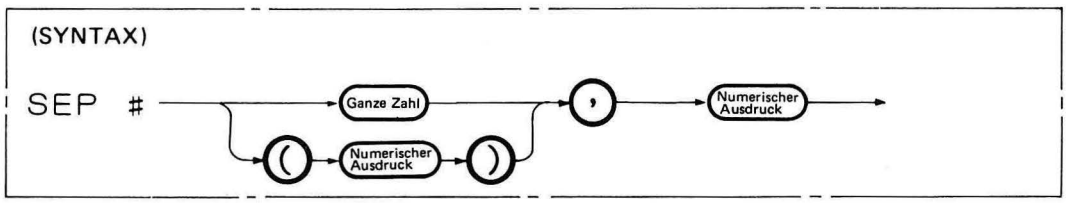
Bestimmt Trennzeichen bei Lesen sequentieller Daten in ASCII-Format.

FORMAT: SEP # n | (X), Y

n : Puffernummer (Integerkonstante)

X : Puffernummer (numerischer Ausdruck, Dezimalstellen werden abgeschnitten)

Y : Zeichen gemäß ASCII-Kode



ERLÄUTERUNG:

1. SEP # legt fest, welche Trennzeichen bei der Zuweisung von Variablen ausgewertet werden sollen. Die zu prüfenden Daten werden über den durch die Integerkonstante n oder den numerischen Ausdruck X definierten Puffer gelesen. Als Trennzeichen wird das mit Y definierte ASCII-Zeichen und EOD verwendet. Eine Tabelle gültiger ASCII-Zeichen befindet sich im Benutzerhandbuch, Anhang 1.
2. Liste der verwendbaren Trennzeichen:

Datentypen	Trennzeichen	
Numerische Konstanten	□, CR, LF, und EOD (SEP # wird ignoriert)	
Stringkonstanten	Wenn durch SEP nicht anders definiert	CR, LF, EOD
	Durch SEP-Anweisung bestimmtes Zeichen	Festgelegtes Zeichen, EOD

EINFACHES BEISPIELE:

1. `10 SEP #1,&53` ..... Verwendet beim Lesen über Puffer 1 das ASCII-Zeichen &53 ("S") und EOD als Trennzeichen.

2. `10 SEP #1,83` ..... Verwendet beim Lesen über Puffer 1 das ASCII-Zeichen 83 dex ("S") und EOD als Trennzeichen.

WICHTIGER HINWEIS

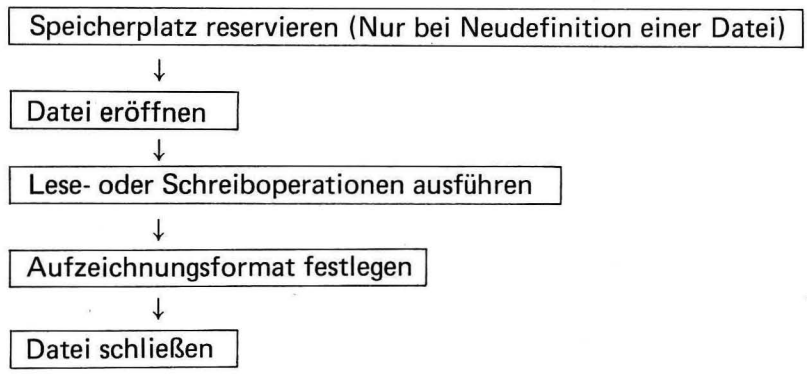
Ein Kennsatzname kann an diese Anweisung nicht angehängt werden. Ferner kann keine Mehrfachanweisung vor dieser Anweisung gegeben werden.

3-2. DIREKTZUGRIFFSDATEIEN

Dieser Abschnitt beschreibt Anweisungen, die zum direkten Lesen und Schreiben einer Randomdatei vorgesehen sind:

RFORMAT #
PUT #
GET #

Diese Anweisungen können auch in der manuellen Betriebsart verwendet werden.



Speicherplatz muß nur beim Neuaufbau einer Datei reserviert werden. Dazu wird die Anweisung CREATE (Seite 332) verwendet. Dieser Schritt ist nicht notwendig, wenn auf eine schon vorhandene Datei zugegriffen werden soll. Folgendes Beispiel soll die nötigen Schritte noch einmal verdeutlichen:

a) Schreiben in eine Randomdatei

```
 10 MAXFILE 1 ..................................... Pufferzahl festlegen
 20 INPUT "Wieviel Teile? ";N
 30 CREATE "LAGER:A0",N,40 ..... Speicherplatz reservieren
 40 OPEN "R",1,"LAGER:A0" ...... Datei eröffnen
 50 RFORMAT #1,C%,H$,T,Z ....... Aufzeichnungsformat festlegen
 60 DISP CLEAR
 70 DISP "Lagernummer:   "
 80 DISP "Name:          "
 90 DISP "Preis:         "
100 DISP "Anzahl:        "
110 CURSOR 17,0:INPUT N
120 IF N=0 THEN "CLOSE"
130 CURSOR 17,1:INPUT H$
140 CURSOR 17,2:INPUT T
150 CURSOR 17,3:INPUT Z
160 CURSOR 17,5:INPUT "Alles IN Ordnung (J/N)? ";A$
170 IF A$="J" THEN "PUT"
180 IF A$="N" THEN 130:ELSE 160
190*"PUT":C%=N:PUT #1,C% ...... Schreibbefehl ausführen
200 GO TO 60
210*"CLOSE":CLOSE 1 ........... Datei schließen
220 DISP CLEAR :DISP "Aufzeichnung beendet":END
```

b) Lesen einer Randomdatei

```
 10 MAXFILE 1 ..... Pufferzahl festlegen
 20 OPEN "R",1,"LAGER:A0" ..... Datei eröffnen
 30 RFORMAT #1,C%,H$,T,Z ..... Aufzeichnungsformat bestimmen
 40 DISP CLEAR
 50 DISP "Lagernummer: "
 60 DISP "Name:        "
 70 DISP "Preis:       "
 80 DISP "Anzahl:      "
 90 CURSOR 17,0:INPUT N
100 IF N=0 THEN CLOSE
110 GET #1,N ..... Lesebefehl ausführen
120 CURSOR 17,1:DISP H$
130 CURSOR 17,2:DISP T
140 CURSOR 17,3:DISP Z
150 CURSOR 17,5:INPUT "Weiter (J/N)? ";A$
160 IF A$="J" THEN 90
170 IF A$="N" THEN "CLOSE":ELSE 150
180*"CLOSE":CLOSE 1 ..... Datei schließen
190 CURSOR 0,6:END
```

c) Schreiben und Lesen einer Randomdatei

```
 10 MAXFILE 1 ..... Pufferzahl festlegen
 20 DIM J$(1)*40
 30 OPEN "R",1,"LISTE:A1" ..... Datei eröffnen
 40 RFORMAT #1,C%,N$,T$,J$(1)... Aufzeichnungsformat festlegen
 50 DISP CLEAR
 60 DISP "Nummer:    "
 70 DISP "Name:      "
 80 DISP "Telefon:   "
 90 DISP "Adresse:   "
100 CURSOR 10,0:INPUT N
110 IF C%=0 THEN 260
120 GET #1,N ..... Lesebefehl ausführen
130 IF C%=0 THEN 200
140 CURSOR 10,1:DISP N$
150 CURSOR 10,2:DISP T$
160 CURSOR 10,3:DISP J$(1)
170 CURSOR 10,5:INPUT "Alles IN Ordnung (J/N)? ";A$
180 IF A$="J" THEN 240
190 IF A$="N":ELSE 170
200 CURSOR 10,1:INPUT N$
210 CURSOR 10,2:INPUT T$
220 CURSOR 10,3:INPUT J$(I)
230 GO TO 170
240 C%=N:PUT #1,C% ..... Schreibbefehl ausführen
250 GO TO 50
260 CLOSE 1 ..... Datei schließen
270 CURSOR 0,6:END
```

In diesem Beispielprogramm werden an der vorhandenen Datei Änderungen und Ergänzungen vorgenommen.

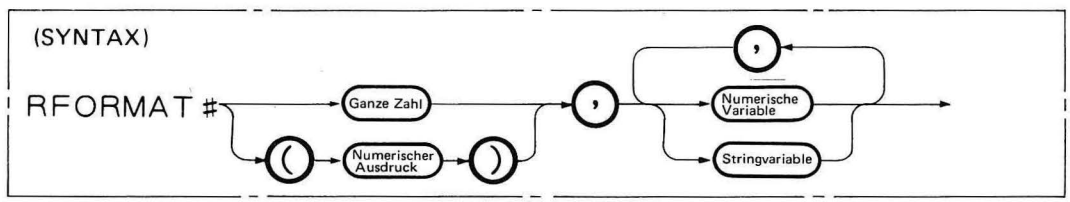
RFORMAT

Legt die Variablenzusammensetzung für Lese- und Schreibvorgänge fest.

〈VERWENDUNG〉
Lesen und Schreiben einer Randomdatei

FORMAT: RFORMAT # n | (X), {N | A$}

n : Puffernummer (Integerkonstante)
X : Puffernummer (numerischer Ausdruck, Dezimalstellen werden abgeschnitten)
N : Numerische Variablen
A$: Stringvariable



ERLÄUTERUNG:

RFORMAT # legt durch Angabe von numerischen Variablen N und Stringvariablen A$ einen Datensatz (Record) fest, deren Inhalt bei Zugriffen auf einen Randomdatei gelesen oder geschrieben wird. Dadurch ist bei Lese- oder Schreibanweisungen nur noch die Angabe der Position innerhalb der Datei nötig, um einen ganzen Datensatz zu bearbeiten. Die Festlegung bezieht sich nur auf den mit n oder X festgelegten Puffer.

EINFACHES BEISPIELE:

1. `100 RFORMAT #3,C%,N$,A$(1),B`
   ..... Der Zugriff auf die über Puffer 3 zugewiesene Datei soll in Form der Variablengruppe C%, N$, A$ (1), B erfolgen.
2. `200 RFORMAT #1,R(*)` ..... Der Zugriff auf die über Puffer 1 zugewiesene Datei soll in Form eines Arrays (R (*)) erfolgen.

WICHTIGER HINWEIS

1. Diese Anweisung muß vor Verwendung der Anweisungen GET # und PUT # ausgeführt werden.
2. Eine kollektive Array-Variable (z.B. X(*) und Y$ (*)) kann dazu verwendet werden, ein ganzes Feld auf einmal zu lesen.
3. Ein Kennsatzname kann an diese Anweisung nicht angehängt werden. Ferner kann keine Mehrfachanweisung vor und nach dieser Anweisung gegeben werden.

ANWENDUNGSBEISPIEL:

Beispiele befinden sich auf Seite 348.

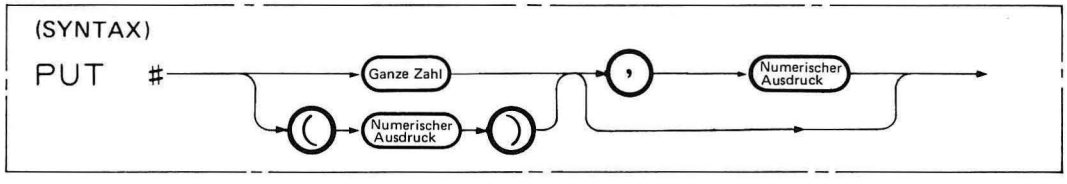
PUT

Schreibt Variablen in eine Randomdatei

〈VERWENDUNG〉
① Aufbau einer Randomdatei
② Erweitern und Ändern einer Randomdatei

FORMAT: PUT # n | (X) [,Y]

n : Puffernummer (Integerkonstante)
X : Puffernummer (numerischer Ausdruck, Dezimalstellen werden abgeschnitten)
Y : Satznummer



ERLÄUTERUNG:

1. PUT # trägt einen Satz mit der Nummer Y über den durch n bzw. X festgelegten Puffer in eine Randomdatei ein. Die im Datensatz enthaltenen Variablen werden mit der Anweisung RFORMAT # festgelegt (Seite 350).
2. Wird Y weggelassen, nimmt die Satznummer bei der Eröffnung der Datei den Wert 1 an. Sie wird dann bei jeder Ausführung der Anweisungen PUT # oder GET # (Seite 346) um 1 inkrementiert.

EINFACHES BEISPIELE:

1. `100 PUT #3,5` ..... Trägt einen mit RFORMAT # definierten Record an 5. Stelle (Recordnummer 5) auf der Diskette im reservierten Speicherbereich ein. Der Vorgang läuft über Puffer 3 ab.
2. Die Inhalte der Variablen A%, B$ und C(1) sollen über Puffer 3 in die Positionen 5 und 6 (Recordnummern 5 und 6) des reservierten Diskettenspeicherbereichs eingetragen werden:

```
  ⋮
100 RFORMAT #3,A%,B$,C(1)
110 PUT #3,5
120 PUT #3
  ⋮
```

WICHTIGER HINWEIS

Vor Ausführung einer Schreibanweisung (PUT #) ist die Zuweisung eines Speicherbereichs (Seite 332) und die Eröffnung einer Randomdatei (Seite 334) notwendig.

ANWENDUNGSBEISPIEL:

Ein Anwendungsbeispiel ist auf Seite 348 angeführt.

GET

Liest Daten aus einer Randomdatei.

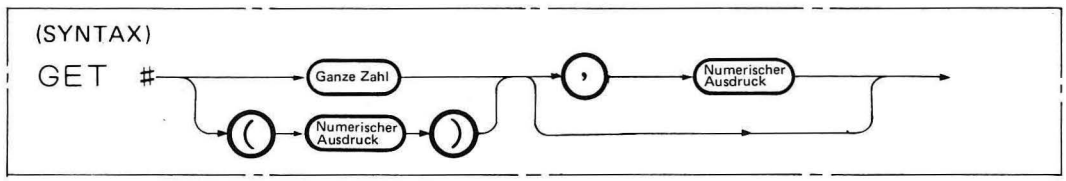
〈VERWENDUNG〉
Lesen aus Randomdateien innerhalb eines Programms

FORMAT: GET # n | (X) [,Y]

n : Puffernummer (Integerkonstante)

X : Puffernummer (numerischer Ausdruck, Dezimalstellen werden abgeschnitten)

Y : Satznummer



ERLÄUTERUNG:

1. Die durch die Satznummer Y festgelegte Datengruppe wird über Puffer n bzw. X gelesen und den mit der RFORMAT-Anweisung (Seite 350) definierten Variablen zugewiesen.
2. Wird Y nicht spezifiziert, nimmt die Satznummer im Moment der Dateieröffnung den Wert 1 an. Sie wird danach bei jeder Ausführung der Anweisungen GET # oder PUT # (Seite 352) um den Wert 1 inkrementiert.

EINFACHES BEISPIELE:

1. `100 GET #3,5` ..... Liest die in Satz 5 gespeicherten Daten über Puffer 3 von der Diskette und weist sie den mit RFORMAT definierte Variablen zu.
2. Den Variablen A%, B$, C(1) sollen die in Satz Nummer 6 vorliegenden Daten zugewiesen werden. Zur Übertragung soll Puffer 3 verwendet werden.

```
    ⋮
100 RFORMAT #3,A%,B$,C(1)
110 PUT #3,5
120 GET #3
    ⋮
```

WICHTIGER HINWEIS

Vor Ausführung einer Schreibanweisung muß ein Speicherbereich für die gewünschte Randomdatei reserviert werden (Seite 334).

ANWENDUNGSBEISPIEL:

Ein Anwendungsbeispiel in auf Seite 348 aufgeführt.

4. PROGRAMMDATEIEN

Dieser Abschnitt behandelt Anweisungen, die zum Aufzeichnen und Lesen von Programmdateien vorgesehen sind:

SAVE ※
STORE ※
LOAD ※
CHAIN ※
MERGE
LOAD SUB ※
ALOAD ※

Alle mit ※ bezeichneten Anweisungen können auch innerhalb eines Programms verwendet werden.

SAVE

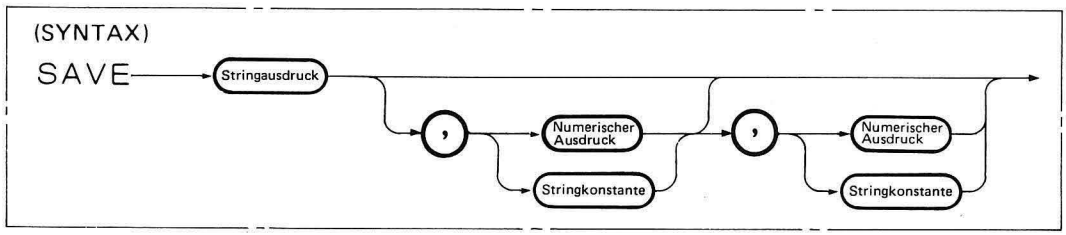
Speichert ein Programm in BASIC-Zwischenkode *) auf Diskette.

〈VERWENDUNG〉
Zur Speicherung eines im Hauptspeicher vorliegenden Programms. Die Aufzeichnung erfolgt in BASIC-Zwischenkode.

● **FUNKTION 1:** Aufzeichnung eines Programms in ungeschützter Form.

FORMAT: SAVE C [, X | S [, Y | T]]

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")
X : Erste Zeilennumer des gewüschten Programmbereichs
S : Anfangsmarke des gewünschten Programmbereichs
Y : Letzte Zeilennummer des gewünschten Programmbereichs
T : Endmarke des gewünschten Programmbereichs



ERLÄUTERUNG:

1. SAVE speichert den zwischen Zeilennummer X und Y im Hauptspeicher vorliegenden Programmteil in BASIC-Zwischenkode unter der Dateikennung C auf Diskette. Anstelle der Zeilennummern können auch Marken (S und T) verwendet werden.
2. Wird die letzte Zeilennummer oder die Endmarke nicht angegeben, werden, beginnend mit der durch X bzw S festgelegten Zeile, allen folgenden Programmzeilen in BASIC-Zwischencode aufgezeichnet.
3. Sind weder Anfangs- noch Endzeile angegeben, wird das gesamte Programm in BASIC-Zwischenkode aufgezeichnet.

EINFACHES BEISPIELE:

1. `SAVE "RADIO"` [ENTER]
   ..... Speichert ein im Hauptspeicher befindliches Programm unter dem Namen RADIO in Zwischenkode auf einer im Laufwerk A0 befindlichen Diskette ab.
2. `SAVE "PROBE,SHARP"` [ENTER]
   ..... Speichert ein im Hauptspeicher befindliches Programm unter dem Namen PROBE auf einer Diskette mit der Bezeichnung SHARP in Zwischenkode ab.

*) BASIC-Zwischencode: Befehle und Anweisungen werden nicht in ASCII-Kode aufgezeichnet, sondern in Form einzelner Zahlen, die die Befehle und Anweisungen symbolisieren. Dadurch kann erheblich Speicherplatz gespart werden.

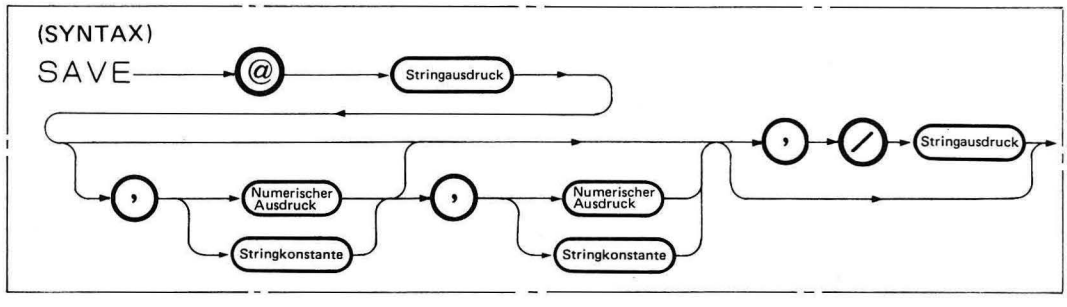
3. SAVE "MEMORY:A1",100,200 ENTER

..... Speichert einen zwischen den Zeilen 100 und 200 liegenden Programmteil in Zwischenkode unter dem Namen MEMORY auf eine Diskette in Laufwerk A1 ab.

● **FUNKTION 2:** Aufzeichnung eines Programms in geschützter Form.

FORMAT: SAVE @ C [, X | S [, Y | T]] [, /D]

@ : Anweisung zur geschützten Aufzeichnung
C : Dateikennung ("Dateiname I, Diskettenname I: Kanal-Laufwerk-Nummer")
X : Anfangszeilennummer des gewünschten Programmteils
S : Anfangsmarke des gewünschten Programmteils
Y : Letzte Zeilennummer des gewünschten Programmteils
T : Endmarke des gewünschten Programmteils
D : Paßwort



ERLÄUTERUNG:

1. Die Eingabe des @-Zeichens vor der Dateikennung führt zur Erstellung einer geschützten Programmaufzeichnung. Die Funktion der übrigen Parameter entspricht Funktion 1.
2. Ein auf diese Weise gesichertes Programm kann weder aufgelistet noch verändert werden. Ebenso können PEEK- und die USR-Funktion (Seite 29 bzw. 284) nicht mehr verwendet werden. Der Sicherungszustand kann nicht mehr aufgehoben werden.
3. Nach Angabe eines in D bestimmten Schlüsselworts kann das Programm wieder aufgelistet werden. Alle anderen Funktionen sind weiterhin nicht ausführbar. Der LIST-Befehl wird auf Seite 85 genauer beschrieben.

EINFACHES BEISPIELE:

1. SAVE @"START:A0","SCHRITT" ENTER

..... Speichert einen mit der Marke SCHRITT beginnenden Programmteil in geschützter Form auf einer Diskette im Laufwerk A0 ab.

2. SAVE @"START:A0","SCHRITT" ENTER

..... Speichert einen mit Zeile 100 beginnenden Programmteil in geschützter Form auf einer Diskette im Laufwerk A0 ab. Das Programm kann später nach Angabe des Schlüsselworts KEY aufgelistet werden.

<HILFREICHER HINWEIS>
Mit Hilfe des Befehls CAT (Seite 320) kann der Sicherungsstatus eines auf Diskette aufgezeichneten Programms festgestellt werden.

STORE

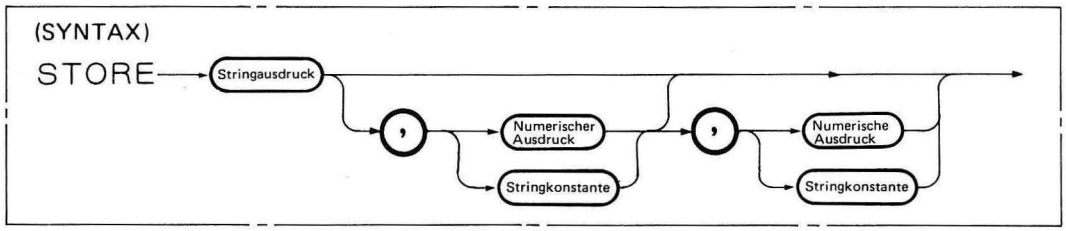
Speichert ein Programm im ASCII-Format auf Diskette ab.

〈VERWENDUNG〉
Zur Aufzeichnung eines im Speicher befindlichen Progamms in ASCII-Format.

● **FUNKTION 1:** Zeichnet ein Programm in ungeschützter Form auf.

FORMAT: STORE C [, X | S [, Y | T]]

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")
X : Erste Zeile des gewünschten Programmteils
S : Anfangsmarke des gewünschten Programmteils
Y : Letzte Zeile des gewünschten Programmteils
T : Endmarke des gewünschten Programmteils



ERLÄUTERUNG:

1. STORE zeichnet den zwischen Zeile X und Y (bzw. Marke S und T) befindlichen Programmteil im ASCII-Format mit der durch C festgelegten Dateikennung auf Diskette auf.
2. Entfällt die Angabe der letzten Zeile (also Y bzw. T), werden alle nach X oder S folgenden Zeilen aufgezeichnet.
3. Werden weder die erste noch die letzte Zeile spezifiziert, wird das gesamte Programm aufgezeichnet.

EINFACHES BEISPIELE:

1. `STORE "PROBE,SHARP:A1"` ENTER
   ..... Ein im Speicher befindliches Programm wird in ASCII-Kode unter dem Namen PROBE auf eine Diskette mit dem Namen SHARP aufgezeichnet.
2. `STORE "PROBE",10,500` ENTER
   ..... Zeichnet einen mit PROBE bezeichneten Programmteil (Zeile 10 bis 500) auf eine Diskette im Laufwerk A0 im ASCII-Kode.

● **FUNKTION 2:** Zeichnet ein Programm in geschützter Form auf.

FORMAT: STORE @ C [, X | S [, Y | T]] [, /D]

@ : Anweisung zur geschützten Aufzeichnung

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")

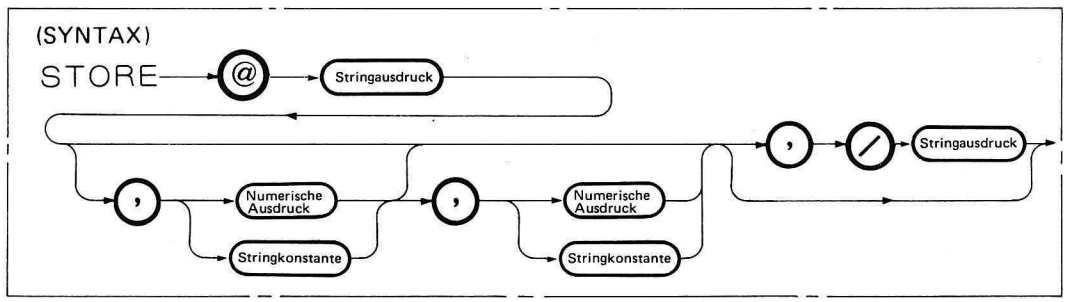
X : Erste Zeilennummer des gewünschten Programmbereichs

S : Anfangsmarke des gewünschten Programmbereichs

Y : Letzte Zeilennummer des gewünschten Programmbereichs

T : Endmarke des gewünschten Programmbereichs

D : Paßwort



ERLÄUTERUNG:

1. Die Eingabe des @-Symbols vor der Dateikennung bewirkt die Aufzeichnung eines geschützten Programms in ASCII-Format. Alle anderen Parameter verhalten sich entsprechend Funktion 1.
2. Die Programmschutzfunktion wird in der Beschreibung der Anweisung SAVE, Funktion 2, beschrieben.

EINFACHES BEISPIELE:

1. STORE @"START",10,200 [ENTER]
   ..... Zeichnet die Programmzeilen 10 bis 200 als geschützte Datei unter dem Namen START auf eine Diskette im Laufwerk A0 auf.
2. STORE @"START:A1",/"J2" [ENTER]
   ..... Ein im Speicher vorliegendes Programm wird unter dem Namen START als geschützte Datei auf einer Diskette im Laufwerk A1 gespeichert. Durch Eingabe des Schlüsselworts J2 kann es später wieder aufgelistet werden.

<HILFREICHER HINWEIS>
Mit Hilfe des CAT-Befehles (Seite 320) kann der Sicherungstatus einer Programmdatei festgestellt werden.

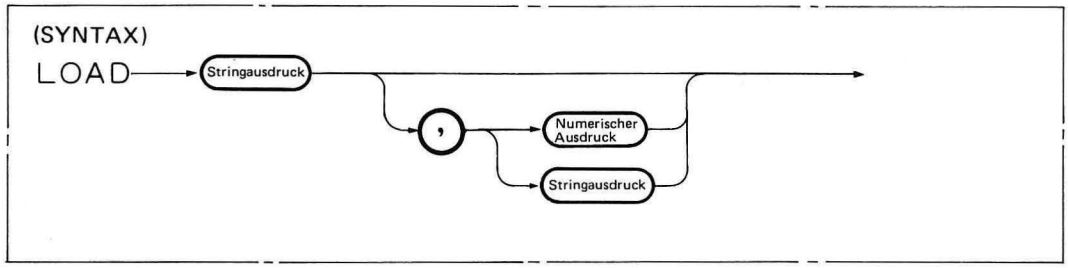
LOAD

Lädt ein Programm von der Diskette in den Speicher.

〈VERWENDUNG〉
Laden eines Programms von Diskette in den Hauptspeicher. Das Programm kann dann ausgeführt, getestet, modifiziert oder ergänzt werden.

FORMAT: LOAD C [, X | S]

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")
X : Zeilennumer, Programmausführung beginnt hier
S : Zeilenmarke, Programmausführung beginnt hier



ERLÄUTERUNG:

1. LOAD lädt ein durch die Dateikennung C festgelegtes Programm (ASCII- oder Zwischenkodeformat) von einer Diskette in den Hauptspeicher und beginnt die Programmausführung mit der Zeilennummer X oder der Marke S.
2. Werden die Optionen X bzw. S nicht angegeben, wird das Programm in der manuellen Betriebsart nur geladen, bei Ausführung des Befehls in einem Programm beginnt die Programmausführung nach dem Ladevorgang mit der niedrigsten Zeilennummer.
3. Im ASCII-Format aufgezeichnete Programme werden beim Ladevorgang in Zwischenkode übersetzt und erst dann im Hauptspeicher abgelegt.

EINFACHES BEISPIELE:

1. `LOAD "PROBE"` ENTER
   ..... Lädt das Programm PROBE von einer Diskette in Laufwerk A0 in den Hauptspeicher. Das Programm wird nicht ausgeführt.
2. `LOAD "START:A0","INIT"` ENTER
   ..... Lädt das Programm START auf einer Diskette in Laufwerk A0 in den Hauptspeicher und beginnt die Programmausführung mit der durch die Marke INIT definierten Zeile.
3. `100 LOAD "SUBPRO"`
   ..... Lädt das Programm SUBPRO von Laufwerk A0 in den Speicher und beginnt die Programmausführung mit der niedrigsten Zeilennummer.

HINWEIS

1. Die Ausführung dieser Anweisung löscht das gesamte Programm und alle im Speicher befindlichen Variablen. Bei Verwendung der Anweisung im Programm werden alle Dateien automatisch geschlossen (Seite 336).
2. Wird eine nicht gesicherte Programmdatei von einem gesicherten Programm aus geladen, wird den Sicherungsstatus auch auf dieses Programm ausgedehnt. Um dies zu vermeiden, muß vor Ausführung der LOAD-Anweisung ein NEW-Befehl gegeben werden.

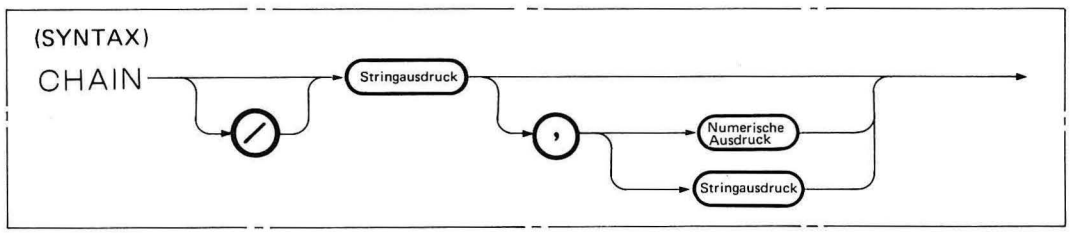
CHAIN

Fügt ein im Speicher vorliegendes Programm mit einem auf Diskette gespeicherten Programm zusammen und führt das so entstandene Programm aus.

〈VERWENDUNG〉
Zur Ausführung eines Unterprogramms auf Diskette in Verbindung mit dem im Speicher befindlichen Programm.

FORMAT: CHAIN [/] C [, X | S]

/ : Offene Dateien werden nicht geschlossen
C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")
X : Zeilennummer, das Programm wird hier fortgeführt
S : Marke, das Programm wird hier fortgeführt



ERLÄUTERUNG:

1. Liest ein durch die Dateikennung C festgelegtes Programm (ASCII- oder oder Zwischenkode) und startet es mit der durch X oder S bestimmten Zeilennummer.
2. Mit der Ausführung dieses Programms werden alle Zeilen des aufrufenden Programms gelöscht, die über der niedrigsten Zeile des durch CHAIN aufgerufenen Programms liegen.
   Alle Variablen sowie die Programmzeilen, die unter der niedrigsten Zeile des geladenen Programms liegen, bleiben erhalten.
3. Wird das / -Zeichen weggelassen, werden bei Ausführung der CHAIN-Anweisung alle offenen Dateien automatisch geschlossen.
4. Ist [, X | S] nicht festgelegt, wird bei Ausführung der CHAIN-Anweisung in der manuellen Betriebsart nur das Programm geladen; innerhalb eines Programms beginnt die Ausführung mit der niedrigsten Zeile.

EINFACHES BEISPIEL:

```
500 CHAIN "SUBPRO",10
```

..... Liest ein Programm mit dem Namen SUBPRO von der Diskette und setzt die Programmausführung mit Zeile 10 fort. Dabei werden alle Zeilen des aufrufenden Programms gelöscht, die im Bereich des überlagernden Programms liegen.

ANWENDUNGSBEISPIEL:

Wir nehmen an, folgendes Programm "A" befinde sich im Speicher:

```
10 DISP TAB 10,"*****************************************"
20 DISP TAB 10,"* BERECHNUNG VON KREISUMFANG UND -FLACHE *"
30 DISP TAB 10,"*****************************************"
40 DISP
50 A=20
60 B=100
70 A$="Flache"
80 END
```

```
 50 INPUT "                    Radius:";R
 60 DISP
 70 DISP "                   Umfang=";2*PI*R
 80 DISP "                   Flache=";PI*R^2
 90 DISP
100 GO TO 50
110*"ENDE":DISP "Ende der Berechnungen"
               :END
```

Auf der Diskette im Laufwerk A1 befinde sich ein Programm "B" mit nebenstehenden Zeilen:

Auf der Diskette im Laufwerk A1 befinde sich ein Programm "B" mit folgenden Zeilen:

CHAIN /"B:A1" [ENTER] ..... Lädt Programm B von Laufwerk A1 in den Speicher.

Dadurch wird der Speicherinhalt wie folgt abgeändert:

```
 10 DISP TAB 10,"*****************************************"
 20 DISP TAB 10,"* BERECHNUNG VON KREISUMFANG UND -FLACHE *"
 30 DISP TAB 10,"*****************************************"
 40 DISP
 50 INPUT "                    Radius:";R
 60 DISP
 70 DISP "                   Umfang=";2*PI*R
 80 DISP "                   Flache=";PI*R^2
 90 DISP
100 GO TO 50
110*"ENDE":DISP "Ende der Berechnungen":END
```

Programm "A" wurde also von Programm "B" ab Zeile 50 überschrieben.

<PRAKTISCHER TIP>

Wird diese Anweisung innerhalb eines Unterprogramms ausgeführt, bleibt die Rückkehradresse gespeichert und die Programmkontrolle wird nach Abarbeitung des Unterprogramms an diese Adresse zurückgegeben.

MERGE

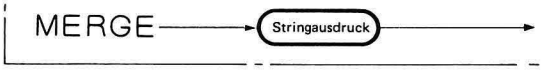
Mischt ein von der Diskette gelesenes Programm mit einem im Speicher vorhandenen Programm.

〈VERWENDUNG〉
Zur Mischung zweier Programme, von denen eines im Speicher und eines auf Diskette vorliegt. Damit kann ein neues Programm erzeugt werden.

FORMAT: MERGE C

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")

(SYNTAX)



ERLÄUTERUNG:

1. MERGE liest ein durch die Dateikennung C festgelegtes Programm von der Diskette (Zwischenkode oder ASCII-Format) und mischt es mit einem im Speicher vorhandenen Programm. Alle Zeilennummern bleiben erhalten, sollte jedoch Zeilen mit gleichen Zeilennummern vorkommen, wird der Inhalt dieser Zeile durch den Zeileninhalt des überlagernden Programms ersetzt.
2. Die Ausführung dieser Anweisung löscht alle in Speicher befindlichen Variablen.

EINFACHES BEISPIEL:

```
MERGE "CALCU" ENTER
```

..... Liest das Progamm CALCU von Diskette und mischt es mit einem im Speicher vorliegenden Programm.

HINWEIS

1. Diese Anweisung kann nicht innerhalb eines Programms verwendet werden.
2. Wird diese Anweisung ausgeführt, ohne daß sich ein Programm im Speicher befindet, wird nur das mit C festgelegte Programm geladen.

ANWENDUNGSBEISPIEL:

Im Speicher soll sich folgendes Programm "A" befinden:

```
 50 DISP TAB 10," 1        TEST"
 55 DISP
 60 DISP TAB 10," 2        MUSIC"
 65 DISP
 70 DISP TAB 10," 3        SPIEL"
 75 DISP
 80 INPUT A
 90 IF A=1 THEN 200
100 END
```

ANWENDUNGSBEISPIEL:

Folgendes Programm soll sich unter dem Namen PLUS in Laufwerk A0 befinden:

```
 80 DISP TAB 10," 1        TEST"
 90 INPUT A
100 IF A=1 THEN 400
110 IF A=2 THEN 1000
120 IF A=3 THEN 1500
130 DISP "          FEHLER!!!":DISP TAB 10,
      "Richtige Nummer eingeben!"
135 GO TO 90
140 END
```

Damit sind die Ausgangsbedingungen geschaffen: Ein Programm im Speicher und ein zweites in Laufwerk A0.

MERGE "ADDITION" [ENTER]

..... Lädt das Programm PLUS von Laufwerk A0 und mischt es mit dem im Speicher befindlichen Programm (Programm A).

Ergebnis:

```
 50 DISP TAB 10," 1        TEST"
 55 DISP
 60 DISP TAB 10," 2        MUSIC"
 65 DISP
 70 DISP TAB 10," 3        SPIEL"
 75 DISP
 80 DISP TAB 10," 1        TEST"
 90 INPUT A
100 IF A=1 THEN 400
110 IF A=2 THEN 1000
120 IF A=3 THEN 1500
130 DISP "          FEHLER!!!":DISP TAB 10,
      "Richtige Nummer eingeben!"
135 GO TO 90
140 END
```

Wie oben gezeigt, wurde ein Programm von Diskette geladen und mit dem im Speicher befindlichen Programm gemischt. Da die Zeilen 80, 90 und 100 in beiden Programmen vorkommen, werden diese Zeilen durch das Programm PLUS überschrieben.

<PRAKTISCHER TIP>

Das Problem sich überlagernder Programmzeilen kann durch sinnvolle Wahl der Zeilennummerierung umgangen werden. Sollen zwei Programme vollständig miteinander gemischt werden, müssen sie notfalls mit dem Befehl REN (Seite 98) neu numeriert werden, bevor die MERGE-Anweisung ausgeführt wird.

LOAD SUB

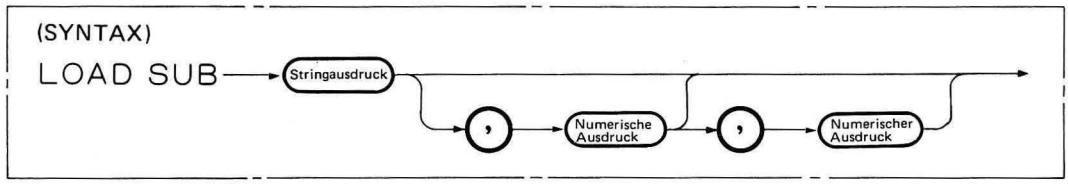
Lädt ein Programm von der Diskette

〈VERWENDUNG〉
Laden eines Programms von Diskette und Verbindung mit einem vorhandenen Programm im Speicher.

● **FUNKTION 1:** Verbinden eines Programms mit einem im Speicher vorhandenen Programm.

FORMAT: LOAD SUB C 〔, X 〔, Y〕〕

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")
X : Neue Zeilennummer
Y : Inkrement (Zeilenabstand)



ERLÄUTERUNG:

1. LOAD SUB lädt ein Programm (Zwischenkode-Format) mit der Dateikennung C zu einem vorhandenen Programm in den Speicher. Das angefügte Programm erhält dazu neue Zeilennummern. Die erste Zeile wird durch X definiert, der Abstand wird durch Y bestimmt.
2. Werden Zeilennummer X und Abstand Y weggelassen, erhält das nachgeladene Programm neue Zeilennummern im Abstand 10; die erste Zeilennummer entspricht der um 10 erhöten höchsten Zeilennummer des vorhandenen Programms.
   höchsten Zeilennummer des vorhandenen Programms.
3. Die Zeilennummer X muß größer als die höchste Zeilennummer des vorhandenen Programms sein.

EINFACHES BEISPIEL:

```
LOAD SUB "PROBE,SHARP",500,20 ENTER
```

..... Lädt das Programm PROBE von einer Diskette mit der Kennzeichnung SHARP und verbindet es mit dem im Speicher vorliegenden Programm. Es erhält neue Zeilennummern mit Abstand 20, basierend auf Zeile 500.

● **FUNKTION 2:** Verbindet zwei Programme und löscht einen festgelegten Bereich des im Speicher befindlichen Programms.

FORMAT: LOAD SUB C; ,Z | S [, X [, Y]]

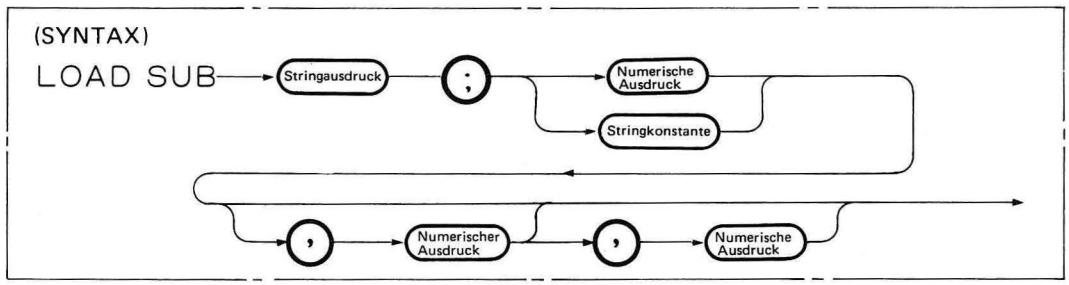
C : Dateikennung ("Dateiname I, Diskettenname I: Kanal-Laufwerk-Nummer")

Z : Erste Zeilennummer des zu löschenden Bereichs

S : Zeilenmarke des zu löschenden Bereichs

X : Neue Basiszeile des angefügten Programms

Y : Inkrement (Zeilenabstand)



ERLÄUTERUNG:

1. LOAD SUB löscht einen Programmbereich, dessen Anfang mit Z bzw.8 festgelegt ist. Danach wird das anzufügende Programm, bezogen auf die Zeilennummer X, mit einem Abstand Y neu numeriert und dann mit dem im Speicher befindlichen Programm verbunden. Das anzufügende Programm ist durch die Dateikennung bestimmt.
2. Diese Anweisung entspricht Funktion 1, wenn X und Y weggelassen werden.
3. Die Basiszeile des anzufügenden Programms muß größer als die durch Z oder S spezifizierte Zeile sein.

   Im Speicher sei z.B. folgendes Programm enthalten:

```
10 INPUT A
20 INPUT B
30 DISP A
40 IF B=3 THEN 80
```

   Löschung

   Entspricht Z dem Wert 30, wird das dieser Zeile folgende Programm gelöscht. Das nachzuladende Programm muß also mit einer Zeilennummer größer 20 beginnen.

4. Wird diese Anweisung innerhalb eines Programms verwendet, muß die mit Z oder S festgelegte Zeilennummer über der Zeile liegen, in der diese Anweisung steht.

EINFACHES BEISPIEL:

```
LOAD SUB "PROBE";100 ENTER
```

..... Löscht alle auf Zeile 100 folgenden Programmzeilen des im Speicher befindlichen Programms, lädt das Progamm PROBE von Laufwerk A0 und versieht es mit neuen Zeilennummern im Abstand 10. Sie schließen direkt an das noch vorhandene Programm an.

WICHTIGER HINWEIS

1. Diese Anweisung kann nicht innerhalb eines Unterprogramms aufgerufen werden.
2. Programme in ASCII-Format können nicht nachgeladen werden.

ANWENDUNGSBEISPIEL:

Programm A

```
100 DIM A$(1)*20,B$(3,4)
110 DISP CLEAR
120 TABLE /,6,15,11,8,15;5,3,3,3,3,3
130 A$(1)="Telefonverzeichnis"
140 DISP A$
150 LOAD SUB "SUB1"
160 END
```

Dieses Programm soll im Speicher vorliegen und den Namen A tragen.

```
10 CURSOR 11,6:DISP "Name"
20 CURSOR 25,6:DISP "Ort"
30 CURSOR 34,6:DISP "Telefon 1"
40 CURSOR 46,6:DISP "Telefon 2"
50 SUBEND
```

Dieses Programm soll auf Diskette in Laufwerk A0 unter dem Namen SUB1 gespeichert sein.

RUN [ENTER] ..... Beide Programme werden durch den Befehl LOAD SUB in Zeile 150 von Programm A miteinander verbunden.

Ergebnis:

```
100 DIM A$(1)*20,B$(3,4)
110 DISP CLEAR
120 TABLE /,6,15,11,8,15;5,3,3,3,3,3
130 A$(1)="Telefonverzeichnis"
140 DISP A$(1)
150 LOAD SUB "SUB1"
160 END
170 CURSOR 11,6:DISP "Name"
180 CURSOR 25,6:DISP "Ort"
190 CURSOR 34,6:DISP "Telefon 1"
200 CURSOR 46,6:DISP "Telefon 2"
210 SUBEND
```

Diese Zeilen des Pogramms SUB1 wurden neu numeriert und an das Programm A angefügt.

Programm B: Aufruf mehrerer Unterprogramme

Es befinde sich folgendes Programm im Speicher:

```
10 A$="A":B$=":A1"
20 FOR I=1 TO 2
30 J=48+I
40 C$=CHR$ (J)
50 D$=A$+C$+B$
60 LOAD SUB D$:100,100,20
70 GO SUB 90
80 NEXT :END
90 PRINT "START"
100 GO TO 10
```

Die durch den String D$ festgelegte Datei wird geladen und an das Programm B im Speicher angefügt. Dabei wird das angefügte Programm basierend auf Zeile 100 mit einem Zeilenabstand von 20 neu numeriert.

Programme auf Diskette:
Programm "A1"

```
10 PRINT "Erstes UnterProGramm!"
20 LIST PRINT
30 RETURN
```

Programm "A2"

```
10 PRINT "Zweites UnterProGramm!"
20 LIST PRINT
30 RETURN
```

Ergebnis:
Es wird folgendes Listing auf dem Drucker ausgegeben:

```
START
Erstes UnterProGramm!
   10 A$="A":B$=":A1"
   20 FOR I=1 TO 2
   30 J=48+I
   40 C$=CHR$ (J)
   50 D$=A$+C$+B$
   60 LOAD SUB D$:100,100,20
   70 GO SUB 90
   80 NEXT :END
   90 PRINT "START"
  100 PRINT "Erstes UnterProGramm!"
  120 LIST PRINT
  140 RETURN
START
Zweites UnterProGramm!
   10 A$="A":B$=":A1"
   20 FOR I=1 TO 2
   30 J=48+I
   40 C$=CHR$ (J)
   50 D$=A$+C$+B$
   60 LOAD SUB D$:100,100,20
   70 GO SUB 90
   80 NEXT :END
   90 PRINT "START"
  100 PRINT "Zweites UnterProGramm!"
  120 LIST PRINT
  140 RETURN
```

Diese Zeilen (Programm A1) wurden zur Programmausführung angefügt.

Diese Zeilen (Programm A2) wurden zur Programmausführung angefügt.

In dieser Form können mehrere Programme an ein Grundprogramm angefügt werden. Dazu kann in obigem Beispiel Zeile 20 abgeändert werden.

ALOAD

Schaltet die beim Einschalten des Geräts wirksame automatische Lade-Funktion ein oder aus.

〈VERWENDUNG〉
Zur automatischen Programmausführung nach dem Einschalten.

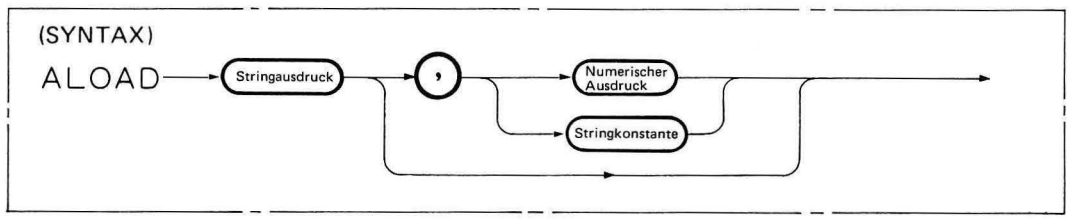
● **FUNKTION 1:** Einschalten der automatischen Lade-Funktion.

FORMAT: ALOAD C 〔, X | S〕

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")

X : Zeilennummer, Programmausführung beginnt hier

S : Zeilenmarke, Programmausführung beginnt hier



ERLÄUTERUNG:

1. ALOAD lädt automatisch ein Programm (ASCII oder Zwischenkode) mit der Dateikennung C von Diskette (Autostart-Funktion).
2. Werden Zeilennummer X oder Zeilenmarke S angegeben, beginnt die Programmausführung mit der dadurch spezifizierten Programmstell. Fehlen beide, wird nur der automatische Ladevorgang ausgeführt.

EINFACHES BEISPIELE:

1. ALOAD "MENUE","SPIEL" ENTER
   ..... Beim Einschalten des Geräts wird automatsch das Programm "MENUE" geladen und die Programmausführung beginnt bei der durch die Marke SPIEL gekennzeichneten Programmzeile.
2. ALOAD "MENUE",10 ENTER
   ..... Beim Einschalten des Geräts wird automatisch das Programm "MENUE" geladen. Die Programmausführung beginnt mit Zeile 10.

WICHTIGER HINWEIS

Die automatische Ladefunktion kann nur mit Laufwerk A0 betrieben werden. Sie bezieht sich nur auf eine Programmdatei.

● **FUNKTION 2:** Setzt die automatische Ladefunktion außer Funktion.
FORMAT: ALOAD

(SYNTAX)
ALOAD ⟶

ERLÄUTERUNG:

Die automatische Ladefunktion wird außer Funktion gesetzt.

EINFACHES BEISPIEL:

ALOAD [ENTER] ..... Setzt die automatische Ladefunktion außer Betrieb. Nach dem Einschaltvorgang wartet das System auf die Eingabe eines FDOS-Befehls über die Tastatur.

WICHTIGER HINWEIS

1. Die automatische Ladefunktion kann nur mit Programmen auf der FDOS-Masterdiskette in Laufwerk A0 ausgeführt werden.
2. Die automatische Ladefunktion kann nur für ein Programm der Masterdiskette definiert werden.

<HILFREICHER TIP>
Ob ein Programm für die automatische Ladefunktion definiert wurde, kann mit Hilfe des Befehls CAT festgestellt werden (Seite 320).

Inhaltsverzeichnis: (Ausschnitt)

```
CAT   A0                              V1.0A
VOLUME NAME-SHARP
ALOAD       -TEST/SR:A0,  100
-----------------------------------------
```

↑ Programm, das mit der automatischen Ladefunktion geladen werden soll.

↑ Zeilennummer oder Marke, mit der die Programmausführung beginnen soll.

5. SONDERDATEIEN

Dieser Abschnitt erläutert Befehle für die Behandlung von Dateien zur Speicherung von Maschinensprache, Bildschirmspeicherinhalten und der Funktionstastenbelegung:

MSAVE
MLOAD
VSAVE
VLOAD
GSAVE
GLOAD
KSAVE
KLOAD

Diese Befehle können auch innerhalb des Programms ausgeführt werden.

MSAVE

Speichert ein Maschinenprogramm.

FORMAT: MSAVE C, X, Y

C : Dateikennung ("Dateiname ., Diskettenname I: Kanal-Laufwerk-Nummer")

X : Anfangsadresse des Maschinenprogramms

Y : Dateilänge (Länge des Maschinenprogramms in Bytes)

(SYNTAX)



ERLÄUTERUNG:

MSAVE speichert ein Maschinenprogramm mit der Dateiteikennung C auf Diskette. Die Anfangsadresse des zu speichernden Programms wird durch X festgelegt, die Größe des zu speichernen Bereichs durch Y.

EINFACHES BEISPIEL:

```
MSAVE "Z80,SHARP",&A000,1024 [ENTER]
```

..... Speichert ein Maschinenprogramm mit dem Namen Z80 auf der Diskette SHARP. Die Anfangsadresse dieses Programms ist &A000, es umfaßt 1024 Bytes (1 K Byte).

<PRAKTISCHER TIP>

Mit dem CAT-Befehl kann des Ergebnis der Aufzeichnung geprüft werden.

MLOAD

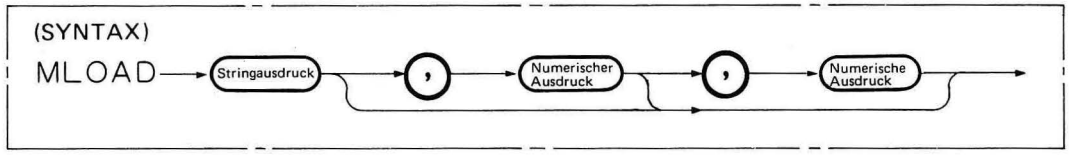
Lädt ein Maschinenprogramm von Diskette.

FORMAT: MLOAD C [, X [, Y]]

C : Dateikennung ("Dateiname, Diskettenname: Kanal-Laufwerk-Nummer")

X : Ladeadresse (Das Programm wird ab dieser Adresse abgelegt)

Y : Programmlänge (in Bytes)



ERLÄUTERUNG:

1. MLODA lädt ein Maschinenprogramm mit der Dateikennung C, beginnend mit Adresse X, in den Speicher. Das geladene Programm umfaßt Y Bytes.
2. Werden X oder Y weggelassen,wird der bei MSAVE verwendete Wert entsprechend übernommen.

EINFACHES BEISPIEL:

```
MLOAD "Z80",&A000,1024 ENTER
```

..... Lädt das Maschinenprogramm Z80 in einer Länge von 1024 Bytes an die Adresse &A000 in den Hauptspeicher.

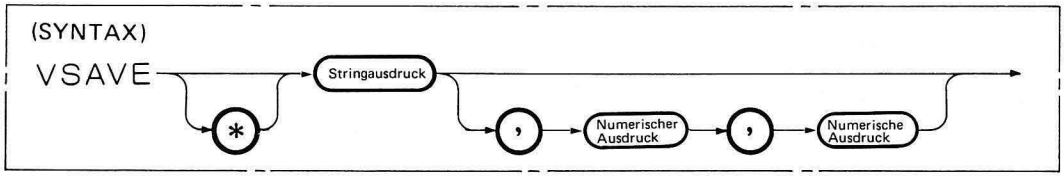
VSAVE

Legt den Bildschirmspeicherinhalt auf Diskette ab.

〈VERWENDUNG〉
Zur Speicherung des Bildschirminhalts auf Diskette.

FORMAT: VSAVE 〔*〕 C 〔, X, Y〕

* : Auswahl der Bildschirminformation
C : Dateikennung ("Dateiname, Diskettennamen: Kanal-Laufwerk-Nummer")
X : Erste Bildschirmzeile
Y : Letzte Bildschirmzeile



ERLÄUTERUNG:

1. VSAVE speichert alle Zeichen zwischen der X+1. und Y+1. Zeile unter der Darstellungsmodus (blinken oder invertieren) sind nicht enthalten.
2. Wird ein *-Symbol vor der Dateikennung eingegeben, werden nur Markierungslinien, Farben und Zeichenmodus (blinken oder invertieren) aufgezeichnet.
3. Werden X und Y weggelassen, wird der gesamte Bildschirminhalt aufgezeichnet.

EINFACHES BEISPIELE:

1. `VSAVE "VIDEO",0,9` ENTER
   ..... Speichert die in den Zeilen 1 bis 10 vorhandenen Zeichen unter dem Namen VIDEO auf Diskette.
2. `VSAVE *"VIDEO2"` ENTER
   ..... Speichert nur Markierungslinien, Farben und Zeichenmodus (blinken oder invertieren) auf mit der Dateikennung VIDEO2 auf Diskette.

<PRAKTISCHER TIP>
Werden Text- und Graphikinformationen gemischt auf dem Bildschirm dargestellt, und sollen diese Informationen gemeinsam auf Diskette gespeichert werden, kann der VSAVE-Befehl zusammen mit dem GSAVE-Befehl (Seite 377) verwendet werden. Um so gespeicherte Informationen wieder sichtbar zu machen, werden die Befehle VLOAD und GLOAD verwendet.

VLOAD

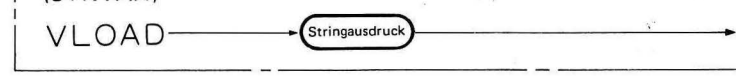
Laden von Textbildschirminhalten.

〈VERWENDUNG〉
Zum Laden und Anzeigen der auf Diskette gespeicherten Textbildschirminhalte.

FORMAT: VLOAD C

C: Dateikennung ("Dateiname, Diskettennamen: Kanal-Laufwerk-Nummer")

(SYNTAX)



ERLÄUTERUNG:

VLOAD lädt einen mit dem VSAVE-Befehl (Seite 375) aufgezeichneten Bildschirminhalt (Zeicheninformationen, oder Markierungslinien, Farben und Zeichenmodus – blinkend oder invertiert –) mit der Dateikennung C von Diskette in den Bildschirmspeicher und zeigt ihn an. Alle Zeilen, die über den mit VLOAD geladenen Bereich hinausgehen, bleiben erhalten.

EINFACHES BEISPIEL:

```
VLOAD "VIDEO,SHARP" ENTER
```

..... Lädt den in der Datei VIDEO auf der Diskette SHARP gespeicherten Bildschirminhalt in die durch VSAVE festgelegten Zeilen und zeigt sie an.

<PRAKTISCHER TIP>

Zur Aufzeichnungen und erneuten Darstellung blinkender Zeichen auf dem Bildschirm sind zwei Schritte nötig:

1. Bildschirminhalt und Zeichenmodus müssen in zwei getrennten Dateien mit Hilfe des VSAVE-Befehls gespeichert werden.
2. Um das gespeicherte Schirmbild in gleicher Form erneut darzustellen, müssen beide Dateien mit dem VLOAD-Befehl geladen werden. Die Ladereihenfolge spielt keine Rolle.

GSAVE

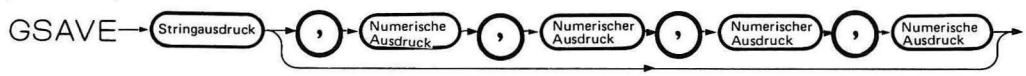
Speichert Bildschirmgraphik auf Diskette.

〈VERWENDUNG〉
Zur Speicherung der Graphikebenen auf Diskette.

FORMAT: GSAVE C [, X, Y, U, V]

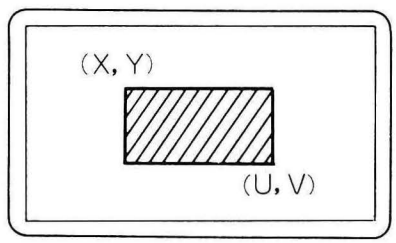
C : Dateikennung ("Dateiname I, Diskettennamen, Kanal-Laufwerk-Nummer")
X : X-Koordinate der linken oberen Ecke
Y : Y-Koordinate der linken oberen Ecke
U : X-Koordinate der rechten unteren Ecke
V : Y-Koordinate der rechten unteren Ecke

(SYNTAX)



ERLÄUTERUNG:

1. GSAVE speichert den durch das Rechteck (X, Y)—(U, V) begrenzten Graphikbereich (Schraffierter Bereich im Bild unten) unter der Dateikennung C auf Diskette. Der gesamte Inhalt der durch ODISP definierten Graphikebene kann aufgezeichnet werden.
   Informationen aus der Zeichenebene des Bildschirms können mit diesem Befehl nicht gespeichert werden.



[Graphikebene]

2. Werden die Koordinaten (X, Y) und (U, V) nicht spezifiziert, werden alle Graphikinformationen des Bildschirms gespeichert.

EINFACHES BEISPIEL:

```
GSAVE "GRAPH,SHARP",10,10,100,100 [ENTER]
```

..... Speichert Graphikinformationen, die in dem durch die Koordinaten (10, 10) und (100, 100) festgelegten Rechteck enthalten sind. Die Aufzeichnung erfolgt unter dem Namen GRAPH auf der Diskette SHARP.

WICHTIGER HINWEIS

1. Graphikinformationen sind im Graphikvideospeicher gespeichert. Informationen darüber enthält das Bedienerhandbuch.
2. Das Abspeichern einer Graphikebene erfaßt eine Speicherbereich von 16 Blöcken (32KBytes). Auf ausreichend Diskettenspeicherplatz ist zu achten.

GLOAD

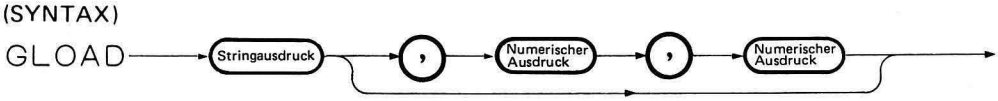
Lädt Bildschirmgraphik von Diskette

〈VERWENDUNG〉
Lädt auf Diskette gespeicherte Graphikinformationen und zeigt sie auf dem Bildschirm an.

FORMAT: GLOAD C [, X, Y]

C : Dateikennung ("Dateiname, Diskettename: Kanal-Laufwerr-Nummer")
X : X-Koordinate des Anzeigestartpunkts
Y : Y-Koordinate des Anzeigestartpunkts

(SYNTAX)



ERLÄUTERUNG:

1. GLOAD lädt eine Graphikdatei mit der Dateikennung C von Diskette in den Graphikspeicher und stellt die enthaltenen Informationen dar. Der Startpunkt des Bereichs, in den die Graphikdaten geladen werden sollen, kann über die Koordinaten (X, Y) bestimmt werden.
2. Werden die Koordinaten (X, Y) nicht festgelegt, werden die im GSAVE-Befehl verwendeten Daten verwendet.

EINFACHES BEISPIEL:

```
GLOAD "GRAPH,SHARP",10,10 ENTER
```

..... Lädt eine mit der Dateikennung "GRAPH, SHARP" festgelegte Graphikdatei beginnend mit den Koordinaten (10, 10) in den Graphikspeicher.

WICHTIGER HINWEIS

1. Über den Anzeigebereich hinausgehende Teile der geladenen Graphikdatei gehen verloren.
2. Sind nur die Graphikebenen 1 und 2 definiert, können auch nur zwei Ebenen aus einer Graphikdatei geladen werden. Dies ist auch dann der Fall, wenn mit dem GSAVE-Befehl 3 Ebenen aufgezeichnet worden sind. (Die Zahl der Ebenen beim Aufzeichnen und Laden sollte gleich sein.)
3. Enthält die Graphikdatei z.B. nur Informationen für eine Ebene, obwohl 2 Ebenen definiert sind, wird beim Laden die zweite Ebene gelöscht. (Sind mehr Graphikebenen definiert als in der Datei gespeicherten sind, werden die verbleibenden Ebenen gelöscht.)
4. Die beim Abspeichern und Laden verwendeten Ebenen müssen nicht übereinstimmen.

KSAVE

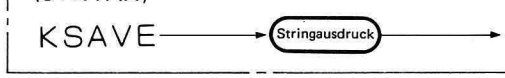
Speichert die Belegung der Funktionstasten.

〈VERWENDUNG〉
Speichern der aktuellen Tastaturbelegung auf Diskette.

FORMAT: KSAVE C

C : Dateikennung ("Dateiname, Diskettename: Kanal-Laufwerk-Nummer")

(SYNTAX)



ERLÄUTERUNG:

KSAVE speichert die aktuelle Belegung der Funktionstasten (DEF1-DEF20) unter dem in C festgelegten Namen auf Diskette.

EINFACHES BEISPIEL:

```
KSAVE "KEY1,SHARP" ENTER
```

..... Speichert die aktuelle Belegung der Tasten DEF1 bis DEF20 als Datei "KEY1" auf einer mit dem Namen "SHARP" versehenen Diskette.

KLOAD

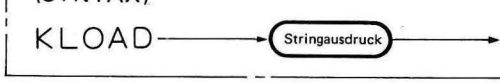
Lädt eine Datei mit Tastaturdaten.

〈VERWENDUNG〉
Definition der Funktionstasten mit einer Diskettendatei.

FORMAT: KLOAD C

C : Dateikennung ("Dateiname, Diskettename: Kanal-Laufwerk-Nummer")

(SYNTAX)



ERLÄUTERUNG:

KLOAD lädt eine durch C bestimmte Datei mit Tastaturdaten in den Speicher und weist diese Daten den Funktionstasten zu.

EINFACHES BEISPIEL:

KLOAD "KEY1,SHARP" ENTER

..... Die Funktionstasten werden mit dem Inhalt der Datei "KEY1" auf der Diskette "SHARP" neu belegt.

STICHWORTVERZEICHNIS

ASCII-Format 39
7-Bit-ASCII-Format 295
8-Bit-ASCII-Format 295
Ablaufteseten 73
Anweisung 2
Anwendermicrobereich 287
Argument 158
Arithmetische Operatoren 3
Array-Elemente 20
Array-Variablen 19,128
Ausgabeformat (Numerische Größen) .. 16
Ausgabemodus, graphischer 241
Ausmalen 264
Autoladefunktion 370
BASIC-Start 1
BASIC-Zwischenkode 355
Befehl 2
Betriebssystem 328
Bildschirm 31
Bildschirm-POKE 289
Bildschirmausgabemodus 237
Bildschirmspeicherdateien 375
Binäre Daten 339
Blinkende Anzeige 228
Bogenmaß 211
Datei 35
Dateibearbeitung 334
Dateikennung 41
Dateiname 35
Dateinamen (Ändern) 323
Dateischutz (Lock) 315
Datenfeld 36
Datenlänge 332
Datensätze 332
Datentypen 340
Datenübertragung 296
Debuggen 55, 71
Disketten-BASIC 305
Diskettenbezeichung (Ändern) 324
Disketteninhaltsverzeichnis .. 47,320,321
Disketteninitialisierung 45
Diskettenname 35
Drucker (RS-232C) 308
Druckerausgabemodus 238
Echofunktion 300
Editieren 66
Eingabeformate 165
Eingabemodus, graphischer 32,243
Einzelschritt-Testen 71
Ellipse 259
Endzeichen 300,347
Entsichern, Dateien 317
ERROR 172
EDOS 40
FDOS-Befehle 43
FDOS-Routienen 41
Farbgraphik 245
Farbkodetabelle 239
Farbmodus 237
Farbwahl 239
Farbwiedergabe 33
Fehlerbehandlung 172
Fehlermeldung 22
Fehlermeldung (akustisch) 213
Fehlernummernliste ..... App. 51
Festkommaformat 17
Fließkommaformat 18
Formatierte Dateien 343
Funktion 23, 199
Funktionsname 76
Funktionstasten 62
Geschützte Programme 356
Gradmaß 210
Graphik, ASCII-Zeichen in der 282
Graphik, Farbbestimmung 245
Graphikanweisungen 240
Graphikausdruck 276
Graphikdateien 378
Graphikebene 32,242,243
Graphikkoordinaten, ... 33
Graphikkursor 277
Graphikpunkt 252
Graphikstrecke 255
Graphiktastatureingabe 279

Größtes Arrayelement 191
Hauptprogramm 158, 161
Hexadezimale Schreibweise 10, 24
Hintergrundfarbe 34, 245
Integer-Variable 10
Invertierte Anzeige 229
Kanal-Laufwerk-Nummer 41
Kanalanforderung 306
Kleinstes Arrayelement 193
Koordinatensysteme 33
Kopieren 318
Kopieren, Dateien 50, 52
Kopieren, Disketten 48
Kopieren, Programmzeilen 188
Kopieren, Programme 67
Kreis 259
Kreissektor 261
Kursor 33, 223
Kursorabfrage 220
Kursorbewegung 225
Kursorpositionierung 223
Laden, Programme 360
Logische Operatoren 5
Löschen, Bildschirm 215
Löschen, Dateien 319
Löschen, Disketten 54
Löschen, Graphikebene 249
Löschen, Graphikpunkt 254
Manuelle Betriebsart 2
Markierungslinien 259
Maschinensprache 1,286,288,289
Maschinensprachedateien 373
Maschinenunterprogramm 288
Musik 202
Nullpunktverschiebung 249
Numerische Konstante 9
Numerische Variable 12
Numerischer Ausdruck 14
Parität 294
Passwort 356
Programm 62
Programmaufbau 61
Programmdatei 38, 354
Programmeingabe 63
Programmieren 62
Programmspeicherung 355
Programmtest 71
Programmunterbrechung 204
Programmzeile 61
Pseudographik 232
RS-232C-Schnittstelle 293
Randomdateien 38,332,348
Real-Variable 12
Record 36
Recordlänge 37
Relationale Operatoren 3
SI/SO-Status 295
Schlüsselwortverzeichnis ..... App. 43
Schleife 149
Sequentielle Dateien 37, 337
Sichern, Dateien 315
Skalierungsmaßstab 249
Sonderdateien 372
Speicheradressen 289
Speicherbelegungsplan ..... App. 50
Standardformat 16
Stapel 7
Stopbits 294
String 15
Stringübertragung 232
Stringkonstante 10
Stringvariable 12
Stringverknüpfung 187
Tabellen 232
Tastaturabfrage 163
Tastaturdateien 379
Terminalbetrieb 310
Textbildschirm 31
Textkoordinaten 33
Trennzeichen 347
Übertragungskanal 294
Übertragungsrate 294
Unterprogramm 134, 157, 160
Unterprogrammaufruf 157
Unterprogrammname 76
Unterprogrammzeiger 171
Variablen 11
Variablendeklaration .. 13,206,207,208
Variablenübergabe 289
Verkettung, Programme 362

Verschieben der BS-Anzeige 235
Vorrangregeln 15
Winkelmessung 210,211
Wortlänge 289
Zeiger 110
Zeilenmarken 75
Zeilennummer 61
Zeitschaltung 181
Zufallszahlen 28, 212
Zwischenkode 38, 355
Zwischenspeicher (RS-232C) 305
Zwischenspeicher (Dateien) 330

BEFEHLE, ANWEISUNGEN

ABS 26
ACS 27
ALOAD 370
AND 5
ASC 23
ASN 27
ATN 27
AUTO 96
BACKUP 48
BASIC 44
BINPUT # 341
BL 228
BPRINT # 339
BYE 328
CALL 157
CAT 43 320
CDEG 29
CDMS 29
CHAIN 362
CHANGE DISP 237
CHANGE PRINT 238
CHANNEL 294
CHR$ 24
CIRCLE 259
CLOSE 336
CLS 247
COLOR 239
CONT 82
COPY 50 318
COS 27
CREATE 332
CURSOR 223
DATA 120
DEBUG 55
DEFFN 199
DEFINT 206
DEFREAL 207
DEFSTR 208
DEG 210
DELETE 93
DFK 58
DIM 128
DISP 107
DISP CLEAR 215
EDIT 90
END 127
ENTER 221
ERASE 95 288
ERL 14
ERN 14
ERROR MUSIC 213
EXP 28
FIXED 17
FLOAT 18
FN 198
FOR/NEXT 149
GCHR 282
GCOLOR 245
GCURSOR 277
GDISP 273
GENTER 269
GET # 353
GINPUT 279
GLOAD 379
GO SUB 134
GO TO 131
GPRINT 276
GSAVE 377
IF---:ELSE 144
IF---THEN 141
IMAGE 113
INIT 45
INPUT 117
INPUT # 345
INT 45
KEY 163
KEYIN 165
KILLALL 54
KILL 319
KLOAD 381
KSAVE 380
LEFT$ 25
LET 104
LEN 24
LINE 226 255
LINPUT 326
LIST 85
LN 28
LOAD 360
LOAD SUB 366
LOCK 315
LOG 28
LPRINT 324

MAXFILE 330
MAXS 191
MERGE 364
MID$ 25
MINS 193
MLOAD 374
MOD 29
MODIFY 28
MOVE 187
MSAVE 373
MUSIC 202
MVCUR 225
NEW 92 287
NOT 6
NORMAL 16
ODISP 243
OFF ERROR 175
OFF KEY 180
OFF TIME 184
ON ERROR 172
ON.....GO TO 138
ON.....GO SUB 139
ON KEY 176
ON TIME 181
OPCHNL 305
OPEN 334
OR 5
PAINT 264
PEEK 29
PI 9
PRESET 254
PRINT 110
PRINT # 342
POINT 30
POKE(PEEK) 289
POLLING 306
POP 171
POS 220
PSET 252
PUT # 352
RAD 211
RANDOM 212
RCV 300
RE 230
READ 121
REM 125
REN 98
RESET 219
RESTORE 123
RESUME 185
RETURN 136
RFORMAT # 350
RIGHT$ 25
RLIST 308
RNAME 323
RND 28
RUN 78
RV 229
SAVE 355
SCALE 249
SCOPY 52
SCROL 235
SDISP 241
SEARCH 195
SEND 296
SENREV 303
SEP # 347
SET 217
SGN 26
SIN 26
SPA 25
SQR 26
STATUS 29
STOP 126
STORE 358
STR$ 24
STRING$ 25
SUB 160
SUBEND 162
TAB 107 110
TABLE 232
TAN 27
TERM 310
TIME 13
TR 100
TR STOP 102
UNLOCK 317
USR 290
VAL 24
VLOAD 376
VRUN 80
VSAVE 375
WAIT 204

* * * **MEMO** * * *

SHARP CORPORATION
OSAKA, JAPAN

Printed in Japan
Gedruckt in Japan
Imprimé au Japon
Stampato in Giappone

5G1.0-K 〈TINSG1020ACZZ〉 ③

© 1984 SHARP CORPORATION