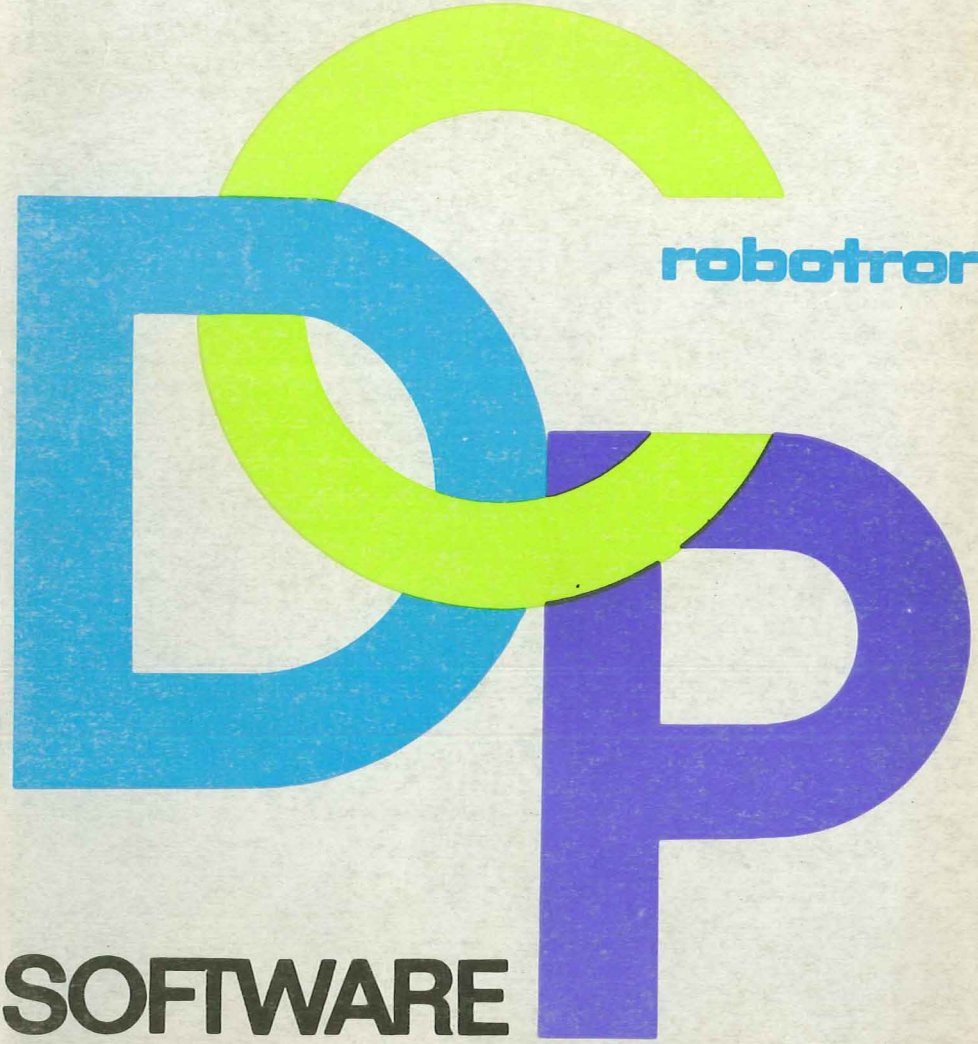


SOFTWARE
DOKUMENTATION

BASIC-Interpreter

Bedienungsanleitung und Sprachbeschreibung

fuer

BASIC-Interpreter (BASI)

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

Die vorliegende 2. Auflage der Dokumentation "Bedienungsanleitung und Sprachbeschreibung fuer BASIC-Interpreter" unter DCP 3.30 entspricht dem Stand vom 30.9.1988 und unterliegt nicht dem Aenderungsdienst.

Die Dokumentation wurde durch ein Kollektiv des

VEB Robotron Buchungsmaschinenwerk

Karl-Marx-Stadt

Software-Zentrum

erarbeitet.

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

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

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

		Seite
1.	Allgemeine Informationen	8
1.1.	Starten des BASIC-Interpreters	8
1.1.1.	Beispiele fuer das Laden von BASIC	11
1.1.2.	Umlenken der Standardein- und -ausgabe	12
1.2.	Betriebsarten	13
1.3.	Eingabe und Korrektur eines BASIC-Programmes	15
1.3.1.	Tastatur	15
1.3.2.	Korrektur der Eingabezeile	26
1.3.3.	Eingeben oder Aendern eines BASIC-Programmes	29
1.3.4.	Aendern von Zeilen auf dem gesamten Bildschirm	31
1.3.5.	Syntaxfehler	32
2.	Hinweise zur Programmierung in BASIC	34
2.1.	Beschreibungsform der Sprache	34
2.2.	Zeilenformat	34
2.3.	Zeichensatz	35
2.4.	Konstanten	36
2.5.	Variablen	38
2.5.1.	Variablennamen	39
2.5.2.	Definition von Variablentypen	39
2.5.3.	Feldvariable	40
2.6.	Typkonvertierung	42
2.7.	Numerische Ausdruecke und Operatoren	44
2.7.1.	Arithmetische Operatoren	45
2.7.2.	Vergleichsoperatoren	46
2.7.3.	Logische Operatoren	47
2.7.4.	Numerische Funktionen	50
2.7.5.	Reihenfolge der Abarbeitung	50
2.8.	Zeichenkettenausdruecke und Operatoren	51
2.8.1.	Verkettung	52
2.8.2.	Zeichenkettenfunktionen	52
3.	Kommandos	53
3.1.	Editierkommandos	53
3.1.1.	NEW	53
3.1.2.	AUTO	54
3.1.3.	EDIT	55
3.1.4.	LIST	56
3.1.5.	LLIST	57
3.1.6.	DELETE	57
3.1.7.	RENUM	58
3.2.	Programmausfuehrungskommandos	59
3.2.1.	RUN	59
3.2.2.	CLEAR	60
3.2.3.	CONT	62
3.2.4.	TRON und TROFF	63
3.2.5.	SYSTEM	64

Seite

3.3. Diskettenbezogene Kommandos 64
3.3.1. RESET 64
3.3.2. LOAD 64
3.3.3. SAVE 65
3.3.4. MERGE 67
3.3.5. NAME 68
3.3.6. FILES 68
3.3.7. KILL 69

4. BASIC-Grundanweisungen 71
4.1. Kommentaranweisung REM 71
4.2. Datum, Uhrzeit, Lautsprecher 72
4.2.1. DATE$ 72
4.2.2. TIME$ 73
4.2.3. BEEP 74
4.3. Typdeklarationsanweisungen DEFTyp 74
4.4. Wertzuweisungen 75
4.4.1. LET 75
4.4.2. SWAP 76
4.4.3. RANDOMIZE 76
4.5. Dialog- Ein-/-Ausgabe 78
4.5.1. INPUT 78
4.5.2. LINE INPUT 80
4.5.3. PRINT 81
4.5.4. PRINT USING 82
4.5.5. LPRINT und LPRINT USING 86
4.5.6. WRITE 88
4.5.7. KEY 88
4.5.8. WIDTH 92
4.6. Arbeit mit dem Konstantenspeicher 95
4.6.1. READ 95
4.6.2. DATA 96
4.6.3. RESTORE 97
4.7. Steueranweisungen 98
4.7.1. STOP 98
4.7.2. END 99
4.7.3. GOTO 99
4.7.4. GOSUB ... RETURN 100
4.7.5. ON .. GOTO und ON ... GOSUB 101
4.7.6. RETURN 103
4.7.7. FOR ... NEXT 103
4.7.8. IF ... THEN ... ELSE 106
4.7.9. WHILE ... WEND 108
4.8. Dimensionieren von Feldern 109
4.8.1. DIM 109
4.8.2. OPTION BASE 110
4.8.3. ERASE 111
4.9. Anwendereigene Funktionsdefinition DEF FN 112
4.10. Programmueberlagerung 113
4.10.1. CHAIN 113
4.10.2. COMMON 115

		Seite
4.11.	Fehlerbehandlung	116
4.11.1.	ERROR	116
4.11.2.	ERR und ERL	117
4.11.3.	ON ERROR GOTO	118
4.11.4.	RESUME	119
4.12.	Unterbrechungsabfrage	120
4.12.1.	KEY(n)	120
4.12.2.	ON KEY(n)	121
4.12.3.	ON TIMER	123
4.12.4.	ON PLAY(n)	124
4.12.5.	COM(n) und ON COM(n)	124
5.	BASIC-Grundfunktionen	125
5.1.	Numerische Funktionen	125
5.1.1.	Arithmetische Funktionen	125
5.1.2.	Konvertierung von Zahlenwerten	127
5.1.3.	Exponentialfunktionen	129
5.1.4.	Trigonometrische Funktionen	130
5.1.5.	Zeichenkettenbezogene numerische Funktionen	133
5.2.	Zeichenketten-Funktionen	136
5.2.1.	Allgemeine Zeichenkettenfunktionen	136
5.2.2.	Zeichenkettenfunktionen fuer die Ein-/Ausgabe	142
5.3.	Dienstleistungs-Funktionen	144
5.4.	Funktionen zur Drucker-/Bildschirmsteuerung	146
6.	Anweisungen und Funktionen zur Dateiarbeit	149
6.1.	Ueberblick	149
6.1.1.	Namensgebung fuer Dateien	149
6.1.2.	Verzeichnisarten	152
6.1.3.	Diskettendateien	153
6.2.	Eroeffnen und Schliessen von Diskettendateien	153
6.2.1.	OPEN	153
6.2.2.	CLOSE	157
6.3.	Sequentielle Dateien	158
6.3.1.	PRINT # und PRINT # USING	158
6.3.2.	WRITE #	160
6.3.3.	INPUT #	161
6.3.4.	LINE INPUT #	162
6.4.	Direktzugriffsdateien	164
6.4.1.	FIELD	164
6.4.2.	LSET und RSET	165
6.4.3.	PUT	166
6.4.4.	GET	167
6.5.	Funktionen	168
6.5.1.	MKI$, MKS$, MKD$	168
6.5.2.	CVI, CVS, CVD	168
6.5.3.	EOF	169
6.5.4.	LOC	170
6.5.5.	LOF	171
6.5.6.	INPUT$	171
6.6.	Treiberunterstuetzung	172

		Seite
7.	Datenfernverarbeitung	173
7.1.	Eroeffnen und Schliessen einer Datenfernverarbeitungsdatei	173
7.1.1.	OPEN "COM ...	173
7.1.2.	CLOSE	177
7.2.	Anweisungen und Funktionen fuer die Datenfernverarbeitungs- Ein-/Ausgabe	177
7.2.1.	Sequentielle Ein-/Ausgabe	178
7.2.2.	GET und PUT fuer Datenfernverarbeitungsdateien	178
7.2.3.	Ein-/ Ausgabefunktionen fuer Datenfernverarbeitung	178
7.3.	Unterbrechungsabfrage	180
7.3.1.	COM(n)	180
7.3.2.	ON COM(n)	181
7.4.	Datenfernverarbeitungsfehler	182
8.	Anweisungen und Funktionen zu Grafik und Musik	183
8.1.	Allgemeine Hinweise zur Arbeit mit dem Bildschirm	183
8.2.	Allgemeingueltige Anweisungen	187
8.3.	Farbe	192
8.3.1.	Farbe im Textmodus	192
8.3.2.	Farbe im Grafikmodus	195
8.4.	Grafik	196
8.4.1.	Anweisungen	196
8.4.2.	Funktionen	217
8.5.	Musik	219
8.5.1.	Anweisungen	219
8.5.2.	Anweisungen zur Programmunterbrechung	225
8.5.3.	Funktionen	227
9.	Prozessorarbeit	228
9.1.	Arbeit mit dem Speicher	228
9.2.	Anweisungen und Funktionen fuer den Zugriff auf den Speicher	228
9.2.1.	DEF SEG	228
9.2.2.	POKE	230
9.2.3.	PEEK	230
9.2.4.	BLOAD	231
9.2.5.	BSAVE	232
9.2.6.	VARPTR	233
9.2.7.	VARPTR$	234
9.2.8.	OUT	235
9.2.9.	INP	236
9.2.10.	WAIT	236
9.3.	Anweisungen und Funktionen fuer die Unterprogrammarbeit	237
9.3.1.	DEF USR	237
9.3.2.	USR	238
9.3.3.	CALL	240

		Seite
10.	DCP-typische BASIC-Erweiterungen	241
10.1.	Verzeichniszugriff	241
10.2.	Laden von DCP-Programmen	244
10.3.	Arbeit mit der BASIC-Umgebungsstabelle	247
10.4.	Verarbeitung von Steuerzeichen bei benutzereigenen Einheitentreibern	252
10.5.	Ermittlung von Einheitenfehlern	254

Anlage A	Liste der Kommandos, Anweisungen und Funktionen in alphabetischer Reihenfolge	256
Anlage B	Reservierte Woerter	269
Anlage C	Zusammenstellung der Fehlercodes und der Fehlermeldungen	271
Anlage D	ASCII-Zeichencode	282
Anlage E	Hexadezimale Umwandlungstabelle	286
Anlage F	Mathematische Funktionen	287
Anlage G	Bildschirm- und Druckersteuerzeichen	288
Anlage H	Technische Informationen	291
Anlage I	Programmiertips	296
Anlage J	Suchcodes	301
Anlage K	Sachwortverzeichnis	303

1. Allgemeine Informationen

Der BASIC-Interpreter **BASI (DCPX)** wird im Betriebssystem **DCP 3.20** bei den Personalcomputern ROBOTRON EC 1834 und abgeleiteten Terminals verwendet.
Mit dem BASIC-Interpreter koennen BASIC-Programme erfasst, korrigiert, getestet und abgearbeitet werden. Mit dem darueberhinaus zur Verfuegung stehenden BASIC-Compiler **BASC (DCPX)** sind diese BASIC-Programme in effektive abarbeitbare Maschinenprogramme ueberfuehrbar.
Der BASIC-Interpreter **BASI.EXE** benoetigt 68K Byte Speicher.

1.1. Starten des BASIC-Interpreters

- Die Systemdiskette ist einzulegen und der Computer einzuschalten.

- Nach Bereitschaftsmeldung des Computers ist der BASIC-Interpreter durch Eingabe von **BASI <ENTER>** zu laden.

Daraufhin wird angezeigt:

```
BASI (DCPX) V1.0
nnnnn Bytes free
```

Es sind Formaterweiterungen fuer diese Kommandozeile moeglich.

Das vollstaendige Format des Befehls **BASI** sieht wie folgt aus:

```
BASI  [Dateiangabe][<Stdein][>Stdaus]
      [/F:Dateien][/S:Puffergroesse][/C:DFV-Puffer]
      [/M:[Max.Arbeitsbereich][,Max.Blockgroesse]][/D]
```

Dateiangabe
ist der Name fuer ein Programm, das geladen und sofort ausgefuehrt werden soll. Es muss aus einer Zeichenkettenkonstanten bestehen, die nicht in Anfuehrungszeichen eingeschlossen werden darf und den Regeln fuer die Angabe von Dateien entspricht, die in Kapitel 6 unter "Namengebung fuer Dateien" beschrieben sind. Die Angabe eines Pfades ist moeglich. Ein Standarddateityp **.BAS** wird benutzt, falls kein anderer Dateityp angegeben ist und die Laenge des Dateinamens aus acht Zeichen oder weniger besteht. Wird **Dateiangabe** beim Laden von BASIC mit angegeben, so arbeitet **BASIC** weiter, als ob der Befehl **RUN Dateiangabe** als erstes eingegeben worden waere, nachdem **BASIC** bereit war. Man sollte beachten, dass durch die Angabe der Auswahl **Dateiangabe** die **BASIC-Textzeile** mit Angabe des freien Speicherplatzes nicht angezeigt wird.

<Stdein:
Ein BASIC-Progamm erhaelt normalerweise seine Eingabe ueber die Tastatur (Standardeingabeeinheit). Mit **<Stdein** kann BASIC die Eingabe von der Datei annehmen, die angegeben wird. Wird **<Stdein** angegeben, muss diese Angabe vor den Schaltern stehen. (Ein Schalter ist eine Auswahl, die mit einem Schraegstrich (/) beginnt und zur Angabe von Parametern benutzt wird.) Unter 1.1.2. "Umlenken der Standardein- und ausgabe" befinden sich weitere Informationen.

>Stdaus:
Ein BASIC-Programm gibt normalerweise seine Ausgabe ueber den Bildschirm aus (Standardausgabeeinheit). Mit **>Stdaus** wird BASIC die Ausgabe zu der Datei oder Einheit uebertragen, die angegeben wird. Wird **>Stdaus** angegeben, muss diese Angabe vor den Schaltern stehen. Unter 1.1.2. "Umlenken der Standardein- und ausgabe" befinden sich weitere Informationen.

Die folgenden Angaben sind Schalter:

/F:Dateien
gibt die maximale Anzahl der Dateien an, die gleichzeitig waehrend der Ausfuehrung eines **BASIC-Programms** eroeffnet werden koennen. Jede Datei benoetigt 62 Byte des Hauptspeichers fuer den Dateisteuerblock plus 128 Byte fuer die Puffergroesse, die mit der Auswahl **/S:** angegeben ist. Wird die Auswahl **/F:** weggelassen, ist die Standardannahme fuer Dateien **3**. Wieviele Dateien tatsaechlich gleichzeitig eroeffnet werden koennen, haengt von dem Wert des Parameters **FILES=** in der DOS-Konfigurationsdatei **CONFIG.SYS** ab.

Der hoechste Wert fuer **FILES=** ist **255**, die Standardannahme ist **FILES=8**. Die ersten drei Dateien werden mit **Stdein,Stdaus, Stderr,Stdaux** und **Stdprn** angesprochen. Eine weitere Datei wird benoetigt fuer **LOAD,SAVE,CHAIN,NAME** und **MERGE**. Bei **FILES=10** werden 6 Dateien fuer **BASIC** Dateiein/-ausgabe belassen. Deshalb ist **F:6** der Maximalwert, der angegeben werden darf, falls **FILES=10** ist und alle Dateien zur gleichen Zeit eroeffnet sein sollen.

Wird versucht, eine Datei mit **OPEN** zu eroeffnen, nachdem die moegliche Anzahl eroeffneter Dateien bereits erschoepft ist, wird der Fehler **"Too many files"** (Zu viele Dateien) ausgegeben.

/S:Puffergroesse
setzt die Puffergroesse fuer Dateien mit direktem Zugriff. Der Parameter fuer die Satzlaenge in der Anweisung **OPEN** darf diesen Wert nicht uebersteigen. Die Standardpuffergroesse ist **128** Bytes. Der groesste eingegebene Wert darf 32767 sein. Fuer den besten Durchsatz bei der Benutzung von Dateien mit direktem Zugriff wird **/S:512** vorgeschlagen.

Mit **/C:DFV-Puffer**
wird die Groesse jedes Datenfernverarbeitungspuffers zum Empfang der Daten bei asynchroner Uebertragung gesetzt. Die Angabe ist nur dann notwendig, wenn ein Adapter fuer asynchrone Uebertragung benutzt wird. Die maximale Groesse darf 32767 sein. Wird die Angabe **/C:** weggelassen, werden jedem Puffer zum Empfangen **256** Bytes zugeordnet, zum Senden **128** Bytes. Wird mit einer hohen Uebertragungsrate gearbeitet, wird **/C:1024** vorgeschlagen. So werden beide Empfangspuffer auf die in der Auswahl angegebene Groesse gesetzt. Man kann die **V.24-Unterstuetzung** ausschalten, wenn man den Wert **Null (/C:0)** angibt. In diesem Fall wird fuer die Datenfernverarbeitung kein Pufferplatz reserviert und die DFV-Unterstuetzung nicht eingefuegt, wenn **BASIC** geladen wird.

/M:Max.Arbeitsbereich
setzt die maximale Anzahl der Bytes, die als **BASIC-Arbeitsbereich** benutzt werden duerfen. **BASIC** ordnet 64 K des Speichers Daten und Speichersegmenten zu. Man kann diese Auswahl benutzen, um Platz fuer Unterprogramme in Maschinensprache zu reservieren oder um einen speziellen Datenspeicher anzulegen. Gegebenenfalls sollte unter "Hauptspeicherbelegung" in **Anlage H** nachgelesen werden, wie **BASIC** den Speicherplatz verwendet. Wird die Auswahl **/M:** weggelassen, so wird der gesamte verfuegbare Hauptspeicherbereich bis zu einem Maximum von **64K** benutzt.

Der Schalter **/M:32768** ordnet z.B. 32768 Byte dem BASIC Daten- und Stapelbereichssegment zu und ermoeglicht es, die anderen 32768 Byte fuer Unterprogramme in Maschinensprache zu benutzen.

Der Parameter **Max.Blockgroesse** kann benutzt werden, wenn Programme oberhalb des **BASIC**-Arbeitsbereichs geladen werden sollen, um Platz fuer sie zu reservieren. Dies ist erforderlich, wenn die Anweisung **SHELL** benutzt werden soll (weitere Angaben hierzu unter "Anweisung **SHELL**" im Kapitel 10). Ein Fehler bei der Anwendung von **Max.Blockgroesse** kann auftreten, wenn **COMMAND.COM** am Anfang der Routinen geladen und eine Anweisung **SHELL** ausgefuehrt wurde.

Die Auswahl **Max.Blockgroesse** gibt die maximale Anzahl von Abschnitten an, die **BASIC** zugeordnet werden koennen, inklusive des Raums ausserhalb des Stapelbereichs fuer Unterprogramme in Maschinensprache. Ein Abschnitt besteht aus Bloecken von jeweils 16 Byte. Wird der Parameter weggelassen, so wird **&H1000 (4096)** angenommen. Dadurch werden 65536 (4096 * 16) Byte fuer die Daten und Stapelbereichsegmente des **BASIC** zugeordnet.

Sollen z.B. 65536 Byte fuer **BASIC** und 512 Byte fuer Unterprogramme in Maschinensprache zugeordnet werden, kann dies durch die Angabe **/M:,&H1020** (4096 Abschnitte fuer **BASIC** und 32 Abschnitte fuer Programme) getan werden.

Die Zuordnung /M:,2048 bewirkt,dass **BASIC** 2048 Abschnitte (32768 Byte) fuer Daten und Stapelbereichsegmente zuordnet und benutzt.

/M:32000,2048 bedeutet, dass **BASIC** maximal 32768 Byte zuordnet, aber nur die unteren 32000 Byte benutzt. Dadurch bleiben 768 Byte (32768 - 32000) fuer Programme uebrig.

Hinweis:

> **Dateien,Max.Arbeitsbereich,Max.Blockgroesse,Puffergroesse** und **DFV-Puffer**

bestehen aus Zahlen, die entweder dezimal, oktal (beginnend mit &O) oder hexadezimal (beginnend mit &H) dargestellt sein koennen.

/D
Durch die Angabe D in der **BASIC-Befehlszeile** koennen die Funktionen ATN, COS, EXP, LOG, SIN, SQR und TAN mit doppelter Genauigkeit berechnet werden.
In diesem Fall werden zusaetzlich ungefaehr 3000 Byte belegt.

1.1.1. Beispiele fuer das Laden von BASIC:

```
BASI LOHN
```

Damit wird **BASIC** gestartet. Es benutzt standardmaessig 64 K des Hauptspeichers und ermoeglicht die Arbeit mit drei Dateien. Das Programm **LOHN.BAS** wird geladen und ausgefuehrt.

```
BASI TEST/F:6
```

Mit dieser Auswahl benutzt **BASIC** 64 K des Hauptspeichers und sechs Dateien, laedt und fuehrt **TEST.BAS** aus (Die Datei **CONFIG.SYS** muss in **FILES=10** geaendert werden, wenn sechs Dateien benutzt werden sollen.).

```
BASI /C:0/M:32768
```

Inaktiviert die Unterstuetzung fuer V.24 und benutzt nur 32 K des Hauptspeichers fuer den **BASIC**-Arbeitsbereich.

```
BASI /F:4/S:512
```

Benutzt vier Dateien und erlaubt eine maximale Satzlaenge von 512 Byte.

```
BASI TTY/C:512
```

Mit dieser Auswahl benutzt **BASIC** 64 K des Hauptspeichers und drei Dateien; dem V.24-Empfangspuffer werden 512 Byte und dem Uebertragungspuffer 128 Byte zugeordnet. Die Datei **TTY.BAS** wird geladen und ausgefuehrt.

1.1.2. Umlenken der Standardein- und -ausgabe

Die **Standardeingabe**, normalerweise von der Tastatur, kann zu jeder in der **BASIC**-Befehlszeile angegebenen Datei umgelenkt werden.
Auch die **Standardausgabe**, die normalerweise dem Bildschirm zugeordnet ist, kann zu einer Datei oder Einheit umgelenkt werden.

BASI Dateiangabe[<Stdein][>][>Stdaus]

Hinweise:

- Nach dem Umlenken werden alle Anweisungen **INPUT**, **INPUT$**, **INKEY$** und **LINE INPUT** aus der angegebenen Eingabedatei gelesen und nicht ueber die Tastatur eingegeben.

- Nach dem Umlenken der Standardein- **und** -ausgabe werden alle Anweisungen **PRINT** und Fehlernachrichten in die angegebene Ausgabedatei oder Ausgabeeinheit geschrieben und nicht zum Bildschirm.

- Wird eine Datei nur zur Ausgabe umgelenkt, werden alle auf dem Bildschirm angezeigten Daten in die angegebene Ausgabedatei gebracht.

- Beim Umlenken der Ausgabe werden alle Fehlernachrichten zum Bildschirm und zur angegebenen Ausgabedatei geschickt. Alle Dateien werden geschlossen, das Programm wird beendet, die Steuerung geht an DCP zurueck.

- Eingaben in eine Datei von **"KYBD:"** werden von der Tastatur aus gelesen.

- Ausgaben in eine Datei von **"SCRN:"** werden zum Bildschirm gesendet.

- **BASIC** erlaubt weiterhin das Belegen von Tasten mit Funktionen, wenn eine Anweisung **ON KEY(n)** angegeben ist.

- Ist die Standardausgabe umgelenkt, so kann durch Betaetigen der Tasten **<CTRL PRTSC>** keine Ausgabe des momentanen Bildschirminhalts auf dem Drucker erfolgen.

- Mit **<CTRL PAUSE>** wird zur Standardausgabe zurueckgekehrt: alle Dateien werden geschlossen, die Steuerung wird an DCP uebergeben.

Beispiele:

1. Im folgenden Beispiel koennen die Daten mit **INPUT**, **INPUT$**, **INKEY$** und **LINE INPUT** weiter von der Tastatur gelesen werden. Alle auf den Bildschirm ausgegebenen Daten, die mit **PRINT** ausgegebenen Daten einschliesslich Fehlernachrichten werden in die Datei **DATA.AUS** geschrieben.

   BASI PROG >DATA.AUS

2. In diesem Beispiel werden die Daten, die mit **INPUT**, **INPUT$**, **INKEY$** und **LINE INPUT** von der Datei **DATA.EIN** gelesen und die mit **PRINT** ausgegebenen Daten weiter am Bildschirm angezeigt.

   BASI PROG <DATA.EIN

3. Im naechsten Beispiel werden Daten, die mit **INPUT**, **INPUT$**, **INKEY$** und **LINE INPUT** eingelesen werden, von der Datei **EINGABE.DAT** empfangen und Daten, die mit **PRINT** ausgegeben werden, werden in die Datei **AUSGABE.DAT** geschrieben.

   BASI PROG <EINGABE.DAT >AUSGABE.DAT

4. In diesem Beispiel werden die Daten, die mit **INPUT**, **INPUT$**, **INKEY$** und **LINE INPUT** aus der Datei **\VERKAUF\PETER\TRANS** gelesen werden, in die Datei **\VERKAUF\VERKAUF.DAT** geschrieben.

   BASI PROG <\VERKAUF\PETER\TRANS >>\VERKAUF\VERKAUF.DAT

1.2. Betriebsarten

Ist **BASIC** gestartet, so wird als Antwort **Ok** ausgegeben. **Ok** ist das Bereitschaftszeichen des **BASIC-Interpreters**. Dieser Status heisst **Befehlsebene.** Zu diesem Zeitpunkt kann **BASIC** in zwei Betriebsarten benutzt werden: der **direkten** oder der **indirekten Betriebsart.**

Direkte Betriebsart

In der direkten Betriebsart werden **BASIC-Anweisungen** sofort nach der Eingabe ausgefuehrt. Dies wird **BASIC** dadurch mitgeteilt, dass eine Anweisung oder ein Kommando **ohne** Zeilennummer eingegeben wird. Man kann Ergebnisse arithmetischer oder logischer Operationen sofort anzeigen oder sie fuer eine spaetere Benutzung speichern. Die Instruktionen selbst werden, nachdem sie ausgefuehrt sind, nicht gespeichert. Dieser Modus ist fuer den Testlauf nuetzlich sowie fuer schnelle Berechnungen, die kein umfangreiches Programm benoetigen.

Beispiel:

```
Ok
PRINT 20+2
 22
Ok
```

Indirekte_Betriebsart

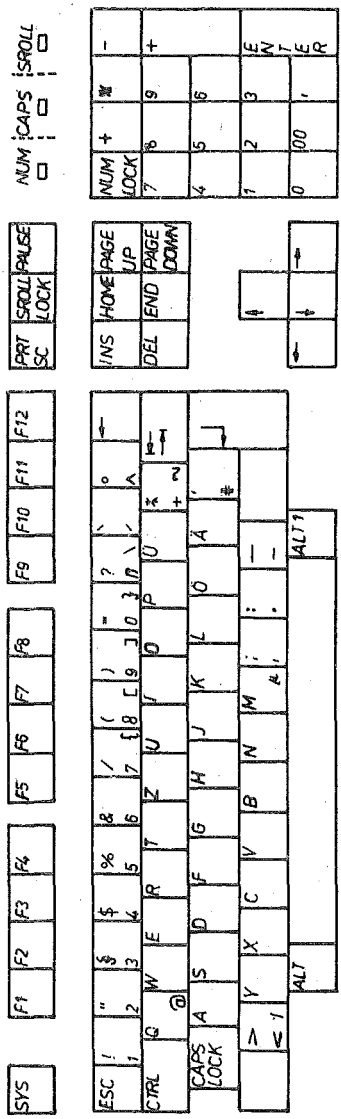
In der indirekten Betriebsart werden gespeicherte Programme abgearbeitet.Jede Zeile eines Programms beginnt mit einer **Zeilennummer**. Die Zeile wird dann als Teil eines Programms im Hauptspeicher gespeichert. Das im Hauptspeicher befindliche Programm kann mit dem Kommando **RUN** beliebig oft ausgefuehrt werden.

Beispiel:

```
Ok
10 INPUT A
20 INPUT B
30 PRINT A+B
40 END
Ok
```

1.3.___Eingabe_und_Korrektur_eines_BASIC-Programmes

1.3.1._Tastatur



1.3.1.1. Funktionstasten

```
+----------+   +----------+   +----------+
|F1|F2|F3|F4|   |F5|F6|F7|F8|   |F9|F10|F11|F12|
+----------+   +----------+   +----------+
```

Benutzung der Funktionstasten:

- Jeder Funktionstaste ist eines der haeufig benutzten **BASIC-Schluesselworte** zugeordnet, das beim Betaetigen der Taste automatisch eingegeben wird. Die Standardbelegung der Funktionstasten wird auf der Zeile 25 des Bildschirms angezeigt.
  Abweichend von der Standardbelegung koennen vom Anwender eigene Zeilenfolgen zugeordnet werden. (Siehe Pkt. 4 **KEY-Anweisung**)

- Die Funktionstasten sind in Zusammenhang mit der Anweisung **ON KEY** fuer Programmunterbrechungen verwendbar (Siehe Pkt. 4, Anweisung **ON KEY(n)**).

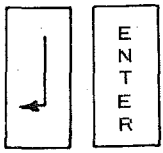
1.3.1.2. Alphanumerische Tastatur

Der alphanumerische Tastaturblock entspricht weitgehend einer Schreibmaschinentastatur. Sie beinhaltet Buchstaben, Ziffern und Sonderzeichen.

Umschalttasten

Grossbuchstaben und die Sonderzeichen ueber den Zahlen der numerischen Tasten werden erzeugt, indem man die Umschalttasten betaetigt und niederhaelt und ausserdem die gewuenschte Taste betaetigt.

ENTER



Die **ENTER-Taste** beendet Tastatureingabeanweisungen **(INPUT)**.

Zusaetzlich ist eine Taste **<ENTER>** noch neben der Zehnertastatur angeordnet. Sie hat die gleiche Wirkung.

Grossbuchstaben-Feststelltaste

Die Taste **<CAPS LOCK>** arbeitet aehnlich dem Umschaltfeststeller, erzeugt aber nur grosse Buchstaben und nicht die Zeichen ueber den numerischen oder anderen Tasten, die man mit der Umschalttaste erhaelt. Nachdem die Taste **<CAPS LOCK>** betaetigt wurde, erhaelt man so lange Grossbuchstaben, bis man sie erneut betaetigt. Man kann im Grossschreibestatus auch Kleinbuchstaben erhalten, indem man eine der Umschalttasten betaetigt und niederhaelt. Laesst man die Umschalttaste los, kommt man in den Grossschreibestatus zurueck.

Ruecktaste

Die Ruecktaste arbeitet etwas anders als die Ruecktaste einer Schreibmaschine. Sie fuehrt nicht nur den Kursor zurueck, sondern loescht auch das eingegebene Zeichen. Man muss die Taste <"Kursor nach links"> verwenden, falls eine vorherige Eingabe nicht geloescht werden soll. Siehe unter "**Programmkorrekturtasten**".

Andere Umschalttasten:

Zusaetzlich zu den Umschalttasten, die die Tastatur von Klein- auf Grossbuchstaben umschalten, gibt es zwei andere Umschalttasten auf der Schreibmaschinentastatur. Es handelt sich um die Tasten **<ALT>** und **<CTRL>**. Beide Tasten werden wie Umschalttasten benutzt, d.h. man betaetigt und haelt die Taste **<ALT>** (oder **<CTRL>**) und betaetigt die gewuenschte Taste. Dann laesst man beide Tasten wieder los. Jedoch werden **<ALT>** und **<CTRL>** fuer verschiedene Funktionen benutzt.

Mit der Taste <ALT> kann man BASIC-Anweisungsschluesselwoerter eingeben. Damit ist es moeglich, mit einem einzigen Tastendruck ein ganzes BASIC-Schluesselwort einzugeben.

Das BASIC-Schluesselwort wird eingegeben, wenn man die Taste <ALT> niederhaelt und eine der alphabetischen Tasten <A bis Z> betaetigt. Die jedem Buchstaben zugeordneten Schluesselwoerter sind unten aufgefuehrt. Buchstaben, denen kein Schluesselwort zugeordnet ist, sind durch "(kein Wort)" gekennzeichnet.

A	AUTO	N	NEXT
B	BSAVE	O	OPEN
C	COLOR	P	PRINT
D	DELETE	Q	(kein Wort)
E	ELSE	R	RUN
F	FOR	S	SCREEN
G	GOTO	T	THEN
H	HEX$	U	USING
I	INPUT	V	VAL
J	(kein Wort)	W	WIDTH
K	KEY	X	XOR
L	LOCATE	Y	(kein Wort)
M	(kein Wort)	Z	(kein Wort)

Die Taste <ALT> dient in Verbindung mit Tasten der Zehnertastatur dazu, Zeichen einzugeben, die auf den Tasten nicht zu finden sind. Man kann dies tun, indem man die Taste <ALT> niederhaelt und den dreistelligen ASCII-Code fuer das Zeichen eintastet (siehe Anlage D "ASCII-Zeichencodes").

CTRL

Mit der Taste <CTRL> ist es ebenfalls moeglich, verschiedene Codes und Zeichen einzugeben, die nicht auf der Tastatur verfuegbar sind. Zum Beispiel ist CTRL-G das Alarmzeichen. Wird diese Taste betaetigt, kommt ueber den Lautsprecher ein Alarmsignal. CTRL-G bedeutet, dass die Taste <CTRL> niedergehalten wird und dann die Taste <G> betaetigt wird. Danach koennen beide Tasten losgelassen werden.

Die Taste <CTRL> wird auch in Verbindung mit anderen Tasten benutzt, um Programme zu veraendern.

1.3.1.3. Sondertasten

PRTSC

<PRTSC> steht fuer "Drucken Bildschirminhalt". Die Taste ergibt eine Kopie des Bildschirminhalts auf dem Drucker (**LPT1:**).

Hinweis:

Zeichen, die der Drucker nicht "versteht", werden als Leerstellen ausgegeben.

Sondertastenkombinationen:

Folgende Tastenkombinationen fuehren Spezialfunktionen aus:

CTRL-PAUSE

<CTRL-PAUSE> unterbricht die Programmausfuehrung an der naechsten Programmanweisung und kehrt auf die **BASIC**-Befehlsebene zurueck. Mit **<CTRL-PAUSE>** verlaesst man auch den **AUTO**-Zeilennumerierungsmodus.

CTRL-NUM LOCK

<CTRL-NUM LOCK> versetzt den Computer in den **Pausen**-Status. Damit kann temporaer das Drucken oder Programmauflisten angehalten werden. Die Pause wird aufgehoben, sobald irgendeine Taste ausser Umschalttasten, die Taste **<PAUSE>** oder die Taste **<INS>** (Einfuegen) betaetigt wird.

CTRL-PRTSC

Durch Bedienen der Tasten **<CTRL-PRTSC>** wird Text, der auf dem Bildschirm angezeigt wird, auch auf dem Drucker ausgegeben.

Spezielle Programmkorrekturtasten

Mit Hilfe der Kursortasten, der Ruecktaste und der Taste **<CTRL>** ist es moeglich, den Kursor an einen Platz auf dem Bildschirm zu positionieren, Zeichen einzugeben oder Zeichen zu loeschen. Die Tasten haben folgende Funktionen:

Taste(n)	Funktion
HOME	HOME Der Kursor wird in die linke obere Ecke des Bildschirms positioniert.
CTRL-HOME	CTRL HOME Der Bildschirm wird geloescht und der Kursor in die linke obere Ecke des Bildschirms gebracht.
↑ (Kursor nach oben)	↑ Bewegt den Kursor eine Zeile nach oben.
↓ (Kursor nach unten)	↓ Bewegt den Kursor eine Zeile nach unten.
<-- (Kursor nach links)	<-- Bewegt den Kursor ein Zeichen nach links. Bewegt sich der Kursor ueber die linke Seite des Bildschirms hinaus, so springt er zum rechten Rand der vorhergehenden Zeile.

Taste(n)	Funktion
--> (Kursor nach rechts)	 Bewegt den Kursor ein Zeichen nach rechts. Geht der Kursor ueber die rechte Seite des Bildschirms hinaus, so springt er zum linken Rand der naechsten Zeile.
CTRL- --> (Naechstes Wort)	  Bewegt den Kursor nach rechts zum naechsten **Wort**. Ein Wort ist definiert als Zeichen oder Zeichengruppe, die mit einem Buchstaben oder einer Zahl beginnt. Woerter werden durch Leerstellen oder Sonderzeichen getrennt. So ist das naechste Wort der naechste Buchstabe oder die naechste Zahl rechts des Kursors, der oder die einem Leerzeichen oder einem Sonderzeichen folgt. Nehmen wir z.B. die folgende Zeile: `LINE (L1,TEXT1)-(MAX,48) ,3 ,BF` Der Kursor steht jetzt in der Mitte des Wortes **TEXT1**. Werden die Tasten (CTRL-Kursor nach rechts) betaetigt, bewegt sich der Kursor zum Beginn des naechsten Wortes, naemlich **MAX**: `LINE (L1,TEXT1)-(MAX,48) ,3 ,BF` Werden diese Tasten erneut betaetigt, bewegt sich der Kursor zur Zahl **48**: `LINE (L1,TEXT1)-(MAX,48) ,3 ,BF`

Taste(n)	Funktion
CTRL- <-- (Vorheriges Wort)	 Bewegt den Kursor nach links zum vorherigen Wort. Das vorherige Wort ist der Buchstabe oder die Zahl links des Kursors, vor dem oder der ein Leerzeichen oder ein Sonderzeichen steht. Nehmen wir z.B. die folgende Zeile: LINE (L1,TEXT1)-(MAX,48) ,3 ,BF_ Betaetigt man die Tasten (CTRL-Kursor nach links), bewegt sich der Kursor an den Anfang des Wortes **BF**: LINE (L1,TEXT1)-(MAX,48) ,3 ,BF Betaetigt man diese Tasten erneut, bewegt sich der Kursor zum vorherigen Wort, diesmal zur Zahl 3: LINE (L1,TEXT1)-(MAX,48) ,3 ,BF Betaetigt man diese Taste noch zweimal, bewegt sich der Kursor zur Zahl 48, danach zum Wort **MAX**: LINE (L1,TEXT1)-(MAX,48) ,3 ,BF
END	 Bewegt den Kursor zum Ende einer logischen Zeile. Zeichen, die von dieser Stelle an eingegeben werden, werden an das Ende der Zeile angefuegt.

Taste(n)	Funktion
CTRL-END	 Loescht ab der Position des Kursors bis zum Ende der logischen Zeile. Alle physischen Bildschirmzeilen bis zum beendenden Eingabezeichen werden geloescht.
INS	 Setzt den Einfuegemodus. Ist der Einfuegemodus ausgeschaltet, wird er eingeschaltet, sobald diese Taste gedrueckt wird und umgekehrt. Der Einfuegemodus wird dadurch angezeigt, dass die untere Haelfte der Zeichenposition vom Kursor verdeckt wird. Ist der Einfuegemodus eingeschaltet, werden die Zeichen ueber dem Kursor und die folgenden nach rechts verschoben, sobald Zeichen an der Stelle des Kursors eingegeben werden. Nach jedem Tastendruck wird der Text um eine Stelle nach rechts verschoben. Ein Zeilenueberlauf wird beachtet. Das bedeutet, sobald ein Zeichen rechts aus dem Bildschirm verschwindet, erscheint es links wieder auf der nachfolgenden Zeile. Ist der Einfuegemodus ausgeschaltet, so werden existierende Zeichen einer Zeile durch neueingegebene ersetzt. Der Einfuegemodus wird nicht nur durch das erneute Druecken der Taste **<INS>** ausgeschaltet, sondern auch durch das Druecken der Tasten, die den Kursor bewegen oder die Taste **<ENTER>**.

Taste(n)	Funktion
DEL	 Loescht das Zeichen an der momentanen Kursorposition. Danach werden alle Zeichen rechts des geloeschten um eine Position nach links verschoben. Zeilenueberlauf wird beachtet. Dass heisst, sobald eine logische Zeile groesser ist als eine physische Zeile, werden auch die Zeichen der folgenden Zeilen um eine Position nach links verschoben. Die Zeichen der ersten Spalte aller nachfolgenden Zeilen werden an das Ende der vorherigen Zeile verschoben.
<-- (Rueckschritt)	 Das letzte eingegebene Zeichen wird geloescht. Das heisst, das Zeichen links vom Kursor wird geloescht. Alle Zeichen rechts des geloeschten Zeichens werden um eine Stelle nach links verschoben. Nachfolgende Zeichen und Zeilen innerhalb einer logischen Zeile werden wie bei der Taste **<DEL>** nach rechts aufgefuellt.
ESC	 Wird diese Taste gedrueckt, wird der ganze logische Satz vom Bildschirm geloescht. Die Zeile wird nicht an **BASIC** zur Verarbeitung uebergeben. Handelt es sich um eine Programmzeile, wird die Zeile des Programms nicht im Hauptspeicher geloescht.

Taste(n)	Funktion
CTRL-PAUSE	 Es wird zur Befehlsebene zurueckverzweigt, ohne dass Aenderungen, die in der laufenden Zeile vorgenommen wurden, gespeichert werden. Die Zeile wird nicht wie bei **<ESC>** vom Bildschirm geloescht.
<-- --> (Tab)	 Bringt den Kursor zum naechsten Tabulatorstopp. Tabulatorstopps stehen nach jeweils acht Zeichenpositionen, d.h. an Stelle 1, 9, 17 usw. Ist der Einfuegemodus ausgeschaltet, wird beim Betaetigen der Tabulatortaste der Kursor ueber alle Zeichen hinweg bis zum naechsten Tabulatorstopp gefuehrt. Beispiel: 10 REM Bemerkung _ Wird die Tabulatortaste betaetigt, geht der Kursor an die Stelle 9. 10 REM Bemerkung_Basic _ Wird die Taste erneut betaetigt, geht der Kursor an die Stelle 17. Ist der Einfuegemodus eingeschaltet und wird die Tabulatortaste betaetigt, werden ab dem Kursor bis zum naechsten Tabulatorstopp Leerstellen eingefuegt. Zeilenueberlauf wird wie bei **<INS>** ausgefuehrt. Beispiel: 10 REM Bemerkung _ Wird zuerst die Taste **<INS>** (einfuegen) und danach die Tabulatortaste betaetigt, werden bis zur Stelle 17 Leerzeichen eingefuegt. 10 REM Be merkung _

1.3.2. Korrektur der Eingabezeile

Befindet sich **BASIC** in der Befehlsebene, wird jede eingegebene Textzeile vom Korrekturprogramm verarbeitet. Dabei koennen alle im vorherigen Abschnitt unter "Spezielle Programmkorrekturtasten" beschriebenen Tasten benutzt werden. **BASIC** befindet sich in der Befehlsebene nach der Ausgabe **Ok**, bis der Befehl **RUN** eingegeben wird.

Eine **logische Zeile** ist eine Zeichenkette, die **BASIC** als Einheit betrachtet. Es ist moeglich, dass eine logische Zeile laenger als eine physische Bildschirmzeile ist, indem man einfach ueber das Ende der Bildschirmzeile eingibt. Der Kursor bewegt sich automatisch in die naechste Zeile. Dies geht auch mit einem Zeilenvorschub (**<CTRL> <ENTER>**). Wird ein Zeilenvorschub eingegeben, so wird der nachfolgende Text auf die naechste Bildschirmzeile geschrieben, ohne dass Leerstellen eingegeben werden muessen, um den Kursor dorthin zu bringen. Die Zeile wird nicht verarbeitet. Dies geschieht erst, wenn **<ENTER>** gedrueckt wird.

Dabei muss beachtet werden, dass durch den Zeilenvorschub der Rest der physischen Bildschirmzeile mit Leerstellen aufgefuellt wird. Das Zeichen fuer Zeilenvorschub wird an den Text nicht angefuegt. Diese Leerstellen sind in den 255 Zeichen enthalten, die fuer eine **BASIC**-Zeile erlaubt sind.

Wird schliesslich **<ENTER>** gedrueckt, so wird die gesamte logische Zeile fuer die Verarbeitung an **BASIC** weitergegeben.

Zeichen aendern

Wird eine Zeile eingegeben und dabei entdeckt, dass etwas falsch eingegeben wurde, kann diese Eingabe korrigiert werden. Mit der Kursortaste links oder einer anderen Taste, die den Kursor bewegt, wird dieser an die Position gebracht, an der der Fehler auftrat, und der korrekte Buchstabe ueber den falschen geschrieben. Jetzt kann der Kursor mit Hilfe der **<Kursortaste rechts>** oder **<END>** an das Ende der Zeile gebracht und die Eingabe fortgesetzt werden.
Folgendes wurde z.B. eingegeben:

```
LOAD "V:PROG_
```

Es wurde aus Versehen **V:** anstatt **B:** eingegeben.
Die Tasten fuer "Vorheriges Wort" (**<CTRL>-Kursortaste links**) werden zweimal betaetigt, bis sich der Positionsanzeiger unter V befindet:

```
LOAD "V:PROG
     _
```

Dann wird **B:** eingegeben:

```
LOAD "B:PROG
     _
```

Danach wird die Taste **<END>** gedrueckt:

```
LOAD "B:PROG_
```

Der Fehler ist behoben und die Eingabe kann fortgesetzt werden:

```
LOAD "B:PROGRAM"_
```

Zeichen loeschen:

Wird bei der Eingabe ein Zeichen bemerkt, das nicht in der Zeile stehen soll, kann man es mit Hilfe der Taste **<DEL>** loeschen. Mit der **<Kursortaste nach links>** oder einer anderen Taste fuer Kursorbewegungen muss der Kursor an das Zeichen bewegt werden, das geloescht werden soll. Betaetigt man jetzt die Taste **<DEL>**, so wird es geloescht. Danach wird mit der **<Kursortaste nach rechts>** oder **<END>** der Kursor zurueck an das Ende der Zeile gebracht und die Eingabe fortgesetzt.

Beispiel:

```
DEELETE_
```

Um das ueberzaehlige **<E>** zu loeschen, muss der Kursor mit der **<Kursortaste nach links>** unter das ueberzaehlige **E** bewegt werden:

```
DEELETE
  _
```

Jetzt betaetigt man die Taste **<DEL>**:

```
DELETE
  _
```

Nun muss die Taste **<END>** betaetigt werden:

```
DELETE_
```

und die Eingabe kann fortgesetzt werden:

```
DELETE 20_
```

War das falsche Zeichen jenes, das gerade eingegeben wurde, kann man es mit der Ruecktaste loeschen. Danach kann man die Eingabe dieser Zeile fortsetzen.

Beispiel:

```
DELETT_
```

Jetzt die Ruecktaste betaetigen:

```
DELET_
```

Nun kann die Eingabe fortgesetzt werden:

```
DELETE 20_
```

Zeichen hinzufuegen:

Falls man bemerkt, dass man Zeichen in einer Zeile, die gerade eingegeben wird, vergessen hat, bringt man den Kursor an die Stelle, an der die neuen Zeichen eingegeben werden sollen. Danach muss die Taste **<INS>** gedrueckt werden, um in den Einfuegemodus zu kommen. Jetzt gibt man die Zeichen ein, die hinzugefuegt werden sollen. Die Zeichen, die man eingibt, werden an der Stelle des Kursors eingefuegt, und das Zeichen ueber und die neben ihm werden nach rechts verschoben.
Wenn man wie zuvor die Eingabe fortsetzen moechte, bringt man den Kursor mit Hilfe der Tasten **<Kursor nach rechts>** oder **<END>** an das Ende der Zeile und setzt die Eingabe fort. Der Einfuegemodus wird automatisch ausgeschaltet, wenn eine dieser Tasten benutzt wird.

Beispiel:

```
LIS 10_
```

Das **T** in **LIST** wurde vergessen. So wird mit der **<Kursortaste nach links>** der Kursor unter die Leerstelle gebracht.

```
LIS_10
```

Jetzt die Taste **<INS>** betaetigen und den Buchstaben **T** eingeben:

```
LIST_10
```

Loeschen eines Teils der Zeile:

Soll eine Zeile ab der Stelle des Kursors geloescht werden, betaetigt man die Tasten **<CTRL><END>**.

Beispiel:

```
10 REM *** FEHLER
           _
```

Der Kursor wurde unter das **F** des Wortes **Fehler** gesetzt. Zum Loeschen muessen nur noch die Tasten **<CTRL><END>** betaetigt werden:

```
10 REM ***_
```

Loeschen einer Zeile:

Soll eine Zeile, die gerade eingegeben wird, geloescht werden, betaetigt man die Taste <ESC> irgendwo auf der Zeile, ohne danach die Taste <ENTER> zu betaetigen. Dadurch wird die gesamte logische Zeile geloescht.

Beispiel:

```
DIESE ZEILE HAT KEINE BEDEUTUNG_
```

Auch wenn sich der Kursor am Ende der Zeile befindet, wird die gesamte Zeile durch die Taste <ESC> geloescht.

1.3.3. Eingeben oder Aendern eines BASIC-Programms

Jede Textzeile, die eingegeben wird und mit einer Zahl beginnt, wird als **Programmzeile** betrachtet.

Eine **BASIC**-Programmzeile beginnt immer mit einer Zeilennummer, endet mit der Taste <ENTER> und kann maximal 255 Zeichen einschliesslich des Zeichens fuer die Taste <ENTER> enthalten. Enthaelt eine Zeile mehr als 255 Zeichen, werden die ueberzaehligen Zeichen abgeschnitten, sobald die Taste <ENTER> betaetigt wird. Obwohl die ueberzaehligen Zeichen noch auf dem Bildschirm stehen, werden sie von **BASIC** nicht verarbeitet.

BASIC-Schluesselwoerter und Variablennamen bestehen aus Grossbuchstaben. Jedoch kann man sie in jeder Kombination von Gross- und Kleinbuchstaben eingeben. Das Korrekturprogramm wandelt alles ausser **Bemerkungen**, **DATA**-Anweisungen und **Zeichenketten**, die in Anfuehrungszeichen eingeschlossen sind, in Grossbuchstaben um.

Das Fragezeichen (?) ist eine Abkuerzung fuer die Eingabe der Anweisung **"PRINT"**. Wird die Programmzeile mit **LIST** aufgelistet, wird das Fragezeichen wieder in **PRINT** mit einer Leerstelle gewandelt.

Warnung:
Erreicht die Zeile die Maximallaenge, muss das **255.** Zeichen die Taste <ENTER> sein.

Hinzufuegen einer neuen Zeile in ein Programm:

Eingeben einer gueltigen Zeilennummer (im Bereich 0 bis 65529), gefolgt von der Taste <ENTER>. Die Zeile wird als Teil des **BASIC**-Programms im Hauptspeicher gespeichert.

Beispiel:

```
10 Hallo Doris
```

Die Zeile wird als Zeilennummer 10 gespeichert. **Hallo Doris** ist keine gueltige **BASIC**-Anweisung. Es wird beim Eingeben der Zeile kein Fehler angezeigt. Programmzeilen werden **nicht** auf richtige Syntax geprueft, bevor sie dem Programm hinzugefuegt werden. Dies geschieht erst, wenn die Programmzeile ausgefuehrt wird.
Ist schon eine Zeile mit der gleichen Zeilennummer vorhanden, wird sie geloescht und durch die neue ersetzt.

Versucht man, eine Zeile einzufuegen und der Hauptspeicher hat zu wenig Platz, erhaelt man die Fehlermeldung "Out of memory" (Zu wenig Hauptspeicherplatz), und die Zeile wird nicht eingefuegt.

Ersetzen oder Aendern einer existierenden Programmzeile:

Eine schon vorhandene Zeile wird wie oben beschrieben veraendert, falls die eingegebene Zeilennummer schon im Programm vorhanden ist. Die alte Zeile wird durch den Text der neuen Zeile ersetzt.

Beispiel:

```
10 DIES IST EINE NEUE ZEILE 10
```

Die vorherige Zeile 10 (**Hallo Doris**) wuerde durch diese neue Zeile 10 ersetzt.

Loeschen einer Programmzeile:

Zum Loeschen einer Programmzeile ist die Zeilennummer einzugeben und die Taste **<ENTER>** zu betaetigen.

Beispiel:

```
10
```

Dadurch wuerde die Zeile 10 im Programm geloescht.

Man kann auch eine Zeilengruppe im Programm mit Hilfe des Kommandos **DELETE** loeschen. Das Kommando wird in Kapitel 3 beschrieben.

Wird versucht, eine nicht vorhandene Zeile zu loeschen, erhaelt man die Fehlernachricht "Undefined line number" (nicht definierte Zeilennummer).

Zum Loeschen von Programmzeilen darf nicht die Taste **<ESC>** verwendet werden. Durch **<ESC>** wird die Zeile nur auf dem Bildschirm geloescht. Befindet sich die Zeile in einem **BASIC**-Programm, bleibt sie dort erhalten.

Loeschen eines ganzen Programms:

Soll ein Programm insgesamt geloescht werden, das sich im Hauptspeicher befindet, muss das Kommando **NEW** eingegeben werden (siehe Kapitel 3).
Normalerweise wird **NEW** benutzt, um den Hauptspeicher zu loeschen, bevor ein neues Programm eingegeben wird.

1.3.4. Aendern von Zeilen auf dem gesamten Bildschirm / Duplizieren

Man kann jede Zeile auf dem Bildschirm veraendern, indem man die Kursortasten benutzt (beschrieben unter "Spezielle Programmkorrekturtasten"). Danach koennen eine oder alle beschriebenen Techniken benutzt werden, um Zeilen zu veraendern, zu loeschen oder Zeichen hinzuzufuegen.

Sollen Programmzeilen veraendert werden, die nicht angezeigt werden, kann man sie mit Hilfe des Kommandos **LIST** anzeigen. Man listet die Zeile oder den Bereich der Zeilen auf, die veraendert werden sollen (siehe Kapitel 3).

Danach bringt man den Kursor auf die zu korrigierende Zeile und veraendert die Zeile mit den schon beschriebenen Techniken.
Mit Hilfe der Taste **<ENTER>** wird die veraenderte Zeile im Programm gespeichert. Auch mit Hilfe des Kommandos **EDIT** kann man die gewuenschte Zeile anzeigen (Siehe Kapitel 3).

Eine Programmzeile kann auf folgende Art dupliziert werden:

Man bringt den Kursor auf die zu duplizierende Zeile. Danach veraendert man die Zeilennummer auf eine neue Zeilennummer, indem man einfach ueber die alte Zahl die neue Zahl eingibt. Wird danach die Taste **<ENTER>** bedient, sind die alte und die neue Zeile im Programm vorhanden.

Man kann auch die Zeilennummer einer Programmzeile veraendern, indem man die Zeile wie zuvor beschrieben dupliziert und danach die Ausgabezeile loescht.

Die Programmzeile eines **BASIC**-Programms wird erst dann veraendert, wenn die Taste **<ENTER>** betaetigt wird. Muessen mehrere Zeilen veraendert werden, ist es einfacher, mit dem Kursor auf dem Bildschirm an all die Stellen zu gehen, an denen Korrekturen ausgefuehrt werden sollen, dann zur ersten veraenderten Zeile zurueckzukehren und die Taste **<ENTER>** am Beginn jeder Zeile zu betaetigen. Dadurch wird jede veraenderte Zeile im Programm gespeichert.

Es ist nicht noetig,den Kursor an das Ende der logischen Zeile zu bringen, bevor **<ENTER>** betaetigt wird. Das Korrekturprogramm weiss, wo jede logische Zeile endet und verarbeitet die gesamte Zeile, auch wenn **<ENTER>** am Beginn einer Zeile gedrueckt wird.

Hinweis:

Das Kommando **AUTO** kann sehr hilfreich bei der Programmeingabe sein. Der **AUTO-Modus** muss aber durch Betaetigen der Tasten **<CTRL><PAUSE>** verlassen werden, bevor irgendeine Zeile ausser der laufenden Eingabezeile geaendert wird.

Dabei muss daran erinnert werden, dass Veraenderungen mit Hilfe dieser Techniken **nur** das Programm im Hauptspeicher veraendern. Soll das Programm mit den Veraenderungen gespeichert werden, ist dies mit Hilfe des Kommandos **SAVE** zu tun (siehe Kapitel 3), bevor man das Kommando **NEW** eingibt oder **BASIC** verlaesst.

1.3.5. Syntaxfehler

Wird waehrend der Programmausfuehrung ein Syntaxfehler entdeckt, zeigt **BASIC** automatisch die Zeile an, die den Fehler verursachte, so dass man ihn beseitigen kann.

Beispiel:

```
Ok
10 A=2$12
RUN
Syntax Error in 10
Ok
10 A=2$12
_
```

Das Korrekturprogramm hat die fehlerhafte Zeile angezeigt und den Kursor unter die Ziffer 1 gestellt.

Man kann den Kursor nach rechts unter das Waehrungszeichen ($) bringen, es auf ein Pluszeichen (+) aendern und die Taste **<ENTER>** betaetigen. Dadurch wird die korrigierte Zeile in das Programm zurueckgespeichert.

Wird eine durch das Programm unterbrochene Zeile veraendert und in das Programm zurueckgespeichert (wie im Beispiel), geschieht folgendes:

- Alle Variablen und Felder werden geloescht. Das heisst, sie werden zurueck auf Null oder auf die Laenge Null gebracht.
- Alle eroeffneten Dateien werden geschlossen.
- Das Programm kann nicht durch die Anweisung **CONT** weiter ausgefuehrt werden.

Sollen vor den Aenderungen die Inhalte einiger Variablen geprueft werden, muessen die Tasten **<CTRL><PAUSE>** betaetigt werden, um in die Befehlsebene zu gelangen. Der Inhalt der Variablen ist noch vorhanden, da keine Programmzeile veraendert wurde.

Jetzt koennen alle Pruefungen durchgefuehrt werden. Danach veraendert man die Zeile und fuehrt das Programm erneut aus.

2. Hinweise zur Programmierung in BASIC

2.1. Beschreibungsform der Sprache

Alle Kommandos, Anweisungen und Funktionen, die in dieser Dokumentation beschrieben werden, haben ihre Syntax, die nach folgenden Konventionen zu nutzen ist.

- Woerter in Grossbuchstaben sind Schluesselwoerter und muessen komplett in Klein- oder Grossbuchstaben eingegeben werden. BASIC wandelt Woerter immer in Grossbuchstaben um (ausser sie sind Teil einer in Anfuehrungszeichen stehenden Zeichenkette, eine Bemerkung oder eine DATA-Anweisung).

- Alle Angaben in Kleinbuchstaben muessen vom Benutzer definiert und eingegeben werden.

- Angaben in eckigen Klammern sind wahlfrei (optional).

- Mehrere Punkte hintereinander zeigen an, dass eine Angabe so oft wie gewuenscht wiederholt werden kann.

- Jede Interpunktion wie z.B. Kommas, Klammern, Semikolons, Hochkommas oder Gleichheitszeichen muss, wo sie angezeigt wird, eingefuegt werden.
  Ausnahme: Eckige Klammern !

Beispiel:

```
INPUT[;][“Abfrage”;] Variable[,Variable...]
```

Damit die Anweisung INPUT gueltig wird, muss zuerst das Schluesselwort INPUT, wahlfrei gefolgt von einem Semikolon, eingegeben werden. Danach kann eine Abfrage (Bedienerfuehrung) zwischen Anfuehrungszeichen eingefuegt werden. Nach einer Abfrage muss ein Semikolon stehen. Mindestens eine Variable wird fuer eine INPUT-Anweisung benoetigt. Es koennen mehr als eine Variable angegeben werden, wenn sie durch Kommas getrennt sind.

Detaillierte Informationen ueber jeden Parameter stehen im Text nach der Syntax. Die Information ueber dieses Beispiel steht in Kapitel 4 unter Anweisung "INPUT".

2.2. Zeilenformat

Programmzeilen eines BASIC-Programms haben folgendes Format:

```
nnnnn BASIC-Anweisung[:BASIC Anweisung...][‘Kommentar]
```

Sie werden durch die Taste <ENTER> beendet. Das Format wird nachfolgend erklaert.

Zeilennummer: "nnnnn" ist die Zeilennummer, die aus maximal fuenf Ziffern oder einem Punkt bestehen kann. Jede BASIC-Programmzeile beginnt mit einer Zeilennummer. Die Zeilennummern dienen zur Anordnung der Programmzeilen im Hauptspeicher und als Ansprechpunkte fuer Verzweigen und Korrektur. Zeilennummern muessen im Bereich 0 bis 65529 liegen.
Ein Punkt (.) kann in den Kommandos **LIST**, **AUTO**, **DELETE** und **EDIT** benutzt werden, um die Eingabezeile anzusprechen.

BASIC-Anweisungen: Eine BASIC-Anweisung ist entweder **ausfuehrbar** oder **nicht ausfuehrbar**. Ausfuehrbare Anweisungen sind Programminstruktionen, die BASIC mitteilen, was zu tun ist.
Zum Beispiel ist **PRINT** X eine ausfuehrbare Anweisung. Nicht ausfuehrbare Anweisungen, wie z.B. **DATA** oder **REM** verursachen keine Aktion, wenn sie BASIC erkennt. Die BASIC-Anweisungen werden im Kapitel 4 genau erklaert.

Man kann, falls erforderlich, BASIC-Anweisungen in einer Zeile schreiben, jedoch muss jede Anweisung in einer Zeile von der naechsten durch einen **Doppelpunkt** getrennt sein. Die Gesamtanzahl der Zeichen darf 255 nicht ueberschreiten.

Beispiel:

```
Ok
10 FOR I = 1 TO 5: PRINT I: NEXT
RUN
 1
 2
 3
 4
 5
Ok
```

Kommentare: Mit Hilfe des Apostrophs (') koennen an das Ende einer Zeile Kommentare angefuegt werden.

2.3. Zeichensatz

Der Zeichensatz von BASIC besteht aus Buchstaben, numerischen Zeichen und Sonderzeichen.

Die alphabetischen Zeichen von BASIC bestehen aus den Gross- und Kleinbuchstaben des Alphabets. Die numerischen Zeichen von BASIC bestehen aus den Ziffern 0 bis 9.

Folgende Sonderzeichen haben in BASIC eine besondere Bedeutung:

Zeichen	Name
	Leerstelle
=	Gleichheitszeichen oder Zuordnungssymbol
+	Pluszeichen oder Verkettungssymbol
-	Minuszeichen
*	Stern oder Multiplikationssymbol
/	Schraegstrich oder Divisionssymbol
\	Umgekehrter Schraegstrich oder Symbol fuer ganzzahlige Division
^	Fehlerzeichen oder Potenzierungssymbol
(	Linke Klammer
)	Rechte Klammer
%	Prozentzeichen oder Definitionszeichen fuer ganzzahligen Typ (Integer)
#	Nummernzeichen oder Definitionszeichen fuer den Typ doppelter Genauigkeit
$	Dollarzeichen/Waehrungszeichen oder Definitionszeichen fuer den Typ Zeichenkette
!	Ausrufezeichen oder Definitionszeichen fuer den Typ einfacher Genauigkeit
&	Ampersand
,	Komma
.	Punkt oder Dezimalpunkt
'	Apostroph oder Trennzeichen fuer Bemerkungen
;	Semikolon
:	Doppelpunkt oder Trennzeichen fuer Anweisungen
?	Fragezeichen (Abkuerzung fuer PRINT)
<	kleiner als
>	groesser als
"	Anfuehrungszeichen oder Trennzeichen fuer Zeichenketten
_	Unterstreichungszeichen

2.4. Konstanten

Konstanten sind aktuelle Werte, die BASIC waehrend der Ausfuehrung benutzt. Es gibt zwei Arten von Konstanten:

Zeichenkettenkonstanten und numerische Konstanten.

Eine **Zeichenkettenkonstante** ist eine Folge von bis zu 255 Zeichen, die in Anfuehrungszeichen stehen. Beispiele von Zeichenketten:

```
"HALLO"
" 25,000.00"
"Anzahl der Befehle"
```

Numerische Konstanten sind positive oder negative Zahlen. Ein Pluszeichen (+) ist fuer eine positive Zahl wahlfrei. Numerische Konstanten duerfen in BASIC keine Kommas enthalten.

Es gibt fuenf verschiedene Arten numerischer Konstanten:

1. INTEGER — Ganze Zahlen von -32768 bis +32767. Ganzzahlige Konstanten haben keinen Dezimalpunkt.

2. Festkomma — Positive oder negative reelle Zahlen, d.h. Zahlen, die einen Dezimalpunkt enthalten.

3. Gleitkomma — Positive oder negative Zahlen, dargestellt in Exponentialform. Eine Gleitkommakonstante besteht aus einer wahlfrei mit einem Vorzeichen versehenen ganzen Zahl oder einer Festkommazahl (Mantisse), gefolgt von dem Buchstaben E und einer wahlfrei mit Vorzeichen versehenen ganzen Zahl (dem Exponenten). Gleitkommakonstanten doppelter Genauigkeit benutzen den Buchstaben D anstatt E. Siehe naechsten Abschnitt "Genauigkeit numerischer Werte".

   Das E (oder D) bedeutet mal 10 hoch.

   Beispiel: 23E-2

   Hier ist 23 die Mantisse und -2 der Exponent. Die Zahl wird gelesen als "dreiundzwanzig mal 10 hoch minus 2". In der regulaeren Festkommadarstellung koennte man auch 0.23 schreiben.

   Weitere Beispiele:

   235.988E-7 bedeutet: .0000235988
   2359D6 bedeutet: 2359000000

   Jede Zahl von 2.9E-39 bis 1.7E+38 kann als Gleitkommakonstante dargestellt werden.

4. Hex — Hexadezimale Zahlen von maximal vier Zeichen und dem Vorsatz &H. Hexadezimale Zeichen sind die Ziffern 0 bis 9, A, B, C, D, E und F.

   Beispiele:

   &H76
   &H32F

5. Oktal — Oktale Zahlen von maximal sechs Ziffern mit dem Vorsatz &O oder nur &. Oktale Ziffern sind 0 bis 7.

   Beispiele:

   &0347
   &1234

Genauigkeit numerischer Werte

Numerische Konstanten koennen intern ganzzahlig oder als reelle Zahlen mit einfacher oder doppelter Genauigkeit gespeichert werden. Konstanten, die ganzzahlig, hexadezimal oder oktal eingegeben werden, werden in zwei Bytes des Hauptspeichers gespeichert. Reelle Zahlen doppelter Genauigkeit werden mit 17 Stellen Genauigkeit gespeichert und mit bis zu 16 Stellen ausgegeben. Bei Konstanten einfacher Genauigkeit werden sieben Stellen gespeichert und bis zu sieben Stellen ausgegeben, obwohl nur sechs Stellen genau sind.
Eine Konstante mit **einfacher Genauigkeit** ist jede numerische Konstante, die nicht in die Kategorie der **ganzen Zahlen** passt und wie folgt geschrieben ist:

- Sieben oder weniger Zeichen oder
- Exponentialform mit E oder
- einem angefuegten Ausrufezeichen (!).

Eine Konstante **doppelter Genauigkeit** ist jede numerische Konstante, die wie folgt geschrieben ist:

- Acht oder mehr Ziffern oder
- Exponentialform mit D oder
- ein angefuegtes Nummernzeichen (#).

Beispiele fuer Konstanten einfacher und doppelter Genauigkeit:

Einfache Genauigkeit	Doppelte Genauigkeit
46.8	345692811
-1.09E-06	-1.0943D-6
3489.0	3489.0#
22.5!	7654321.1234

2.5. Variablen

Variablen sind Namen, die zum Speichern von Werten in einem BASIC-Programm benutzt werden. Wie bei Konstanten gibt es zwei Variablenarten: numerische und Zeichenketten. Eine **numerische Variable** besitzt immer einen Wert, der aus einer Zahl besteht. Eine **Zeichenkettenvariable** darf nur aus Zeichenkettenwerten bestehen.

Die Laenge einer Zeichenkettenvariablen ist nicht fest. Sie darf zwischen 0 (Null) und 255 Zeichen enthalten. Die Laenge der Zeichenkette, die der Variablen zugeordnet wird, bestimmt die Laenge der Variablen.

Man kann den Wert einer Variablen auf eine Konstante setzen oder man kann ihn als Ergebnis von Berechnungen oder verschiedenen Dateneingabeanweisungen in einem Programm setzen. In jedem Fall muss der Variablentyp (Zeichenkette oder numerisch) gleich dem Typ der Daten sein, die der Variablen zugeordnet werden sollen.

Wird eine numerische Variable benutzt, bevor ihr ein Wert zugeordnet wurde, ist ihr Wert Null. Zeichenkettenvariablen besitzen keinen Inhalt. Das bedeutet, sie enthalten keine Zeichen und haben die Laenge Null.

2.5.1. Variablennamen

Die Variablennamen in BASIC duerfen jede Laenge annehmen. Ist der Name laenger als 40 Zeichen, werden nur die ersten 40 Zeichen geprueft.

Die in einem Variablennamen erlaubten Zeichen sind Buchstaben und Zahlen und der Dezimalpunkt. Das erste Zeichen muss ein Buchstabe sein. Sonderzeichen, die den Typ der Variablen kennzeichnen, sind als letztes Zeichen des Namens erlaubt. Weitere Informationen ueber Typen siehe naechsten Abschnitt "Definition von Variablentypen".

Ein Variablenname darf kein reserviertes Wort sein, darf aber reservierte Woerter eingeschlossen haben. (Siehe unter Anlage B "Reservierte Woerter",in der eine vollstaendige Liste der reservierten Woerter enthalten ist.) Ein Variablenname darf auch kein reserviertes Wort mit einem der Zeichen am Ende sein, die seinen Typ deklarieren (%, !, #).

Beispiel:

```
10 EXP = 5
```

ist nicht erlaubt, weil EXP ein reserviertes Wort ist. Jedoch ist

```
10 EXPONENT = 5
```

erlaubt, da EXP nur Teil des Variablennamens ist.

Bei einer Variablen, die mit FN beginnt, wird angenommen, dass es sich um den Aufruf einer benutzerdefinierten Funktion handelt. (Siehe Anweisung "DEF FN" in Kapitel 4.)

2.5.2. Definition von Variablentypen

Der Name der Variablen bestimmt ihren Typ (Zeichenkette oder numerisch und falls numerisch, welche Genauigkeit).

Zeichenkettenvariablen besitzen als letztes Zeichen das Waehrungszeichen $.

Beispiel:

```
A$ = "VERKAUFSBERICHT"
```

Das Waehrungszeichen ist ein Definitionszeichen fuer den Variablentyp. Es definiert eine Zeichenkettenvariable. Sonder-

zeichen, die den Typ der Variablen kennzeichnen, sind als letztes Zeichen des Namens erlaubt - siehe naechster Abschnitt.

- Variablen groesserer Genauigkeit benoetigen mehr Platz im Hauptspeicher.

- Der Computer benoetigt mehr Zeit fuer Berechnungen mit Zahlen groesserer Genauigkeit. Ein Programm mit wiederholten Berechnungen laeuft mit ganzzahligen Variablen schneller.

Die Definitionszeichen fuer den Typ der numerischen Variablen und die Anzahl der Bytes, die fuer jeden Typ benoetigt werden, sind folgende:

% Ganzzahlige Variable (2 Bytes)

! Variable einfacher Genauigkeit (4 Bytes)

Variable doppelter Genauigkeit (8 Bytes)

Ist der Variablentyp nicht explizit definiert, wird standardmaessig einfache Genauigkeit angenommen.

Beispiele von Variablennamen in BASIC:

PI#	definiert einen Wert doppelter Genauigkeit
MINIMUM!	definiert einen Wert einfacher Genauigkeit
LIMIT%	definiert einen ganzzahligen Wert
N$	definiert eine Zeichenkette
ABC	zeigt einen Wert mit einfacher Genauigkeit

Es gibt noch eine zweite Methode, mit der der Typ der Variablen definiert werden kann. Die BASIC-Anweisungen **DEFINT**, **DEFSNG**, **DEFDBL** und **DEFSTR** koennen in ein Programm eingefuegt werden, um den Typ bestimmter Variablennamen zu definieren. Diese Anweisungen werden unter "DEFtyp-Anweisungen" in Kapitel 4 beschrieben.

2.5.3. Feldvariable

Ein Feld ist eine Gruppe oder eine Tabelle aus Werten, die ueber denselben Namen angesprochen werden koennen. Jeder einzelne Wert des Feldes wird **Element** genannt. Feldelemente sind Variablen und koennen in Ausdruecken und in jeder BASIC-Anweisung oder Funktion, die diese Variablen benutzt, verwendet werden.

Die Definition des Namens und des Typs eines Feldes und das Setzen der Anzahl der Elemente und ihr Aufeinanderfolgen in dem Feld wird **Definieren** oder **Dimensionieren** des Feldes genannt. Dies wird gewoehnlich mit Hilfe der Anweisung DIM getan.

Beispiel:

```
10 DIM B$(5)
```

Damit wird ein eindimensionales Feld mit dem Namen B$ erzeugt. Alle seine Elemente sind Zeichenketten variabler Laenge. Den Elementen ist nichts zugeordnet.

```
20 DIM A(2,3)
```

Damit wird ein zweidimensionales Feld mit dem Namen A erzeugt. Da sich an dem Namen kein Definitionszeichen fuer den Typ befindet, besteht das Feld aus Werten einfacher Genauigkeit. Zu Anfang sind alle Feldelemente auf Null gesetzt.

Jedes Feldelement wird mit dem Feldnamen und Index angesprochen. Ein Feldvariablenname hat so viele Indizes, wie das Feld Dimensionen hat.

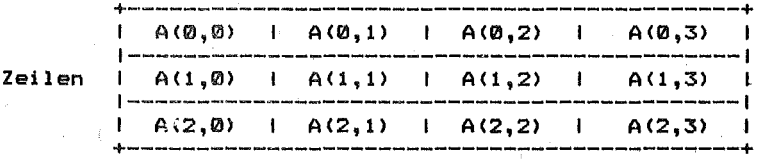
Der Index zeigt die Position des Elements im Feld an. Null ist die niedrigste Position, wenn sie nicht explizit geaendert wird (siehe Anweisung "OPTION BASE" in Kapitel 4). Die maximale Anzahl der Dimensionen eines Feldes ist 255. Die maximale Anzahl von Elementen pro Dimension ist 32767.

Um die oberen Beispiele fortzusetzen, kann man sich B$ als eine Liste von Zeichenketten wie folgt vorstellen:

```
+---------+
| B$(0)   |
|---------|
| B$(1)   |
|---------|
| B$(2)   |
|---------|
| B$(3)   |
|---------|
| B$(4)   |
|---------|
| B$(5)   |
+---------+
```

Die erste Zeichenkette in der Liste hat den Namen B$(0).

Das Feld A kann man sich als Tabelle aus Zeilen und Spalten wie folgt vorstellen:



Das Element der zweiten Zeile und ersten Spalte hat den Namen A(1,0).

Wird ein Feldelement benutzt, bevor das Feld definiert wurde, kann der Maximalwert der Indizes 10 betragen.

Findet BASIC z.B. die Anweisung

```
50 SIS(3) = 500
```

und das Feld SIS wurde zuvor nicht definiert, so wird es auf ein eindimensionales Feld mit 11 Elementen gesetzt, numeriert SIS(0) bis SIS(10). Dies ist die einzige Methode, um ein eindimensionales Feld implizit zu definieren.

Beispiel:

```
Ok
10 DIM TABELLE(3,4)
20 TABELLE(2,3) = 1982
30 FOR ZEILE = 0 TO 3
40 FOR SPALTE = 0 TO 4
50 PRINT TABELLE(ZEILE,SPALTE);
60 NEXT SPALTE
70 PRINT
80 NEXT ZEILE
RUN

 0  0  0  0  0
 0  0  0  0  0
 0  0  0  1982  0
 0  0  0  0  0
Ok
```

2.6. Typkonvertierung

Falls noetig, wandelt BASIC Zahlen einer Genauigkeit in eine andere Genauigkeit um. Dafuer gelten folgende Regeln:

1. Wird der numerische Wert einer Genauigkeit einer numerischen Variablen einer anderen Genauigkeit zugeordnet, wird die Zahl in der Genauigkeit gespeichert, die der Variablenname hat.

Beispiel:

```
 Ok
10 A% = 23.42
20 PRINT A%
RUN
 23
Ok
```

2. Runden, im Unterschied zum Abschneiden, passiert, wenn ein Wert hoeherer Genauigkeit einer Variablen mit niedrigerer Genauigkeit zugeordnet wird (Beispiel: Aendern von doppelter in einfache Genauigkeit).

Beispiel:

```
Ok
10 C = 55.8834567#
20 PRINT C
RUN
 55.88346
Ok
```

Dies betrifft nicht nur Zuordnungsanweisungen (zB. I% = 2.5 ergibt I% = 3), sondern betrifft auch Funktions- und Anweisungsberechnungen (z.B.TAB(4.5) ergibt die fuenfte Position, A(1.5) ist das gleiche wie A(2) und X = 11.5 MOD 4 ergibt einen Wert von 0 fuer X).

3. Wird eine Zahl einfacher Genauigkeit in eine Zahl hoeherer Genauigkeit umgewandelt, kann die sich ergebende Zahl hoeherer Genauigkeit nicht genauer sein als die Zahl mit niedrigerer Genauigkeit. Wird zB. ein Wert einfacher Genauigkeit A, einer Variablen doppelter Genauigkeit B#, zugeordnet, sind nur die ersten sechs Ziffern von B# genau. Das kommt daher, dass von A nur sechs Ziffern Genauigkeit geliefert wurden. Der Fehler kann mit der folgenden Formel eingegrenzt werden:

```
ABS(B#-A) < 6.3E-8*A
```

Das heisst, der absolute Wert der Differenz zwischen der gedruckten Zahl doppelter Genauigkeit und dem urspruenglichen Wert einfacher Genauigkeit ist kleiner als 6.3E-8 mal dem urspruenglichen Wert einfacher Genauigkeit.

Beispiel:

```
Ok
10 A = 2.04
20 B# = A
30 PRINT A; B#
RUN
 2.04  2.039999961853027
```

4. Waehrend der Berechnung der Ausdruecke werden alle Operanden einer arithmetischen oder logischen Operation in die gleiche Genauigkeit umgewandelt, und zwar in die des genauesten Operanden. Auch das Ergebnis einer arithmetischen Operation wird in dieser Genauigkeit uebertragen.

Beispiele:

```
Ok
10 D# = 6#/7
20 PRINT D#
RUN
 .8571428571428571
Ok
```

Die Berechnung wurde in doppelter Genauigkeit durchgefuehrt und das Ergebnis wurde als ein Wert doppelter Genauigkeit an D# uebergeben.

```
Ok
10 D = 6#/7
20 PRINT D
RUN
 .8571429
Ok
```

Die Berechnung wurde in doppelter Genauigkeit durchgefuehrt. Das Ergebnis wurde an D uebergeben (Variable einfacher Genauigkeit), gerundet und als Wert einfacher Genauigkeit ausgegeben.

5. Logische Operatoren (siehe "Logische Operatoren" in diesem Kapitel) wandeln ihre Operanden in ganzzahlige Werte um und uebergeben ein ganzzahliges Ergebnis. Die Operanden muessen im Bereich -32768 bis 32767 liegen, sonst wird der Fehler Overflow (Ueberlauf) angezeigt.

2.7. Numerische Ausdruecke und Operatoren

Ein numerischer Ausdruck kann eine numerische Konstante oder Variable sein oder kann dazu benutzt werden, Konstanten und Variablen mit Hilfe von Operatoren zu kombinieren, um daraus einen einfachen numerischen Wert zu errechnen.
Die numerischen Operatoren fuehren mathematische oder logische Operationen meistens mit numerischen Werten und manchmal mit Werten aus Zeichenketten durch. Man nennt sie "numerische" Operatoren, weil sie einen Wert ergeben, der aus einer Zahl besteht. Die numerischen Operatoren von BASIC kann man wie folgt in Kategorien einteilen:

- Arithmetik
- Vergleich
- Logik
- Funktionen

2.7.1. Arithmetische Operatoren

Die arithmetischen Operatoren in der Reihenfolge ihrer Prioritaet sind:

Operator	Operation	Beispiel-Ausdruck
^	Potenzierung	X^Y
-	Negation	-X
*,/	Multiplikation,	X*Y
	Gleitkommadivision	X/Y
\	Ganzzahlige Division	X\Y
MOD	Modulo-Arithmetik	X MOD Y
+,-	Addition, Subtraktion	X+Y,X-Y

Die ganzzahlige Division und Modulo-Arithmetik sollen etwas genauer erlaeutert werden.

Ganzzahlige Division:
Die ganzzahlige Division wird durch den umgekehrten Schraegstrich (\) dargestellt. Die Operanden werden zu ganzen Zahlen gerundet, bevor die Division ausgefuehrt wird und muessen sich im Bereich -32768 bis 32767 befinden.

Beispiel:

```
Ok
10 A = 10\4
20 B = 25.68\6.99
30 PRINT A;B
40 RUN
 2  3
Ok
```

Modulo-Arithmetik:
Modulo-Arithmetik wird durch den Operator MOD gekennzeichnet. Dabei wird ein ganzzahliger Wert als Rest der ganzzahligen Division zurueckgegeben.

Beispiel:

```
Ok
10 A = 7 MOD 4
20 PRINT A
RUN
 3
Ok
```

Das Ergebnis resultiert aus 7\4 gleich 1 Rest 3.

```
Ok
10 PRINT 25.68 MOD 6.99
RUN
 5
Ok
```

Das Ergebnis ist 5, weil 26\7 gleich 3 mit dem Rest 5 ist. (Dabei muss man sich daran erinnern, dass BASIC rundet, wenn in ganze Zahlen umgewandelt wird.)

2.7.2. Vergleichsoperatoren

Vergleichsoperatoren werden dazu benutzt, zwei Werte zu vergleichen. Die Werte koennen entweder beide numerisch oder beide Zeichenketten sein. Das Ergebnis eines Vergleichs ist entweder "wahr" (-1) oder "falsch" (0). Normalerweise wird das Ergebnis dazu benutzt, den Programmablauf zu aendern. (Siehe Anweisung "IF" in Kapitel 4.)

Operator	Getesteter Vergleich	Beispiel-Ausdruecke
=	Gleich	X=Y
<> oder ><	Ungleich	X<>Y
		X><Y
<	Kleiner als	X<Y
>	Groesser als	X>Y
<= oder =<	Kleiner als oder	X<=Y
	gleich	X=<Y
>= oder =>	Groesser als oder	X>=Y
	gleich	X=>Y

(Das Gleichheitszeichen wird auch benutzt, um einer Variablen einen Wert zuzuordnen. Siehe Anweisung "LET" in Kapitel 4.)

Numerische Vergleiche:
Treten arithmetische und Vergleichsoperatoren im gleichen Ausdruck auf, werden die arithmetischen immer zuerst ausgefuehrt. ZB. ist der Ausdruck

```
X + Y < (T-1)/Z
```

wahr (-1), wenn der Wert von X plus Y kleiner als der Wert von T-1 geteilt durch Z ist.

Weitere Beispiele:

```
Ok
10 X = 100
20 IF X<>200 THEN PRINT "NICHT GLEICH" ELSE PRINT "GLEICH"
RUN
NICHT GLEICH
Ok
```

Hier ist der Vergleich wahr (100 ist nicht gleich 200). Da das Ergebnis wahr ist, wird der THEN Teil der Anweisung IF ausgefuehrt.

```
Ok
PRINT 5<2; 5<10
 0 -1
Ok
```

Hier ist das erste Ergebnis falsch (Null), da 5 nicht kleiner als 2 ist. Das zweite Ergebnis ist -1, weil es wahr ist.

Zeichenkettenvergleiche:
Zeichenkettenvergleiche kann man sich als alphabetische Vergleiche vorstellen. Das heisst, eine Zeichenkette ist kleiner als eine andere, wenn die erste Zeichenkette vor der anderen im Alphabet steht. Kleinbuchstaben sind groesser als ihre zugehoerigen Grossbuchstaben. Zahlen sind kleiner als Buchstaben.
Zwei Zeichenketten werden so verglichen, indem jedesmal von jeder Zeichenkette ein Zeichen genommen wird und die ASCII-Codes verglichen werden. (Siehe Anlage D, "ASCII-Zeichencodes" fuer eine vollstaendige Liste aller ASCII-Codes.) Sind alle ASCII-Codes dieselben, so sind die Zeichenketten gleich. Sind die ASCII-Codes verschieden, so ist die Zeichenkette mit der niedrigeren Code-Nummer kleiner als die Zeichenkette mit der hoeheren Code-Nummer. Wird waehrend eines Zeichenkettenvergleichs das Ende einer Zeichenkette erreicht, ist die kuerzere Zeichenkette kleiner.
Fuehrende und nachfolgende Leerzeichen werden beachtet. Zum Beispiel sind alle folgenden Vergleichsausdruecke wahr (d. h. das Ergebnis der Vergleichsoperation ist -1).

```
"AA" < "AB"
"DATEINAME" = "DATEINAME"
"X&" > "X#"
"WR "> "WR"
"kg" > "KG"
"SCHMID" < "SCHMIED"
BR > "718"     (wobei BR = "12543")
```

Alle Zeichenkettenkonstanten, die in Vergleichsausdruecken benutzt werden, muessen in Anfuehrungszeichen eingeschlossen sein.

2.7.3. Logische Operatoren

Mit logischen Operatoren werden logische oder Bool'sche Operationen mit numerischen Werten durchgefuehrt. So wie gewoehnlich mit Vergleichsoperatoren Entscheidungen ueber den Programmablauf getroffen werden, werden logische Operatoren normalerweise dazu benutzt, zwei oder mehr Beziehungen zu verbinden und einen Wert fuer wahr oder falsch zurueckzugeben, der in einer Entscheidung benutzt wird. (Siehe Anweisung "IF" in Kapitel 4.)

Ein logischer Operator nimmt eine Kombination aus Wahr-Falsch-Werten und gibt als Ergebnis einen Wert als wahr oder falsch zurueck. Ein Operand eines logischen Operators wird als wahr angenommen, wenn er nicht Null ist (wie -1, die von einem Vergleichsoperator uebergeben wird) oder falsch, wenn er gleich Null ist. Das Ergebnis einer logischen Operation ist wieder eine Zahl, die wahr ist, wenn sie nicht gleich Null ist oder falsch, wenn sie gleich Null ist.
Die Zahl wird berechnet, indem die Operation Bit fuer Bit durchgefuehrt wird. Dies wird nachfolgend in allen Einzelheiten erklaert.

Die logischen Operatoren sind **NOT** (logisches Komplement), **AND** (Konjunktion), **OR** (Disjunktion), **XOR** (exklusives Oder), **IMP** (Implikation) und **EQV** (Aequivalenz).

Jeder Operator gibt die Werte zurueck, die in den folgenden Tabellen angezeigt sind ("T" steht fuer wahr oder einen Wert ungleich Null. "F" steht fuer falsch oder Wert Null). Die Operatoren sind in ihrer Rangfolge aufgelistet.

NOT

X	NOT X
T	F
F	T

AND

X	Y	X AND Y
T	T	T
T	F	F
F	T	F
F	F	F

OR

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

XOR

X	Y	X XOR Y
T	T	F
T	F	T
F	T	T
F	F	F

EQV

X	Y	X EQV Y
T	T	T
T	F	F
F	T	F
F	F	T

IMP

X	Y	X IMP Y
T	T	T
T	F	F
F	T	T
F	F	T

Es folgen einige Beispiele, wie logische Operatoren fuer Entscheidungen benutzt werden:

```
IF ER>60 AND SIE<20 THEN 1000
```

Hier ist das Ergebnis wahr, falls die Variable ER groesser als 60 ist und auch der Wert fuer SIE kleiner als 20 ist.

```
IF I>10 OR K<0 THEN 50
```

Das Ergebnis ist wahr, falls entweder I groesser als 10 oder K kleiner als 0 ist oder beides.

```
50 IF NOT (P=-1) THEN 100
```

Hier verzweigt das Programm zu Zeile 100, falls P nicht -1 ist. NOT (P=-1) ergibt nicht das gleiche Ergebnis wie NOT P. Siehe naechster Abschnitt "Arbeitsweise logischer Operatoren".

```
100 FLAG% = NOT FLAG%
```

Dies ist eine Moeglichkeit, einen Wert laufend von wahr auf falsch und falsch auf wahr umzuwandeln.

Arbeitsweise logischer Operatoren: Die Operanden werden in ganze Zahlen im Bereich -32768 bis +32767 umgewandelt. Befinden sich die Operanden nicht in diesem Bereich, wird der Fehler "Overflow" (Ueberlauf) angezeigt. Ist der Operand negativ, wird das Zweierkomplement benutzt. Jeder Operand wird in einer Folge von 16 Bits dargestellt. Die Operation wird an diesen Folgen ausgefuehrt. Das bedeutet, jedes Bit des Ergebnisses ist das Ergebnis der zwei korrespondierenden Bits in den beiden Operanden, gemaess der Tabellen fuer die Operatoren, die vorher aufgelistet wurden. Das Bit 1 bedeutet wahr und das Bit 0 bedeutet falsch.

Man kann also logische Operatoren dazu benutzen, eine bestimmte Bit-Folge zu testen. Zum Beispiel kann der Operator AND dazu benutzt werden, alle bis auf ein Bit des Status-Bytes eines Maschinenein-/-ausgabeanschlusses zu maskieren.

Die folgenden Beispiele zeigen, wie die logischen Operatoren arbeiten.

```
A = 63 AND 16
```

Hier wird A auf den Wert 16 gesetzt. Da 63 = binaer 111111 und 16 = binaer 10000 ist, ergibt 63 AND 16 010000 binaer, was wiederum 16 ist.

```
B = -1 AND 8
```

B wird auf den Wert 8 gesetzt. Da -1 = binaer 11111111 11111111 und 8 = binaer 1000 ist, ergibt -1 AND 8 binaer 00000000 00001000 oder 8.

```
C = 4 OR 2
```

Hier wird C auf den Wert 6 gesetzt. Da 4 = binaer 100 und 2 = binaer 010 ist, ergibt 4 OR 2 = binaer 110 oder 6.

```
X = 2
ZWEICOMP = (NOT X) +1
```

Dieses Beispiel zeigt, wie man das Zweierkomplement einer Zahl erstellen kann. X ist 2 oder 10 binaer. NOT X ist dann binaer 11111111 11111101, was dezimal -3 bedeutet. -3 plus 1 ist -2, das Komplement von 2. Das bedeutet, das Zweierkomplement einer ganzen Zahl ist das Bit-Komplement plus 1.

Dabei muss beachtet werden, dass, falls beide Operanden 0 oder -1 sind, ein logischer Operator 0 oder -1 uebergibt.

2.7.4. Numerische Funktionen

Eine Funktion wird wie eine Variable in einem Ausdruck benutzt, um eine vorbestimmte Operation mit einem oder mehreren Operanden auszufuehren. BASIC besitzt eingebaute Funktionen, die im System enthalten sind, wie zB. SQR (Quadratwurzel) oder SIN (Sinus). Alle eingebauten BASIC-Funktionen sind im Kapitel 5, "BASIC-Funktionen" beschrieben.

Es ist auch moeglich, eigene Funktionen mit Hilfe der Anweisung DEF FN zu definieren. Siehe Anweisung "DEF FN" in Kapitel 4.

2.7.5. Reihenfolge der Abarbeitung

Die Reihenfolge der Abarbeitung von numerischen Ausdruecken ergibt sich aus der Prioritaet der numerischen Operationen:

1. Funktionsaufrufe werden zuerst berechnet.

2. Arithmetische Operationen werden danach in dieser Reihenfolge ausgefuehrt:
   a. ^
   b. - (Minusvorzeichen)
   c. *,/
   d. \
   e. MOD
   f. +,-

3. Vergleichsoperationen werden danach ausgefuehrt.

4. Logische Operationen werden zuletzt in dieser Reihenfolge ausgefuehrt:
   a. NOT
   b. AND
   c. OR
   d. XOR
   e. EQV
   f. IMP

Operationen auf derselben Ebene in der Liste werden von links nach rechts ausgefuehrt. Soll die Reihenfolge der Ausfuehrung einer Operation geaendert werden, muessen Klammern benutzt werden. Operationen in Klammern werden zuerst ausgefuehrt. Innerhalb der Klammern gilt die gewoehnliche Rangordnung der Ausfuehrung von Operationen.

Es folgen einige algebraische Ausdruecke und ihre Darstellung in BASIC.

Algebraischer Ausdruck	BASIC-Ausdruck
$X+2Y$	X+Y*2
$X-\frac{Y}{Z}$	X-Y/Z
$\frac{XY}{Z}$	X*Y/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X^2)^Y
X^{Y^Z}	X^(Y^Z)
$X(-Y)$	X*(-Y)

Hinweis:

Zwei aufeinanderfolgende Operatoren muessen, wie im letzten Beispiel gezeigt, durch Klammern getrennt werden.

2.8. Zeichenkettenausdruecke und Operatoren

Ein Zeichenkettenausdruck kann eine Zeichenkettenkonstante oder Variable sein oder er kann Konstanten und Variablen mit Hilfe von Operatoren kombinieren, um eine neue Zeichenkette zu erstellen.

Zeichenkettenoperatoren werden benutzt, um Zeichenketten auf verschiedene Weise aneinanderzufuegen. Es gibt nur zwei Zeichenkettenoperatoren:

- Verkettung

- Funktionen

Werden Vergleichsoperatoren =, <>, <, >, <= und >= benutzt, um Zeichenketten zu vergleichen, sind sie nicht als Zeichenkettenoperatoren zu betrachten, da sie ein numerisches Ergebnis und nicht ein Zeichenkettenergebnis bringen. Der Abschnitt "Vergleichsoperatoren" in diesem Kapitel gibt eine Erklaerung, wie Zeichenketten mit Hilfe von Vergleichsoperatoren verglichen werden koennen.

2.8.1. Verkettung

Werden zwei Zeichenketten aneinandergehaengt, nennt man das **Verkettung.** Zeichenketten werden mit dem Plussymbol (+) verkettet.

Beispiel:

```
Ok
10 KOMBINAT$ = "ROBOTRON-"
20 TYPE$ = "EPC"
30 PRINT KOMBINAT$ + TYPE$
RUN
ROBOTRON-EPC
Ok
```

2.8.2. Zeichenkettenfunktionen

Eine Zeichenkettenfunktion funktioniert wie eine numerische Funktion, nur dass sie als Ergebnis eine Zeichenkette uebergibt. Eine Zeichenkettenfunktion wird in einem Ausdruck benutzt, um eine vorbestimmte Operation aufzurufen, die mit einer oder mehreren Operanden ausgefuehrt werden soll. BASIC besitzt "eingebaute" Funktionen, die sich im System befinden, wie zB. MID$, die eine Zeichenkette aus der Mitte einer anderen Zeichenkette uebergibt, oder CHR$, die das Zeichen fuer den angegebenen ASCII-Code uebergibt. Alle eingebauten BASIC-Funktionen werden in Kapitel 5 beschrieben.

Es ist moeglich, eigene Funktionen mit Hilfe der Anweisung DEF FN zu definieren. Siehe Anweisung "DEF FN" in Kapitel 4.

3. Kommandos

In den nachfolgenden Kapiteln wurden in der Syntax einige Parameter wie folgt abgekuerzt:

x, y, z	stellen numerische Ausdruecke dar
i, j, k, m, n	stellen ganzzahlige Ausdruecke dar
x$, y$	stellen Zeichenkettenausdruecke dar
v, v$	stellen numerische bzw. Zeichenkettenvariablen dar

Werden Werte einfacher oder doppelter Genauigkeit uebergeben, wo ein einfacher Wert benoetigt wird, rundet BASIC den Teil hinter dem Dezimalpunkt und benutzt die daraus resultierende ganze Zahl.

3.1. Editierkommandos

3.1.1. NEW

Loescht das im Hauptspeicher befindliche Programm und alle Variablen.

Syntax: **NEW**

Bemerkungen:

Wird benutzt, um den Hauptspeicher zu loeschen, bevor ein neues Programm eingegeben wird. BASIC kehrt immer in die Befehlsebene zurueck, nachdem **NEW** ausgefuehrt wurde. Durch **NEW** werden alle Dateien geschlossen und die Programmablaufverfolgung ausgeschaltet, falls sie eingeschaltet war (siehe Kommandos "TRON und TROFF" in diesem Kapitel).
NEW kann auch als Programmanweisung genutzt werden.

Beispiel:

```
Ok
NEW
Ok
```

Das Programm, das sich im Hauptspeicher befand, ist jetzt geloescht.

3.1.2. AUTO

Automatische Zeilennummergenerierung, jedesmal nachdem die Taste <ENTER> gedrueckt wurde.

Syntax: AUTO [Nummer][,[Schrittweite]]

Bemerkungen:

Nummer — Nummer, die fuer den Beginn der Zeilennummerierung benutzt werden soll. Ein Punkt (.) kann anstelle der Zeilennummer benutzt werden, um die gerade eingegebene Zeile anzuzeigen.

Schrittweite — Der jeder Zeilennummer hinzugefuegte Wert, um die naechstfolgende Zeilennummer zu erhalten.

Die Zeilennumerierung beginnt mit **Nummer** und wird fuer jede folgende Zeilennummer mit der **Schrittweite** erhoeht. Werden beide Werte weggelassen, ist die Standardannahme 10,10. Folgt der **Nummer** ein Komma (,), aber die **Schrittweite** ist weggelassen, wird die letzte fuer den Befehl **AUTO** angegebene **Schrittweite** angenommen. Wird **Nummer** weggelassen, aber die **Schrittweite** angegeben, beginnt die Zeilennumerierung mit 0 (Null).

AUTO wird normalerweise benutzt, um Programme einzugeben. Damit ist es nicht notwendig, jede Zeilennummer einzugeben.

Erzeugt **AUTO** eine Zeilennummer, die schon im Programm existiert, wird nach der Nummer ein Stern (*) ausgegeben, um anzuzeigen, dass eine Eingabe die existierende Zeile ersetzen wird.

Drueckt man jedoch direkt nach dem * die Taste <ENTER>, wird die bestehende Zeile nicht ersetzt und **AUTO** erzeugt die naechste Zeilennummer.

AUTO wird durch Betaetigen der Taste <CTRL> und der Taste <PAUSE> beendet. Die Zeile, bei der diese Tasten betaetigt wurden, wird nicht gespeichert. Danach kehrt BASIC in die Befehlsebene zurueck.

Hinweis:

- Im **AUTO**-Modus koennen nur Aenderungen innerhalb der Eingabezeile vorgenommen werden. Sollen andere Zeilen des Bildschirms geaendert werden, muss man vorher **AUTO** durch Betaetigen der Tasten <CTRL-PAUSE> verlassen.

- Zwischen 2 Programmzeilen sollten sich immer genuegend freie Zeilen befinden, um eventuelle Programmzeilen einfuegen zu koennen. Eine Schrittweite < 5 sollte man deshalb nicht verwenden.

Beispiel:

```
AUTO
```

Dieser Befehl generiert die Zeilennummern 10, 20, 30, 40, ...

```
AUTO 100,50
```

Es werden die Zeilennummern 100, 150, 200, ... generiert.

```
AUTO 500,
```

Es werden die Zeilennummern 500, 550, 600, 650, ... generiert. Die Schrittweite ist 50, weil die Schrittweite des vorherigen Befehls **AUTO** auch 50 war.

```
AUTO ,20
```

Es werden die Zeilennummern 0, 20, 40, 60, ... generiert.

3.1.3. EDIT

Zeigt eine Zeile fuer eine Aenderung an.

Syntax: **EDIT Zeile**

Bemerkungen:

Zeile Zeilennummer einer sich im Programm befindenden Zeile. Gibt es diese Zeile nicht, erhaelt man den Fehler "Undefined Line Number" (Nicht definierte Zeilennummer). Es kann ein Punkt (.) angegeben werden, wenn die Eingabezeile angezeigt werden soll.

Das Kommando **EDIT** zeigt einfach die angegebene Zeile an und stellt den Kursor unter die erste Ziffer der Zeilennummer. Jetzt kann die Zeile, wie unter Kapitel 1.3 beschrieben, veraendert werden.

Ein Punkt (.) kann benutzt werden, um die laufende Eingabezeile anzuzeigen. Wurde z.B. gerade eine Zeile eingegeben und diese Zeile soll geaendert werden, zeigt der Befehl **EDIT.** diese Zeile fuer eine Aenderung an.

Mit **LIST** koennen auch Programmzeilen fuer Aenderungen angegeben werden. Siehe Kommando "LIST" in diesem Kapitel.

3.1.4. LIST

Listet das im Hauptspeicher befindliche Programm auf dem Bildschirm oder auf einer anderen angegebenen Einheit auf.

Syntax: LIST [Zeile1][-[Zeile2]][,Dateiangabe]

Bemerkungen:

Zeile1, Zeile2
Gueltige Zeilennummern im Bereich 0 bis 65529. **Zeile1** ist die erste Zeile, die aufgelistet wird. **Zeile2** ist die letzte Zeile, die aufgelistet wird. Ein Punkt (.) kann als Zeilennummer benutzt werden, um die laufende Zeile anzuzeigen.

Dateiangabe
Zeichenkettenausdruck fuer die Dateispezifikation, wie unter Kapitel 6 erklaert. Es kann ein Pfadname in dieser Angabe enthalten sein. Wird **Dateiangabe** weggelassen, werden die angegebenen Zeilen auf dem Bildschirm angezeigt.
Das Auflisten auf dem Bildschirm oder dem Drucker kann mit Hilfe der Tasten <CTRL-PAUSE> unterbrochen werden.
Wird der Zeilenbereich nicht angegeben, wird das gesamte Programm aufgelistet.
Wird der Bindestrich (-) im Zeilenbereich angegeben, ist folgendes moeglich:

- Wird nur **Zeile1** angegeben, werden diese Zeile und alle Zeilen mit hoeheren Nummern aufgelistet.

- Wird nur **Zeile2** angegeben, werden allen Zeilen von Beginn des Programmes bis zur **Zeile2** aufgelistet.

- Werden beide Zeilennummern angegeben, werden alle Zeilen von **Zeile1** bis **Zeile2** aufgelistet.

Erfolgt die Auflistung in eine Datei auf Diskette, wird der angegebene Teil des Programmes im ASCII-Format gespeichert. Diese Datei kann spaeter mit **MERGE** benutzt werden.
Nachdem **LIST** ausgefuehrt wurde, kehrt **BASIC** immer in die Befehlsebene zurueck.

Beispiel:

```
LIST
```

zeigt das gesamte Programm auf dem Bildschirm an.

```
LIST 35,"SCRN:"
```

zeigt Zeile 35 auf dem Bildschirm an.

```
LIST 10-20, "LPT1:"
```

listet die Zeilen 10 bis 20 auf dem Drucker auf.

```
LIST 100-, "COM1:1200,N,8"
```

uebertraegt alle Zeilen von Zeile 100 bis zum Ende des Programms zum ersten Datenfernverarbeitungsanschluss mit 1200 Bps, keine Paritaet, 8 Daten-Bits, 1 Stopp-Bit.

3.1.5. LLIST

Druckt das gesamte oder einen Teil des Programms, das sich im Hauptspeicher befindet, auf dem Drucker (LPT1:).

Syntax: **LLIST [Zeile1][-[Zeile2]]**

Bemerkungen:

Der Zeilennummernbereich fuer **LLIST** arbeitet wie **LIST**. BASIC steht in der Befehlsebene, nachdem **LLIST** ausgefuehrt wurde.

Beispiel:

```
LLIST
```

druckt eine Liste des gesamten Programms.

```
LLIST 35
```

druckt Zeile 35.

```
LLIST 10-20
```

listet die Zeilen 10 bis 20 auf dem Drucker auf.

```
LLIST 100-
```

druckt alle Zeilen ab Zeile 100 bis zum Ende des Programms.

```
LLIST -200
```

druckt ab der ersten Zeile bis zur Zeile 200.

3.1.6. DELETE

Loescht Programmzeilen

Syntax: **DELETE [Zeile1][-Zeile2]**
DELETE [Zeile1-]

Bemerkungen:

Zeile1 Zeilennummer der ersten zu loeschenden Zeile

Zeile2 Zeilennummer der letzten zu loeschenden Zeile

Der Befehl **DELETE** loescht den angegebenen Zeilenbereich aus dem Programm. BASIC kehrt immer zur Befehlsebene zurueck, nachdem **DELETE** ausgefuehrt wurde.

Anstelle der Zeilennummer kann ein Punkt (.) benutzt werden, um die laufende Zeile zu ersetzen. Wird eine Zeilennummer angegeben, die nicht im Programm existiert, erscheint der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Das Kommando **DELETE** kann auch als Programmanweisung genutzt werden, um z.B. Programmzeilen zu loeschen, die bei der weiteren Abarbeitung des Programms nicht mehr benoetigt werden.

Beispiel:

```
DELETE 40
```

Die Zeile 40 wird geloescht.

```
DELETE 40-100
```

Die Zeilen 40 bis 100 werden geloescht.

```
DELETE 40-
```

Alle Zeilen ab Zeile 40 bis zum Programmende werden geloescht.

```
DELETE -40
```

Alle Zeilen bis zu und einschliesslich Zeile 40 werden geloescht.

3.1.7. RENUM

Neue Numerierung von Programmzeilen.

Syntax: RENUM [neue Nummer][,[alte Nummer][,Schrittweite]]

Bemerkungen:

neue Nummer Erste Zeilennummer, die in der neuen Folge benutzt werden soll. Die Standardannahme ist 10.

alte Nummer Zeile im laufenden Programm, mit der die Neunumerierung beginnen soll. Die Standardannahme ist die erste Zeile des Programms.

Schrittweite Ist die Schrittweite, die in der neuen Folge benutzt werden soll. Die Standardannahme ist 10.

RENUM aendert auch automatisch alle Zeilennummerreferenzen, die auf GOTO-, GOSUB-, THEN-, ELSE-, ON...GOTO-, ON...GOSUB-, RESTORE-, RESUME- und ERL-Anweisungen folgen, um den neuen Zeilennummern zu entsprechen. Wird dabei eine Zeilennummer aufgerufen, die nicht existiert, wird die Nachricht "Undefined Line Number xxxxx in yyyyy" (Nicht definierte Zeilennummer xxxxx in yyyyy) gedruckt. Die ungueltige Zeilennummerreferenz (xxxxx) wird von RENUM nicht geaendert. Die Zeilennummer yyyyy kann aber geaendert sein.

Hinweis:

- Mit **RENUM** kann die Reihenfolge der Programmzeilen nicht veraendert oder Zeilennummern groesser als 65529 erzeugt werden. Es ergibt sich der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

- Man sollte regelmaessig nach Fertigstellung oder Aenderung von Programmen das **RENUM**-Kommando verwenden. Dadurch bleibt das Programm uebersichtlich und kann leicht geaendert werden.

Beispiel:

```
RENUM
```

Das gesamte Programm wird neu numeriert Die erste neue Zeile hat die Nummer 10. Die Zeilenschrittweite ist 10.

```
RENUM 300,,50
```

Das gesamte Programm wird neu numeriert. Die erste neue Zeilennummer ist 300. Die Zeilenschrittweite ist 50.

```
RENUM 1000,900,20
```

Die Zeilen ab 900 aufwaerts werden neu numeriert, so dass sie mit der Zeilennummer 1000 anfangen und eine Schrittweite von 20 haben.

3.2. Programmausfuehrungskommandos

3.2.1. RUN

Beginnt die Programmausfuehrung.

Syntax: **RUN [Zeile]**
RUN Dateiangabe[,R]

Bemerkungen:

Zeile Zeilennummer des Programms im Hauptspeicher, an der mit der Ausfuehrung begonnen werden soll.

Dateiangabe Zeichenkettenausdruck fuer eine Dateispezifikation, wie unter Kapitel 6 beschrieben. Es kann auch ein Pfadname enthalten sein.

RUN oder **RUN Zeile** beginnt mit der Ausfuehrung des im Hauptspeicher befindlichen Programms. Wird **Zeile** angegeben, beginnt die Ausfuehrung mit der angegebenen Zeilennummer; andernfalls beginnt die Ausfuehrung mit der niedrigsten Zeilennummer.

RUN Dateiangabe laedt eine Datei von Diskette in den Hauptspeicher und fuehrt sie aus. Alle eroeffneten Dateien werden geschlossen und der laufende Inhalt des Hauptspeichers wird geloescht, bevor das ausgewaehlte Programm geladen wird. Mit der Angabe R bleiben jedoch alle Datendateien eroeffnet.

Das Kommando **RUN** kann auch als Programmanweisung genutzt werden.

Beispiel:

```
Ok
10 PRINT 1/7
RUN
 .1428572
Ok

10 PI=3.141593
20 PRINT PI
RUN 20
 0
Ok
```

Im Beispiel wird die erste Form von **RUN** benutzt. Das erste Programm wird von Anfang an ausgefuehrt. Im zweiten Beispiel wird die Angabe **Zeile** mit RUN angegeben, um das Programm ab Zeile 20 auszufuehren. In diesem Fall wird Zeile 10 nicht ausgefuehrt, so dass PI nicht den richtigen Wert erhaelt. Eine 0 (Null) wird ausgegeben, weil alle numerische Variablen zu Anfang auf 0 gesetzt werden.

```
RUN "NEWFILE",R
```

Im Beispiel wird das Programm "NEWFILE" geladen und ausgefuehrt, wobei die Dateien geoeffnet bleiben.

3.2.2. CLEAR

Setzt alle numerische Variablen auf Null und loescht den Inhalt aller Zeichenkettenvariablen. Zusaetzlich kann die Groesse des Arbeitsspeichers und die Groesse eines Stapelbereiches definiert werden.

Syntax: CLEAR[,[n][,m]]

Bemerkungen:

n ist ein Bytezaehler, der - falls angegeben - die maximale Anzahl der Bytes fuer einen BASIC-Arbeitsbereich definiert (in dem das Programm und die Daten zusammen mit einem Arbeitsbereich fuer den Interpreter gespeichert sind). Der Standardwert fuer n ist 65535. Fuer n kann auch ein kleinerer Wert angegeben werden, um den gesamten adressierbaren Bereich zu verkleinern. Dadurch wird der im oberen Hauptspeicherbereich (nach dem BASIC-Datensegment) zur Verfuegung stehende Speicher groesser. n wird dann eingefuegt, wenn Hauptspeicherplatz fuer Programme in Maschinensprache reserviert werden soll, d.h. es entsteht ein geschuetzter Speicher, in den weitere vom Programm aufrufbare Moduln geladen werden koennen.

m setzt die Groesse des Stapelbereichs fuer BASIC. Der Standardwert ist 512 Bytes oder ein Achtel des verfuegbaren Hauptspeichers (der kleinere der beiden Werte). m wird eingefuegt, wenn in dem Programm sehr viele verschachtelte Anweisungen **GOSUB** oder Schleifen **FOR...NEXT** enthalten sind oder wenn **PAINT** benutzt wird, um komplexe Bilder darzustellen.

CLEAR loescht den gesamten fuer Daten benutzten Hauptspeicher, ohne das gerade im Hauptspeicher befindliche Programm zu loeschen. Nach **CLEAR** sind alle Bereiche nicht mehr definiert. Numerische Variablen haben einen Wert von Null, Zeichenkettenvariablen haben keinen Inhalt. Alle Informationen, die mit der Anweisung DEF gesetzt wurden, sind geloescht. (Dies beinhaltet DEF FN, DEF SEG, DEF USR, DEFINT, DEFDBL, DEFSNG und DEFSTR.)

Die Ausfuehrung des Befehls **CLEAR** schaltet jeden Ton ab und setzt auf Musikvordergrund zurueck. Das Kommando kann auch als Programmanweisung genutzt werden.

Die Anweisung **ERASE** kann verwendet werden, einigen Hauptspeicherplatz freizumachen, ohne damit alle Daten des Programms zu loeschen. Es werden nur angegebene Bereiche aus dem Arbeitsbereich geloescht. Siehe Anweisung "**ERASE**" in Kapitel 4.

Beispiel:

```
CLEAR
```

Es werden alle Daten aus dem Hauptspeicher geloescht (ohne das Programm zu loeschen).

```
CLEAR,32768
```

Es werden alle Daten geloescht und der maximale Arbeitsbereich auf 32K Bytes gesetzt.

```
CLEAR,,2000
```

Es werden die Daten geloescht, und die Groesse des Stapelbereichs auf 2000 Bytes gesetzt.

```
CLEAR,32768,2000
```

Im letzten Beispiel werden alle Daten geloescht, der maximale Arbeitsbereich fuer BASIC wird auf 32K Bytes und der Stapelbereich auf 2000 Bytes gesetzt.

3.2.3. CONT

Nimmt die Programmausfuehrung nach einer Unterbrechung wieder auf.

Syntax: CONT

Bemerkungen:

Das Kommando **CONT** kann dazu benutzt werden, die Programmausfuehrung wieder aufzunehmen, nachdem die Tasten **<CTRL-PAUSE>** betaetigt wurden, die Anweisungen **STOP** oder **END** ausgefuehrt wurden oder ein Fehler auftrat. Die Ausfuehrung wird an dem Punkt wieder aufgenommen, wo die Unterbrechung auftrat. Geschah die Unterbrechung nach dem Warten auf eine Eingabe fuer die Anweisung **INPUT**, wird mit der Neuanzeige der Anfrage fortgesetzt.

CONT wird normalerweise in Verbindung mit **STOP** fuer das Testen benutzt. Wird die Ausfuehrung angehalten, koennen die Werte von Variablen mit Hilfe von Direktmodusanweisungen geprueft oder geaendert werden. Danach kann mit **CONT** die Ausfuehrung erneut aufgenommen, oder mit der Direktmodusanweisung **GOTO** die Ausfuehrung an einer bestimmten Zeilennummer wieder aufgenommen werden.

CONT ist ungueltig, falls das Programm waehrend der Unterbrechung veraendert wurde.

Beispiel:

Im folgenden Beispiel wird eine Schleife erzeugt.

```
Ok
10 FOR A=1 TO 50
20 PRINT A;
30 NEXT A
RUN
 1  2  3  4  5  6  7  8  9  10  11  12
 13  14  15  16  17  18  19  20  21  22
 23  24  25  26  27  28  29
```

(An diesem Punkt wird die Schleife durch Betaetigen der Tasten <CTRL-PAUSE> unterbrochen).

```
:
:
:

Break in 20
Ok
CONT
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50
Ok
```

3.2.4. TRON und TROFF

Zeigt eine Ablaufverfolgung der Programmanweisungen bei der Ausfuehrung des Programms an.

Syntax: TRON
TROFF

Bemerkungen:

Als Hilfe beim Programmtest aktiviert das Kommando **TRON** (das in der indirekten Betriebsart eingegeben werden kann) eine Programmablaufverfolgung, d.h. jede Zeilennummer des Programms wird angezeigt und die zugehoerige Programmzeile ausgefuehrt. Die Nummer erscheint in eckigen Klammern. Durch das Kommando **TROFF** wird diese Ablaufverfolgung ausgeschaltet.

Beispiel:

```
10 K=10
20 FOR I=1 TO 2
30 L=K+10
40 PRINT I;K;L
50 K=K+10
60 NEXT
70 END
TRON
Ok
RUN

[10][20][30][40] 1  10  20
[50][60][30][40] 2  20  30
[50][60][70]
Ok
TROFF
Ok
```

In diesem Beispiel wird mit Hilfe von **TRON** und **TROFF** der Programmablauf bei der Ausfuehrung einer Schleife gezeigt. Die Nummern in eckigen Klammern sind die Zeilennummern. Die nicht in eckigen Klammern stehenden Zahlen am Ende jeder Zeile sind die Werte fuer I, K und L, die das Programm anzeigt.

3.2.5. SYSTEM

Verzweigt aus BASIC und kehrt zu DCP zurueck.

Syntax: SYSTEM

Bemerkungen:

Durch **SYSTEM** werden alle Dateien abgeschlossen, bevor zu DCP zurueckgekehrt wird. Das BASIC-Programm geht verloren.

Wurde zu BASIC durch eine Stapelverarbeitungsdatei in DCP verzweigt, verzweigt das Kommando **SYSTEM** zu der Stapelverarbeitungsdatei zurueck und faehrt mit der Verarbeitung an dem Punkt fort, an dem die Datei verlassen wurde.
Das Kommando kann auch als Programmanweisung genutzt werden.

3.3. Diskettenbezogene Kommandos

3.3.1. RESET

Schliesst alle Diskettendateien ab und loescht den Systempuffer.

Syntax: RESET

Bemerkungen:

Sind alle Dateien auf der Diskette eroeffnet, wirkt **RESET** wie **CLOSE** ohne Angabe von Dateinummern.

3.3.2. LOAD

Laedt ein Programm von einer angegebenen Einheit in den Hauptspeicher und fuehrt es wahlweise aus.

Syntax: LOAD Dateiangabe[,R]

Bemerkungen:

Dateiangabe Zeichenkettenausdruck fuer die Dateispezifikation, der den Regeln entsprechen muss, die unter Kapitel 6 beschrieben sind; andernfalls wird ein Fehler angezeigt und das Laden abgebrochen. Es kann ein Pfadname angegeben werden.

LOAD schliesst alle eroeffneten Dateien ab und loescht alle Variablen und Programmzeilen, die sich im Hauptspeicher befinden, bevor das angegebene Programm geladen wird. Wird die Angabe **R** weggelassen, geht BASIC in die direkte Betriebsart, nachdem das Programm geladen wurde.

Wird jedoch **R** im Kommando **LOAD** angegeben, wird das Programm nach dem Laden ausgefuehrt. In diesem Fall bleiben alle eroeffneten Dateien offen. Das bedeutet, dass **LOAD** mit der Angabe **R** dazu benutzt werden kann, mehrere Programme zu verketten (oder Segmente des gleichen Programms). Informationen koennen zwischen den Programmen mit Hilfe von Datendateien uebergeben werden.

LOAD Dateiangabe, R ist gleichbedeutend mit **RUN Dateiangabe.** Der Dateityp **.BAS** wird an den Dateinamen angefuegt, falls kein Dateityp angegeben ist und der Dateiname aus acht oder weniger Buchstaben besteht. Das Kommando **LOAD** ist auch als Programmanweisung nutzbar.

Beispiel:

```
LOAD "MENU"
```

laedt das Programm mit dem Namen MENU, fuehrt es aber nicht aus.

```
LOAD "INVENT",R
```

laedt das Programm INVENT und fuehrt es aus.

```
RUN "INVENT"
```

wirkt wie LOAD "INVENT",R.

```
LOAD "B:REPORT.BAS"
```

laedt die Datei REPORT.BAS vom Diskettenlaufwerk B (.BAS wird nicht angegeben).

3.3.3. SAVE

Speichert eine BASIC-Programmdatei auf Diskette.

Syntax:

```
SAVE Dateiangabe[,A]
SAVE Dateiangabe[,P]
```

Bemerkungen:

Dateiangabe Zeichenkettenausdruck fuer eine Dateispezifikation. Entspricht **Dateiangabe** nicht den Regeln, die unter Kapitel 6 beschrieben sind, wird ein Fehler angezeigt und das Speichern abgebrochen. Es kann ein Pfadname angegeben werden.

Enthaelt der Dateiname acht Zeichen oder weniger und es fehlt der Dateityp, wird automatisch als Dateityp .BAS an den Namen angefuegt. Steht schon eine Datei mit dem gleichen Namen auf der Diskette, wird sie ueberschrieben.

Mit der Angabe A wird das Programm im ASCII-Format gespeispeichert, andernfalls speichert BASIC die Datei in einem komprimierten Binaerformat. ASCII-Dateien benoetigen mehr Platz, aber einige Zugriffsarten benoetigen Dateien im ASCII-Format.

Eine Datei, die mit einem Programm gemischt werden soll, muss im ASCII-Format gespeichert sein (siehe Kommando MERGE). Programme, die im ASCII-Format gespeichert sind, koennen als Datendateien gelesen werden. Soll ein BASIC-Programm nachfolgend mit dem BASIC-Compiler uebersetzt werden, ist ebenfalls der Parameter A zu verwenden.

Mit der Angabe P koennen Programme geschuetzt gespeichert werden. Solche geschuetzten Programme koennen spaeter geladen und abgearbeitet werden. Jeder Versuch, sie mit LIST oder EDIT anzusprechen und damit einzusehen und zu veraendern ergibt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Es ist nicht moeglich, ein geschuetztes Programm wieder in den ungeschuetzten Status zu bringen.

Hinweis: Das Inhaltsverzeichnis fuer Disketten zeigt fuer BASIC-Programme nicht an, dass sie geschuetzt oder im ASCII-Format gespeichert sind.

Beispiel:

```
SAVE "INVENT"
```

Das Programm, das sich im Hauptspeicher befindet, wird unter dem Namen INVENT gespeichert und als Dateityp .BAS angefuegt.

```
SAVE "B:PROG",A
```

Das Programm PROG.BAS wird in Laufwerk B im ASCII-Format gespeichert, so dass es spaeter mit Programmen im Hauptspeicher gemischt werden kann.

```
SAVE "A:KUNDEN.DAT",P
```

Das Programm KUNDEN.DAT wird in Laufwerk A geschuetzt gespeichert, so dass es nicht veraendert werden kann.

3.3.4. MERGE

Mischt Zeilen aus einer ASCII-Programmdatei in ein Programm, das sich im Hauptspeicher befindet.

Syntax: MERGE Dateiangabe

Bemerkungen:

Dateiangabe Zeichenkettenausdruck fuer eine Dateispezifikation, der den Regeln fuer die Namensgebung von Dateien, wie unter Kapitel 6 erklaert, entsprechen muss; andernfalls wird ein Fehler angezeigt und MERGE abgebrochen. Es kann ein Pfadname angegeben werden.

Die Diskette wird nach dem Namen der Datei abgesucht. Sobald die Datei gefunden ist, werden die Programmzeilen in der Datei mit den Zeilen im Hauptspeicher vermischt. Haben Zeilen aus der Datei, mit denen vermischt werden soll, die gleichen Zeilennummern wie Zeilen des Programms im Hauptspeicher, ersetzen die Zeilen der Datei die Zeilen im Hauptspeicher, d.h., die urspruenglichen Zeilen werden durch die neuen Zeilen ersetzt.

Nach der Ausfuehrung des Befehls MERGE bleibt das vermischte Programm im Hauptspeicher und BASIC kehrt zur Befehlsebene zurueck.

Wurde das Programm, das in das Hauptprogramm eingemischt werden soll, nicht im ASCII-Format gespeichert (mit Hilfe der Angabe A im Kommando SAVE), erhaelt man den Fehler "Bad File Mode" (Falscher Dateityp). Das Programm im Hauptspeicher wird nicht veraendert.

Beispiel:

```
MERGE "A:NUMMER"
```

Damit wird die Datei mit dem Namen NUMMER auf Laufwerk A mit dem Programm im Hauptspeicher vermischt.
Die Datei "NUMMER" muss eine ASCII-Datei sein (Kommando SAVE, Option A).

3.3.5. NAME

Aendert den Namen einer Diskettendatei. Der Befehl **NAME** in BASIC ist aehnlich dem Befehl **RENAME** in DCP.

Syntax: **NAME Dateiangabe AS Dateiname**

Bemerkungen:

Dateiangabe Dateispezifikation, wie unter Kapitel 6 beschrieben. Es kann auch ein Pfadname enthalten sein.

Dateiname Neuer Dateiname. Es muss sich um einen gueltigen Dateinamen handeln, wie in diesem Abschnitt beschrieben.

Die in **Dateiangabe** angegebene Datei muss existieren, aber **Dateiname** darf nicht auf der Diskette vorhanden sein, sonst erhaelt man einen Fehler. Es muss beachtet werden, dass als Dateityp nicht standardmaessig **.BAS** angehaengt wird.

Nach der Ausfuehrung des Befehls **NAME** steht die Datei auf derselben Diskette im gleichen Bereich unter einem neuen Namen. Das Kommando kann auch als Programmanweisung verwendet werden.

Beispiel:

```
NAME "A:TEST.BAS" AS "TEST1.BAS"
```

Nach der Ausfuehrung dieses Beispiels hat die Datei TEST.BAS auf der Diskette im Laufwerk A den Namen TEST1.BAS.

3.3.6. FILES

Zeigt die Dateinamen auf einer Diskette an. Das Kommando **FILES** in BASIC ist aehnlich dem Befehl **DIR** in DCP.

Syntax: **FILES [Dateiangabe]**

Bemerkungen:

Dateiangabe Zeichenkettenausdruck fuer eine Dateispezifikation, wie in Kapitel 6 erklaert. Ist **Dateiangabe** weggelassen, werden alle Dateien im DCP-Standardlaufwerk angelistet.

Alle Dateien, die dem Dateinamen entsprechen, werden angezeigt. Der Dateiname kann Fragezeichen (?) enthalten. Ein Fragezeichen entspricht jedem Zeichen des Namens oder des Dateitypes. Ein Stern (*) als erstes Zeichen des Namens oder des Dateityps entspricht jedem Namen oder jedem Dateityp.

Ist ein Laufwerk als Teil der **Dateiangabe** spezifiziert, werden die Dateien auf der Diskette in diesem Laufwerk, die dem angegebenen Dateinamen entsprechen, aufgelistet; andernfalls wird das DCP-Standardlaufwerk angenommen.

Beispiel:

```
FILES
```

Damit werden alle Dateien im aktuellen Verzeichnis des DCP-Standarddiskettenlaufwerkes angezeigt.

```
FILES "*.BAS"
```

Damit werden alle Dateien mit dem Dateityp .BAS im aktuellen Verzeichnis des DCP-Standarddiskettenlaufwerkes angezeigt.

```
FILES "B:*.*"
```

Damit werden alle Dateien im Laufwerk B angezeigt.

```
FILES "TEST??.BAS"
```

Damit werden alle Dateien im aktuellen Verzeichnis des DCP-Standardlaufwerkes angezeigt, die einen Dateinamen haben, der mit TEST, gefolgt von zwei oder weniger anderen Zeichen, beginnt, und den Dateityp **.BAS** haben.

```
FILES "B:"
```

Damit werden alle Dateien des Verzeichnisses im Laufwerk B (nicht aktuelles Laufwerk) angezeigt.

3.3.7. KILL

Loescht eine Datei von einer Diskette. Der Befehl **KILL** in BASIC ist aehnlich dem Befehl **ERASE** im DCP.

Syntax: KILL **Dateiangabe**

Bemerkungen:

Dateiangabe Gueltige Dateiangabe, wie unter Kapitel 6 beschrieben. Fehlt die Laufwerksangabe, wird das Standardlaufwerk von DCP benutzt. Es kann auch ein Pfadname enthalten sein.

KILL kann fuer alle Diskettendateitypen verwendet werden. Es muss der Dateityp angegeben werden, falls ein solcher existiert. Man kann z.B. ein BASIC-Programm mit Hilfe des Befehls

```
SAVE "TEST"
```

speichern. BASIC liefert dafuer den Dateityp **.BAS** fuer den Befehl **SAVE**, aber nicht fuer den Befehl **KILL**.

Soll die Programmdatei spaeter geloescht werden, muss eingegeben werden

KILL "TEST.BAS" und nicht KILL "TEST"

Wird der Befehl KILL fuer eine Datei gegeben, die gerade eroeffnet ist, erhaelt man den Fehler "File Already Open" (Datei schon eroeffnet).

Beispiel:

```
KILL "A:DATA1"
```

In diesem Beispiel wird die Datei DATA1 auf dem Laufwerk A geloescht.

Um eine Datei im Unterverzeichnis LEVEL2 zu loeschen, muss folgendes eingegeben werden:

```
KILL "LEVEL1\LEVEL2\PROG.BAS"
```

Mit KILL koennen nur Dateien geloescht werden. Verzeichnisse koennen nur mit dem Kommando RMDIR entfernt werden.

4. BASIC-Grundanweisungen

4.1. Kommentaranweisung REM

Einfuegen erklaerender Bemerkungen in ein Programm.

Syntax: REM Bemerkung

Bemerkungen:

Bemerkung kann irgendeine Zeichenfolge sein

Anweisungen **REM** werden nicht ausgefuehrt. Sie verlaengern jedoch die Programmlaufzeit und benoetigen Platz im Hauptspeicher.

Zu Anweisungen **REM** kann verzweigt werden (mit Hilfe einer Anweisung GOTO oder GOSUB), die Ausfuehrung geht aber mit der ersten ausfuehrbaren Anweisung nach der Anweisung REM weiter.

An das Ende einer Zeile koennen auch Bemerkungen hinzugefuegt werden, indem man vor die Bemerkungen einen Apostroph setzt. Soll die Bemerkung in die gleiche Zeile kommen wie andere BASIC-Anweisungen, muss die Bemerkung die letzte Anweisung der Zeile sein.
Es ist nicht moeglich, am Ende einer DATA-Anweisung Komentare durch einen Apostroph anzufuegen. Dies muss durch : REM ... geschehen.

Beispiel:

```
10 REM Berechnen Gesamtsumme
20 SUM = 0: REM Loeschen Summenspeicher
30 FOR I = 1 TO 20
40 SUM = SUM+S(I)
```

In Zeile 20 koennte auch stehen:

```
20 SUM = 0 'Loeschen Summenspeicher
```

Dies Beispiel zeigt beide Arten, wie Kommentare in ein Programm eingefuegt werden koennen.

4.2. Datum, Uhrzeit, Lautsprecher

4.2.1. DATE$
(Variable und Anweisung)

Setzt und liest das Datum.

Syntax: **Als Variable:** v$ = DATE$

Als Anweisung: DATE$ = x$

Bemerkungen:

Fuer die Variable (v$ = DATE$):

Eine Zeichenkette aus zehn Zeichen der Form mm-tt-jjjj wird uebergeben. mm sind die zwei Ziffern fuer den Monat, tt ist der Tag des Monats (auch zwei Ziffern), und jjjj ist das Jahr. Das Datum kann durch DOS gesetzt werden, bevor BASIC aufgerufen wird.

Fuer die Anweisung (DATE$ = x$):

x$ ist ein Zeichenkettenausdruck, der benutzt wird, um das laufende Datum zu setzen. x$ kann in einer der folgenden Formen eingegeben werden:

```
mm-tt-jj
mm/tt/jj
mm-tt-jjjj
mm/tt/jjjj
```

Das Jahr muss im Bereich 1980 bis 2099 liegen. Wird fuer den Monat oder fuer den Tag nur eine Ziffer angegeben, wird davor eine Null (0) angenommen. Wird fuer das Jahr nur eine Ziffer angegeben, wird eine Null (0) angehaengt, um daraus zwei Ziffern zu machen. Werden nur zwei Ziffern fuer das Jahr gegeben, wird das Jahr als 19jj angenommen.

Beispiel:

```
OK
10 DATE$ = "8/29/88"
20 PRINT DATE$
RUN
08-29-1988
OK
```

In diesem Beispiel wird das Datum auf den 29. August 1988 gesetzt. Wird das Datum mit Hilfe der Funktion DATE$ eingelesen, so wird vor dem Monat eine Null (0) gesetzt, um daraus zwei Ziffern zu machen, und aus dem Jahr wurde 1988. Auch wurde Monat, Tag und Jahr durch Bindestriche (-) getrennt, obwohl Schraegstriche (/) eingegeben wurden.

4.2.2. TIME$ (Variable und Anweisung)

Setzt die laufende Zeit und ruft sie ab.

Syntax: **Als Variable:** **v$ = TIME$**

Als Anweisung: **TIME$ = x$**

Bemerkungen:

Fuer die Variable (v$ = TIME$):

Die laufende Zeit wird als Zeichenkette aus acht Zeichen uebergeben. Die Zeichenkette hat die Form **hh:mm:ss**, wobei **hh** die Stunden (00 bis 23), **mm** die Minuten (00 bis 59) und **ss** die Sekunden (00 bis 59) bedeuten. Die Zeit kann schon im DCP gesetzt worden sein, bevor zu BASIC verzweigt wurde.

Fuer die Anweisung (TIME$ = x$):

Die laufende Zeit wird gesetzt. **x$** ist ein Zeichenkettenausdruck, der die zu setzende Zeit enthaelt. **x$** kann in einer der folgenden Formen angegeben werden:

hh setzt die Stunden im Bereich 0 bis 23. Die Standardannahme fuer Minuten und Sekunden ist 00.

hh:mm setzt die Stunden und Minuten. Die Minuten muessen im Bereich 0 bis 59 liegen. Sekunden werden auf 00 gesetzt.

hh:mm:ss setzt die Stunden, Minuten und Sekunden. Die Sekunden muessen im Bereich 0 bis 59 liegen.

Eine fuehrende Null kann fuer jeden dieser Werte weggelassen werden, aber es muss mindestens eine Ziffer eingesetzt werden. Soll z.B. die Zeit fuer eine halbe Stunde nach Mitternacht gesetzt werden, wird folgendes eingegeben:

```
TIME$="0:30"  aber nicht

TIME$=":30".
```

Liegt irgendein Wert ausserhalb des angegebenen Bereichs, wird die Fehlermeldung "Illegal function Call" (Ungueltiger Funktionsaufruf) angezeigt. Dabei wird die vorherige Zeitangabe beibehalten. Ist x$ keine gueltige Zeichenkette, wird der Fehler "Type Mismatch" (keine Typuebereinstimmung) angezeigt.

Beispiel:

```
10 CLS
20 LOCATE 10,15
30 PRINT TIME$
40 GOTO 20
```

4.2.3. BEEP

Der Lautsprecher erzeugt einen Ton.

Syntax: **BEEP**

Bemerkungen:

Durch die Anweisung BEEP ertoent im Lautsprecher eine Viertelsekunde lang ein Ton von 800 Hertz. BEEP hat den gleichen Effekt wie **PRINT CHR$(7);**

Beispiel:

```
2340 IF X<20 THEN BEEP
```

In diesem Beispiel prueft das Programm, ob X kleiner als 20 ist. Ist dies der Fall, ertoent der Lautsprecher.

4.3. Typdeklarationsanweisungen DEFTyp

Definiert Variablentypen als ganzzahlig, einfache Genauigkeit, doppelte Genauigkeit oder Zeichenketten.

Syntax: **DEFTyp Buchstabe[-Buchstabe][,Buchstabe [-Buchstabe]]...**

Bemerkungen:

Typ INT, SNG, DBL oder STR.

Buchstabe Ein Buchstabe des Alphabets (A bis Z).

Die Anweisung **DEFTyp** definiert, dass der Variablenname, der mit dem oder den angegebenen Buchstaben beginnt, der Typ der Variablen ist. Jedoch hat das Typdefinitionszeichen (%, !, #, $) immer den Vorrang vor der Anweisung **DEFTyp** fuer die Typgebung einer Variablen. Siehe "Definitionen von Variablentypen" in Kapitel 2.

Fehlen die Typdefinitionsanweisungen, nimmt BASIC an, dass alle Variablen ohne Definitionszeichen Variablen einfacher Genauigkeit sind.

Werden Typdefinitionsanweisungen benutzt, muessen sie am Beginn des Programms stehen. Die Anweisung **DEFTyp** muss ausgefuehrt werden, bevor eine Variable benutzt wird, die damit definiert wird.

Beispiel:

```
OK
10 DEFDBL L-P
20 DEFSTR A
30 DEFINT X, D-H
40 ORDER=1#/3:PRINT ORDER
50 ART="KATZE":PRINT ART
60 X=10/3:PRINT X
RUN
 .3333333333333333
KATZE
 3
```

In Zeile 10 wird definiert, dass alle Variablen, die mit Buchstaben L, M, N, O oder P beginnen, Variablen doppelter Genauigkeit sind.

Durch Zeile 20 werden alle Variablen, die mit dem Buchstaben A beginnen, Zeichenkettenvariablen.

In Zeile 30 wird definiert, dass alle Variablen, die mit dem Buchstaben X, D, E, F, G oder H beginnen, ganzzahlige Variablen sind.

4.4. Wertzuweisungen

4.4.1. LET

Ordnet den Wert eines Ausdrucks einer Variablen zu.

Syntax: [LET] Variable=Ausdruck

Bemerkungen:

Variable — Name einer Variablen oder eines Feldelements, das einen Wert erhalten soll. Es kann sich um numerische Variablen, Zeichenkettenvariablen oder Feldelemente handeln.

Ausdruck — Ausdruck, dessen Wert einer Variablen zugeordnet werden soll. Der Typ des Ausdrucks (Zeichenkette oder numerisch) muss mit dem Typ der Variablen uebereinstimmen; andernfalls erhaelt man den Fehler "Type Mismatch" (Keine Typuebereinstimmung).

Das Wort LET ist wahlfrei, d.h. es kann weggelassen werden.

Beispiel:

```
10 LET ZAEHLER = 60
20 LET K = ZAEHLER+4
30 LET TEXT$ = "ART.-NR.:"
```

In diesem Beispiel wird der Wert 60 der Variablen ZAEHLER zugeordnet. Dann wird der Wert 64, d.h. der Wert aus dem Ausdruck ZAEHLER+4, der Variablen K zugeordnet. Die Zeichenkette "ART.-NR.:" wird der Variablen TEXT$ zugeordnet.

Diese Anweisungen haette man auch wie folgt schreiben koennen:

```
10 ZAEHLER = 60
20 K = ZAEHLER+4
30 TEXT$ = "ART.-NR.:"
```

4.4.2. SWAP

Tauscht die Werte zweier Variablen aus.

Syntax: **SWAP Variable1,Variable2**

Bemerkungen:

Variable1, Variable2 Namen zweier Variablen oder Feldelemente

Mit jedem Variablentyp kann dieser Austausch vorgenommen werden (ganzzahlig, einfache Genauigkeit, doppelte Genauigkeit, Zeichenkette), jedoch muessen beide Variablen vom gleichen Typ sein, sonst wird der Fehler "Type Mismatch" (Keine Typueberseinstimmung) angezeigt.

Beispiel:

```
10 A$ = "Julia":B$ =" und ":C$ ="Romeo"
20 PRINT A$;B$;C$
30 SWAP A$,C$
40 PRINT A$;B$;C$
50 END
```

Nachdem Zeile 30 ausgefuehrt wurde, ist der Wert von C$ "Julia" und der Wert von A$ "Romeo".

4.4.3. RANDOMIZE

Uebergibt einen neuen Anfangswert fuer den Zufallszahlengenerator.

Syntax: RANDOMIZE[n]
RANDOMIZE TIMER

Bemerkungen:

n ist ein ganzzahliger Ausdruck oder ein Ausdruck einfacher oder doppelter Genauigkeit, der als Anfangswert fuer Zufallszahlen benutzt wird.

Wird n weggelassen, verlaesst BASIC die Programmausfuehrung und fordert durch folgende Anzeige zur Eingabe eines Wertes auf:

Random number seed (-32768 to 32767)?

Danach wird **RANDOMIZE** ausgefuehrt.

Erhaelt der Zufallszahlengenerator keinen neuen Anfangswert, uebergibt die Funktion RND jedesmal, wenn das Programm ausgefuehrt wird, die gleiche Folge von Zufallszahlen. Damit diese Folge bei jedem Programmdurchlauf geaendert wird, muss die Anweisung **RANDOMIZE** am Anfang des Programms stehen, und der Anfangswert fuer jeden Lauf muss geaendert werden.

Es kann die interne Uhr verwendet werden, um einen Anfangswert fuer den Zufallszahlengenerator zu erhalten. Man kann z.B. mit der Funktion **VAL** die letzten 2 Ziffern von **TIME$** in eine Zahl umwandeln und diese Zahl fuer n nutzen.

Man kann auch einen neuen Zufallszahlenwert erhalten, ohne dass eine Eingabe verlangt wird. Das geht mit der Funktion **TIMER** im Ausdruck.

Beispiel:

```
10 RANDOMIZE
20 FOR I = 1 TO 4
30 PRINT RND;
40 NEXT I
Ok
RUN
Random number seed (-32768 to 32767)?
```

Die Antwort soll drei sein. Das Programm faehrt wie folgt fort:

```
Random number seed (-32768 to 32767)? 3
 .2226007  .5941419  .2414202  .2013798
Ok
RUN
Random number seed (-32768 to 32767)?
```

Diesmal soll mit vier geantwortet werden. Das Programm faehrt fort:

```
Random number seed (-32768 to 32767)? 4
 .628988  .765605  .5551561  .775797
Ok
RUN
Random number seed (-32768 to 32767)?
```

Wird wieder die drei versucht, bekommt man die gleiche Zahlenfolge wie beim ersten Lauf:

```
Random number seed (-32768 to 32767)? 3
 .2226007   .5941419   .2414202   .2013798
Ok
```

Die Konstruktion RANDOMIZE TIMER generiert bei jedem Aufruf eine neue Zahlenfolge.

Beispiel:

```
10 RANDOMIZE TIMER
20 FOR I% = 1 TO 20
30 PRINT INT(RND*100+1);
40 NEXT
```

4.5. Dialog - Ein-/Ausgabe

4.5.1. INPUT

Eingabe von Werten ueber die Tastatur waehrend der Programmausfuehrung und Zuweisung an bestimmte Variable.

Syntax: INPUT[;]["Anfrage";]Variable[,Variable]...

Bemerkungen:

"Anfrage" Zeichenkettenkonstante, die als Bedienerfuehrung und Eingabeaufforderung fuer eine gewuenschte Eingabe benutzt wird.

Variable Name einer numerischen oder Zeichenkettenvariablen oder eines Feldelements, wohin die eingegebenen Werte gespeichert werden.

Wenn im Programm die Anweisung INPUT erscheint, wird mit einem Fragezeichen (?) am Bildschirm angezeigt,dass auf Dateneingabe gewartet wird. Ist "Anfrage" angegeben, wird diese Zeichenkette vor dem Fragezeichen angezeigt.Danach koennen die benoetigten Daten von der Tastatur eingegeben werden.

Falls statt des Semikolons (;) ein Komma (,) nach "Anfrage" eingegeben wird, wird das Fragezeichen unterdrueckt.

Beispiel:

Wird die Anweisung INPUT "GEBURTSTAG EINGEBEN", B% eingegeben, wird die "Anfrage" ohne Fragezeichen angezeigt.

Die eingegebenen Daten werden der oder den Variablen der Variablenliste uebergeben. Die angegebenen Daten muessen durch Kommas (,) getrennt sein, und die Anzahl der Daten muss mit der Anzahl der Variablen in der Liste uebereinstimmen.

Die Eingabe von Werten erfolgt in der Reihenfolge der Variablen, wie sie in der INPUT-Anweisung aufgefuehrt sind. Die Eingabefolge wird durch die Taste <ENTER> abgeschlossen.

Der Typ der Daten muss mit dem Typ der dazugehoerigen Variablen uebereinstimmen. Zeichenketten, die auf Anfrage einer Anweisung INPUT eingegeben werden, muessen nicht in Anfuehrungszeichen (") eingeschlossen sein, ausser sie enthalten Kommas oder signifikante fuehrende oder nachstehende Leerstellen.

Werden auf die Eingabeaufforderung zu viele oder zu wenige Daten eingegeben oder wird ein falscher Typ fuer einen Wert eingegeben (Buchstaben statt Zahlen usw.), zeigt BASIC die Nachricht "? Redo from Start" (von Anfang an neu eingeben) an. Die Eingabe ist zu wiederholen. BASIC ordnet die Eingabewerte erst dann den Variablen zu, wenn alle Eingaben richtig sind.

Folgt der Anweisung INPUT sofort ein Semikolon (;), wird nach der Eingabe der Daten und anschliessendem Betaetigen der Taste <ENTER> keine Zeilenschaltung auf dem Bildschirm erzeugt. Das bedeutet, der Kursor bleibt auf derselben Zeile, auf der die Eingabe erfolgte.

Beispiele:

```
10 INPUT X
20 PRINT X "quadriert ist" X^2
30 END
RUN
?
```

In diesem Beispiel zeigt das Fragezeichen (?) an, dass etwas eingegeben werden soll. Wird z.B. eine 5 eingegeben, faehrt das Programm fort:

```
? 5
5 quadriert ist 25
```

```
10 PI = 3.14
20 INPUT "Eingabe Radius";R
30 A = PI*R^2
40 PRINT "Die Flaeche des Kreises ist ";A
50 END
RUN
Eingabe Radius? 1
```

In diesem zweiten Beispiel wurde in der INPUT-Anweisung, Zeile 20, eine Bedienernachricht eingefuegt. Auf dem Bildschirm erscheint die Ausschrift: "Eingabe Radius?". Es wird mit 1 geantwortet. Das Programm faehrt fort:

```
Die Flaeche des Kreises ist 3.14
Ok
```

4.5.2. LINE INPUT

Eingabe einer Zeile (bis zu 255 Zeichen) von der Tastatur in eine Zeichenkettenvariable, wobei Trennzeichen ignoriert werden.

Syntax: LINE INPUT[;][**"Anfrage"**;] Zeichenkettenvariable

Bemerkungen:

"Anfrage" Zeichenkettenkonstante, die am Bildschirm angezeigt wird, bevor eine Eingabe angenommen wird. Ein Fragezeichen (?) wird nur angezeigt, wenn es Teil einer Anfragezeichenkette ist.

Zeichenkettenvariable Name einer Zeichenkettenvariablen oder eines Feldelements fuer Zeichenketten, dem die Zeile zugeordnet wird. Die ganze Eingabe ab Ende der Anfrage bis zum Betaetigen der Taste **<ENTER>** wird der **Zeichenkettenvariablen** zugeordnet. Es werden alle Zeichen einschliesslich Anfuehrungszeichen, Komma und Leerzeichen akzeptiert. Nachfolgende Leerzeichen werden ignoriert.

Falls dem **LINE INPUT** sofort ein Semikolon (;) folgt und danach die Taste **<ENTER>** betaetigt wird, wird am Ende der Eingabezeile keine Zeilenschaltung ausgefuehrt. Das bedeutet, dass der Kursor auf der gleichen Zeile bleibt wie die Eingabe.

Die Anweisung **LINE INPUT** kann durch Betaetigen der Tasten **<CTRL-PAUSE>** abgebrochen werden. BASIC kehrt zur Befehlsebene zurueck und zeigt Ok an. Durch die Eingabe CONT kann man die Ausfuehrung von **LINE INPUT** wieder aufnehmen.

Beispiel:

Im folgenden Beispiel wird **LINE INPUT** dazu genutzt, Informationen von der Tastatur einzugeben, die Kommas oder andere Trennzeichen enthalten.

```
10 LINE INPUT "Name, Vorname? ";N$
20 PRINT N$
30 END
RUN
Name, Vorname? Meier, Kurt
Meier, Kurt
Ok
```

4.5.3. PRINT

Nichtformatierte Ausgabe von Daten auf dem Bildschirm.

Syntax: PRINT [Liste der Ausdruecke][;]
? [Liste der Ausdruecke][;]

Bemerkungen:

Liste der Ausdruecke — Liste der numerischen und/oder Zeichenkettenausdruecke, die durch Komma (,), Leerstellen oder Semikolon (;) getrennt sind. Jede Zeichenkonstante in dieser Liste muss in Anfuehrungszeichen eingeschlossen sein.

Wird die Liste der Ausdruecke weggelassen, wird eine Leerstelle angezeigt. Wird die Liste der Ausdruecke angegeben, werden die Werte der Ausdruecke auf dem Bildschirm angezeigt.

Hinweis:
Das Fragezeichen (?) kann als Kurzform fuer die Eingabe von PRINT benutzt werden.

Bildschirmpositionen

Die Position jedes angezeigten Wertes wird durch die Zeichen bestimmt, die die Angaben in der Liste trennen. BASIC teilt die Zeile in Spalten zu je 14 Zeichen ein. In der Liste der Ausdruecke bewirkt:

- ein **Komma**, dass der naechste Wert zu Beginn der naechsten Spalte angezeigt wird.
- ein **Semikolon**, dass der naechste Wert unmittelbar nach dem letzten Wert angezeigt wird.
- die Angabe von einer oder mehreren **Leerstellen** zwischen den Ausdruecken, dass der gleiche Effekt wie bei Angabe eines Semikolons erreicht wird.

Endet die Liste der Ausdruecke mit einem Komma, Semikolon oder einer Funktion SPC oder TAB, beginnt die naechste Anweisung PRINT auf derselben Zeile mit den gleichen Zwischenraeumen. Endet die Liste der Ausdruecke ohne Komma, Semikolon oder einer Funktion SPC oder TAB, wird eine Zeilenschaltung an das Ende der Zeile angefuegt, d.h. BASIC setzt den Kursor an den Beginn der naechsten Zeile.

Reicht die Anzahl der Zeichenpositionen in der laufenden Zeile nicht mehr aus, um einen Wert anzuzeigen, wird dieser Wert am Beginn der naechsten Zeile ausgegeben. Ist der anzuzeigende Wert laenger als in WIDTH definiert, zeigt BASIC soviel wie moeglich in der laufenden Zeile an und den Rest des Wertes in der naechsten physischen Zeile.

Nach angezeigten Zahlen steht immer eine Leerstelle. Vor positiven Zahlen steht auch eine Leerstelle. Vor negativen Zahlen steht ein Minuszeichen. Zahlen einfacher Genauigkeit, die mit sieben oder weniger Stellen im Festkommaformat dargestellt werden koennen, werden im Festkommaformat oder als Integer ausgegeben. Z.B. wird 10^(-7) als .0000001 und 10^(-8) als 1E-8 ausgegeben.

BASIC fuehrt automatisch eine Zeilenschaltung aus, nachdem x gleich 40 oder 80 ist, wie in der Anweisung WIDTH definiert, ausser die Anweisung PRINT endet mit einem Semikolon (;).

Mit LPRINT werden Informationen auf dem Drucker ausgegeben. Siehe "Anweisung LPRINT und LPRINT USING" in diesem Kapitel.

Beispiel:

```
10 X = 5
20 PRINT X+5,X-5,X*(-5)
30 END
Ok
RUN
 10        0             -25
Ok
```

In diesem Beispiel wird durch die Kommata in der Anweisung PRINT jeder Wert an den Beginn der naechsten Spalte gesetzt.

```
10 INPUT X
20 PRINT X "quadriert ist" X^2 "und";
30 PRINT X "hoch 3 ist" X^3
Ok
RUN
? 5
5 quadriert ist 25 und 5 hoch 3 ist 125
Ok
```

Hier werden durch ein Semikolon am Ende der Zeile 20 beide Anweisungen PRINT auf derselben Zeile ausgegeben.

4.5.4. PRINT USING

Gibt Zeichenketten und Zahlen formatiert auf dem Bildschirm aus.

Syntax: PRINT USING vK; Liste der Ausdruecke[;]

Bemerkungen:

vK Ist eine Zeichenkettenkonstante oder Variable, die aus speziellen Formatzeichen besteht. Diese Formatzeichen (siehe unten) bestimmen das Feld und das Format der angezeigten Zeichenketten und Zahlen.

Liste der Ausdruecke — Zeichenkettenausdruecke oder numerische Ausdruecke, die getrennt durch Semikolon (;) oder Komma (,) angezeigt werden.

Zeichenkettenfelder

Wenn mit PRINT USING Zeichenketten angezeigt werden, kann fuer das Format der Zeichenkettenfelder eines von drei Formatzeichen verwendet werden:

! gibt an, dass nur das erste Zeichen einer gegebenen Zeichenkette angezeigt werden soll.

\n Leerstellen\
gibt an, dass 2+n Zeichen aus einer Zeichenkette angezeigt werden sollen. Werden die umgekehrten Schraegstriche ohne Leerstellen eingegeben, werden zwei Zeichen angezeigt, mit einer Leerstelle werden drei Zeichen angezeigt usw.

Ist die Zeichenkette laenger als das Feld, werden die ueberzaehligen Zeichen ignoriert. Ist das Feld laenger als die Zeichenkette, wird die Zeichenkette linksbuendig im Feld gespeichert und rechts mit Leerstellen aufgefuellt.

& Definiert ein Zeichenkettenfeld variabler Laenge. Ist das Feld mit "&" angegeben, entspricht die Zeichenkettenausgabe exakt der Eingabe.

Beispiel:

```
10 A$ = "VOR":B$ = "SICHT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
VSICHT
Ok
```

Numerische Felder

Wird PRINT USING benutzt, um Zahlen anzuzeigen, kann mit folgenden Formatzeichen ein numerisches Feld formatiert werden:

Mit dem Nummernzeichen kann jede Zifernposition dargestellt werden. Ziffernpositionen werden immer aufgefuellt. Hat die darzustellende Zahl weniger Ziffern als angegebene Positionen, wird die Zahl rechtsbuendig im Feld (mit vorgestellten Leerstellen) angezeigt.

. An jeder Stelle des Feldes kann ein Dezimalpunkt eingegeben werden. Gibt das Zeichenkettenformat an, dass vor dem Dezimalpunkt eine Ziffer stehen soll, wird diese Ziffer immer ausgegeben (Ø, falls notwendig). Zahlen werden falls noetig gerundet.

```
PRINT USING "###.##    ";15.3,6.789,.78
 15.30    6.79     Ø.78
```

Im Beispiel wurden am Ende der Formatzeichenkette drei Leerstellen eingefuegt, um die dargestellten Werte in der Zeile zu trennen.

+ Durch ein Pluszeichen am Beginn oder Ende der Formatzeichenkette wird das Vorzeichen der Zahl (+ oder -) vor oder hinter der Zahl angezeigt.

```
PRINT USING "+##.##  ";2.4,-.9
+2.40   -Ø.90
```

- Durch ein Minuszeichen am Ende des Formatfeldes werden negative Zahlen mit einem nachgestellten Minuszeichen dargestellt.

```
PRINT USING "##.##-  ";-68.95,2.5
68.95-   2.50
```

** Durch zwei Sterne am Beginn der Formatzeichenkette werden fuehrende Leerstellen eines numerischen Feldes mit Sternen aufgefuellt.

```
PRINT USING "**#.#   ";12.39,-Ø.9,765.1
*12.4   *-Ø.9   765.1
```

ØØ Durch ein doppeltes Waehrungszeichen wird ein Waehrungszeichen direkt links vor die formatierte Zahl gesetzt. Durch ØØ werden zwei weitere Stellenpositionen definiert, eine davon ist das Waehrungszeichen. Das Exponentialformat kann nicht in Verbindung mit ØØ benutzt werden.
Negative Zahlen koennen nicht verwendet werden, ausser das Minuszeichen steht rechts neben der Zahl.

```
PRINT USING "ØØ###.##";456.78
 Ø456.78
```

**Ø **Ø am Beginn der Formatzeichenkette kombiniert den Effekt der beiden oberen Symbole. Fuehrende Leerstellen werden mit Sternen aufgefuellt, und ein Waehrungszeichen wird genau vor der Zahl ausgegeben. **Ø spezifiziert drei weitere Ziffernpositionen, wobei eine davon das Waehrungszeichen ist.

```
PRINT USING "**Ø##.##";2.34
***Ø2.34
```

Ein Komma links des Dezimalpunktes in einer Formatzeichenkette bewirkt, dass vor jeder dritten Stelle, die links des Dezimalpunkts steht, ein Komma eingefuegt wird. Ein Komma, das sich am Ende der Formatzeichenkette befindet, wird als Teil der Zeichenkette ausgegeben. Durch ein Komma wird eine weitere Zifferpposition definiert.

Das Komma hat keinen Einfluss, falls es in Verbindung mit dem Exponentialformat (^^^^) benutzt wird.

```
PRINT USING "####,.##";1234.5
1,234.50
PRINT USING "####.##,";1234.5
1234.50,
```

^^^^ Vier Zeichen koennen hinter die Zifferpositionszeichen geschrieben werden, um das Exponentialformat zu definieren. Durch die vier Zeichen wird Platz gelassen, um E±nn oder D±nn darzustellen. Jede Dezimalposition kann angegeben werden. Die signifikanten Ziffern werden linksbuendig angelegt und der Exponent errechnet. Ist kein fuehrendes + oder nachstehendes + oder - angegeben, wird eine Zifferpposition links des Dezimalpunkts benutzt, um eine Leerstelle oder ein Minuszeichen auszugeben.

```
Ok
PRINT USING "##.##^^^^";234.56
 2.35E+02
Ok
PRINT USING ".##^^^^-";-88888
.89E+05-
Ok
PRINT USING "+.##^^^^";123
+.12E+03
Ok
```

_ Durch ein Unterstreichungszeichen in der Formatzeichenkette wird das naechste Zeichen als Literalzeichen ausgegeben.

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

Das Literalzeichen kann selbst ein Unterstreichungszeichen sein, wenn man in der Formatzeichenkette "__" angibt.

Ist die auszugebende Zahl groesser als das angegebene numerische Feld, wird vor der Zahl ein Prozentzeichen (%) ausgegeben. Wird durch Aufrundung der Zahl das Feld zu klein, wird das Prozentzeichen vor der gerundeten Zahl ausgegeben.

```
Ok
PRINT USING "##.##";111.22
%111.22
Ok
PRINT USING ".##";.999
%1.00
Ok
```

Ist die Anzahl der angegebenen Ziffern groesser als 24, wird der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf) angezeigt.

Beispiel:

In diesem Beispiel wird gezeigt, wie Zeichenkettenkonstanten in eine Formatzeichenkette eingefuegt werden koennen.

```
Ok
PRINT USING "DIES IST BEISPIEL _##"; 1
DIES IST BEISPIEL #1
Ok
```

4.5.5. LPRINT und LPRINT USING

Druckt Daten auf dem Drucker (LPT1:).

Syntax: LPRINT [Liste von Ausdruecken] [;]
LPRINT USING v$; Liste von Ausdruecken[;]

Bemerkungen:

Liste von Ausdruecken Liste der numerischen und/oder Zeichenkettenvariablenausdruecke, deren Werte gedruckt werden sollen. Die Ausdruecke muessen durch Komma (,) oder Semikolon (;) getrennt sein.

v$ ist eine Zeichenkettenkonstante oder Variable, die das zum Drucken verwendete Format angibt. Dies ist genauer unter Anweisung "PRINT USING" erklaert.

Die Anweisungen funktionieren wie PRINT und PRINT USING, nur dass die Ausgabe ueber den Drucker erfolgt. Siehe Anweisung "PRINT" und "PRINT USING".

LPRINT nimmt eine Druckbreite von 80 Zeichen auf dem Drucker an. Das heisst BASIC fuegt automatisch eine Zeilenschaltung nach dem Drucken von 80 Zeichen ein. Daraus ergibt sich, dass um zwei Zeilen vorgeschoben wird, wenn exakt 80 Zeichen gedruckt werden, ausser die Anweisung endet mit einem Semikolon. Der Wert fuer die Druckbreite kann mit der Anweisung WIDTH "LPT1:" geaendert werden.

Wird ein Seitenvorschub (LPRINT CHR$(12);), gefolgt von einem anderen LPRINT, ausgefuehrt und der Drucker benoetigt mehr als 10 Sekunden fuer den Seitenvorschub, kann der Fehler "Device Timeout" (Einheitenzeitunterbrechung) fuer den zweiten LPRINT auftreten. Um dieses Problem zu vermeiden, muss wie folgt programmiert werden:

```
1  ON ERROR GOTO 65000
.
.
.
65000 IF ERR = 24 THEN RESUME '24 = Zeitunterbrechung
```

Man kann hier zur Absicherung noch ERL testen, um zu sehen, ob die Zeitunterbrechung durch die Anweisung LPRINT entstand.

Beispiel:

In diesem Beispiel werden an den Drucker K 6313 mit Hilfe von LPRINT und CHR$ spezielle Steuerzeichen uebergeben. Diese Drucksteuerzeichen sind im Bedienerhandbuch aufgelistet.

```
10  LPRINT CHR$(14);"      Titelzeile"
20  FOR I = 2 TO 4
30  LPRINT "Berichtszeile";I
40  NEXT I
50  LPRINT CHR$(15);"Komprimiertes Drucken;132 Zeichen/Zeile"
60  LPRINT CHR$(18);"Zurueck zu Normaldruck"
70  LPRINT CHR$(27);"E"
80  LPRINT "Dies ist verstaerktes Drucken"
90  LPRINT CHR$(27);"F"
100 LPRINT "Wieder zurueck zu Normaldruck"
```

Durch dieses Programm erhaelt man die folgende Ausgabe:

```
T i t e l z e i l e

    Berichtszeile 2
    Berichtszeile 3
    Berichtszeile 4
Komprimiertes Drucken; 132 Zeichen/Zeile

Zurueck zu Normaldruck

Dies ist verstaerktes Drucken

Wieder zurueck zu Normaldruck.
```

4.5.6. WRITE

Gibt Daten auf dem Bildschirm aus.

Syntax: WRITE [Liste der Ausdruecke]

Bemerkungen:

Liste der Ausdruecke Liste der numerischen und/oder Zeichenkettenausdruecke, die durch Komma (,) oder Semikolon (;) getrennt sind.

Wird die **Liste der Ausdruecke** weggelassen, wird eine Leerzeile ausgegeben. Wird die Liste der Ausdruecke eingefuegt, wird der Wert der Ausdruecke auf dem Bildschirm angezeigt.

Werden die Werte der Ausdruecke ausgegeben, wird jede Angabe von der letzten durch ein Komma getrennt ausgegeben. Zeichenketten werden in Anfuehrungszeichen (") eingeschlossen. Ist die letzte Angabe der Liste ausgegeben, wird eine Zeilenschaltung ausgefuehrt.

WRITE ist aehnlich wie **PRINT.** Der Unterschied zwischen WRITE und PRINT besteht darin, dass **WRITE** Kommas zwischen die Angaben einfuegt, wenn sie angezeigt werden, und Zeichenketten in Anfuehrungszeichen (") einschliesst. Auch stehen vor positiven Zahlen keine Leerstellen.

Beispiel:

Das Beispiel zeigt, wie WRITE numerische und Zeichenkettenwerte anzeigt.

```
10 A=80: B=90: C$="DAS IST ALLES"
20 WRITE A,B,C$
RUN
80,90,"DAS IST ALLES"
Ok
```

4.5.7. KEY

Setzt und zeigt die Funktionstasten an.

Syntax:
```
KEY n, x$
KEY LIST
KEY ON
KEY OFF
KEY n,CHR$(Tastenkennzeichen)+CHR$(Suchcode)
```

Bemerkungen:

n ist die Nummer einer Funktionstaste im Bereich 1 bis 10.

x$ ist ein Zeichenkettenausdruck, der der Taste zugeordnet wird. (Zeichenkettenkonstanten muessen in Anfuehrungszeichen (") eingeschlossen sein.)

Durch die Anweisung **KEY** ist es moeglich, die Funktionstasten so zu belegen, dass sie automatisch jede Zeichenfolge ausgeben. Die Zeichenkette kann bis zu 15 Zeichen lang sein und kann einer oder allen zehn Funktionstasten zugeordnet werden. Wird die Taste betaetigt, entspricht die Zeichenkette einer Eingabe in BASIC.

Standardmaessig sind diesen Funktionstasten die folgenden Werte zugeordnet:

```
F1  LIST            F2  RUN  <---
F3  LOAD"           F4  SAVE"
F5  CONT <---       F6  ,"LPT1:" <---
F7  TRON <---       F8  TROFF <---
F9  KEY             F10 SCREEN 0,0,0 <---
```

Der Pfeil (<---) zeigt an, dass die Betaetigung der <ENTER>-Taste nicht erforderlich ist.
Durch **KEY ON** wird der Inhalt dieser Tasten in der 25. Zeile des Bildschirms angezeigt. Ist die Breite 40, werden fuenf der zehn Tasten angezeigt; ist die Breite 80, werden alle zehn angezeigt. In jeder Breite werden nur die ersten sechs Zeichen jedes Wertes angezeigt. ON ist der Standardstatus fuer die Anzeige der Funktionstasten.

KEY OFF loescht die Anzeige der Tasten von der 25. Zeile. Dadurch wird die Zeile fuer die Programmbenutzung frei. Die Funktionstasten werden dadurch nicht inaktiviert.

Nachdem mit KEY OFF die Funktionstastenanzeige ausgeschaltet ist, kann mit Hilfe der Anweisung LOCATE 25,1, gefolgt von der Anweisung **PRINT**, jede gewueschte Nachricht auf der untersten Zeile des Bilschirms angezeigt werden. Die Information auf der Zeile 25 wird nicht nach oben verschoben wie die Zeile 1 bis 24.

KEY LIST zeigt den Inhalt aller zehn Funktionstasten auf dem Bilschirm an. Alle 15 Zeichen jedes Wertes werden angezeigt.

**KEY n, x$** ordnet den Wert von x$ der angegebenen Funktionstaste zu (1 bis 10). x$ kann eine Laenge von max. 15 Zeichen haben. Ist der Inhalt laenger als 15 Zeichen, werden nur die ersten 15 Zeichen zugeordnet.

Wird eine leere Zeichenkette zugeordnet (Laenge der Zeichenkette ist Null), wird die Funktion dieser Taste inaktiviert.

Befindet sich der fuer n angegebene Wert nicht im Bereich 1 bis 10, erhaelt man den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Die vorher zugeordnete Zeichenkette fuer diese Taste bleibt erhalten.

Wird eine Funktionstaste betaetigt, uebergibt die Funktion INKEY$ bei jedem Aufruf ein Zeichen der zugeordneten Zeichenkette. Ist die Funktion dieser Taste aufgehoben, uebergibt INKEY$ eine Zeichenkette aus zwei Zeichen. Das erste Zeichen ist eine binaere Null, das zweite ist der Tastensuch-Code. Siehe Anhang D: "ASCII-Zeichencodes".

Zusaetzlich gibt es 6 definierbare Tasten. Sie werden mit der folgenden Anweisung definiert:

KEY n,CHR$(Tastenkennzeichen)+CHR$(Suchcode)

n
ist ein numerischer Ausdruck im Bereich 15 bis 20

Tastenkennzeichen
ist eine Maske fuer die Tasten, die zusammen mit einer Umschalttaste, der Taste <ALT> oder der Taste <CTRL> betaetigt wurden. Das entsprechende Bit in Tastenkennzeichen muss gesetzt werden, um die Taste zu finden, die zusammen mit einer Umschalttaste, der Taste <ALT> oder der Taste <CTRL> betaetigt wurde. Die hexadezimalen Werte fuer Tastenkennzeichen sind:

Gross | (CAPS LOCK)	&H0-Gross | (CAPS LOCK) ist nicht aktiviert.
Gross | (CAPS LOCK)	&H40-Gross | (CAPS LOCK) ist aktiviert.
Num | (NUM LOCK)	&H0-NUM | (NUM LOCK) ist nicht aktiviert
Num | (NUM LOCK)	&H20-NUM | (NUM LOCK) ist aktiviert
ALT	&H08-Taste <ALT> wurde betaetigt
CTRL	&H04-Taste <CTRL> wurde betaetigt
Linke Umschalttaste	&H02-Die linke Umschalttaste wurde betaetigt
Rechte Umschalttaste	&H01-Die rechte Umschalttaste wurde betaetigt

Suchcode
ist die Nummer einer der 83 Tasten, die identifiziert werden sollen. Siehe "Anhang J, Suchcodes".

Hierbei muss beachtet werden, dass die rechte und die linke Umschalttaste als gleichwertig betrachtet werden, so dass die Werte &H01,&H02 oder &H03 (Summe von hex01 und hex02) benutzt werden koennen, um eine Umschalttaste zu identifizieren.

Man kann auch den Status mehrerer Umschalttasten addieren, wie z.B. <CTRL> und <ALT>. Die Werte fuer den Umschaltstatus muessen hexadezimal angegeben werden.

Wenn eine Taste oder eine Tastenkombination identifiziert werden soll, muss der Status von NUM | (NUM LOCK) und Gross | (CAPS LOCK) bekannt sein.

Die identifizierten Tasten werden in der folgenden Reihenfolge verarbeitet:

1. Die Tasten CTRL-Druck <CTRL-PRTSC>, die den Drucker aktivieren, werden zuerst verarbeitet. Auch wenn <CTRL-PRTSC> als Unterbrechungstastenfolge definiert sind, koennen sie betaetigt werden, um eine gedruckte Kopie des Bildschirminhalts zu erhalten.

2. Anschliessend werden die Funktionstasten F1-F10,Kursor nach oben, unten, rechts und links(1-14) verarbeitet. Setzen der Suchcodes 59 bis 68, 72, 75, 77, oder 80 als Unterbrechungstasten hat keinen Einfluss, da sie als vordefiniert gelten.

3. Zum Schluss werden die fuer 15-20 definierten Tasten verarbeitet.

Hinweise:

- Identifizierte Tasten werden nicht als Code an den BIOS-Puffer uebergeben,so dass nur BASIC erkennt, ob die Tasten betaetigt wurden.

- Mit der Definition der Tastenkombination <CTRL-PAUSE> oder <CTRL-ALT-DEL> muss vorsichtig umgegangen werden, da das System ausgeschaltet werden muss, um das Programm zu stoppen, wenn kein Test in der Unterbrechungsroutine enthalten ist.

Im Abschnitt "4.12.1 Anweisung KEY(n)" wird dargestellt, wie eine Funktionstastenunterbrechung in BASIC aktiviert und inaktiviert wird.

Beispiel:

```
50 KEY ON
```

zeigt die Funktionstasten auf der 25. Zeile an.

```
200 KEY OFF
```

loescht die Anzeige fuer die Funktionstasten. Die Funktionstasten sind immer noch aktiv, werden aber nicht angezeigt.

```
10 KEY 1,"FILES"+CHR$(13)
```

ordnet die Zeichenkette "FILES" und die Taste <ENTER> der Funktionstaste 1 zu. Dies ist ein Weg, um einen haeufig benutzten Befehl einer Funktionstaste zuzuordnen.

```
20 KEY 1,""
```

inaktiviert die Funktionstaste 1 als Funktionstaste.

```
10 KEY 15, CHR$(&H40)+CHR$(25)
20 ON KEY(15) GOSUB 1000
30 KEY(15) ON
```

setzt eine Tastenunterbrechung fuer den Grossbuchstaben P. Es ist zu beachten, dass alle drei KEY-Anweisungen - KEY, KEY(n) und ON KEY - fuer die Tastenunterbrechung benutzt werden.

```
10 KEY 20,CHR$(&H04+&H03)+CHR$(30)
20 ON KEY(20) GOSUB 2000
30 KEY(20) ON
```

setzt eine Tastenunterbrechung fuer CTRL-Umschalttaste A <CTRL-Shift-A>. Beachten Sie, dass die Hex-Werte fuer <CTRL> (&H04) und Umschalttaste (Shift)(&H03) addiert werden, um den Umschaltstatus zu erhalten.

4.5.8. WIDTH

Legt die Anzahl der Zeichen einer Ausgabezeile fest. Nachdem die angegebene Anzahl von Zeichen ausgegeben wurde, fuegt BASIC eine Zeilenschaltung hinzu.

Syntax: WIDTH Groesse
WIDTH #Dateinummer, Groesse
WIDTH Einheit, Groesse

Bemerkungen:

Groesse — Numerischer Ausdruck im Bereich 0 bis 255. WIDTH 0 bedeutet das gleiche wie WIDTH 1.

Dateinummer — Numerischer Ausdruck im Bereich 1 bis 15. Dies ist die Nummer der Datei, die fuer eine der unten aufgelisteten Einheiten eroeffnet wurde.

Einheit — Zeichenkettenausdruck fuer die Einheitenidentifikation. Gueltige Einheiten sind SCRN:, LPT1:, LPT2:, LPT3:, COM1: oder COM2:.

Abhaengig von der angegebenen Einheit sind die folgenden Aktionen moeglich:

WIDTH Groesse oder WIDTH "SCRN:",Groesse
setzt die Bildschirmbreite. Nur Spaltenbreite 40 oder 80 erlaubt. Befindet sich der Bildschirm im grafischen Modus fuer mittlere Aufloesung (dies wuerde durch die Anweisung SCREEN 1 geschehen), wird durch WIDTH 80 der Bildschirm in die hohe Aufloesung gebracht (wie durch die Anweisung SCREEN 2).

Befindet sich der Bildschirm im grafischen Modus fuer hohe Aufloesung (wie durch die Anweisung SCREEN 2),wird durch WIDTH 40 der Bildschirm in die mittlere Aufloesung gebracht (wie mit der Anweisung SCREEN 1). Siehe dazu Kapitel 8.

Hinweis:

Durch die Aenderung der Bildschirmbreite wird der Bildschirm geloescht und der Bildschirmrand auf schwarz gesetzt.

WIDTH Einheit,Groesse
Wird fuer eine verzoegerte Breitenzuordnung fuer eine Einheit benutzt. Diese Form der Breite speichert den neuen Wert fuer Breite, ohne die aktuelle Breite zu veraendern. Ein nachfolgendes OPEN fuer diese Einheit benutzt diesen Wert fuer die Breite, solange die Einheit eroeffnet ist. Die Breite aendert sich nicht sofort, wenn die Einheit bereits eroeffnet ist.

Hinweis:

LPRINT, LLIST und LIST, "LPTn:" bewirken ein implizites OPEN und werden deshalb von dieser Anweisung nicht beeinflusst.

WIDTH #Dateinummer,Groesse
Die Breite der Einheit, der die Dateinummer zugeordnet ist, wird sofort auf die neue angegebene Groesse geaendert. Dies erlaubt, die Breite zu aendern, waehrend eine Datei eroeffnet ist.

Jeder eingegebene Wert, der sich nicht innerhalb des angegebenen Bereichs befindet, ergibt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Der vorherige Wert bleibt erhalten.

WIDTH hat auf die Tastatur (KYBD:) keinen Einfluss.

Die Druckbreite fuer den Drucker ist standardmaessig 80, wenn mit BASIC begonnen wird. Die Maximalbreite fuer den Drucker K 6313 ist 132. Es wird jedoch kein Fehler fuer Werte zwischen 132 und 255 uebergeben.

Es ist dem Programmierer ueberlassen, die physische Breite des Druckers zu setzen. Bei einigen Druckern geschieht dies durch Senden von Sonderzeichen, manche haben Schalter. Der Drucker K 6313 muss mit der Anweisung LPRINT CHR$(15); auf verdichtetes Schreiben gesetzt werden, wenn mit einer Breite groesser als 80 gedruckt wird. Mit LPRINT CHR$(18); wird auf normale Schriftbreite zurueckgesetzt. Der Drucker fuehrt automatisch ein Wagenruecklaufzeichen hinzu, wenn die maximale Zeilenbreite ueberschritten ist.

Durch die Angabe einer Breite von 255 wird der Zeilenbruch inaktiviert. Dies hat den Effekt einer "unendlichen" Breite. Fuer Datenfernverarbeitungsdateien ist WIDTH 255 der Standardwert.

Durch das Aendern der Breite fuer die Datenfernverarbeitungs-datei aendert sich weder der Empfangs- noch der Uebertragungs-puffer. BASIC sendet nur ein Wagenruecklaufzeichen nach jeder angegebenen Anzahl von Zeichen.

Das Aendern des Bildschirmmodus beeinflusst die Bildschirm-breite nur, wenn zwischen SCREEN 2 und SCREEN 1 oder SCREEN 0 gewechselt wird. Siehe "Anweisung SCREEN" in Kapitel 8.

Beispiel:

```
10    WIDTH "LPT1:",75
20    OPEN "LPT1:" FOR OUTPUT AS #1
:
:
:
6020 WIDTH #1,40
```

Im vorherigen Beispiel wird in Zeile 10 die Druckbreite auf 75 Zeichen pro Zeile gesetzt. In Zeile 20 wird die Datei #1 fuer den Drucker eroeffnet und die Druckbreite fuer nachfolgende Anweisungen PRINT #1,...auf 75 gesetzt. In Zeile 6020 wird die gegenwaertige Druckbreite auf 40 Zeichen pro Zeile geaendert.

4.6. Arbeit mit dem Konstantenzeicher

4.6.1. READ

Liest Werte aus der Anweisung DATA und weist sie Variablen zu (siehe Anweisung "DATA" in diesem Kapitel).

Syntax: READ Variable[,Variable]...

Bemerkungen:

Variable — Numerische oder Zeichenkettenvariable oder ein Feldelement, dem ein Wert zugeordnet werden soll, der aus der DATA-Tabelle gelesen wird.

Die Anweisung READ muss immer in Verbindung mit der Anweisung DATA benutzt werden. Mit READ werden Werte der Anweisung DATA den Variablen in der Anweisung READ zugeordnet. Die Variablen der Anweisung READ muessen numerische oder Zeichenkettenvariablen sein, und die gelesenen Werte muessen mit dem Variablentyp uebereinstimmen. Stimmen sie nicht ueberein, wird ein "Syntax Error" (Syntaxfehler) angezeigt.

Mit READ koennen eine oder mehrere Anweisungen DATA angesprochen werden (sie werden immer in Reihenfolge abgearbeitet). Auch koennen mehrere Anweisungen READ die gleiche Anweisung DATA ansprechen. Ist die Anzahl der Variablen in der Liste groesser als die Anzahl der Elemente in der (den) Anweisung(en) DATA, erscheint der Fehler "Out of Data" (Zu wenig Daten). Ist die Anzahl der angegebenen Variablen kleiner als die Anzahl der Elemente in der (den) Anweisung(en) DATA, lesen nachfolgende Anweisungen READ ab dem ersten nicht gelesenen Element. Sind keine weiteren Anweisungen READ vorhanden, werden die Extradaten ignoriert.

Um die gleichen Daten von einer Zeile in der Liste der Anweisungen DATA wieder zu lesen, muss die Anweisung RESTORE verwendet werden (siehe Anweisung "RESTORE" in diesem Kapitel).

Beispiel:

```
80  FOR I = 1 TO 10
90  READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
```

Dieser Programmteil liest Werte der Anweisungen **DATA** in das Feld A. Nach der Ausfuehrung enthaelt A(1) den Wert 3.08 usw.

```
Ok
10 PRINT "NAME","VORNAME","STAMM-NR"
20 READ N$,V$,S
30 DATA "MEIER,",ANTON,80211
40 PRINT N$,V$,S
RUN
NAME          VORNAME          STAMM-NR
MEIER,        ANTON             80211
Ok
```

In diesem Programm werden Zeichenketten und numerische Daten aus der Anweisung **DATA** in Zeile 30 gelesen. ANTON muss nicht in Anfuehrungszeichen (") stehen, da es keine Komma, Semikolon oder signifikante fuehrende oder nachstehende Leerstellen enthaelt. "MEIER," muss in Anfuehrungszeichen stehen, da es ein Komma enthaelt.

4.6.2. DATA

In dieser Anweisung werden numerische oder Zeichenkettenkonstanten gespeichert, die mit der Anweisung **READ** im Programm gelesen werden koennen.

Syntax: DATA Konstante[,Konstante]...

Bemerkungen:

Konstante kann eine numerische oder Zeichenkettenkonstante sein. In der Liste sind keine Ausdruecke erlaubt. Die numerische Konstante kann jedes Format haben - ganzzahlig, Festkomma, Gleitkomma, hexadezimal oder oktal. Zeichenkettenkonstanten in der Anweisung **DATA** muessen in Anfuehrungszeichen eingeschlossen sein, wenn die Zeichenkette Kommas (,), Doppelpunkte (:) oder signifikante fuehrende oder nachstehende Leerstellen enthaelt.

Anweisungen **DATA** sind nicht ausfuehrbar und koennen ueberall im Programm stehen. Eine Anweisung **DATA** kann so viele Konstanten enthalten, wie in eine Zeile passen. Die Anzahl der Anweisungen **DATA** im Programm ist nicht begrenzt. Die Informationen, die in der Anweisung **DATA** stehen, koennen als fortlaufende Liste von Angaben betrachtet werden, unabhaengig davon, wie viele Angaben in einer Zeile, oder wo die Zeilen im Programm stehen. Die Anweisungen **READ** lesen die Anweisungen **DATA** in der Reihenfolge der Zeilennummern.

Der Variablentyp (numerisch oder Zeichenkette) in der Anweisung **READ** muss mit dem Typ der zu lesenden Konstanten in der Anweisung **DATA** uebereinstimmen; andernsfalls wird ein "Syntax Error" (Syntaxfehler) angezeigt.

Mit Hilfe der Anweisung **RESTORE** kann eine Information von irgendeiner Zeile der Liste der Anweisungen **DATA** erneut gelesen werden (siehe Anweisung "RESTORE" in diesem Kapitel).

Beispiel:

Siehe unter Anweisung "READ" in diesem Kapitel.

4.6.3. RESTORE

Erlaubt es, Anweisungen **DATA** ab einer angegebenen Zeile wieder zu lesen.

Syntax: RESTORE [Zeile]

Bemerkungen:

Zeile Zeilennummer der Anweisung **DATA** im Programm.

Nachdem die Anweisung **RESTORE** ausgefuehrt wurde, liest die naechste Anweisung **READ** den ersten Wert in der ersten Anweisung **DATA** des Programms. Ist **Zeile** angegeben, liest die naechste Anweisung **READ** den ersten Wert der angegebenen Anweisung **DATA**.

Beispiel:

```
Ok
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
50 PRINT A;B;C;D;E;F
RUN
 57 68 79 57 68 79
Ok
```

Die Anweisung **RESTORE** in Zeile 20 setzt den DATA-Zeiger auf den Beginn, so dass die Werte, die in Zeile 30 gelesen werden, 57, 68 und 79 sind.

4.7. Steueranweisungen

4.7.1. STOP

Beendet die Programmausfuehrung und kehrt zur Befehlsebene zurueck.

Syntax: STOP

Bemerkungen:

Anweisungen **STOP** koennen ueberall im Programm verwendet werden, um die Programmausfuehrung zu beenden. Sobald BASIC eine Anweisung **STOP** ausfuehrt, wird die nachfolgende Nachricht angezeigt:

Break in nnnnn

nnnnn ist die Zeilennummer, in der STOP auftrat.

Die Anweisug **STOP** schliesst Dateien nicht ab, im Gegensatz zu **END.**

BASIC kehrt immer zur Befehlsebene zurueck, nachdem **STOP** ausgefuehrt wurde. Die Ausfuehrung des Programms kann mit dem Kommando **CONT** wieder aufgenommen werden (siehe Kommando "CONT" in Kapitel 3).

Beispiel:

```
10 INPUT A,B
20 ZAHL = A*B
30 STOP
40 SUMME = ZAHL+200:PRINT SUMME
RUN
? 26,2.1
Break in 30
PRINT ZAHL
 54.6
Ok
CONT
 254.6
Ok
```

In diesem Beispiel wird der Wert fuer ZAHL errechnet, danach die Ausfuehrung beendet. Waehrend das Programm steht, kann der Wert von ZAHL geprueft werden. Danach kann mit **CONT** die Programmausfuehrung an Zeile 40 wieder aufgenommen werden.

4.7.2. END

Beendet die Programmausfuehrung, schliesst alle Dateien ab und kehrt zur Befehlsebene zurueck.

Syntax: END

Bemerkungen:

Anweisungen **END** koennen ueberall im Programm stehen und beenden die Ausfuehrung. **END** und **STOP** unterscheiden sich in zwei Punkten:

- Bei **END** wird die Nachricht "Break" (Unterbrechung) nicht angezeigt.
- **END** schliesst alle Dateien ab.

Die Anweisung **END** am Ende eines Programms ist wahlfrei. BASIC kehrt immer zur Befehlsebene zurueck, nachdem **END** ausgefuehrt wurde.

Beispiel:

```
520 IF K>1000 THEN END ELSE GOTO 20
```

In diesem Beispiel endet das Programm, sobald K groesser als 1000 ist; sonst verzweigt das Programm zur Zeilennummer 20.

4.7.3. GOTO

Dies ist eine unbedingte Verzweigung aus dem normalen Programmablauf zu einer angegebenen Zeilennummer.

Syntax: GOTO Zeile

Bemerkungen:

Zeile Zeilennummer einer Programmzeile

Ist **Zeile** die Zeilennummer einer ausfuehrbaren Anweisung, werden diese Anweisung und nachfolgende ausgefuehrt. Ist **Zeile** eine nicht ausfuehrbare Anweisung (wie z.B. REM oder DATA), faehrt das Programm mit der ersten ausfuehrbaren Anweisung nach **Zeile** fort.

Die Anweisung **GOTO** kann auch in der direkten Betriebsart benutzt werden, um in einem Programm zu einem bestimmten Punkt zu verzweigen. Dies kann beim Testen nuetzlich sein.

Mit **ON...GOTO** kann zu verschiedenen Zeilen, abhaengig vom Ergebnis eines Ausdrucks, verzweigt werden.

Beispiel:

```
Ok
 5 DATA 5,7,12
10 READ R
20 PRINT "R = ";R,
30 A = 3.14*R^2
40 PRINT "FLAECHE =";A
50 GOTO 5
RUN
R = 5     FLAECHE = 78.5
R = 7     FLAECHE = 153.86
R =12     FLAECHE = 452.16
Out of Data in 10
Ok
```

Durch die Anweisung **GOTO** in Zeile 50 geht das Programm in eine unendliche Schleife, die gestoppt wird, wenn das Programm keine Daten in der Anweisung **DATA** mehr findet.

4.7.4. GOSUB .... RETURN

Verzweigen in ein BASIC-Unterprogramm und Rueckkehr von einem Unterprogramm.
Ein BASIC-Unterprogramm ist eine zusammenhaengende Folge von beliebigen Anweisungen, die durch die Anweisung **GOSUB** angesprungen und durch eine Anweisung **RETURN** abgeschlossen wird.

Unterprogramme werden immer dann verwendet, wenn eine bestimmte Folge von Anweisungen an mehreren Stellen des Programmes benoetigt wird.

Syntax:

```
GOSUB Zeile
   .
   .
   .
 RETURN
```

Bemerkungen:

Zeile Zeilennummer, bei der das BASIC-Unterprogramm gestartet werden soll. Es muss nicht immer die 1. Zeile im BASIC-Unterprogramm sein. Ein solches BASIC-Unterprogramm kann auch mehrere Startzeilen enthalten (Unterprogramm mit **ENTRY**-Punkten).

Ein Unterprogramm kann sooft wie noetig in einem Programm aufgerufen werden, und ein Unterprogramm kann in einem anderen Unterprogramm aufgerufen werden. Diese Verschachtelung von Unterprogrammen wird nur vom verfuegbaren Hauptspeicher begrenzt.

Durch die Anweisung **RETURN** verzweigt BASIC zu der Anweisung zurueck, die der letzten Anweisung **GOSUB** folgt. Ein Unterprogramm kann mehrere Anweisungen **RETURN** enthalten, falls man von verschiedenen Punkten des Unterprogramms zurueckverzweigen moechte. Unterprogramme koennen ueberall im Programm stehen.

Damit in einem Programm nicht durch Zufall in ein Unterprogramm verzweigt wird, kann vor das Unterprogramm eine der Anweisungen **STOP**, **END** oder **GOTO** gesetzt werden, um dieses Unterprogramm zu uebergehen.

Mit **ON...GOSUB** kann zu verschiedenen Unterprogrammen als Ergebnis eines Ausdrucks verzweigt werden.

Beispiel:

```
10 GOSUB 40
20 PRINT "VOM UP ZURUECK"
30 END
40 PRINT "UP";
50 PRINT " IN";
60 PRINT " ARBEIT"
70 RETURN
RUN
UP IN ARBEIT
VOM UP ZURUECK
Ok
```

Dieses Beispiel zeigt, wie ein Unterprogramm arbeitet. Anweisung **GOSUB** in Zeile 10 ruft das Unterprogramm in Zeile 40. Jetzt verzweigt das Unterprogramm zur Zeile 40 und beginnt mit der Ausfuehrung der Anweisungen, bis die Anweisung **RETURN** in Zeile 70 erscheint. An diesem Punkt verzweigt das Programm zu der Anweisung nach dem Unterprogrammaufruf zurueck. Das ist die Zeile 20. Die Anweisung **END** in Zeile 30 verhindert, dass das Unterprogramm ein zweites Mal ausgefuehrt wird.

4.7.5. ON...GOSUB und ON...GOTO

Verzweigt zu einer von mehreren angegebenen Zeilennummern, abhaengig vom Wert des Ausdrucks.

Syntax: **ON n GOTO Zeile[,Zeile]..**

ON n GOSUB Zeile[,Zeile]..

Bemerkungen:

n ist ein numerischer Ausdruck, der -falls noetig- zu einer ganzen Zahl gerundet wird. Er muss sich im Bereich 1 bis 255 befinden oder man erhaelt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Zeile Zeilennummer einer Zeile, zu der verzweigt werden soll.

Der Wert von n bestimmt, welche Zeilennummer in der Liste fuer die Verzweigung benutzt wird. Ist der Wert von n z.B. 3, wird die dritte Zeilennummer der Liste fuer die Verzweigung ausgewaehlt.

In der Anweisung **ON...GOSUB** muss jede Zeilennummer in der Liste die erste Zeilennummer eines Unterprogramms sein. Eventuell benoetigt man eine Anweisung **RETURN**, um das Programm zu der Zeile zurueckzubringen, die der Anweisung **ON...GOSUB** folgt.

Ist der Wert von n 0 (Null) oder groesser als die Anzahl der angegebenen Zeilennummern in der Liste (aber kleiner oder gleich 255), faehrt BASIC mit der naechsten ausfuehrbaren Anweisung fort.

Beispiele:

```
100 ON L GOTO 150, 300, 320, 390
```

Im ersten Beispiel wird verzweigt:

zur Zeile 150, falls L gleich 1
zur Zeile 300, falls L gleich 2
zur Zeile 320, falls L gleich 3
zur Zeile 390, falls L gleich 4.

Ist L gleich 0 (Null) oder groesser als 4, geht das Programm zur naechsten Anweisung.

Im naechsten Beispiel wird gezeigt, wie die Anweisung **ON...GOSUB** benutzt wird.

```
1200 ON A GOSUB 1300,1400
1300 REM START des Unterprogramms fuer A=1
  :
  :
1390 RETURN
1400 REM START des Unterprogrammes fuer A=2
  :
  :
```

4.7.6. RETURN

Verzweigt aus einen Unterprogramm zurueck. Siehe Anweisungen **"GOSUB ... RETURN"** in diesem Kapitel.

Syntax: RETURN [Zeile]

Bemerkungen:

Zeile Zeilennummer des Programms, zu der zurueck verzweigt werden soll.

Obwohl die Benutzung von Zeile in **RETURN** in jedem Unterprogramm verwendet werden kann, wurde diese Erweiterung fuer das Zurueckverzweigen aus den nicht lokalen Ereignisunterbrechungsroutinen hinzugefuegt. Aus einer dieser Routinen will man im BASIC-Programm oft an eine feste Zeilennummer zurueckverzweigen und den GOSUB-Eingang in das Unterprogramm umgehen. Man muss mit dieser nicht lokalen Anweisung **RETURN** in Verbindung mit der Zeilennummer vorsichtig sein, da alle anderen Anweisungen **GOSUB**, **WHILE** und **FOR**, die zur Zeit der Unterbrechung aktiv sind, auch aktiv bleiben.

4.7.7. FOR ... NEXT

Fuehrt einen Zyklus von Anweisungen aus, wobei die Anzahl von Durchlaeufen festliegt.

Syntax:

```
FOR Variable=x TO y [STEP z]
  .
  .
  .
NEXT [Variable][,Variable]...
```

Bemerkungen:

Variable Ganzzahlige Variable oder eine Variable mit einfacher Genauigkeit, die als Zaehler benutzt wird.

x ist ein numerischer Ausdruck, der den Anfangswert des Zaehlers enthaelt.

y ist ein numerischer Ausdruck, der den Endwert des Zaehlers enthaelt.

z ist ein numerischer Ausdruck, der als Schrittweite benutzt wird.

Die Programmzeilen, die der Anweisung **FOR** folgen, werden ausgefuehrt, bis eine Anweisung **NEXT** erreicht wird. Der Zaehler wird um die Summe erhoeht, die im STEP-Wert (z) angegeben ist. Wird der Wert fuer z nicht angegeben, wird als Schrittweite 1 (eins) angenommen. Eine Pruefung wird ausgefuehrt, ob

der Wert des Zaehlers jetzt groesser ist als der Endwert y. Ist er nicht groesser, verzweigt BASIC zurueck zur Anweisung nach der Anweisung **FOR**, und der Prozess wird wiederholt. Ist er groesser, so geht die Ausfuehrung mit der Anweisung weiter, die der Anweisung **NEXT** folgt. Dies ist eine **FOR...NEXT**-Schleife.

Ist der Wert von **z** negativ, laeuft der Test umgekehrt ab. Der Zaehler wird jedesmal, wenn die Schleife beendet ist, verringert. Die Schleife wird solange ausgefuehrt, bis der Zaehler kleiner als der Endwert ist.

Der Rest der Schleife wird uebergangen, wenn x schon groesser als y ist, falls der Wert fuer **STEP** positiv ist, oder falls x kleiner als y ist, wenn der Wert fuer **STEP** negativ ist. Ist z Null, wird eine unendliche Schleife erstellt, bis man auf irgendeine Weise den Zaehler groesser als den Endwert setzt.

Die Programmlaufzeit wird beschleunigt, wenn ganzzahlige Zaehler verwendet werden.

Geschachtelte Schleifen

FOR...NEXT-Schleifen koennen geschachtelt werden, d.h. eine **FOR...NEXT**-Schleife kann in einer anderen **FOR...NEXT**-Schleife enthalten sein. Sind die Schleifen geschachtelt, muss jede Schleife als Zaehler eigene Variablennamen haben. Die Anweisung **NEXT** der inneren Schleife muss vor der fuer die aeussere Schleife erscheinen. Haben geschachtelte Schleifen das gleiche Ende, kann fuer alle die gleiche Anweisung **NEXT** benutzt werden.

Die Anweisung NEXT der Form

NEXT Var1, Var2, Var3 ...

entspricht der Folge von Anweisungen:

```
NEXT Var1
NEXT Var2
NEXT Var3
.
.
.
```

Die Variable(n) in der Anweisung **NEXT** koennen weggelassen werden. In diesem Fall bezieht sich die Anweisung **NEXT** auf die letzte Anweisung **FOR**. Werden geschachtelte **FOR...NEXT**-Schleifen benutzt, sollten die Variablen fuer alle Anweisungen **NEXT** eingefuegt werden, um Programmfehler zu vermeiden. Erforderlich ist dies z.B., wenn man aus einer geschachtelten Schleife heraus verzweigt. Werden Variablenamen in den Anweisungen **NEXT** benutzt, wird das Programm etwas langsamer ausgefuehrt.

Wird eine Anweisung **NEXT** vor der zugehoerigen Anweisung **FOR** ausgefuehrt, erhaelt man den Fehler "NEXT without FOR" (NEXT ohne FOR).

Beispiele:

Im ersten Beispiel wird eine **FOR...NEXT**-Schleife mit einer Schrittweite von 2 gezeigt.

```
10 J = 10: K = 30
20 FOR I =1 TO J STEP 2
30 PRINT I;
40 K = K+10
50 PRINT K
60 NEXT
RUN
 1  40
 3  50
 5  60
 7  70
 9  80
Ok.
```

Im naechsten Beispiel wird die Schleife nicht ausgefuehrt, weil der Anfangswert der Schleife groesser als der Endwert ist.

```
Ok
10 J = 0
20 FOR I=1 TO J
30 PRINT I
40 NEXT I
RUN
Ok
```

Im letzten Beispiel wird die Schleife zehnmal ausgefuehrt. Der Endwert der Schleifenvariablen wird immer vor dem Anfangswert gesetzt.

```
Ok
10 I = 5
20 FOR I = 1 TO I+5
30 PRINT I;
40 NEXT
RUN
 1  2  3  4  5  6  7  8  9  10
Ok
```

4.7.8. IF

Bedingte Anweisung, die den Programmablauf aendert, abhaengig vom Ergebnis eines Ausdrucks.

Syntax: `IF Ausdruck[,]THEN Angabe [ELSE Angabe]`
`IF Ausdruck[,]GOTO Zeile [[,]ELSE Angabe]`

Bemerkungen:

Ausdruck kann irgendein logischer oder numerischer Ausdruck sein.

Angabe kann eine BASIC-Anweisung oder eine Folge von Anweisungen sein (durch Doppelpunkt (:) getrennt); oder es kann einfach eine Zeilennummer sein, zu der verzweigt werden soll.

Zeile Zeilennummer einer im Programm existierenden Zeile.

Eine bedingte Anweisung darf max. 255 Zeichen (logische BASIC-Zeile) umfassen.
Ist der **Ausdruck** wahr (nicht Null), wird die Angabe **THEN** oder **GOTO** ausgefuehrt. **THEN** kann entweder eine Zeilennummer zur Verzweigung oder eine oder mehrere Anweisungen, die ausgefuehrt werden sollen, folgen.
Hinter **GOTO** steht immer eine Zeilennumer.

Ist das Ergebnis des Ausdruckes falsch (Null), wird die Angabe THEN oder GOTO ignoriert, und die Angabe ELSE, falls vorhanden, wird ausgefuehrt. Die Ausfuehrung geht mit der naechsten ausfuehrbaren Anweisung weiter.

Wird die Anweisung **IF...THEN** im Direktmodus eingegeben, und es wird eine Zeilennummer angesprochen, erhaelt man den Fehler "Undefined Line Number" (Nicht definierte Zeilennummer), ausser es wurde zuvor eine Zeile mit der angegebenen Zeilennummer eingegeben.

Hinweis:

Wird die Anweisung **IF** dazu benutzt, die Gleichheit zweier Werte zu testen, die das Ergebnis einer Berechnung fuer einfache oder doppelte Genauigkeit sind, wird daran erinnert, dass die interne Darstellung der Werte nicht immer genau ist. Dies kommt daher, weil die Werte einfacher und doppelter Genauigkeit intern in binaerer Gleitkommadarstellung gespeichert sind. Deshalb muss dieser Test ueber einen Bereich erfolgen, in dem die Genauigkeit des Wertes variiert. Der Inhalt der Variablen A kann z.B. gegen den Wert 1.0 wie folgt getestet werden:

```
IF ABS(A-1.0)<1.0E-6 THEN...
```

Dieser Test uebergibt ein richtiges Ergebnis, falls der Wert von A gleich 1.0 mit einem relativen Fehler kleiner als 1.0E-6 ist.

Auch muss beachtet werden, dass **IF...THEN...ELSE** eine Anweisung ist. Das heisst die Angabe ELSE kann keine eigene Programmzeile sein.

Beispiel:

```
10 IF A = B THEN X=4
20 ELSE P = Q
```

Dieses Beispiel ist falsch; es muss wie folgt aussehen:

```
10 IF A = B THEN X = 4 ELSE P = Q
```

Verschachtelungen von Anweisungen IF:

Anweisungen **IF...THEN...ELSE** koennen verschachtelt werden. Die Verschachtelung ist nur durch die Laenge der Zeile begrenzt.

Beispiel:

```
IF X>Y THEN PRINT "GROESSER" ELSE IF Y>X
THEN PRINT "KLEINER" ELSE PRINT "GLEICH"
```

Es handelt sich um eine gueltige Anweisung. Enthaelt die Anweisung nicht die gleiche Anzahl von ELSE- und THEN-Angaben, wird ELSE mit dem naechsten unverglichenen THEN verglichen.

Beispiel:

```
IF A = B THEN IF B = C THEN PRINT "A = C"
ELSE PRINT "A<>C"
```

Dieses Beispiel druckt nicht "A<>C", falls A<>B ist, sondern "A = C".

Beispiel:

Dieses Beispiel liest Satz I, falls I nicht Null ist:

```
200 IF I THEN GET #1,I
```

Liegt im naechsten Beispiel I zwischen 10 und 20, wird DB errechnet und zur Zeilennummer 300 verzweigt. Befindet sich I nicht in diesem Bereich, wird die Nachricht **AUSSERHALB DES BEREICHS** angezeigt. Es sollte beachtet werden, dass in der Angabe THEN zwei Anweisungen enthalten sind.

```
100 IF (I>10) AND (I<20) THEN
       DB = 1982-I:GOTO 300
       ELSE PRINT "AUSSERHALB DES BEREICHS"
```

Die naechste Anweisung bewirkt, dass die Ausgabe entweder zum Bildschirm oder zum Drucker geht, abhaengig vom Wert der Variablen (IOFLAG). Ist IOFLAG falsch (Null), geht die Ausgabe zum Drucker; andernfalls geht sie zum Bildschirm.

```
210 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$
```

4.7.9. WHILE und WEND

Fuehrt einen Zyklus von Anweisungen in einer variablen Anzahl von Durchlaeufen aus, d.h.,wenn eine angegebene Bedingung wahr (richtig) ist.

Syntax:

```
WHILE Ausdruck
    .
    .
    .
(Schleifenanweisungen)
    .
    .
    .
 WEND
```

Bemerkungen:

Ausdruck Numerischer Ausdruck

Ist der **Ausdruck** wahr (nicht Null), werden die Schleifenanweisungen solange ausgefuehrt, bis die Anweisung **WEND** kommt. BASIC kehrt jetzt zur Anweisung **WHILE** zurueck und prueft den **Ausdruck**. Ist er immer noch wahr, wird dieser Prozess wiederholt. Ist der Ausdruck nicht wahr, wird mit der Anweisung fortgesetzt, die der Anweisung **WEND** folgt.
Ein Ausdruck ist dann wahr, wenn er als numerischer Ausdruck einen von Null verschiedenen Wert bzw. als logischer Ausdruck den Wahrheitswert "**TRUE**" hat.
WHILE...WEND-Schleifen koennen in allen Ebenen verschachtelt werden. Jedes **WEND** gehoert zum naechstliegenden **WHILE**. Fehlt bei einer Anweisung **WHILE** ein WEND, erscheint der Fehler "WHILE without WEND" (WHILE ohne WEND). Fehlt bei der Anweisung **WEND** eine Anweisung WHILE, erscheint der Fehler "WEND without WHILE" (WEND ohne WHILE).
Innerhalb einer **WHILE**-Schleife stehen nur ausfuehrbare Anweisungen.

Beispiel:

In diesem Beispiel werden die Elemente des Zeichenkettenfeldes A$ alphabetisch geordnet. A$ wurde mit J Elementen definiert.

```
100 'Sortierbereich A%
110 ZYK=1
120 WHILE ZYK
130 ZYK=0
140 FOR I=1 TO J-1
150   IF A%(I)>A%(I+1) THEN
      SWAP A%(I),A%(I+1):ZYK=1
160 NEXT I
170 WEND
```

4.8. Dimensionieren von Feldern

4.8.1. DIM

Vereinbarungsanweisung fuer ein- und mehrdimensionale Felder. Definiert die maximalen Werte fuer die Indizes von Feldvariablen und belegt danach den erforderlichen Speicherplatz.

Syntax: **DIM Variable(Indizes)[,Variable(Indizes)]...**

Bemerkungen:

Variable Der fuer das Feld benutzte Name.

Indizes Eine Liste numerischer Ausdruecke, getrennt durch Kommas, die die Dimensionen des Feldes angeben.

Bei der Ausfuehrung setzt die Anweisung DIM alle Elemente eines angegebenen numerischen Feldes auf 0 (Null). Feldelemente fuer Zeichenketten haben alle eine variable Laenge und zu Beginn keinen Inhalt (Laenge 0).

Es ist moeglich, Felder mit einem Indexbereich zwischen 0 und 9 ohne vorherige **DIM**-Anweisung zu benutzen. Wird ein Index benutzt, der groesser als das angegebene Maximum ist, erhaelt man den Fehler "Subscript out of Range" (Index ausserhalb des Bereichs).

Der kleinste Wert (Standard) fuer einen Index ist immer Null (0). Mit **OPTION BASE 1** (siehe Anweisung "OPTION BASE" in diesem Kapitel) kann die untere Indexgrenze auf 1 festgelegt werden. Die groesste Anzahl von Dimensionen fuer ein Feld ist 255. Die maximale Anzahl von Elementen pro Dimension ist 32767. Beide Angaben sind auch durch die Groesse des Hauptspeichers und die Laenge der Anweisungen begrenzt.

Die Dimensionierung eines Feldes muss vor dem ersten Zugriff zu einem beliebigen Element des Feldes erfolgen.

Wird versucht, ein Feld mehr als einmal zu dimensionieren, tritt der Fehler "Duplicate Definition" (Doppelte Definition) auf. Es ist jedoch moeglich, mit der Anweisung **ERASE** ein Feld zu loeschen, so dass es neu dimensioniert werden kann. Weitere Informationen ueber Felder siehe Kapitel 2.5.3.

Beispiel:

```
Ok
 10  TEXTMAX = 2
 20  DIM TEXT$(TEXTMAX,1)
 30  DATA 26.5,37,8.29,80,9.9,10
 40  FOR I = 0 TO 5
 50  READ NUM(I)
 60  NEXT I
 70  DATA MONTAG,DIENSTAG
 80  DATA MITTWOCH,DONNERSTAG
 90  DATA FREITAG,SONNABEND
100  FOR I = 0 TO 2:FOR J = 0 TO 1
110  READ TEXT$(I,J)
120  NEXT J,I
130  PRINT NUM(2);TEXT$(1,1)
RUN
 8.29 DONNERSTAG
Ok
```

In diesem Beispiel werden zwei Felder erstellt: ein eindimensionales numerisches Feld mit dem Namen NUM mit 6 Elementen - NUM(0) bis NUM(5) - und ein zweidimensionales Feld fuer Zeichenkette mit dem Namen TEXT$, bestehend aus drei Zeilen und zwei Spalten.

4.8.2. OPTION BASE

Definiert die untere Indexgrenze fuer Feldindizes.

Syntax: **OPTION BASE n**

Bemerkungen:

n Wert 1 oder 0

Der Standardwert ist 0 (Null). Wird die Anweisung:

```
OPTION BASE 1
```

ausgefuehrt, darf der kleinste Wert fuer den Feldindex 1 sein.

Die Anweisung **OPTION BASE** muss vor der Definition oder Benutzung irgendeines Feldes stehen.
Ein Fehler wird angezeigt, wenn der Basiswert geaendert wird und Felder existieren.

4.8.3. ERASE

Loescht Felder aus dem Programm.

Syntax: ERASE Feldname[,Feldname]...

Bemerkungen:

Feldname Name eines Feldes, das geloescht werden soll.

Nicht mehr benoetigte Felder koennen geloescht werden, und der Platz im Hauptspeicher, dem die Felder zugeordnet waren, kann fuer andere Zwecke benutzt werden.

ERASE kann auch benutzt werden, um Felder im Programm neu zu dimensionieren. Wird versucht, ein Feld neu zu dimensionieren, bevor es geloescht wurde, erhaelt man den Fehler "Duplicate Definition" (Doppelte Definition).

Mit dem Befehl **CLEAR** koennen alle Variablen aus dem Arbeitsbereich geloescht werden.

Beispiel:

```
Ok
10 START = FRE("")
20 DIM GROESSE(100,100)
30 MITTE = FRE("")
40 ERASE GROESSE
50 DIM GROESSE(10,10)
60 ENDE = FRE("")
70 PRINT START, MITTE, ENDE
RUN
 61973      21142      61452
Ok
```

In diesem Beispiel wird die Funktion FRE benutzt, um zu zeigen, wie mit ERASE der Hauptspeicher geloescht werden kann. Das Feld "GROESSE" benutzte bis zu 40K Bytes des Hauptspeichers (61973 bis 21142), als es mit GROESSE(100,100) dimensioniert war. Nach dem Loeschen konnte es auf GROESSE(10,10) neu dimensioniert werden und benoetigte dafuer etwas mehr als 500 Bytes (61973 bis 61452).

4.9. Anwendereigene Funktionsdefinition DEF FN

Definiert eine einzeilige Funktion, die man selbst schreibt und gibt ihr einen Namen.

Syntax: DEF FN**Name**[(**Arg**[,**Arg**]...)] = **Ausdruck**

Bemerkungen:

Name ist ein gueltiger Variablenname. Dieser Name, dem **FN** vorangestellt wird, wird der Name der Funktion.

Arg ist ein Argument. Dieser Variablenname in der Funktionsdefinition wird durch einen Wert ersetzt, wenn die Funktion aufgerufen wird. Die Argumente in der Liste stellen die Werte, die der Funktion beim Aufruf mitgegeben werden, auf der Basis 1 : 1 dar.

Ausdruck definiert den Wert, den die Funktion uebergibt. Der Typ des **Ausdrucks** (numerisch oder Zeichenkette) muss mit dem Typ uebereinstimmen, der mit **Name** definiert wird.

Die Definition einer Funktion ist auf eine Zeile (255 Zeichen) beschraenkt. Argumente (**Arg**), die in der Funktionsdefinition erscheinen, dienen nur zur Definition der Funktion und sind formale Parameter; sie haben keinen Einfluss auf Programmvariablen mit dem gleichen Namen. Bei Aufruf der Funktion werden die formalen Parameter durch die jeweiligen aktuellen Variablen ersetzt. Ein Variablenname, der in **Ausdruck** benutzt wird, muss nicht in der Liste der Argumente erscheinen. Erscheint er in der Liste, wird der Wert fuer das Argument verwendet, wenn die Funktion aufgerufen wird. Im anderen Fall wird der laufende Wert der Variablen benutzt.

Der Funktionstyp bestimmt, ob die Funktion einen numerischen Wert oder einen Zeichenkettenwert uebergibt. Der Typ der Funktion wird als letztes Zeichen in **Name** deklariert, und zwar auf dieselbe Art, wie Variablen deklariert werden (siehe "Definition von Variablentypen" in Kapitel 3). Stimmt der Typ des Ausdrucks (Zeichenkette oder numerisch) nicht mit dem Funktionstyp ueberein, erhaelt man den Fehler "Type Mismatch" (Keine Typuebereinstimmung). Ist die Funktion numerisch, wird der Wert fuer den Ausdruck in die durch **Name** angegebene Genauigkeit umgewandelt, bevor er an die aufrufende Anweisung uebergeben wird.

Die Vereinbarung einer Funktion muss immer vor ihrem ersten Aufruf erfolgen. Wird eine Funktion aufgerufen, bevor sie definiert ist, erhaelt man den Fehler "Undefined User Function" (Nicht definierte Benutzerfunktion).

Zum anderen darf die Funktion mehr als einmal definiert werden. Die zuletzt ausgefuehrte Definition wird benutzt.

Hinweis:

Man kann auch eine **rekursive** Funktion benutzen, d.h. eine Funktion, die sich selbst aufruft. Wird sie jedoch so geschrieben, dass die Rekursion nicht gestoppt wird, bekommt man den Fehler "Out of Memory" (Hauptspeicherplatz reicht nicht aus).

DEF FN ist in der direkten Betriebsart ungueltig.

Beispiel:

```
Ok
10 PI =  3.141593
20 DEF FNAREA(R) = PI * R^2
30 INPUT "Radius? ", RADIUS
40 PRINT "Flaeche ist" FNAREA(RADIUS)
RUN
Radius?
```

(Mit zwei wird geantwortet.)

```
Radius? 2
Flaeche ist 12.56637
Ok
```

In Zeile 20 wird die Funktion FNAREA definiert, die die Flaeche eines Kreises mit dem Radius R berechnet. Die Funktion wird in Zeile 40 aufgerufen.

Ein weiteres Beispiel mit zwei Argumenten:

```
Ok
10 DEF FNMUD(X,Y) = X-(INT(X/Y)*Y)
20 A = FNMUD(7.4,4)
30 PRINT A
RUN
 3.4
Ok
```

4.10. Programmueberlagerung

4.10.1. CHAIN

Dient dem Aufbau von Ueberlagerungsstrukturen. Es werden Programmsegmente nachgeladen und Variable aus dem laufenden Programm uebergeben.

Syntax: **CHAIN [MERGE] Dateiangabe[,[Zeile] [,[ALL][,DELETE Bereich]]]**

Bemerkungen:

Dateiangabe ist ein Zeichenkettenausdruck fuer Dateispezifikation, folgt den Regeln fuer Dateispezifikationen, wie im Kapitel 6, "Namensgebung fuer Dateien" beschrieben und kann einen Pfadnamen erhalten. Der Dateiname ist der Name des Programms, an das die Steuerung uebergeben wird.

Beispiel:

```
CHAIN "A:PROG1"
```

Zeile Zeilennummer oder Ausdruck, der eine Zeilennummer in dem Programm berechnet, an das die Steuerung uebergeben wird. Damit wird die Zeile angegeben, bei der in dem aufgerufenen Programm die Ausfuehrung beginnen soll. Ist sie weggelassen, beginnt die Ausfuehrung mit der ersten Zeile des aufgerufenen Programms.

Beispiel:

```
CHAIN "A:PROG1",1000
```

Zeile (1000 in diesem Beispiel) wird mit dem Befehl RENUM nicht veraendert. Wird PROG1 neu numeriert, muesste die in dem Beispiel angegebene Anweisung CHAIN geaendert werden, um auf die neue Zeilennummer zu verweisen.

ALL bedeutet, dass jede Variable des laufenden Programms an das aufgerufene uebergeben werden soll. Wird die Angabe **ALL** weggelassen, muss in das aufgerufene Programm eine Anweisung **COMMON** eingefuegt werden, um ausgewaehlte Variable an das aufgerufene Programm aus dem aktuellen Programm zu uebergeben. Siehe Anweisung "**COMMON**" in diesem Kapitel.

Beispiel:

```
CHAIN "A:PROG1",1000,ALL
```

MERGE uebergibt den Programmcode an das BASIC-Programm als Ueberlagerung. Das bedeutet, dass das nachzuladende Programm in das vorhandene Programm eingemischt wird.
Das aufgerufene Programm muss eine Datei im ASCII-Code sein, falls es eingemischt werden soll.

Beispiel:

```
CHAIN MERGE "A:OVRLAY",1000
```

Nachdem eine Ueberlagerung stattgefunden hat, koennen diese Programmzeilen geloescht werden, bevor eine neue Ueberlagerung eingelesen werden kann. Dies erfolgt mit Hilfe der Auswahl **DELETE**, die wie das Kommando DELETE arbeitet. Wie beim Kommando DELETE muessen die Zeilennummern der ersten und letzten

angegebenen Zeile existieren, oder es wird der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf) angezeigt.

Beispiel:

```
CHAIN MERGE "A:OVRLAY2",1000,DELETE 1000-5000
```

In diesem Beispiel werden die Zeilen 1000 bis 5000 des aufrufenden Programms geloescht, bevor die Ueberlagerung (aufgerufenes Programm) geladen wird. Die Zeilennummern in **Bereich** werden von dem Befehl **RENUM** auch neu numeriert.

Hinweise:

1. Die Anweisung CHAIN laesst Dateien eroeffnet.

2. Die Anweisung CHAIN mit der Angabe MERGE erhaelt den Status des letzten Setzens von OPTION BASE.

3. CHAIN ohne MERGE erhaelt nicht die Variablen oder benutzerdefinierten Funktionen fuer die Benutzung im aufgerufenen Programm. Das bedeutet, dass die Anweisungen DEFINT, DEFSNG, DEFDBL, DEFSTR oder DEF FN, die gemeinsame Variablen enthalten, im aufgerufenen Programm erneut auf ihren alten Status gesetzt werden muessen.

4. Die Anweisung CHAIN fuehrt ein RESTORE aus, bevor das aufgerufene Programm ausgefuehrt wird.

4.10.2. COMMON

Uebergibt Variablen aus dem aktuellen Programm an ein aufgerufenes Programm.

Syntax: COMMON Variable[,Variable]...

Bemerkungen:

Variable Name einer Variablen, die an das aufgerufene Programm uebergeben werden soll. Feldvariable werden durch Anfuegen von "()" an den Variablennamen definiert. Es sind alle Variablen aufzufuehren, deren aktuelle Werte in den nachzuladenden Pogrammzeilen zur Verfuegung stehen muessen.

Die Anweisung **COMMON** wird in Verbindung mit der Anweisung CHAIN benutzt. Die Anweisungen COMMON koennen irgendwo im Programm stehen, obwohl es empfohlen wird, sie an den Anfang des Programms zu stellen. Die Anzahl der Anweisungen **COMMON** in einem Programm ist nicht begrenzt, aber die gleiche Variable darf nur einmal in einer Anweisung **COMMON** vorkommen. Sollen alle Variablen uebergeben werden, benutzt man am besten CHAIN mit der Auswahl **ALL** und unterdrueckt die Anweisung **COMMON**.

Die Felder, die uebergeben werden sollen, muessen im aufgerufenen Programm nicht dimensioniert werden.

Beispiel:

Dieses Programm ruft das Programm PROG3 auf der Diskette im Laufwerk A: auf und uebergibt das Feld D und die Variablen A, BER1, C und G$.

```
100 COMMON A, BER1,C,D(),G$
110 CHAIN "A:PROG3"
```

4.11. Fehlerbehandlung

4.11.1. ERROR

Simuliert den Auftritt eines BASIC-Fehlers oder ermoeglicht die Definition eigener Fehlercodes.

Syntax: ERROR n

Bemerkungen:

n muss ein ganzzahliger Ausdruck zwischen 0 und 255 sein.

Entspricht der Wert von n einem von BASIC benutzten Fehlercode (siehe Anhang C "Fehlermeldungen"), simuliert die Anweisung **ERROR** das Auftreten dieses Fehlers. Wurde durch die Anweisung **ON ERROR** eine Fehlerbehandlungsroutine definiert, wird in diese Fehlerroutine verzweigt; andernfalls wird die Fehlermeldung, die diesem Code entspricht, angezeigt, und die Ausfuehrung wird angehalten (siehe erstes Beispiel).

Soll ein eigener Fehlercode definiert werden, muss ein Wert benutzt werden, der in BASIC nicht verwendet wird. (Es wird vorgeschlagen, die hoechsten verfuegbaren Werte zu benutzen; z.B. Werte ueber 200.) Die neuen Fehlercodes koennen dann in Fehlerbehandlungsroutinen getestet werden, genau wie andere Fehler (siehe zweites Beispiel).

Definiert man seinen eigenen Code auf diese Weise, und er wird nicht durch eine Fehlerbehandlungsroutine verarbeitet, zeigt **BASIC** die Nachricht "Unprintable Error" (Nicht druckbarer Fehler) an, und die Ausfuehrung stoppt.

Beispiel:

Das erste Beispiel simuliert den Fehler "String too long" (Zeichenkette zu lang).

```
Ok
10 T = 15
20 ERROR T
RUN
String too long in 20
```

Das naechste Beispiel ist Teil eines Spielprogramms, in dem Werte gesetzt werden koenen. Mit Hilfe des Fehlercodes 210, der in BASIC nicht benutzt wird, faengt das Programm diesen Fehler ab, sobald das Hauslimit ueberschritten wird.

```
110 ON ERROR GOTO 400
120 INPUT "Wieviel setzen Sie";B
130 IF B>5000 THEN ERROR 210
 .
 .
 .
400 IF ERR = 210 THEN PRINT "HAUSLIMIT IST 5000"
410 IF ERL = 130 THEN RESUME 120
```

4.11.2. ERR und ERL

Uebergibt den Fehlercode und die Zeilennummer, in der der Fehler aufgetreten ist.

Syntax: v = ERR
v = ERL

Bemerkungen:

Die Variable ERR enthaelt den Fehlercode fuer den letzten Fehler, die Variable ERL enthaelt die Zeilennummer der Zeile, in der der Fehler entdeckt wurde. Die Variablen ERR und ERL werden gewoehnlich in den Anweisungen IF...THEN benutzt, um den Programmablauf zur Fehlerbehandlungsroutine verzweigen zu lassen (siehe Anweisung "ON ERROR" in diesem Kapitel).

Wird ERL in der Anweisung IF...THEN getestet, muss die Zeilennummer auf der rechten Seite des logischen Operators stehen:

```
IF ERL = Zeilennummer THEN...
```

Die Nummer muss auf der rechten Seite des Operators stehen, damit sie mit RENUM neu numeriert wird.

War die Anweisung, die den Fehler verursachte, eine Anweisung in der direkten Betriebsart, enthaelt ERL den Wert 65535. Soll sich die Zeilennummer waehrend RENUM nicht aendern und man

moechte pruefen, ob sich ein bestimmter Fehler in einer Anweisung in der direkten Betriebsart ereignete, muss man folgende Form waehlen:

```
IF 65535 = ERL THEN...
```

ERR und ERL koennen mit Hilfe der Anweisung **ERROR** gesetzt werden (siehe Anweisung ERROR).

BASIC-Fehlercodes sind im Anhang C "Fehlernachrichten" aufgelistet.

Beispiel:

```
10 ON ERROR GOTO 100
20 LPRINT "Auf dem Drucker wird gedruckt"
30 END
100 IF ERR = 27 THEN LOCATE 23,1:
    PRINT "Drucker ueberpruefen":RESUME
```

In diesem Beispiel wird ein haeufiger Fehler getestet:
Papier fehlt im Drucker, oder er ist nicht eingeschaltet.

4.11.3. ON ERROR GOTO

Aktiviert die Fehlerunterbrechung und gibt die erste Zeile des Fehlerbehandlungsunterprogramms an.

Syntax: ON ERROR GOTO Zeile

Bemerkungen:

Zeile Zeilennummer der ersten Zeile der Fehlerunterbrechungsroutine. Fehlt diese Zeilennummer, erhaelt man den Fehler "Undefined Line Number" (Nicht definierte Zeilennummer).

Wurde die Fehlerunterbrechung aktiviert, wird fuer jeden entdeckten Fehler (einschliesslich Fehler in der direkten Betriebsart) in das angegebene Fehlerbehandlungsunterprogramm verzweigt.

Um die Fehlerunterbrechung zu inaktivieren, muss ON ERROR GOTO 0 ausgefuehrt werden. Fuer nachfolgende Fehler wird eine Fehlernachricht angezeigt und die Ausfuehrung angehalten. Steht diese Anweisung im Unterprogramm fuer Fehlerbehandlung, haelt BASIC an. Die Fehlernachricht fuer den Fehler, der die Unterbrechung verursachte, wird angezeigt. Es wird empfohlen, dass alle Fehlerbehandlungsunterprogramme den Befehl ON ERROR GOTO 0 beinhalten, falls ein Fehler auftritt, der nicht behoben werden kann.

Hinweis:

Tritt ein Fehler waehrend der Ausfuehrung des Fehlerbehandlungsunterprogramms auf, wird die BASIC-Fehlernachricht angezeigt und die Ausfuehrung abgebrochen. Im Fehlerbehandlungsunterprogramm erfolgt keine Fehlerunterbrechung.

Mit Hilfe der Anweisung **RESUME** verlaesst man die Fehlerbehandlungsroutine. Siehe Anweisung "RESUME" in diesem Kapitel. Mit der Anweisung "GOTO" ist dies nicht moeglich.

4.11.4. RESUME

Faehrt mit der Programmausfuehrung fort, nachdem eine Fehlerbehandlungsroutine ausgefuehrt wurde.

Syntax:
```
RESUME [0]
RESUME NEXT
RESUME Zeile
```

Bemerkungen:

Jedes der oben angezeigten Formate kann benutzt werden, abhaengig davon, wo die Ausfuehrung wieder fortsetzen soll:

RESUME oder RESUME 0
Die Ausfuehrung wird mit der Anweisung, durch die der Fehler verursacht wurde, wieder aufgenommen.

Hinweis:

Wird versucht, ein Programm neu zu numerieren, das die Anweisung **RESUME 0** enthaelt, tritt dabei der Fehler "Undefined Line Number" (Undefinierte Zeilennummer) auf. Beim Starten des Programms wird die Anweisung **RESUME 0** trotzdem richtig abgearbeitet.

RESUME NEXT Die Ausfuehrung wird nach der fehlerhaften Anweisung fortgesetzt.

RESUME Zeile Die Ausfuehrung geht an der angegebenen Zeilennummer weiter.

Die Anweisung **RESUME** darf nur in einer Fehlerbehandlungsroutine stehen, sonst erscheint die Nachricht "RESUME without Error" (RESUME ohne Fehler).

Beispiel:

Zeile 900 ist der Beginn der Fehlerbehandlungsroutine. Durch die Anweisung RESUME wird zu Zeile 80 verzweigt, falls der Fehler 230 in Zeile 90 auftrat.

```
10 ON ERROR GOTO 900
 .
 .
 .
900 IF (ERR = 230) AND (ERL = 90) THEN PRINT
    "NEUER VERSUCH":RESUME 80
```

4.12. Unterbrechungsabfrage

4.12.1. KEY(n)

Aktiviert und inaktiviert die Unterbrechung fuer eine angegebene Taste in einem BASIC-Programm. Siehe "Anweisung ON KEY(n)" in diesem Kapitel.

Syntax:

```
KEY(n) ON
KEY(n) OFF
KEY(n) STOP
```

Bemerkungen:

n — ist ein numerischer Ausdruck im Bereich 1 bis 14 und gibt die Taste an, durch die eine Unterbrechung erfolgen soll.

1 bis 10	Funktionstasten F1 bis F10
11	Kursor nach oben
12	Kursor nach links
13	Kursor nach rechts
14	Kursor nach unten
15 bis 20	Tasten wie folgt definiert: KEY(n),CHR$(Tastenkennzeichen)+CHR$(Suchcode).

KEY(n) ON muss ausgefuehrt werden, um das Unterbrechen fuer eine Funktionstaste oder fuer eine Kursorsteuertaste zu aktivieren. Nach Ausfuehrung der Anweisung KEY(n) ON wird jedesmal, wenn BASIC eine neue Anweisung ausfuehrt, geprueft, ob die angegebene Taste betaetigt wurde. Wurde eine dieser Tasten betaetigt, wird ein GOSUB zu der Zeilennummer ausgefuehrt, die in der Anweisung **ON KEY(n)** angegeben wurde. Eine Anweisung **KEY(n)** kann nicht vor der Anweisung **ON KEY(n)** stehen.

Wird **KEY(n) OFF** ausgefuehrt, findet keine Unterbrechung statt. Wenn eine Taste betaetigt wurde, wird dies nicht bemerkt.

Wurde die Anweisung **KEY(n) STOP** ausgefuehrt, findet keine Unterbrechung statt. Wird jedoch eine der angegebenen Tasten betaetigt, findet eine sofortige Unterbrechung statt, wenn die Anweisung **KEY(n) ON** ausgefuehrt wird.

KEY(n) ON hat keinen Einfluss auf die Anzeige der Funktionstastenwerte auf der untersten Bildschirmzeile.

Unter Kapitel 4.5.7. "Anweisung KEY" steht die Erklaerung ueber die Anweisung KEY ohne (n).

4.12.2. ON KEY(n)

Setzt fuer BASIC eine Zeilennummer, zu der bei einer Unterbrechung verzweigt werden soll, falls eine angegebene Funktionstaste oder Kursortaste betaetigt wurde.

Syntax: **ON KEY(n) GOSUB Zeile**

Bemerkungen:

n ist ein numerischer Ausdruck im Bereich 1 bis 20, der die Taste anzeigt, die eine Unterbrechung erzeugen soll.

1 bis 10	Funktionstasten F1 bis F10
11	Kursor nach oben
12	Kursor nach links
13	Kursor nach rechts
14	Kursor nach unten
15 bis 20	Tasten wie folgt definiert: KEY n,CHR$(Tastenkennzeichen)+CHR$(Suchcode)

Die Tasten 15 bis 20 erzeugen eine Unterbrechung. Weitere Informationen dazu siehe Anweisungen KEY und KEY(n).

Zeile Zeilennummer des Beginns der Unterbrechungsroutine fuer die angegebene Taste. Wird **Zeile** auf 0 (Null) gesetzt, wird die Unterbrechung fuer diese Taste inaktiviert.

Die Anweisung **KEY(n) ON** muss ausgefuehrt werden, um diese Anweisung zu aktivieren. Nach KEY(n) ON wird -falls in ON KEY(n) eine Zeilennummer ungleich 0 (Null) angegeben wurde- von BASIC vor dem Beginn jeder neuen Anweisung geprueft, ob die angegebene Taste betaetigt wurde. War dies der Fall, fuehrt BASIC ein GOSUB zur angegebenen **Zeile** aus.

Wird **KEY(n) OFF** ausgefuehrt, findet fuer die angegebene Taste keine Unterbrechung statt. Auch wenn die Taste betaetigt wurde, wird dies nicht gespeichert.

Wurde die Anweisung KEY(n) STOP ausgefuehrt, findet fuer die Taste keine Unterbrechung statt. Wurde die Taste jedoch betaetigt, wird dies gespeichert, und sobald KEY(n) ON ausgefuehrt wird, tritt eine sofortige Unterbrechung ein.

Nach der Unterbrechung wird automatisch KEY(n) STOP ausgefuehrt, so dass eine rekursive Unterbrechung nicht auftreten kann. Die Anweisung RETURN aus der Unterbrechungsroutine fuehrt automatisch ein KEY(n) ON aus, ausser es wurde explizit innerhalb der Unterbrechungsroutine KEY(n) OFF ausgefuehrt.

Unterbrechungen koennen nicht auftreten, wenn BASIC kein Programm ausfuehrt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung ON ERROR), wird jede Unterbrechung automatisch inaktiviert (einschliesslich ON COM, ON ERROR, ON PLAY und ON TIMER).

Es ist moeglich, dass die Tastenunterbrechung nicht arbeitet, wenn andere Tasten vor der angegebenen betaetigt wurden. Die Taste, die die Unterbrechung verursachte, kann nicht mit Hilfe von INPUT$ oder INKEY$ getestet werden, so dass die Unterbrechungsroutine fuer jede Taste unterschiedlich sein muss, falls eine andere Funktion gewuenscht wird.

Es ist moeglich, mit RETURN und Angabe einer Zeilennummer im BASIC-Programm zu einer festen Zeilennummer zurueckzuverzweigen. Jedoch muss dies mit Vorsicht getan werden, da andere GOSUBs, WHILEs und FORs zur Zeit der Unterbrechung aktiv sind und aktiv bleiben.

KEY(n) ON hat keinen Einfluss darauf, ob die Funktionstastenwerte in der untersten Zeile des Bildschirms angezeigt werden.

Beispiel:

Dieses Beispiel zeigt eine Unterbrechungsroutine fuer die Funktionstaste 5:

```
100 ON KEY(5) GOSUB 200
110 KEY(5) ON
 .
 .
 .
200 REM FUNKTIONSTASTE 5 BETAETIGT
 .
 .
 .
290 RETURN 140
```

4.12.3. ON TIMER

Uebergibt nach Ende einer angegebenen Zeitspanne die Steuerung an eine angegebene Zeilennummer.

Syntax: **ON TIMER(n) GOSUB Zeile**

Bemerkungen:

n ist ein numerischer Ausdruck im Bereich von 1 bis 86400 (1 Sekunde bis 24 Stunden). Werden Werte zugeordnet, die sich auserhalb dieses Bereiches befinden, so erscheint der Fehler "Illegal function call"(Ungueltiger Funktionsaufruf).

Zeile ist die erste Zeilennummer der Unterbrechungsroutine fuer TIMER. Eine Zeilennummer 0 (Null) stoppt die Unterbrechung fuer TIMER.

Die Anweisung **TIMER ON** muss benutzt werden, um die Anweisung **ON TIMER** zu starten. Nach **TIMER ON** prueft BASIC die vorhergehenden Sekunden, falls in der Anweisung **ON TIMER(n)** eine Zeilennummer ungleich Null angegeben war. Nachdem **n** Sekunden vergangen sind, fuehrt BASIC ein GOSUB zur angegebenen Zeile aus. Die Unterbrechungsverarbeitung beginnt, und BASIC beginnt erneut mit dem Zaehlen ab Null.

Wird **TIMER OFF** benutzt, findet keine Unterbrechung statt. Auch wenn **TIMER** ausgefuehrt wird, wird dies nicht gespeichert.

Wird die Anweisung **TIMER STOP** ausgefuehrt, findet keine Unterbrechung statt. TIMER-Aktivitaet wird jedoch gespeichert und eine sofortige Unterbrechung tritt ein, sobald **TIMER ON** ausgefuehrt wurde.

Nach der Unterbrechung wird automatisch **TIMER STOP** ausgefuehrt, so dass keine rekursive Unterbrechung stattfinden kann. Die Anweisung **RETURN** aus der Unterbrechungsroutine fuehrt automatisch ein **TIMER ON** aus, ausser es wurde explizit in der Unterbrechungsroutine **TIMER OFF** ausgefuehrt.

Unterbrechungen koennen nicht auftreten, wenn BASIC kein Programm ausfuehrt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung **ON ERROR**), wird jede Unterbrechung automatisch inaktiviert (einschliesslich **ON COM**, **ON ERROR**, **ON PLAY** und **ON TIMER**).

Es ist moeglich, mit **RETURN Zeile** im BASIC-Programm zu einer festen Zeilennummer zu verzweigen. Jedoch muss das mit Vorsicht getan werden, da andere **GOSUB**,**WHILE** und **FOR**-Anweisungen, die zur Zeit der Unterbrechung aktiv sind, aktiv bleiben.

Aktive Schleifen werden verlassen, indem die Schleifenzaehlervariable ausserhalb des gueltigen Bereiches angegeben wird oder innerhalb der Schleife eine bedingte Anweisung gesetzt wird, die die Schleife stoppt. Dadurch wird sichergestellt, dass jede Wiederholung eines **FOR** ein entsprechendes **NEXT** hat und jede Wiederholung eines **WHILE** ein entsprechendes **WEND** hat.

ON TIMER ist in Programmen nuetzlich, die einen Intervallzeitgeber benoetigen, z.B. um die Zeit jede Minute in Zeile I anzuzeigen.

Beispiel:

```
10 CLS
20 ON TIMER(60) GOSUB 10000
30 TIMER ON
 :
 :
10000 ZEILE=CSRLIN'laufende Zeile speichern
10010 SPALTE=POS(0)'laufende Spalte speichern
10020 LOCATE 1,1:PRINT TIME$;
10040 LOCATE ZEILE,SPALTE'Zeile und Spalte zurueckspeichern
10040 RETURN
```

4.12.4. ON PLAY(n)

sh. Kapitel 8.5.2.

4.12.5. COM(n) und ON COM(n)

sh. Kapitel 7.3. Unterbrechungsabfrage Datenfernverarbeitung

5. BASIC-Grundfunktionen

Diese Funktionen koennen in einem BASIC-Programm aufgerufen werden, ohne dass sie vorher definiert werden muessen.
Die Argumente der Funktionen werden immer in Klammern gesetzt. Durch Kombination von Standardfunktionen koennen weitere gelaeufige mathematische Funktionen realisiert werden (siehe Anlage F).

Hinweis:

Numerische Funktionen uebergeben nur ganze Zahlen oder Werte einfacher Genauigkeit. Funktionen mit doppelter Genauigkeit werden nur vom BASIC-Compiler unterstuetzt.

Ausnahme: Die Exponential- und trigonometrischen Funktionen koennen auch in doppelter Genauigkeit berechnet werden, wenn beim Laden des BASIC-Interpreters die Angabe /D erfolgt.

5.1. Numerische Funktionen

5.1.1. Arithmetische Funktionen

= ABS

Uebergibt den absoluten Wert des Ausdrucks x.

Syntax: `v = ABS(x)`

Bemerkungen:

x ist ein numerischer Ausdruck. Der absolute Wert einer Zahl ist immer positiv oder Null.

Beispiel:

```
Ok
PRINT ABS(7*(-5))
 35
Ok
```

Der absolute Wert von -35 ist positiv 35.

= INT

Uebergibt die groesste ganze Zahl, die kleiner oder gleich x ist.

Syntax: `v = INT(x)`

Bemerkungen:

x ist ein numerischer Ausdruck.

Siehe Funktionen FIX und CINT; diese uebergeben ebenfalls ganzzahlige Werte.

Beispiel:

```
Ok
PRINT INT(145.83)
 145
Ok
PRINT INT(-3.75)
-4
Ok
```

In diesem Beispiel wird gezeigt, dass INT positive ganzzahlige Werte abschneidet, aber negative Werte aufrundet (in negativer Richtung).

=_SGN

Ermittelt das Vorzeichen von x.

Syntax: v = SGN(x)

Bemerkungen:

x ist ein numerischer Ausdruck.

SGN(x) ist die mathematische Signumfunktion:
- Ist x positiv, uebergibt SGN(x) eine 1.
- Ist x gleich Null (0), uebergibt SGN(x) eine 0.
- Ist x negativ, uebergibt SGN(x) eine -1.

Beispiel:

```
ON SGN(x)+2 GOTO 100,200,300
```

In dieser Anweisung wird nach 100 verzweigt, falls X negativ, nach 200, falls X gleich 0 und nach 300, falls X positiv ist.

=_FIX

Schneidet x auf eine ganze Zahl ab.

Syntax: v = FIX(x)

Bemerkungen:

x ist ein numerischer Ausdruck.

FIX schneidet alle Ziffern rechts des Dezimalpunkts ab und uebergibt den Wert der Ziffern, die links des Dezimalpunkts stehen.

Der Unterschied zwischen FIX und INT ist der, dass FIX nicht die naechstkleinere Zahl uebergibt, falls x negativ ist.

Siehe auch die Funktionen **INT** und **CINT**, die ebenfalls ganze Zahlen uebergeben.

Beispiel:

```
Ok
PRINT FIX(145.83)
 145
Ok
PRINT FIX(-3.75)
-3
Ok
```

In diesen Beispielen muss beachtet werden, dass FIX **nicht** rundet, wenn es in eine ganze Zahl umwandelt.

5.1.2. Konvertierung von Zahlenwerten

CINT

Wandelt x in eine ganze Zahl um.

Syntax: v = CINT(x)

Bemerkungen:

x kann jeder numerische Ausdruck sein. Liegt x nicht im Bereich -32768 bis 32767, wird der Fehler "Overflow" (Ueberlauf) angezeigt.

x wird in eine ganze Zahl umgewandelt, indem die Zahlen hinter dem Dezimalpunkt gerundet werden.

Auch die Funktionen **FIX** und **INT** uebergeben ganze Zahlen. Siehe auch die Funktionen **CDBL** und **CSNG**, die Zahlen in einfache und doppelte Genauigkeit umwandeln.

Beispiel:

```
Ok
PRINT CINT(45.67)
 46
Ok
PRINT CINT(-2.89)
-3
Ok
```

sollte beachtet werden, wie in den beiden Beispielen ndet wird.

CSNG

Wandelt x in eine Zahl einfacher Genauigkeit um.

Syntax: v = CSNG(x)

Bemerkungen:

x ist ein numerischer Ausdruck, der in eine Zahl einfacher Genauigkeit umgewandelt wird.

Siehe auch die Funktionen **CINT** und **CDBL** fuer die Umwandlung von Zahlen in ganzzahlige Werte und Werte doppelter Genauigkeit.

Beispiel:

```
Ok
10 A# = 975.3421222#
20 PRINT A#; CSNG(A#)
RUN
 975.3421222  975.3421
Ok
```

Der Wert der Zahl A# mit doppelter Genauigkeit wird an der siebenten Stelle gerundet und als **CSNG(A#)** uebergeben.

CDBL

Wandelt x in eine Zahl doppelter Genauigkeit um.

Syntax: v = CDBL(x)

Bemerkungen:

x kann jeder numerische Ausdruck sein.

Die Funktionen **CINT** und **CSNG** wandeln Zahlen in ganze Zahlen und in Zahlen einfacher Genauigkeit um.

Beispiel:

```
Ok
10 A = 454.67
20 PRINT A; CDBL(A)
RUN
 454.67  454.6700134277344
Ok
```

Der Wert von **CDBL(A)** ist nach dem Runden nur auf die zweite Dezimalstelle genau. Die anderen Ziffern haben keine Bedeutung, weil A nur zwei Dezimalstellen Genauigkeit liefert.

5.1.3. Exponential-Funktionen

- EXP

Errechnet die Exponentialfunktion.

Syntax: v = EXP(x)

Bemerkungen:

x ist ein numerischer Ausdruck.

Die Funktion uebergibt die mathematische Zahl e hoch x, wobei e die Basis fuer den natuerlichen Logarithmus ist. Ein Ueberlauf geschieht, falls x groesser als 8.02969 ist.

Beispiel:

```
Ok
10 X = 2
20 PRINT EXP(X-1)
RUN
 2.718282
Ok
```

Das Beispiel berechnet e hoch (2-1), was einfach e ist.

- LOG

Uebergibt den natuerlichen Logarithmus von x.

Syntax: v = LOG(x)

Bemerkungen:

x muss ein numerischer Ausdruck sein, der groesser als 0 (Null) ist.

Der natuerliche Logarithmus ist der Logarithmus zur Basis e.

Beispiele:

Im ersten Beispiel wird der Logarithmus des Ausdrucks 45/7 berechnet:

```
Ok
PRINT LOG(45/7)
 1.860752
Ok
```

Im zweiten Beispiel wird der Logarithmus von e errechnet.

```
Ok
E=2.718282
Ok
? LOG(E)
 1
Ok
```

=_SQR

Uebergibt die Quadratwurzel von x.

Syntax: v = SQR(x)

Bemerkungen:

x muss groesser oder gleich Null sein.

Beispiele:

```
Ok
10 FOR X=10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT
RUN
 10          3.162278
 15          3.872984
 20          4.472136
 25          5
Ok
```

In diesem Beispiel werden die Quadratwurzeln der Zahlen 10, 15, 20 und 25 errechnet.

```
Ok
PRINT SQR(3#)
 1.732050807568877
Ok
```

Die Quadratwurzel wird mit doppelter Genauigkeit berechnet.

5.1.4. Trigonometrische Funktionen

=_SIN

Errechnet den Sinuswert fuer einen Wert im Bogenmass.

Syntax: v = SIN(x)

Bemerkungen:

x ist der Winkel im Bogenmass.

Um Gradmasse in Bogenmasse umzuwandeln, muss man mit PI/180 multiplizieren, wobei PI = 3.141593 ist.

SIN(x) wird standardmaessig in einfacher Genauigkeit berechnet.

Beispiel:

```
10 PI=3.141593
20 GRAD=90
30 BOGENMASS=GRAD * PI/180
40 PRINT SIN(BOGENMASS)
RUN
 1
Ok
```

In diesem Beispiel wird der Sinus von 90 Grad errechnet, nachdem die Grade in Bogenmass umgewandelt wurden.

COS

Errechnet und uebergibt den Cosinuswert.

Syntax: v = COS(x)

Bemerkungen:

x ist der Winkel, dessen Cosinus errechnet werden soll. Der Wert von x muss in Bogenmass vorliegen. Um von Grad in das Bogenmass umzuwandeln, muessen die Grade mit PI/180 multipliziert werden, wobei PI = 3.141593 ist.

Die Berechnung des COS(x) wird standardmaessig in einfacher Genauigkeit durchgefuehrt.

Beispiel:

```
Ok
10 PI = 3.141593
20 PRINT COS(PI)
30 GRAD = 180
40 BOGEN = GRAD * PI/180
50 PRINT COS(BOGEN)
RUN
-1
-1
Ok
```

Das Beispiel zeigt zuerst, dass der Cosinus von PI im Bogenmass -1 ist. Dann wird der Cosinus von 180 Grad berechnet, indem zuerst die Grade in das Bogenmass umgewandelt werden (dabei ist 180 Grad das gleiche wie PI im Bogenmass).

=_TAN

Uebergibt den Tangens von x.

Syntax: v = TAN(x)

Bemerkungen:

x ist ein Winkel im Bogenmass. Um Grade in Bogenmass umzuwandeln, muss mit PI/180 multipliziert werden, wobei PI = 3.141593 ist.

TAN(x) wird standardmaessig mit einfacher Genauigkeit berechnet.

Beispiel:

```
10 PI=3.141593
20 GRAD=45
30 PRINT TAN(GRAD * PI/180)
RUN
 1
Ok
```

In diesem Beispiel wird der Tangens von 45 Grad berechnet.

=_ATN

Uebergibt den Arcustangens von x.

Syntax: v = ATN(x)

Bemerkungen:

x ist ein numerischer Ausdruck jedes numerischen Typs. Die Errechnung des **ATN** wird standardmaessig in einfacher Genauigkeit ausgefuehrt.

Das Ergebnis der Funktion **ATN** ist ein Wert im Bogenmass im Bereich -PI/2 bis PI/2, wobei PI = 3.141593 ist.

Soll Bogenmass in Gradmass umgewandelt werden, so muss der Wert mit 180/PI multipliziert werden.

Beispiel:

```
Ok
PRINT ATN(3)
 1.249046
Ok

10 PI=3.141593
20 BOGENMASS=ATN(1)
30 GRAD=BOGENMASS * 180/PI
40 PRINT BOGENMASS,GRAD
RUN
 .7853983       45
Ok
```

Das erste Beispiel zeigt die Benutzung der Funktion ATN zur Berechnung des Arcustangens von 3. Das zweite Beispiel errechnet den Winkel, dessen Tangens 1 ist. Es ist im Bogenmass 0.7853983 oder 45 Grad.

5.1.5. Zeichenkettenbezogene numerische Funktionen

ASC

Uebergibt den ASCII-Code fuer das erste Zeichen der Zeichenkette x$.

Syntax: v = ASC(x$)

Bemerkungen:

x$ ist ein Zeichenkettenausdruck.

Das Ergebnis der Funktion ASC ist ein numerischer Wert, der den ASCII-Code fuer das erste Zeichen der Zeichenkette x$ repraesentiert (siehe Anlage D "ASCII-Zeichencodes"). Enthaelt x$ keinen Wert, so wird die Fehlermeldung "Illegal Function Call" (Ungueltiger Funktionsaufruf) uebergeben.

Die Funktion CHR$ ist das Gegenteil der Funktion ASC und wird dazu benutzt, den ASCII-Code in ein Zeichen umzuwandeln.

Beispiel:

```
Ok
10 x$="TEST"
20 PRINT ASC(x$)
RUN
 84
Ok
```

Das Beispiel zeigt, dass der ASCII-Code fuer das grosse T den Wert 84 hat. Print ASC("TEST") gibt dasselbe Ergebnis.

=_INSTR

Sucht in einer vergebenen Zeichenkette x$ eine andere vorgegebene Zeichenkette y$ und ermittelt die Position, an der die Zeichenkette y$ gefunden wurde. Der wahlfreie Relativzeiger n setzt die Position fest, an deren Stelle mit dem Suchen in x$ begonnen werden soll.

Syntax: v = INSTR([n,]x$,y$)

Bemerkungen:

n ist ein numerischer Ausdruck im Bereich 1 bis 255.

x$, y$ koennen Zeichenkettenvariablen, Zeichenkettenausdruecke oder Zeichenkettenkonstanten sein.

Ist n groesser LEN(x$) oder ist x$ leer oder kann y$ nicht gefunden werden, uebergibt **INSTR** eine 0 (Null). Ist y$ leer, uebergibt **INSTR** n (oder 1, falls n nicht angegeben ist).

Liegt n nicht im vorgegebenen Bereich, wird der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf) uebergeben.

Beispiel:

```
10 A$="ABCDEB"
20 B$="B"
30 PRINT INSTR(A$,B$);INSTR(4,A$,B$)
RUN
 2  6
Ok
```

In diesem Beispiel wird die Zeichenkette "B" in der Zeichenkette "ABCDEB" gesucht. Wird die Zeichenkette von Anfang an durchsucht, wird "B" an Position 2 gefunden. Wird die Zeichenkette erst ab Position 4 durchsucht, wird "B" an Stelle 6 gefunden.

=_LEN

Ermittelt die Anzahl der Zeichen der Zeichenkette x$.

Syntax: v = LEN(x$)

Bemerkungen:

x$ ist ein Zeichenkettenausdruck

Nicht druckbare Zeichen und Leerzeichen sind in der Anzahl der Zeichen enthalten.

Beispiel:

```
10 x$="VEB R-BWK, KMST"
20 PRINT LEN(x$)
RUN
 15
Ok
```

In der Zeichenkette "VEB R-BWK, KMST" sind 15 Zeichen enthalten, weil das Komma und die Leerstellen mitgezaehlt werden.

VAL

Uebergibt den numerischen Wert der Zeichenkette x$.

Syntax: v = VAL(x$)

Bemerkungen:

x$ ist ein Zeichenkettenausdruck.

Die Funktion VAL entfernt Leerstellen, Tabulatoren und Zeilenvorschubzeichen aus der Zeichenkette des Arguments, um das Ergebnis bestimmen zu koennen.

Beispiel:

```
VAL("    -3")      Es wird -3 uebergeben.
```

Sind die ersten Zeichen von x$ nicht numerisch, uebergibt VAL(x$) eine 0 (Null).

Siehe auch die Funktion **STR$** fuer die Umwandlung einer numerischen Variablen in eine Zeichenkette.

Beispiel:

```
Ok
PRINT VAL("9071 KARL-MARX-STADT")
 9071
Ok
```

In diesem Beispiel wird mit Hilfe von VAL die Postleitzahl aus einer Adresse herausgezogen.

5.2. Zeichenkettenfunktionen

5.2.1. Allgemeine Zeichenkettenfunktionen

CHR$

Wandelt den Wert eines ganzzahligen Ausdrucks in das entsprechende ASCII-Zeichen um.

Syntax: v$=CHR$(n)

Bemerkungen:

n muss im Bereich 0 bis 255 liegen.

Die Funktion **CHR$** uebergibt die Zeichenkette, bestehend aus einem Zeichen fuer den ASCII-Code n (ASCII-Codes sind in der Anlage D "ASCII-Zeichencodes" aufgelistet). **CHR$** wird normalerweise dazu benutzt, ein Sonderzeichen auf den Bildschirm oder Drucker zu uebertragen. Das Zeichen BEL, durch das der Lautsprecher ertoent, kann zB. als Anfang einer Fehlermeldung mit **CHR$(7)** (statt BEEP) eingefuegt werden. Die Funktion **ASC** zeigt, wie ein Zeichen zurueck in seinen ASCII-Code umgewandelt werden kann.

Beispiel:

```
Ok
PRINT CHR$(65)
A
Ok
```

Im naechsten Beispiel wird der Funktionstaste F1 die Zeichenkette "AUTO", verbunden mit dem Zeichen fuer Eingabe, zugeordnet. Dies ist eine gute Loesung fuer das Setzen der Funktionstasten. Die Wirkung der Taste <ENTER> erfolgt automatisch, wenn die Funktionstaste gedrueckt wird.

```
Ok
KEY 1,"AUTO"+CHR$(13)
Ok
```

HEX$

Uebergibt eine Zeichenkette, die den hexadezimalen Wert eines dezimalen Arguments darstellt.

Syntax: v$ = HEX$(n)

Bemerkungen:

n ist ein numerischer Ausdruck im Bereich -32768 bis 65535.

Ist n negativ, wird das Zweierkomplement benutzt. Das heisst, HEX$(-n) ist das gleiche wie HEX$(65536-n).

Beispiel:
Im folgenden Beispiel wird die Funktion **HEX$** benutzt, um die hexadezimale Darstellung von zwei eingegebenen dezimalen Werten zu errechnen.

```
Ok
10 INPUT X
20 A$=HEX$(X)
30 PRINT X "DEZIMAL IST " A$ " HEXADEZIMAL"
Ok
RUN
? 34
 34 DEZIMAL IST 22 HEXADEZIMAL
```

_OCT$

Uebergibt eine Zeichenkette, die den oktalen Wert eines dezimalen Arguments darstellt.

Syntax: v$ = OCT$(n)

Bemerkungen:

n ist ein numerischer Ausdruck im Bereich -32768 bis 65535.

Ist n negativ, wird das Zweier-Komplement benutzt. Das bedeutet, OCT$(-n) ist das gleiche wie OCT$(65536-n).

Siehe auch Funktion **HEX$** fuer hexadezimale Umwandlung.

Beispiel:

```
Ok
PRINT OCT$(24)
 30
Ok
```

Dieses Beispiel zeigt, dass 24 dezimal 30 in oktaler Schreibweise ist.

_STR$

Wandelt den Wert eines numerischen Ausdrucks in eine Zeichenkette um.

Syntax: v$ = STR$(x)

Bemerkungen:

x ist ein numerischer Ausdruck.

Ist x positiv, enthaelt die mit **STR$** uebergebene Zeichenkette eine fuehrende Leerstelle (die Leerstelle ist fuer das Plusvorzeichen reserviert).

Beispiel:

```
Ok
PRINT STR$(346); LEN(STR$(346))
 346 4
Ok
```

Die Funktion VAL ist die Gegenfunktion von **STR$** (siehe Funktion VAL).

Beispiel:
Das Beispiel verzweigt zu verschiedenen Abschnitten des Programms, abhaengig von der Anzahl der Ziffern einer eingegebenen Zahl. Die Ziffern der Zahl werden dadurch gezaehlt, dass man mit Hilfe von **STR$** die Zahl in eine Zeichenkette umwandelt und dann abhaengig von der Laenge der Zeichenkette verzweigt.

```
10 INPUT "NUMMER EINGEBEN";N
20 ON LEN(STR$(N))-1 GOSUB 30,1000,1100,1200
```

LEFT$

Bildet eine Teilzeichenkette der Laenge n aus der Zeichenkette x$, links beginnend.

Syntax: v$ = LEFT$(x$,n)

Bemerkungen:

x$ ist ein Zeichenkettenausdruck.

n ist ein numerischer Ausdruck im Bereich 0 bis 255. Er gibt die Anzahl der Zeichen an, die das Ergebnis enthalten soll.

Ist n groesser als **LEN(x$)**, wird die gesamte Zeichenkette (x$) uebergeben. Ist n gleich Null, wird eine leere Zeichenkette (Laenge Null) uebergeben.

Siehe auch die Funktionen **MID$** und **RIGHT$**.

Beispiel:

```
Ok
10 A$="BASIC PROGRAMM"
20 B$=LEFT$(A$,5)
30 PRINT B$
RUN
BASIC
Ok
```

In diesem Beispiel werden mit der Funktion **LEFT$** die ersten fuenf Zeichen aus der Zeichenkette "BASIC PROGRAMM" herausgenommen.

- RIGHT$

Bildet eine Teilzeichenkette der Laenge n aus der Zeichenkette x$, rechts beginnend.

Syntax: v$ = RIGHT$(x$,n)

Bemerkungen:

x$ ist ein Zeichenkettenausdruck.

n ist ein ganzzahliger Ausdruck, der die Anzahl der Zeichen angibt, die im Ergebnis stehen sollen.

Ist n groesser oder gleich LEN(x$), wird x$ uebergeben. Ist n gleich Null, wird eine leere Zeichenkette (Laenge Null) uebergeben.

Siehe auch die Funktionen MID$ und LEFT$.

Beispiel:

```
Ok
10 A$="DIE BASIC-ANWEISUNG"
20 PRINT RIGHT$(A$,9)
RUN
ANWEISUNG
Ok
```

Die am weitesten rechts stehenden neun Zeichen der Zeichenkette A$ werden uebergeben.

- MID$ (Funktion und Anweisung)

Waehlt eine Teilzeichenkette aus einer angegebenen Zeichenkette aus. Wenn -wie im zweiten Format- als Anweisung benutzt, wird ein Teil einer Zeichenkette durch eine andere Zeichenkette ersetzt.

Syntax: **Als Funktion:** v$ = MID$(x$,n[,m])

Als Anweisung: MID$(v$,n[,m]) = y$

Bemerkungen:

Fuer die Funktion (v$ = MID$...)

x$ ist ein Zeichenkettenausdruck.

n ist ein ganzzahliger Ausdruck im Bereich 1 bis 255.

m ist ein ganzzahliger Ausdruck im Bereich 0 bis 255.

Die Funktion uebergibt eine Zeichenkette der Laenge m Zeichen aus x$, beginnend mit dem n. Zeichen. Wird m weggelassen oder stehen weniger als m Zeichen rechts des n. Zeichens, werden alle rechtsbuendigen Zeichen, beginnend mit dem n. Zeichen uebergeben. Ist m gleich Null (0) oder ist n groesser als **LEN(x$)**, uebergibt **MID$** eine leere Zeichenkette.

Siehe auch Funktionen **LEFT$** und **RIGHT$**.

Fuer die Anweisung (MID$...=y$):

v$ ist eine Zeichenkettenvariable oder ein Bereichselement, dessen Zeichen ersetzt werden.

n ist ein ganzzahliger Ausdruck im Bereich 1 bis 255.

m ist ein ganzzahliger Ausdruck im Bereich 0 bis 255.

y$ ist ein Zeichenkettenausdruck.

Die Zeichen in v$, beginnend mit Position n werden durch die Zeichen in y$ ersetzt. Die wahlfreie Auswahl m bezieht sich auf die Anzahl der Zeichen von y$, die fuer das Ersetzen genutzt werden. Wird m weggelassen, wird der ganze Inhalt von y$ benutzt.
Jedoch wird unabhaengig davon, ob m weggelassen oder benutzt wird, die Laenge von v$ nicht veraendert. Ist z.B. v$ vier Zeichen lang, enthaelt v$ nach dem Ersetzen nur die ersten vier Zeichen von y$.

Hinweis:

Befinden sich n oder m nicht im angegebenen Bereich, erhaelt man den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Beispiel:

Im ersten Beispiel wird mit der Funktion **MID$** der mittlere Teil der Zeichenkette B$ ausgewaehlt.

```
10 A$="GUTEN "
20 B$="MORGEN TAG ABEND"
30 PRINT A$; MID$(B$,8,3)
RUN
GUTEN TAG
Ok
```

Im naechsten Beispiel werden mit Hilfe der Anweisung **MID$** Zeichen in der Zeichenkette A$ ersetzt.

```
10 A$="BERECHNEN SIE DIE FLAECHE EINES KREISES"
20 MID$(A$,15)="DEN UMFANG "
30 PRINT A$
RUN
BERECHNEN SIE DEN UMFANG  EINES KREISES
```

=_SPACE$

Uebergibt eine Zeichenkette, die aus n Leerstellen besteht.

Syntax: v$=SPACE$(n)

Bemerkungen:

n muss sich im Bereich 0 bis 255 befinden.
Siehe auch Funktion SPC.

Beispiel:

```
Ok
10 FOR I=1 TO 4
20 A$=SPACE$(I)
30 PRINT A$;I
40 NEXT I
RUN
 1
  2
   3
    4
Ok
```

In diesem Beispiel wird die Funktion SPACE$ benutzt, um jede Zahl I in einer Zeile mit vorangestellten I Leerstellen auszugeben. Eine zusaetzliche Leerstelle wird eingefuegt, weil BASIC vor jede positive Zahl eine Leerstelle setzt.

=_STRING$

Uebergibt eine Zeichenkette der Laenge n, deren Zeichen alle aus dem ASCII-Code m oder dem ersten Zeichen von x$ bestehen.

Syntax: v$ = STRING$(n,m)
v$ = STRING$(n,x$)

Bemerkungen:

n,m liegen im Bereich 0 bis 255.
x$ ist ein Zeichenkettenausdruck.

Beispiel:

```
10 X$=STRING$(10,45)
20 PRINT X$ "BILANZ" X$
RUN
----------BILANZ----------
Ok
```

Im ersten Beispiel wird der ASCII-Wert von 45 wiederholt und eine Zeichenkette aus Bindestrichen ausgegeben.

```
10 X$="ABCD"
20 Y$=STRING$(10,X$)
30 PRINT Y$
RUN
AAAAAAAAAA
Ok
```

Im zweiten Beispiel wird das erste Zeichen der Zeichenkette "ABCD" wiederholt.

5.2.2. Zeichenkettenfunktionen fuer die Ein-/Ausgabe

- INKEY$

Liest ein Zeichen von der Tastatur.

Syntax: v$ = INKEY$

Bemerkungen:

INKEY$ liest nur ein Zeichen, auch wenn sich mehrere Zeichen im Tastaturpuffer befinden. Der uebergebene Wert ist eine Zeichenkette aus Null, einem oder zwei Zeichen.

- Eine Null-Zeichenkette (Laenge Null) zeigt an, dass sich im Puffer kein Zeichen befindet. Es wurde nichts eingetastet.

- Eine Zeichenkette aus einem Zeichen enthaelt ein von der Tastatur gelesenes Zeichen.

- Eine Zeichenkette aus zwei Zeichen zeigt einen speziellen erweiterten Code an. Das erste Zeichen ist hexadezimal 00. Eine komplette Liste dieser Codes enthaelt Anhang D "ASCII-Zeichencodes".

Das Ergebnis von INKEY$ muss einer Zeichenkettenvariablen zugeordnet werden, bevor das Zeichen in einer BASIC-Anweisung oder -Funktion benutzt wird.

Wird INKEY$ benutzt, wird auf dem Bidschirm kein Zeichen angezeigt, und alle Zeichen werden an das Programm uebergeben. Ausnahmen (Tastatur):

- <CTRL-PAUSE> bricht die Programmausfuehrung ab
- <ALT-CTRL-DEL> setzt das System zurueck
- <CTRL-PRTSC> druckt den Inhalt des Bildschirms

Wird die Eingabetaste als Antwort auf INKEY$ gedrueckt, wird das Zeichen fuer Wagenruecklauf an das Programm uebergeben.

Beispiel:
Der folgende Programmabschnitt haelt das Programm an, bis irgendeine Taste der Tastatur gedrueckt wird.

```
1130 A$=INKEY$: IF A$="" THEN 1130
1140 ...
```

Im naechsten Beispiel wird ein uebergebener Zwei-Zeichen-Code getestet.

```
100 KB$=INKEY$
110 IF LEN(KB$)=2 THEN KB$=RIGHT$(KB$,1)
```

INPUT$

Uebergibt eine Zeichenkette von n Zeichen, die von der Tastatur oder von einer Datei mit der Nummer **Dateinummer** gelesen wurde.

Syntax: v$ = INPUT$(n[,[#]Dateinummer])

Bemerkungen:

n	Ist die Anzahl der Zeichen, die von der Datei gelesen werden oder ueber die Tastatur eingegeben werden sollen.
Dateinummer	Dateinummer, die in der Anweisung **OPEN** benutzt wird. Wird **Dateinummer** weggelassen, wird von der Tastatur gelesen.

Wird die Tastatur fuer die Eingabe benutzt, wird kein Zeichen auf dem Bildschirm angezeigt. n hat hier den Wert zwischen 1 und 255. Alle Zeichen (einschliesslich der Steuerzeichen) werden an das Programm uebergeben, ausser <CTRL-PAUSE>, das benutzt wird, um die Ausfuehrung der Funktion **INPUT$** zu unterbrechen. Wird von der Tastatur auf die Funktion **INPUT$** geantwortet, ist es nicht noetig, die Taste <ENTER> zu betaetigen.

Mit der Funktion **INPUT$** koennen Zeichen ueber Tastatur eingelesen werden, die fuer die Programm-Korrektur wichtig sind, wie z.B. Ruecksetzen (ASCII-Code 8). Sollen diese Sonderzeichen gelesen werden, muss **INPUT$** oder **INKEY$** benutzt werden (nicht **INPUT** oder **LINE INPUT**).

Fuer Datenfernverarbeitungsdateien ist es besser, die Funktion **INPUT$** zu benutzen, als **INPUT#** oder **LINE INPUT#**, da in der Datenfernverarbeitung alle ASCII-Zeichen von Bedeutung sein koennen.
Siehe Kapitel 7 "Datenfernverarbeitung".

Beispiel:

Das folgende Programm listet den Inhalt einer sequentiellen Datei in hexadezimaler Darstellung an:

```
10 OPEN "DATA" FOR INPUT AS #1
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Im naechsten Beispiel wird ein einzelnes Zeichen als Antwort auf eine Frage von der Tastatur gelesen:

```
100 PRINT "EINGABE F FUER FORTFAHREN, S FUER STOPP"
110 x$=INPUT$(1)
120 IF x$="F" THEN 600
130 IF x$="S" THEN 800 ELSE 150
```

5.3. Dienstleistungsfunktionen

TIMER

Ermittelt einen Wert einfacher Genauigkeit, der die Sekunden darstellt, die seit Mitternacht oder einem Warmstart des Systems vergangen sind.

Syntax: v = TIMER

Bemerkungen:

Teile von Sekunden werden zum naechstmoeglichen Grad berechnet. TIMER kann nur gelesen werden.

Beispiel:

```
10 TIME$ = "23:59:59"
20 FOR I = 1 TO 20
30 PRINT "TIME$ = ";TIME$;"TIMER =";TIMER
40 NEXT
RUN
TIME$ = 23:59:59          TIMER = 86399.06
TIME$ = 23:59:59          TIMER = 86399.11
TIME$ = 23:59:59          TIMER = 86399.22
TIME$ = 23:59:59          TIMER = 86399.33
TIME$ = 23:59:59          TIMER = 86399.44
TIME$ = 24:00:00          TIMER = 86399.55
TIME$ = 24:00:00          TIMER = 86399.66
TIME$ = 24:00:00          TIMER = 86399.77
TIME$ = 24:00:00          TIMER = 86399.83
TIME$ = 24:00:00          TIMER = 86399.94
TIME$ = 00:00:00          TIMER = .05
TIME$ = 00:00:00          TIMER = .16
TIME$ = 00:00:00          TIMER = .27
TIME$ = 00:00:00          TIMER = .32
TIME$ = 00:00:00          TIMER = .43
TIME$ = 00:00:00          TIMER = .54
TIME$ = 00:00:01          TIMER = .6
TIME$ = 00:00:01          TIMER = .71
TIME$ = 00:00:01          TIMER = .82
TIME$ = 00:00:01          TIMER = .9299999
Ok
```

=_RND

Uebergibt eine Zufallszahl zwischen 0 und 1.

Syntax: v = RND[(x)]

Bemerkungen:

x ist ein numerischer Ausdruck, der den uebergebenen Wert wie unten beschrieben beeinflüsst.

Bei jeder Programmausfuehrung wird die gleiche Folge von Zufallszahlen generiert, wenn der Anfangswert fuer den Zufallszahlengenerator nicht neu gesetzt wird. Dies kann sehr leicht mit Hilfe der Anweisung **RANDOMIZE** geschehen (siehe **"Anweisung RANDOMIZE"**).

Der Anfangswert des Zufallszahlengenerators kann auch neu gesetzt werden, wenn die Funktion **RND** mit Hilfe von x aufgerufen wird, wobei x negativ sein muss. Damit wird immer eine bestimmte Folge fuer ein gegebenes x erzeugt. Auf diese Folge hat **RANDOMIZE** keinen Einfluss.

Soll also fuer jede Programmausfuehrung eine andere Folge generiert werden, muss dazu jedesmal fuer x ein neuer Wert genommen werden.

Ist x positiv oder nicht eingefuegt, erzeugt **RND(x)** die naechste Zufallszahl in Folge.

RND(0) wiederholt die letzte generierte Zahl.

Mit Hilfe der Formel:

INT(RND * (n+1)) erhaelt man Zufallszahlen im Bereich 0 bis n.

Beispiel:

```
10 FOR K=1 TO 4
20 PRINT INT(RND * (100+2));
30 NEXT K
RUN
 12  66  88  74
Ok
```

=_FRE

Uebergibt die Anzahl der Bytes im Hauptspeicher, die nicht von BASIC benutzt werden. Die Zahl beinhaltet nicht die Groesse des reservierten Teils fuer den Arbeitsbereich des Interpreters (normalerweise 2,5K bis 4K Bytes).

Syntax: v = FRE(x)
v = FRE(x$)

Bemerkungen:

x und x% sind Scheinargumente.

Da die Zeichenketten in **BASIC** eine variable Laenge haben koennen (nach jeder Zuweisung zu einer Zeichenkettenvariablen kann sich ihre Laenge aendern), werden Zeichenketten dynamisch behandelt. Deshalb kann sich der Bereich fuer Zeichenketten aus mehreren Teilen zusammensetzen, die durch Zwischenraeume getrennt sind.

Wird **FRE** mit einer Zeichenkette aufgerufen, sammelt BASIC alle Daten, die noch benoetigt werden, und loescht alle unbenutzten Felder des Hauptspeichers, die einmal fuer Zeichenketten benutzt wurden. Die Daten werden alle aneinandergehaengt, so dass das Programm seine Ausfuehrung fortsetzen kann, bis der Platz im Hauptspeicher wirklich nicht mehr ausreicht.

Dieses Saeubern des Hauptspeichers geschieht mit BASIC automatisch, wenn der benutzte Arbeitsbereich nicht mehr ausreicht. Wird dies durch **FRE("")** periodisch vorgenommen, dauert diese Saeuberung nicht solange und ergibt nur kleinere Verzoegerungen.

CLEAR,n setzt die maximale Anzahl von Bytes fuer einen BASIC-Arbeitsbereich. **FRE** uebergibt die Anzahl der freien Plaetze in diesem BASIC-Arbeitsbereich. Steht nichts im Arbeitsbereich, ist der Wert, der von **FRE** uebergeben wird, zwischen 2,5K und 4K Bytes (die Groesse des reservierten Arbeitsbereichs fuer den Interpretierer) kleiner als die Anzahl der Bytes, die von **CLEAR** gesetzt wurde.

Beispiel:

```
Ok
PRINT FRE(1)
 62064
Ok
```

5.4. Funktionen zur Drucker-/Bildschirmsteuerung

- TAB

Setzt den Kursor auf die angegebene Position n.

Syntax: PRINT TAB(n)

Bemerkungen:

n muss im Bereich 1 bis 255 liegen.

Befindet sich die laufende Druckposition schon hinter der Position **n**, geht **TAB** zur Position **n** der naechsten Zeile. Stelle 1 ist die am weitesten links liegende Position, die am weitesten rechts liegende Position ist durch **WIDTH** definiert.

TAB kann nur mit den Anweisungen **PRINT**, **LPRINT** und **PRINT #** benutzt werden.

Steht die Funktion **TAB** am Ende einer Datenliste, fuehrt BASIC keine Zeilenschaltung aus, da die Funktion **TAB** automatisch ein Semikolon (;) anfuegt.

Beispiel:

Mit Hilfe von **TAB** wird im naechsten Beispiel die Information auf dem Bildschirm in Spalten aufgeteilt.

```
10 PRINT "NAME" TAB(30) "SUMME": PRINT
20 READ A$,B$
30 PRINT A$ TAB(30) B$
40 DATA "C. WERNER","100,00 M"
Ok
RUN
NAME                           SUMME

C. WERNER                      100,00 M
```

SPC

Uebergeht n Leerstellen in einer Anweisung **PRINT**.

Syntax: **PRINT SPC(n)**

Bemerkungen:

n muss sich im Bereich 0 bis 255 befinden

Ist n groesser als die durch WIDTH definierte Breite fuer die Einheit, wird die Anzahl anzugebender Leerzeichen nach der Formel n MOD WIDTH errechnet. Z.B. bei einer gesetzten Breite von 40 Z. gibt die Funktion SCP(134) 14 Leerzeichen aus (134 MOD 40). **SPC** kann nur mit den Anweisungen **PRINT**, **LPRINT** und **PRINT #** benutzt werden.

Befindet sich die Funktion **SPC** am Ende einer Datenliste, fuehrt BASIC keine Zeilenschaltung aus, da die Funktion **SPC** immer ein Semikolon (;) anfuegt.

Siehe auch Funktion **SPACE$**.

Beispiel:

```
Ok
PRINT "HIER" SPC(15) "DORT"
HIER               DORT
Ok
```

Dieses Beispiel druckt HIER und DORT getrennt durch 15 Leerstellen.

=_POS

siehe Kapitel 8

=_LPOS

Uebergibt die laufende Position des Druckkopfs im Druckpuffer fuer **LPT1:**.

Syntax: v = LPOS(n)

Bemerkungen:

n Mit **n** wird der zu testende Drucker festgelegt.
Ø oder 1 LPT1:
2 LPT2:
3 LPT3:

Mit der Funktion **LPOS** wird nicht notwendigerweise die physische Position des Druckkopfs auf dem Drucker angezeigt.

Beispiel:

Im folgenden Beispiel wird - falls die Zeilenlaenge mehr als 60 Zeichen betraegt - ein Zeichen fuer Wagenruecklauf zum Drucker gesendet, so dass ein Vorschub zur naechsten Zeile ausgefuehrt wird.

```
100 IF LPOS(Ø)>6Ø THEN LPRINT CHR$(13)
```

6. Anweisungen und Funktionen zur Dateiarbeit

6.1. Ueberblick

Eine Datei ist eine strukturierte Datenmenge, die z. B. auf dem externen Datentraeger Diskette gespeichert sein kann. Um auf die Datensaetze einer Datei zugreifen zu koennen, muss sie eroeffnet werden. Dann kann die Datei fuer die Eingabe und/oder Ausgabe benutzt werden.
BASIC unterstuetzt das Konzept von Ein-/Ausgabedateien auf allgemeinen Einheiten, d. h. jede Art von Ein-/Ausgabe wird behandelt wie Ein-/Ausgabe auf eine Datei, ob es sich nun tatsaechlich um die Benutzung einer Diskettendatei handelt oder um die Verbindung mit einem anderen Computer.

Dateinummer:

BASIC fuehrt Ein-/Ausgabeoperationen mit Hilfe einer Dateinummer aus. Diese Nummer wird einer Datei oder Einheit zugeordnet, wenn sie mit der Anweisung OPEN eroeffnet wird. Die Datei kann irgendeine Nummer, Variable oder ein Ausdruck zwischen 1 und n sein, wobei n die maximale Anzahl der gleichzeitig eroeffneten Dateien darstellt. Standardmaessig ist n=3. Beim Laden des BASIC-Systems kann n mit Hilfe des Schalters /F: veraendert werden. (siehe Kapitel 1.1).

6.1.1. Namensgebung fuer Dateien

Die physische Datei wird durch ihre **Dateiangabe** beschrieben.

Die **Dateiangabe** ist eine Zeichenkette der folgenden Form:

[Einheit:][Pfad] Dateiname

Der **Einheitenname** sagt BASIC, welche Eingabe-/Ausgabeeinheit benutzt wird. Der **Pfad** gibt BASIC das Verzeichnis an, in welchem sich die gewuenschte Datei befindet. Der **Dateiname** sagt BASIC, welche Datei auf einer bestimmten Einheit gesucht werden soll. Die Angabe von Einheitenname und Pfad ist wahlfrei.

Hinweis:

Die Dateispezifikation fuer Datenfernverarbeitungseinheiten ist etwas anders. Der Dateiname wird durch eine Liste von Auswahlen ersetzt, die z.B. den Parameter Uebertragungsgeschwindigkeit spezifizieren. Siehe "Anweisung OPEN COM..." in Kapitel 7.

Wird eine Zeichenkettenkonstante fuer die Dateiangabe verwendet, muss sie in Anfuehrungszeichen eingeschlossen werden.

Einheitenname:

Der Einheitenname besteht aus 1...4 Zeichen, gefolgt von einem Doppelpunkt (:). Im folgenden wird eine Liste aller Einheitennamen gezeigt, die aussagt, wozu die Einheit benutzt werden kann (Eingabe oder Ausgabe).

Tabelle der Einheitennamen:

KYBD: Eingabetastatur. Nur Eingabe.

SCRN: Bildschirm. Nur Ausgabe.

LPT1: Erster Drucker. Ausgabe oder Direktzugriff.

LPT2: Zweiter Drucker. Ausgabe oder Direktzugriff.

LPT3: Dritter Drucker. Ausgabe oder Direktzugriff.

DATENFERNVERARBEITUNGSEINHEITEN

COM1: Erster Anschluss fuer asynchrone Uebertragung. Eingabe und Ausgabe.

COM2: Zweiter Anschluss fuer asynchrone Uebertragung. Ein- und Ausgabe.

SPEICHEREINHEITEN

A: Erste Disketteneinheit. Eingabe und Ausgabe.

B: Zweite Disketteneinheit. Eingabe und Ausgabe.

C:... Weitere Disketteneinheiten. Eingabe und Ausgabe.

Siehe unter "Suchfolge fuer Anschluesse" in Anhang H, "Technische Informationen". Dort stehen Informationen darueber, welche Anschluesse sich auf die Drucker- und Datenfernverarbeitungsnamen beziehen.

PFAD

Es ist die Angabe eines **Pfades** moeglich. Ein Pfad besteht aus einer Liste von Verzeichnissnamen, die durch umgekehrte Schraegstriche getrennt sind.

Pfade koennen mit den folgenden Befehlen angegeben werden:

BLOAD	KILL	OPEN
BSAVE	LOAD	RMDIR
CHAIN	MERGE	RUN
CHDIR	MKDIR	SAVE
FILES	NAME	

Hinweis:

- Ein Pfad darf nicht mehr als 63 Zeichen enthalten.
- Wird ein Einheitenname an einer anderen Stelle angegeben als vor einem Pfad, erhaelt man die Fehlernachricht "Bad file name" (Falscher Dateiname).
- Wird fuer den Pfad eine Zeichenkettenkonstante benutzt, muss sie in Anfuehrungszeichen eingeschlossen sein.

```
"B:\VERKAUF\MAI\JOURNAL"
```

Ist die Datei nicht im laufenden Verzeichnis enthalten, muss man BASIC einen Pfad von Verzeichnisnamen anbieten, die zu dem laufenden Verzeichnis fuehren.

Dateiname:

Bei Diskettendateien muss sich der Name an die DOS-Konventionen halten:

- Der Name darf aus zwei Teilen bestehen, die durch einen Punkt getrennt sind:

  Name.Dateityp

- Der **Name** kann ein bis acht Zeichen lang sein. Der **Dateityp** darf nicht laenger als drei Zeichen sein.

- Ist der **Dateityp** laenger als drei Zeichen, werden die Extrazeichen unterdrueckt. Ist der **Name** laenger als acht Zeichen und der **Dateityp** ist nicht eingeschlossen, fuegt BASIC einen Punkt nach dem achten Zeichen ein und nimmt die Extrazeichen (bis zu drei) als **Dateityp**. Ist der **Name** laenger als acht Zeichen und ein **Dateityp** ist eingeschlossen, wird ein Fehler angezeigt.

- Nur die folgenden Zeichen sind fuer **Name** und **Dateityp** erlaubt:

  ```
  A bis Z
  0 bis 9
  ( ) { } # $ % ^ & ! @ _ - ' ` \ /
  ```

  Beispiele fuer Dateinamen sind:

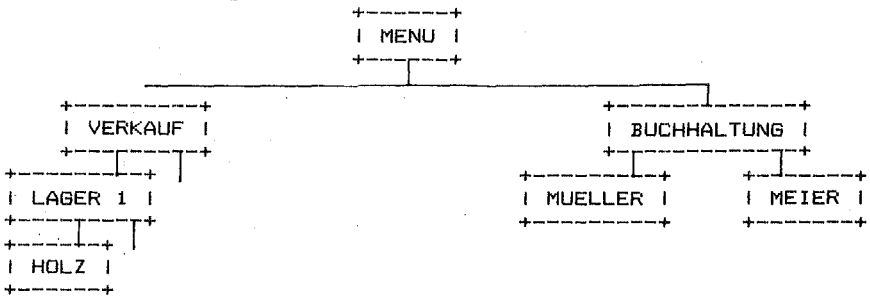
  ```
  PROGRAMM.BAS
  TEST1.ASC
  KUNDEN
  ```

6.1.2. Verzeichnisarten

Auf jeder Diskette wird ein Inhaltsverzeichnis (Directory) erstellt, wenn sie formatiert wird. Dieses Verzeichnis ist das **Basisverzeichnis.** Die hoechste Dateianzahl im Basisverzeichnis einer Diskette haengt von der Art der benutzten Diskette ab. Zusaetzlich zu den Dateinamen enthaelt das Basisverzeichnis auch die Namen der anderen Verzeichnisse, genannt **Unterverzeichnisse.** Im Unterschied zum Basisverzeichnis sind die Unterverzeichnisse in Wirklichkeit Dateien und koennen jede Anzahl zusaetzlicher Dateien und Unterverzeichnisse enthalten, limitiert nur durch den verfuegbaren Platz auf der Diskette.
Die Unterverzeichnisnamen haben dasselbe Format wie die Dateinamen. Alle fuer die Dateinamen gueltigen Zeichen gelten auch fuer einen Verzeichnisnamen. Jedes Verzeichnis kann auch Datei- und Verzeichnisnamen enthalten, die in anderen Verzeichnissen vorkommen.

Laufendes Verzeichnis

So wie sich BASIC an ein Standardlaufwerk erinnert, erinnert es sich auch an ein Standardverzeichnis fuer jedes Laufwerk im System. Dies wird das **laufende Verzeichnis** genannt und ist das Verzeichnis, in dem BASIC einen angegebenen Dateinamen sucht, wenn kein anderes Verzeichnis angegeben ist. Mit dem Befehl **CHDIR** kann man das laufende Verzeichnis aendern (siehe Befehl CHDIR im Kapitel 10).
Ist ein Dateiname eingefuegt, muss auch er vom letzten Verzeichnisnamen durch einen umgekehrten Schraegstrich getrennt werden. Beginnt ein Pfad mit einem umgekehrten Schraegstrich, beginnt BASIC mit dem Suchen beim Basisverzeichnis; sonst beginnt das Suchen mit dem laufenden Verzeichnis.
Fruehere BASIC-Versionen arbeiteten mit einer einfachen Verzeichnisstruktur, die fuer die Verwaltung der Diskettendateien ausreichte. Jetzt ist es moeglich, Disketten besser zu organisieren, indem man Gruppen von zusammengehoerenden Dateien eigene Verzeichnisse zuordnet, alle auf der gleichen Diskette. Diese Verzeichnisse werden **baumstrukturierte Verzeichnisse** genannt. Sie gehen von einem Basisverzeichnis aus und verzweigen in Unterverzeichnisse.
Die Firma XYZ hat zum Beispiel zwei Abteilungen, den Verkauf und die Buchhaltung. Die Organisation der Dateikategorien kann man sich wie folgt vorstellen:



6.1.3. Diskettendateien

Mit einem BASIC-Programm koennen zwei Arten von Diskettendateien erstellt und verarbeitet werden:

sequentielle Dateien
Dateien fuer Direktzugriff

Sequentielle Dateien:

Sequentielle Dateien sind leichter zu schaffen als Dateien mit Direktzugriff, sind aber eingeschraenkt in ihrer Flexibilitaet und Verarbeitungsgeschwindigkeit.
Daten, die in eine sequentielle Datei geschrieben werden, werden in der Reihenfolge der Uebergabe der einzelnen Daten sequentiell gespeichert. Alle Daten werden auf die gleiche Weise gelesen - von der ersten Angabe in der Datei bis zur letzten.

Dateien fuer Direktzugriff:

Fuer das Erstellen und Verarbeiten von Direktzugriffs-Dateien sind mehr Programmschritte erforderlich als fuer sequentielle Dateien, aber die Benutzung von Direktzugriffs-Dateien bringt Vorteile.
Zum Beispiel benoetigen Direktzugriffs-Dateien weniger Platz auf der Diskette als sequentielle Dateien, da die Daten im Binaerformat gespeichert werden, waehrend numerische Werte in sequentiellen Dateien im ASCII-Format gespeichert werden.
Ausserdem kann jeder Datensatz direkt ueber seine Satznummer angesprochen werden.

6.2. Eroeffnen und Schliessen von Diskettendateien

6.2.1. OPEN

Erlaubt die Ein-/Ausgabe zu und von einer Datei oder Einheit.

Syntax: Erste Form:

**OPEN Dateiangabe [FOR Modus] AS [#]
Dateinummer [LEN=Satzlaenge]**

Alternative Form:

**OPEN Modus2, [#] Dateinummer, Dateiangabe
[,Satzlaenge]**

Bemerkungen:

Modus
ist in der ersten Form eines der folgenden Woerter:

OUTPUT Spezifiziert sequentielle Ausgabe
INPUT Spezifiziert sequentielle Eingabe
APPEND Spezifiziert Hinzufuegen zu einer sequentiellen Datei, wobei die Datei bei der Eroeffnung auf das Datenende positioniert wird.

Man muss beachten, dass **Modus** eine Zeichenkettenkonstante sein muss, die nicht innerhalb von Anfuehrungszeichen (") stehen darf. Wird Modus weggelassen, wird Direktzugriff angenommen.

Modus 2
ist in der alternativen Form ein Zeichenkettenausdruck, dessen erstes Zeichen eines der folgenden sein muss.

O Spezifiziert sequentielle Ausgabe
I Spezifiziert sequentielle Eingabe
R Spezifiziert Ein-/Ausgabe fuer Direktzugriff

Fuer beide Formate gilt:

Dateinummer
Ganzzahliger Ausdruck, dessen Wert zwischen 1 und der maximalen Anzahl der erlaubten Dateien liegt.
Standardanzahl ist 3, kann aber mit der Auswahl /F: beim Laden von BASIC geaendert werden.

Dateiangabe
Zeichenkettenausdruck fuer die Dateispezifikation, wie unter 6.1.1. beschrieben.

Satzlaenge
Ganzzahliger Ausdruck, der - falls eingefuegt - die Satzlaenge fuer Dateien mit Direktzugriff setzt. Er kann sich im Bereich 1 bis 32767 befinden. Die Standardsatzlaenge ist 128 Bytes. Sie darf den Wert, der durch die Auswahl /S: beim Laden von BASIC gesetzt wurde, nicht ueberschreiten.

OPEN ordnet der Datei oder Einheit fuer die Ein-/Ausgabe einen Puffer zu und bestimmt den Zugriffsmodus, der fuer den Puffer benutzt wird.

Dateinummer ist die Nummer, die der Datei zugeordnet ist, solange sie eroeffnet ist, und die von anderen Ein-/Ausgabeanweisungen benutzt wird, um die Datei oder Einheit anzusprechen.

OPEN muss ausgefuehrt werden, bevor irgendeine Ein-/Ausgabe zu einer Einheit oder Datei mit Hilfe einer der folgenden Anweisungen oder irgendeiner Anweisung oder Funktion, die eine Dateinummer benoetigt, erfolgen kann:

PRINT #	**INPUT #**
PRINT # USING	**LINE INPUT #**
WRITE #	**GET**
INPUT$	**PUT**
IOCTL #	

GET und **PUT** sind fuer Dateien mit Direktzugriff gueltig (oder Datenfernverarbeitungsdateien - siehe Kapitel 7). Eine Diskettendatei kann entweder direkt oder sequentiell sein, und ein Drucker kann entweder in direktem oder sequentiellem Modus eroeffnet werden. Alle anderen Einheiten jedoch koennen nur fuer sequentielle Operationen eroeffnet werden. Siehe Anweisung OPEN "COM... in Kapitel 7.

BASIC fuegt normalerweise nach jedem CHR$(13), das zum Drucker gesendet wird, einen Zeilenvorschub an. Wird jedoch der Drukker (LPT1:, LPT2: oder LPT3:) als Datei mit direktem Zugriff mit einer Breite von 255 eroeffnet, wird der Zeilenvorschub unterdrueckt.

APPEND ist nur fuer Diskettendateien gueltig. Der Dateizeiger wird auf das Ende der Datei gesetzt und die Satznummer auf den letzten Satz der Datei. **PRINT #** oder **WRITE #** fuegen Saetze an die Datei an.

Hinweis:

Es ist jederzeit moeglich, eine bestimmte Datei fuer mehr als eine Dateinummer zu eroeffnen. Dies erlaubt, fuer verschiedene Zwecke in verschiedenen Modi zu arbeiten, d. h. dass mit verschiedenen Dateinummern eine Datei in verschiedenen Zugriffsarten verarbeitet werden kann. Jeder Dateinummer ist ein eigener Puffer zugeordnet. Deshalb muss man aufpassen, wenn man mit einer Dateinummer schreibt und mit einer anderen Dateinummer liest.
Jedoch kann eine Datei nicht fuer sequentielle Ausgabe oder Hinzufuegen (APPEND) eroeffnet werden, wenn die Datei schon eroeffnet ist.

Fehlt der Einheitenname, wird die Standardeinheit fuer DCP angenommen.

Soll eine Eingabedatei, die nicht existiert, eroeffnet werden, erhaelt man den Fehler "File not found" (Datei nicht gefunden). Wird eine Datei, die nicht existiert, fuer Ausgabe, Hinzufuegen oder Direktzugriff eroeffnet, so wird diese Datei angelegt.

Jeder angegebene Wert, der sich ausserhalb der angegebenen Bereiche befindet, ergibt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Die Datei wird nicht eroeffnet.

Informationen ueber die Eroeffnung von Datenfernverarbeitungsdateien mit der Anweisung OPEN "COM... (siehe Kapitel 7, Punkt 7.1.).

Beispiel:

```
10 OPEN "DATA" FOR OUTPUT AS #1
```

oder

```
10 OPEN "O",#1,"DATA"
```

Jede dieser Anweisungen eroeffnet die Datei "DATA" fuer sequentielle Ausgabe auf der Standardeinheit Diskette. Dabei muss beachtet werden, dass bei Neueroeffnung alle Daten, die sich in der Datei befinden, ueberschrieben werden. Sollen die Daten, die bereits vorhanden sind, nicht zerstoert werden, muss mit APPEND eroeffnet werden.

```
20 OPEN "B:DATFILE" AS 1 LEN=256
```

oder

```
20 OPEN "R",1,"B:DATFILE",256
```

Jede der zwei vorhergehenden Anweisungen eroeffnet die Datei "DATFILE" auf der Diskette in Laufwerk B fuer direkte Ein- und Ausgabe. Die Satzlaenge ist 256.

```
10 FILE$="A:DATA.ASC"
20 OPEN FILE$ FOR APPEND AS 3
```

In diesem Beispiel wird die Datei "DATA.ASC" auf der Diskette in Laufwerk A eroeffnet und der Dateizeiger an das Ende der Datei gebracht, so dass jede Ausgabe zu dieser Datei an das Ende der Datei angefuegt wird.

In Zeile 10 des naechsten Beispiels wird der Drucker fuer direkten Modus eroeffnet. Da die Standardbreite 80 ist, enden die mit den Zeilen 20 und 30 gedruckten Zeilen mit einem Wagenruecklauf/Zeilenvorschub. In Zeile 40 wird die Druckbreite auf 255 geaendert. Dadurch wird der Zeilenvorschub nach dem Wagenruecklauf unterdrueckt.
Deshalb endet die mit Zeile 50 gedruckte Zeile nur mit einem Wagenruecklauf und nicht mit einem Zeilenvorschub. Dadurch ueberdruckt die durch Zeile 70 gedruckte Zeile den Text "Diese Zeile wird unterstrichen" und unterstreicht die Druckausgabezeile. In Zeile 60 wird die Druckbreite wieder auf 80 geaendert, so dass das Unterstreichen und die folgenden Zeilen wieder mit einem Zeilenvorschub enden.

```
Ok
10 OPEN "LPT1:" AS #1
20 PRINT #1,"DRUCKBREITE 80"
30 PRINT #1,"Jetzt Druckbreite 255"
40 WIDTH #1,255
50 PRINT #1,"Diese Zeile wird unterstrichen"
60 WIDTH #1,80
70 PRINT #1,STRING#(28,"_")
80 PRINT #1,"Druckbreite 80 mit CR/LF"
RUN
Ok
Druckbreite 80
Jetzt Druckbreite 255
Diese_Zeile_wird_unterstrichen
Druckbreite 80 mit CR/LF
Ok
```

Die folgenden Beispiele zeigen die Verwendung von **Pfad** fuer Dateiangabe.

```
10 OPEN "KUNDEN\LAGER1\DATEN" FOR OUTPUT AS #1
```

oder

```
10 OPEN "0",#1,"KUNDEN\LAGER1\DATEN"
```

Jede dieser Anweisungen eroeffnet die Datei "DATEN" fuer sequentielle Ausgabe auf der Standardeinheit im Verzeichnis "LAGER1".

```
20 OPEN "B:LAGER1\ARTIKEL" AS 1 LEN=256
```

oder

```
20 OPEN "R",1,"B:LAGER1\ARTIKEL",256
```

Jede dieser Anweisungen eroeffnet die Datei "ARTIKEL" im Verzeichnis "LAGER1" auf der Diskette im Laufwerk B fuer Ein-/Ausgabe im Direktzugriff. Die Satzlaenge ist 256 Byte.

6.2.2. CLOSE

Schliesst die Ein-/Ausgabe fuer eine Einheit oder Datei ab.

Syntax: **CLOSE [[#] Dateinummer [,[#] Dateinummer]...]**

Bemerkungen:

Dateinummer: In der Anweisung **OPEN** benutzte Nummer.

Die Beziehung zwischen einer bestimmten Datei oder Einheit und ihrer Dateinummer wird aufgehoben, sobald **CLOSE** ausgefuehrt wird. Nachfolgende Ein-/Ausgabeoperationen, die diese Dateinummer verwenden, sind nicht mehr gueltig. Die Datei oder Einheit kann unter derselben oder einer anderen Dateinummer erneut eroeffnet werden oder die Dateinummer kann erneut benutzt werden, um eine andere Einheit oder Datei zu eroeffnen. Durch das Abschliessen einer Datei oder einer Einheit, die fuer sequentielle Ausgabe eroeffnet wurde, wird der Inhalt des letzten Puffers an die Datei oder Einheit uebergeben.

Durch **CLOSE** ohne angegebene Dateinummern werden alle Einheiten und Dateien, die eroeffnet sind, abgeschlossen.

Durch die Ausfuehrung von **END**, **NEW**, **RESET**, **SYSTEM** oder **RUN** ohne die Angabe R werden alle eroeffneten Dateien und Einheiten automatisch abgeschlossen. **STOP** schliesst keine Datei oder Einheit ab.

Siehe Anweisung **OPEN** in diesem Kapitel fuer Informationen ueber das Eroeffnen von Dateien.

Beispiel:

```
100 CLOSE 1,#2,#3
```

Dadurch werden die Dateien und Einheiten, denen die Dateinummern 1, 2 und 3 zugeordnet sind, abgeschlossen.

```
200 CLOSE
```

Alle eroeffneten Dateien und Einheiten werden abgeschlossen.

6.3. Sequentielle Dateien

6.3.1. PRINT # und PRINT # USING

Schreibt Daten sequentiell in eine Datei.

Syntax: **PRINT #Dateinummer,[USING v$;]**
Liste der Ausdruecke

Bemerkungen:

Dateinummer Nummer, die benutzt wurde, als die Datei fuer Ausgabe eroeffnet wurde.

v$ ist ein Zeichenkettenausdruck, der aus den Formatzeichen besteht, die im Abschnitt 4, Anweisung "PRINT USING" beschrieben wurden.

Liste der Ausdruecke
Liste der numerischen und/oder Zeichenkettenausdruecke, die in die Datei geschrieben werden sollen

PRINT # komprimiert die Daten nicht in der Datei. Ein Abbild der Daten wird so in die Datei geschrieben, als ob sie mit der Anweisung **PRINT** auf dem Bildschirm angezeigt wuerden. Deshalb muessen die Daten in der Datei unbedingt mit Trennzeichen versehen werden, so dass sie als Eingabe korrekt aus der Datei gelesen werden koennen.

In der Liste der Ausdruecke muessen numerische Ausdruecke immer durch Semikolon (;) getrennt werden.

Beispiel:

```
PRINT #1,A;B;C;X;Y;Z
```

Werden Komma (,) als Trennzeichen benutzt, werden die extra Leerzeichen, die zwischen den Druckfeldern eingefuegt werden, auch in die Datei geschrieben. (sh. Anweisung "PRINT")

Zeichenkettenausdruecke muessen mit Hilfe von Semikolons in der Liste getrennt werden. Um die Zeichenkettenausdruecke richtig in der Datei zu formatieren, muessen explizite Trennzeichen in der Liste der Ausdruecke verwendet werden.

Beispiel:

```
A$="CAMERA" und B$="93604-1"
```

Mit der Anweisung

```
PRINT #1,A$;B$
```

wird CAMERA93604-1 in die Datei geschrieben. Da hier keine Trennzeichen vorhanden sind, koennte dies nicht als Ausgabe fuer zwei separate Zeichenketten benutzt werden. Um dieses Problem zu umgehen, muessen explizite Trennzeichen in die Anweisung **PRINT #** wie folgt eingefuegt werden:

```
PRINT #1,A$;",";B$
```

Das Abbild, das jetzt in die Datei geschrieben wird, ist wie folgt:

```
CAMERA,93604-1
```

Dies kann wieder in zwei Zeichenkettenvariablen zurueckgelesen werden.

Enthalten die Zeichenketten Kommata, Semikolons, signifikante fuehrende Leerstellen, Wagenruecklauf oder Zeilenvorschub, muessen sie in die Datei in eingeschlossenen Anfuehrungszeichen mit Hilfe von **CHR$(34)** geschrieben werden.

Beispiel:

A$="CAMERA,AUTOMATIC" und B$="93604-1".

Die Anweisung

```
PRINT #1,A$;B$
```

schreibt folgendes Abbild in die Datei:

```
CAMERA,AUTOMATIC 93604-1
```

und die Anweisung

```
INPUT #1,A$,B$
```

liest die Zeichenkette "CAMERA" nach A$ und "AUTOMATIC 93604-1" nach B$.

Um diese zwei Zeichenketten richtig in der Datei zu trennen, muessen in das Abbild fuer die Datei mit Hilfe von CHR$(34) doppelte Anfuehrungszeichen geschrieben werden. Die Anweisung:

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

schreibt das folgende Abbild in die Datei:

```
"CAMERA,AUTOMATIC" "93604-1"
```

und die Anweisung:

```
INPUT #1,A$,B$
```

liest "CAMERA,AUTOMATIC" nach A$ und "93604-1" nach B$.

Die Anweisung **PRINT #** kann auch in Verbindung mit der Angabe **USING** benutzt werden, um das Format in der Datei zu steuern.

Beispiel:

```
PRINT #1,USING"####.##";J;K;L
```

Der einfachste Weg, all diese Probleme zu vermeiden, ist die Benutzung der Anweisung **WRITE #** anstelle der Anweisung **PRINT #** (siehe "Anweisung WRITE #).

6.3.2. WRITE

Schreibt Daten in eine sequentielle Datei.

Syntax: **WRITE #Dateinummer, Liste der Ausdruecke**

Bemerkungen:

Dateinummer Nummer, unter der die Datei fuer Ausgabe eroeffnet wurde.

Liste der Ausdruecke
Liste von Zeichenketten und/oder numerischen Ausdruecken, die durch Kommata (,) oder Semikolons (;) getrennt sind.

Der Unterschied zwischen **WRITE #** und **PRINT #** ist der, dass **WRITE #** Kommata zwischen die Daten einfuegt und Zeichenketten in Anfuehrungszeichen (") einschliesst, wenn sie aufgezeichnet werden. Deshalb braucht der Benutzer keine Trennzeichen in die Liste einzufuegen. **WRITE #** setzt auch keine Leerstellen vor eine positive Zahl. Ein Wagenruecklauf/Zeilenvorschub wird nach der letzten Angabe der Liste geschrieben.

Beispiel:

A$="CAMERA" und B$="93604-1".

Die Anweisung: WRITE #1,A$,B$ schreibt das folgende Bild in die Datei:

"CAMERA","93604-1"

Eine nachfolgende Anweisung **INPUT #**, wie z.B.

INPUT #1,A$,B$

wuerde **"CAMERA"** der Variablen **A$** und **"93604-1"** der Variablen **B$** zuordnen.

6.3.3. INPUT

Liest Daten aus einer sequentiellen Einheit oder Datei und weist sie den Programmvariablen zu.

Syntax: **INPUT #Dateinummer, Variable[,Variable]...**

Bemerkungen:

Dateinummer Nummer, die fuer die Eingabedatei benutzt wurde.

Variable Name einer Variablen, der Daten aus der Datei zugeordnet werden. Es kann sich um eine Zeichenketten- oder numerische Variable handeln oder um ein Feldelement.

Die sequentielle Datei kann sich auf Diskette befinden, oder es kann sich um sequentielle Daten des Datenfernverarbeitungsanschlusses handeln. Die Daten koennen auch von der Tastatur kommen (KYBD:).

Der Typ der Dateidaten muss mit dem Typ der angegebenen Variablennamen uebereinstimmen. Mit **INPUT #** wird nicht wie mit **INPUT** ein Fragezeichen (?) angezeigt.

Die Daten in der Datei muessen so aussehen, als waeren sie als Antwort auf eine Anweisung **INPUT** eingegeben worden. Bei numerischen Werten werden fuehrende Leerstellen, Wagenruecklaeufe und Zeichen fuer neue Zeilen ignoriert. Das erste Zeichen, das nicht gleich einer Leerstelle, einem Wagenruecklauf oder einem Zeilenvorschub ist, wird als Anfang der Zahl angenommen. Die Zahl endet mit einer Leerstelle, einem Wagenruecklauf, einem Zeilenvorschub oder einem Komma.

Sucht sich BASIC die Daten fuer eine Zeichenkette, werden fuehrende Leerstellen, Wagenruecklaeufe und Zeilenvorschuebe auch ignoriert. Alle anderen Zeichen werden als Beginn der Zeichenkette angenommen. Ist das erste Zeichen ein Anfuehrungszeichen ("), besteht die Zeichenkette aus allen Zeichen, die zwischen dem ersten und dem zweiten Anfuehrungszeichen stehen. Das bedeutet, dass eine in Anfuehrungszeichen eingeschlossene Zeichenkette kein Anfuehrungszeichen als Zeichen enthalten darf. Ist das erste Zeichen einer Zeichenkette kein Anfuehrungszeichen ("), dann handelt es sich um eine Zeichenkette ohne Anfuehrungszeichen; das Lesen wird durch ein Komma (,), einen Wagenruecklauf, einen Zeilenvorschub oder nach 255 gelesenen Zeichen beendet. Wird bei der Eingabe eines numerischen Wertes oder einer Zeichenkette das Dateiende erreicht, wird diese Eingabe unterdrueckt.

INPUT # kann auch mit einer Datei mit Direktzugriff benutzt werden.

6.3.4. LINE INPUT

Liest eine ganze Zeile (bis zu 255 Zeichen), Trennzeichen ignorierend, von einer sequentiellen Datei in eine Zeichenkettenvariable.

Syntax: **LINE INPUT #Dateinummer,Zeichenkettenvariable**

Bemerkungen:

Dateinummer Nummer, unter der die Datei eroeffnet wurde.

Zeichenkettenvariable
Name einer Zeichenkettenvariablen oder eines Feldelementes, dem die Zeile zugeordnet werden soll.

LINE INPUT # liest alle Zeichen aus einer sequentiellen Datei bis zum Zeichen Wagenruecklauf. Dann wird die Zeichenfolge Wagenruecklauf/Zeilenvorschub uebersprungen, und mit der naechsten Anweisung **LINE INPUT #** werden die naechsten Zeichen bis zum naechsten Wagenruecklauf gelesen. (Wird die Zeichenfolge Zeilenvorschub/Wagenruecklauf gefunden, wird sie konserviert; das bedeutet, die Zeichen fuer Zeilenvorschub/Wagenruecklauf werden als Teil der Zeichenkette uebergeben.)

LINE INPUT # ist speziell dann nuetzlich, wenn jede Zeile einer Datei in Felder aufgeteilt wurde oder wenn ein BASIC-Programm, das im ASCII-Modus gespeichert wurde, als Daten von einem anderen Programm eingelesen werden soll.

LINE INPUT # kann auch fuer Dateien mit Direktzugriff benutzt werden.

Beispiel:

Im folgenden Beispiel wird **LINE INPUT** benutzt, um Informationen von der Tastatur zu lesen, die Kommas (,) oder andere Trennzeichen enthalten kann. Danach wird die Information in eine sequentielle Datei geschrieben und aus der Datei mit Hilfe von **LINE INPUT #** gelesen.

```
10 OPEN "LIST" FOR OUTPUT AS #1
20 LINE INPUT "Stadt?";C$
30 PRINT #1,C$
40 CLOSE 1
50 OPEN "LIST" FOR INPUT AS #1
60 LINE INPUT #1,C$
70 PRINT C$
80 CLOSE 1
RUN
Stadt?
```

Wird z.B. mit KARL-MARX-STADT geantwortet, macht das Programm wie folgt weiter:

```
        .
        .
        .
Stadt? KARL-MARX-STADT
KARL-MARX-STADT
Ok
```

6.4. Direktzugriffsdateien

6.4.1. FIELD

Legt Platz fuer Variablen in einem Puffer fuer Dateien mit Direktzugriff an.

Syntax: **FIELD [#]Dateinummer, Laenge AS Zeichenkettenvariable**
[,Laenge AS Zeichenkettenvariable]...

Bemerkungen:

Dateinummer Nummer, mit der die Datei eroeffnet wurde.

Laenge Numerischer Ausdruck. Gibt die Anzahl der Zeichenpositionen an, die einer **Zeichenkettenvariablen** zugeordnet werden sollen.

Zeichenkettenvariable
Zeichenkettenvariable, die fuer den Zugriff auf Dateien mit Direktzugriff benutzt wird.

Die Anweisung **FIELD** definiert Variablen, die benutzt werden, um mit Hilfe von **GET** Daten aus dem Puffer fuer Direktzugriff zu lesen oder mit **PUT** Daten in den Puffer einzugeben.

Die Anweisung:

FIELD 1,20 AS N$,10 AS ID$,40 AS ADD$

uebergibt die ersten 20 Positionen (Bytes) aus dem Puffer fuer die Datei mit Direktzugriff an die Zeichenkettenvariable N$, die naechsten 10 Positionen an ID$ und die naechsten 40 Positionen an ADD$. **FIELD** schreibt **nicht** selbst die Daten in den Puffer fuer eine Datei mit Direktzugriff. Dies geschieht durch die Anweisungen **LSET** und **RSET** (siehe "Anweisungen LSET und RSET" in diesem Kapitel).

FIELD "liest" auch keine Daten aus der Datei. Die Informationen werden mit Hilfe der Anweisung **GET** (Datei) aus der Datei in den Puffer fuer eine Datei mit Direktzugriff gelesen. Die Informationen werden aus dem Puffer gelesen, indem man die Variablen in der Anweisung **FIELD** definiert.

Die Gesamtanzahl der Bytes, die in der Anweisung **FIELD** zugeordnet werden, duerfen die Satzlaenge, die bei der Eroeffnung der Datei angegeben wurde, nicht ueberschreiten; anderenfalls erhaelt man den Fehler "Field Overflow" (Felduebertauf).

Es koennen beliebig viele Anweisungen **FIELD** fuer die gleiche Dateinummer ausgefuehrt werden, die alle zur gleichen Zeit gueltig sind. Jede neue Anweisung **FIELD** faengt wieder mit der ersten Zeichenposition im Puffer an. Das hat den Effekt, dass man mehrere Felddefinitionen fuer dieselben Daten haben kann.

Hinweis:

Man muss vorsichtig sein, wenn man **FIELD**-Variablennamen in einer Eingabe- oder Zuordnungsanweisung benutzt. Sobald einmal ein Variablenname in einer Anweisung **FIELD** definiert wurde, zeigt **FIELD** auf den richtigen Platz in dem Puffer fuer Dateien mit Direktzugriff. Wird eine nachfolgende Eingabeanweisung oder Anweisung **LET** mit diesem Variablennamen auf der linken Seite des Gleichheitszeichens ausgefuehrt, wird die Variable in dem Platz fuer Zeichenkettenvariablen angelegt und steht nicht mehr laenger im Dateipuffer.

Beispiel:

```
10 OPEN "A:KUND" AS #1
20 FIELD  1,2 AS KUNDNR$,30 AS KUNDNAME$,35 AS ADR$
30 LSET KUNDNAME$="ROBOTRON-BUCHUNGSMASCHINENWERK"
40 LSET ADR$="ANNABERGER STR. 93, KARL-MARX-STADT"
50 LSET KUNDNR$=MKI$(7850)
60 PUT 1,1
70 GET 1,1
80 KNUM%=CVI(KUNDNR$)
90 N$=KUNDNAME$
100 PRINT KNUM%,N$,ADR$
```

In diesem Beispiel wird eine Datei mit Namen KUND als Datei mit Direktzugriff eroeffnet. Die Variable KUNDNR$ wird den ersten zwei Stellen jedes Satzes zugeordnet, KUNDNAME$ den naechsten 30 Positionen und ADR$ den naechsten 35 Positionen. In den Zeilen 30 bis 50 wird die Information in den Puffer gebracht, und die Anweisung **PUT** in Zeile 60 schreibt den Puffer in die Datei. Zeile 70 liest den gleichen Satz wieder ein, und in Zeile 100 werden die drei Felder angezeigt.

6.4.2. LSET und RSET

Uebergibt Daten in einen Dateipuffer fuer Direktzugriff als Vorbereitung auf die Anweisung **PUT** (Datei).

Syntax:

```
LSET Zeichenkettenvariable = x$
RSET Zeichenkettenvariable = x$
```

Bemerkungen:

Zeichenkettenvariable
Name einer Variablen, die in der Anweisung **FIELD** definiert wurde.

x$
Zeichenkettenausdruck fuer die Information, die in das durch Zeichenkettenvariable identifizierte Feld gebracht werden soll.

Benoetigt x$ weniger Bytes als fuer die **Zeichenkettenvariable** in der Anweisung **FIELD** definiert wurden, wird durch **LSET** die Zeichenkette linksbuendig im Feld gespeichert - mit **RSET** rechtsbuendig. (Die Extrastellen werden durch Leerstellen aufgefuellt.) Ist x$ laenger als die **Zeichenkettenvariable**, werden die Zeichen rechtsbuendig abgeschnitten.

Numerische Werte muessen in Zeichenketten umgewandelt werden, bevor sie mit **LSET** oder **RSET** verarbeitet werden koennen. Siehe "Funktionen **MKI$**, **MKS$**, **MKD$**" in diesem Kapitel.

Hinweis:

LSET und **RSET** koennen auch mit einer Zeichenkettenvariablen benutzt werden, die nicht in einer Anweisung **FIELD** definiert wurde , um eine Zeichenkette in einem gegebenen Feld rechts- oder linksbuendig anzuordnen. Beispiel:

```
10 A$=SPACE$(20)
20 RSET A$=N$
```

Die Programmzeilen uebergeben die Zeichenkette N$ rechtsbuendig in ein 20-Zeichen-Feld. Damit kann die Druckausgabe formatiert werden.

Beispiel:

In diesem Beispiel wird der numerische Wert AMT in eine Zeichenkette umgewandelt und linksbuendig in dem Feld A$ zur Vorbereitung fuer eine Anweisung PUT (Datei) gespeichert:

```
10 LSET A$=MKS$(AMT)
```

6.4.3. PUT

Schreibt einen Satz aus dem Puffer fuer Direktzugriff in eine Datei mit Direktzugriff.

Syntax: **PUT [#]Dateinummer[,Nummer]**

Bemerkungen:

Dateinummer Nummer, unter der die Datei eroeffnet wurde.

Nummer Satznummer fuer einen zu schreibenden Satz im Bereich 1 bis 16 Megabyte.

Wird **Nummer** weggelassen, bekommt der Satz die naechste verfuegbare Satznummer (nach dem letzten PUT oder GET). Nicht ganzzahlige Satznummern werden abgeschnitten.

Bevor die Anweisung **PUT** ausgefuehrt wird, koennen mit **PRINT #**, **PRINT # USING**, **WRITE #**, **LSET** und **RSET** Zeichen in den Dateipuffer fuer Direktzugriff gebracht werden. Im Falle von **WRITE #** fuellt BASIC den Puffer mit Leerstellen bis zum Wagenruecklauf auf.

Jeder Versuch, ueber das Ende des Puffers hinauszuschreiben oder zu lesen, ergibt den Fehler "Field Overflow" (Felduebertlauf).

Da BASIC und DCP soviel Saetze wie moeglich in 512-Byte-Sektoren blockt, fuehrt die Anweisung PUT erst ein physisches Schreiben auf die Diskette aus, wenn der Puffer gefuellt ist.

PUT kann fuer eine Datenfernverarbeitungsdatei benutzt werden. In diesem Fall bedeutet **Nummer** die Anzahl Bytes, die in die Datenfernverarbeitungsdatei geschrieben werden sollen. Diese Zahl muss kleiner oder gleich dem Wert sein, der mit der Auswahl **LEN** in der Anweisung **OPEN "COM...** gesetzt wurde.

6.4.4. GET

Liest einen Satz aus einer Datei mit Direktzugriff in einen Puffer fuer Direktzugriff.

Syntax: **GET [#]Dateinummer[,Nummer]**

Bemerkungen:

Dateinummer Nummer, mit der die Datei eroeffnet wurde.

Nummer Nummer des zu lesenden Satzes im Bereich 1 bis 16 Megabyte. Ist **Nummer** nicht angegeben, wird der naechste Satz (nach dem letzten GET o. PUT) in den Puffer gelesen.

Nach der Anweisung **GET** koennen **INPUT #, LINE INPUT #** oder Bezugnahmen auf Variablen, die in der Anweisung **FIELD** definiert sind, benutzt werden, um Zeichen aus dem Puffer der Datei mit Direktzugriff zu lesen.
Da BASIC und DCP so viele Saetze wie moeglich in 512-Byte-Sektoren blocken, fuehrt die Anweisung **GET** nur wenn es notwendig ist ein physisches Lesen von der Diskette aus.

GET kann auch fuer Datenfernverarbeitungsdateien benutzt werden. In diesem Falle ist **Nummer** die Anzahl der Bytes, die aus dem Datenfernverarbeitungspuffer gelesen werden sollen. Diese Nummer darf den Wert, der in der Angabe **LEN** der Anweisung **OPEN "COM...** gesetzt wurde, nicht ueberschreiten.

Beispiel:

```
10 OPEN "A:KUND" AS #1
20 FIELD 1,30 AS KUNDNAME$,30 AS ADR$,35 AS STADT$
30 GET 1
40 PRINT KUNDNAME$,ADR$,STADT$
```

In diesem Beispiel wird die Datei KUND fuer Direktzugriff eroeffnet. Die Felder sind in Zeile 20 definiert. Die Anweisung **GET** in Zeile 30 liest einen Satz in den Dateipuffer. Zeile 40 zeigt die Informationen des gelesenen Satzes an.

6.5. Funktionen

6.5.1. MKI$, MKS$, MKD$

Wandelt numerische Werte in Zeichenkettenwerte um.

Syntax:

```
v$ = MKI$(ganzzahliger Ausdruck)
v$ = MKS$(Ausdruck einfacher Genauigkeit)
v$ = MKD$(Ausdruck doppelter Genauigkeit)
```

Bemerkungen:

Jeder numerische Wert, der mit einer Anweisung **LSET** oder **RSET** in den Dateipuffer fuer Direktzugriff gebracht wird, muss in eine Zeichenkette umgewandelt werden. **MKI$** wandelt eine ganze Zahl in eine 2-Byte-Zeichenkette um. **MKS$** wandelt eine Zahl einfacher Genauigkeit in eine 4-Byte-Zeichenkette um. **MKD$** wandelt eine Zahl doppelter Genauigkeit in eine 8-Byte-Zeichenkette um.

Diese Funktionen unterscheiden sich von **STR$** dadurch, dass sie nicht die Daten-Bytes aendern, sondern dadurch, wie BASIC diese Bytes interpretiert.
Siehe auch "Funktionen CVI, CVS, CVD" in diesem Kapitel.

Beispiel:

```
100 FIELD #1,4 AS D$,20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET N$=A$
130 PUT #1
```

In diesem Beispiel wird eine Datei mit Direktzugriff (#1) mit Feldern, die in Zeile 100 definiert sind, benutzt. Im ersten Feld D$ soll ein numerischer Wert AMT stehen. In Zeile 110 wird AMT mit Hilfe von **MKS$** in einen Zeichenkettenwert umgewandelt und mit **LSET** der aktuelle Wert von AMT in den Dateipuffer fuer Direktzugriff gespeichert. In Zeile 120 wird eine Zeichenkette in den Puffer gebracht. In Zeile 130 werden die Daten aus dem Dateipuffer fuer Direktzugriff in die Datei geschrieben.

6.5.2. CVI, CVS, CVD

Wandelt den Inhalt von Zeichenkettenvariablen in numerische Werte um.

Syntax:

```
v = CVI(2-Byte-Zeichenkette)
v = CVS(4-Byte-Zeichenkette)
v = CVD(8-Byte-Zeichenkette)
```

Bemerkungen:

Numerische Werte, die aus einer Datei fuer Direktzugriff gelesen wurden, muessen von Zeichenketten in Zahlen umgewandelt werden. **CVI** wandelt eine 2-Byte-Zeichenkette in eine ganze Zahl um. **CVS** wandelt eine 4-Byte-Zeichenkette in eine Zahl einfacher Genauigkeit um.
CVD wandelt eine 8-Byte-Zeichenkette in eine Zahl doppelter Genauigkeit um.

Die Funktionen **CVI**, **CVS** und **CVD** aendern **nicht** die Bytes der aktuellen Daten. Sie aendern nur die Art, wie BASIC diese Bytes interpretiert.
Siehe auch "Funktionen MKI$, MKS$, MKD$" in diesem Kapitel.

Beispiel:

```
70 FIELD #1, 4 AS N$, 12 AS B$
80 GET #1
90 Y = CVS(N$)
```

In diesem Beispiel wird eine Datei mit Direktzugriff (#1) benutzt, die Felder, wie in Zeile 70 definiert, enthaelt. In Zeile 80 wird ein Satz aus der Datei gelesen. In Zeile 90 wird die Funktion **CVS** benutzt, um die ersten vier Bytes (N$) des Satzes als Zahl einfacher Genauigkeit zu interpretieren. N$ war wahrscheinlich eine Zahl, die vorher mit Hilfe der Funktion **MKS$** in die Datei geschrieben wurde.

6.5.3. EOF

Zeigt die Dateiendebedingung an.

Syntax: v = **EOF(Dateinummer)**

Bemerkungen:

Dateinummer ist die in der Anweisung OPEN angegebene Nummer.

Die Funktion **EOF** ist nuetzlich, um den Fehler "Input past End"(Eingabe nach logischem Ende) zu vermeiden. EOF uebergibt -1 (richtig), wenn das Dateiende bei einer angegebenen Datei erreicht wurde. Eine 0(Null) wird uebergeben, wenn das Dateiende nicht erreicht wurde.

EOF ist nur dann von Bedeutung, wenn eine Datei fuer sequentielle Eingabe von Diskette oder eine Datenfernverarbeitungsdatei eroeffnet wurde. Eine -1 fuer eine Datenfernverarbeitungsdatei bedeutet, dass der Puffer leer ist.

EOF(0) uebergibt die Dateiendebedingung der durch Umadressierung vorliegenden Standardeingabeeinheiten.

Beispiel:

```
10 OPEN "DATA" FOR INPUT AS #1
20 C = 0
30 IF EOF(1) THEN END
40 INPUT #1, M(C)
50 C = C+1:GOTO 30
```

In diesem Beispiel werden die Informationen von der sequentiellen Datei DATA gelesen. Die Werte werden in den Bereich M gelesen, bis das Dateiende erreicht ist.

6.5.4. LOC

Uebergibt die aktuelle Position in der Datei.

Syntax: v = LOC(Dateinummer)

Bemerkungen: **Dateinummer** ist die Dateinummer, die zur Eroeffnung der Datei benutzt wurde.

Fuer Dateien mit Direktzugriff uebergibt **LOC** die Satznummer des letzten gelesenen oder geschriebenen Satzes.
Bei sequentiellen Dateien uebergibt **LOC** die Anzahl der gelesenen oder geschriebenen Saetze der Datei, seit sie eroeffnet wurde. (Ein Satz ist ein Datenblock aus 128 Bytes.)
Ist eine Datei fuer sequentielle Eingabe eroeffnet, liest BASIC den ersten Sektor der Datei, so dass **LOC** eine 1 uebergibt, bevor irgendeine Eingabe von der Datei gelesen wurde.

Fuer eine Datenfernverarbeitungsdatei uebergibt **LOC** die Anzahl der Zeichen im Eingabepuffer, die darauf warten, gelesen zu werden. Die Standardgroesse fuer den Eingabepuffer ist 256 Zeichen. Dies kann aber mit der Auswahl /C: beim Laden des BASIC-Interpreters geaendert werden. Stehen mehr als 255 Zeichen im Puffer, uebergibt **LOC** 255.
Da eine Zeichenkette auf 255 Zeichen beschraenkt ist, erubrigt sich das Testen der Zeichenkettenlaenge, bevor Daten eingelesen werden. Bleiben weniger als 255 Zeichen im Puffer, uebergibt **LOC** den aktuellen Zaehler.

Beispiel:

```
100 IF LOC(1) > 50 THEN STOP
```

Im ersten Beispiel stoppt das Programm, falls ein Satz nach dem 50. Satz der Datei gelesen wurde.

```
100 PUT #1, LOC(1)
```

Das zweite Beispiel kann benutzt werden, um den gerade gelesenen Satz zu ueberschreiben.

6.5.5. LOF

Uebergibt die Anzahl der Bytes, die einer Datei zugeordnet sind (Laenge der Datei).

Syntax: v = **LOF(Dateinummer)**

Bemerkungen:

Dateinummer Dateinummer, die benutzt wurde, um die Datei zu eroeffnen.

Fuer Diskettendateien, uebergibt **LOF** die aktuelle Anzahl von Bytes, die der Datei zugeordnet wurden.

Fuer Datenfernverarbeitungsdateien uebergibt **LOF** die Anzahl der freien Stellen im Eingabepuffer. Das heisst, **Groesse-LOC(Dateinummer)**, wobei **Groesse** die Groesse des Datenfernverarbeitungspuffers ist, dessen Standardannahme 256 Bytes ist, die aber durch die Auswahl /C: beim Laden von BASIC geaendert werden kann. Mit Hilfe von **LOF** kann geprueft werden, wann der Eingabepuffer voll wird.

Beispiel:

Mit folgender Anweisungen wird der letzte Satz der Datei **DATA** gelesen, wobei angenommen wird, dass **DATA** mit einer Satzlaenge von 128 Bytes erstellt wurde.

```
10 OPEN "DATA" AS #1
20 GET #1, LOF(1)/128
```

6.5.6. INPUT$

Uebergibt eine Zeichenkette von n Zeichen, die von einer Datei gelesen werden.
(Siehe Fkt. INPUT$ unter Kapitel 5. "Grundfunktionen"!)

6.6. Treiberunterstuetzung

Das Dateiein-/ausgabesystem in BASIC ermoeglicht die Verwendung von benutzerinstallierten Einheiten.
Der Benutzer koennte beispielsweise sein eigenes Programm fuer einen Einheitentreiber schreiben, um **LPT1:** zu ersetzen. Dieses Programm wuerde den Treiber wie folgt eroeffnen :

OPEN"LPT1" FOR Modus AS Dateinummer

Beim Erstellen eines Einheitentreibers muessen folgende Regeln beachtet werden:

- Der Name der installierten Einheit darf nicht mit einem Doppelpunkt (:) enden, da Einheitennamen mit einem Doppelpunkt fuer bestimmte Einheiten reserviert sind (KEYBD:,SCRN:usw.)

- LPT1:, LPT2: und LPT3: sind die einzigen Einheitentreiber des Systems, die durch einen installierten Einheitentreiber gleichen Namens ersetzt werden koennen.

- Die Satzlaenge ist auf 1 gesetzt, ausser sie wird durch eine Anweisung OPEN geaendert.

  OPEN Dateiangabe [FOR Modus] AS Dateinummer
  LEN = Satzlaenge

  Bei dieser Auswahl speichert BASIC die Zeichenanzahl, die der **Satzlaenge** entspricht, bevor sie zum Treiber gesendet wird.

- BASIC fuegt am Zeilenende nur einen Wagenruecklauf (&HØD) an. Wird ausserdem ein Zeilenvorschub (&HØA) benoetigt, muss dieser vom Treiber vorgesehen werden.

- Der Einheitentreiber muss an BASIC eine "Dateiende"-Bedingung uebergeben koennen, wenn eine sequentielle Eingabedatei zu einem Einheitentreiber eroeffnet werden soll. Wenn BASIC versucht, ueber das Ende des Einheiteneingabestroms zu lesen, muss der Treiber ein ^Z(CTRL-Z) uebergeben, wodurch der Fehler "Input past end" (Eingabe nach logischem Ende) auftritt.

7. Datenfernverarbeitung

In diesem Kapitel werden die BASIC-Anweisungen und -Funktionen beschrieben, die benoetigt werden, um die Schnittstelle V.24 asynchrone Datenfernverarbeitung mit anderen Computern und Peripheriegeraeten zu unterstuetzen.
Weitere Hinweise und Beispiele zur Datenfernverarbeitung sind aus der Dokumentation "Anwendung des BASIC-Systems" zu entnehmen.

7.1. Eroeffnen und Schliessen einer Datenfernverarbeitungsdatei

7.1.1. OPEN "COM...

Eroeffnet eine Datei fuer Datenfernverarbeitung.

Syntax: **OPEN "COMn: [Geschwindigkeit][,Paritaet][,Daten][,Stopp][,RS][,CS[n]][,DS[n]][,CD[n]][,LF][,PE]" AS [#] Dateinummer [LEN=Nummer]**

Bemerkungen:

Durch **OPEN"COM...** wird ein Puffer fuer die Ein-/Ausgabe in der gleichen Art und Weise wie mit dem OPEN fuer Diskettendateien angelegt.

n ist 1 oder 2 und bedeutet die Nummer des Adapters fuer asynchrone Datenfernverarbeitung.

Geschwindigkeit Ganzzahlige Konstante, die die Sende-/Empfangs-Bit-Rate in Bits pro Sekunde (bps) angibt. Gueltige Geschwindigkeiten sind 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 und 9600 bps. Die Standardannahme ist 300 bps.

Paritaet Konstante, bestehend aus einem Zeichen, die die Paritaet fuer Senden und Empfangen wie folgt angibt:

S SPACE: Das Paritaets-Bit wird immer als Leerstelle (0 Bit) gesendet oder empfangen.

O ODD: Fuer Senden und Empfangen wird auf ungerade Bit-Paritaet geprueft.

M MARK: Das Paritaets-Bit wird immer als Einer-Bit gesendet oder empfangen.

E EVEN: Beim Senden und Empfangen wird auf gerade Bit-Paritaet geprueft.

N NONE: Beim Senden und Empfangen wird nicht auf Bit-Paritaet geprueft.

Die Standardannahme ist **EVEN** (E) = gerade.

Daten
Ganzzahlige Konstante, die angibt, wie viele Daten-Bits gesendet oder empfangen werden sollen. Gueltige Werte sind 5, 6, 7 oder 8. Die Standardannahme ist 7.

STOPP
Ganzzahlige Konstante, die die Anzahl der Stopp-Bits angibt. Gueltige Werte sind 1 oder 2. Die Standardannahme sind zwei Stopp-Bits fuer 75 und 110 bps, ein Stopp-Bit fuer alle anderen. Wird fuer Daten 5 ausgewaehlt, bedeutet hier eine 2 (zwei):1 1/2 Stopp-Bits.

Dateinummer
Ganzzahliger Ausdruck, der eine gueltige Dateinummer errechnet. Diese Zahl ist so lange einer Datei zugeordnet, wie sie eroeffnet ist, und wird mit anderen Ein-/Ausgabeanweisungen fuer Datenfernverarbeitung benutzt, um diese Datei anzusprechen.

Nummer
Maximale Anzahl von Bytes, die aus dem oder in den Datenfernverarbeitungspuffer mit Hilfe von **GET** oder **PUT** gelesen oder geschrieben werden koennen. Die Standardannahme ist 128 Bytes.

OPEN "COM... legt in der gleichen Art und Weise fuer Ein-/Ausgabe einen Puffer an, wie **OPEN** fuer Diskettendateien. Die Schnittstelle V.24 asynchrone Datenfernverarbeitung mit anderen Computern und peripheren Einheiten wird unterstuetzt.

Die Datenfernverarbeitungseinheit kann zu gleicher Zeit nur fuer eine Dateinummer eroeffnet werden.

Die Angaben **RS, CS, DS, CD, LF** und **PE** beeinflussen die Leitungssignale wie folgt:

RS unterdrueckt RTS (Request to send) (Sendeteil einschalten)

CS[n] steuert CTS (Clear to send) (Sendebereitschaft)

DS[n] steuert DSR (Data set ready) (Betriebsbereitschaft)

CD[n] steuert CD (Carrier defect) (Empfangssignalpegel)

LF sendet nach jedem Wagenruecklauf einen Zeilenvorschub

PE ermoeglicht eine Paritaetspruefung

Der Empfangssignalpegel CD (Carrier defect) wird auch RLSD (Receive Line Signal Detect) (Empfangssignalpegel) genannt.

Hinweis: Die Parameter **Geschwindigkeit, Paritaet, Daten** und **Stopp** muessen in dieser Reihenfolge angegeben sein. Nicht aber RS, CS, DS, CD, LF und PE.

Die Leitung RTS (Sendeteil einschalten) wird eingeschaltet, wenn die Anweisung **OPEN "COM...** ausgefuehrt wird, ausser es wird die Auswahl **RS** angegeben.

Das Argument **n** in den Auswahlen **CS**, **DS** und **CD** gibt die Anzahl der Millisekunden an, die auf das Signal gewartet werden soll, bevor der Fehler "Device Timeout" (Einheitenzeitsperre) angezeigt wird. n kann sich im Bereich 0 bis 65535 befinden. Wird n weggelassen oder ist gleich 0 (Null), wird der Leitungsstatus nicht geprueft.

Die **Standardannahmen** sind **CS1000**, **DS1000** und **CD0**. Wurde **RS** angegeben, ist die Standardannahme **CS0**.

Das bedeutet, dass normalerweise Ein-/Ausgabeanweisungen zu einer Datenfernverarbeitungsdatei keine Verbindung ergeben, wenn CTS (Sendebereitschaft) oder DSR (Betriebsbereitschaft) aus sind. Das System wartet eine Sekunde, bevor es den Fehler "Device Timeout" (Einheitenzeitsperre) zurueecksendet. Die Angaben **CS** und **DS** erlauben es, diese Leitungen zu ignorieren oder die Zeit anzugeben, die zu warten ist, bevor eine Zeitsperre erfolgen soll.

Normalerweise wird **CD** (Empfangssignalpegel) oder RLSD ignoriert, wenn die Anweisung **OPEN "COM...** ausgefuehrt wird. Die Auswahl **CD** erlaubt es, diese Leitung zu testen, indem man den Parameter in der gleichen Weise wie fuer **CS** und **DS** einfuegt. Wird n weggelassen oder ist n gleich 0 (Null), wird der Empfangssignalpegel ueberhaupt nicht getestet (dies ist das gleiche, als ob die Auswahl **CD** weggelassen wuerde).

Der Parameter **LF** (Zeilenvorschub) ist vorgesehen, wenn Datenfernverarbeitungsdateien verwendet werden sollen, um eine Ausgabe auf einen seriellen Drucker zu uebertragen. Wird **LF** angegeben, wird ein Zeilenvorschubzeichen (Hex 0A) automatisch nach jedem Wagenruecklaufzeichen (Hex 0D) gesendet. (Dies schliesst einen gesendeten Wagenruecklauf als Ergebnis der Druckbreitensetzung ein.) Es ist zu beachten, dass **INPUT#** und **LINE INPUT#** stoppen, sobald ein Wagenruecklaufzeichen erkannt wird, wenn sie dazu benutzt werden, von einer mit Auswahl **LF** eroeffneten Datenfernverarbeitungsdatei zu lesen. Das Zeilenvorschubzeichen wird immer ignoriert.

Mit der Auswahl **PE** wird die Paritaetspruefung aktiviert. Die Standardannahme ist keine Paritaetspruefung. Durch die Auswahl **PE** wird bei Paritaetsfehlern "Device I/O error" (Einheiteneinausgabefehler) angezeigt und das hoechstrangige Bit wird fuer 7 oder weniger Datenbits eingeschaltet. Die Auswahl **PE** beeinflusst nicht Rahmenfehler und Ueberlauffehler. Durch diese Fehler wird immer das hoechstrangige Bit eingeschaltet und der Fehler "Device I/O error" (Einheiteneinausgabefehler) angezeigt.

Jeder Codierfehler innerhalb des Zeichenkettenausdrucks, beginnend mit **Geschwindigkeit**, ergibt den Fehler "Bad File Name" (Falscher Dateiname). Es wird nicht angegeben, welcher Parameter falsch ist.

Wenn acht Daten-Bits angegeben werden, muss die Paritaet N spezifiziert werden. BASIC benutzt alle acht Bits in einem Byte, um Zahlen zu speichern. Das bedeutet, dass acht Daten-Bits angegeben werden muessen, wenn numerische Daten gesendet oder empfangen werden (zB. mit Hilfe von PUT). (Dies ist nicht der Fall, wenn numerische Daten als Text gesendet werden.)

Beispiele:

```
10 OPEN "COM1:" AS #1
```

Die Datei 1 wird fuer Datenfernverarbeitung mit allen Standardannahmen eroeffnet. Die Geschwindigkeit ist 300 bps mit gerader Paritaet, sieben Daten-Bits und einem Stopp-Bit.

```
20 OPEN "COM2:1200,N,8" AS #1
```

Die Dateinummer 1 wird fuer asynchrone Ein-/Ausgabe mit 1200 bps eroeffnet, wobei keine Paritaet erzeugt und geprueft wird. Acht Bits werden gesendet und empfangen, und ein Stopp-Bit wird gesendet.

```
10 OPEN "COM1:9600,N,8,,CS,DS,CD" AS #1
```

Eroeffnet COM1: mit 9600 bps, keiner Paritaet und acht Daten-Bits. CTS, DSR und CD werden nicht geprueft.

```
20 OPEN "COM1:1200,,,,CS,DS2000" AS #1
```

Eroeffnet COM1: mit 1200 bps und der Standardannahme gerade Paritaet, sieben Daten-Bits und ein Stopp-Bit. RTS wird gesendet, CTS wird nicht geprueft, und "Device Timeout" (Einheitenzeitsperre) wird angezeigt, falls DSR nicht innerhalb von zwei Sekunden angezeigt wird.

Man muss beachten, dass die Kommata benoetigt werden, um die Positionen fuer die Parameter **Paritaet**, **Start** und **Stopp** anzuzeigen, auch wenn diese Werte nicht angegeben werden.

Die Anweisung **OPEN** kann in Verbindung mit der Anweisung **ON ERROR** benutzt werden, um sich zu versichern, dass ein Modem richtig arbeitet, bevor Daten gesendet werden. Im folgenden Programm wird geprueft, ob **CD** (Empfangssignalpegel) vom Modem gesendet wird, bevor mit der Uebertragung begonnen wird. In Zeile 20 wird eine Zeitsperre nach 10 Sekunden gesetzt. T wird auf 6 gesetzt, so dass das Programm abbricht, wenn das Signal **CD** nicht innerhalb einer Minute gesendet wird. Ist die Datenfernverarbeitungsleitung einmal aufgebaut, wird die Datei mit einer kuerzeren Verzoegerung bis zur Zeitsperre neu eroeffnet.

```
10 T=6:ON ERROR GOTO 100
20 OPEN "COM1:300,N,8,2,CS,DS,CD1000" AS #1
30 ON ERROR GOTO 0
40 CLOSE #1   arbeitet, deshalb fortfahren
50 GOTO 1000
 .
 .
 .
100 T=T-1
110 IF T=0 THEN ON ERROR GOTO 0  abbrechen
120 RESUME
 .
 .
 .
1000 OPEN "COM1:300,N,8,2,CS,DS,CD2000" AS #1
```

Das naechste Beispiel zeigt einen typischen Weg, wie eine Datenfernverarbeitungsdatei benutzt werden kann, um einen seriellen Drucker zu steuern. Durch den Parameter **LF** in der Anweisung **OPEN** wird verhindert, dass die Ausgabezeilen uebereinander gedruckt werden.

```
10 WIDTH "COM1:" ,132
20 OPEN "COM1:1200,N,8,,CS10000,DS10000,CD10000,LF" AS #1
```

7.1.2. CLOSE

Schliesst eine Datei fuer Datenfernverarbeitung.

Syntax: **CLOSE[[#] Dateinummer [,[#] Dateinummer]...]**

siehe auch 6. Dateiarbeit, Anweisung **CLOSE**

7.2. Anweisungen und Funktionen fuer die Datenfernverarbeitungs-Ein-/Ausgabe

Da jeder Datenfernverarbeitungsanschluss als Datei eroeffnet wird, sind alle Ein-/Ausgabeanweisungen und Funktionen, die fuer Diskettendateien gueltig sind, auch fuer Datenfernverarbeitung gueltig.

7.2.1. Sequentielle Ein-/Ausgabe

Sequentielle Eingabeanweisungen fuer Datenfernverarbeitung sind die gleichen wie fuer Diskettendateien. Es sind die folgenden:

```
INPUT#
LINE INPUT#
INPUT$
```

Die sequentiellen Ausgabeanweisungen fuer Datenfernverarbeitung sind die gleichen wie fuer Diskettendateien. Es sind die folgenden:

```
PRINT#
PRINT# USING
WRITE#
```

Siehe dazu auch unter Kapitel 6 "Dateiarbeit".

7.2.2. GET und PUT fuer Datenfernverarbeitungsdateien

GET und PUT unterscheiden sich fuer Datenfernverarbeitungsdateien nur wenig von denen fuer Diskettendateien. Sie werden fuer Ein-/Ausgabe fester Laenge zu oder von Datenfernverarbeitungsdateien benutzt. Statt die Satznummer anzugeben, die geschrieben oder gelesen werden soll, gibt man die Anzahl der Bytes an, die in oder aus dem Dateipuffer uebertragen werden sollen.

Diese Zahl darf den Wert, der durch die Angabe LEN in der Anweisung OPEN"COM... gesetzt wurde, nicht ueberschreiten. Siehe unter GET und PUT in Kapitel 6.

7.2.3. Ein-/Ausgabefunktionen fuer Datenfernverarbeitung

Der schwierigste Aspekt fuer die asynchrone Datenfernverarbeitung ist es, die Zeichen so schnell zu verarbeiten, wie sie empfangen werden. Bei Datenraten von 1200 bps oder hoeher ist es noetig, die Zeichenuebertragung vom anderen Computer einige Zeit zu unterbrechen, um "aufzuholen". Dies kann durch Senden von XOFF(CHR$(19)) zum anderen Computer geschehen, und XON(CHR$(17)), wenn die Uebertragung wieder aufgenommen werden kann. XOFF sagt dem anderen Computer, dass er mit der Uebertragung aufhoeren soll, XON sagt ihm, dass er wieder senden kann.

Hinweis:

Dies ist eine allgemein benutzte Konvention, die aber nicht universal ist. Sie haengt von dem implementierten Protokoll ab, das zwischen dem anderen Computer oder Peripheriegeraet und dem eigenen Computer aufgestellt wurde.

- Testen der Ueberlaufbedingungen:

Es gibt drei Funktionen, die helfen eine **Ueberlaufbedingung** zu bestimmen. Es sind die folgenden (siehe dazu auch Kapitel 6):

LOC(f) Uebergibt die Anzahl der Zeichen im Eingabepuffer, die darauf warten, gelesen zu werden. Die Standardgroesse fuer den Eingabepuffer ist 256 Zeichen, die aber durch die Auswahl /C: beim Laden von BASIC geaendert werden kann. Ist die Anzahl groesser als 255, uebergibt LOC 255.

LOF(f) Uebergibt die Anzahl der freien Stellen im Eingabepuffer. Dies ist das gleiche wie n-LOC(f), wobei n die Groesse des Datenfernverarbeitungspuffers ist, wie er durch die Auswahl/C: beim Laden des BASIC-Interpreters gesetzt wurde. Die Standardannahme fuer n ist 256.
Mit Hilfe von LOF kann geprueft werden, wann der Eingabepuffer voll ist.

EOF(f) Uebergibt Wahr (-1), wenn der Eingabepuffer leer ist; und "Falsch" (0), wenn irgendwelche Zeichen darauf warten, gelesen zu werden.

Hinweis:
Es kann ein "Communication Buffer Overflow" (Datenfernverarbeitungspufferueberlauf) auftreten, wenn ein Lesen versucht wird, nachdem der Eingabepuffer voll ist (wenn **LOF(f)** eine 0 uebergibt).

- Funktion INPUT$

Die Funktion **INPUT$** wird den Anweisungen **INPUT#** und **LINE INPUT#** vorgezogen, wenn Datenfernverarbeitungsdateien gelesen werden sollen, da alle ASCII-Zeichen signifikant in der Datenverarbeitung sein koennen. **INPUT#** wird am wenigsten benutzt, da die Eingabe stoppt, sobald ein Komma oder ein Zeichen CR/LF (Wagenruecklauf/Zeilenschaltung) gelesen wird. **LINE INPUT#** stoppt, wenn ein Zeichen CR/LF gelesen wird.

Mit **INPUT$** koennen alle Zeichen gelesen und einer Zeichenkette zugeordnet werden. **INPUT$(n,f)** uebergibt n Zeichen aus der Datei #f. Mit den folgenden Anweisungen kann von einer Datenfernverarbeitungsdatei gelesen werden:

```
110 WHILE NOT EOF(1)
120 A$ = INPUT$(LOC(1),#1)
  .
  .
  .
(Daten in A$ verarbeiten)
  .
  .
  .
190 WEND
```

Mit diesen Anweisungen werden die Zeichen aus dem Puffer nach AB gebracht und verarbeitet, und zwar so lange, wie sich Zeichen im Eingabepuffer befinden. Befinden sich mehr als 255 Zeichen im Puffer, werden auf einmal nur 255 Zeichen uebertragen, damit kein String Overflow (Zeichenkettenueberlauf) auftritt. In diesem Fall wird EOF(1) "falsch" (Ø) und die Eingabe geht weiter, bis der Eingabepuffer leer ist.

Um die Daten schnell zu verarbeiten, sollte man nach Moeglichkeit vermeiden, jedes Zeichen zu pruefen, wenn es empfangen wird. Werden Sonderzeichen gesucht (wie z.B. Steuerzeichen), kann man dies mit Hilfe der Funktion **INSTR** tun, um sie in der Eingabezeichenkette zu finden.

7.3. Unterbrechungsabfrage

7.3.1. COM(n)

Aktiviert oder inaktiviert die Ablaufverfolgung von Datenfernverarbeitungsaktivitaeten mit dem angegebenen Uebertragungsadapter.

Syntax: **COM(n) ON**
COM(n) OFF
COM(n) STOP

Bemerkungen:

n Nummer des Datenfernverarbeitungsanschlusses (1 oder 2)

Die Anweisung **COM(n) ON** muss ausgefuehrt werden, damit mit der Anweisung ON COM(n) eine Unterbrechung abgefragt werden kann. Wurde in der Anweisung ON COM(n) eine Leitung ungleich Null angegeben, prueft BASIC nach Ausfuehrung von COM(n) ON nach jeder Ausfuehrung einer neuen Anweisung, ob ein Zeichen ueber den Datenfernverarbeitungsanschluss gekommen ist.

Ist die Anweisung **COM(n) OFF** ausgefuehrt, findet keine Unterbrechung statt, und es wird keine Datenfernverarbeitungsaktivitaet registriert, auch wenn sie stattgefunden hat.

Wurde die Anweisung **COM(n) STOP** ausgefuehrt, kann keine Unterbrechung stattfinden. Es wird jedoch jede Datenfernverarbeitungsaktivitaet, die stattfindet, registriert, so dass eine sofortige Unterbrechung stattfindet, sobald die Anweisung COM(n) ausgefuehrt wird.

7.3.2. ON COM(n)

Setzt fuer BASIC eine Zeilennummer, zu der verzweigt werden soll, sobald eine Information im Datenfernverarbeitungspuffer steht.

Syntax: ON COM(n) GOSUB Zeile

Bemerkungen:

n ist die Nummer des Datenfernverarbeitungsanschlusses (1 oder 2)

Zeile Zeilennummer des Beginns einer Unterbrechungsroutine. Wird **Zeile** auf 0 (Null) gesetzt, wird die Unterbrechung fuer die Datenfernverarbeitungsaktivitaet fuer den angegebenen Adapters inaktiviert.

Die Anweisung **COM(n) ON** muss ausgefuehrt werden, um diese Anweisung fuer den Anschluss n zu aktivieren. Nach der Ausfuehrung dieser Anweisung - falls die Zeilennummer nicht 0 (Null) war - prueft BASIC vor der Ausfuehrung jeder neuen Anweisung, ob ein Zeichen in dem angegebenen Datenfernverarbeitungsanschluss angekommen ist. Steht ein neues Zeichen zur Verfuegung, verzweigt BASIC mit **GOSUB** zur angegebenen Zeile.

Wurde **COM(n) OFF** ausgefuehrt, findet keine Unterbrechung fuer den Anschluss statt. Falls ueber die Datenfernverarbeitungsleitung Zeichen uebertragen werden, wird dies nicht registriert.

Wird die Anweisung **COM(n) STOP** ausgefuehrt, findet fuer den Anschluss keine Unterbrechung statt. Es werden jedoch alle empfangenen Zeichen gespeichert, so dass sofort eine Unterbrechung stattfindet, wenn **COM(n) ON** ausgefuehrt wird.
Erfolgt eine Unterbrechung, wird automatisch **COM(n) STOP** ausgefuehrt, so dass niemals eine rekursive Unterbrechung auftreten kann.

Durch **RETURN** aus der Unterbrechungsroutine wird automatisch **COM(n) ON** ausgefuehrt, ausser es wird explizit **COM(n) OFF** innerhalb der Unterbrechungsroutine ausgefuehrt.

Eine Unterbrechung kann nicht stattfinden, wenn BASIC kein Programm ausfuehrt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung **ON ERROR**), werden alle Unterbrechungen automatisch inaktiviert (einschliesslich ON COM, ON ERROR, ON PLAY und ON TIMER).

Die Datenfernverarbeitungs-Unterbrechungsroutine liest eine gesamte Nachricht von der Datenfernverarbeitungsleitung, bevor zurueckverzweigt wird. Es wird nicht empfohlen, die Datenfernverarbeitungsunterbrechung fuer Nachrichten aus einzelnen Buchstaben zu benutzen, da hohe Baud-Raten zu viel Zeit fuer die Unterbrechung und das Lesen jedes einzelnen Zeichens benoetigen, so dass vielleicht der Unterbrechungspuffer fuer Datenfernverarbeitungen ueberlaufen kann.

Man kann mit **RETURN Zeile** in einem BASIC-Programm zu einer festen Zeilennummer zurueckverzweigen. Die Benutzung dieser Zurueckverzweigung muss sehr vorsichtig gehandhabt werden, da andere GOSUBs, WHILEs oder FORs zur Zeit der Unterbrechung aktiv sind und aktiv bleiben.

Beispiel:

```
100 ON COM(1) GOSUB 600
110 COM(1) ON
.
.
.
600  ankommende Zeichen
.
.
.
690 RETURN 400
```

In diesem Beispiel wird fuer den ersten Datenfernverarbeitungsanschluss eine Unterbrechungsroutine in Zeile 600 angelegt.

7.4. Datenfernverarbeitungsfehler

Fehler treten fuer Datenfernverarbeitungsdateien in der folgenden Reihenfolge auf:

1. **Beim Eroeffnen der Datei** -
   a) "Device Timeout" (Einheitenzeitsperre): Wenn eines der zu testenden Signale (**CTS**, **DSR** oder **CD**) fehlt.

2. **Beim Empfangen von Daten** -
   a) "Com Buffer Overflow" (Datenfernverarbeitungspufferueberlauf): Wenn ein Ueberlauf eintritt.
   b) "Device I/O Error" (Einheitenein-/ausgabefehler): Fuer Ueberlauf, Unterbrechung, Paritaet, oder Rahmenfehler.
   c) "Device Fault" (Einheitenfehler): Wenn **DSR** oder **CD** verlorengeht.

3. **Beim Senden von Daten** -
   a) "Device Fault" (Einheitenfehler): Falls **CTS**, **DSR** oder **CD** bei einer Modemstatusunterbrechung verlorengehen, waehrend BASIC etwas anderes ausfuehrte.
   b) "Device Timeout" (Einheitenzeitsperre): Wenn **CTS**, **DSR** oder **CD** verlorengehen, waehrend das Programm darauf wartet, Daten in den Ausgabepuffer zu stellen.

8. Anweisungen und Funktionen zu Grafik und Musik

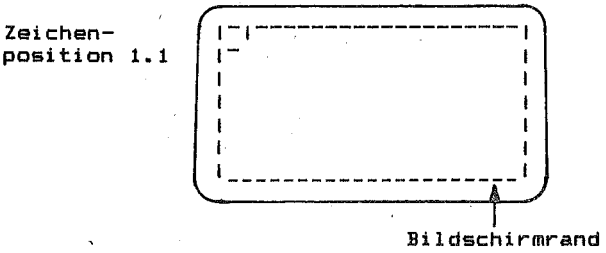
8.1. Allgemeine Hinweise zur Arbeit mit dem Bildschirm

Mit **BASIC** kann man Texte, Sonderzeichen, Punkte, Linien oder komplexere Gebilde farbig oder in schwarz-weiss anzeigen.
Der Personalcomputer besitzt zwei Bildschirmanschluesse: den Schwarz/Weiss-Bildschirm- und den Farb-/Grafik-Bildschirmanschluss.
Mit dem Schwarz/Weiss-Bildschirmanschluss ist eine Textanzeige in schwarz und weiss moeglich. Als Text werden die Buchstaben, Zahlen und alle Sonderzeichen des normalen Zeichensatzes verstanden. Es ist moeglich, Bilder mit Hilfe von Speziallinien und Blockzeichen zu bilden. Blinken, Umkehranzeige, Anzeigenunterdrueckung, Intensivanzeige und Unterstreichen werden durch Setzen von Parametern in der Anweisung **COLOR** unterstuetzt.

Der Farb-/Grafik-Bildschirmanschluss arbeitet im Grafik- und auch im Textmodus. Er erlaubt, Texte in 16 verschiedenen Farben anzuzeigen (die Anzeige nur in schwarz-weiss ist ebenfalls moeglich). Es sind alle grafischen Moeglichkeiten gegeben, komplexe Bilder zu zeichnen.
Mit dieser grafischen Faehigkeit kann man den Bildschirm mit allen Punkten in mittlerer und hoher Aufloesung adressieren.

Textmodus

Der Bildschirm kann wie folgt dargestellt werden:



Die Zeichen werden in 25 Zeilen auf dem Bildschirm angezeigt. Diese Zeilen sind von oben nach unten von 1 bis 25 numeriert. Jede Zeile besteht aus 40 Zeichenpositionen (oder 80, abhaengig vom Wert der Anweisung **WIDTH**). Sie sind von links nach rechts von 1 bis 40 (oder 80) numeriert.
Die Zeile 25 wird normalerweise benutzt, die "Funktionstasten" anzuzeigen (siehe "Anweisung **KEY**" in Kapitel 4), aber es ist moeglich, diesen Bereich des Bildschirms zu ueberschreiben, wenn die Anzeige der "Funktionstasten" ausgeschaltet wird. Die Zeile 25 wird von **BASIC** niemals nach oben geschoben.

Jedes Zeichen auf dem Bildschirm besteht aus zwei Teilen: Vordergrund und Hintergrund. Der Vordergrund ist das Zeichen selbst. Der Hintergrund ist der Teil um das Zeichen. Man kann mit Hilfe der Anweisung **COLOR** die Vordergrund- und Hintergrundfarbe fuer jedes Zeichen setzen. Es ist auch moeglich, die Zeichen blinken zu lassen.

Es stehen 16 Farben zur Verfuegung, wenn der Farb-/Grafikbildschirmanschluss benutzt wird (siehe "Anweisung **COLOR**").

Die meisten Fernseher oder Monitore haben einen **"Rand"**, der ausserhalb des Bereiches liegt, der fuer die Zeichen benutzt wird. Mit der Anweisung **COLOR** kann auch der Rand farbig angezeigt werden.
Es folgt eine Auflistung aller Anweisungen, die benutzt werden koennen, um im Textmodus Informationen anzuzeigen:

CLS	SCREEN
COLOR	WIDTH
LOCATE	WRITE
PRINT	

Die folgenden Funktionen und Systemvariablen koennen im Textmodus benutzt werden:

CSRLIN	SPC
POS	TAB
SCREEN	

Im Textmodus ist es moeglich, wenn der Farb-/Grafikbildschirmanschluss vorhanden ist, mit mehreren Bildschirmseiten zu arbeiten.
Der Farb-/Grafikbildschirmanschluss hat einen **16K**-Bildschirmpuffer. Im Textmodus werden aber nur **2K** benoetigt (oder **4K** fuer eine Breite von 80 Spalten). So ist der Puffer in verschiedene **Seiten** aufgeteilt, die individuell beschrieben und/oder angezeigt werden koennen. Bei der Breite von 40 Spalten gibt es acht Seiten, numeriert von 0 bis 7, bei der Breite von 80 Spalten 4 Seiten, numeriert von 0 bis 3.
Einzelheiten siehe Anweisung **SCREEN**.

Grafischer Modus

Der grafische Modus ist nur verfuegbar, wenn der Farb-/Grafikbildschirmanschluss vorhanden ist.

Die **BASIC**-Anweisungen erlauben es, in zwei grafischen Aufloesungen zu arbeiten:

- mittlere Aufloesung: 320 x 200 Punkte und 4 Farben
- hohe Aufloesung: 640 x 200 Punkte und 2 Farben

Die Aufloesung kann mit Hilfe der Anweisung **SCREEN** ausgewaehlt werden.

Ausserdem ist es moeglich, rechteckige Ausschnitte auf dem Bildschirm zu definieren, in denen die Abbildungen erfolgen sollen und die Bildschirmkoordinaten neu festzulegen.

Die in **BASIC** verfuegbaren Anweisungen fuer Grafiken sind folgende:

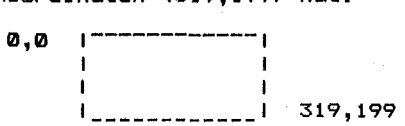
CIRCLE	PAINT
COLOR	PRESET
DRAW	PSET
GET	PMAP
LINE	PUT
	SCREEN
	VIEW
	WINDOW

Als grafische Funktion ist verfuegbar:

POINT

Mittlere Aufloesung:

Bei der mittleren Aufloesung gibt es horizontal 320 und vertikal 200 Punkte. Diese Punkte werden von links nach rechts und von oben nach unten, beginnend mit 0 (Null), numeriert. Das bedeutet, dass die obere linke Ecke des Bildschirmes die Koordinaten (0,0) und die untere rechte Ecke des Bildschirmes die Koordinaten (319,199) hat.



In der mittleren Aufloesung kann man eine Farbnummer von 0,1,2 oder 3 angeben.
Mit der Anweisung **COLOR** wird die Hintergrundfarbe und eine aus zwei Farbpaletten ausgewaehlt.
Eine Farbpalette besteht aus drei Farben, die den Farbnummern 1,2 und 3 zugeordnet werden. Wird eine Palette mit Hilfe der Anweisung **COLOR** geaendert, werden alle Farben des Bildschirmes geaendert, um sich der neuen Farbpalette anzugleichen.

Auch wenn man sich im grafischen Modus befindet, ist es moeglich, Textzeichen am Bildschirm anzuzeigen. Die Groesse der Zeichen ist die gleiche wie im Textmodus. Das bedeutet 25 Zeilen zu je 40 Zeichen.
Bei der mittleren Aufloesung ist die Standardannahme fuer den Vordergrund die Farbe Nr. 3, die Farbe Nr. 0 ist die Hintergrundfarbe.

- Hohe Aufloesung:

Bei der hohen Aufloesung gibt es horizontal 640 Punkte und vertikal 200 Punkte. Wie bei der mittleren Aufloesung werden diese Punkte beginnend mit Null numeriert, so dass der unterste rechte Bildschirmpunkt die Koordinaten (639,199) hat.

Die hohe Aufloesung besteht nur aus zwei Farben: 0 (Null) und 1 (Eins). Null ist immer schwarz und Eins ist immer weiss.

Werden Textzeichen in hoher Aufloesung ausgegeben, gehen 80 Zeichen auf eine Zeile. 1 (Eins) ist die Vordergrundfarbe und 0 (Null) die Hintergrundfarbe. So sind die Zeichen immer Weiss auf schwarz.

Angaben von Koordinaten:
Die grafischen Anweisungen benoetigen eine Information, wo auf dem Schirm gezeichnet werden soll. Man gibt diese Information mit Hilfe von Koordinaten an. Koordinaten haben allgemein die Form (x,y), wobei x die horizontale Position und y die vertikale Position angibt. Diese Form ist auch als **absolute Form** bekannt und bezieht sich auf die aktuellen Koordinaten des Punktes auf dem Bildschirm, ohne den letzten angesprochenen Punkt zu beachten.

Es gibt auch die relative Form der Angabe der Koordinaten. Mit Hilfe dieser Form wird **BASIC** angezeigt, wo sich der Punkt relativ zum letzten angesprochenen Punkt befindet. Die Form sieht wie folgt aus:

(x-Relativzeiger,y-Relativzeiger)

Innerhalb der Klammern wird der relative Abstand der horizontalen und vertikalen Richtungen vom letzten Punkt aus angegeben.

Der "letzte angesprochene Punkt" wird von jeder Grafikanweisung gesetzt.

Hinweis:

Mit den Grafikanweisungen darf nicht ueber die Grenzen des Bildschirms gezeichnet werden.

Das Beispiel zeigt die Benutzung der beiden Koordinatenformen:

```
10 SCREEN 1
20 PSET (150,150) 'absolute Form
30 PSET STEP (50,-50) 'relative Form
```

Hierbei werden zwei Punkte auf dem Bildschirm gesetzt. Ihre aktuellen Koordinaten sind (150,150) und (200,100).

8.2. Allgemeingueltige Anweisungen

CLS

Loeschen des Bildschirmes

Syntax: CLS

Bemerkungen:

Im Textmodus wird die aktive Seite (siehe Anweisung **SCREEN**) auf die Hintergrundfarbe geloescht (siehe Anweisung **COLOR**).

Im Grafikmodus (mittlere oder hohe Aufloesung) wird der gesamte Bildschirmpuffer auf die Hintergrundfarbe geloescht.

Durch die Anweisung **CLS** wird der Kursor auf seine Grundposition gebracht. Im Textmodus bedeutet das, dass der Kursor in die obere linke Ecke des Bildschirms gebracht wird. Im grafischen Modus bedeutet dies, dass der Kursor in die Mitte des Bildschirms ((160,100) fuer mittlere Aufloesung, (320,100) fuer hohe Aufloesung) gebracht wird. Diese Position wird gleichzeitig zum "letzten angesprochenen Punkt" nach dem Loeschen des Bildschirms.

Wird der Bildschirmmodus oder die Breite durch die Anweisung **SCREEN** oder **WIDTH** geaendert, wird der Bildschirm ebenfalls geloescht. Der Bildschirm kann auch durch Betaetigen der Tasten **CTRL-HOME** geloescht werden.
Wird die Anweisung **VIEW** verwendet, wird nur der momentane Bildschirmausschnitt geloescht. Um den ganzen Bildschirm zu loeschen, muss die Anweisung **VIEW** benutzt werden, um den Bildschirmausschnitt zu inaktivieren und der Bildschirm anschliessend mit **CLS** geloescht werden.

Beispiel:

```
10 COLOR 4,2
20 CLS
```

Mit dem Farb-/Grafikbildschirmanschluss wird der Bildschirm in diesem Beispiel auf gruen geloescht.

=_SCREEN

Setzt die Bildschirmattribute, die von den nachfolgenden Anweisungen benutzt werden sollen.
Nur mit dem Anschluss fuer den Farb-/Grafikbildschirm von Bedeutung.

Syntax: **SCREEN [Modus][,[Aendern][,[Aseite][,Vseite]]]**

Bemerkungen:

Modus Numerischer Ausdruck, dessen Ergebnis ein ganzzahliger Wert von 0,1 oder 2 ist. Gueltige Modi sind:

- **0** Textmodus mit der laufenden Breite (40 oder 80).
- **1** grafischen Modus mittlere Aufloesung (320 x 200). Darf nur mit dem Anschluss fuer Farb-/Grafikbildschirm verwendet werden.
- **2** Grafischer Modus fuer hohe Aufloesung (640 x 200). Darf nur mit dem Anschluss fuer Farb-/Grafikbildschirm benutzt werden.

Aendern Numerischer Ausdruck, dessen Ergebnis 0 oder 1 ist. Er aktiviert oder inaktiviert die Farbe. Da "Farbe aendern" bei dem verwendeten Bildschirm immer aktiviert ist, hat dieser Parameter keinen Einfluss im Programm.

Aseite (aktive Seite): Ganzzahliger Ausdruck im Bereich 0 bis 7 fuer die Breite 40 oder 0 bis 3 fuer die Breite 80. Damit wird die Seite ausgewaehlt, die mit Hilfe von Ausgabeanweisungen auf dem Bildschirm angezeigt werden soll, und ist nur im Textmodus (Modus=0) gueltig.

Vseite (visuelle Seite): Auswahl der Seite, die auf dem Bildschirm angezeigt werden soll, in der gleichen Weise wie **Aseite**. Die visuelle Seite kann von der aktiven Seite verschieden sein. **Vseite** ist nur im Textmodus (Modus=0) gueltig. Wird der Parameter weggelassen, ist **Vseite** Standardwert fuer **Aseite**.

Sind alle Parameter gueltig, wird der neue Bildschirmmodus gespeichert, der Bildschirm geloescht, die Vordergrundfarbe auf weiss und der Hintergrund und die Bildrandfarben auf schwarz gesetzt.

Ist der neue Bildschirmmodus der gleiche wie der vorherige, wird nichts veraendert.

Ist der Modus Text und sind nur **Aseite** und **Vseite** angegeben, werden die Bildschirmseiten zum Ansehen abgeaendert. Am Anfang ist der Standardwert fuer die aktiven und visuellen Seiten 0 (Null). Durch Behandeln der aktiven und visuellen Seiten kann man eine Seite anzeigen, waehrend die andere aufgebaut wird. Damit kann man visuelle Seiten umschalten.

Hinweis:

Es gibt nur einen gemeinsamen Kursor fuer alle Seiten. Wird bei aktiven Seiten auf vorherige und nachfolgende umgeschaltet, muss die Stelle des Kursors der laufenden aktiven Seite gespeichert werden (mit Hilfe von **POS(0)** und **CSRLIN**), bevor eine andere aktive Seite angezeigt wird.
Kehrt man jetzt zur Originalseite zurueck, kann man die Stelle des Kursors mit Hilfe der Anweisung **LOCATE** wieder herstellen.
Jeder Parameter kann weggelassen werden. Fuer weggelassene Werte, ausser **Vseite**, wird der alte Wert angenommen.

Alle Werte, die sich ausserhalb des angegebenen Bereichs befinden, ergeben den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Die alten Werte bleiben erhalten.

Es ist ratsam, die Anweisungen **SCREEN 0,,0** und **WIDTH 40** an den Anfang eines Programms zu stellen.

Beispiele: **SCREEN 0**

Waehlt Textmodus aus.

SCREEN 0,,1,1

Modus bleibt erhalten. Die aktive Seite und die Anzeigeseite werden auf 1 gesetzt.

SCREEN 1

Schaltet in den grafischen Modus fuer mittlere Aufloesung um.

SCREEN 2

Schaltet in grafischen Modus fuer hohe Aufloesung um.

=_LOCATE

Setzen des Kursors auf dem aktiven Bildschirm. Wahlfreie Parameter schalten das Blinken des Kursors an und aus und definieren die Groesse des blinkenden Kursors.

Syntax: **LOCATE [Zeile][,[Spalte] [,[Anzeige,][,[Start][,Stopp]]]]**

Bemerkungen:

Zeile Numerischer Ausdruck im Bereich 1 bis 25. Er gibt die Bildschirmzeilennummer an, an die der Kursor gebracht werden soll.

Spalte Numerischer Ausdruck im Bereich 1 bis 40 oder 1 bis 80, abhaengig von der Bildschirmbreite. Er gibt die Bildschirmspaltennummer an, an die der Kursor gebracht werden soll.

Anzeige Wert, der angibt, ob der Kursor sichtbar sein soll. Eine 0 zeigt Aus an, eine 1 zeigt Ein an.

Start Beginnende Suchzeile fuer den Kursor. Es muss ein numerischer Ausdruck im Bereich 0 bis 31 sein.

Stopp Stoppsuchzeile fuer den Kursor. Es muss ein numerischer Ausdruck im Bereich 0 bis 31 sein.

Anzeige, Start und **Stopp** koennen nicht im grafischen Modus angewendet werden.

Mit **Start** und **Stopp** kann der Kursor jede gewuenschte Groesse annehmen. Man gibt damit die beginnenden und endenden Suchzeilen an. Die Suchzeilen werden ab 0 (Null) ueber der Zeichenposition numeriert. Die unterste Suchzeile ist 7 fuer den Farb-/Grafikbildschirmanschluss und 13 fuer den Schwarz/Weiss-Bildschirmanschluss. Wird **Start** angegeben und **Stopp** weggelassen, wird fuer **Stopp** der Wert von **Start** angenommen. Ist **Start** groesser als **Stopp**, erhaelt man einen zweiteiligen Kursor. Der Kursor geht von der unteren Zeile zurueck zur oberen.

Nach der Anweisung **LOCATE** werden Zeichen in Ein-/Ausgabeanweisungen fuer den Bildschirm an die angegebene Stelle gebracht.

Laeuft ein Programm ab, ist der Kursor normalerweise ausgeschaltet - mit **LOCATE,,1** kann er eingeschalten werden.

Normalerweise schreibt **BASIC** nicht in die Zeile 25. Man kann jedoch die Anzeige der Funktionstasten mit Hilfe von **KEY OFF** ausschalten und dann **LOCATE 25,1** angeben, um in Zeile 25 zu schreiben.

Jeder Parameter kann weggelassen werden. Fuer weggelassene Parameter wird der laufende Wert uebernommen.
Jeder Wert, der sich nicht innerhalb der angegebenen Bereiche befindet, ergibt den Fehler "Illegal Function Call" (Unguel-tiger Funktionsaufruf). Die vorherigen Werte bleiben dann erhalten.

Beispiele: **LOCATE 1,1**
Bringt den Kursor in die obere linke Ecke des Bildschirms.

LOCATE 10,10,,6,7
Setzt den Kursor auf Zeile 10, Spalte 10. Anzei-ge des Kursors unter den Zeichen auf dem Farb-/ Grafikbildschirm (Start in Suchzeile 6, Stopp in Suchzeile 7).

=_CSRLIN

Uebergeben der vertikalen Koordinate des Kursors.

Syntax: v=CSRLIN

Bemerkungen:

Die Variable **CSRLIN** uebergibt die laufende Zeile des Kursors auf der aktiven Seite. Der uebergebene Wert liegt im Bereich 1 bis 25.

Die Funktion **POS** uebergibt die Spalte, an der der Kursor steht (siehe POS).

Beispiel: siehe unter Funktion **POS**

=_POS

Uebergibt die Spaltenposition, in der sich gerade der Kursor befindet.

Syntax: v = POS(n)

Bemerkungen:

n ist ein Scheinargument.

Es wird die laufende Spalte des Kursors uebergeben. Der uebergebene Wert befindet sich im Bereich 1 bis 40 oder 1 bis 80, abhaengig vom Setzen von **WIDTH** (Breite).

Mit **CSRLIN** kann die Zeile des Kursors gefunden werden (siehe **CSRLIN**).
Siehe auch **"Funktion LPOS"**.

Beispiel:

```
10 X=CSRLIN     'Retten der laufenden Zeile
20 Y=POS(0)     'Retten der laufenden Spalte
25 LOCATE 23,1
30 INPUT NR$
40 LOCATE X,Y   'Kursor auf alte Zeile und Spalte setzen
```

In diesem Beispiel wird die Zeile und Spalte des Kursors in den Variablen X und Y gespeichert. Anschliessend wird der Kursor auf die Zeile 23 positioniert, eine Eingabe realisiert und danach wird der Kursor wieder auf seine alte Position gebracht.

8.3. Farbe

Die **COLOR**-Anweisung setzt die Farben fuer den Vordergrund, Hintergrund und den Bildschirmrand.

Die Syntax der Anweisung **COLOR** haengt davon ab, ob sich das System im Textmodus oder Grafikmodus befindet (durch die Anweisung **SCREEN** gesetzt).

Im Textmodus kann folgendes gesetzt werden:

```
Vordergrund     - 1 von 16 Farben, Zeichen blinken
Hintergrund     - 1 von 8 Farben
Bildschirmrand - 1 von 16 Farben
```

Das Folgende kann in mittlerer Aufloesung fuer den Grafikmodus gesetzt werden:

```
Hintergrund     - 1 von 16 Farben
Palette         - 1 von 2 Paletten mit je 3 Farben
```

Der Bildschirmrand hat dieselbe Farbe wie der Hintergrund.

8.3.1. Farbe im Textmodus - COLOR

Syntax: **COLOR [Vordergrund][,[Hintergrund] [,Bildschirmrand]]**

Bemerkungen:

Vordergrund Numerischer Ausdruck im Bereich 0 bis 31, der die Farbe der Zeichen angibt.

Hintergrund Numerischer Ausdruck im Bereich 0 bis 7 fuer die Hintergrundfarbe.

Bildschirmrand Numerischer Ausdruck im Bereich 0 bis 15. Er gibt die Farbe fuer den Bildschirmrand an.

Mit dem Farb-/Grafikbildschirmanschluss sind die folgenden Farben fuer den Vordergrund erlaubt:

0	Schwarz	8	Grau
1	Blau	9	Hellblau
2	Gruen	10	Hellgruen
3	Kobaltblau	11	Hellkobaltblau
4	Rot	12	Hellrot
5	Violett	13	Hellviolett
6	Braun	14	Gelb
7	Weiss	15	Hochintensives Weiss

Man kann sich die Farben 8 bis 15 als helle oder hochintensive Werte der Farben 0 bis 7 vorstellen.
Es ist moeglich, die Zeichen blinken zu lassen, indem man den Vordergrund gleich 16 plus der Zahl der gewuenschten Farbe setzt. Das bedeutet, ein Wert von 16 bis 31 erzeugt blinkende Zeichen.

Fuer den Hintergrund koennen nur die Farben 0 bis 7 gewaehlt werden.

Ist der Schwarz/Weiss-Bildschirmanschluss vorhanden, koennen die folgenden Farben fuer den Vordergrund gewaehlt werden:

0	Schwarz
1	Unterstrichenes Zeichen mit weissem Vordergrund
2 bis 7	Weiss

Aehnlich wie beim Farb-/Grafikbildschirmanschluss kann hier durch Addition der Zahl 8 zur gewuenschten Farbe hochintensiv angezeigt werden. Der Wert 15 z.B. ergibt Weiss hochintensiv, 9 ergibt Weiss hochintensiv und unterstrichen. Es gibt kein hochintensives Schwarz.

Wie beim Farb-/Grafikbildschirmanschluss kann man die Zeichen blinken lassen, indem man 16 zur gewuenschten Farbe addiert. 16 ergibt schwarz blinkende Zeichen, 31 ergibt hochintensive weiss blinkende Zeichen.

Fuer den Hintergrund des Schwarz/Weiss-Bildschirms koennen die folgenden Werte gewaehlt werden:

0 bis 6	Schwarz
7	Weiss

Hinweis:

Weiss (Farbe 7) als Hintergrundfarbe zeigt sich beim Schwarz/Weiss-Bildschirm nur als weiss, wenn es zusammen mit den Vordergrundfarben 0, 8, 16 oder 24 (schwarz) benutzt wird. Dies erzeugt Zeichen in Umkehrabbildung.

Schwarz (Farbe 0, 8, 16 oder 24) als Vordergrundfarbe zeigt sich nur als schwarz, wenn es zusammen mit der Hintergrundfarbe 0 benutzt wird (welches die Zeichen unsichtbar macht) oder 7 (welches die Zeichen in Umkehranzeige darstellt).

Andere Kombinationen fuer Vordergrund- und Hintergrundfarben ergeben die Standardanzeige (weiss auf schwarz auf dem Schwarz/Weiss-Bildschirm).

Hinweis fuer jeden Anschluss:

1. Die Vordergrundfarbe kann gleich der Hintergrundfarbe sein. Durch diesen Effekt werden Zeichen unsichtbar gemacht. Das Aendern der Vordergrund- oder Hintergrundfarbe laesst die Zeichen wieder sichtbar werden.

2. Parameter koennen weggelassen werden. Fuer weggelassene Parameter wird der alte Wert angenommen.

3. Endet die Anweisung **COLOR** mit einem Komma (,), erhaelt man die Fehlernachricht "Missing Operand" (Fehlender Operand), aber die Farbe wird veraendert.

   Beispiel:

   ```
   COLOR ,7,
   ```

   ist ungueltig.

4. Durch jeden eingegebenen Wert ausserhalb des Bereichs 0 bis 255 erhaelt man den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Fruehere Werte werden beibehalten.

   Beispiel:

   ```
   10 COLOR 4,2,0
   ```

   Dadurch wird der Vordergrund auf rot, der Hintergrund auf gruen und der Bildschirmrand auf schwarz gesetzt.

Beispiel:

```
10 PRINT "EINGABE "
20 COLOR 15,0          'Intensivanzeige,
                        schwarzer Hintergrund
30 PRINT "Name,Code"
40 COLOR 7             'weisses Zeichen
50 INPUT Name $
60 COLOR 0,7           'Umkehrung Anzeige
                        (schwarz auf weiss)
70 INPUT Code %
```

8.3.2. Farbe im Grafikmodus

Syntax: COLOR [Hintergrund][,[Palette]]

Bemerkungen:

Hintergrund Numerischer Ausdruck, der die Hintergrundfarbe angibt. Die fuer **Hintergrund** erlaubten Farben sind 0 bis 15, wie im Abschnitt 8.3.1. beschrieben (Farbe 8 - 15 sind hier die hochintensiven Farben der Farben 0 - 7).

Palette Numerischer Ausdruck, der die Palette der Farben auswaehlt.

Die fuer jede Palette auszuwaehlenden Farben sind die folgenden:

Farbe	**Palette 0**	**Palette 1**
1	Gruen	Kobaltblau
2	Rot	Violett
3	Braun	Weiss

Ist **Palette** eine gerade Zahl, wird Palette 0 ausgewaehlt. Dadurch werden die Farben Gruen, Rot und Braun den Farbnummern 1, 2 und 3 zugeordnet. Palette 1 wird ausgewaehlt (Kobaltblau/Violett/Weiss), wenn **Palette** eine ungerade Zahl ist.

Die fuer den Hintergrund ausgewaehlte Farbe kann die gleiche sein, wie eine der Palettenfarben.

Parameter der Anweisung **COLOR** koennen weggelassen werden. Fuer weggelassene Parameter wird der alte Wert angenommen.

Im Grafikmodus setzt die Anweisung **COLOR** die Hintergrundfarbe und eine Palette aus drei Farben. Jede dieser vier Farben kann fuer die Anzeige mit den Anweisungen **PSET, PRESET, LINE, CIRCLE, PAINT** und **DRAW** ausgewaehlt werden. Dies gilt nur fuer die mittlere Aufloesung (gesetzt durch die Anweisung **SCREEN 1**).
Wird **COLOR** fuer hohe Aufloesung benutzt, erhaelt man den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Jeder eingegebene Wert ausserhalb des Bereichs 0 bis 255 erzeugt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Die frueheren Werte werden beibehalten.

Beispiel:

```
10 SCREEN 1          Setzt den Hintergrund auf violett
20 COLOR 5,1         und waehlt die Palette 1 aus.

20 COLOR ,0          Der Hintergrund bleibt violett
                     und die Palette 0 wird ausgewaehlt.

20 COLOR 9           Der Hintergrund wird auf hellblau
                     gesetzt, Palette 0 bleibt.
```

8.4. Grafik

Alle unter diesem Punkt angefuehrten Anweisungen und Funktionen sind **nur** im Grafikmodus zu verwenden.

8.4.1. Anweisungen

- PSET und PRESET

Zeichnen eines Punktes an eine angegebene Stelle auf dem Bildschirm.

Syntax: **PSET** (x,y)[,**Farbe**]
PRESET (x,y)[,**Farbe**]

Bemerkungen:

(x,y) sind die Koordinaten des zu setzenden Punktes. Sie koennen in absoluter oder relativer Form angegeben werden (sh. Pkt. 8.1.).

Farbe Ist ein ganzzahliger Ausdruck und definiert die zu benutzende Farbe im Bereich 0 bis 3. In der mittleren Aufloesung waehlt **Farbe** die Farbe der laufenden Palette aus, wie mit der Anweisung **COLOR** definiert. 0 (Null) ist die Hintergrundfarbe.
Die Standardfarbe fuer den Vordergrund ist die Farbe Nummer 3. In der hohen Aufloesung kennzeichnet **Farbe** gleich 0 schwarz, die Standardannahme von 1 kennzeichnet weiss. In der hohen Aufloesung wird ein Farbwert von 2 als 0 (Null), 3 als 1 behandelt.

PRESET ist fast identisch mit **PSET**. Der einzige Unterschied ist der, dass - falls fuer **PRESET** nicht der Parameter **Farbe** angegeben wird - mit der Hintergrundfarbe (0) gezeichnet wird (Loeschen des Punktes). Ist **Farbe** angegeben, ist **PRESET** gleich **PSET**.

Wird fuer **PSET** oder **PRESET** eine Koordinate angegeben, die sich nicht im Bereich befindet, wird nichts ausgefuehrt und auch kein Fehler angezeigt. Ist **Farbe** groesser als 3, wird der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf) angezeigt.

Beispiel:

Zeichnen einer Linie mit den Zeilen 30 - 50 und anschliessendes Loeschen dieser Linie in den Zeilen 60 - 80.

```
10 CLS
20 SCREEN 1
30 FOR X=1 TO 50
40 PSET(X,X)
50 NEXT X
60 FOR X=1 TO 50
70 PRESET(X,X)
80 NEXT X
```

LINE

Zeichnen einer Linie oder eines Viereckes auf dem Bildschirm.

Syntax: **LINE** [(x1,y1)]-(x2,y2)[,[Farbe][,B[F]][,Stil]]

Bemerkungen:

(x1,y1),(x2,y2) sind Koordinaten in absoluter oder relativer Form.

Farbe Ganzzahliger Ausdruck und definiert die Farbnummer im Bereich 0 bis 3. In der mittleren Aufloesung waehlt **Farbe** die Farbe aus der laufenden Palette aus, wie in der Anweisung **COLOR** definiert. 0 ist die Hintergrundfarbe. Die Standardannahme fuer die Vordergrundfarbe ist die Farbe Nummer 3. In der hohen Aufloesung zeigt **Farbe** gleich 0 schwarz und die Standardannahme 1 weiss an.

Stil Ganzzahlige 16-Bit-Maske, die benutzt wird, um Punkte auf dem Bildschirm darzustellen. Die Auswahl kann fuer Linien und Quadrate verwendet werden, aber nicht bei gefuellten Quadraten.
LINE benutzt das laufende Bit in **Stil**, um Punkte auf dem Bildschirm zu zeichnen. Ist das Bit 0 (Null), wird kein Punkt gezeichnet. Ist das Bit 1 (eins), so wird ein Punkt gezeichnet. Nach jedem Punkt wird die naechste Bit-Position in Stil ausgewaehlt. Ist die letzte Bit-Position in **Stil** ausgewaehlt, so beginnt **LINE** wieder mit der ersten Position. Dadurch koennen verschiedene Linien gezeichnet werden.

B Zeichnen eines Rechteckes mit den Punkten (x1,y1) und (x2,y2) als gegenueberliegende Ecken.

BF Es wird das gleiche Rechteck wie mit **B** gezeichnet; zusaetzlich wird der Inhalt des Rechtecks mit der ausgewaehlten Farbe ausgefuellt.

Koordinaten, die sich ausserhalb des Bereiches befinden, werden abgeschnitten.
Der letzte angesprochene Punkt nach einer Anweisung **LINE** ist der Punkt (x2,y2).
Wird fuer die zweite Koordinate die relative Form gewaehlt, ist sie relativ zur ersten Koordinate.

Die einfachste Form von **LINE** ist:

```
LINE -(X2,Y2)
```

Damit wird eine Linie vom letzten angesprochenen Punkt zum Punkt **(X2,Y2)** in der Vordergrundfarbe gezeichnet.

Beispiele:

```
LINE (100,150)-(200,150),2
```

Zeichnen einer Linie in der Farbe 2 der ausgewaehlten Palette.

```
LINE (100,50)-(150,100),1,BF
```

Es wird ein Rechteck gezeichnet und dieses wird mit der Farbe 1 der ausgewaehlten Palette gefuellt.

```
LINE (100,150)-STEP(30,-50)
```

Zeichnen einer Linie von (100,150) bis (130,100).

```
LINE (0,0)-(319,199),,,&HAAAA
```

Zeichnen einer gepunkteten Linie diagonal ueber den Bildschirm. Das Bitmuster dieser Linie sieht wie folgt aus:

```
1010101010101010
```

und das ist gleichbedeutend mit &HAAAA in hexadezimaler Darstellung.

=_CIRCLE

Zeichnen einer Ellipse auf den Bildschirm mit dem Mittelpunkt (x,y) und dem Radius r.

Syntax: **CIRCLE (x,y),r[,Farbe[,Start,End[,Bild]]]**

Bemerkungen:

(x,y) Sind die Koordinaten des Mittelpunkts der Ellipse. Die Koordinaten koennen entweder in absoluter oder relativer Form angegeben werden.

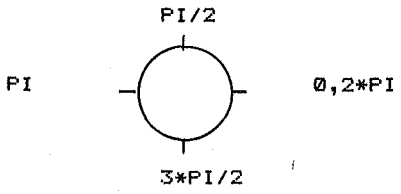
r Ist der Radius (Hauptachse) der Ellipse in Punkten.

Farbe Ist ein ganzzahliger Ausdruck, der die Farbe der Ellipse angibt und liegt im Bereich Ø bis 3. **Farbe** waehlt die Farbe aus der laufenden Palette aus, die in der Anweisung **COLOR** definiert ist. Ø ist die Hintergrundfarbe. Die Standardannahme fuer die Vordergrundfarbe ist die Farbe Nummer 3. In der hohen Aufloesung zeigt Ø (Null) fuer **Farbe** schwarz an und die Standardannahme 1 (eins) weiss.

Start,End Sind Winkel im Bogenmass und koennen im Bereich -2*PI bis 2*PI liegen, wobei PI=3.141593 ist.

Bild Ist ein numerischer Ausdruck.

Start und **End** Geben an, wo das Zeichnen der Ellipse beginnen und enden soll. Die Winkel werden auf dem normalen mathematischen Weg positioniert, d.h. mit Ø rechts und entgegen dem Uhrzeiger:



Ist der Start- oder Endwinkel negativ (-0 ist nicht erlaubt), wird die Ellipse mit einer Linie mit dem Mittelpunkt verbunden, und die Winkel werden behandelt, als seien sie positiv. Der Anfangswinkel kann kleiner oder groesser sein als der Endwinkel.

Beispiel:

```
CIRCLE (100,100),50,1,-PI/2,-PI
```

Dieses Beispiel zeichnet den Teil eines Kreises, aehnlich dem folgenden:

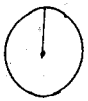
Bild hat Einfluss auf das Verhaeltnis des x-Radius zum y-Radius. Die Standardannahme fuer **Bild** ist 5/6 in der mittleren Aufloesung und 5/12 in der hohen Aufloesung. Diese Werte geben visuell einen Kreis in der Annahme, dass das Bildverhaeltnis des Standardbildschirms 4/3 ist.

Ist **Bild** kleiner als 1, so ist r der x-Radius. Das heisst, der Radius wird in Punkten in der horizontalen Richtung gemessen. Ist **Bild** groesser als 1, so ist r der y-Radius.

Beispiel:

```
CIRCLE (100,100),50,,,,10/4
```

Dieses Beispiel zeichnet eine Ellipse wie folgt:



In vielen Faellen gibt **Bild** gleich 1 (eins) einen besser aussehenden Kreis in der mittleren Aufloesung. Damit wird der Kreis auch schneller gezeichnet.

Der letzte angesprochene Punkt, nachdem ein Kreis gezeichnet ist, ist der Mittelpunkt des Kreises.

Punkte, die sich ausserhalb des Bildschirms befinden, werden mit **CIRCLE** nicht gezeichnet.

=_DRAW

Zeichnen eines Objekts, das durch eine Zeichenkette angegeben ist.

Syntax: **DRAW Zeichenkette**

Bemerkungen:

Die Anweisung **DRAW** zeichnet mit Hilfe einer **Grafik-Definitionssprache.** Die Sprachbefehle befinden sich in einem **Zeichenkettenausdruck.** Die Zeichenkette definiert ein Objekt, das gezeichnet wird, wenn **BASIC** die Anweisung **DRAW** ausfuehrt. Waehrend der Ausfuehrung prueft **BASIC** den Wert der Zeichenkette und interpretiert die einzelnen Buchstabenbefehle aus dem Inhalt der **Zeichenkette.** Die Befehle werden unten ausfuehrlich erklaert.

Die folgenden Bewegungsbefehle beginnen ihre Bewegung am zuletzt angesprochenen Punkt. Nach jedem Befehl ist der zuletzt angesprochene Punkt der letzte Punkt, den der Befehl zeichnet.

U n	Nach oben
D n	Nach unten
L n	Nach links
R n	Nach rechts
E n	Diagonal nach oben und rechts
F n	Diagonal nach unten und rechts
G n	Diagonal nach unten und links
H n	Diagonal nach oben und links
n	in jedem der vorhergehenden Befehle zeigt die Laenge der Bewegung an. Die Anzahl der Punkte, um die sich bewegt wird, ist **n** mal dem Skalenfaktor (gesetzt durch den Befehl **S**).
M x,y	Bewegen absolut oder relativ. Hat **x** ein Pluszeichen (+) oder ein Minuszeichen (-) vor seinem Wert, ist es relativ; andernfalls absolut.

Die folgenden zwei Befehle koennen jedem der oben erwaehnten Bewegungsbefehle vorangestellt werden.

B	Bewegen, aber keine Punkte zeichnen.
N	Bewegen, aber nach der Ausfuehrung auf die alte Position zurueckkehren.

Die folgenden Befehle sind ebenfalls verfuegbar:

A n Setzt den Winkel n. n kann im Bereich 0 bis 3 liegen, wobei 0 = 0, 1 = 90, 2 = 180 und 3 = 270 Grad bedeuten. Figuren, die um 90 oder 270 Grad gedreht werden, sind so skaliert, dass sie in der gleichen Groesse auf dem Bildschirm erscheinen wie mit 0 oder 180 Grad im Standardbildschirmverhaeltnis 4/3.

TA n Dreht den Winkel n. n kann von -360 bis +360 gehen. Ist n positiv (+), dreht sich der Winkel gegen den Uhrzeigersinn. Ist n negativ (-), dreht sich der Winkel im Uhrzeigersinn. Eingegebene Werte, die sich nicht im Bereich -360 bis +360 befinden, ergeben den Fehler "Illegal function call".

C n Setzt die Farbe n. n darf sich fuer mittlere Aufloesung im Bereich 0 bis 3 befinden, fuer hohe Aufloesung im Bereich 0 bis 1. Fuer die mittlere Aufloesung waehlt n die Farbe aus der laufenden Palette aus, die mit der Anweisung **COLOR** definiert wurde. 0 ist die Hintergrundfarbe. Der Standardwert fuer die Vordergrundfarbe ist die Farbe Nummer 3. Fuer die hohe Aufloesung ist n gleich 0 schwarz, die Standardannahme 1 (eins) bedeutet weiss.

S n Setzt den Skalenfaktor. n darf sich im Bereich 1 bis 255 befinden. n dividiert durch 4 ist der Skalenfaktor. Ist z.B. n gleich 1, dann ist der Skalenfaktor 1/4. Der Skalenfaktor, multipliziert mit den Abstaenden, die mit den Befehlen **U, D, L, R, E, F, G, H** und dem relativen Befehl **M** gegeben werden, ergibt die Laenge, um die bewegt werden soll.
Der Standardwert fuer n ist 4, so dass der Skalenfaktor 1 ist.

X Variable$
Fuehrt den Teil einer Zeichenkette aus. Dies erlaubt die Ausfuehrung einer zweiten Zeichenkette innerhalb einer Zeichenkette.

P Farbe, Rand
Setzt die Farbe der Abbildung auf **Farbe**, und den Rand der Abbildung auf **Rand.**
Farbe ist ein ganzzahliger Ausdruck und waehlt eine Farbe (0...3) aus der laufenden Palette aus. Der Parameter fuer **Rand** ist die Randfarbe der Abbildung, die ausgemalt werden soll.
Er kann aus den fuer den laufenden Bildschirmmodus zur Verfuegung stehenden Attributen gewaehlt werden. Es muessen beide Parameter angegeben werden, da sonst ein Fehler auftritt.

In all diesen Befehlen koennen die Argumente n, x, y eine Konstante, wie z.B. 123 oder auch **=Variable;** sein, wobei **Variable** der Name einer numerischen Variablen ist. Das Semikolon (;) wird benoetigt, wenn die Variable auf diese Weise oder in einem Befehl X benutzt wird. Sonst ist ein Semikolon zwischen Befehlen wahlfrei. Leerstellen in der **Zeichenkette** werden ignoriert. Man kann z.B. Variable in einer Bewegungsanweisung auf die folgende Weise verwenden:

```
DRAW "E15"   oder
A = 15 :DRAW "E"="+VARPTR$(A)
```

Man kann die Variablen auch in der Form **VARPTR$(Variable)** anstelle von **=Variable;** definieren. Dies ist fuer die Programme notwendig, die spaeter compiliert werden sollen.

Beispiel:

```
DRAW "XA$;"              kann als
DRAW "X"+VARPTR$(A$)     geschrieben werden.
```

Der Befehl **X** kann ein sehr nuetzlicher Befehl als Teil von **DRAW** sein, weil man damit Teile eines Objekts separat vom ganzen Objekt definieren kann.
Man kann mit X auch Zeichenketten erzeugen, die laenger als 255 Zeichen lang sind.

Das Abbildungsverhaeltnis auf dem Bildschirm bestimmt den Abstand der horizontalen, vertikalen und diagonalen Punkte auf dem Bildschirm.

In der Anweisung **DRAW** wird das Abbildungsverhaeltnis des laufenden Bildschirmmodus nicht beruecksichtigt. **DRAW L50 U50** zeichnet genau 50 Punkte horizontal und 50 Punkte vertikal. Diese beiden Linien haben jedoch nicht die gleiche Laenge.

In **BASIC** werden zwei Bildschirmaufloesungen unterstuetzt mit den entsprechenden Abbildungsverhaeltnissen:

```
Mittlere Aufloesung 320 x 200 Punkte
                              Abbildungsverhaeltnis 5/6

Hohe Aufloesung     640 x 200 Punkte
                              Abbildungsverhaeltnis 5/12
```

Um ein Quadrat zu zeichnen, ist der Wert fuer die Y-Achse mit dem entsprechenden Abbildungsverhaeltnis zu multiplizieren oder der Wert fuer die X-Achse mit 1/Abbildungsverhaeltnis zu multiplizieren.

Die Anweisung

```
DRAW "R120 D100 L120 U100"
```

zeichnet z.B. ein Quadrat mit einer Hoehe 100.

Beispiel:

Zeichnen eines Dreiecks:

```
DRAW "E20 F20 L40"
```

Zeichnen eines Parallelogramms:

```
10 SCREEN 1
20 COLOR 3,0
30 PSET (100,50),2
40 DRAW "C2 R100 G50 L100 E50"
50 PAINT (150,75),2
```

=_GET

Lesen von Punkten aus einem Bereich des Bildschirms.

Syntax: **GET (x1,y1)-(x2,y2),Bereich**

Bemerkungen:

(x1,y1),(x2,y2) Sind Koordinaten in absoluter oder relativer Form.

Bereich Name des Bereichs, in dem die Information gespeichert werden soll.

GET liest die Farben der Punkte innerhalb des angegebenen Vierecks in einen Bereich. Das angegebene Viereck hat die Punkte **(x1,y1)** und **(x2,y2)** als gegenueberliegende Ecken (dies ist das gleiche Viereck, das unter der Anweisung **LINE** mit der Auswahl **B** gezeichnet wurde).

Mit **GET** und **PUT** koennen Objekte im Grafikmodus mit hoher Geschwindigkeit bewegt werden. Man kann sich die Operationen **GET** und **PUT** so vorstellen, dass sie die Bits sehr schnell auf **(PUT)** den Bildschirm bringen und vom **(GET)** Bildschirm nehmen. **PUT** und **GET** werden auch benutzt, um Dateien mit wahlfreiem Zugriff anzusprechen; die Syntax dieser Anweisungen ist jedoch verschieden.
Der Bereich wird dazu benutzt, das Abbild zu speichern, und muss numerisch sein. Jedoch ist jede Genauigkeit erlaubt. Die benoetigte Groesse des Bereichs in Bytes ist:

```
4+INT((x*Bit-Anzahl+7)/8)*y
```

wobei x und y die Laengen der horizontalen und vertikalen Seiten des Rechtecks sind. Der Wert fuer die **Bit-Anzahl** ist 2 fuer die mittlere Aufloesung und 1 fuer die hohe Aufloesung.

Soll die Anweisung **GET** dazu benutzt werden, ein 10 x 12-Abbild mittlerer Aufloesung zu bekommen, ist die Anzahl der benoetigten Bytes **4+INT((10*2+7)/8)*12** oder 40 Bytes.

Die Anzahl Bytes pro Element eines Bereichs sind:

- 2 fuer ganzzahlig
- 4 fuer einfache Genauigkeit
- 8 fuer doppelte Genauigkeit

Deshalb reicht ein ganzzahliger Bereich mit mindestens 20 Elementen aus.

Die Information vom Bildschirm wird im Bereich wie folgt gespeichert:

Zwei Bytes fuer die x-Dimension in Bits
Zwei Bytes fuer die y-Dimension in Bits
Daten

Es ist moeglich, die x- und y-Dimensionen und selbst die Daten zu pruefen, wenn ein ganzzahliger Bereich benutzt wurde. Die x-Dimension steht im Element 0 des Bereichs, die y-Dimension steht im Element 1. Man muss jedoch wissen, dass ganze Zahlen so gespeichert werden, dass zuerst das Byte mit niedrigem Stellenwert, danach das Byte mit hohem Stellenwert gespeichert wird.

Die Daten fuer jede Punktzeile des Rechtecks werden linksbuendig an einer Byte-Grenze gespeichert, so dass, falls weniger als ein Vielfaches von acht Bits gespeichert wird, der Rest des Bytes mit Nullen aufgefuellt wird.

PUT und **GET** arbeiten wesentlich schneller, wenn in mittlerer Aufloesung **x1 MOD 4** gleich Null ist, und in hoher Aufloesung, wenn **x1 MOD 8** gleich Null ist.
In diesem Spezialfall fallen die Rechteckgrenzen mit den Byte-Grenzen zusammen.

Beispiel:

siehe Anweisung **PUT**

PUT

Faerben eines angegebenen Bereiches des Bildschirms.

Syntax: **PUT (x,y),Bereich[,Aktion]**

Bemerkungen:

(x,y) Sind die Koordinaten der linken oberen Ecke des zu uebertragenden Bildes.

Bereich Name eines numerischen Bereichs, der die zu uebertragende Information enthaelt (sh. auch Anweisung **GET** in diesem Kapitel).

Aktion
ist eine der folgenden:

PSET
PRESET
XOR
OR
AND

Standardannahme ist **XOR**.

PUT ist das Gegenteil von **GET** - in dem Sinn, dass es Daten aus einem Bereich herausnimmt und auf dem Bildschirm anzeigt. Jedoch kann man Daten, die schon auf dem Bildschirm stehen, mit Hilfe von **Aktion** manipulieren.

PSET speichert einfach die Daten aus dem Bereich auf den Bildschirm, so dass dies das genaue Gegenteil von **GET** bewirkt.

PRESET ist das gleiche wie **PSET** - nur dass ein negatives Bild produziert wird. Das bedeutet, dass die Farbe Ø (Null) im Bereich bewirkt, dass der dazugehoerige Punkt die Farbe Nummer 3 bekommt und umgekehrt. Die Farbe 1 im Bereich bewirkt, dass der korrespondierdene Punkt die Farbe Nummer 2 bekommt und umgekehrt (bei mittlerer Aufloesung).

AND, OR und **XOR** geben die logischen Bit-Operationen fuer jedes Abbild an.

AND wird benutzt, wenn ein Abbild nur uebertragen werden soll, falls das Abbild schon unter dem uebertragenen Abbild existiert.

OR wird benutzt, um ein Abbild ueber ein schon existierendes Abbild zu legen.

XOR ist ein Spezialmodus, der fuer Bewegungen benutzt werden kann. Durch **XOR** werden Punkte auf dem Bildschirm umgedreht, wenn der Punkt im Bereichsabbild existiert. Wird das Abbild durch die Anweisung **PUT** gegen einen komplexen Hintergrund zweimal angewendet, wird der Hintergrund ohne Veraenderung neu dargestellt. Dadurch kann ein Objekt bewegt werden, ohne dass der Hintergrund geloescht wird.

In der mittleren Aufloesung haben **AND**, XOR und OR den folgenden Einfluss auf die Farbe:

AND

Bildschirm	Bereichswert 0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

OR

Bildschirm	Bereichswert 0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

XOR

Bildschirm	Bereichswert 0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Die Bewegung eines Objekts kann wie folgt durchgefuehrt werden:

1. Das Objekt mit **PUT** auf den Bildschirm bringen (mit **XOR**)

2. Die neue Position des Objekts berechnen

3. Das Objekt durch **PUT** (mit **XOR**) ein zweites Mal an die alte Stelle auf den Bildschirm bringen, um das alte Bild zu entfernen

4. Zu 1. zurueckgehen und diesmal das Objekt an den neuen Platz stellen

Eine solche Bewegung laesst den Hintergrund unveraendert. Das Flackern des Bildschirms kann durch Verkuerzung der Zeit zwischen den Schritten 4 und 1 reduziert werden. Auch sollte zwischen den Schritten 1 und 3 genuegend Zeitverzoegerung bestehen. Soll mehr als ein Objekt bewegt werden, sollten alle Objekte zusammen schrittweise zur gleichen Zeit verarbeitet werden.

Falls es nicht wichtig ist, den Hintergrund zu erhalten, kann Bewegung auch durch die Aktion **PSET** erzielt werden. Man benoetigt jedoch einen Abbildungsbereich, der die Abbildungen des Objekts zuvor und danach enthaelt. Auf diese Weise loescht dieser Extrabereich das alte Abbild. Diese Methode ist etwas schneller als die Methode mit Hilfe von **XOR** wie oben beschrieben, da nur ein **PUT** benoetigt wird, um das Objekt zu bewegen (obwohl mit einem groesseren Abbild gearbeitet wird).

Ist die zu uebertragende Abbildung zu gross, um auf den Bildschirm zu passen, erscheint der Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Beispiel:

Bewegen einer Ellipse mit Hilfe von **XOR** ueber den Bildschirm

```
 10 CLS: SCREEN 1: COLOR 4,0
 20 DIM JBER(262)
 30 CIRCLE (100,100),30,2,,,5/18
 40 PAINT (100,100)3,2
 50 GET (70,80)-(130,120)JBER : CLS
 60 X = 30, Y = 20
 70 FOR J=1 TO 20
 80 PUT(X,Y),JBER,XOR
 90 PUT(X,Y),JBER,XOR
100 X = X+10 : Y = Y+5
110 NEXT J
```

PAINT

Fuellen eines Bildschirmbereichs mit einer ausgewaehlten Farbe.

Syntax: **PAINT (x,y)[,Farbe[,Rand][,Hintergrund]]**

Bemerkungen:

(x,y) Koordinaten eines Punktes innerhalb des Bereiches, der mit Farbe gefuellt werden soll. Die Koordinaten des Bereichs koennen in absoluter oder relativer Form angegeben werden. Dieser Punkt wird als Anfangspunkt benutzt.

Farbe Kann ein numerischer Ausdruck oder ein Zeichenkettenausdruck sein.
Er wird verwendet, um einen angegebenen Bereich mit Farbe oder mit einem Muster auszufuellen oder einzurahmen.
Ist **Farbe** ein numerischer Ausdruck, wird eine Farbe aus den moeglichen Farben fuer den laufenden Bildschirmmodus gewaehlt. Bei mittlerer Aufloesung ist dies eine Farbe der laufenden Palette (durch die Anweisung **COLOR** definiert). Die Farbe fuer den Hintergrund ist immer 0. Die Standardannahme fuer die Vordergrundfarbe ist die Farbe 3 fuer mittlere Aufloesung, die Farbe 1 fuer die hohe Aufloesung.
Ist **Farbe** ein Zeichenkettenausdruck, wird "Mustermalen" ausgefuehrt.

Rand Numerischer Ausdruck im Bereich 0 bis 3, er definiert die Farbe des Randes eines Bildes, dessen Inhalt ausgemalt werden soll.

Hintergrund Ist ein 1 Byte langer Zeichenkettenausdruck, der fuer das Mustermalen verwendet wird.

Da es fuer die hohe Aufloesung nur zwei Farben gibt, ergibt es keinen Sinn, wenn **Farbe** und **Rand** verschieden sind.
Man kann entweder eine weisse Flaeche schwarz oder eine schwarze Flaeche weiss ausmalen.

In der mittleren Aufloesung kann man z.B. mit der Farbe 1 auffuellen, wenn der Rand die Farbe 2 hat. Visuell ist dies eine gruene Flaeche mit einem roten Rand.
Farbe beginnt an dem angegebenen Anfangspunkt und bedeckt eine Flaeche, bis die angegebene Randfarbe erreicht ist.

Der Anfangspunkt von **PAINT** muss deshalb innerhalb der auszufuellenden Figur sein. Hat der angegebene Anfangspunkt dasselbe Attribut wie **Rand**, so wird das Ausfuellen mit diesem Punkt wieder beendet und scheinbar nicht ausgefuellt. Wird **Farbe** weggelassen, wird die Vordergrundfarbe benutzt (3 in der mittleren Aufloesung, 1 in der hohen Aufloesung). Mit **PAINT** kann jeder Figurtyp bemalt werden, aber gezackte Raender in einer Figur erhoehen die Groesse des Stapelspeichers, der von **PAINT** benoetigt wird. Soll eine komplexe Zeichnung hergestellt werden, ist es ratsam, am Anfang des Programms **CLEAR** auszufuehren, um den verfuegbaren Stapelspeicher zu vergroessern.

Die Anweisung **PAINT** erlaubt es, verschiedene Ansichten mit sehr wenigen Anweisungen darzustellen.

Beispiel:

```
10 SCREEN 1: COLOR 4,0
20 CIRCLE (100,100),50,2
30 PAINT (100,100)3,2
```

In diesem Beispiel wird der Kreis mit der Farbe 3 der Palette 0 ausgemalt. Der Rand wird mit der Farbe 2 gemalt.

Fuer das Mustermalen muss **Farbe** ein Zeichenkettenausdruck sein in der Form

CHR$(&Hnn) + CHR$(&Hnn) + CHR$(&Hnn)

Die **CHR$**-Zeichenfolge definiert eine Bitmaske, die 1 Byte breit ist. Wenn diese Maske horizontal und vertikal innerhalb der durch **Rand** festgelegten Flaeche gezeichnet wird, entsteht ein Muster und keine einzelne Farbe.
Die beiden Hexadezimalwerte in der **CHR$**-Zeichenfolge entsprechen 8 Bits oder einem Byte.
Die Zeichenkettenfolge kann bis zu 64 Byte enthalten. Das Muster, das durch den Zeichenkettenausdruck erzeugt wird, kann wie folgt abgebildet werden:

```
          x nimmt zu --->
        7 6 5 4 3 2 1 0
0,0     x x x x x x x x     Musterbyte 0
0,1     x x x x x x x x     Musterbyte 1
0,2     x x x x x x x x     Musterbyte 2
 :
 :
0,63    x x x x x x x x     Musterbyte 63
```

Das Muster wird gleichmaessig ueber die ganze durch **Rand** definierte Flaeche verteilt. Wird keine Flaeche definiert, wird die gesamte Bildschirmflaeche verwendet.
Jedes Byte der Muster-Zeichenkette entspricht 8 Bits entlang der X-Achse.
Da es in der hohen Aufloesung nur 1 Bit pro Bildschirmpunkt gibt, wird fuer jede Position der Bitmaske, die den Wert 1 hat, ein Punkt gezeichnet.

hat, ein Punkt gezeichnet.
Im folgenden Beispiel kann der Bildschirm mit dem Buchstaben **A** ausgefuellt werden (in der hohen Aufloesung):

```
10 CLS: SCREEN 2: KEY OFF
20 PAINT(100,100),CHR$(&H18)+CHR$(&H24)
        +CHR$(&H42)+CHR$(&H81)+CHR$(&HFF)
        +CHR$(&H81)+CHR$(&H81)+CHR$(&H81)
```

Das Muster erscheint folgendermassen auf dem Bildschirm:

```
                x nimmt zu --->
Musterbyte 0    0 0 0 1 1 0 0 0   CHR$(&H18)
           1    0 0 1 0 0 1 0 0   CHR$(&H24)
           2    0 1 0 0 0 0 1 0   CHR$(&H42)
           3    1 0 0 0 0 0 0 1   CHR$(&H81)
           4    1 1 1 1 1 1 1 1   CHR$(&HFF)
           5    1 0 0 0 0 0 0 1   CHR$(&H81)
           6    1 0 0 0 0 0 0 1   CHR$(&H81)
           7    1 0 0 0 0 0 0 1   CHR$(&H81)
```

Im Bildschirmmodus **SCREEN 1** beschreibt ein Musterbyte in mittlerer Aufloesung 4 Bildschirmpunkte, da in mittlerer Aufloesung 2 Bit auf einen Bildschirmpunkt kommen.
Jeweils 2 Bit des Musterbyte beschreiben 1 von 4 moeglichen Farben, die sich auf jeden der 4 Bildschirmpunkte, die abgebildet werden, beziehen.
Die folgende Tabelle zeigt die binaeren und hexadezimalen Werte, die jeder Farbe zugeordnet sind:

Palette 0	Palette 1	Farbe (binaer)	Muster fuer Linie (binaer)	(hexadezim.)
Gruen	Kobaltblau	01	01010101	&H55
Rot	violett	10	10101010	&HAA
Braun	Weiss	11	11111111	&HFF

Im folgenden Beispiel wird der Bildschirm mit **H** ausgefuellt in der Farbe Gruen bei Auswahl Palette 0:

```
10 CLS: SCREEN 1: COLOR 0,0: KEY OFF
20 PAINT(100,100),CHR$(&H41)+CHR$(&H41)
        +CHR$(&H55)+CHR$(&H41)+CHR$(&H41)
        +CHR$(&H0)
```

```
                7 6 5 4 3 2 1 0
Musterbyte 0    0 1 0 0 0 0 0 1   CHR$(&H41)
           1    0 1 0 0 0 0 0 1   CHR$(&H41)
           2    0 1 0 1 0 1 0 1   CHR$(&H55)
           3    0 1 0 0 0 0 0 1   CHR$(&H41)
           4    0 1 0 0 0 0 0 1   CHR$(&H41)
           5    0 0 0 0 0 0 0 0   CHR$(&H00)
```

Soll ueber eine bereits mit Farbe ausgefuellte Flaeche ein Muster gelegt werden, wobei die Flaeche dieselbe Farbe oder dasselbe Muster aufweist, wie zwei aufeinanderfolgende Byte in der Maske fuer das Muster, so bedeutet dies normalerweise eine Endebedingung, da der Punkt mit Punkten im selben Bit-Muster umgeben wird.

Das Attribut fuer **Hintergrund** kann verwendet werden, um diese Endebedingung zu ignorieren.

Es koennen nicht mehr als zwei aufeinanderfolgende Zeilen im Muster angegeben werden, die dem jeweiligen Hintergrund-Attribut entsprechen.

Werden mehr als zwei Zeilen angegeben, wird der Fehler "Illegal function call" (Ungueltiger Funktionsaufruf) angezeigt.

Im folgenden Beispiel wird "Mustermalen" mit dem Hintergrund-Attribut demonstriert:

```
 10 CLS: SCREEN 1: COLOR 0,1: KEY OFF
 20 MUST$=CHR$(&H5F)+CHR$(&H5F)+
        CHR$(&H27)+CHR$(&H81)
 30 VIEW(1,50)-(150,150),0,2
 40 LOCATE (1,1):PRINT "ohne Hinter-": LOCATE (2,1):
 50 PRINT "Grundzeichen"
 60 PAINT (125,50),CHR$(&H5F)
 70 PAINT (125,50),MUST$,2
 80  Mit Hintergrundzeichen
 90 VIEW (160,50)-(310,150),0,2
100 LOCATE (1,21):PRINT "Mit Hinter-":LOCATE (2,21):
110 PRINT "Grundzeichen"
120 PAINT (125,50),CHR$(&H5F)
130 PAINT (125,50),MUST$,2,CHR$(&H5F)
140 END
```

=_VIEW

Definieren eines rechteckigen Ausschnittes des Bildschirms.

Syntax: **VIEW [[SCREEN][(x1,y1)-(x2,y2)[,[Farbe][,Rand]]]]**

Bemerkungen:

(x1,y1)-(x2,y2) Linke obere und rechte untere Koordinate des definierten Ausschnitts.
Die x- und y-Koordinaten muessen sich in den aktuellen Grenzen des Bildschirms befinden, sonst erscheint die Nachricht "Illegal function call" (Ungueltiger Funktionsaufruf).

Farbe Ganzzahliger Ausdruck, der eine Farbe aus den moeglichen Farben des laufenden Bildschirmmodus auswaehlt. Bei mittlerer Aufloesung ist dies eine Farbe der laufenden Palette. Die Farbe fuer den Hintergrund ist immer 0.
Fuer den Vordergrund ist es die Farbe 3 bei mittlerer Ausfloesung und die Farbe 1 bei hoher Aufloesung.
Farbe ermoeglicht das Ausfuellen eines definierten Ausschnittes mit einer Farbe. Wird **Farbe** weggelassen, wird der Ausschnitt nicht ausgefuellt.

Rand Ganzzahliger Ausdruck in dem unter **Farbe** beschriebenen Bereich.
Rand erlaubt das Zeichnen eines Randes um den Ausschnitt (falls Platz verfuegbar ist). Wird **Rand** weggelassen, wird kein Rand gezeichnet.

Es ist zu beachten, dass **VIEW** die x- und y-Argumentpaare sortiert, indem die kleineren Werte von x und y an den Anfang gestellt werden.

```
Aus VIEW (100,100)-(10,10) wird VIEW (10,10)-(100,100)
aus VIEW (300,100)-(150,150) wird VIEW (150,100)-(300,150)
```

Es sind alle Angaben von x und y gueltig, nur muss x1 ≠ x2 und y1 ≠ y2 sein und der Ausschnitt darf nicht groesser sein als die sichtbare Oberflaeche.
Gezeichnet werden kann nur innerhalb der definierten Flaeche. Ueber den Rand hinaus **Linien** und **Objekte** werden abgeschnitten. Ohne die Angabe des Argumentes **SCREEN** in der Anweisung **VIEW** sind alle nachfolgend gezeichneten Punkte relativ zum Ausschnitt, d.h. x1 und y1 werden zu der x- und y-Koordinate addiert, bevor etwas auf dem Bildschirm ausgegeben wird.

Beispiel:

```
10 VIEW (10,10)-(100,100)
20 PSET (0,0),2
```

Der gezeichnete Punkt hat die aktuellen Bildschirmkoordinaten (10,10).

Wird das Argument **SREEN** eingefuegt, so sind alle gezeichneten Punkte absolut und koennen sich innerhalb oder ausserhalb der Bildschirmgrenzen befinden. Sichtbar sind jedoch nur die Punkte, die sich innerhalb des ausgewaehlten Ausschnittes befinden.

Beispiel:

```
10 VIEW SCREEN (10,10)-(100,100)
20 PSET (0,0),2
```

Der mit **PSET** gezeichnete Punkt erscheint nicht auf dem Bildschirm, da sich (0,0) ausserhalb des Ausschnittes befindet.

```
10 VIEW SCREEN (10,10)-(100,100)
20 PSET (10,10),2
```

In diesem Fall wird der Punkt in die linke obere Ecke des Ausschnittes gezeichnet.

VIEW ohne Argument definiert die gesamte Bildschirmflaeche als Ausschnitt.
Man kann mehrere Ausschnitte definieren, aber es darf immer nur einer aktiv sein.
Die Anweisungen **RUN** und **SCREEN** inaktivieren alle Ausschnitte.

Hinweis:

Wird **VIEW** benutzt, so loescht **CLS** nur den laufenden Ausschnitt. Um den gesamten Bildschirm zu loeschen, muessen mit **VIEW** alle Ausschnitte inaktiviert werden.

=_WINDOW

Neudefinieren der Bildschirmkoordinaten.

Syntax: **WINDOW [[SCREEN](x1,y1)-(x2,y2)]**

Bemerkungen:

(x1,y1),(x2,y2) sind vom Programmierer definierte Koordinaten, sogenannte Weltkoordinaten.
Diese Koordinaten sind Gleitkommazahlen einfacher Genauigkeit. Sie definieren den Bereich in Weltkoordinaten, der in einen Bereich physikalischer Koordinaten umgerechnet wird.

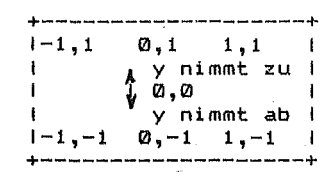
WINDOW erlaubt das Zeichnen von Objekten im Raum ("Weltkoordinatensystem"), ohne durch die logischen Grenzen des Bildschirms begrenzt zu sein ("physikalisches Koordinatensystem").

BASIC wandelt die Weltkoordinatenpaare so um, dass sie anschliessend innerhalb des Ausschnittes auf dem Bildschirm angezeigt werden koennen.

Der rechtwinklige Bereich im Weltkoordinatenbereich heisst **Window** (Fenster).
Wird das Attribut **SCREEN** weggelassen, so wird der Bildschirm in richtigen Kartesischen Koordinaten dargestellt:

WINDOW (-1,-1)-(1,1)

definiert folgenden Bildschirm:

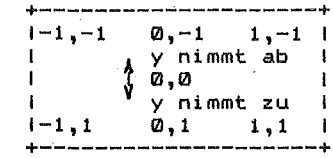


Hierbei ist zu beachten, dass die y-Koordinate umgedreht ist, so dass (x1,y1) der untere linke und (x2,y2) der obere rechte Koordinatenpunkt sind.

Wird das Attribut **SCREEN** eingefuegt, werden die Koordinaten nicht umgedreht, so dass (x1,y1) der linke obere und (x2,y2) der rechte untere Koordinatenpunkt sind.

WINDOW SCREEN (-1,-1)-(1,1)

definiert den Bildschirm wie folgt:



Hinweis:

WINDOW sortiert die Argumentenpaare x und y und stellt die kleinsten Werte fuer x und y an den Anfang.

Beispiel:

Aus WINDOW (-10,10)-(10,-10)
wird WINDOW (-10,-10)-(10,10)

Es sind alle moeglichen Angaben von x und y moeglich, nur muss x1 ≠ x2 und y1 ≠ y2 sein. Punkte, die ausserhalb des Koordinatenbereiches liegen, werden nicht sichtbar und Objekte, die teilweise innerhalb und teilweise ausserhalb des Koordinatenbereiches liegen, werden abgeschnitten.

Mit **WINDOW** kann man auch vergroessern oder verkleinern.
Durch die Angabe kleiner oder grosser Fenstergroessen kann ein Objekt vergroessert werden, bis es den gesamten Bildschirm ausfuellt oder verkleinert werden, dass nur noch ein Punkt zu sehen ist.

Durch **RUN, SCREEN** und **WINDOW** ohne Argumente werden alle **WINDOW**-Koordinaten inaktiviert und der Bildschirm wieder in physischen Koordinaten angesprochen.

Beispiel:

```
10 CLS: SCREEN 1: COLOR 0,1
20 WINDOW (-10,-10)-(10,10)
30 CIRCLE (0,0),5,1
40 WINDOW (-4,-4)-(4,4)
50 WINDOW (-100,-100)-(100,100)
```

_PMAP

Umrechnen der physikalischen Koordinaten in Weltkoordinaten und umgekehrt.

Syntax: v = PMAP (x,n)

Bemerkungen:

x — Koordinate des umzurechnenden Punktes

n — Wert im Bereich 0 bis 3

n = 0: umrechnen der Weltkoordinate x in die physikalische Koordinate x
1: umrechnen der Weltkoordinate y in die physikalische Koordinate y
2: umrechnen der pyhsikalischen Koordinate x in die Weltkoordinate x
3: umrechnen der physikalischen Koordinate y in die Weltkoordinate y

PMAP (x,0) und **PMAP (x,1)** rechnen Werte des Weltsystems in das physikalische Koordinatensystem um.

PMAP (x,2) und **PMAP (x,3)** rechnen Werte des physikalischen Koordinatensystems in das Weltkoordinatensystem um.

Beispiel:

```
10 SCREEN 1
20 WINDOW (-1,-1)-(1,1)
30  x1 = PMAP (-1,0)     'uebergibt den physikalischen x-
                          Koordinatenwert von 0
40 y1 = PMAP (-1,1)      'uebergibt den physikalischen y-
                          Koordinatenwert von 199
50 x2 = PMAP (1,0)       'uebergibt den physikalischen x-
                          Koordinatenwert von 319
60 y2 = PMAP (1,1)       'uebergibt den physikalischen y-
                          Koordinatenwert von 0
70 PRINT x1,y1,x2,y2
```

8.4.2. Funktionen

POINT

Uebergeben der Farbe eines angegebenen Punkts auf dem Bildschirm oder der laufenden grafischen Koordinate oder des Wertes der laufenden x- oder y-Koordinaten.

Syntax: `v = POINT (x,y)`
`v = POINT (n)`

Bemerkungen:

(x,y) Koordinaten des benutzten Punktes. Die Koordinaten muessen in absoluter Form verfuegbar sein.

Liegt der angegebene Punkt nicht im Bereich, wird der Wert -1 uebergeben. In der mittleren Aufloesung sind gueltige uebergebene Wert 0, 1, 2 und 3, in der hohen Aufloesung 0 und 1.

n uebergibt den Wert der laufenden grafischen Koordinate x oder y. n kann folgende Werte annehmen:

0 - uebergibt die laufende physikalische x-Koordinate
1 - uebergibt die laufende physikalische y-Koordinate
2 - uebergibt die laufende Weltkoordinate x, falls **WINDOW** aktiv ist, sonst wie 0
3 - uebergibt die laufende Weltkoordinate y, falls **WINDOW** aktiv ist, sonst wie 1

Beispiel:

Uebergabe von Werten in Abhaengigkeit von **WINDOW**

```
 10 CLS: SCREEN 1: KEY OFF
 20 PRINT "POINT(n) ohne WINDOW"
 30 GOSUB 110
 40 WINDOW(0,0)-(319,199)
 50 PRINT "POINT(n) mit WINDOW"
 60 GOSUB 110
 70 PRINT "POINT(n) mit WINDOW und SCREEN"
 80 WINDOW SCREEN (0,0)-(319,199)
 90 GOSUB 110
100 END
110 PSET(10,20)
120 FOR I=0 TO 3
130 PRINT POINT(I);
140 NEXT
150 PRINT : PRINT
160 RETURN
```

```
RUN
POINT(n) ohne WINDOW
 10  20  10  20
POINT(n) mit WINDOW
 10  179  10  20
POINT(n) mit WINDOW und SCREEN
 10  20  10  20
Ok
```

=_SCREEN

Uebergeben des ASCII-Codes (0 bis 255) fuer ein Zeichen auf dem aktiven Bildschirm an der angegebenen Zeile und Spalte.

Syntax: v = SCREEN (Zeile,Spalte)[,z]

Bemerkungen:

Zeile Numerischer Ausdruck im Bereich 1 bis 25

Spalte Numerischer Ausdruck im Bereich 1 bis 40 oder 1 bis 80, abhaengig vom Setzen der Breite **WIDTH**

z Numerischer Ausdruck, dem der Wert wahr oder falsch zugeordnet wird. **z** ist nur im Textmodus gueltig.

Anlage D enthaelt eine Liste aller ASCII-Codes.

Ist im Textmodus z eingefuegt und wahr (nicht Null), wird das Farbattribut fuer dieses Zeichen statt des Zeichencodes uebergeben. Das Farbattribut ist eine Zahl im Bereich 0 bis 255.

Die Zahl v kann wie folgt entziffert werden:

(v MOD 16) ist die Vordergrundfarbe

(((v-Vordergrundfarbe)/16) MOD 128) ist die Hintergrundfarbe, wobei **Vordergrund** wie oben berechnet wird

(v>127) ist wahr **(-1)**, falls das Zeichen blinkt, falsch **(0)**, falls es nicht blinkt

Unter Punkt 8.1. befindet sich die Liste aller Farben und ihrer zugehoerigen Nummern.

Enthaelt die angegebene Stelle im grafischen Modus grafische Informationen (Punkte oder Linien im Gegensatz zu einem Zeichen), uebergibt die Funktion **SCREEN** eine 0 (Null).
Jeder eingegebene Wert ausserhalb des Bereichs erzeugt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Beispiel:

```
X = SCREEN (12,10)
```

Ist das Zeichen an der Position 12,10 ein A, ist X gleich 65.

8.5. Musik

8.5.1. Anweisungen

- SOUND

Erzeugen eines Tones ueber den Lautsprecher.

Syntax: **SOUND Freq, Dauer**

Bemerkungen:

Freq Gewuenschte Frequenz in Hertz (Zyklen pro Sekunde). Numerischer Ausdruck im Bereich 37 bis 32767.

Dauer Gewuenschte Dauer in Zeittakten. Die Zeittakte treten 18.2 mal pro Sekunde auf. **Dauer** muss ein numerischer Ausdruck im Bereich 0 bis 65535 sein.

Erzeugt die Anweisung **SOUND** einen Ton, faehrt das Programm mit der Ausfuehrung fort, bis eine neue Anweisung **SOUND** erreicht wird. Wird in der neuen Anweisung die Dauer als **SOUND 0** (Null) angegeben, wird der Ton der laufenden Anweisung **SOUND** ausgeschaltet. Im anderen Fall wartet das Programm, bis der erste Ton beendet ist, bevor die neue Anweisung **SOUND** ausgefuehrt wird.

Die Toene koennen gepuffert werden, so dass die Ausfuehrung nicht stoppt, wenn eine neue Anweisung **SOUND** von **BASIC** erkannt wird. Siehe Zeichenbefehl **MB**, der unter "Anweisung **PLAY**" in diesem Kapitel in allen Einzelheiten erklaert wird.

Laeuft keine Anweisung **SOUND** ab, hat **SOUND x,0** keinen Einfluss.

Die Note A hat die Frequenz 440. In der folgenden Tabelle werden die Noten und ihre Frequenzen gezeigt.

Note	Frequenz	Note	Frequenz
C	130.810	c**	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
B	246.940	B	987.770
C*	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
B	493.880	B	1975.500

* mittleres C
** kleines c

Hoehere (oder tiefere) Noten koennen durch Verdoppelung (oder Halbieren) der Frequenz der entsprechenden Note in der naechsttieferen (hoeheren) Oktave angenaehert werden.

Zur Erstellung von Pausenperioden muss **SOUND 32767,Dauer** ausgefuehrt werden.

Die Dauer eines Schlages kann aus den Schlaegen pro Minute errechnet werden, indem man die Schlaege pro Minute durch 1092 teilt (die Anzahl der Zeitschlaege pro Minute).

Die naechste Tabelle zeigt typische Tempi als Ausdruck von Zeitschlaegen:

Tempo		Schlaege/ Minute	Einheiten/ Schlag
Sehr langsam	Larghissimo		
	Largo	40-60	27.3-18.2
	Larghetto	60-66	18.2-16.55
	Grave		
	Lento		
	Adagio	66-76	16.55-14.37
Langsam	Adagietto		
	Andante	76-108	14.37-10.11
Mittel	Andantino		
	Moderato	108-120	10.11-9.1
Schnell	Allegretto		
	Allegro	120-168	9.1-6.5
	Vivace		
	Veloce		
	Presto	168-208	6.5-5.25
Sehr schnell	Prestissimo		

Beispiel:

```
SOUND 262,5
```

Es wird ein Ton erzeugt der Frequenz 262 Hz (mittleres C) eine viertel Sekunde lang.

PLAY

Spielen von Musik mit Hilfe einer **Zeichenkette.**

Syntax: **PLAY Zeichenkette**

Bemerkungen:

PLAY arbeitet nach einem aehnlichen Konzept wie **DRAW**, indem einer Zeichenkette eine "Tondefinitionssprache" zugeordnet wird.

Zeichenkette Zeichenkettenausdruck, der aus einzelnen Zeichen fuer Musikbefehle besteht.

Die einzelnen Zeichenbefehle fuer **PLAY**:

A bis G mit wahlfreiem **#**, **+** oder **-**

Spielt die angegebenen Noten in der laufenden Oktave. Ein nachgestelltes Nummernzeichen (#) oder Pluszeichen (+) zeigt einen Halbton hoeher, ein Minuszeichen (-) zeigt einen Halbton tiefer an.
#, **+** oder **-** sind nur erlaubt, wenn sie einer schwarzen Taste auf dem Klavier entsprechen. **B#** ist z.B. eine ungueltige Note.

On

Oktave. Setzt die laufende Oktave fuer die folgenden Noten. Es gibt sieben Oktaven, numeriert von **0 bis 6**. Jede Oktave geht von **C bis B**. Die Oktave 3 beginnt mit dem kleinen c. Oktave 4 ist die Standardoktave.

>n

Eine Oktave hoeher gehen und die Note n spielen. Jedesmal wenn die Note gespielt wird, wird die Oktave erhoeht, bis die Oktave 6 erreicht ist. Zum Beispiel **PLAY ">A"** erhoeht die Oktave und spielt die Note A. Bei jedem Durchlauf der Anweisung wird die Oktave erhoeht, bis die Oktave 6 erreicht ist, danach wird bei jeder Ausfuehrung in der Oktave 6 gespielt.

<n

Eine Oktave niedriger gehen und die Note n spielen. Jedesmal wenn die Note gespielt wird, wird die Oktave verringert, bis die Oktave **0** erreicht ist.
Zum Beispiel **PLAY "<A"** verringert die Oktave und spielt die Note A. Bei jedem Durchlauf der Anweisung wird die Oktave verringert, bis die Oktave 0 erreicht ist, danach wird bei jeder Ausfuehrung in Oktave 0 gespielt.

Nn

Spielt die Note n. n kann sich im Bereich 0 bis 84 befinden. In den sieben moeglichen Oktaven befinden sich 84 Noten. **n=0** bedeutet Pause. Dies ist eine weitere Alternative, um Noten auszuwaehlen.

Ln

Setzt die Laenge der folgenden Noten. Die aktuelle Notenlaenge ist 1/n. n kann sich im Bereich 1 bis 64 befinden. In der folgenden Tabelle wird dies verdeutlicht.

Laenge	Bedeutung
L1	Ganze Note
L2	Halbe Note
L3	Ein Teil einer Triole aus drei halben Noten (1/3 eines Viertaktmasses)
L4	Viertelnote
L5	1/5 eines Taktmasses
L6	Ein Teil einer Triole aus drei Viertelnoten
:	
:	
L64	Eine 64stel Note

Die Laengenangabe kann auch hinter einer Note stehen, wenn man die Laenge nur fuer diese Note aendern moechte, z.B. ist **A16** gleichbedeutend mit **L16A**.

Pn Pause. n kann sich im Bereich 1 bis 64 befinden und zeigt die Laenge der Pause in der gleichen Art und Weise an wie L (Laenge).

. (Punkt). Ein Punkt nach einer Note bedeutet eine punktierte Note, d.h. die Laenge wird mit 3/2 multipliziert. Nach einer Note kann mehr als ein Punkt auftreten, und die Laenge wird anhand der Punkte errechnet.
Z.B. wird "A.." 9/4mal so lang gehalten wie L angibt, "A..." wird 27/8mal so lang gehalten usw. Punkte koennen auch nach einer Pause (P) erscheinen, um die Pausenlaenge auf die gleiche Art zu berechnen.

Tn Tempo. Setzt die Anzahl der Viertelnoten in einer Minute. n kann im Bereich 32 bis 255 liegen. Die Standardannahme ist **120**. Unter Anweisung **SOUND** befindet sich eine Tabelle, die haeufige Tempi und die zugehoerigen Schlaege pro Minute enthaelt.

MF Musikvordergrund. Musik (erstellt durch **SOUND** oder **PLAY**) laeuft im Vordergrund. Das bedeutet, jede nachfolgende Note oder Ton kann erst beginnen, nachdem die vorherige Note oder Ton beendet ist. **PLAY** kann durch **CTRL-PAUSE** beendet werden. Der Standardstatus ist Musik im Vordergrund.

MB Musikhintergrund. Musik (erstellt durch **SOUND** oder **PLAY**) laeuft im Hintergrund statt im Vordergrund ab. Das bedeutet, dass jede Note oder jeder Ton in einen Puffer gesetzt wird, wobei das Programm weiter laeuft, waehrend im Hintergrund Musik spiel. Bis zu 32 Noten (oder Pausen) koennen gleichzeitig im Hintergrund gespielt werden.

MN Normale Musik. Jede Note wird 7/8 der Zeit, die in **L** (Laenge) angegeben ist , gehalten. Dies ist die Standardannahme fuer **MN**, **ML** und **MS**.

ML Musik legato. Jede Note wird die volle Periode gehalten, die in **L** (Laenge) gesetzt wurde.

MS Musik staccato. Jede Note wird 3/4 der Zeit, die in **L** angegeben wurde, gehalten.

X Variable; fuehrt die angegebene Zeichenkette aus.

In all diesen Befehlen kann das Argument **n** eine Konstante, wie 12 oder **=Variable;** sein, wobei **Variable** der Name einer Variablen ist. Das Semikolon (;) wird benoetigt, wenn eine Variable auf diese Art und Weise und der Befehl **X** benutzt werden. Sonst ist das Semikolon zwischen Befehlen wahlfrei. Nicht erlaubt ist das Semikolon nach **MF**, **MB**, **MN**, **ML** oder **MS**. Auch Leerstellen in der **Zeichenkette** werden ignoriert.

Variable koennen in der Form **VARPTR$(Variable)** statt **=Variable;** angegeben werden. Dies ist fuer Programme nuetzlich, die spaeter compiliert werden.

Beispiel:

```
Methode                 Alternative Methode

PLAY "XA$;"             entspricht PLAY "X"+VARPTR$(A$)

PLAY "O=I;"             entspricht PLAY "O="VARPTR$(I)
```

X kann dazu benutzt werden, eine "Untermelodie" in einer Zeichenkette zu speichern und sie wiederholt in verschiedenen Tempi oder Oktaven von einer anderen Zeichenkette aufzurufen.

Beispiel:

Das folgende Beispiel spielt die Tonleiter.

```
PLAY "O3 CDEFGAB04C"
```

8.5.2. Anweisungen zur Programmunterbrechung

- PLAY ON/PLAY OFF/PLAY STOP

Syntax: **PLAY ON**
PLAY OFF
PLAY STOP

Bemerkungen:

PLAY ON aktiviert die Programmunterbrechung.

PLAY OFF inaktiviert die Programmunterbrechung. Eine Ausfuehrung von **PLAY** wird nicht gespeichert.

PLAY STOP inaktiviert die Programmunterbrechung ebenfalls. Es wird aber jede Ausfuehrung von **PLAY** gespeichert und nach der Ausfuehrung von **PLAY ON** erfolgt eine sofortige Unterbrechung (sh. auch Anweisung **ON PLAY(n)**).

- ON PLAY(n)

Spielen von Musik waehrend der Programmausfuehrung.

Syntax: **ON PLAY(n) GOSUB Zeile**

Bemerkungen:

n Ganzzahliger Ausdruck im Bereich 1 bis 32. Er zeigt die zu unterbrechenden Noten an. Werte ausserhalb dieses Bereiches ergeben den Fehler "Illegal function call" (ungueltiger Funktionsaufruf).

Zeile Erste Zeilennummer der Unterbrechungsroutine fuer **PLAY**. Die Zeilennummer 0 (Null) stoppt die Spielunterbrechung.

Eine **PLAY ON**-Anweisung muss benutzt werden, um die **ON PLAY(n)** Anweisung zu starten.

Ist nach **PLAY ON** in der Anweisung **ON PLAY(n)** eine Zeilennummer ungleich Null angegeben, prueft BASIC vor jeder Ausfuehrung einer neuen Anweisung, ob der Musikpuffer von n auf n -1 Noten verringert wurde. Ist dies der Fall, fuehrt BASIC ein **GOSUB** in der angegebenen Zeile aus.

Nach der Unterbrechung wird automatisch **PLAY STOP** ausgefuehrt, so dass eine rekursive Unterbrechung nicht auftreten kann.

Die Anweisung **RETURN** aus der Unterbrechungsroutine fuehrt automatisch ein **PLAY ON** aus, wenn nicht explizit innerhalb der Unterbrechungsroutine ein **PLAY OFF** ausgefuehrt wurde.

Tritt eine Fehlerunterbrechung auf (resultierend aus **ON ERROR**), wird jede Unterbrechung automatisch inaktiviert.

Hinweis:

- Eine Unterbrechung fuer **PLAY** findet nur statt, wenn **PLAY** im Hintergrundmusik-Modus verwendet wird (**PLAY "MB..."**). Eine Unterbrechung findet nicht statt, wenn **PLAY** im Vordergrundmusik-Modus verwendet wird (**PLAY "MF..."**).

- Eine Unterbrechung fuer **PLAY** findet nicht statt, wenn der Hintergrundmusikpuffer bereits leer ist, wenn die Anweisung **PLAY ON** ausgefuehrt wird.

- Bei der Auswahl der Werte fuer n ist Vorsicht geboten. Die Anweisung **ON PLAY(32)** wuerde so viele Unterbrechungen bewirken, dass nur wenig Zeit fuer das restliche Programm bleibt.

Beispiel:

In diesem Beispiel wird die Unterbrechungsroutine aufgerufen, wenn 4 Noten im Hintergrundpuffer uebriggeblieben sind.

```
 10 ON PLAY(4) GOSUB 100
 20 PLAY ON
 25 PLAY "MB O5CDEFGABO6C"
 30 PRINT "Musik im Hintergrund"
 40 FOR I=1 TO 1000
 45 PRINT I
 50 NEXT I
 60 END

100 REM Unterprogramm fuer Hintergrundmusik
110 PLAY "MB O3CDEFGABO4C",
120 RETURN 30
```

8.5.3. Funktionen

PLAY(n)

Uebergeben der Anzahl Noten, die sich im Puffer fuer Hintergrundmusik befinden.

Syntax: v = PLAY(n)

Bemerkungen:

n Scheinargument, dem jeder Wert zugeordnet werden kann.

PLAY(n) uebergibt eine 0, wenn das Programm im Modus Vordergrundmusik ablaeuft.

PLAY(n) uebergibt die Anzahl Noten, die sich im Puffer befinden, wenn der Modus Hintergrundmusik **(MB)** verwendet wird.

Der groesste Wert, der uebergeben werden kann, ist 32, d.h. die maximale Anzahl von Noten, die sich im Puffer befinden kann.

9. Prozessorarbeit

9.1. Arbeit mit dem Speicher

Wird mit dem BASIC-Interpreter gearbeitet, wird der gesamte verfuegbare Hauptspeicher bis 64K Bytes benutzt. Dieser **BASIC-Arbeitsbereich** enthaelt das BASIC-Programm und die Daten, auch den Arbeitsbereich fuer den Interpretierer (zwischen 2,5 und 4K Bytes) und einen BASIC-Stapelbereich.

Sollen Unterprogramme in Maschinensprache in ein BASIC-Programm eingebunden werden, kann Hauptspeicherplatz innerhalb oder ausserhalb dieses 64K-BASIC-Arbeitsbereiches angelegt werden. Wo diese Assembler-Routinen untergebracht werden, haengt davon ab, wieviel Hauptspeicher insgesamt zur Verfuegung steht und wie gross die zu ladenden Programme sind. Dazu sind Kenntnisse ueber den Speicheraufbau des Computers erforderlich (siehe Anlage H).

Sollen Unterprogramme in Maschinensprache innerhalb des 64K-BASIC-Arbeitsbereiches gespeichert werden, muss der Hauptspeicher fuer BASIC begrenzt werden, damit BASIC die Unterprogramme in Maschinensprache nicht ueberschreibt. Dazu ist entweder das Kommando "**CLEAR**" (siehe Punkt 3.2.2.) zu verwenden oder beim Initialisieren von BASIC der Schalter /M: zu setzen.

Nur die hoechsten Hauptspeicherplaetze koennen fuer Unterprogramme reserviert werden.

Sollen Unterprogramme in Maschinensprache ausserhalb dieses 64K-BASIC-Arbeitsbereiches gespeichert werden, werden mehr als 64K Byte Hauptspeicher benoetigt.
Dann belegen DCP ungefaehr 12K Bytes und BASIC weitere 10K Bytes, so dass mindestens ein 96K Byte-System benoetigt wird, um Platz fuer Unterprogramme in Maschinensprache zu schaffen.
In diesem Falle wird mit der Anweisung **DEF SEG** der externe Unterprogrammbereich ausserhalb des BASIC-Arbeitsbereiches adressiert.

Weitere Hinweise und Beispiele zur Prozessorarbeit sind aus der Dokumentation "Anwendung des BASIC-Systems" zu entnehmen.

9.2. Anweisungen und Funktionen fuer den Zugriff auf den Speicher

9.2.1. DEF SEG

Definiert das gerade angesprochene "Segment" des Hauptspeichers. Eine nachfolgende BLOAD-, BSAVE-, CALL-, PEEK-, POKE- oder USR-Anweisung definiert die aktuelle physische Adresse der Anweisung als einen Relativzeiger in dieses Segment.

Syntax: DEF SEG[=Segment]

Bemerkungen:

Segment ist ein numerischer Ausdruck im Bereich 0 bis 65535.

Wird BASIC gestartet, ist das erste Segment das Datensegment von BASIC und gleichzeitig der Anfang des Benutzerbereichs im Hauptspeicher. Wird eine Anweisung DEF SEG ausgefuehrt, die das Segment aendert, wird der Wert **nicht** auf das BASIC-Datensegment zurueckgesetzt, wenn der Befehl RUN ausgefuehrt wird.

Wird **Segment** in der Anweisung DEF SEG weggelassen, wird das Segment auf das Datensegment des BASIC gesetzt.

Ist **Segment** angegeben, muss sein Wert eine 16-Byte-Grenze haben. Der Wert wird um vier Bits nach links verschoben (mit 16 multipliziert), um die Segmentadresse fuer nachfolgende Operationen zu formen. Das heisst, falls das **Segment** hexadezimal ist, wird eine 0 (Null) angefuegt, um die aktuelle Segmentadresse zu erhalten. BASIC fuehrt keine Pruefung durch, um zu sehen, ob der Segmentwert gueltig ist.

DEF und **SEG** muessen durch eine Leerstelle getrennt werden. Im anderen Fall wird BASIC die Anweisung DEFSEG=100 so interpretieren:

"Den Wert 100 an die Variable DEFSEG uebergeben."

Jeder eingegebene Wert, der sich ausserhalb des angegebenen Bereichs befindet, erzeugt den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf). Der vorherige Wert wird beibehalten.

Beispiel:

```
100 DEF SEG 'wieder auf das BASIC-Datensegment setzen
.
.
.
200 'Segment auf Farbbildschirmpuffer setzen
210 DEF SEG = &HB800
```

Im Beispiel liegt der Bildschirmpuffer fuer den Farb-/Grafikbildschirmanschluss auf der absoluten Adresse B8000 hexadezimal (Segment B800H, Relativzeiger 0). Da Segmente in 16-Byte-Grenzen angegeben werden, wird die letzte Hexadezimalziffer in der Anweisung DEF SEG weggelassen.

9.2.2. POKE

Schreibt ein Byte in eine Hauptspeicherstelle.

Syntax: POKE n,m

Bemerkungen:

n muss sich im Bereich 0 bis 65535 befinden und zeigt die Adresse im Hauptspeicher an, an die die Daten geschrieben werden sollen. Es ist der Relativzeiger des laufenden Segments, wie in der Anweisung DEF SEG definiert (siehe Anweisung "DEF SEG" in diesem Kapitel).

m sind die Daten, die an den angegebenen Platz geschrieben werden sollen. Sie muessen sich im Bereich 0 bis 255 befinden.

Die Gegenfunktion zu **POKE** ist **PEEK**. (Siehe "PEEK" in diesem Kapitel.) **POKE** und **PEEK** sind vorteilhaft, um Datenspeicher effizient auszunutzen, um Unterprogramme in Maschinensprache zu laden und um Argumente und Ergebnisse in und aus den Maschinensprache-Unterprogrammen zu uebergeben.

Warnung:

BASIC fuehrt keine Adresspruefung aus. Deshalb ist es nicht erlaubt, mit **POKE** etwas in BASIC-Stapelbereiche, in den BASIC-Variablenbereich oder in das BASIC-Programm zu schreiben.

Beispiel:

```
100 POKE &HF800,&H41
```

9.2.3. PEEK

Uebergibt ein gelesenes Byte von der angegebenen Hauptspeicherposition.

Syntax: v = PEEK(n)

Bemerkungen:

n ist eine ganze Zahl im Bereich 0 bis 65535. n ist der Relativzeiger des laufenden Segments, wie in der Anweisung **DEF SEG** definiert, und zeigt die Hauptspeicheradresse an, aus der gelesen werden soll. (Siehe Anweisung "DEF SEG" in diesem Kapitel.)

Der uebergebene Wert ist eine ganze Zahl im Bereich 0 bis 255.

PEEK ist die Gegenfunktion der Anweisung **POKE**.

Beispiel:

```
100 PRINT PEEK(&HF800)
```

9.2.4. BLOAD

Laedt eine Datei mit einem Hauptspeicherabbild in den Hauptspeicher.

Syntax: **BLOAD Dateiangabe[,Relativzeiger]**

Bemerkungen:

Dateiangabe Zeichenkettenausdruck, wie in "Namensgebung fuer Dateien" in Kapitel 6 beschrieben. Es ist die Angabe eines Pfades moeglich. Entspricht die Dateiangabe nicht den Regeln, wird der Fehler "Bad file name" (falscher Dateiname) angezeigt und das Laden abgebrochen.

Relativzeiger Numerischer Ausdruck im Bereich 0 bis 65535. Er gibt die Adresse an, wo das Laden beginnen soll, angegeben als ein Relativzeiger in das durch die letzte Anweisung **DEF SEG** deklarierte Segment.

Wird der Relativzeiger weggelassen, so wird der im **BSAVE** angegebene angenommen. Das bedeutet, dass die Datei an denselben Platz geladen wird, von dem sie gespeichert wurde.

Wird der Befehl **BLOAD** ausgefuehrt, wird die durch einen Namen angegebene Datei, beginnend mit dem angegebenen Platz, in den Hauptspeicher geladen.

Fehlt der Einheitenname, wird die DCP-Standarddisketteneinheit benutzt. BLOAD wird bei einer Datei verwendet, die zuvor mit BSAVE gesichert wurde.

BLOAD und **BSAVE** sind fuer das Laden und Speichern von Programmen in Maschinensprache sehr nuetzlich (siehe Anweisung **CALL** in diesem Kapitel) , jedoch sind **BLOAD** und **BSAVE** nicht nur auf Programme in Maschinensprache beschraenkt. Jedes Segment kann mit Hilfe der Anweisung **DEF SEG** als Quelle oder Ziel dieser Anweisungen angegeben werden. Das ist ein nuetzlicher Weg, Bildschirmabbildungen zu speichern oder anzuzeigen, indem man sie aus dem Bildschirmpuffer speichert oder in den Bildschirmpuffer laedt.

Warnung:

BLOAD fuehrt keine Adressbereichspruefung durch. Das heisst, es ist moeglich, irgendwohin in den Hauptspeicher zu laden. Es darf nicht ueber den BASIC-Stapelbereich, ueber den BASIC-Variablenbereich oder das BASIC-Programm geladen werden. (siehe Hauptspeicherbelegung in Anlage H).

Beispiel:

```
10 'Laden des Bildschirmpuffers
20 'Mit SEG auf den Bildschirmpuffer zeigen
30 DEF SEG = &HB800
40 'Datei BILD in den Bildschirmpuffer laden
50 BLOAD "BILD",0
```

In diesem Beispiel wird der Bildschirmpuffer fuer den Farb-/Grafikbildschirmanschluss geladen, der sich an der absoluten Adresse hexadezimal B8000 befindet. Sollte der Bildschirmpuffer fuer den Schwarz/-Weiss-Bildschirm und Paralleldruckeranschluss geladen werden, muesste die Zeile 30 in &HB000 (die aktuelle Adresse ist hexadezimal B0000) geaendert werden. Es ist zu beachten, dass die Anweisung **DEF SEG** in Zeile 30 und der Relativzeiger 0 in Zeile 50 wichtig sind. Dies garantiert, dass die richtige Adresse benutzt wird.

Das Beispiel fuer **BSAVE** im naechsten Abschnitt zeigt, wie BILD gespeichert wurde.

9.2.5. BSAVE

Damit werden Teile des Hauptspeichers in der angegebenen Einheit gespeichert.

Syntax: **BSAVE Dateiangabe, Relativzeiger, Laenge**

Bemerkungen:

Dateiangabe Zeichenkettenausdruck fuer eine Dateispezifikation. Sie muss den Regeln der "Namensgebung fuer Dateien" in Kapitel 6 entsprechen; andernfalls wird der Fehler "Bad File Name" (falscher Dateiname) angezeigt, und das Speichern wird abgebrochen. Es ist die Angabe eines Pfades moeglich.

Relativzeiger Numerischer Ausdruck im Bereich 0 bis 65535. Das ist der Relativzeiger fuer das Segment, das in der letzten Anweisung **DEF SEG** deklariert wurde. Ab dieser Position wird gespeichert.

Laenge Numerischer Ausdruck im Bereich 0 bis 65535. Das ist die Laenge des Hauptspeicherabbilds, das gespeichert werden soll.

Wird Relativzeiger oder Laenge weggelassen, tritt ein "Syntax Error" (Syntaxfehler) auf, und das Speichern wird abgebrochen.

Fehlt der Einheitenname, wird die DCP-Standarddisketteneinheit benutzt.

BLOAD und **BSAVE** sind nuetzlich fuer das Laden und Speichern von Programmen in Maschinensprache (die mit Hilfe der Anweisung **CALL** aufgerufen werden koennen). Jedoch sind **BLOAD** und **BSAVE** nicht nur auf Programme in Maschinensprache beschraenkt, sondern mit Hilfe der Anweisung **DEF SEG** kann jedes Segment als Ziel oder Quelle dieser Anweisungen angegeben werden. Das Abbild eines Bildschirms kann z.B. gespeichert werden, indem man mit **BSAVE** den Bildschirmpuffer speichert.

Beispiel:

```
10 'Speichern des Puffers eines Farbbildschirms
15 'Mit SEG die Adresse des Bildschirmpuffers angeben
20 DEF SEG = &HB800
25 'Speichern des Puffers in der Datei BILD
30 BSAVE "BILD",0,&H4000
```

Wie schon unter Anweisung "**BLOAD**" im vorherigen Abschnitt beschrieben, ist die Adresse des 16K-Bildschirmpuffers fuer den Farb-/Grafikbildschirmanschluss hexadezimal **B8000**. Die Adresse des 4K-Bildschirmpuffers fuer den Schwarz/-Weiss-Bildschirm und Paralleldruckeranschluss ist hexadezimal **B0000**.

Mit der Anweisung **DEF SEG** muss die Segmentadresse auf den Anfang des Bildschirmpuffers gesetzt werden. Relativzeiger **0** und Laenge **&H4000** geben an, dass der gesamte 16K-Bildschirmpuffer gespeichert werden soll.

9.2.6. VARPTR

Uebergibt den Relativzeiger zu dem aktuellen Speichersegment der Variablen.

Syntax: v = VARPTR(Variable)

Bemerkungen:

Variable Name einer numerischen oder Zeichenkettenvariablen oder eines Feldelements im Programm. Bevor **VARPTR** aufgerufen wird, muss der Variablen ein Wert zugeordnet werden, sonst erhaelt man den Fehler "Illegal Function Call" (Ungueltiger Funktionsaufruf).

Die uebergebene Adresse ist eine ganze Zahl im Bereich 0 bis 65535. Diese Nummer ist der Relativzeiger fuer das BASIC-Datensegment des ersten Datenbytes, das mit Variable angesprochen wird. Das Format dieser Daten wird in der Anlage H unter "Speicherung von Variablen" beschrieben.

Hinweis:
Allen einfachen Variablen sollte ein Wert zugeordnet sein, bevor **VARPTR** fuer ein Feld aufgerufen wird, weil sich die Adressen von Feldern jedesmal aendern, wenn eine neue einfache Variable zugeordnet wird.

VARPTR wird normalerweise benutzt, um den Relativzeiger einer Variablen oder eines Feldes zum BASIC-Datensegment zu erhalten, so dass man mit diesem Zeiger auf Unterprogramme in Maschinensprache zugreifen und Variablen oder Felder an sie uebergeben kann. Ein Funktionsaufruf der Form VARPTR(A(0)) wird normalerweise angegeben, wenn ein Feld uebergeben werden soll, wobei das am niedrigsten adressierte Element des Feldes uebergeben wird.

Beispiel:
In diesem Beispiel werden mit Hilfe von VARPTR Daten aus einer Variablen gelesen. In Zeile 30 wird die Adresse der Daten nach P uebertragen. Ganzzahlige Daten werden in zwei Bytes gespeichert, das niederwertige Byte zuerst. Der Wert, der in Stelle P gespeichert ist, wird in Zeile 40 errechnet. Die Bytes werden mit der Funktion PEEK gelesen, und das zweite Byte wird mit 256 multipliziert, da es die hoeherwertigen Bits enthaelt.

```
10 DEFINT A-Z
20 DATA1 = 500
30 P = VARPTR(DATA1)
40 A = PEEK(P)+256*PEEK(P+1)
50 PRINT A
```

9.2.7. VARPTR$

Es wird die Zeichenform der Adresse einer Variablen im Hauptspeicher uebergeben. Sie wird vorzugsweise mit PLAY und DRAW in Programmen benutzt, die spaeter compiliert werden.

Syntax: v$ = VARPTR$(Variable)

Bemerkungen:

Variable Name einer Variablen, die sich im Programm befindet.

Hinweis:
Zu allen einfachen Variablen sollte ein Wert zugeordnet sein, bevor VARPTR$ fuer ein Feldelement aufgerufen wird, da sich die Adressen von Feldern jedesmal aendern, wenn eine einfache Variable zugeordnet wird.

VARPTR$ uebergibt eine Drei-Byte-Zeichenkette der folgenden Form:

Byte 0	Byte 1	Byte 2
Typ	Niedrigwertiges Byte einer Variablenadresse	Hoeherwertiges Byte einer Variablenadresse

Typ: Zeigt den Variablentyp an:

2	Ganzzahlig
3	Zeichenkette
4	Einfache Genauigkeit
8	Doppelte Genauigkeit

Der uebergebene Wert ist der gleiche wie:

CHR$(Typ)+MKI$(VARPTR(Variable))

Mit Hilfe von **VARPTR$** kann man einen Variablennamen in der Befehlszeichenkette fuer **PLAY** oder **DRAW** angeben:

Methode 1	Alternative Methode
PLAY "XA$;"	PLAY "X"+VARPTR$(A$)
PLAY "O=I;"	PLAY "O="+VARPTR$(I)

9.2.8. OUT

Sendet ein Byte zu einem Maschinenausgabetor.

Syntax: OUT n,m

Bemerkungen:

n ist ein numerischer Ausdruck fuer eine Tornummer im Bereich 0 bis 65535.

m ist ein numerischer Ausdruck fuer zu sendende Daten im Bereich 0 bis 255.

OUT ist die Gegenanweisung zur Funktion **INP**. Siehe Funktion "INP" in diesem Kapitel.

Mit **OUT** kann man die Videoausgabe beeinflussen. Bei manchen Bildschirmen, die an den Farb-/Grafikbildschirmanschluss angeschlossen sind, ist es moeglich, dass die ersten zwei oder drei Zeichen einer Zeile nicht am Bildschirm angezeigt werden. Falls der Bildschirm keine horizontale Ausgleichssteuerung hat, kann mit Hilfe der folgenden Anweisungen die Bildschirmanzeige verschoben werden:

```
OUT 980,2: OUT 981,43
```

Mit diesem Beispiel wird die Anzeige um zwei Zeichen nach rechts fuer eine Zeichenbreite von 40 verschoben (oder 16 Punkte im grafischen Modus fuer mittlere Aufloesung oder 32 Punkte im grafischen Modus fuer hohe Aufloesung).

```
OUT 980,2: OUT 981,85
```

In diesem Beispiel wird die Bildschirmanzeige um fuenf Zeichen nach rechts fuer eine Zeichenbreite von 80 verschoben.

Die Verschiebung durch diese OUT-Anweisung bleibt solange aktiviert, bis eine Anweisung **WIDTH** oder **SCREEN** ausgefuehrt wird. Auch mit dem Befehl **MODE** in DCP kann diese Verschiebung der Bildschirmanzeige vorgenommen werden. Dieser Befehl hat den Vorteil, dass er gueltig bleibt, bis ein Warmstart am System ausgefuehrt wird.

Beispiel:

```
100 OUT 32,100
```

Damit wird der Wert 100 zum Ausgabetor 32 gesendet.

9.2.9. INP

Uebergibt ein gelesenes Byte vom Tor n.

Syntax: `v = INP(n)`

Bemerkungen:

n muss im Bereich 0 bis 65535 liegen.

INP ist die Gegenfunktion zur Anweisung **OUT** (siehe Anweisung "OUT" in diesem Kapitel).

INP fuehrt die gleiche Funktion aus, wie der Befehl **IN** in Assembler-Sprache.

Beispiel:

```
100 A = INP(255)
```

Diese Anweisung liest ein Byte vom Tor 255 und ordnet seinen Inhalt der Variablen A zu.

9.2.10. WAIT

Verlaesst die Programmausfuehrung zur Beobachtung des Status eines Maschineneingabetores.

Syntax: `WAIT Tor, n[,m]`

Bemerkungen:

Tor Tornummer im Bereich 0 bis 65535.

n,m sind ganzzahlige Ausdruecke im Bereich 0 bis 255.

n=AND-Maske
m=XOR-Maske

Durch die Anweisung **WAIT** wird die Ausfuehrung so lange ausgesetzt, bis das angegebene Maschineneingabetor ein vorgegebenes Bit-Abbild entwickelt.

Der Wartealgorithmus arbeitet wie folgt:

Der Wert, der bei der Toradresse anliegt, wird mit dem Wert der XOR-Maske durch ein logisches exklusives ODER verknuepft. Das Ergebnis wird mit dem Wert der AND-Maske durch ein logisches UND verknuepft. Ist das Ergebnis Ø (Null), verzweigt BASIC zurueck und liest die Daten des Tores erneut. Ist das Ergebnis nicht Ø (Null), wird das BASIC-Programm mit der naechsten Anweisung fortgesetzt. Wird die XOR-Maske nicht angegeben, wird standardmaessig der Wert Ø angenommen.
Durch die Anweisung WAIT kann man eine oder mehrere Bit-Positionen eines Eingabetores testen. Man kann die Bit-Positionen entweder auf Ø oder 1 testen. Die zu testenden Bit-Positionen werden dadurch angegeben, dass man die Positionen von **n** auf 1 setzt, die getestet werden sollen. Wird **m** nicht angegeben, werden die Bits des Eingabetores auf 1 getestet. Ist **m** angegeben, so wird jede 1 in irgendeiner Bit-Position in **m** (fuer die ein 1-Bit in n existiert), durch **WAIT** auf Ø fuer dieses Eingabe-Bit geprueft.

Bei der Ausfuehrung geht die Anweisung **WAIT** in eine Schleife und testet diese Eingabe-Bits, die mit 1 in **n** angegeben sind. Ist eines dieser Bits 1 (oder Ø, falls das korrespondierende Bit in m 1 ist), geht das Programm mit der naechsten Anweisung weiter. Das bedeutet, dass **WAIT** nicht auf das gesamte Bit-Abbild, sondern nur auf ein Bit wartet.

Hinweis:

Es ist moeglich, durch die Anweisung **WAIT** eine Endlosschleife zu erhalten. Aus dieser Schleife kommt man durch Betaetigen der Tasten <CTRL-PAUSE> oder ein Systemzuruecksetzen (Warmstart) heraus.

Beispiel:

Die Programmausfuehrung wird erst fortgesetzt, wenn Tor 32 ein Einer-Bit in der zweiten Bit-Position enthaelt.

```
100 WAIT 32,2
```

9.3.Anweisungen_und_Funktionen_fuer_die_Unterprogrammarbeit

9.3.1._DEF_USR

Spezifiziert die Startadresse eines Unterprogrammes in Maschinensprache, das spaeter durch die Funktion **USR** aufgerufen wird.

Syntax: **DEF USR[n] = Relativzeiger**

Bemerkungen:

n kann eine Ziffer zwischen 0 und 9 sein. Sie gibt die Nummer der USR-Routine an, deren Adresse spezifiziert wurde. Ist n weggelassen, wird DEF USR0 angenommen.

Relativzeiger
Ganzzahliger Ausdruck im Bereich 0 bis 65535. Der Wert des Relativzeigers wird zum laufenden Segmentwert addiert, um die aktuelle Startadresse der USR-Routine zu erhalten. Siehe Anweisung "DEF SEG" in diesem Kapitel.

Es ist moeglich, die Adresse fuer die USR-Routine neu zu definieren. Die Anzahl der Anweisungen **DEF USR** in einem Programm ist nicht begrenzt, so dass auf so viele Unterprogramme wie noetig zugegriffen werden kann. Der zuletzt ausgefuehrte Wert wird als Relativzeiger benutzt.

Beispiel:

```
200 DEF SEG = 0
210 DEF USR0 = 24000
  .
  .
  .
500 X = USR0(Y+2)
```

Dieses Beispiel ruft ein Unterprogramm auf, das ab der absoluten Adresse 24000 im Hauptspeicher steht.

9.3.2. USR

Ruft das angegebene Unterprogramm in Maschinensprache mit dem Argument arg auf.

Syntax: v = USR[n](arg)

Bemerkungen:

n liegt im Bereich 0 bis 9 und entspricht der Ziffer, die mit der Anweisung **DEF USR** fuer das gewuenschte Unterprogramm uebergeben wurde (siehe Anweisung "DEF USR" in diesem Kapitel). Wird n weggelassen, wird USR0 angenommen.

arg ist ein numerischer Ausdruck oder eine Zeichenkettenvariable, die das Argument des Unterprogramms in Maschinensprache ist. Erfordert das Unterprogramm kein Argument, so muss arg trotzdem als Scheinargument angegeben werden.

Die Anweisung CALL ist ein weiterer Weg, Unterprogramme in Maschinesprache aufzurufen .

Wird die Funktion USR aufgerufen, enthaelt Register AL einen Wert, der die Art des angegebenen Arguments beinhaltet. In AL steht einer der folgenden Werte:

Wert in AL	Argumenttyp
2	ganzzahliger Wert (Zweier-Komplement)
3	Zeichenkette
4	Zahl mit einfacher Genauigkeit
8	Zahl mit doppelter Genauigkeit

Ist das Argument eine **Zeichenkette**, zeigt das Register DX auf eine 3-Byte lange Zeichenkettenbeschreibung. Die Zeichenkettenbeschreibung wird in Anlage H, 2. Speicherung von Variablen, beschrieben.

Ist das Argument ein **Zahl** und keine Zeichenkette, wird der Wert des Arguments in dem Gleitkomma-Akummulator (FAC- = Floating-point accumulator) bereit gestellt. Darunter versteht man einen 8-Byte grossen Bereich im BASIC-Datenbereich. In diesem Fall enthaelt das Register BX den Relativzeiger innerhalb des BASIC-Datenbereichs zum fuenften Byte des 8-Byte langen FAC. In den folgenden Beispielen werden fuer FAC die Byte hexadezimal 49F bis hexadezimal 4A6 angenommen, d.h. BX enthaelt den Wert hexadezimal 4A3.

Ist das Argument **ganzzahlig:**

- Hexadezimal 4A4 enthaelt die oberen 8 Bit des Arguments.

- Hexadezimal 4A3 enthaelt die unteren 8 Bit des Arguments.

Ist das Argument eine **Zahl mit einfacher Genauigkeit:**

- Hexadezimal 4A6 enthaelt den Exponenten minus 128, und der Binaerpunkt befindet sich links vom hoechstwertigsten Bits der Mantisse. Hexadezimal 4A5 enthaelt die 7 hoechsten Bits der Mantisse, wobei die fuehrende 1 unterdrueckt wird (implizit). Bit 7 ist das Vorzeichen der Zahl (Ø=positiv; 1=negativ).

- Hexadezimal 4A4 enthaelt die mittleren 8 Bit der Mantisse.

- Hexadezimal 4A3 enthaelt die unteren 8 Bit der Mantisse.

Ist das Argument eine **Zahl doppelter Genauigkeit:**

- Hexadezimal 4A3 bis hexadezimal 4A6 sind gleich wie fuer Zahlen mit einfacher Genauigkeit.

- Hexadezimal 49F bis hexadezimal 4A2 enthaelt 4 weitere Byte der Mantisse (hexadezimal 49F enthaelt die untersten 8 Bits).

Beispiel:

```
10 DEF USR0 = &HF000
.
.
.
50 C = USR0(B/2)
60 D = USR(B/3)
```

Die Funktion USR0 wird in Zeile 10 definiert. In Zeile 50 wird die Funktion USR0 mit dem Argument B/2 aufgerufen. In Zeile 60 wird USR0 wieder aufgerufen. diesmal mit dem Argument B/3.

9.3.3. CALL

Ruft ein Unterprogramm in Maschinensprache auf.

Syntax: CALL numerische Variable
[(Variable[,Variable]...)]

Bemerkungen:

Numerische Variable
Name einer numerischen Variablen. Der Wert der Variablen beinhaltet die Startadresse des Unterprogrammms im Hauptspeicher, die als Relativzeiger in das laufende Segment des Hauptspeichers gerufen wird (wie mit der letzten Anweisung DEF SEG definiert).

Variable
Name einer Variablen, die als Argument an das Unterprogramm in Maschinensprache uebergeben wird.

Die Anweisung CALL ist ein Weg, eine Verbindung zwischen einem Unterprogramm in Maschinensprache und BASIC herzustellen. Der andere Weg ist die Funktion USR.

Beispiel:

```
100 DEF SEG = &H8000
110 UP = 0
120 CALL UP(A,B%,C)
```

Zeile 100 setzt das Segment auf die Adresse hexadezimal 80000. UP wird auf Null gesetzt, so dass der Aufruf von UP das Unterprogramm ab Adresse hexadezimal 80000 ausfuehrt. Die Variablen A, B%, und C werden als Argumente an das Unterprogramm in Maschinensprache uebergeben.

10. DOS-typische BASIC-Erweiterungen

10.1. Verzeichniszugriff

- MKDIR

Erstellt ein Verzeichnis auf einer angegebenen Diskette.

Syntax: MKDIR Pfad

Bemerkungen:

Pfad ist ein Zeichenkettenausdruck nicht laenger als 63 Zeichen, der das neu zu erstellende Verzeichnis angibt. Weitere Informationen stehen in Kapitel 6 unter "Namensgebung fuer Dateien".

Beispiele:

Erstellen des Unterverzeichnisses LAGER vom Basisverzeichnis aus.

```
MKDIR "LAGER"
```

Erstellen des Unterverzeichnisses LAGER1 unter dem Verzeichnis LAGER vom Basisverzeichnis aus.

```
MKDIR "LAGER\LAGER1"
```

Erstellen des Unterverzeichnisses HOLZ unter dem Verzeichnis LAGER1 vom Basisverzeichnis aus.

```
MKDIR "LAGER\LAGER1\HOLZ"
```

Erstellen eines Unterverzeichnisses PRODUKTION vom Basisverzeichnis.

```
MKDIR "PRODUKTION"
```

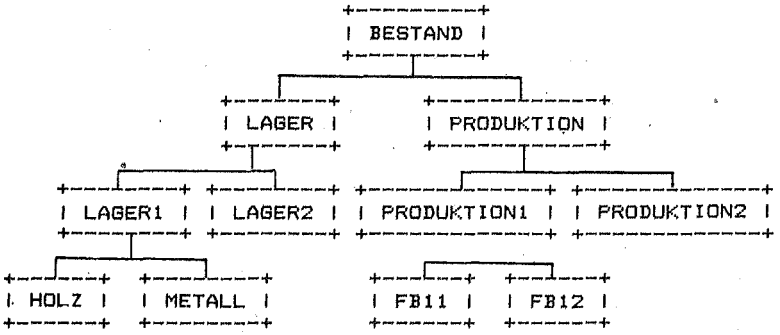
PRODUKTION wird laufendes Verzeichnis. Danach Erstellen der beiden Unterverzeichnisse **PRODUKTION1** und **PRODUKTION2.**

```
CHDIR "PRODUKTION"
MKDIR "PRODUKTION1"
MKDIR "PRODUKTION2"
```

Die gleiche Struktur haette man auch vom Basisverzeichnis aus erstellen koennen.

```
MKDIR "PRODUKTION\PRODUKTION1"
MKDIR "PRODUKTION\PRODUKTION2"
```

Durch die obigen Beispiele wurde die folgende Baumstruktur erzeugt:



=_CHDIR

Ermoeglicht die Aenderung des laufenden Verzeichnisses.

Syntax: **CHDIR Pfad**

Bemerkungen:

Pfad ist ein Zeichenkettenausdruck, der nicht laenger als 63 Zeichen sein darf und ein neues Verzeichnis angibt, das damit zum laufenden Verzeichnis wird. Weitere Informationen ueber Pfade stehen unter "Namensgebung fuer Dateien" in Kapitel 6.

Die folgenden Beispiele beziehen sich auf die oben gezeigte Baumstruktur.

Wechsel von einem Unterverzeichnis zum Basisverzeichnis:

```
CHDIR "\"
```

Aenderung vom Basisverzeichnis zum Verzeichnis **HOLZ**:

```
CHDIR "LAGER\LAGER1\HOLZ"
```

Aenderung vom Verzeichnis **PRODUKTION** zum Verzeichnis **PRODUKTION1**:

```
CHDIR "PRODUKTION1"
```

Aenderung vom Verzeichnis **LAGER1** zum Verzeichnis **LAGER**:

```
CHDIR ".."
```

Um **LAGER** zum laufenden Verzeichnis auf dem aktuellen Laufwerk A und **PRODUKTION** zum laufenden Verzeichnis auf Laufwerk C zu machen, muss folgendes eingegeben werden:

```
CHDIR "LAGER"
CHDIR "C:PRODUKTION"
```

Das Verzeichnis **PRODUKTION** muss auf Laufwerk C vorhanden sein. Wird nun **Dateiangabe** fuer Laufwerk A benutzt, gilt dies fuer die Dateien im Verzeichnis **LAGER**.
Wird Dateiangabe fuer Laufwerk C benutzt, gilt dies fuer die Dateien im Verzeichnis **PRODUKTION**.

RMDIR

Loescht das Verzeichnis von der angegebenen Diskette.

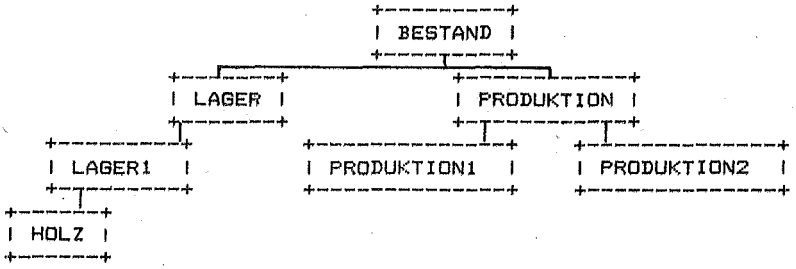
Syntax: RMDIR Pfad

Bemerkungen:

Pfad ist ein Zeichenkettenausdruck, nicht groesser als 63 Zeichen, der das Unterverzeichnis benennt, das aus dem bestehenden Verzeichnis geloescht werden soll. Weitere Informationen stehen in Kapitel 6 unter "Namensgebung fuer Dateien".

Das Verzeichnis darf keine Dateien und Unterverzeichnisse mehr enthalten, wenn es geloescht werden soll, mit Ausnahme der Eintragungen "." und "_", da sonst der Fehler "Path/file access error" (Pfad-/Dateizugriffsfehler) erscheint.

Beispiel:



Die folgenden Beispiele beziehen sich auf die obige Baumstruktur. Befindet man sich im Basisverzeichnis und will das Verzeichnis **HOLZ** loeschen:

```
RMDIR LAGER\LAGER1\HOLZ
```

Soll **PRODUKTION** das laufende Verzeichnis werden und das Verzeichnis **PRODUKTION2** geloescht werden:

```
CHDIR "PRODUKTION"
RMDIR "PRODUKTION2"
```

Ein anderer Weg, das Verzeichnis **PRODUKTION2** zu loeschen, ist das Basisverzeichnis zum laufenden Verzeichnis zu machen und **PRODUKTION2** zu loeschen:

```
CHDIR "\"
RMDIR "PRODUKTION\PRODUKTION2"
```

Das Verzeichnis vor dem laufenden kann nicht geloescht werden. Nehmen wir an, dass in der oberen Baumstruktur **LAGER1** das laufende Verzeichnis sei. Wird jetzt versucht, **LAGER** zu loeschen, erscheint die Fehlerausschrift "Path/file access error" (Pfad-/Dateizugriffsfehler). Versucht man mit dem Kommando **KILL** ein Verzeichnis zu loeschen, erscheint wieder der obige Fehler.

10.2. Laden von DCP-Programmen

- SHELL

Laedt eine andere Programmdatei und fuehrt das Programm aus (Dateien mit dem Dateityp **.COM**, **.BAT**, **.EXE**). Jedes Programm, das unter BASIC ausgefuehrt wird, wird als **"Kind-Prozess"** bezeichnet. Wenn die Ausfuehrung des Kind-Prozesses beendet ist, wird die Steuerung mit der auf die Anweisung **SHELL** folgenden Anweisung wieder an den **"Vater-Prozess"**, das BASIC-Programm, uebergeben.

Syntax: **SHELL[Befehlszeichenkette]**

Bemerkungen:

Befehlszeichenkette ist ein Zeichenkettenausdruck, der den Namen eines Programmes enthaelt, das ausgefuehrt werden soll, und wahlweise beliebige Parameter, die an dem Kind-Prozess uebergeben werden sollen.
Jedes Programm, das unter BASIC ausgefuehrt wird, wird als "Kind-Prozess" bezeichnet.
Kind-Prozesse werden von **SHELL** ausgefuehrt, indem eine Kopie von **COMMAND.COM** mit dem Schalter **/C** geladen und ausgefuehrt wird. Wird **COMMAND** in dieser Weise verwendet, werden alle Parameter richtig an die Standard-Dateisteuerbloecke uebergeben. Standard-Eingabe und -Ausgabe koennen umgeleitet werden und interne Befehle wie **DIR**, **PATH** und **SORT** koennen ausgefuehrt werden.
Wird **SHELL** ohne **Befehlszeichenkette** angegeben, wird eine Kopie von **COMMAND.COM** geladen, die DCP-Systemanfrage wird angezeigt und es kann jeder unter DCP gueltige Befehl eingegeben werden (z.B. **DIR**, **COPY**, **SORT** usw.). Die Rueckkehr zu BASIC erfolgt durch Eingabe von **EXIT**. Mit der Anweisung **SHELL** koennen auch Stapelverarbeitungsdateien aufgerufen werden. Um zum Vater-Prozess, dem BASIC-Programm, zurueckzukehren, muss die Stapelverarbeitungsdatei als letzte Anweisung **EXIT** enthalten.

Werden von einem BASIC-Programm aus mit Hilfe der Anweisung **SHELL** Kind-Prozesse ausgefuehrt, gibt es einige Regeln, die befolgt werden muessen, damit das Anwendungsprogramm richtig ausgefuehrt wird.
Um sicherstellen, dass bei der Rueckkehr zu BASIC auch der richtige Bildschirmmodus aktiv ist, kann folgendes getan werden:

- Den BIOS-Interrupt **10H**, Funktionsaufruf 15, verwenden, um den laufenden Bildschirmmodus zu speichern. Bei der Rueckkehr zu BASIC den Funktionsaufruf **0** verwenden, um den Bildschirmmodus wieder herzustellen, oder

- im BASIC-Programm unmittelbar nach der Anweisung **SHELL** die Anweisung **SCREEN**, gefolgt von **CLS**, ausfuehren.

Bevor BASIC die Anweisung **SHELL** ausfuehrt, werden alle verwendeten Interrupt-Vektoren gespeichert; dies garantiert jedoch nicht, dass alle vom Programm verwendeten Interrupt-Vektoren (die von BASIC nicht verwendet werden) auch gespeichert werden. Es sollte daher sichergestellt werden, dass alle Interrupt-Vektoren des Programmes gespeichert werden, damit das richtige Interface fuer DCP erhalten bleibt.

Ein Kind-Prozess, der eine Datei veraendert, die durch das BASIC-Anwendungsprogramm eroeffnet wurde, kann zu unvorhersehbaren Ergebnissen fuehren. Muss eine solche Datei geaendert werden, so muss die Datei vom BASIC-Programm aus geschlossen werden, bevor mit der Anweisung **SHELL** ein Kind-Prozess durchgefuehrt wird. Es ist auch zu beachten, dass Dateien, die waehrend der Umleitung von Standard-Eingabe und -Ausgabe eroeffnet wurden, wie **eroeffnete** Dateien behandelt werden, und dass diese Dateien nicht waehrend eines mit **SHELL** eingeleiteten Prozesses veraendert werden sollten.

Bevor BASIC die Anweisung **SHELL** ausfuehrt, wird der nicht benoetigte Speicherplatz freigesetzt, ausser wenn BASIC mit dem Parameter M aufgerufen wird. Wird der Parameter /M angegeben, geht BASIC davon aus, dass ein Programm in Maschinensprache in den ueber dem BASIC-Datensegment liegenden Speicher geladen wird. Dadurch kann BASIC vor der Ausfuehrung von **SHELL** den Arbeitsbereich nicht komprimieren und folglich kann fuer **SHELL** der Fehler "Out of memory" (Hauptspeicher reicht nicht aus) auftreten, wenn der Parameter M angegeben wurde.

Es empfiehlt sich bei Programmen , in denen die Anweisung **SHELL** verwendet wird, zunaechst die Unterprogramme in Maschinensprache zu laden, bevor BASIC geladen wird. Das bedeutet, dass in den Unterprogrammen ein Code eingefuegt werden muss, der ermoeglicht, dass die Unterprogramme beendet werden und resident bleiben (INT27H), wenn sie unter DCP aufgerufen werden.

Ein Programm, das durch die Anweisung **SHELL** ausgefuehrt wird, sollte nie beendet werden und resident bleiben, da sonst fuer BASIC nicht genuegend Speicherplatz zur Verfuegung steht, um den Arbeitsbereich wiederherzustellen. Alle Dateien werden in diesem Fall geschlossen, die Nachricht **"Can't continue after SHELL"** (Programm kann nach SHELL nicht fortgesetzt werden) wird angezeigt und es erfolgt die Rueckkehr zu DCP.

Waehrend der Ausfuehrung des Kind-Prozesses bleibt BASIC im Speicher. Nach Beendigung des Kind-Prozesses wird das BASIC-Programm fortgesetzt.

Der Programmname, der in der **Befehlszeichenkette** angegeben wird, kann einen beliebigen Dateityp haben. Wird kein Dateityp angegeben, sucht DCP zuerst nach dem Dateityp **.COM**, dann nach **.EXE** und zuletzt nach **.BAT**. Wird der **Dateiname** nicht gefunden, wird von **COMMAND** eine Fehlernachricht ausgegeben.

Weiterer Text nach dem Programmnamen in der **Befehlszeichenkette** (durch mindestens eine Leerstelle vom Programmnamen getrennt) wird als Programmparameter verarbeitet.

Bei der Ausfuehrung von BASIC wird die Umgebung von DCP uebernommen. Aenderungen, die durch das Anwendungsprogramm an der Umgebung vorgenommen werden, werden auf den Kind-Prozess uebertragen.

Hinweis:

Weitere Informationen ueber die Umgebung befinden sich im DCP-Handbuch.

SHELL kann nicht von DCP zu BASIC durchgefuehrt werden. Wird versucht, BASIC als Kind-Prozess auszufuehren, wird eine Fehlernachricht angezeigt. Nach dieser Fehlernachricht wird die Steuerung wieder an BASIC uebergeben.

Beispiel:

Im folgenden Beispiel wird eine Datei erstellt, zum DCP-Dienstprogramm **SORT** gewechselt und wieder zu BASIC zurueckgekehrt.

```
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
20 'schreibt die zu sortierenden Daten
.
.
.
100 CLOSE 1
110 SHELL "SORT <SORTIN.DAT >SORTOUT.DAT"
120 OPEN "SORTOUT.DAT" FOR INPUT AS #1
130 'verarbeitet die sortierten Daten
```

Im folgenden Beispiel wird das Verzeichnis der Diskette von BASIC aus angezeigt.

```
SHELL
A>DIR (auf Systemanfrage DCP-Befehl DIR eingeben)
A>EXIT (EXIT eingeben, um zu BASIC zurueckzukehren)
```

Das gleiche Ergebnis kann mit

```
SHELL "DIR"
```

erzielt werden.

10.3. Arbeit mit der BASIC-Umgebungstabelle

ENVIRON

Die Anweisung aendert Parameter in der BASIC-Umgebungstabelle. Mit **ENVIRON** kann der Parameter **"PATH"**(Pfad) fuer einen Kind-Prozess geaendert werden oder koennen Parameter durch Erstellen eines neuen Umgebungsparameters an Kind-Prozesse uebergeben werden. Siehe **ENVIRON$**, **SHELL** und DCP-Befehl **PATH**.

Syntax: ENVIRON Parameter=Zeichenkette

Bemerkungen:

Zeichenkette wird in folgender Form dargestellt:

"Parameter[=][Text][;]"

Parameter ist der Name eines Parameters, z.B. "PFAD".

Text ist ein Zeichenkettenausdruck, der den neuen Parameter definiert.

Parameter muss von Text durch ein Gleichheitszeichen(=) oder eine Leerstelle getrennt sein. **ENVIRON** interpretiert alle Angaben links des Gleichheitszeichens oder der Leerstelle als **Parameter**. Die erste Angabe, die auf das Gleichheitszeichen oder die Leerstelle nach Parameter folgt, wird als **Text** betrachtet.
Hat **Zeichenkette** die Laenge Null oder besteht sie nur aus einem Semikolon(;), wie:

```
"PFAD=;"
```

wird der Parameter in der Umgebungstabelle geloescht und die Tabelle komprimiert.

ENVIRON unterscheidet zwischen Gross- und Kleinbuchstaben (siehe auch Funktion **ENVIRON$**).

Ist der angegebene **Parameter** nicht vorhanden, wird der neue Parameter an das Ende der Umgebungstabelle angefuegt.

Besteht der angegebene **Parameter** bereits, wird er geloescht, die Umgebungstabelle wird komprimiert und **Parameter** an das Ende der Tabelle angefuegt.

Hinweis:

Beim Aufruf von BASIC entspricht die Groesse seiner Umgebungstabelle der Groesse der DCP-Umgebungstabelle (aufgerundet auf den naechsten 16-Byte-Abschnitt). Die BASIC-Umgebungstabelle kann nur vergroessert werden, wenn zuvor die DCP-Umgebungstabelle entsprechend vergroessert wurde.

Mit **ENVIRON** kann der Parameter **"PFAD"** fuer einen Kind-Prozess geaendert werden oder koennen Parameter durch Erstellen eines neuen Umgebungsparameters an Kind-Prozesse uebergeben werden.

Beispiel:

Durch die folgende Anweisung kann ein Standardpfad zum Basisverzeichnis von Laufwerk A angegeben werden:

```
ENVIRON "PFAD=A:\"
```

Anschliessend kann DCP mit Hilfe der Anweisung **SHELL** vom BASIC-Programm aus aufgerufen und ein beliebiger DCP-Befehl eingegeben werden. Wird eine Diskettendatei zur Ausfuehrung des Befehls benoetigt, sucht DCP im Basisverzeichnis von Laufwerk A nach ihr (**.COM**, **.EXE** oder **.BAT**), falls sie sich nicht auf dem aktuellen Laufwerk oder im aktuellen Verzeichnis befindet.

```
SHELL          COMMAND.COM aufrufen

A> REM wechselt zu Verzeichnis "ARBEIT" auf Laufwerk B
A> CD B:\ARBEIT
B> REM laedt PROJ unter DEBUG, obwohl kein Laufwerk angegeben ist. DEBUG und PROJ befinden sich auf verschiedenen Laufwerken.
B> DEBUG PROJ
   .
   .
   .
B> REM Zurueck zum BASIC-Programm
B> EXIT  (verlaesst DCP und kehrt zum BASIC-Programm zurueck)
```

Ein neuer Parameter kann wie folgt an die Umgebungstabelle angefuegt werden:

```
ENVIRON  "HILFE = C:\HILFE" 'definiert den Dateiparameter
                             "HILFE"
CHDIR  ENVIRON$  ("HILFE")  'aendert das Verzeichnis in
                             "HILFE"
```

Dieser Parameter kann folgendermassen wieder aus der Tabelle geloescht werden:

```
ENVIRON "HILFE=;"   loescht den Parameter "Hilfe" aus der
                    Tabelle
```

Die durch die BASIC-Anwendung geschaffene Umgebung wird an **COMMAND.COM** uebergeben, wenn diese Datei durch die Anweisung **SHELL** aufgerufen wird. Hierdurch koennen Parameter ueber die Umgebungstabelle von einem "Vater" (BASIC) an ein "Kind" uebergeben werden.

Hinweis:

Diesbezuegliche Informationen stehen auch unter Funktion **ENVIRON$** und Anweisung **SHELL** in diesem Handbuch bzw. unter dem Befehl **SET** und dem Funktionsaufruf **EXEC** im DCP-Handbuch.

=_ENVIRON$

Die Funktion zeigt eine angegebene Zeichenkette der BASIC-Umgebungstabelle an.

Syntax: **v$=ENVIRON$(Parameter$)**
oder
v$=ENVIRON$(n)

Bemerkungen:

Parameter$ ist ein Zeichenkettenausdruck, der den anzuzeigenden **Parameter** enthaelt.

n ist ein ganzzahliger Ausdruck im Bereich 1 bis 255.

Ist **Parameter$** angegeben, so zeigt **ENVIRON$** aus der Umgebungstabelle eine Zeichenkette an, die den auf **Parameter$** folgenden Text enthaelt. Ist **Parameter$** nicht vorhanden oder folgt auf das Gleichheitszeichen kein Text, wird eine leere Zeichenkette angezeigt.

Wird ein Zahlenwert angegeben, zeigt **ENVIRON$** eine Zeichenkette, die den n. **Parameter$** aus der Umgebungstabelle enthaelt, zusammen mit dem **Parameter$=Text** an. Ist n. **Parameter$** nicht vorhanden, wird eine leere Zeichenkette angezeigt.

ENVIRON$ unterscheidet zwischen Gross- und Kleinbuchstaben. Wird der Tabelle folgender Ausdruck hinzugefuegt:

```
ENVIRON$ "load = high"
```

und soll ueberprueft werden, ob die Operation erfolgreich durchgefuehrt werden konnte, kann die Funktion **ENVIRON$** wie folgt benutzt werden:

```
PRINT ENVIRON$("load")
```

Wird jedoch

```
PRINT ENVITON$("LOAD")
```

eingegeben, wird eine leere Zeichenkette angezeigt, da die Funktion den Wert in Grossbuchstaben nicht erkennt.

Beispiel:

Beim einleitenden Programmladen von DCP wird dem Parameter **"COMSPEC"** ein Wert zugeordnet, der die Position der **COMMAND.COM**-Datei angibt, ausserdem wird ein Nullpfad definiert. Um den Inhalt der Umgebungstabelle direkt nach dem Laden anzuzeigen, wird folgendes eingegeben:

```
PRINT ENVIRON$(1)
```

Nun erscheint folgende Anzeige:

```
PATH=
```

Wird folgendes angegeben:

```
PRINT ENVIRON$(2)
```

so wird die Nachricht:

```
COMSPEC = A:\COMMAND.COM
```

angezeigt.

Auf die Anweisung

```
PRINT ENVIRON$("COMSPEC")
```

wird folgendes angezeigt:

```
A:\COMMAND.COM
```

Das folgende Programm speichert die BASIC-Umgebungstabelle in einem Bereich, so dass sie fuer ein Kind-Prozess veraendert werden kann. Nach Ausfuehrung des Kind-Prozesses wird die urspruengliche Umgebung wiederhergestellt.

Beispiel:

```
 10 DIM TABLE$(10) 'nicht mehr als 10 Parameter
 20 PARMS=1 'urspruengliche Anzahl Parameter
 30 WHILE LEN(ENVIRON$(PARMS))>0
 40 TABLE$(PARMS) = ENVIRON$(PARMS)
 50 PARMS = PARMS+1
 60 WEND
 70 PARMS = PARMS-1 'richtige Zahl
 80 'neue Umgebung speichern
 90 ENVIRON "DATAIN = C:\DATAIN\INP.FIL"
100 ENVIRON "SORT.DAT = SORT.DAT<"+
    ENVIRON$("DATAIN") + ">LPT1:"
  :
  :
  :
1000 SHELL ENVIRON$("SORT.DAT") 'Daten sortieren
1010 FOR I=1 TO PARMS
1020 ENVIRON TABLE$(I) 'Parameter wiederherstellen
1030 NEXT I
  :
  :
  :
```

Hinweis:

Diesbezuegliche Informationen stehen auch unter Anweisung **ENVIRON** und Anweisung **SHELL** in diesem Handbuch bzw. unter dem Befehl **SET** und dem Funktionsaufruf **EXEC** im DCP-Handbuch.

10.4. Verarbeitung von Steuerzeichen bei benutzereigenen Einheitentreibern

=_IOCTL

Die Anweisung ermoeglicht, eine Steuerdatenzeichenkette jederzeit an einen Einheitentreiber zu senden, nachdem dieser mit **OPEN** eroeffnet wurde.

Syntax: **IOCTL[#]Dateinummer,Zeichenkette**

Bemerkungen:

Dateinummer ist die Dateinummer fuer den Einheitentreiber.

Zeichenkette ist ein Zeichenkettenausdruck, der die Steuerdaten enthaelt.

Das Dateiein-/ausgabesystem in BASIC ermoeglicht es, benutzereigene Einheitentreiber zu erstellen und zu installieren.

Durch die Anweisung **IOCTL** und die Funktion **IOCTL$** koennen Steuerdaten vom Einheitentreiber gelesen und zum Einheitentreiber gesendet werden.

Eine **IOCTL**-Befehlszeichenkette kann bis zu 255 Byte lang sein. Mehrere Befehle innerhalb der Zeichenkette koennen durch Semikolons getrennt werden:

```
"LF;PL66;LW132"
```

Inhalt und Format der Steuerdatenzeichenkette koennen vom Benutzer festgelegt werden, wobei die moeglichen Befehle durch die Charakteristika des installierten Treibers bestimmt sind.

Beispiel:

Urspruenglich sind die Zeicheneinheitentreiber fuer **LPT1:** **LPT2:** und **LPT3:** installiert, sie koennen aber ersetzt werden. Wird z.B. der neue Treiber **LPT1** installiert, um **LPT1:** zu ersetzen, koennte folgende Befehlszeichenkette zum Setzen oder Aendern der Seitenlaenge benutzt werden:

"PLn" (wobei n die Seitenlaenge darstellt)

Anschliessend kann der neue Treiber **LPT1** eroeffnet und die Seitenlaenge wie folgt festgelegt werden:

```
OPEN "LPT1" FOR OUTPUT AS #1
IOCTL #1, "PL60"
```

Es koennte auch ein Einheitentreiber geschrieben werden, der den Bildschirm steuert, den Bildschirmmodus auf Farbe setzt und die Bildschirmbreite setzt:

```
OPEN "OPT" FOR OUTPUT AS #2
IOCTL #2, "CL:W40"
```

Wenn der neue Treiber die Befehle **"CL"** zum Aendern des Bildschirms in Farbe und **"Wn"** zum Setzen der Bildschirmbreite annimmt, werden diese Befehle an den Treiber uebergeben, und ueber den Bildschirm wird eine Antwort angezeigt.

Hinweis:

Diesbezuegliche Angaben stehen auch unter Funktion **IOCTL$** in diesem Handbuch, im Abschnitt "Einheitenunterstuetzung" im **BASIC-Handbuch** und im Abschnitt ueber Einheitentreiber im Handbuch fuer DCP.

IOCTL$

Die Funktion liest eine Steuerdatenzeichenkette von einem eroeffneten Einheitentreiber.

Syntax: v$ = IOCTL$(# Dateinummer)

Bemerkungen:

Dateinummer ist die Nummer fuer die Datei, die fuer die Einheit eroeffnet wurde.

Die Funktion **IOCTL$** kann benutzt werden, um festzustellen, ob ein **IOCTL**-Befehl erfolgreich ausgefuehrt werden konnte.
Sie kann auch benutzt werden, um Angaben zur Einheitenkonfiguration, wie z.B. Einheitenbreite, zu erhalten.

Beispiel:

Im folgenden Beispiel wird ueberprueft, ob Steuerdaten empfangen werden konnten.

```
10 OPEN "COM" AS #1
20 IOCTL #1, "SW132;GW"
30 IF IOCTL$(1) = "132"
   THEN PRINT "BREITE KONNTE GESETZT WERDEN"
```

Wenn der Eiheitentreiber **"COM"** auf die **IOCTL$**-Anfrage nicht 132 uebergibt, wurde der Befehl nicht erfolgreich verarbeitet. Der Benutzer sollte ueberpruefen, ob Fehler vorliegen.

Wenn ein Einheitenfehler auftritt, sollten die Systemvariablen **ERDEV** und **ERDEV$** ueberprueft werden (sh. Pkt. 10.5.).

10.5. Ermittlung von Einheitenfehlern

- ERDEV und ERDEV$

Variablen, die nur gelesen werden koennen. Sie enthalten den Unterbrechungsfehlercode **INT24** fuer einen Einheitenfehler und den Namen der Einheit, die den Fehler verursachte.

Syntax:

```
v = ERDEV
v$ = ERDEV$
```

Bemerkungen:

ERDEV ist eine Variable, die nur gelesen werden kann. Wenn ein Fehler in DOS festgestellt wird, speichert **ERDEV** den Unterbrechungsfehlercode INT24 in den unteren 8 Bits, waehrend die oberen 8 Bits die Bits 15, 14, 13, 3, 2, 1 und 0 des Attributworts des Einheitenleitsatzes im folgenden Format enthalten, wobei die unbenutzten Bits (XX) Null sind.

15	14	13	XX	3	2	1	0

ERDEV$** ist ebenfalls eine Variable, die nur gelesen werden kann. Trat der Fehler bei einer Einheit auf, die Zeichen verarbeitet, enthaelt **ERDEV$ den Namen dieser Einheit in 8 Byte gespeichert. Trat der Fehler bei einer anderen Einheit auf, enthaelt **ERDEV$** den Blockeinheitennamen in 2 Byte gespeichert (A:,B:,C: usw.).

Beispiel:

Verriegelung des Diskettenlaufwerks B oeffnen und folgendes eingeben:

```
FILES "B:"
```

Es erscheint folgende Anzeige:

```
Disk not ready
```

Anschliessend

```
PRINT ERDEV, ERDEV$
```

eingeben, worauf **BASIC**

```
2        B:
```

anzeigt.

Bei Eingabe von

```
PRINT HEX$(16386)
```

wird

```
4 0 0 2
```

angezeigt.

BASIC uebergibt 02 an die unteren 8 Bits.

Hinweis:

Im Handbuch fuer DCP steht unter der Auflistung der **INT24**-Fehlercodes als Fehler 2 "Drive not ready". Die oberen 8 Bits (die Wortattribut-Bits) haben alle den Wert hexadezimal 40. Im Handbuch fuer DCP wird im Abschnitt ueber installierbare Einheitentreiber (Kapitel ueber Attributfelder) die Bedeutung der einzelnen Bits erklaert. Siehe auch Anweisung **IOCTL** und Funktion **IOCTL$**.

Mit folgendem Beispiel wird ein Druckerfehler simuliert:

```
10 CLS
20 ON ERROR GOTO 60
30 LPRINT "Der Drucker ist eingeschaltet"
40 PRINT "Der Drucker ist eingeschaltet"
50 END
60 V$=HEX$(ERDEV)
70 PRINT "ERDEV =";V$
80 B$=ERDEV$
90 PRINT "ERDEV$= ";B$
100 RESUME 50
```

Ist der Drucker ausgeschaltet, wird folgende Nachricht angezeigt:

```
ERDEV = 8009
ERDEV$ = LPT1
```

Die unteren 8 Bits (0 bis 7) des gleichwertigen binaeren Ausdrucks entsprechen 9, was dem Fehlercode INT24 fuer "Printer out of paper" (Kein Papier mehr im Drucker) entspricht. Die Bedeutung der Bits 13, 14 und 15 in der Variablen **ERDEV** wird im Handbuch fuer DCP im Abschnitt ueber installierbare Einheitentreiber (Kapitel ueber Attributfelder) erklaert.

Liste der Kommandos, Anweisungen und Funktionen in alphabetischer Reihenfolge

Syntax	Wirkungsweise	Seite
ABS(x)	Uebergibt den absoluten Wert des Ausdrucks x.	125
ASC(x$)	Uebergibt den ASCII-Code fuer das erste Zeichen der Zeichenkette x$.	133
ATN(x)	Uebergibt den Arcustangens von x.	132
AUTO [Nummer] [,[Schrittweite]]	Erzeugt automatisch Zeilennummern.	54
BEEP	Der Lautsprecher erzeugt einen Ton.	74
BLOAD Dateiangabe [,Relativzeiger]	Laedt eine Datei mit einem Hauptspeicherabbild (z.B. Programme in Maschinensprache) in den Hauptspeicher.	231
BSAVE Dateiangabe ,Relativzeiger,Laenge	Speichert Teile des Hauptspeichers in der angegebenen Datei.	232
CALL Numerische Variable [(Variable[, Variable]...)]	Ruft ein Unterprogramm in Maschinensprache auf.	240
CDBL(x)	Wandelt x in eine Zahl doppelter Genauigkeit um.	128
CHAIN [MERGE] Dateiangabe [,[Zeile][,ALL] [,DELETE Bereich]]	Ruft ein anderes Programm auf, uebertraegt die Steuerung an dieses und uebergibt Variable aus dem laufenden Programm.	113
CHDIR Pfad	Ermoeglicht das Aendern des laufenden Verzeichnisses.	242
CHR$(n)	Uebergibt das Zeichen mit dem ASCII-Code.	136
CINT(x)	Wandelt x in eine ganze Zahl um durch Runden.	127

CIRCLE (x,y),r[,Farbe [,Start,End[,Bild]]]	Zeichnen einer Ellipse auf dem Bildschirm mit dem Mittelpunkt (x,y) und dem Radius r.	199
CLEAR [,[n][,m]]	Loescht die Variablen, wahlfrei kann die Groesse des Arbeitsspeichers und des Stapelbereiches definiert werden.	60
CLOSE [[#]Dateinummer,...]	Schliesst die Ein-/Ausgabe fuer eine Einheit oder Datei ab.	157 177
CLS	Loescht den Bildschirm.	187
COLOR [Vordergrund] [,[Hintergrund] [,Bildschirmrand]]	Setzt im Textmodus die Farben fuer den Vordergrund, Hintergrund und den Bildschirmrand.	192
COLOR [Hintergrund] [,[Palette]]	Setzt im Grafikmodus die Farbe fuer den Hintergrund und waehlt eine von zwei Paletten aus.	195
COM(n) ON/OFF/STOP	Aktiviert oder inaktiviert die Unterbrechung fuer Datenfernverbindungsaktivitaeten.	180
COMMON Variable	Uebergibt Variable an das durch CHAIN aufgerufene Programm.	115
CONT	Nimmt die Programmausfuehrung nach einer Unterbrechung wieder auf.	62
COS(x)	Errechnet und uebergibt den Cosinuswert von x.	131
CSNG(x)	Wandelt x in eine Zahl einfacher Genauigkeit um.	128
CSRLIN	Uebergibt die vertikale Koordinate des Kursors.	191
CVI(2-Byte-Zeichenkette) CVS(4-Byte-Zeichenkette) CVD(8-Byte-Zeichenkette)	Wandelt die Zeichenkette in einen numerischen Wert um.	168

DATA Konstante [,Konstante]...	Speichert numerische und Zeichenkettenkonstanten, die mit der Anweisung READ im Programm gelesen werden koennen.	96
DATE$	Liest das Datum.	72
DATE$=x$	Setzt das Datum.	72
DEF FNName[(Arg[,Arg]...)] = Ausdruck	Definiert eine Funktion.	112
DEF SEG [=Segment]	Definiert das laufende Segment des Hauptspeichers.	228
DEFTyp Buchstabe[-Buchstabe] [,Buchstabe [-Buchstabe]]...	Definiert Variablentypen als ganzzahlig, einfache Genauigkeit, doppelte Genauigkeit oder Zeichenketten. Wobei Typ sein kann: INT, SNG, DBL, STR.	74
DEF USR(n)=Relativzeiger	Definiert die Startadresse eines Unterprogramms in Maschinensprache.	237
DELETE [Zeile1][-Zeile2] DELETE [Zeile1-]	Loescht den angegebenen Programmzeilenbereich.	57
DIM Variable(Indizes) [,Variable(Indizes)]...	Definiert die maximalen Werte fuer die Indizes von Bereichsvariablen und ordnet den erforderlichen Speicherplatz zu.	109
DRAW Zeichenkette	Zeichnet ein Objekt, das durch eine Zeichenkette angegeben ist.	201
EDIT Zeile	Zeigt eine Programmzeile zur Aenderung an.	55
END	Beendet die Programmausfuehrung, schliesst alle Dateien ab und kehrt zur Befehlsebene zurueck.	99
ENVIRON Parameter=Zeichenkette	Aendert Parameter in der BASIC-Umgebungstabelle.	247
ENVIRON$ (Parameter$) ENVIRON$ (n)	Zeigt eine angegebene Zeichenkette der BASIC-Umgebungstabelle an.	249

EOF (Dateinummer)	Zeigt eine Dateiendbedingung an.	169
ERASE Feldname [,Feldname]...	Loescht Felder aus einem Programm.	111
ERDEV	Variable, die den Unterbrechungsfehlercode INT24 fuer einen Einheitenfehler enthaelt.	254
ERDEV$	Variable, die den Namen der Einheit enthaelt, die den Fehler verursachte.	254
ERL	Uebergibt die Zeilennummer, in der der letzte Fehler aufgetreten ist.	117
ERR	Uebergibt den Fehlercode fuer den letzten Fehler.	117
ERROR n	Simuliert das Auftreten eines BASIC-Fehlers oder ermoeglicht die Definition eigener Fehlercodes.	115
EXP(n)	Errechnet die Exponentialfunktion.	129
FIELD [#]Dateinummer, Laenge AS Zeichenkettenvariable [,Laenge AS Zeichenkettenvariable]...	Legt Platz fuer Variable in einem Puffer fuer Dateien mit wahlfreiem Zugriff an.	164
FILES [Dateiangabe]	Zeigt die Dateinamen auf einer Diskette an.	68
FIX(x)	Schneidet x auf eine ganze Zahl ab.	126
FOR Variable=x TO y [STEP z]	Fuehrt eine Folge von Anweisungen in einer Schleife aus. Die Anweisung NEXT schliesst die Schleife ab.	103
FRE(x) FRE(x$)	Uebergibt die Anzahl der Byte im Hauptspeicher, die nicht von BASIC benutzt werden.	145
GET [#]Dateinummer [,Nummer]	Liest einen Satz aus einer Datei mit wahlfreiem Zugriff in einen Puffer fuer wahlfreien Zugriff.	167

GET (x1,y1)-(x2,y2), Bereich	Liest Punkte aus einem Bereich des Bildschirms.	204
GOSUB Zeile	Verzweigen in ein Unterprogramm. Mit Hilfe der Anweisung RETURN erfolgt die Rueckkehr aus dem Unterprogramm.	100
GOTO Zeile	Unbedingte Verzweigung aus dem normalen Programmablauf zu einer angegebenen Zeilennummer.	99
HEX$(n)	Wandelt den dezimalen Wert n in eine hexadezimale Zeichenkette um.	136
IF Ausdruck[,] THEN Angabe [ELSE Angabe] IF Ausdruck[,] GOTO Zeile [[,]ELSE Angabe]	Aendert den Programmablauf, abhaengig vom Ergebnis eines Ausdrucks.	106
INKEY$	Liest ein Zeichen von der Tastatur.	142
INP(n)	Uebergibt ein vom Tor n gelesenes Byte.	236
INPUT[;][“Anfrage”;] Variable[,Variable]...	Liest Daten von der Tastatur.	72
INPUT #Dateinummer,Variable [,Variable]...	Liest Daten aus einer sequentiellen Einheit oder Datei und weist sie Programmvariablen zu.	161
INPUT$(n[,[#]Dateinummer])	Uebergibt eine Zeichenkette von n Zeichen, die von der Tastatur oder von einer Datei mit der Nummer Dateinummer gelesen wurde.	143
INSTR([n,]x$,y$)	Sucht das erste Auftreten der Zeichenkette y$ in x$ und uebergibt die Position, an der die Zeichenkette gefunden wurde. n setzt die Position fest, an der mit der Sucher begonnen werden soll.	134
INT(x)	Uebergibt die groesste ganze Zahl, die kleiner oder gleich x ist.	125

IOCTL [#]Dateinummer, Zeichenkette	Sendet eine Steuerdaten-zeichenkette an einen Einheitentreiber.	251
IOCTL$([#]Dateinummer)	Liest eine Steuerdaten-zeichenkette von einem eroeffneten Einheiten-treiber.	253
KEY ON/OFF/LIST	Zeigt den Inhalt der Funk-tionstasten an oder loescht die Anzeige.	88
KEY n,x$	Ordnet den Wert x$ der an-gegebenen Funktionstaste zu.	88
KEY n, CHR$(Tastenkenn-zeichen) + CHR$ (Suchcode)	Definiert zusaetzliche Funk-tionstasten.	88
KEY(n) ON/OFF/STOP	Aktiviert oder inaktiviert die Unterbrechung fuer eine angegebene Taste in einem BASIC-Programm.	120
KILL Dateiangabe	Loescht eine Datei von der Diskette.	69
LEFT$(x$,n)	Uebergibt die am weitesten links liegenden n Zeichen aus x$.	138
LEN(x$)	Uebergibt die Anzahl der Zeichen in x$.	134
LET Variable=Ausdruck	Ordnet den Wert eines Aus-drucks einer Variablen zu.	75
LINE [(x1,y1)]-(x2,y2) [,[Farbe][,B[F]] [,Stil]]	Zeichnet eine Linie oder ein Viereck auf dem Bildschirm.	197
LINE INPUT[;][“Anfrage”;] Zeichnekettenvariable	Liest eine Zeile von der Tastatur in eine Zeichen-kettenvariable.	80
LINE INPUT # Dateinummer, Zeichenkettenvariable	Liest eine Zeile von einer sequentiellen Datei in eine Zeichenkettenvariable.	162
LIST [Zeile1][-[Zeile2]] [,Dateiangabe]	Listet das im Hauptspeicher stehende Programm auf dem Bildschirm oder auf einer anderen angegebenen Ein-heit auf.	56

LLIST [Zeile1][-[Zeile2]]	Listet das gesamte oder einen Teil des Programms auf dem Drucker aus.	57
LOAD Dateiangabe[,R]	Laedt ein Programm von einer angegebenen Einheit in den Hauptspeicher und fuehrt es wahlweise aus.	64
LOC(Dateinummer)	Uebergibt die aktuelle Position in der Datei.	170
LOCATE[Zeile][,Spalte] [,[Anzeige][,[Start] [,Stopp]]]]	Setzt den Kursor auf dem aktiven Bildschirm. Wahlfreie Parameter schalten das Blinken des Kursors an und aus und definieren die Groesse des blinkenden Kursors.	190
LOF(Dateinummer)	Uebergibt die Anzahl Bytes, die einer Datei zugeordnet sind (Laenge der Datei).	171
LOG(x)	Uebergibt den natuerlichen Logarithmus von x.	129
LPOS(n)	Uebergibt die laufende Position des Druckkopfs im Druckpuffer.	148
LPRINT [Liste von Ausdruecken[;]]	Gibt Daten auf dem Drucker aus.	86
LPRINT USING v$; Liste von Ausdruecken[;]	Gibt Daten auf dem Drucker aus mit Hilfe des durch v$ angegebenen Formates.	86
LSET Zeichenkettenvariable=x$	Uebergibt Daten in einen Dateipuffer fuer Direktzugriff (linksbuendig).	165
MERGE Dateiangabe	Mischt Zeilen aus einer ASCII-Programmdatei in ein Programm, das sich im Hauptspeicher befindet.	67
MID$(x$,n[,m])	Uebergibt m Zeichen aus x$, beginnend an der Position n.	139
MKDIR Pfad	Erstellt ein Verzeichnis auf einer angegebenen Diskette.	241

MKI$(ganzzahliger Ausdruck) MKS$(Ausdruck einfacher Genauigkeit) MKD$(Ausdruck doppelter Genauigkeit)	Wandelt numerische Werte in Zeichenkettenwerte um.	168
NAME Dateiangabe AS Dateiname	Aendert den Namen einer Diskettendatei.	68
NEW	Loescht das im Hauptprogramm befindliche Programm und alle Variablen.	53
NEXT[Variable[,Variable]..]	Schliesst eine FOR...NEXT-Schleife ab.	103
OCT$(n)	Uebergibt eine Zeichenkette, die den oktalen Wert eines dezimalen Arguments darstellt.	137
ON COM(n) GOSUB Zeile	Setzt fuer BASIC eine Zeilennummer, zu der verzweigt wird, sobald eine Information im Datenfernverarbeitungspuffer steht.	181
ON ERROR GOTO Zeile	Aktiviert die Fehlerunterbrechung und gibt die erste Zeile des Fehlerbehandlungsprogramms an.	118
ON n GOTO Zeile[,Zeile]... ON n GOSUB Zeile[,Zeile]...	Verzweigt zu einer von mehreren angegebenen Zeilennummern, abhaengig vom Wert des Ausdrucks.	101
ON KEY(n) GOSUB Zeile	Aktiviert die Unterbrechung, falls eine angegebene Funktions- oder Kursortaste betaetigen wurde und gibt die erste Zeile der Unterbrechungsroutine an.	121
ON PLAY(n) GOSUB Zeile	Erlaubt fortlaufende Musik waehrend der Programmausfuehrung.	225
ON TIMER(n) GOSUB Zeile	Uebergibt nach Ende einer angegebenen Zeitspanne die Steuerung an eine angegebene Zeilennummer.	123

OPEN Dateiangabe [FOR Modus] AS [#]Dateinummer [LEN=Satzlaenge] OPEN Modus2,[#]Dateinummer, Dateiangabe[,Satz-laenge]	Erlaubt die Ein-/Ausgabe zu und von einer Datei oder Einheit.	153
OPEN "COMn:[Geschwindigkeit] [,Paritaet][,Daten] [,Stopp][,RS][,CS[n]] [,DS[n]][,CD[n]][,LF] [,PE]" AS[#]Dateinummer [LEN=Nummer]	Eroeffnet eine Datei fuer Datenfernverarbeitung.	173
OPTION BASE n	Definiert den kleinsten Wert fuer Feldindizes.	110
OUT n,m	Sendet ein Byte zu einem Maschinenausgabetor.	235
PAINT (x,y)[[,Farbe][,Rand] [,Hintergrund]]	Fuellt einen Bildschirmbe-reich mit einer ausge-waehlten Farbe.	209
PEEK(n)	Uebergibt ein gelesenes Byte von der angegebenen Hauptspeicherposition.	230
PLAY Zeichenkette	Spielt Musik mit Hilfe einer Zeichenkette.	221
PLAY(n)	Uebergibt die Anzahl der Noten, die sich gerade im Puffer fuer Hintergrund-musik befinden.	227
PLAY ON/OFF/STOP	Aktivieren/Inaktivieren der Programmunterbrechung.	225
PMAP(x,n)	Rechnet physikalische Koordinaten in Weltko-ordinaten und Weltkoordi-naten in physikalische Koordinaten um.	216
POINT(x,y) POINT(n)	Uebergibt die Farbe eines angegebenen Punktes auf dem Bildschirm oder die laufende grafische Koordinate oder den Wert der laufenden x- oder y-Koordinaten.	217
POKE n,m	Schreibt ein Byte in eine Hauptspeicherstelle.	230

POS(n)	Uebergibt die Spalten-position, in der sich der Kursor gerade befindet.	191
PRINT [Liste der Ausdruecke] [;] ? [Liste der Ausdruecke][;]	Zeigt Daten auf dem Bild-schirm an.	81
PRINT USING v$ Liste der Ausdruecke[;]	Gibt Daten mit Hilfe eines Formates aus.	82
PRINT #Dateinummer, [USING v$;] Liste der Ausdruecke	Schreibt Daten sequentiell in eine Datei.	158
PSET(x,y)[,Farbe] PRESET(x,y)[,Farbe]	Zeichnet einen Punkt an eine angegebene Stelle auf dem Bildschirm.	196
PUT [#] Dateinummer [,Nummer]	Schreibt einen Satz aus dem Puffer fuer Direktzugriff in eine Direktzugriffsdatei.	166
PUT (x,y), Bereich[,Aktion]	Faerbt einen angegebenen Be-reich des Bildschirms.	205
RANDOMIZE[n] RANDOMIZE TIMER	Uebergibt einen neuen An-fangswert fuer den Zufalls-zahlengenerator.	76
READ Variable[,Variable]...	Liest Werte aus der An-weisung DATA und ordnet sie Variablen zu.	95
REM Bemerkung	Einfuegen erklaerender Be-merkungen in ein Programm.	71
RENUM [neue Nummer] [,[alte Nummer] [,Schrittweite]]	Neunumerierung von Programm-zeilen.	58
RESET	Schliesst alle Disketten-dateien ab und loescht den Systempuffer	64
RESTORE [Zeile]	Erlaubt es, DATA-Anwei-sungen ab einer angegebenen Zeile wieder zu lesen.	97
RESUME [0] RESUME NEXT RESUME Zeile	Faehrt mit der Programmaus-fuehrung fort, nachdem eine Fehlerbehandlungsroutine ausgefuehrt wurde.	119
RETURN [Zeile]	Verzweigt aus einem Unter-programm zurueck.	103

RIGHT$(x$,n)	Uebergibt die rechten n Zeichen der Zeichenkette x$.	139
RMDIR Pfad	Loescht das Verzeichnis von der angegebenen Diskette.	243
RND[(x)]	Uebergibt eine Zufallszahl zwischen 0 und1.	145
RSET Zeichenkettenvariable =x$	Uebergibt Daten in einen Dateipuffer fuer Direktzugriff (rechtsbuendig).	165
RUN [Zeile] RUN Dateiangabe[,R]	Beginnt die Programmausfuehrung.	59
SAVE Dateiangabe[,A] SAVE Dateiangabe[,P]	Speichert eine BASIC-Programmdatei auf Diskette.	65
SCREEN (Zeile,Spalte[,z])	Uebergibt den ASCII-Code (0 bis 255) fuer ein' Zeichen auf dem aktiven Bildschirm an der angegebenen Zeile und Spalte.	218
SCREEN [Modus][[Aendern] [,[,Aseite] [,Vseite]]]	Setzt die Bildschirmattribute, die von den nachfolgenden Anweisungen benutzt werden sollen.	188
SGN(x)	Uebergibt das Vorzeichen von x.	126
SHELL [Befehlszeichenkette]	Laedt eine andere Programmdatei und fuehrt das Programm aus.	244
SIN(x)	Errechnet und uebergibt den Sinuswert fuer x (x im Bogenmass).	130
SOUND Freq., Dauer	Erzeugt Toene ueber den Lautsprecher.	219
SPACE$(n)	Uebergibt eine Zeichenkette, die aus n Leerstellen besteht.	141
SPC(n)	Druckt in einer PRINT- oder LPRINT-Anweisung n Leerstellen.	147
SQR(x)	Uebergibt die Quadratwurzel von x.	130

STOP	Beendet die Programmausfuehrung und kehrt zur Befehlsebene zurueck.	98
STR$(x)	Uebergibt die Zeichenkettendarstellung des Wertes x.	137
STRING$(n,m) STRING$(n,x$)	Uebergibt eine Zeichenkette der Laenge n, deren Zeichen alle aus dem ASCII-Code m oder dem ersten Zeichen von x$ bestehen.	141
SWAP Variable1,Variable2	Tauscht die Werte zweier Variablen aus.	76
SYSTEM	Verlaesst BASIC und kehrt zu DOS zurueck.	64
TAB(n)	Setzt in Verbindung mit der Anweisung PRINT an der Position n einen Tabulator.	146
TAN(x)	Uebergibt den Tangens von x.	132
TIME$	Uebergibt die laufende Zeit.	73
TIME$=x$	Setzt die laufende Zeit.	73
TIMER	Uebergibt einen Wert einfacher Genauigkeit, der die Sekunden darstellt, die seit Mitternacht oder einem Warmstart des Systems vergangen sind.	144
TRON/TROFF	Zeigt eine Ablaufverfolgung der Programmanweisung bei der Ausfuehrung des Programms an.	63
USR[n](arg)	Ruft das angegebene Unterprogramm in Maschinensprache mit dem Argument "arg" auf.	238
VAL(x$)	Uebergibt den numerischen Wert der Zeichenkette x$.	135
VARPTR(Variable)	Uebergibt den Relativzeiger zu dem aktuellen Speichersegment der Variablen.	233
VARPTR$(Variable)	Uebergibt die Zeichenform der Adresse einer Variablen im Hauptspeicher.	234

VIEW[[SCREEN][(x1,y1) -(x2,y2)[,[Farbe] [,Rand]]]]	Definiert einen rechteckigen Ausschnitt des Bildschirms.	212
WAIT Tor,n[,m]	Verlaesst die Programmausfuehrung zur Beobachtung des Status eines Maschineneingabetores.	236
WEND	Schliesst die Anweisung WHILE ab.	108
WHILE Ausdruck	Fuehrt eine Serie von Anweisungen in einer Schleife so lange aus, wie eine angegebene Bedingung richtig ist.	108
WIDTH Groesse WIDTH Einheit,Groesse WIDTH #Dateinummer, Groesse	Legt die Anzahl der Zeichen einer Ausgabezeile fest. Nachdem die angegebene Anzahl von Zeichen ausgegeben wurde, fuegt BASIC eine Zeilenschaltung hinzu.	92
WINDOW [[SCREEN] (x1,y1)-(x2,y2)]	Erlaubt die Neudefinition der Bildschirmkoordinaten.	214
WRITE [Liste der Ausdruecke]	Gibt Daten auf dem Bildschirm aus.	88
WRITE #Dateinummer, Liste der Ausdruecke	Schreibt Daten in eine sequentielle Datei.	160

Anlage B

Reservierte Woerter

Einige Woerter bei BASIC haben eine bestimmte Bedeutung. Diese Woerter nennt man **Reservierte Woerter**. Reservierte Woerter beinhalten alle BASIC-Kommandos, -Anweisungen, -Funktionsnamen und -Operatornamen. Reservierte Woerter duerfen nicht als Variablennamen benutzt werden. Reservierte Woerter muessen immer von Daten oder anderen Teilen der BASIC-Anweisung durch Leerstellen oder andere Sonderzeichen, die in der Syntax erlaubt sind, getrennt werden. Das bedeutet, dass **Reservierte Woerter** immer getrennt stehen muessen, so dass sie von BASIC erkannt werden koennen.

Die folgende Liste zeigt alle reservierten Woerter:

ABS	ELSE	LINE
AND	END	LIST
ASC	ENVIRON	LLIST
ATN	ENVIRON$	LOAD
AUTO	EOF	LOC
BEEP	EQV	LOCATE
BLOAD	ERASE	LOF
BSAVE	ERDEV	LOG
CALL	ERDEV$	LPOS
CDBL	ERL	LPRINT
CHAIN	ERR	LSET
CHDIR	ERROR	MERGE
CHR$	EXP	MID$
CINT	FIELD	MKDIR
CIRCLE	FILES	MKD$
CLEAR	FIX	MKI$
CLOSE	FNxxxxxxxx	MKS$
CLS	FOR	MOD
COLOR	FRE	NAME
COM	GET	NEW
COMMON	GOSUB	NEXT
CONT	GOTO	NOT
COS	HEX$	OCT$
CSNG	IF	OFF
CSRLIN	IMP	ON
CVD	INKEY$	OPEN
CVI	INP	OPTION
CVS	INPUT	OR
DATA	INPUT$	OUT
DATE$	INPUT#	PAINT
DEF	INSTR	PEEK
DEFDBL	INT	PLAY
DEFINT	IOCTL	PMAP
DEFSNG	IOCTL$	POINT
DEFSTR	KEY	POKE
DELETE	KILL	POS
DIM	LEFT$	PRESET
DRAW	LEN	PRINT
EDIT	LET	PRINT#

PSET
PUT
RANDOMIZE
READ
REM
RENUME
RESET
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET
RUN
SAVE
SCREEN
SGN
SHELL
SIN
SOUND
SPACE$
SPC
SQR
STEP
STOP
STR$
STRING$
SWAP
SYSTEM
TAB
TAN
THEN
TIME$
TIMER
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VARPTR$
VIEW
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
WRITE#
XOR

ANLAGE_C

Zusammenstellung_der_Fehlercodes_und_der_Fehlermeldungen

Falls BASIC einen Fehler entdeckt, der das Programm stoppt, wird eine Fehlernachricht angezeigt. Es ist moeglich, mit der Anweisung ON ERROR und den Variablen ERR und ERL Fehler in einem BASIC-Programm zu lesen und zu testen. (Eine vollstaendige Erklaerung fuer ON ERROR, ERR, ERL, ERDEV und ERDEV$ steht unter der jeweiligen Anweisung oder Funktion in diesem Handbuch.

In diesem Anhang sind alle BASIC-Fehlernachrichten mit ihren zugehoerigen Fehlernummern aufgefuehrt:

Nummer	Nachricht
1	**NEXT without FOR** (NEXT ohne FOR). Der Anweisung NEXT fehlt die zugehoerige Anweisung FOR. Es kann sein, dass eine Variable in der Anweisung NEXT nicht mit einer Variablen der Anweisung FOR uebereinstimmt. Das Programm aendern, so dass NEXT eine zugehoerige Anweisung FOR besitzt.
2	**Syntax error** (Syntaxfehler) Eine Zeile enthaelt eine nicht gueltige Folge von Zeichen, wie z.B. ungerade Anzahl von Klammern, einen falsch geschriebenen Befehl, eine falsch geschriebene Anweisung, eine ungueltige Interpunktion, oder die Daten einer Anweisung DATA sind nicht vom gleichen Typ (numerisch oder Zeichenkette) wie die Variable in einer Anweisung READ oder es wurde ein reserviertes Wort als Variablenname benutzt. Wenn dieser Fehler auftritt, zeigt das BASIC-Korrekturprogramm automatisch die fehlerhafte Zeile an. Die Zeile des Programms muss verbessert werden.
3	**RETURN without GOSUB** (RETURN ohne GOSUB) Eine Anweisung RETURN benoetigt vorher eine Anweisung GOSUB. Programm korrigieren. Wahrscheinlich fehlt eine Anweisung STOP oder END vor dem Unterprogramm, die bewirkt, dass das Programm nicht in den Unterprogrammcode "faellt".
4	**Out of data** (Zuwenig Daten) Eine Anweisung READ versucht, mehr Daten zu lesen als in der Anweisung DATA angegeben sind. Das Programm korrigieren, so dass genug Konstanten in den Anweisungen DATA fuer alle Anweisungen READ des Programms zur Verfuegung stehen.

Nummer	Nachricht

5 **Illegal function call** (Ungueltiger Funktionsaufruf)
Ein sich nicht im gueltigen Bereich befindlicher Parameter wurde an eine Systemfunktion uebergeben. Der Fehler kann auch als Ergebnis der folgenden Punkte auftreten:

- Ein negativer oder zu grosser Index.
- Versuch, eine negative Zahl mit einer Zahl zu potenzieren, die nicht ganzzahlig ist.
- Funktion USR: Aufruf,bevor die Startadresse mit DEF USR definiert ist.
- Eine negative Satznummer fuer GET oder PUT (Datei).
- Ein ungueltiges Argument fuer eine Funktion oder Anweisung.
- Versuch ein geschuetztes BASIC-Programm aufzulisten oder zu veraendern.
- Versuch, Zeilennummern zu loeschen, die nicht existieren.

Korrigieren des Programms: siehe entsprechenden Befehl.

6 **Overflow** (Ueberlauf)
Der Bereich einer Zahl ist zu gross, um im BASIC-Zahlenformat dargestellt werden zu koennen. Ueberlauf fuer ganze Zahlen stoppt die Ausfuehrung, anderenfalls wird die groesste in der Maschine darstellbare Zahl mit dem richtigen Vorzeichen als Ergebnis uebergeben und die Ausfuehrung fortgesetzt. Nur ein ganzzahliger Ueberlauf kann abgefragt werden.
Um einen Ueberlauf fuer ganze Zahlen zu korrigieren, muessen kleinere Zahlen verwendet werden oder zu Variablen mit einfacher oder doppelter Genauigkeit uebergegangen werden.
Hinweis: Ist eine Zahl zu klein, um im Zahlenformat von BASIC dargestellt zu werden, erhaelt man eine **Unterlaufbedingung.** Wenn dies auftritt, ist das Ergebnis Null, und die Ausfuehrung wird ohne einen Fehler fortgesetzt.

7 **Out of memory** (Hauptspeicher reicht nicht aus)
Ein Programm ist zu gross, enthaelt zu viele FOR-Schleifen oder GOSUB s, zu viele Variablen, Ausdruecke, die zu kompliziert sind, oder komplexes Malen.
Mit CLEAR am Anfang des Programms kann der Stapel- oder Hauptspeicherbereich vergroessert werden.

8 **Undefined line number** (Nicht definierte Zeilennummer)
Eine Zeilenreferenz in einer Anweisung oder in einem Befehl spricht eine Zeile an, die sich nicht im Programm befindet.
Die Zeilennummern im Programm pruefen und die richtige Zeilennummer benutzen.

Nummer	Nachricht

9 **Subscript out of range** (Index ausserhalb des Bereichs)
Ein Feldelement wurde mit einem Index ausserhalb der Dimensionen des Feldes oder mit einer falschen Anzahl von Indizes angesprochen.
Benutzung der Feldvariablen ueberpruefen. Vielleicht wurde an eine Variable ein Index angefuegt, die keine Feldvariable ist, oder eine eingebaute Funktion wurde falsch codiert.

10 **Duplicate Definition** (Doppelte Definition)
Es wurde versucht, die Groesse des gleichen Feldes zweimal zu definieren. Dies kann auf verschiedene Weise passieren:

- Das gleiche Feld wurde in zwei Anweisungen DIM definiert.
- Das Programm fuehrt eine Anweisung DIM fuer ein Feld aus, nachdem die Standarddimension von 10 fuer dieses Feld gesetzt wurde.
- Das Programm erkennt eine Anweisung OPTION BASE, nachdem ein Feld entweder durch eine Anweisung DIM oder durch die Standardannahme dimensioniert wurde.

Die Anweisung OPTION BASE muss so im Programm gesetzt werden, dass sie ausgefuehrt wird, bevor irgendein Feld benutzt wird; oder das Programm muss so korrigiert werden, dass jedes Feld nur einmal definiert wird.

11 **Division by zero** (Division durch Null)
In einem Ausdruck wurde versucht, durch Null zu dividieren, oder es wurde versucht, Null mit einer negativen Zahl zu potenzieren.
Es ist nicht noetig, diese Bedingung zu korrigieren, da das Programm in der Ausfuehrung fortfaehrt. Das Ergebnis der Division ist die groesstmoegliche darstellbare Zahl mit dem Vorzeichen der Zahl, die dividiert werden sollte (Dividend); oder eine positive groesstmoegliche darstellbare Zahl als Ergebnis der Potenzierung. Dieser Fehler kann nicht abgefragt werden.

12 **Illegal direct** (Ungueltiger Direktmodus)
Es wurde versucht, eine Anweisung im Direktmodus einzugeben, fuer die der Direktmodus ungueltig ist (wie z.B. DEF FN).
Die Anweisung muss als Teil einer Programmzeile eingegeben werden.

Nummer	Nachricht
13	**Type mismatch** (Keine Typuebereinstimmung) Es wurde ein Zeichenkettenwert anstelle eines numerischen Wertes eingegeben, oder es wurde ein numerischer Wert anstelle eines Zeichenkettenwertes eingegeben. Dieser Fehler kann auch auftreten, wenn man versucht, Variablen verschiedenen Typs auszutauschen, wie z.B. Variablen einfacher mit Variablen doppelter Genauigkeit.
14	**Out of string space** (Platz fuer Zeichenketten zu Ende) BASIC ordnet den Platz fuer Zeichenketten dynamisch zu, bis kein Hauptspeicherplatz mehr zur Verfuegung steht. Die Nachricht bedeutet, dass ausgeloest durch Zeichenkettenvariablen, BASIC mehr als den verbleibenden Hauptspeicherplatz benoetigt, nachdem alles geloescht wurde, was geloescht werden konnte.
15	**String too long** (Zeichenkette zu lang) Es wurde versucht, eine Zeichenkette zu erstellen, die laenger als 255 Zeichen ist. Versuchen, die Zeichenkette in kleinere Zeichenketten aufzuteilen.
16	**String formula too complex** (Formel fuer Zeichenkette zu komplex) Ein Zeichenkettenausdruck ist zu lang oder zu komplex. Der Ausdruck sollte in kleinere Ausdruecke zerlegt werden.
17	**Can't continue** (Programmausfuehrung kann nicht fortgesetzt werden) Es wurde versucht, mit CONT die Ausfuehrung eines Programms fortzusetzen, das: - durch einen Fehler gestoppt wurde; - waehrend einer Ausfuehrungsunterbrechung veraendert wurde; - nicht existiert. Das Programm muss geladen und mit RUN ausgefuehrt werden.
18	**Undefined user function** (Nicht definierte Benutzerfunktion) Eine Funktion wurde aufgerufen, bevor sie mit der Anweisung DEF FN definiert wurde. Das Programm muss die Anweisung DEF FN ausfuehren, bevor die Funktion benutzt werden kann.

Nummer	Nachricht
19	**No RESUME** (Keine Wiederaufnahme des Programms) Das Programm verzweigt zu einer aktiven Fehlerbehandlungsroutine als Ergebnis einer Fehlerbedingung oder einer Anweisung ERROR. In dieser Routine fehlt eine Anweisung RESUME. (Das physische Ende des Programms wurde in der Fehlerbehandlungsroutine gefunden.) RESUME muss in die Fehlerbehandlungsroutine eingefuegt werden, um mit der Programmausfuehrung fortfahren zu koennen. Man kann in die Fehlerbehandlungsroutine auch die Anweisung ON ERROR GOTO Ø einfuegen, so dass BASIC fuer jeden nicht gelesenen Fehler die Nachricht anzeigt.
20	**RESUME without error** (RESUME ohne Fehler) Das Programm hat eine Anweisung RESUME gefunden, ohne durch einen Fehler dorthin gekommen zu sein. In die Fehlerbehandlungsroutine sollte nur verzweigt werden, wenn ein Fehler auftrat oder eine Anweisung ERROR ausgefuehrt wird. Wahrscheinlich muss eine Anweisung STOP oder END vor der Fehlerbehandlungsroutine eingefuegt werden, um zu verhindern, dass das Programm in die Fehlerbehandlungsroutine "faellt".
22	**Missing operand** (Fehlender Operand) Ein Ausdruck enthaelt einen Operator, wie z.B. * (Stern) oder OR, dem kein Operand folgt. Alle benoetigten Operanden muessen in den Ausdruck eingefuegt werden.
23	**Line buffer overflow** (Zeilenpufferueberlauf) Es wurde versucht, eine Zeile einzugeben, die aus zu vielen Zeichen besteht. Einzeilige Mehrfachanweisungen in mehrere Zeilen aufteilen. Es koennen auch Zeichenkettenvariablen anstelle von Konstanten benutzt werden, wo dies moeglich ist.
24	**Device Timeout** (Einheitenzeitsperre) BASIC erhielt innerhalb einer vorgegebenen Zeitspanne keine Information von einer Ein-/Ausgabeeinheit. Fuer Datenfernverarbeitungsdateien bedeutet diese Nachricht, dass ein oder mehrere Signale, die mit OPEN "COM... gesetzt wurden, in einer angegebenen Zeitperiode nicht gefunden wurden. Operation wiederholen.

Nummer	Nachricht
25	**Device Fault** (Einheitenfehler) Ein Hardware-Fehler wurde von einem Anschluss angezeigt. Diese Nachricht kann auch auftreten, wenn Daten in eine Datenfernverarbeitungsdatei uebertragen werden. In diesem Fall wird damit angezeigt, dass ein oder mehrere Signale, die getestet werden (angegeben mit der Anweisung OPEN "COM...), nicht in der vorgegebenen Zeitspanne gefunden wurden.
26	**FOR without NEXT** (FOR ohne NEXT) FOR wurde ohne ein zugehoeriges NEXT gefunden. Das heisst, eine FOR-Schleife war aktiv, als das physische Ende des Programms erreicht wurde. Das Programm durch Einfuegen einer Anweisung NEXT verbessern.
27	**Out of Paper** (Kein Druckerpapier) Im Drucker ist kein Papier oder er ist nicht eingeschaltet. Papier einlegen (falls noetig), die Verbindung des Druckers ueberpruefen oder den Drucker einschalten. Danach mit dem Programm fortfahren.
29	**WHILE without WEND** (WHILE ohne WEND) Einer Anweisung WHILE fehlt die zugehoerige Anweisung **WEND.** Das heisst, WHILE war noch aktiv, als das physische Ende des Programms erreicht wurde. Das Programm durch Einfuegen einer Anweisung WEND fuer jedes WHILE korrigieren.
30	**WEND without WHILE** (WEND ohne WHILE) Eine Anweisung WEND wurde gefunden, bevor die zugehoerige Anweisung WHILE ausgefuehrt wurde. Das Programm so verbessern, dass fuer jede Anweisung WEND eine Anweisung WHILE existiert.
50	**FIELD overflow** (FELD-Ueberlauf) Eine Anweisung FIELD versucht, mehr Bytes zuzuordnen als fuer die Satzlaenge in einer Datei fuer wahlfreien Zugriff in der Anweisung OPEN angegeben waren, oder es wurde das Ende des FIELD-Puffers gefunden, waehrend eine sequentielle Ein-/Ausgabe (PRINT#, WRITE#, INPUT#) zu einer Datei mit wahlfreiem Zugriff erfolgte. Die Anweisungen OPEN und FIELD sind zu ueberpruefen und beide auf Uebereinstimmung zu bringen. Bei sequentieller Ein-/Ausgabe zu einer Datei mit wahlfreiem Zugriff darf die Laenge der gelesenen oder geschriebenen Daten die Satzlaenge der Datei fuer wahlfreien Zugriff nicht ueberschreiten.

Nummer	Nachricht
51	**Internal error** (Interner Fehler) In BASIC trat ein interner Fehler auf. Diskette neu kopieren. Die Hardware pruefen und die Operation wiederholen. Im erneuten Fehlerfalle das Serviceunternehmen ueber die Fehlerbedingungen informieren.
52	**Bad file number** (Falsche Dateinummer) Eine Anweisung benutzt eine Dateinummer einer Datei, die nicht eroeffnet ist, oder die Dateinummer befindet sich nicht in dem moeglichen Dateinummernbereich, der bei der Initialisierung angegeben wurde; oder der Einheitenname in der Dateispezifikation ist zu lang oder ungueltig oder der Dateiname war zu lang oder ungueltig. Sicherstellen, dass die gewuenschte Datei eroeffnet oder die Dateinummer richtig in die Anweisung eingefuegt wurde. Auch auf gueltige Dateispezifikation ueberpruefen (siehe unter "Namensgebung fuer Dateien" in Kapitel 6).
53	**File not found** (Datei nicht gefunden) Mit LOAD, KILL, NAME, FILES oder OPEN wurde eine Datei angesprochen, die nicht auf der Diskette in der angegebenen Einheit steht. Ueberpruefen, ob sich die richtige Diskette in der angegebenen Einheit befindet und ob die Dateispezifikation stimmt. Danach die Operation wiederholen.
54	**Bad file mode** (Falscher Dateityp) Mit PUT oder GET wurde versucht, eine sequentielle Datei oder eine geschlossene Datei anzusprechen, oder ein OPEN mit einem anderen Dateityp als Eingabe, Ausgabe, Hinzufuegen oder wahlfreier Zugriff auszufuehren. Sicherstellen, dass die Anweisung OPEN richtig eingegeben und ausgefuehrt wurde. GET und PUT benoetigen eine Datei fuer wahlfreien Zugriff. Dieser Fehler tritt auch auf, wenn man versucht, eine Datei, die nicht im ASCII-Format vorliegt, zu mischen. In diesem Fall muss man sicherstellen, dass die richtige Datei gemischt wird. Wenn noetig, das Programm laden und mit der Auswahl A speichern.
55	**File already open** (Datei schon eroeffnet) Es wurde versucht, eine Datei fuer sequentielle Ausgabe oder fuer sequentielles Hinzufuegen zu eroeffnen, und die Datei war schon eroeffnet; oder es wurde versucht, KILL fuer eine schon eroeffnete Datei zu benutzen. Sicherstellen, dass nur ein OPEN fuer eine Datei ausgefuehrt wird, wenn sie sequentiell beschrieben werden soll. Die Datei abschliessen, bevor KILL benutzt wird.

Nummer	Nachricht
57	**Device I/O Error** (Einheitenein-/-ausgabefehler) Ein Fehler passierte bei einer Einheitenein-/-ausgabeoperation. DCP kann diesen Fehler nicht beheben. Bei Empfang von Datenfernverarbeitungsdaten kann dieser Fehler durch Ueberlauf, Datenfernverarbeitungsrahmen, Unterbrechung oder Paritaetsfehler auftreten. Werden Daten mit sieben oder weniger Daten-Bits empfangen, wird das achte Bit in das fehlerhafte Byte eingefuegt.
58	**File already exists** (Die Datei existiert schon) Eine in der Anweisung NAME angegebener Dateiname existiert schon auf der Diskette. Den Befehl NAME mit einem anderen Namen wiederholen.
61	**Disk full** (Diskette voll) Der Speicher auf der Diskette ist vollstaendig benutzt. Dateien werden abgeschlossen, wenn dieser Fehler auftritt. Befinden sich Dateien auf der Diskette, die nicht mehr benoetigt werden, sollten sie geloescht werden; oder es sollte eine neue Diskette benutzt werden. Danach die Operation wiederholen oder das Programm neu laufen lassen.
62	**Input past end** (Eingabe nach logischem Ende) Dies ist ein Dateiendefehler. Eine Eingabeanweisung wurde fuer eine leere Datei ausgefuehrt oder nachdem alle Daten einer sequentiellen Datei schon gelesen waren. Um diesen Fehler zu vermeiden, sollte die Funktion EOF benutzt werden, um das Ende der Datei zu erkennen. Der gleiche Fehler wird angezeigt, falls versucht wird, von einer Datei zu lesen, die fuer Ausgabe oder Hinzufuegen eroeffnet wurde. Soll von einer sequentiellen Ausgabe- (oder Hinzufuege-) Datei gelesen werden, muss sie geschlossen und wieder fuer Eingabe eroeffnet werden.
63	**Bad record number** (Falsche Satznummer) In einer Anweisung PUT oder GET ist die Satznummer entweder groesser als die maximal erlaubte oder gleich Null. Die Anweisung PUT oder GET korrigieren, damit eine gueltige Satznummer angesprochen wird.
64	**Bad file name** (Falscher Dateiname) Eine ungueltige Form wurde fuer den Dateinamen mit BLOAD, BSAVE, KILL, NAME, OPEN oder FILES benutzt. In Kapitel 6 unter "Namensgebung fuer Dateien" nachsehen, wie gueltige Dateinamen aussehen muessen, und den fehlerhaften Dateinamen korrigieren.

Nummer	Nachricht
66	**Direct Statement in file** (Anweisung fuer direkten Modus in der Datei) Eine Anweisung fuer direkten Modus wurde gefunden, waehrend ein Programm mit LOAD oder CHAIN im ASCII-Format geladen wurde. LOAD oder CHAIN werden beendet. Die ASCII-Datei darf nur Anweisungen enthalten, denen eine Zeilennummer vorausgeht. Der Fehler kann auch auftreten, weil sich ein Zeichen fuer Zeilenvorschub im Eingabestrom befindet.
67	**Too many files** (Zu viele Dateien) Es wurde versucht, eine neue Datei zu erstellen (mit Hilfe von SAVE oder OPEN), obwohl alle Bibliothekseintragungen der Diskette belegt waren oder die Dateispezifikation ungueltig war. Ist die Dateispezifikation in Ordnung, muss eine neue formatierte Diskette verwendet und die Operation wiederholt werden.
68	**Device Unavailable** (Einheit nicht verfuegbar) Es wurde versucht, eine Datei in einer Einheit zu eroeffnen, die nicht vorhanden ist. Entweder fehlt die Hardware, um die Einheit zu unterstuetzen (wie z.B. Druckeranschluesse fuer den zweiten oder dritten Drucker) oder die Einheit wurde inaktiviert (Es wurde z.B. /C:0 beim Laden von BASIC benutzt. Dadurch wuerden die Datenfernverarbeitungseinheiten inaktiviert). Die Einheit muss richtig installiert sein. Falls noetig, zu DCP zurueckverzweigen, wo der BASIC-Befehl erneut eingegeben werden kann.
69	**Communication buffer overflow** (Ueberlauf des Datenfernverarbeitungspuffers) Eine Eingabeanweisung fuer Datenfernverarbeitung wurde ausgefuehrt, aber der Eingabepuffer war schon voll. Die Anweisung ON ERROR muss benutzt werden, um die Eingabe zu wiederholen, falls diese Bedingung auftritt. Nachfolgende Eingaben versuchen, diesen Fehler zu loeschen, ausgenommen,die Zeichen werden weiterhin schneller empfangen, als das Programm sie verarbeiten kann. Es gibt verschiedene Loesungen: - Die Groesse des Datenfernverarbeitungspuffers mit Hilfe der Auswahl /C: erweitern, wenn mit BASIC gestartet wird. - Ein "Hand-Shaking"-Protokoll mit dem anderen Computer implementieren, um ihm mitzuteilen, so lange nicht zu senden, bis die empfangenen Daten verarbeitet sind. - Eine niedrigere Uebertragungsrate fuer Senden und Empfangen benutzen.

Nummer	Nachricht
70	**Disk Write Protect** (Diskettenschreibschutz) Es wurde versucht, eine Diskette mit Disketten-schreibschutz zu beschreiben. Ueberpruefen, ob die richtige Diskette benutzt wird. Wenn dies der Fall ist, den Schreibschutz entfernen und die Operation wiederholen. Dies kann auch ein Hardware-Fehler sein.
71	**Disk not Ready** (Diskette nicht bereit) Die Diskettenverriegelung ist offen oder in der Einheit befindet sich keine Diskette. Die richtige Diskette in das Laufwerk einlegen und mit dem Programm fortfahren.
72	**Disk Media Error** (Diskettenfehler) Die Steueranschlusskarte entdeckt einen Hardware-Fehler oder einen Diskettenfehler. Im allgemeinen bedeutet dies, dass eine Diskette fehlerhaft ist. Alle sich auf der Diskette befindlichen Dateien auf eine neue Diskette kopieren und die fehlerhafte Diskette neu formatieren. Ist das Formatieren nicht moeglich, sollte die Diskette nicht mehr benutzt werden.
74	**Rename across disks** (Umbenennen auf Diskette) Es wurde versucht, eine Datei umzubenennen, aber die falsche Diskette wurde angegeben. Die Umbenennung findet nicht statt. Wenn **RENAME** benutzt wird, muss fuer den alten und den neuen Dateinamen das gleiche Laufwerk angegeben werden, ausser der DCP-Befehl **ASSIGN** ist aktiviert. In diesem Fall kann der Laufwerksname unterschiedlich sein, es muss jedoch dieselbe physische Einheit angegeben werden.
75	**Path/file access error** (Pfad/Datei-Zugriffsfehler) Waehrend einer **OPEN**-, **RENAME**-, **MKDIR**-, **CHDIR**- oder **RKDIR**-Operation wurde der Versuch unternommen, einen Pfad oder Dateinamen zu einer Datei zu benutzen, auf die nicht zugegriffen werden kann. Zum Beispiel wurde versucht, ein Inhaltsverzeichnis oder einen Datentraeger-Kennsatz zu eroeffnen, oder eine Datei, die nur fuer Lesen definiert war, fuer Schreiben zu eroeffnen, oder das laufende Verzeichnis zu loeschen. Die Operation wird nicht beendet.

Nummer	Nachricht
76	**Path not found** (Pfad nicht gefunden) Waehrend einer **OPEN-**, **MKDIR-**, **CHDIR-** oder **RMDIR-**Operation kann DCP den angegebenen Pfad nicht finden. Die Operation wird nicht beendet.
	- **Unprintable error** (Nichtdruckbarer Fehler) Fuer die existierende Fehlerbedingung ist keine Fehlernachricht verfuegbar. Dies passiert gewoehnlich durch eine Anweisung ERROR mit einem nicht definierten Fehlercode. Im Programm pruefen, ob alle erstellten Fehlercodes auch bearbeitet werden.
	- **Incorrect DCP-Version** (Falsche DCP-Version) Der gerade eingegebene Befehl gehoert zu einer anderen DCP-Version.
	- **You cannot SHELL to Basic** (SHELL kann nicht zu BASIC durchgefuehrt werden). BASIC kann nicht als Kind-Prozess ausgefuehrt werden.
	- **Can't continue after SHELL** (Programm kann nach **SHELL** nicht fortgesetzt werden) Ein Kind-Prozess, der durch die Anweisung **SHELL** ausgefuehrt wurde, sollte nie beendet werden und resident bleiben, da sonst fuer BASIC nicht genuegend Speicherplatz zur Verfuegung steht und die Programmausfuehrung angehalten wird.

Anlage D

ASCII-Zeichencodes

In der folgenden Tabelle sind alle ASCII-Codes (dezimal, hexadezimal und ihre zugehoerigen Zeichen) aufgelistet.
Diese Zeichen koennen mit Hilfe von **PRINT CHR$(n)** angezeigt werden, wobei n der ASCII-Code ist.
In der Tabelle sind die Standardinterpretationen der ASCII-Codes 0 bis 31 aufgefuehrt. Dies sind nicht darstellbare Zeichen und werden gewoehnlich fuer Steuerfunktionen oder Datenfernverarbeitung benutzt.

Jedes dieser Zeichen kann ueber die Tastatur eingegeben werden, indem man die Taste "ALT" betaetigt und niederhaelt und dann die Ziffern fuer den ASCII-Code auf der Zehnertastatur eingibt. Dabei muss beachtet werden, dass einige dieser Codes eine besondere Bedeutung fuer das BASIC-Korrekturprogramm haben - das BASIC-Korrekturprogramm benutzt seine eigene Interpretation fuer diese Codes und kann die hier aufgelisteten Sonderzeichen nicht anzeigen.

Hinweis:

Die Zeichen fuer die ASCII-Codes 128 bis 255 koennen nur im Textmodus dargestellt werden. Siehe Anweisung **SCREEN** in diesem Handbuch und Abschnitt "Benutzung des Bildschirms" im **BASIC-Handbuch.**

ASCII-Codetabelle

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

ASCII-Codetabelle (Fortsetzung)

Dez	Hex	CHR	Dez	Hex	CHR	Dez	Hex	CHR	Dez	Hex	CHR
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	∈
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▓	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	▉	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Erweiterte_Codes

Fuer gewisse Tasten und Tastenkombinationen, die nicht im Standard ASCII-Code dargestellt werden koennen, wird ein erweiterter Code von der Systemvariablen **INKEY$** uebergeben. Eine Leerstelle (ASCII-Code 000) wird als erstes Zeichen einer Zwei-Zeichen-Zeichenkette uebergeben. Wird **INKEY$** eine Zeichenkette aus zwei Zeichen empfangen, sollte man zurueckverzweigen und das zweite Zeichen pruefen, um zu bestimmen, welche Taste betaetigt wurde. Im allgemeinen, aber nicht immer, ist dieser zweite Code der Pruefcode der Primaertaste, die betaetigt wurde. Die ASCII-Codes (dezimal) fuer dieses zweite Zeichen und die zugehoerigen Tasten sind unten aufgelistet.

Zweiter Code	Bedeutung
3	(Leerstelle)NUL
15	(Umschalten Tabulator)| <--
16-25	ALT-Q, W, E, R, T, Y, U, I, O, P
30-38	ALT-A, S, D, F, G, H, J, K, L
44-50	ALT-Z, X, C, V, B, N, M
59-68	Funktionstasten F1 bis F10 (falls inaktiviert als Funktionstasten)
71	HOME
72	Kursor nach oben
73	PG UP
75	Kursor nach links
77	Kursor nach rechts
79	END
80	Kursor nach unten
81	PG DN
82	INS
83	DEL
84-93	F11-F20 (Shift-F1 bis F10)
94-103	F21-F30 (CTRL-F1 bis F10)
104-113	F31-F40 (ALT-F1 bis F10)
114	CTRL-PRTSC
115	CTRL-Kursor nach links (vorheriges Wort)
116	CTRL-Kursor nach rechts (naechstes Wort)
117	CTRL-END
118	CTRL-PG DN
119	CTRL-HOME
120-131	ALT-1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =
132	CTRL-PG UP

ANLAGE_E

Hexadezimale_Umwandlungstabelle

Hex	Dezimal	Hex	Dezimal
1	1	10	16
2	2	20	32
3	3	30	48
4	4	40	64
5	5	50	80
6	6	60	96
7	7	70	112
8	8	80	128
9	9	90	144
A	10	A0	160
B	11	B0	176
C	12	C0	192
D	13	D0	208
E	14	E0	224
F	15	F0	240
100	256	1000	4096
200	512	2000	8192
300	768	3000	12288
400	1024	4000	16384
500	1280	5000	20480
600	1536	6000	24576
700	1792	7000	28672
800	2048	8000	32768
900	2304	9000	36864
A00	2560	A000	40960
B00	2816	B000	45056
C00	3072	C000	49152
D00	3328	D000	53248
E00	3584	E000	57344
F00	3840	F000	61440

ANLAGE_F

Mathematische_Funktionen

Funktionen, die nicht im BASIC des Personalcomputers eingebaut sind, koennen wie folgt berechnet werden:

Funktion	Aequivalent
Logarithmus zur Basis B	LOG(X) LOG(X)/LOG(B)
Sekans	SEC(X) = 1/COS(X)
Cosekans	CSC(X) = 1/SIN(X)
Cotangens	COT(X) = 1/TAN(X)
Arcussinus	ARCSIN(X) = ATN(X/SQR(1-X*X))
Arcuscosinus	ARCCOS(X) = 1.570796-ATN(X/SQR(1-X*X))
Arcussekans	ARCSEC(X) = ATN(SQR(X*X-1))+(X<0)*3.141593
Arcuscosekans	ARCCSC(X) = ATN(1/SQR(X*X-1))+(X<0)*3.141593
Arcuscotangens	ARCCCOT(X) = 1.57096-ATN(X)
Sinus Hyperbolikus	SINH(X) = (EXP(X)-EXP(-X))/2
Cosinus Hyperbolikus	COSH(X) = (EXP(X)+EXP(-X))/2
Tangens Hyperbolikus	TANH(X) = (EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
Sekans Hyperbolikus	SECH(X) = 2/(EXP(X)+EXP(-X))
Cosekans Hyperbolikus	CSCH(X) = 2/(EXP(X))-EXP(-X))
Cotangens Hyperbolikus	COTH(X) = (EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
Arcussinus Hyperbolikus	ARCSINH(X) = LOG(X+SQR(X*X+1))
Arcuscosinus Hyperbolikus	ARCCOSH(X) = LOG(X+SQR(X*X-1))
Arcustangens Hyperbolikus	ARCTANH(X) = LOG((1+X)/(1-X))/2
Arcussekans Hyperbolikus	ARCSECH(X) = LOG((1+SQR(1-X*X))/X
Arcuscosekans Hyperbolikus	ARCCSCH(X) = LOG((1+SGN(X)*SQR(1+X*X))/X
Arcuscotangens Hyperbolikus	ARCCOTH(X) = LOG((X+1)/(X-1))/2

Werden diese Funktionen benutzt, ist es ein guter Weg, sie mit der Anweisung DEF FN zu codieren. Statt die Formel fuer den Arcussinus Hyperbolikus jedesmal, wenn er benoetigt wird, zu codieren, koennte man dies folgendermassen tun:

```
DEF FNARCSINH(X) = LOG(X+SQR(X*X+1))
```

Danach koennte man die Funktionen wie folgt aufrufen:

```
FNARCSINH(Y)
```

Anlage_G

Interpretation_spezieller_Zeichen_durch_den_Bildschirm

ASCII	Zeichen	Wirkung
0	(leer)	
1	☺	
2	●	
3	♥	
4	♦	
5	♣	
6	♠	
7	(Alarm)	
8	■	
9	(Tabulator)	
10	(LF)	Kursor eine Zeile nach unten
11		
12	(FF)	Bildschirm loeschen und Kursor an den Anfang
13	(CR)	Kursor an Zeilenanfang der naechsten Zeile
14	♫	
15	☼	
16	►	
17	◄	
18	↕	
19	!!	
20	¶	
21	§	
22	▬	
23	↨	
24	↑	
25	↓	
26	-->	
27	<--	
28		Kursor nach rechts
29		Kursor nach links
30		Kursor nach oben
31		Kursor nach unten

Steuerzeichen_fuer_Drucker_K6313/14

Die Steuerung des Druckers erfolgt auf Basis des ASCII-Codes. Spezielle Steuerzeichenfolgen werden mit Escape (27) eingeleitet.

Beispiel:	Wirkung:
LPRINT CHR$(27);"E"	Einschalten Fettschrift.
LPRINT CHR$(12)	Formularvorschub

Steuerzeichen	Wirkung
SO (14)	Einschalten Sperrschrift
SI (15)	Einschalten komprimierte Schrift
DC2 (18)	Ausschalten komprimierte Schrift
DC4 (20)	Ausschalten Sperrschrift
ESC E	Einschalten Fettschrift
ESC F	Ausschalten Fettschrift
ESC G	Einschalten Doppeldruck
ESC H	Ausschalten Doppeldruck
ESC S	Einschalten Hoch- oder Tiefschrift
ESC T	Ausschalten Hoch- oder Tiefschrift
ESC W	Ein- oder Ausschalten Sperrschrift
ESC -	Ein- oder Ausschalten Unterstreichmodus
ESC K	Einzelpunktmodus 480 Sprossen pro 8 Zoll
ESC L	Einzelpunktmodus 960 Sprossen pro 8 Zoll
ESC Y	Einzelpunktmodus 960 Sprossen pro 8 Zoll
ESC Z	Einzelpunktmodus 1920 Sprossen pro 8 Zoll
ESC 0	Setzen des Zeilenabstandes auf 1/8"
ESC 1	Setzen des Zeilenabstandes auf 7/72"
ESC 2	Start des variablen Zeilenabstandes
ESC 3	Setzen des Zeilenabstandes auf n/216"
ESC A	Voreinstellen eines variablen Zeilenabstandes von n/72"
LF (10)	Zeilenvorschub
VT (11)	Vertikaltabulation
FF (12)	Formularvorschub

Steuerzeichen	Wirkung
ESC J	Ausfuehrung eines Zeilenvorschubes von n/216"
ESC N	Einstellen Formularendezeile
ESC O	Ausschalten Formularendezeile
NUL (Ø)	Ende Tabulator setzen
HT (9)	Horizontal-Tabulation
CR (13)	Druckposition auf Zeilenanfang setzen
ESC C	Einstellen Formularlaenge (n Zeilen)
ESC CØ	Einstellen Formularlaenge (n Zoll)
ESC D	Setzen von Horizontaltabulations-Marken
CAN (24)	Loeschen des Druckpuffers
BEL (8)	Summer
ESC 8	Ausschalten Papierendekontrolle
ESC 9	Einschalten Papierendekontrolle
ESC 6	Auswahl Zeichensatz 2
ESC 7	Auswahl Zeichensatz 1
ESC U	Einstellen uni- und bidirektionaler Druck
ESC <	Einzeiliger unidirektionaler Druck 1 Zeile von links beginnend
ESC x	Ein- oder Ausschalten des NLQ-Modus

Anlage_H

Technische_Informationen

1. Hauptspeicherbelegung

2. Speicherung von Variablen

3. Tastaturpuffer

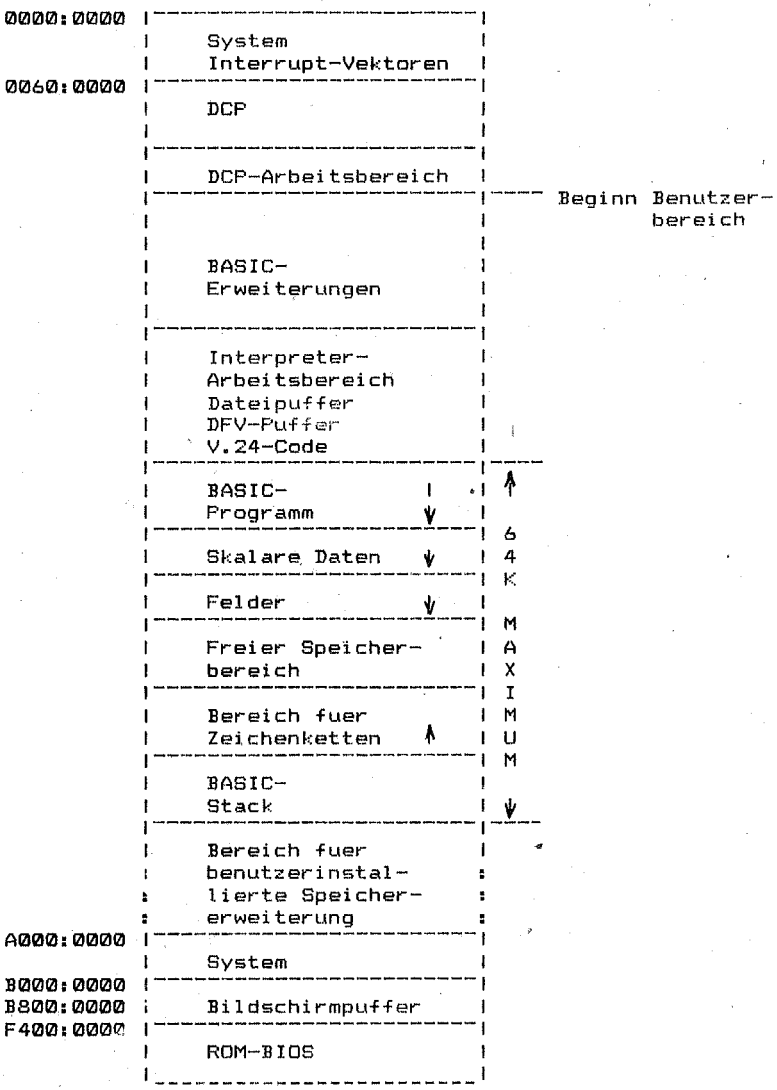
4. Suchordnung fuer Anschluesse

5. Umschalten der Bildschirme

1. Hauptspeicherbelegung

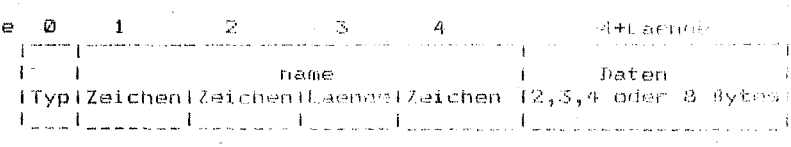
Die Adressen werden in folgender Hexadezimalform dargestellt:

Segment:Relativzeiger.



2. Speicherung von Variablen

Skalarvariablen werden im **BASIC-Datenbereich** wie folgt gespeichert:



Typ
Gibt den Variablentyp an:

2 Ganzzahlig
3 Zeichenkette
4 Einfache Genauigkeit
8 Doppelte Genauigkeit

Name
Ist der Name der Variablen. Die ersten zwei Zeichen des Namens sind in Byte 1 und Byte 2 gespeichert. Byte 3 sagt aus, wieviel weitere Zeichen sich im Variablennamen befinden. Diese zusaetzlichen Zeichen werden ab Byte 4 gespeichert.
Dies bedeutet, dass jeder Variablenname mindestens drei Bytes belegt. Ein Name aus ein oder zwei Zeichen benoetigt ekakt drei Bytes. Ein Name aus x Zeichen benoetigt **x+1** Bytes.

Daten
Folgt dem Namen der Variablen und kann zwei, drei, vier oder acht Bytes lang sein (wie unter **Typ** beschrieben). Der Wert, der von der Funktion **VARPTR** uebergeben wird, zeigt auf diese Daten.

Fuer **Zeichenkettenvariablen** bedeutet **Daten** die **Zeichenkettenbeschreibung:**

- Das erste Byte der Zeichenkettenbeschreibung enthaelt die Laenge der Zeichenkette (0 bis 255).

- Die letzten zwei Bytes der Zeichenkettenbeschreibung enthalten die Adresse der Zeichenkette im **BASIC**-Datenbereich (den Relativzeiger in das Standardsegment). Adressen werden mit dem niedrigen Byte zuerst und mit hoeheren Byte danach gespeichert:

  - Das zweite Byte der Zeichenkettenbeschreibung enthaelt also das niedrige Byte des Relativzeigers.

  - Das dritte Byte der Zeichenkettenbeschreibung enthaelt das hohe Byte des Relativzeigers.

Fuer numerische Variablen enthaelt **Daten** den aktuellen Wert der Variablen:

Ganzzahlige Werte werden in zwei Byte gespeichert, das niedrige Byte zuerst und das hohe Byte danach.

Werte einfacher Genauigkeit werden in vier Bytes gespeichert, und zwar in der internen Gleitkommabinaerdarstellung von **BASIC**.

Werte doppelter Genauigkeit werden in acht Bytes gespeichert, und zwar in der internen Gleitkommabinaerdarstellung von **BASIC**.

3. Tastaturpuffer

Zeichen, die ueber die Tastatur eingegeben werden, werden im Tastaturpuffer gespeichert, bis sie verarbeitet werden. Bis zu 15 Zeichen koennen im Puffer gespeichert werden; wird versucht, mehr als 15 Zeichen einzugeben, gibt der Computer ein Alarmzeichen.

INKEY$ liest **nur** ein Zeichen aus dem Tastaturpuffer, selbst wenn mehrere Zeichen vorhanden sind. Mit **INPUT$** koennen mehrere Zeichen gelesen werden. Steht jedoch die angegebene Anzahl von Zeichen nicht im Puffer, wartet **BASIC**, bis genug Zeichen eingegeben sind.

Der Systemtastaturpuffer kann durch die folgende Programmzeile geloescht werden:

```
WHILE INKEY$ <> "": WEND
```

Diese Technik kann nuetzlich sein, wenn der Puffer geloescht werden soll, bevor der Benutzer "irgendeine Taste" betaetigen soll.

BASIC hat seinen eigenen Zeilenpuffer, in dem das **BASIC-Korrekturprogramm** Zeichen verarbeitet, die von dem Systemtastaturpuffer empfangen werden. Der **BASIC-Zeilenpuffer** kann durch die folgende Programmzeile geloescht werden:

```
DEF SEG: POKE 106,0
```

4. Suchordnung fuer Anschluesse

Die Drucker mit Zuordnung **LPT1:**, **LPT2:** und **LPT3:** werden zugeordnet, wenn der Computer eingeschaltet wird. Das System prueft die Druckeranschluesse nach einer bestimmten Reihenfolge. Der erste gefundene Druckeranschluss erhaelt **LPT1:**, der zweite Anschluss (falls vorhanden) wird **LPT2:** und der dritte (falls vorhanden) wird **LPT3:**. Die Suchordnung ist wie folgt:

1. Schwarz/Weiss-Bildschirm und Paralleldruckeranschluss
2. Paralleldruckeranschluss
3. Paralleldruckeranschluss, der veraendert wurde, um die Basisadresse zu aendern.

Wenn ein Drucker mit Hilfe des Befehls **MODE** von **DCP** neu verbunden wird, ist diese Aenderung auch in **BASIC** effektiv.

Die Datenfernverarbeitungseinheiten **COM1:** und **COM2:** erhalten ihre Zuordnung aehnlich wie die Drucker.

Ihre Suchordnung ist:

1. Anschlusskarte fuer asynchrone Uebertragung
2. Eine veraenderte Anschlusskarte fuer asynchrone Uebertragung.

5. Umschalten der Bildschirme

Besitzt der Personalcomputer die Anschlusskarte fuer Farb-/Grafikbildschirm und Schwarz/Weiss-Bildschirm mit Paralleldrucker, so gibt **BASIC** normalerweise zum Schwarz/Weiss-Bildschirm aus. Jedoch kann man mit **BASIC** durch den folgenden Programmcode von einem Bildschirm zum anderen umschalten:

```
10 'Umschalten auf Schw.-weiss-BS
20 DEF SEG=0
30 POKE &H410,(PEEK(&H410)OR &H30)
40 SCREEN 0
50   WIDTH 40
60   WIDTH 80
70   LOCATE,,1,12,13

10 'Umschalten auf Farb-BS
20 DEF SEG=0
30 POKE &H410,(PEEK(&H410) AND (&HCF) OR &H10)
```

Hinweis:

Mit dieser Technik wird der Bildschirm, zu dem umgeschaltet wird, geloescht. Die Stellung des Kursor kann fuer jeden Bildschirm verschieden sein.

Anlage_I

Programmiertips

Oft gibt es verschiedene Wege, etwas in **BASIC** zu programmieren, um das gleiche Ergebnis zu erhalten.
Dieser Abschnitt enthaelt verschiedene Programmiertips, die Programmausfuehrungszeit zu verbessern.

ALLGEMEIN

- Anweisungen kombinieren,
wo es moeglich ist, um den Vorteil der Anweisungslaenge von 255 Zeichen auszunutzen. Beispiel:

Besser

```
100 FOR I=1 TO 10 : READ A(I) : NEXT I
```

Als

```
100 FOR I=1 TO 10
110 READ A(I)
120 NEXT I
```

- Wiederholte Berechnungen fuer Ausdruecke vermeiden.
Werden identische Berechnungen in mehreren Anweisungen ausgefuehrt, kann man diese Berechnungen einmal ausfuehren und das Ergebnis in einer Variablen speichern, bevor diese in spaeteren Anweisungen benutzt wird. Beispiel:

Besser

```
300 X=C*3+D
310 A=X+Y
320 B=X+Z
```

Als

```
310 A=C*3+D+Y
320 B=C*3+D+Z
```

Jedoch geht die Zuordnung einer Konstanten zu einer Variablen schneller, als die Zuordnung des Wertes einer Variablen zu einer anderen Variablen.

- Einfache Arithmetik benutzen.
Im allgemeinen geht eine Addition schneller als eine Multiplikation, und die Multiplikation ist schneller als eine Division oder Potenzierung.

Betrachten wir diese Beispiele:

Besser

```
250 B=A*.5
500 B=A+A
650 B=A*A*A
750 B%=A%/4
```

Als

```
250 B=A/2
500 B=A*2
650 B=A^3
750 B%=INT(A%/4)
```

- Eingebaute Funktionen benutzen.
Eingebaute Funktionen sind immer schneller, als die gleiche Moeglichkeit, in **BASIC** programmiert.

- Mit Bemerkungen sparen.
BASIC braucht jedesmal ein klein wenig Zeit, um eine Bemerkung zu identifizieren. Mit einem Apostroph (') Bemerkungen besser an das Ende einer Zeile anfuegen, als eine eigene Anweisung benutzen. Dadurch wird der Durchsatz verbessert und es wird Hauptspeicher eingespart, weil keine Zeilennummer benoetigt wird. Beispiel:

Besser

```
10 FOR I=1 TO 10
20 A(I)=30 'Initialisiere A
30 NEXT I
```

Als

```
10 FOR I=1 TO 10
15 'Initialisiere A
20 A(I)=30
30 NEXT I
```

- Hinweis zum BASIC.
Wenn **BASIC** zu einer Zeilennummer verzweigt, weiss es nicht genau, wo sich die Zeile im Hauptspeicher befindet. Deshalb muss **BASIC** die Zeilennummern des Programms durchsuchen, und zwar vom Beginn des Programms bis zum Auffinden der Zeilennummer.
In einigen anderen **BASICs** muss dieses Suchen jedesmal durchgefuehrt werden, wenn eine Verzweigung im Programm stattfindet. In diesem BASIC wird das Suchen nur einmal durchgefuehrt. Danach ist die Verzweigung direkt. Das Programm laeuft also nicht schneller, wenn haeufig benutzte Unterprogramme an den Beginn des Programms gelegt werden.

LOGISCHE STEUERUNG

- Die Faehigkeiten der Anweisung IF nutzen.
Durch die Benutzung von **AND** und **OR** und der Angabe **ELSE** koennen mehrere Anweisungen **IF** und zusaetzlicher Code im Programm vermieden werden. Beispiel:

Besser

```
200 IF A=B AND C=D THEN Z=12 ELSE Z=B
```

Als

```
200 IF A=B THEN GOTO 210
205 GOTO 215
210 IF C=D THEN 225
215 Z=B
220 GOTO 230
225 Z=12
230 .....
```

- Anweisungen IF so anordnen, dass die am haeufigsten auftretende Bedingung zuerst getestet wird. Dadurch werden unnoetige Tests vermieden, z.B. wenn eine Dateneingabedatei fuer Kundenbestellungen vorliegt, die aus verschiedenen Satzarten und sehr vielen individuellen Transaktionen besteht.

Eine typische Satzgruppe sieht wie folgt aus:

Code der Satzart	**Satzart**
A	Kennsatz
B	Kundenname und -adresse
C	Transaktion
C	Transaktion
:	:
:	:
C	Transaktion
D	Endsatz

Besser

```
100 IF TYPE$="C" THEN 3000
110 IF TYPE$="A" THEN 1000
120 IF TYPE$="B" THEN 2000
130 IF TYPE$="D" THEN 4000
```

Als

```
100 IF TYPE$="A" THEN 1000
110 IF TYPE$="B" THEN 2000
120 IF TYPE$="C" THEN 3000
130 IF TYPE$="D" THEN 4000
```

Wuerden 100 Gruppen mit jeweils zehn Transaktionen pro Gruppe vorliegen, wuerden dadurch, dass der Test der Transaktionen an den Anfang der Liste gelegt wird, 1800 IF-Anweisungen weniger ausgefuehrt.

Im folgenden wird ein weiteres Beispiel angezeigt, in dem Anweisungen IF so angeordnet werden, dass so wenig Tests wie moeglich stattfinden:

Besser

```
200 IF A<>1 THEN 250
210 IF B=1 THEN X=0
220 IF B=2 THEN X=1
230 IF B=3 THEN X=2
240 GOTO 280
250 IF B=1 THEN X=3
260 IF B=2 THEN X=4
270 IF B=3 THEN X=5
280 .....
```

Als

```
200 IF A=1 AND B=1 THEN X=0
210 IF A=1 AND B=2 THEN X=1
220 IF A=1 AND B=3 THEN X=2
230 IF A<>1 AND B=1 THEN X=3
240 IF A<>1 AND B=2 THEN X=4
250 IF A<>1 AND B=3 THEN X=5
```

SCHLEIFEN

- Ganzzahlige Zaehler in FOR...NEXT-Schleifen benutzen, wo dies moeglich ist. Arithmetik mit ganzen Zahlen geht schneller als Arithmetik mit Zahlen einfacher und doppelter Genauigkeit.

- Die Variable in der Anweisung **NEXT** weglassen, wo dies moeglich ist. Wird eine Variable eingefuegt, benoetigt **BASIC** ein wenig Zeit, um zu pruefen, ob sie in Ordnung ist. Es kann noetig sein, eine Variable in der Anweisung **NEXT** anzufuegen, wenn aus geschachtelten Schleifen heraus verzweigt wird.
Siehe unter Anweisung **FOR** und **NEXT** in Kapitel 4.

- FOR...NEXT Schleifen statt Kombinationen aus Anweisung **IF** und **GOTO** benutzen. Beispiel:

Besser

```
200 FOR I=1 TO 10
:
:
300 NEXT I
```

Als

```
200 I=1
210 .....

290 I=I+1
300 IF I<=10 THEN 210
```

- Unnoetigen Code aus Schleifen entfernen.
Das sind Anweisungen, die die Schleife nicht beeinflussen, sowie nicht ausfuehrbare Anweisungen, wie **REM** und **DATA**. Beispiel:

Besser

```
10 A=B+1
20 FOR X=1 TO 100
30 IF D(X)>A THEN D(X)=A
40 NEXT X
```

Als

```
10 FOR X=1 TO 100
20 A=B+1
30 IF D(X)>A THEN D(X)=A
40 NEXT X
```

Im vorherigen Beispiel ist es nicht noetig, den Wert von A jedesmal zu berechnen, wenn die Schleife verarbeitet wird, da in der Schleife der Wert von A nie geaendert wird.

Im naechsten Beispiel wird eine nicht ausfuehrbare Anweisung gezeigt.

Besser

```
200 DATA 5,12,1943
210 FOR I=1 TO 100
:
:
300 NEXT I
```

Als

```
200 FOR I=1 TO 100
210 DATA 5,12,1943
:
:
300 NEXT I
```

Anlage_J

Suchcodes

Taste	dez.	hex.	Taste	dez.	hex.
Eing Loesch (ESC)	01	01	O	24	18
			P	25	19
! 1	02	02	{ [	26	1A
@ 2	03	03	}]	27	1B
# 3	04	04	<-┘	28	1C
¤ 4	05	05	CTRL	29	1D
% 5	06	06	A	30	1E
^ 6	07	07	S	31	1F
& 7	08	08	D	32	20
* 8	09	09	F	33	21
(9	10	0A	G	34	22
) 0	11	0B	H	35	23
_ -	12	0C	J	36	24
+ =	13	0D	K	37	25
<--	14	0E	L	38	26
|<-- -->|	15	0F	: ;	39	27
		10	" '	40	28
Q	16		~ `	41	29
W	17	11			
			↑ Links (Left)	42	2A
E	18	12			
R	19	13	| \	43	2B
T	20	14	Z	44	2C
Y	21	15	X	45	2D
U	22	16	C	46	2E
I	23	17	V	47	2F

Taste	dez.	hex.
B	48	30
N	49	31
M	50	32
< ,	51	33
> .	52	34
? /	53	35
↑ Rechts (Right)	54	36
Druck (PRTSC)	55	37
ALT	56	38
Leertaste	57	39
Gross ↓ (CAPS LOCK)	58	3A
F1	59	3B
F2	60	3C
F3	61	3D
F4	62	3E
F5	63	3F
F6	64	40
F7	65	41
F8	66	42

Taste	dez.	hex.
F9	67	43
F10	68	44
NUM ↓ (NUM LOCK)	69	45
Abbr (SROLL LOCK)	70	46
7 Posi (HOME)	71	47
8 ↑	72	48
9 Bild ↑ (PGUP)	73	49
-	74	4A
4 <--	75	4B
5	76	4C
6 -->	77	4D
+	78	4E
1 END	79	4F
2 ↓	80	50
3 Bild ↓ (PGDN)	81	51
0 Einfg (INS)	82	52
.Loesch (.DEL)	83	53

Anlage_K

Sachwortverzeichnis

Hier werden technische Ausdruecke erklaert, die in dieser BASIC-Broschuere vorkommen.

Abfangen:
Eine Reihe von Bedingungen, die ein Ereignis beschreiben, das abgefangen werden soll, und die Aktion, die nach dem Abfangen ausgefuehrt werden soll.

Abschneiden:
Die letzten Elemente einer Zeichenkette abschneiden oder Abschneiden der Kommastellen hinter einer Zahl.

Absolute Koordinatenform:
In der Grafik bedeutet dies die Position eines Punktes, abhaengig vom Ursprung des Koordinatensystems.

Adressen:
Der Platz eines Registers, eines bestimmten Teils des Hauptspeichers oder irgendeine andere Datenquelle oder, Bestimmung oder der Bezug auf eine Einheit oder auf eine Datenangabe durch die Adresse.

Adressierbarer Punkt:
Bei Computergrafiken kann jeder Punkt auf dem Bildschirm adressiert werden. Diese Punkte sind zusammen eine endliche Anzahl und bilden ein diskretes Gitter ueber dem Bildschirm.

Aktive Seite:
Beim Farb-/Grafikbildschirmanschluss der Bildschirmpuffer, der Informationen zum Bildschirm gesendet hat. Er kann von dem Bildschirmpuffer verschieden sein, dessen Information gerade angezeigt wird.

Aktiviert:
Ein Status derjenigen Verarbeitungseinheit, die gewisse Unterbrechungsarten erlaubt.

Algorithmus:
Ein endlicher Satz definierter Regeln fuer die Loesung eines Problems in einer endlichen Anzahl von Schritten.

Alphabetisches Zeichen:
Ein Buchstabe des Alphabets.

Alphanumerisch:
Ein Zeichensatz, der aus Buchstaben und Ziffern besteht.

Anfrage:
Eine Frage, die der Computer sendet, wenn Informationen eingegeben werden sollen.

Anschluss oder Adapter:
Eine Moeglichkeit, um Peripherie anzuschliessen.

Anweisung:
Ein Ausdruck, der eine Operation beschreiben oder spezifizieren kann und im Sinne der **BASIC**-Programmiersprache komplett ist.

Anwenderprogramm:
Ein Programm, das von oder fuer einen Benutzer geschrieben wurde und sich auf seine Arbeit bezieht, z.B. ein Lohnabrechnungsprogramm.

Argument:
Ein Wert, der von einem aufrufenden Programm an eine Funktion uebergeben wird.

Arithmetischer Ueberlauf:
siehe Ueberlauf

ASCII:
American National Standard Code for Information Interchange. Der Standardcode, der benutzt wird, um Informationen zwischen Datenverarbeitungssystemen und den dazugehoerigen Einheiten auszutauschen. Der ASCII-Zeichensatz besteht aus Steuerzeichen und grafischen Zeichen.

Asynchron:
Ohne regulaere Zeitbeziehung; nicht vorhersehbar in Bezug auf die Ausfuehrung von Programminstruktionen.

Attribut:
Numerischer Wert, der die Farbe jedes Bildschirmpunktes beschreibt.

Auffuellen:
Ein Block wird mit Scheindaten aufgefuellt, normalerweise mit Nullen oder Leerstellen.

Aufloesung:
In Computergrafiken das Mass der Schaerfe eines Bildes, ausgedrueckt durch die Anzahl der Zeilen pro Laengeneinheiten in diesem Bereich.

Aufruf (Invoke):
Eine Prozedur an einem ihrer Eingangspunkte aufrufen.

Ausfuehren:
Eine Instruktion oder ein Computerprogramm ablaufen lassen.

Ausmalen:
Methode, eine begrenzte Flaeche auszumalen.

Ausrichten:
Vertikales oder horizontales Ausrichten von Zeichen, um ein bestimmtes Format zu erreichen.

Ausschnitt:
Bei der grafischen Darstellung ein definierter Bereich im Weltkoordinatensystem.

Baud:
Gibt die Anzahl von Signalen pro Sekunde an.

Baumstrukturiertes Verzeichnis:
Eine Gruppe von zusammenhaengenden Dateien oder Verzeichnissen auf einem Laufwerk, die hierarchisch strukturiert sind.

Bedingungen:
Eine Bedingung, die eine Programmunterbrechung verursachen koennte. Es kann sich um die Entdeckung eines unerwarteten Fehlers handeln oder um eine Bedingung, die erwartet wurde, wobei aber die Zeit nicht vorhersagbar ist.

Bereich:
Der Satz von Werten, den eine Menge oder eine Funktion aufnehmen kann. Eine Anordnung von Elementen in einer oder mehreren Dimensionen (siehe Feld).

Bereinigen:
Wenn **BASIC** Zeichenkettenbereiche verdichtet, indem es alle nuetzlichen Daten sammelt und unbenutzte Bereiche des Hauptspeichers freimacht, die vorher fuer Zeichenketten benutzt wurden.

Betriebssystem:
Software, die die Ausfuehrung von Programmen steuert. Bezieht sich auf DCP.

Binaer:
Bezieht sich auf eine Bedingung, die aus zwei Werten bestehen kann. Siehe auch Nummernsystem zur Basis 2.

Bit:
Eine Binaerziffer.

Blaettern/Verschieben:
Bildschirminhalt nach oben, unten, rechts oder links verschieben, so dass neue Informationen angezeigt werden.

Blinken:
Beabsichtigtes regelmaessiges Wechseln der Anzeigeintensitaet eines Zeichens auf dem Bildschirm.

Bool'scher Wert:
Ein numerischer Wert, der als "richtig" oder "wahr" (falls er nicht Null ist) oder "falsch" (falls er Null ist) interpretiert wird.

bps:
Bits pro Sekunde.

Byte:
Enthaelt die binaere Darstellung eines Zeichens und besteht aus acht Bits.

Call:
Um ein Programm oder ein Unterprogramm aufzurufen, gewoehnlich durch die Angabe der Eingangsbestimmungen und Verzweigung zu einem Eingangspunkt.

Code:
Zur Darstellung von Daten oder von einem Computerprogramm in symbolischer Form, das vom Computer angenommen werden kann; das Schreiben einer Routine. Auch ein oder mehrere Computerprogramme oder der Teil eines Programms.

Datei:
Ein Satz zusammenhaengender Saetze, die als Einheit betrachtet werden.

Dateiende (EOF):
Ein Markierungszeichen, das sofort hinter dem letzten Dateisatz steht und damit das Ende der Datei anzeigt.

Datenfernverarbeitung:
Die Uebertragung und das Empfangen von Daten.

Datenstation:
Einheit, gewoehnlich ausgeruestet mit einer Tastatur und einem Bildschirm, der faehig ist, Informationen zu senden und zu empfangen.

Diagnose:
Hilft bei der Auffindung und Eingrenzung einer Funktion, die falsch ist, oder eines Fehlers.

Doppelte Genauigkeit:
Beschreibt das Speichern eines numerischen Wertes in 8 Byte des Speichers und wird benutzt, um eine groessere Genauigkeit bei mathematischen Berechnungen zu erhalten.

DCP:
Disk Control Program
Betriebssystem fuer robotron EC 1834.

Duplex:
In der Datenfernverarbeitung bezieht es sich auf eine unabhaengige simultane Zweiwege-Uebertragung in beide Richtungen.

Dynamisch:
Passiert zur Zeit der Ausfuehrung.

Echo:
Reflektiert empfangene Daten zum Sender.
Beispiel:
Die Antwort auf das Betaetigen einer Taste ist normalerweise die Anzeige eines Zeichens am Bildschirm.

Einfache Genauigkeit:
Methode, einen numerischen Wert in nur 4 Byte des Speichers zu speichern.

Element:
Ein Teil eines Satzes, im besonderen eine Angabe in einem Bereich.

Ereignis:
Bezieht sich besonders auf Ereignisse, die durch **COM(n)**, **PLAY(n)** und **KEY(n)** getestet werden.

Ergebnisverfolgung:
Eine Reihe von Bedingungen, die ein abzufangendes Ereignis und die anschliessend ausgefuehrten Aktionen beschreiben.

Erweiterung:
Ein fortlaufender Platz auf einer Diskette, der fuer eine bestimmte Datei belegt oder reserviert ist.

Fehlender Operand:
Fehlende Daten in Anweisung bewirken, dass diese Anweisung nicht ausfuehrbar ist.

Fehler:
Eine Bedingung, durch die eine Einheit nicht in der geforderten Weise arbeitet.

Feld:
In einem Satz ist dies ein angegebener Bereich, der benutzt wird, um bestimmte Daten zu speichern (sh. Bereich).

Feste Laenge:
Die Laenge veraendert sich nicht. Dateien fuer Direktzugriff haben z.B. eine feste Satzlaenge, d.h. jeder Satz der Datei hat die gleiche Laenge.

Format:
Eine bestimmte Anordnung von Daten auf einem Datenmedium, wie z.B. Bildschirm oder Diskette.

Formularvorschub (FF):
Ein Zeichen, durch das die Druck- oder Anzeigeposition zur naechsten Seite gebracht wird.

Fortschreiben:
Normalerweise eine Stammdatei mit laufenden Informationen aendern.

Fuehrend:
Fuehrende Nullen oder fuehrende Leerstellen als erster Teil z.B. einer Zeichenkette.

Funktion:
Eine Prozedur, die einen Wert uebergibt, der vom Wert einer oder mehrerer unabhaengigen Variablen in einem spezifizierten Weg abhaengt.

Funktionstaste:
Eine der zehn Tasten mit den Namen **F1** bis **F10** auf der obersten Reihe der Tastatur.

Ganze Zahl:
Eine der Zahlen 0, ±1, ±2, ±3, ...

Gedruckte Maschinenausgabe (Hard copy):
Eine gedruckte Kopie einer Maschinenausgabe in einer visuellen lesbaren Form.

Genauigkeit:
Die Genauigkeit der Zahlendarstellung. Fuer eine Maschine kann dies genau angegeben werden und bezieht sich auf die Groesse des Fehlers zwischen der aktuellen Zahl und ihrem Wert, wie er in der Maschine gespeichert ist.

Grafik:
Ein Symbol, das durch einen Prozess erzeugt wird, wie z.B. Handschrift, Drucken oder Zeichnen.

Halbduplex:
Bezieht sich in der Datenfernverarbeitung auf eine alternierende unanhaengige Uebertragung in eine Richtung zu einer Zeit.

Hauptspeicher:
Speicher, den man an jeder gewuenschten Position lesen und beschreiben kann.

Hertz (Hz):
Eine Frequenzeinheit gleich einem Zyklus pro Sekunde.

Hierarchie:
Eine Struktur, die aus mehreren Ebenen besteht, die in Baumform angeordnet sind. Hierarchie von Operationen bezieht sich auf die relative Prioritaet, die arithmetischen oder logischen Operationen zugeordnet ist, die ausgefuehrt werden sollen.

Hintergrund:
Der Bereich, der ein Objekt umgibt, in diesem Falle den Teil des Bildschirms, der ein Zeichen umgibt.

Implizite Definition:
Das Festsetzen einer Dimension fuer ein Feld, ohne es explizit mit der Anweisung **DIM** definiert zu haben.

Inaktivieren:
Ein Status, der das Auftreten gewisser Unterbrechungsarten verhindert.

Index:
Eine Zahl, die die Position eines Elements in einem Feld identifiziert.

Inhaltsverzeichnis:
Eine Tabelle von Angaben, die sich auf bestimmte Daten beziehen. Das Inhaltsverzeichnis der Diskette enthaelt z.B. die Namen der Dateien auf der Diskette sowie Informationen, die dem DCP sagen, wo die Datei auf der Diskette zu finden ist.

Initialisieren:
Zum Setzen von Zaehlern, Schaltern, Adressen oder Inhalten des Hauptspeichers auf Null oder einen anderen Startwert zu Beginn oder an vorgeschriebenen Punkten einer Operation eines Computerprogramms.

Instruktion:
In einer Programmiersprache ein Ausdruck, der eine Operation und seine Operanden, falls welche vorhanden sind, angibt.

Interpretieren:
Uebertragen und Ausfuehren jeder Anweisung eines Computerprogrammes in der Quellsprache, bevor die naechste Anweisung uebertragen und ausgefuehrt wird.

K:
Auf die Hauptspeicherkapazitaet bezogen bedeutet dies 2^{10} oder 1024 in dezimaler Darstellung.

Kanal:
Ein Pfad, auf dem Signale gesendet werden koennen, z.B. ein Datenkanal oder ein Ausgabekanal.

Kennzeichen:
Eine der verschiedenen Arten von Indikatoren, die fuer eine Identifikation benutzt werden, z.B. ein Zeichen, das das Auftreten einer Bedingung signalisiert.

Kommentar:
Eine Anweisung, die benutzt wird, um ein Programm zu dokumentieren. Kommentare sollten Informationen enthalten, die beim Ablauf des Programms oder bei der Durchsicht der Ausgabeliste hilfreich sein koennen.

Komplement:
Das Gegenteil. In diesem Fall eine Zahl, die von einer gegebenen Zahl abgeleitet werden kann, indem man sie von einer anderen gegebenen Zahl abzieht.

Komprimieren:
Daten so anordnen, dass sie nur einen minimalen Speicherraum beanspruchen.

Konstante:
Ein fester Wert oder eine Datenangabe.

Koordinaten:
Zahlen, die den Platz auf dem Bildschirm identifizieren.

Koordinatenuebertragung:
Uebertragung von Koordinatenwerten zwischen dem Weltkoordinatensystem, wie durch die Anweisung **WINDOW** festgelegt, und dem physischen Koordinatensystem, wie durch die Anweisung **VIEW** festgelegt.

Kopfsatz:
Ein Satz, der allgemeine oder konstante oder identifizierende Informationen ueber eine Anzahl von Saetzen enthaelt, die folgen.

Korrigieren:
Zur Eingabe, zum Aendern und zum Loeschen von Daten.

Kursor:
Eine bewegliche Markierung, die benutzt wird, um einen Platz auf dem Bildschirm anzuzeigen.

Laderoutine:
Eine existierende Version, vielleicht eine primitive Version eines Computerprogramms, das benutzt wird, um eine andere Version des Programms aufzubauen. Man kann es sich als ein Programm vorstellen, das sich selbst laedt.

Laufendes Verzeichnis:
Das Standardinhaltsverzeichnis fuer jedes Laufwerk im System. Es ist das Verzeichnis, in dem BASIC nach einem eingegebenen Dateinamen sucht, wenn nicht angegeben wurde, in welchem Verzeichnis sich die Datei befindet.

Leerstellen:
Ein Teil eines Datenmediums, in dem kein Zeichen gespeichert ist.

Literal:
Eine explizite Darstellung eines Wertes, besonders eines Zeichenkettenwerts; eine Konstante.

M:
Mega; eine Million. Bezieht es sich auf den Hauptspeicher, so ist es 2^{20} oder 1.048.576 in dezimaler Schreibweise.

Mantisse:
Fuer eine Zahl, die in Gleitkommadarstellung ausgedrueckt wird, die Zahl, die nicht der Exponent ist.

Maschinenunendlichkeit:
Die groesste Zahl, die im internen Computerformat dargestellt werden kann.

Maske:
Eine Zeichenschablone, die benutzt wird, um die Beibehaltung oder Eliminierung einer anderen Zeichenschablone zu steuern.

Massstab:
Aenderung der Darstellung einer Menge, indem sie in anderen Einheiten ausgedrueckt wird.

Massstabaenderung:
Aendern der Groesse des Abbildes in Grafiken, indem der Ausschnitt geaendert wird (siehe Anweisung **VIEW**).

Matrix:
Ein Feld aus zwei oder mehr Dimensionen.

Matrixdrucker:
Ein Drucker, bei dem jedes Zeichen durch eine Punktschablone dargestellt ist.

Menue:
Eine Liste der verfuegbaren Operationen. Man waehlt die gewuenschte Operation von der Liste aus.

Minifloppy:
Eine 5 1/4"-Diskette.

Nachstehend:
Am Ende einer Zeichenkette oder Zahl stehend. Die Zahl 1000 hat z.B. drei nachstehende Nullen.

Notation:
Ein Satz von Symbolen und die Regeln fuer ihre Benutzung in der Darstellung von Daten.

Null:
Leer, ohne Bedeutung. Meistens eine Zeichenkette ohne Inhalt.

Oktal:
Das Zahlensystem zur Basis 8.

Operand:
Das was verarbeitet wird.

Operation:
Definierte Aktion, durch die bei Anwendung einer erlaubten Kombination bekannter Einheiten eine neue Einheit erzeugt wird.

Palette:
Farbauswahl

Parameter:
Ein Name in einer Prozedur, der benutzt wird, um ein Argument anzusprechen, das zu dieser Prozedur uebertragen wurde.

Paritaetspruefung:
Eine Technik zum Testen uebertragener Daten. Normalerweise wird eine Binaerziffer angehaengt, um die Summe aller Ziffern immer gerade (gerade Paritaet) oder immer ungerade (ungerade Paritaet) zu machen.

Pfad:
Weg, um eine bestimmte Datei zu finden. Wird bei Verzeichnissen und Kommandos oder Anweisungen, die Dateispezifikationen akzeptieren, benutzt.

Physisches Koordinatensystem:
Die logischen Grenzen des Bildschirmes (siehe Anweisungen **VIEW** und **WINDOW**).

Platz:
Eine Adresse, an der Daten gespeichert werden koennen.

Port (Tor):
Ein Anschlusspunkt fuer Datenein- oder ausgabe.

Position:
In einer Zeichenkette jeder Satz, der durch ein Zeichen besetzt werden und durch eine Nummer identifiziert werden kann.

Puffer:
Ein Bereich im Hauptspeicher, der benutzt wird, um eine Differenz in der Uebertragungsrate von Daten oder der Zeit fuer das Auftreten von Ereignissen zu kompensieren, wenn Daten von einer Einheit zu einer anderen uebertragen werden. Bereich, der fuer Ein-/Ausgabeoperationen reserviert ist, aus dem Daten gelesen oder in den Daten geschrieben werden.

Punkte ausserhalb des Koordinatenbereiches:
Ausserhalb des Koordinatenbereiches angesprochene Punkte sind innerhalb des Koordinatenbereiches nicht sichtbar. Alle ueber die Grenzen des Koordinatenbereiches hinausgehenden Punkte werden abgeschnitten, so dass nur die Punkte innerhalb des Bereiches angezeigt werden.

Rasterpunkt:
Ein Punkt oder ein Platz auf dem Bildschirm, der benutzt wird, um eine Abbildung auf dem Bildschirm darzustellen. Auch die Bits, die die Angaben fuer diesen Punkt enthalten.

Rasterverfahren:
Verfahren, bei dem ein Bild durch Anzeige einzelner Zeilen auf dem ganzen Bildschirm erzeugt wird.

Read-only:
Ein Typ des Datenzugriffs, der nur Lesen aber kein Aendern erlaubt.

Rekursiv:
Ein Prozess, in dem jeder Schritt das Ergebnis frueherer Schritte benutzt, wie z.B. eine Funktion, die sich selbst aufruft.

Relative Koordinaten:
In der Grafik sind dies Werte, die den Platz eines Punktes identifizieren, indem die Entfernung zu einem anderen Punkt angegeben wird.

Relativzeiger (Offset):
Die Anzahl der Einheiten von einem Startpunkt (in einem Satz, Steuerblock oder im Hauptspeicher) zu einem anderen Punkt. Zum Beispiel wird in **BASIC** die aktuelle Adresse eines Hauptspeicherplatzes als Relativzeiger in Bytes ab dem Platz gegeben, der in der Anweisung **DEF SEG** definiert ist.

Reserviertes Wort:
Ein Wort, das in **BASIC** fuer einen speziellen Zweck definiert ist und das nicht als Variablennamen benutzt werden darf.

Routine:
Teil eines Programmes oder eine Folge von Anweisungen, die durch ein Programm aufgerufen werden, die haeufig verwendet werden.

Satz:
Eine Sammlung von Informationen, die sich aufeinander beziehen und als Einheit angesehen werden. Zum Beispiel kann bei der Lagerverwaltung jede Rechnung ein Satz sein.

Scan:
Sequentiell Teil fuer Teil pruefen.

Schachtelung:
Die Struktur einer Art in einer anderen Struktur der gleichen Art aufnehmen. Zum Beispiel kann man Schleifen innerhalb anderer Schleifen schachteln oder Unterprogramme von anderen Unterprogrammen aufrufen.

Scheinargument:
Sieht wie eine definierte Angabe aus, kann aber nicht wie eine Angabe funktionieren, z.B. das Scheinargument einer Funktion.

Schleife:
Ein Instruktionssatz, der wiederholt ausgefuehrt werden kann, solange eine bestimmte Bedingung wahr ist.

Schluesselwort:
Eines der vordefinierten Woerter einer Programmiersprache; ein reserviertes Wort.

Schnittstelle:
Eine gemeinsame Grenze.

Schreiben:
Daten in einer Speichereinheit oder auf einem Datenmedium aufzeichnen.

Schrittweite:
Ein Wert, der benutzt wird, um einen Zaehler zu veraendern.

Schutz:
Um den Zugriff zu einem oder Teilen eines Datenverarbeitungssystems zu verhindern.

Segment:
Ein bestimmter 64K-Byte-Bereich des Hauptspeichers.

Seite:
Teil des Bildschirmpuffers, der unabhaengig angezeigt und/oder geschrieben werden kann.

Sequentieller Zugriff:
Eine Zugriffsmethode, bei der Saetze in der gleichen Reihenfolge abgerufen werden, in der sie geschrieben wurden. Jeder neue Zugriff auf die Datei bezieht sich auf den naechsten Satz der Datei.

Sicherungskopie:
Bezieht sich auf ein System, eine Einheit, Datei, die im Falle einer Falschfunktion oder bei Verlust von Daten verwendet werden kann.

Skalar:
Ein Wert oder eine Variable, die kein Feld ist.

Speicher:
Eine Einheit oder ein Teil einer Einheit, der Daten enthalten kann. Hauptspeicher

Stammverzeichnis:
Das Verzeichnis, das beim Formatieren einer Diskette erstellt wird. Das Basis- oder Hauptverzeichnis.

Standardannahme:
Ein Wert oder eine Auswahl, die angenommen wird, wenn nichts angegeben wurde.

Stapeln (Stack):
Eine Methode, Daten temporaer so zu speichern, dass die letzte gespeicherte Angabe als erste Angabe verarbeitet wird.

Steuerzeichen:
Ein Zeichen, dessen Auftreten in einem bestimmten Text eine Steueroperation einleitet, veraendert oder stoppt. Eine Steueroperation ist eine Aktion, die das Aufnehmen, die Verarbeitung, Uebertragung oder Interpretation von Daten beeinflusst, z.B. Wagenruecklauf oder Uebertragungsende.

Stopp-Bit:
Ein Signal, das einem Zeichen oder Block folgt und die empfangende Einheit darauf vorbereitet, das naechste Zeichen oder den naechsten Block zu empfangen.

Syntax:
Die Regeln fuer die Struktur einer Sprache.

Tabelle:
Eine Anordnung von Daten in Zeilen und Spalten.

Taktgeber:
Eine Einheit, die periodisch Signale erzeugt, die fuer die Synchronisation benutzt werden.

Testen:
Fehler in einem Programm auffinden und entfernen.

Tor:
Siehe Port.

Trennzeichen:
Ein Zeichen, das Woerter oder Werte in einer Eingabezeile gruppiert oder trennt.

Ueberlagerung:
Benutzung derselben Bereiche des Hauptspeichers fuer verschiedene Teile eines Programms zu verschiedenen Zeiten.

Ueberlauf:
Das Ergebnis einer Operation ueberschreitet die Kapazitaet einer angegebenen Speichereinheit.

Ueberschreiben:
Einen Bereich des Hauptspeichers beschreiben, wobei die Daten, die vorher gespeichert waren, zerstoert werden.

Umkehranzeige:
Hervorheben eines Zeichenfeldes oder des Kursors, indem dessen Farbe mit der Hintergrundfarbe vertauscht wird.

Umwandlungsprogramm:
Programm, das den Programmcode einer Programmiersprache in den Maschinencode des Computers umsetzt. Ein Compiler ist ein Umwandlungsprogramm fuer hoehere Programmiersprachen wie **BASIC, FORTRAN.** Ein Assembler ist ein Umwandlungsprogramm fuer die mnemonische Sprache des Maschinencodes eines Computers.

Unterbrechung:
Eine Verarbeitung auf eine solche Art stoppen, dass sie wieder aufgenommen werden kann.

Unterprogramm:
Teil eines Programms oder eine Folge von Instruktionen, die von einem Programm aufgerufen werden, und fuer allgemeine oder haeufige Benutzung bestimmt sind.

Unterverzeichnis:
Jedes Verzeichnis, das im Stammverzeichnis oder in einem anderen Unterverzeichnis enthalten ist.

Variable:
Eine Quantitaet, die irgendeinen Wert eines gegebenen Satzes von Werten annehmen kann.

Variabel langer Satz:
Ein Satz, der eine Laenge unabhaengig von der Laenge anderer Saetze in der Datei hat.

Vektor:
In der Grafik ein in eine bestimmte Richtung weisendes Zeilensegment. Im allgemeinen ein geordneter Satz von Zahlen und daher ein eindimensionales Feld.

Verdichtung:
Eine Datenanordnung, die einen minimalen Platz benoetigt.

Vergroessern:
Vergroessern eines Ausschnittes der Bildschirmanzeige.

Verkettung:
Durch diese Operationen werden zwei Zeichenketten in der angegebenen Reihenfolge verbunden und bilden eine einzelne Zeichenkette mit einer Laenge gleich der Summe der Laenge der zwei Zeichenketten.

Verkleinern:
Verwendung eines Ausschnittes, dessen Koordinaten groesser sind als das anzuzeigende Bild. Das Bild wird so lange verkleinert, bis nur noch ein Punkt auf dem Bildschirm zu sehen ist.

Verschachtelung:
Die Struktur einer Art in einer anderen Struktur der gleichen Art aufnehmen. Zum Beispiel kann man Schleifen innerhalb anderer Schleifen schachteln oder Unterprogramme von anderen Unterprogrammen aufrufen.

Verschieben:
Den ganzen oder Teile des Bildschirms vertikal oder horizontal verschieben, so dass neue Daten auf einer Seite erscheinen und alte Daten auf der anderen Seite verschwinden.

Vordergrund:
Der Teil des Bildschirmbereichs, der aus dem Zeichen selbst besteht.

Weltkoordinatensystem:
Ein Koordinatensystem, das sich gedanklich bis ins Unendliche fortsetzt.

Wagenruecklauf (CR):
Ein Zeichen, durch das die Druck- oder Bildschirmanzeigeposition an die erste Position derselben Zeile zurueckgesetzt wird.

Warteschlange:
Eine Zeile oder eine Liste von Angaben, die auf Bedienung warten; die erste Angabe, die in der Schlange steht, wird als erste bedient.

Zeichen:
Ein Buchstabe, eine Ziffer oder ein anderes Symbol, das als Teil der Organisation, Steuerung oder Datendarstellung benutzt wird. Eine verbundene Folge von Zeichen nennt man eine Zeichenkette.

Zeichenkette:
Eine Folge von Zeichen.

Zeichensatz:
Eine Gruppe oder ein Satz mit Zeichen einer bestimmten Groesse oder eines bestimmten Types.

Zeichenumsetzung:
Eine Technik, Daten in eine gewuenschte Form umzuwandeln, wenn sie nicht in dieser Form eingegeben werden. Zum Beispiel koennen kleine Buchstaben in grosse Buchstaben umgewandelt werden.

Zeile:
Bezogen auf einen Text, Bildschirm oder Drucker bedeutet sie eine oder mehrere Zeichen als Ausgabe, bevor auf die erste Druck- oder Anzeigeposition zurueckgegangen wird. Fuer Eingabe bedeutet sie eine Zeichenkette, die vom System als ein Eingabeblock angenommen wird; z.B. alle Zeichen, die eingegeben werden, bevor die Eingabetaste betaetigt wird. In der Grafik bedeutet sie eine Punktserie, die auf dem Bildschirm gezeichnet wird, um eine gerade Linie zu zeichnen. In der Datenfernverarbeitung bedeutet sie ein physikalisches Medium wie z.B. ein Draht oder ein Mikrowellenstrahl, der benutzt wird, um Daten zu uebertragen.

Zeilenvorschub (LF):
Ein Zeichen, durch das die Druck- oder Anzeigeposition auf die gleiche Position der naechsten Zeile gebracht wird.

Zentralsystem:
Der primaere oder steuernde Computer in einer Installation mehrerer Computer.

Zugriffsmethode:
Eine Technik, die benutzt wird, um einen speziellen Satz aus einer Datei zu lesen oder in eine Datei zu schreiben.

Zuordnung:
Eine bestimmte Quelle, wie z.B. eine Diskettendatei oder einen Teil des Hauptspeichers, einer speziellen Aufgabe zuordnen.

Zweierkomplement:
Eine Form der Darstellung negativer Zahlen im binaeren Zahlensystem.

Zyklische Adressfolge:
Die Technik, Angaben anzuzeigen, deren Koordinaten ausserhalb des Bildschirmbereichs liegen.

III-12-12 Kv 315/89