IBM

Personal Computer

BASIC Handbuch

Personal Computer Hardware Reference Library

BASIC Handbuch

Diese Veröffentlichung ist eine Übersetzung des:
IBM Personal Computer Hardware Reference Library BASIC, 6025013
herausgegeben von International Business Machines Corporation, USA

Zuständig für den Inhalt dieser Veröffentlichung:
IBM Deutschland GmbH
ZV Neue Produkte 2125

Herausgegeben von:
VU Literatur 0426
Oktober 1982

Vorwort

Das BASIC des IBM Personalcomputers besteht aus drei aufwärts verträglichen Versionen: Kassette, Diskette und erweitertes BASIC. Dieses Referenzhandbuch bezieht sich auf alle drei Versionen des BASIC, Release 1.10. Der Ausdruck BASIC bezieht sich in diesem Handbuch auf alle BASIC-Versionen.

Das BASIC-Umwandlungsprogramm für den IBM Personalcomputer ist eine zusätzlich von IBM verfügbare Software.

Benutzung dieses Handbuchs

Wer dieses Handbuch benutzt, sollte sich mit allgemeinen Programmierkonzepten auskennen. Das Handbuch versucht nicht zu lehren, wie ein Programm geschrieben wird.

Das Handbuch besteht aus vier Kapiteln und einer Anzahl Anhängen.

Kapitel 1 gibt einen kurzen Überblick über die drei BASIC-Versionen des IBM Personalcomputers.

In Kapitel 2 wird gesagt, was benötigt wird, um auf dem Computer mit BASIC zu beginnen. Es wird beschrieben, wie der Computer mit Hilfe von BASIC bedient wird.

In Kapitel 3 werden viele Dinge besprochen, die man wissen muß, bevor man mit der Programmierung beginnt. Viele Informationen beziehen sich auf die Datendarstellung, wenn BASIC benutzt wird. Es werden Dateinamen zusammen mit vielen speziellen Ein- und Ausgabeeinrichtungen erklärt, die im BASIC des IBM Personalcomputers verfügbar sind.

Kapitel 4 ist der Referenzabschnitt. Es enthält die Syntax und Semantik jedes Befehls, jeder Anweisung und Funktion in BASIC, alphabetisch geordnet.

Die Anhänge enthalten andere nützliche Informationen, wie z.B. eine Liste der Fehlermeldungen, ASCII-Codes und mathematischen Funktionen; auch hilfreiche Informationen über Unterprogramme in der Maschinensprache, Diskettenein- und -ausgabe und Datenfernverarbeitung. Sie enthält auch detaillierte Informationen über fortgeschrittene Subjekte für den erfahrenen Programmierer.

Es wird geraten, Kapitel 2 und 3 vollständig zu lesen, um BASIC kennenzulernen. Während der Programmierung kann man dann auf Kapitel 4 zugreifen, um über jeden benutzten Befehl oder Anweisung die benötigte Information zu bekommen.

Syntaxdiagramm

Alle Befehle, Anweisungen und Funktionen, die in diesem Buch beschrieben werden, haben ihre Syntax, die nach den folgenden Konventionen beschrieben ist:

- Wörter in Großbuchstaben sind Schlüsselwörter und müssen, wie gezeigt, in Klein- oder Großbuchstaben eingegeben werden. BASIC wandelt Wörter immer in Großbuchstaben um (außer sie sind Teil einer in Anführungszeichen stehenden Zeichenkette, eine Bemerkung oder eine DATA-Anweisung).

- Alle Angaben in kleingeschriebenen kursiven Buchstaben müssen angegeben werden.

- Angaben in eckigen Klammern ([]) sind wahlfrei.

- Mehrere Punkte hintereinander (...) zeigen an, daß eine Angabe so oft wie gewünscht wiederholt werden kann.

- Jede Interpunktion außer eckigen Klammern (wie z.B. Kommas, Klammern, Semikolons, Hochkommas oder Gleichheitszeichen) müssen, wo sie angezeigt werden, eingefügt werden.

Beispiel:

INPUT[;]["*abfrage*";] *variable*[,*variable*]...

Damit die Anweisung INPUT gültig wird, muß zuerst das Schlüsselwort INPUT wahlfrei, gefolgt von einem Semikolon, eingegeben werden. Danach kann eine *abfrage* zwischen Anführungszeichen eingefügt werden. Nach einer *abfrage* muß ein Semikolon stehen. Mindestens eine *variable* wird für eine INPUT-Anweisung benötigt. Es können mehr als eine *variable* angegeben werden, wenn sie durch Kommas getrennt sind.

Detaillierte Information über jeden Parameter stehen im Text nach jedem Diagramm. Die Information über dieses Beispiel steht in Kapitel 4 unter "Anweisung INPUT".

Referenzliteratur

Die folgenden Broschüren enthalten Informationen, die nützlich sein können:

- *IBM Personalcomputer Bedienerhandbuch.*
- *IBM Personalcomputer Betriebssystem (DOS) Handbuch.*
- *IBM Personalcomputer Technisches Handbuch.*

Zusammenfassung der Änderungen

Die folgenden Änderungen wurden in Release 1.10 vorgenommen:

- Jede Auflistung auf dem Bildschirm oder dem Drucker kann durch Betätigen der Tasten Ctrl-Break beendet werden.

- Die Drucker (LPT1:, LPT2: und LPT3:) können im Modus für wahlfreien Zugriff eröffnet werden. In Release 1.00 wurden diese Einheiten immer für sequentielle Ausgabe eröffnet. Dadurch fügte BASIC ein Zeilenvorschubzeichen nach jedem Schreibkopfrücklaufzeichen an.

  Wird der Drucker im Modus für wahlfreien Zugriff mit einer Breite von 255 Zeichen eröffnet, wird der Zeilenvorschub nach dem Schreibkopfrücklauf unterdrückt, so daß alle Zeichen ohne Änderung zum Drucker gesendet werden können. In diesem Modus können verschiedene Typen von Grafikdruckern unterstützt werden.

- Die OPEN "COM... -Anweisung enthält die folgenden neuen Auswahlen:

  RS unterdrückt RTS (Request To Send - Sendeeinheit einschalten)

  CS[*n*] steuert CTS (Clear To Send - Sendebereitschaft)

  DS[*n*] steuert DSR (Data Set Ready - Betriebsbereitschaft)

  CD[*n*] steuert CD (Carrier Detect - Empfangssignalpegel)

  LF sendet nach jedem Schreibkopfrücklauf ein Zeilenvorschubzeichen

  Auch die Auswahl LEN=*nummer* wurde der OPEN "COM...- Anweisung hinzugefügt, um die maximale Byteanzahl anzugeben, die mit GET oder PUT vom Dateipuffer gelesen werden können. Diese Auswahl wurde zur Verträglichkeit mit dem BASIC-Umwandlungsprogramm eingefügt.

- Die Funktion STRIG im erweiterten BASIC kann jetzt vier Spielpultknöpfe lesen.*

- Die Funktion VARPTR$ wurde hinzugefügt. Dadurch ist die Verträglichkeit zum Umwandlungsprogramm gegeben, das diese Funktion für DRAW und PLAY benötigt.

*Das ist hilfreich, wenn vier eindimensionale Spielkonsolen vorhanden sind.

INHALT

Notizen:

KAPITEL 1. BASIC-VERSIONEN

Inhalt

Notizen:

BASIC-Versionen

Für den BASIC- Interpretierer des IBM Personalcomputers gibt es drei Versionen:

- Kassette
- Diskette
- Erweitert

Die drei BASIC-Versionen sind aufwärts verträglich. Das Disketten-BASIC enthält Zusätze zum Kassetten-BASIC und das erweiterte BASIC enthält Zusätze zum Disketten-BASIC. Diese Zusätze werden nachstehend im Detail besprochen.

Die BASIC-Befehle, -Anweisungen und -Funktionen für alle drei Versionen werden in allen Einzelheiten im "Kapitel 4. BASIC-Befehle, -Anweisungen, -Funktionen und -Variablen" beschrieben. Jede Beschreibung enthält einen Abschnitt mit dem Namen **Versionen:** dort wird gesagt, in welcher Version von BASIC der Befehl, die Anweisung oder Funktion unterstützt wird.

So steht z.B. unter "Anweisung CHAIN" im Kapitel 4:

Versionen:	Kassette	Disk	Erweitert	Umwandlungs-
		***	***	programm
				(**)

Disk steht für Diskette, Umwandlung für Umwandlungsprogramm. Die Sterne (***) zeigen an, welche Version von BASIC die Funktion unterstützt. Das Beispiel zeigt, daß die Anweisung CHAIN in Programmen für die Disketten- und erweiterte BASIC-Versionen benutzt werden kann.

In diesem Beispiel stehen die Sterne unter dem Wort "Umwandlungsprogramm" in Klammern. Dies bedeutet, daß die Anweisung mit dem BASIC Interpretierer anders als mit dem Umwandlungsprogramm arbeitet. Das BASIC-Umwandlungsprogramm ist ein zusätzliches Software-Paket für den IBM Personalcomputer.

Kassetten-BASIC

Der BASIC-Kern ist die Kassettenversion, die in den 32K-Festspeicher (ROS) des IBM Personalcomputers eingebaut ist. Das Kassetten-BASIC des IBM Personalcomputers läuft auf jeder Hauptspeichergröße. Die Hauptspeichergröße, die für Programme und Daten verwendet werden kann, hängt davon ab, wieviel Hauptspeicher der IBM Personalcomputer enthält. Die Anzahl der "freien Bytes" wird nach dem Einschalten des Computers angezeigt.

Die einzige Speichereinheit, die zum Speichern von Informationen benutzt werden kann, ist im Kassetten-BASIC der Kassettenrecorder. Disketten können mit dem Kassetten-BASIC nicht benutzt werden.

Einige spezielle Einrichtungen, die hier und in den anderen zwei Versionen des BASIC vorhanden sind:

- Ein erweiterter Zeichensatz von 256 verschiedenen Zeichen, die angezeigt werden können. Zusätzlich zu den normalen Buchstaben, Zahlen und Sonderzeichen gibt es auch internationale Zeichen wie ñ, â und ¿. Es gibt auch Symbole für wissenschaftliche und mathematische Anwendungen, wie z.B. griechische Buchstaben und eine Vielzahl anderer Symbole.

- Grafische Fähigkeiten. Besitzt der Computer den Adapter für Farb-/Grafik-Bildschirm, so kann man Punkte, Linien, sogar ganze Bilder zeichnen. Der Bildschirm kann mit allen Punkten in mittlerer und hoher Auflösung adressierbar sein. Weitere Informationen darüber siehe Kapitel 3.

- Spezielle Ein-/Ausgabeeinheiten. Der IBM Personalcomputer besitzt einen Lautsprecher, mit dem Töne erzeugt werden können. BASIC unterstützt auch einen Lichtstift und Spielpulte, welche dazu dienen, Programme interessanter zu gestalten und als Spielgerät zu benutzen.

Disketten-BASIC

Diese BASIC-Version kommt als Programm auf einer IBM Diskette für das Betriebssystem (DOS) vor. DOS ist ein eigens bei der IBM verfügbares Produkt. Vor der Benutzung muß BASIC in den Hauptspeicher geladen werden. Das Disketten-BASIC benötigt eine Maschine mit angeschlossener Disketteneinheit und mindestens 32K Hauptspeicher. Die freie Hauptspeichergröße, die für Programme und Daten benutzt werden kann, wird am Bildschirm angezeigt, wenn BASIC gestartet wird.

Spezielle Einrichtungen des Disketten-BASIC sind:

- Zusätzlich zur Kassettenausgabe Ein-/Ausgabe auf die Diskette. Siehe Anhang B. BASIC-Disketteneinein- und -ausgabe, wo besondere Betrachtungen für die Benutzung von Diskettendateien enthalten sind.

- Eine interne "Uhr," die Datum und Zeit anzeigt.

- Asynchrone Übertragung (RS232). Einzelheiten siehe im "Anhang F. Datenfernverarbeitung."

- Unterstützung für zwei zusätzliche Drucker.

Erweitertes BASIC

Das erweiterte BASIC für den IBM Personalcomputer beinhaltet alle Sprachelemente des Kassetten- und Disketten-BASIC und Erweiterungen. Wie das Disketten-BASIC besteht es aus einem Programm auf einer IBM DOS-Diskette, die vor der Benutzung in den Hauptspeicher geladen werden muß. Das erweiterte BASIC benötigt eine Maschine mit Disketteneinheit und mindestens 48K Hauptspeicher. Wie bei den anderen Versionen werden die Anzahl der freien Bytes für Programme und Daten auf dem Bildschirm angezeigt, wenn BASIC gestartet wird.

Einrichtungen, die nur im erweiterten BASIC vorhanden sind:

- Ereignisunterbrechung. Ein Programm kann durch ein spezielles Ereignis automatisch zu einer bestimmten Programmzeile verzweigen. Ereignisse beinhalten: Datenfernverarbeitungsaktivität, eine betätigte Funktionstaste, ein betätigter Knopf an einem Spielpult und den aktivierten Lichtstift.

- Erweiterte Grafiken. Zusätzliche Anweisungen sind CIRCLE, PUT, GET, PAINT und DRAW. Diese Operationen erleichtern die Erstellung komplexerer Grafiken mit dem Adapter für Farb-/Grafikbildschirm.

- Erweiterte Musikunterstützung. Die Anweisung PLAY erleichtert die Benutzung des eingebauten Lautsprechers, um Musiktöne zu erzeugen.

Notizen:

KAPITEL 2. STARTEN UND BENUTZEN VON BASIC

Inhalt

BASIC starten

Es ist einfach, mit dem IBM Personalcomputer BASIC zu starten.

Kassetten-BASIC starten:

Einfach den Computer einschalten. Hat das System Diskettenеinheiten, darf sich keine Diskette im Laufwerk A befinden oder die Diskettenverriegelung muß offen bleiben.

Die Worte "Version C" und die Releasenummer werden zusammen mit der Anzahl der freien Bytes angezeigt.

Disketten-BASIC starten:

1. DOS starten. Dazu wird:

   a. die IBM DOS-Diskette in das Laufwerk A eingelegt,

   b. der Computer eingeschaltet.

2. Den Befehl BASIC eingeben, wenn DOS die Eingabe eines Befehls erwartet.

   Die Worte "Version D" und die Releasenummer werden zusammen mit der Anzahl der freien Bytes angezeigt.

Erweitertes BASIC starten:

1. DOS starten wie oben beschrieben.

2. Als Antwort auf die DOS-Anfrage den Befehl BASICA eingeben.

   Die Worte "Version A" und die Releasenummer werden zusammen mit der Anzahl der freien Bytes angezeigt.

Auswahlen des Befehls BASIC

Zusätzliche Auswahlen können für die Befehle **BASIC** und **BASICA** angegeben werden, wenn das Disketten BASIC oder das erweiterte BASIC gestartet wird. Mit diesen Angaben wird die Hauptspeichergröße festgelegt, die BASIC für Programme und Daten und für Pufferbereiche verfügbar hat. Damit ist es auch möglich, dem BASIC mitzuteilen, daß ein Programm geladen und sofort ausgeführt werden soll.

Diese Auswahlen werden nicht benötigt - BASIC kann auch ohne sie arbeiten. Neulinge in BASIC können diesen Abschnitt übergehen und mit dem nächsten Abschnitt "Operationsmodus" weiterarbeiten. Wenn BASIC und seine Möglichkeiten bekannter sind, kann dieser Abschnitt durchgearbeitet werden.

Das vollständige Format des Befehls **BASIC** sieht wie folgt aus:

BASIC[A] [*dateiangabe*] [/F:*dateien*] [/S:*puffergröße*] [/C:*DFV-puffer*] [/M:*max.arbeitsbereich*]

dateiangabe ist die Dateiangabe für ein Programm, das geladen und sofort ausgeführt werden soll. Sie muß aus einer Zeichenkettenkonstante bestehen, die nicht in Anführungszeichen eingeschlossen werden darf und den Regeln für die Angabe von Dateien angepaßt sein muß, die in Kapitel 3 unter "Namengebung für Dateien" beschrieben sind. Eine Standarderweiterung .BAS wird benutzt, falls keine andere angegeben ist und die Länge des Dateinamens aus acht Zeichen oder weniger besteht. Wird *dateiangabe* angegeben, so arbeitet BASIC weiter, als ob der Befehl RUN*dateiangaben* als erstes eingegeben worden wäre, nachdem BASIC bereit war. Man sollte beachten, daß durch die Angabe der Auswahl *dateiangabe* die BASIC-Textzeile mit den Copyright-Angaben nicht angezeigt wird.

/F:*dateien* gibt die maximale Anzahl der Dateien an, die zu jeder Zeit während der Ausführung eines BASIC-Programms eröffnet werden können. Jede Datei benötigt 188 Bytes des Hauptspeichers für den Dateisteuerblock plus die Puffergröße, die mit der Auswahl /S: angegeben ist. Wird die Auswahl /F: weggelassen, ist die Standardannahme für Dateien 3. Der Maximalwert ist 15.

/S:*puffergröße* setzt die Puffergröße für die Benutzung mit Dateien für direkten Zugriff. Der Parameter für die Satzlänge in der Anweisung OPEN darf diesen Wert nicht übersteigen. Die Standardpuffergröße ist 128 Bytes. Der größte eingegebene Wert darf 32767 sein. Für den besten Durchsatz bei der Benutzung von Dateien mit direktem Zugriff wird /S:512 vorgeschlagen.

Mit /C:*DFV-puffer* wird die Größe jedes Datenfernverarbeitungspuffers zum Empfangen bei asynchroner Übertragung gesetzt. Die Angabe kommt nur zum Tragen, wenn das System den Datenfernverarbeitungsanschluß für asynchrone Übertrangung hat. Die maximale Größe darf 32767 sein. Wird die Angabe /**C:** weggelassen, werden jedem Puffer zum Empfangen 256 Bytes zugeordnet, zum Senden 128 Bytes. Wird mit einer hohen Übertragungsrate gearbeitet, wird **/C:1024** vorgeschlagen. So werden beide Empfangspuffer auf die in der Auswahl angegebene Größe gesetzt. Man kann die RS232-Unterstützung ausschalten, wenn man den Wert Null (**/C:0**) angibt. In diesem Fall wird für die Datenfernverarbeitung kein Pufferplatz reserviert und die DFV-Unterstützung nicht eingefügt, wenn BASIC geladen wird.

/M:*max.arbeitsbereich* setzt die maximale Anzahl der Bytes, die als BASIC-Arbeitsbereich benutzt werden dürfen. 64K ist der größte Wert, der gesetzt werden darf (hex FFFF). Man kann diese Auswahl benutzen, um Platz für Unterprogramme in Maschinensprache oder einen speziellen Datenspeicher zu reservieren. Gegebenenfalls sollte unter "Hauptspeicherbelegung" im Anhang J nachgelesen werden, wie BASIC den Speicherplatz benutzt. Wird die Auswahl /**M:** weggelassen, so wird der gesamte verfügbare Hauptspeicherbereich bis zu einem Maximum von 64K benutzt.

> **Hinweis:** *dateien, max.arbeitsbereich, puffergröße* und *DFV-puffer* bestehen aus Zahlen, die entweder dezimal, oktal (beginnend mit &O) oder hexadezimal (beginnend mit &H) dargestellt sein können.

Beispiele für den Befehl BASIC:

```
BASIC LOHN.BAS
```

Damit wird das Disketten-BASIC gestartet. Es benutzt standardmäßig, wie eben beschrieben, den gesamten Hauptspeicher und drei Dateien. Das Programm LOHN.BAS wird geladen und ausgeführt.

```
BASICA INVEN/F:6
```

Mit dieser Auswahl benutzt das erweiterte BASIC den gesamten Hauptspeicher und sechs Dateien, lädt und führt INVEN.BAS aus. Die Standarderweiterung ist .BAS.

```
BASIC /M:32768
```

Mit diesem Befehl wird der maximale Arbeitsbereich für das Disketten-BASIC auf 32768 gesetzt. Das heißt, BASIC benutzt nur 32K des Hauptspeichers. Es können zur gleichen Zeit drei Dateien benutzt werden.

```
BASIC B:PRUEF.TST/F:2/M:&H9000
```

Dieser Befehl setzt den maximalen Arbeitsbereich auf Hex 9000. Das bedeutet, das erweiterte BASIC kann bis zu 36K des Hauptspeichers benutzen. Es werden auch Dateisteuerblöcke für zwei Dateien gesetzt, und das Programm PRUEF.TST auf der Diskette in Laufwerk B wird geladen und ausgeführt.

Operationsmodus

Ist BASIC gestartet, so gibt es als Antwort **Ok** aus. **Ok** bedeutet, daß BASIC bereit ist, etwas zu tun. Dieser Status heißt *Befehlsebene.* Zu diesem Zeitpunkt kann BASIC in zwei Modi benutzt werden: dem *direkten* oder dem *indirekten Modus.*

Indirekter Modus

Im indirekten Modus werden Programme eingegeben. Um BASIC die einzugebende Zeile anzugeben, die Teil eines Programms ist beginnt jede Zeile mit einer *Zeilennummer*. Die Zeile wird dann als Teil eines Programms im Hauptspeicher gespeichert. Das im Hauptspeicher befindliche Programm kann mit dem Befehl RUN ausgeführt werden. Beispiel:

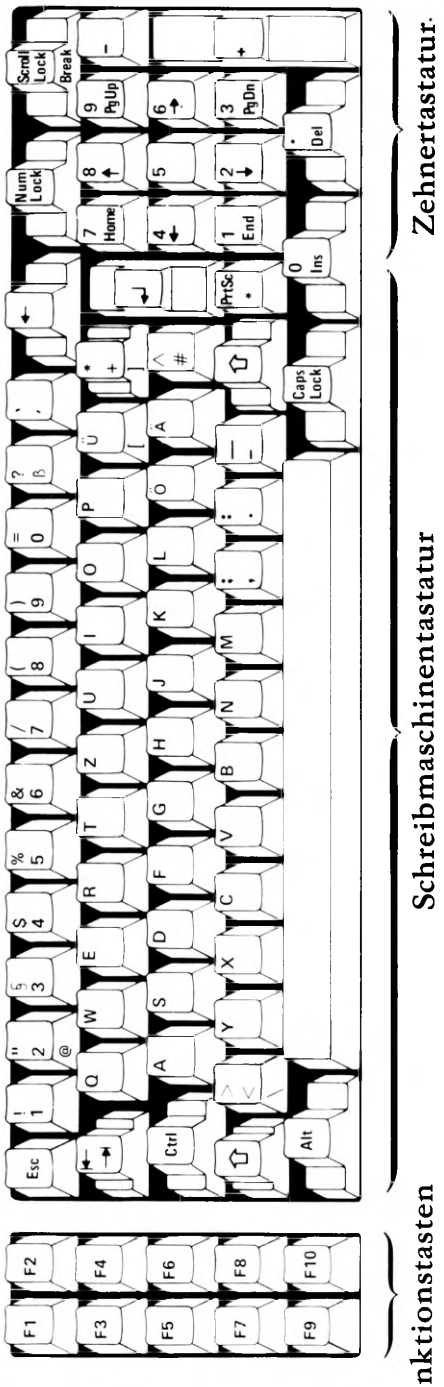
```
Ok
1 PRINT 2Ø+2
RUN
 22
Ok
```

Direkter Modus

Im direkten Modus soll BASIC eine Anfrage sofort nach der Eingabe ausführen. Dies wird BASIC dadurch gesagt, daß eine Anweisung oder ein Befehl *ohne* Zeilennummer eingegeben wird. Man kann Ergebnisse arithmetischer oder logischer Operationen sofort anzeigen oder sie für eine spätere Benutzung speichern. Die Instruktionen selbst werden, nachdem sie ausgeführt sind, nicht gespeichert. Dieser Modus ist für den Testlauf nützlich sowie für schnelle Berechnungen, die kein vollständiges Programm benötigen. Beispiel:

```
Ok
PRINT 2Ø+2
 22
Ok
```

Tastatur


Funktionstasten
Schreibmaschinentastatur
Zehnertastatur

Die Tastatur ist in drei Teile aufgeteilt:

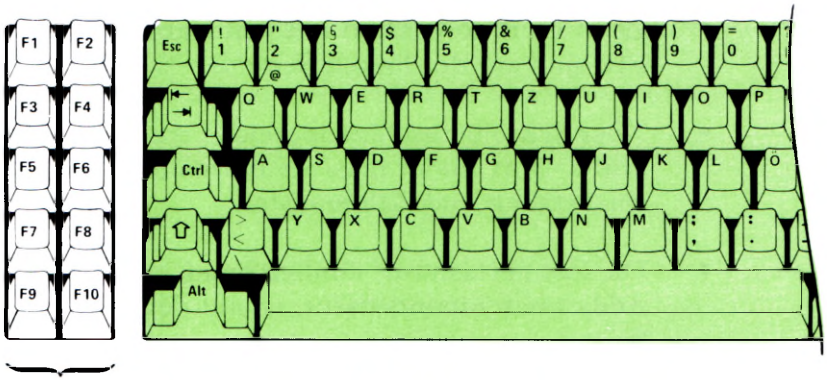
- Zehn Funktionstasten mit den Namen F1 bis F10 sind auf der linken Seite der Tastatur.

- "Der Schreibmaschinenbereich" ist in der Mitte. Dort findet man die normalen Buchstaben und Zahlentasten.

- Die Zehnertastatur, ähnlich einer Addiermaschinentastatur, ist auf der rechten Seite.

Alle Tasten in allen drei Tastaturbereichen sind Dauerfunktionstasten. Das heißt, sie wiederholen sich solange, wie sie gedrückt werden. Jeder Tastaturbereich wird anschließend genauer besprochen.

Funktionstasten



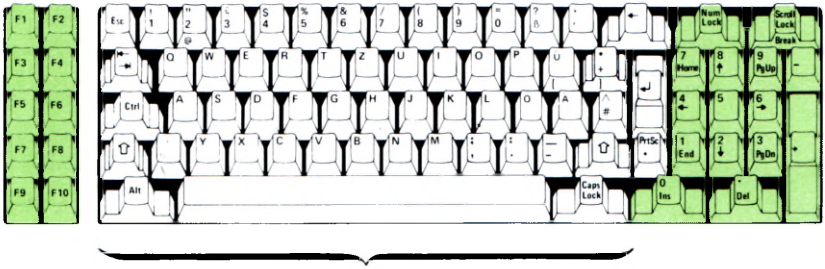
Funktionstasten

Benutzung der Funktionstasten:

- Man kann jeder Funktionstaste eine Zeichenfolge zuordnen, die beim Betätigen der Taste automatisch angezeigt wird. Häufig benutzte Befehle wurden diesen Tasten schon zugeordnet. Sie können jedoch abgeändert werden. Einzelheiten siehe Kapitel 4 unter "Anweisung KEY".

- Als Programmunterbrechungen im erweiterten BASIC mit Hilfe der Anweisung ON KEY. Siehe "Anweisung ON KEY(n)" in Kapitel 4.

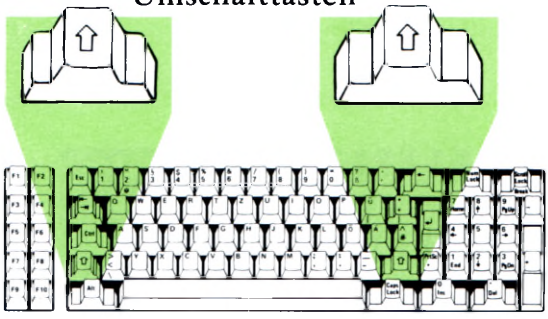
Schreibmaschinentastatur



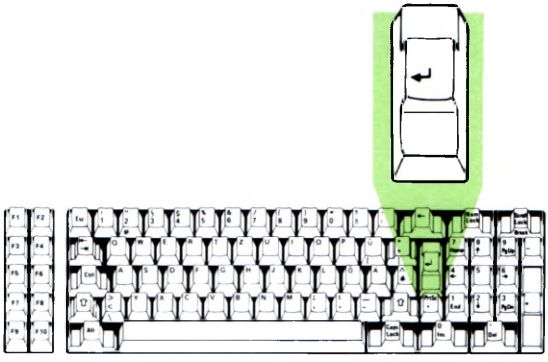
Schreibmaschinentastatur

Der Schreibmaschinenbereich der Tastatur funktioniert fast wie eine Standardschreibmaschine. Alle Buchstaben sind dort zu finden. Die Zahlen 1 bis 9 und 0 findet man in der obersten Zeile zusammen mit einigen Sonderzeichen.

Umschalttasten



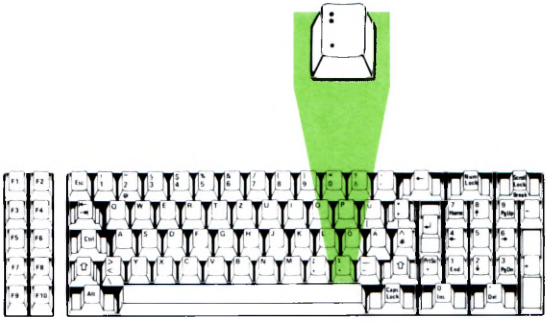
Großbuchstaben und die Sonderzeichen über den Zahlen der numerischen Tasten werden angezeigt, indem man die Umschalttasten betätigt und niederhält und die gewünschte Taste betätigt.



Die Taste mit dem Symbol ↵ ist die Schreibkopfrücklauftaste. Sie wird gewöhnlich betätigt, um eine Information in dem Computer zu speichern. Sie wird von jetzt an als *Eingabetaste* bezeichnet.

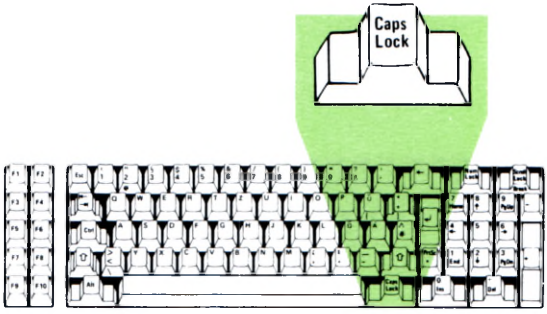
Es gibt mehrere wichtige Unterschiede zwischen dieser Tastatur und einer normalen Schreibmaschine.

Sonderzeichen



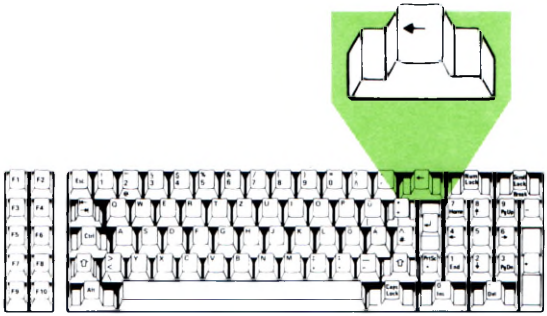
Die Tastatur enthält einige Sonderzeichen, die auf einer normalen Schreibmaschine nicht zu finden sind, wie z.B. ^, [,und]. Einige Zeichen stehen an einem anderen Platz als man sie auf einer normalen Schreibmaschine erwartet. So erhält man z.B., wenn man die Umschalttaste zusammen mit der Punkttaste (.) drückt, keinen Punkt, sondern einen Doppelpunkt (:).

Großbuchstaben



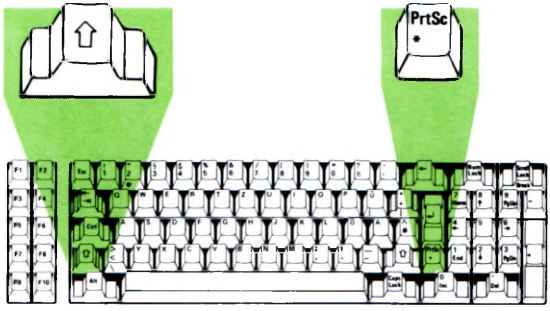
Die Tastatur enthält keinen Umschaltfeststeller. Die Taste Caps Lock rechts neben der Leertaste arbeitet ähnlich dem Umschaltfeststeller, erzeugt aber nur große Buchstaben und nicht die Zeichen über den numerischen oder anderen Tasten, die man mit der Umschalttaste erhält. Nachdem die Taste Caps Lock betätigt wurde, erhält man so lange Großbuchstaben, bis man sie erneut betätigt. Man kann im Großschreibungsstatus auch Kleinbuchstaben erhalten, indem man eine der Umschalttasten betätigt und niederhält. Läßt man die Umschalttaste los, kommt man in den Großschreibungsstatus zurück.

Rücktaste



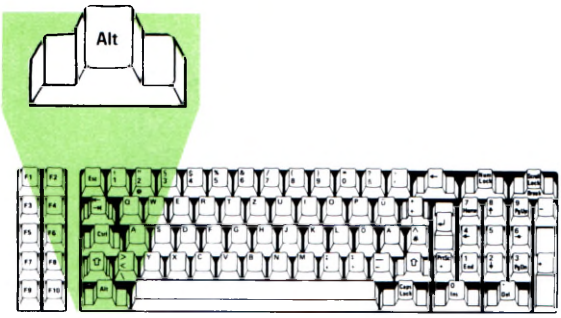
Die Rücktaste arbeitet ein bißchen anders als die Rücktaste einer Schreibmaschine. Sie führt nicht nur den Positionsanzeiger zurück, sondern löscht auch was eingegeben wurde. Man muß die Taste "Positionsanzeiger nach links" verwenden, falls eine vorherige Eingabe nicht gelöscht werden soll. Siehe unter "BASIC Korrekturprogramm" später in diesem Kapitel.

PrtSc



Unter der Eingabetaste befindet sich eine Taste, die mit PrtSc oben und einem * unten beschriftet ist. PrtSc steht für "Drucken Bildschirm". Steht die Tastatur auf Kleinbuchstaben, so wird durch Betätigen dieser Taste ein Stern ausgegeben. Steht die Tastatur jedoch auf Großbuchstaben, arbeitet die Taste wie eine Sondertaste und ergibt eine Kopie des Bildschirms auf dem Drucker (LPT1:). Soll also eine Kopie des gerade angezeigten Bildschirms erstellt werden, muß nur eine der Umschalttasten betätigt und gehalten und die Taste PrtSc betätigt werden.(Hinweis: Zeichen, die der Drucker nicht versteht, werden als Leerstellen ausgegeben.)

Andere Umschaltungen: Zusätzlich 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 betätigt und hält die Taste Alt (oder Ctrl) und betätigt die gewünschte Taste. Dann läßt man beide Tasten wieder los. Jedoch werden Alt and Ctrl für verschiedene Funktionen benutzt.

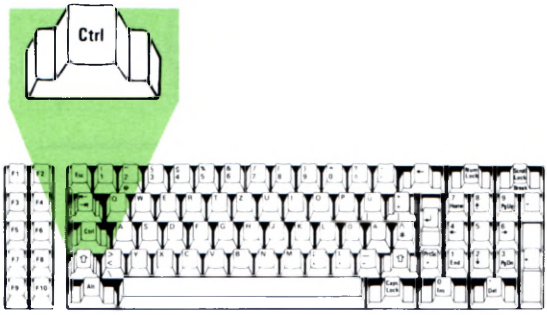


Mit der Taste Alt kann man BASIC-Anweisungsschlüsselwörter eingeben. Damit ist es möglich, mit einem einzigen Tastendruck ein ganzes BASIC-Schlüsselwort einzugeben.

Das BASIC-Schlüsselwort wird eingegeben, wenn man die Taste Alt niederhält und eine der alphabetischen Tasten A bis Z betätigt. Die jedem Buchstaben zugeordneten Schlüsselwörter sind unten aufgeführt. Buchstaben, denen kein Schlüsselwort 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	MOTOR	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 niederhält und den dreistelligen ASCII-Code für das Zeichen eintastet. (Siehe "Anhang G. ASCII-Zeichencodes". Er enthält eine vollständige Liste der ASCII-Codes).



Mit der Taste Ctrl ist es auch möglich, gewisse Codes und Zeichen einzugeben, die sonst nicht auf der Tastatur verfügbar sind. Zum Beispiel ist **Ctrl-G** das *Alarmzeichen.* Wird diese Taste betätigt, kommt über den Lautsprecher ein Alarmzeichen. Ctrl-G bedeutet, daß die Taste Ctrl niedergehalten wird und dann die Taste G betätigt wird. Danach können beide Tasten losgelassen werden.

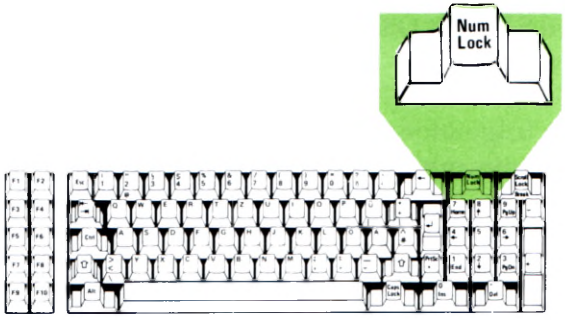
Die Taste Ctrl wird auch in Verbindung mit anderen Tasten benutzt, wenn mit dem Korrekturprogramm Programme verändert werden.

Zehnertastatur

Zehnertastatur

Normalerweise benutzt man die Zehnertastatur mit ihren Funktionen in Verbindung mit dem Korrekturprogramm. Diese Tasten ermöglichen es, den Positionsanzeiger nach oben, nach unten, nach rechts und nach links zu bewegen. Mit Hilfe dieser Tasten ist es möglich, Zeichen einzufügen und zu löschen. Eine vollständige Information findet man im folgenden Abschnitt "BASIC-Korrekturprogramm".

Hinweis: Die Tasten Scroll Lock, Pg Up und Pg Dn werden von BASIC nicht benutzt, sie können aber in einem Programm eine Bedeutung bekommen.

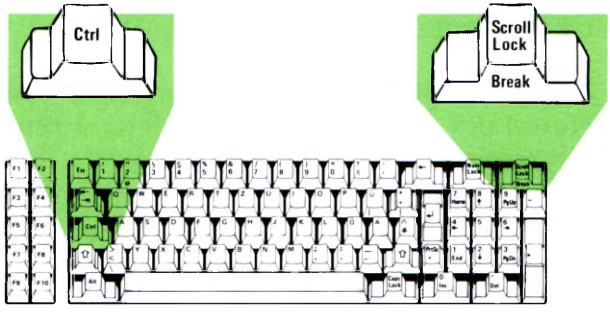


Zehnertastatur Umschaltung: Mit Hilfe der Taste Num Lock ist es möglich, die Zehnertastatur so zu setzen, daß sie wie eine Addiermaschinentastatur arbeitet. Wird die Taste Num Lock gedrückt, wird die Zehnertastatur in ihren eigenen Großbuchstabenmodus umgeschaltet, so daß man die Zahlen 0 bis 9 und den Dezimalpunkt so bekommt, wie sie auf den Tasten oben angezeigt sind. Wird die Taste Num Lock noch einmal gedrückt, wird die Tastatur wieder in ihren normalen Steuermodus umgeschaltet. Wie bei normalem Modus kann der numerische Status temporär umgeschaltet werden, indem man eine der Umschalttasten betätigt.

Sondertastenkombinationen

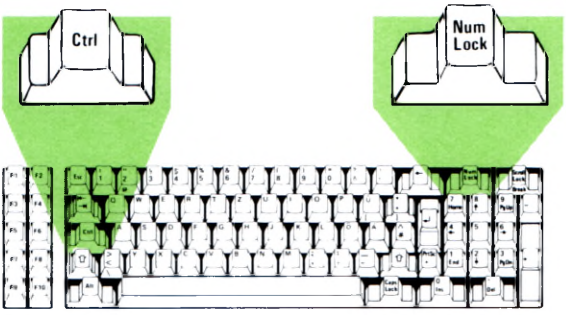
Die folgenden Spezialfunktionen von Tastenkombinationen sind möglich:

Ctrl-Break



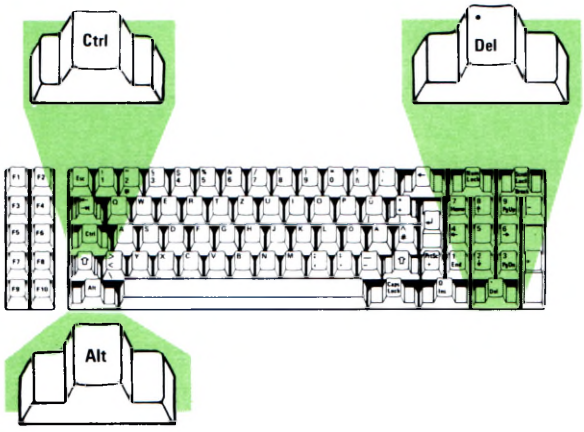
Ctrl-Break unterbricht die Programmausführung an der nächsten Programminstruktion und kehrt auf die BASIC-Befehlsebene zurück. Mit Ctrl-Break verläßt man auch den AUTO-Zeilennumerierungsmodus.

Ctrl-Num Lock



Ctrl-Num Lock versetzt den Computer in den *Pausen*-Status. Damit kann temporär das Drucken oder Programmauflisten angehalten werden. Die Pause wird aufgehoben, sobald irgendeine Taste außer Umschalttasten, die Taste Break oder die Taste Ins (einfügen) betätigt wird. (Siehe "Großbuchstaben", "Andere Umschaltungen" und "Zehnertastaturumschaltung" in diesem Abschnitt.)

Alt-Ctrl-Del



Alt-Ctrl-Del führt einen *Systemwiederanlauf* aus, in anderen Worten, es ist, als würde der Computer neu eingeschaltet. Die Tasten Ctrl und Alt müssen zusammen und jetzt die Taste Del betätigt werden. Danach können alle drei Tasten wieder freigegeben werden. Einen Systemwiederanlauf mit diesen Tasten auszuführen geht schneller, als den Strom aus- und wieder einzuschalten.

BASIC-Korrekturprogramm

Jede Textzeile, die eingegeben wurde, als sich BASIC in der Befehlsebene befand, wurde vom BASIC-Korrekturprogramm verarbeitet.

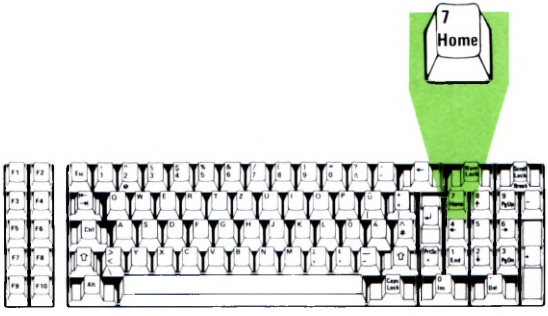
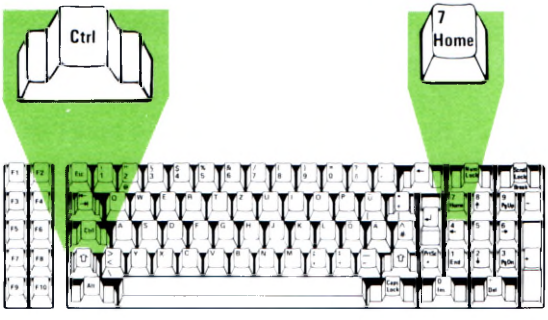
Das BASIC-Korrekturprogramm ist ein "Aufbereitungsprogramm für Bildschirmzeilen". Das bedeutet, jede Zeile irgendwo auf dem Bildschirm kann geändert werden, aber nur eine Zeile zu einer Zeit. Die Änderung wird erst ausgeführt, wenn die Eingabetaste auf dieser Zeile betätigt wird.

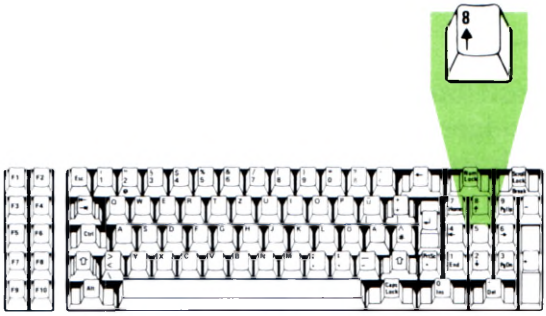
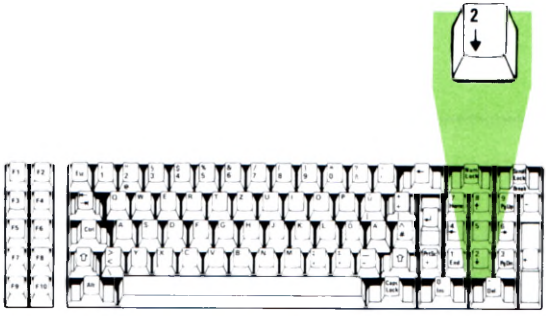
Die Benutzung des Korrekturprogramms kann während der Programmentwicklung sehr viel Zeit sparen. Um die Einrichtungen kennenzulernen, ist es das beste, ein Beispielprogramm einzugeben und alle Korrekturmöglichkeiten auszuprobieren. Der beste Weg ist, wenige Zeilen zu ändern, während die folgenden Informationen gelesen werden.

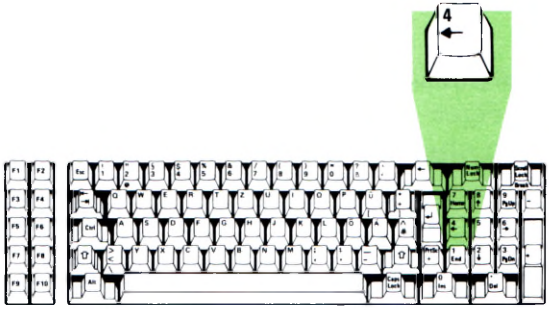
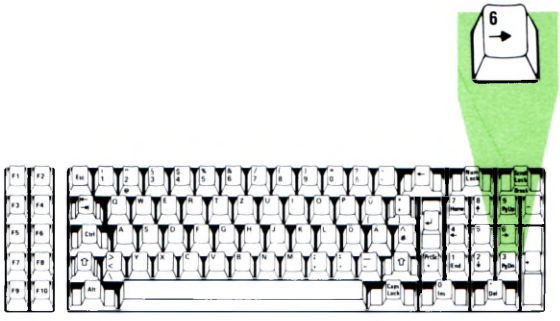
Wenn etwas in den Computer eingegeben wird, sieht man einen blinkenden Strich oder ein Viereck rechts neben dem letzten eingegebenen Zeichen. Dieses Zeichen oder Viereck nennt man den *Positionsanzeiger.* Er zeigt die nächste Position an, an der ein Zeichen eingegeben, eingefügt oder gelöscht werden kann.

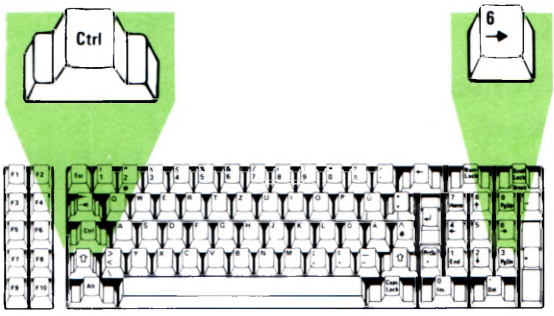
Spezielle Programmkorrekturtasten

Mit Hilfe der Tasten der Zehnertastatur, der Rücktaste und der Taste Ctrl ist es möglich, den Positionsanzeiger an einen Platz auf dem Bildschirm zu bringen, Zeichen einzugeben oder Zeichen zu löschen. Die Tasten und ihre Funktionen sind:

Taste(n)	Funktion
Home	 Der Positionsanzeiger wird in die obere linke Ecke des Bildschirms gebracht.
Ctrl-Home	 Der Bildschirm wird gelöscht und der Positionsanzeiger in die obere linke Ecke des Bildschirms gebracht.

Taste(n)	Funktion
↑ **Positions-anzeiger nach oben**	 Bewegt den Positionsanzeiger eine Stelle nach oben.
↓ **Positions-anzeiger nach unten**	 Bewegt den Positionsanzeiger eine Stelle nach unten.

Taste(n)	Funktion
← (Positions-anzeiger nach links)	 Bewegt den Positionsanzeiger eine Stelle nach links. Bewegt sich der Positionsanzeiger über die linke Seite des Bildschirms hinaus, so springt er auf die rechte Seite des Bildschirms zur vorhergehenden Zeile.
→ (Positions-anzeiger nach rechts)	 Bewegt den Positonsanzeiger eine Stelle nach rechts. Geht der Positionsanzeiger über die rechte Seite des Bildschirms hinaus, so springt er zur linken Seite des Bildschirms auf die nächste Zeile.

Taste(n)	Funktion
Ctrl- → (Nächstes Wort)	

Bewegt den Positionsanzeiger nach rechts zum nächsten *Wort*. Ein Wort ist definiert als Zeichen oder Zeichengruppe, die mit einem Buchstaben oder einer Zahl beginnt. Wörter werden durch Leerstellen oder Sonderzeichen getrennt. So ist das nächste Wort, der nächste Buchstabe oder die nächste Zahl rechts des Positionsanzeigers, der oder die einem Leerzeichen oder einem Sonderzeichen folgt.

Nehmen wir z.B. die folgende Zeile:

```
LINE (L1,LOW2)-(MAX,48)  ,3 , BF
```

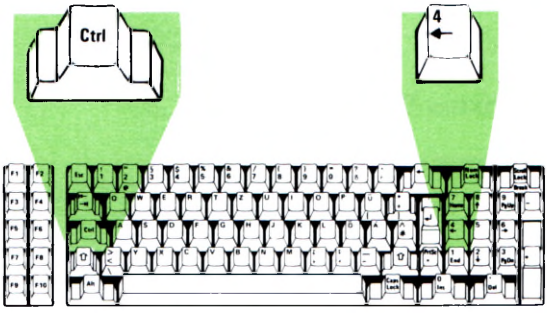
Der Positionsanzeiger steht jetzt in der Mitte des Wortes L0W2. Werden die Tasten (Ctrl-Positionsanzeiger nach rechts) betätigt, bewegt sich der Positionsanzeiger zum Beginn des nächsten Wortes, nämlich MAX:

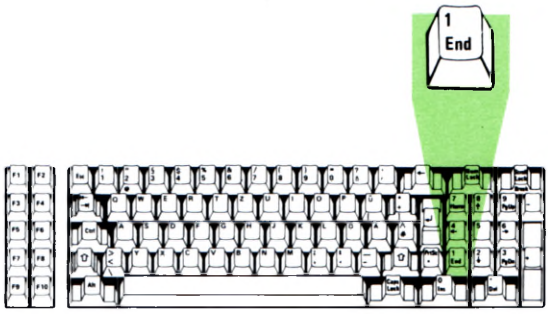
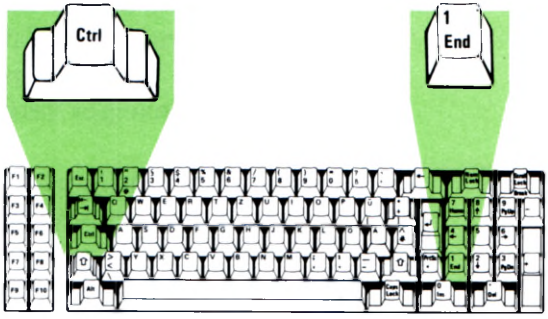
```
LINE (L1,LOW2)-(MAX,48)  ,3 , BF
```

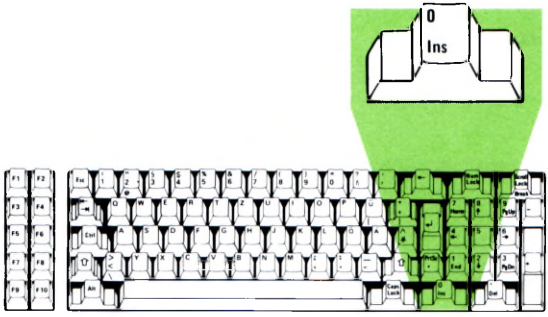
Werden dieses Tasten erneut betätigt, bewegt sich der Positionsanzeiger zu der Zahl 48:

```
LINE (L1,LOW2)-(MAX,48)  ,3 , BF
```

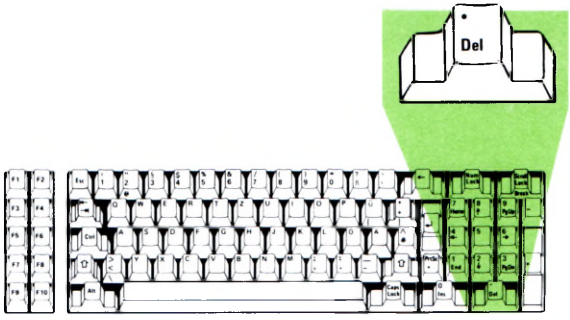
Taste(n)	Funktion

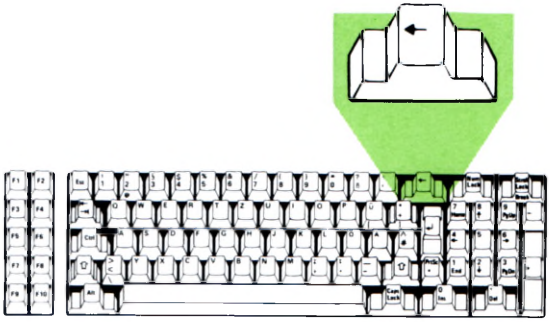
Ctrl- ← (Vorheriges Wort)	
	Bewegt den Positionsanzeiger nach links zum vorherigen Wort. Das vorherige Wort ist der Buchstabe oder die Zahl links des Positionsanzeigers, vor dem oder der ein Leerzeichen oder ein Sonderzeichen steht.
	
	Nehmen wir z.B. die folgende Zeile:
	`LINE (L1,LOW2)-(MAX,48) ,3 , BF_`
	
	Betätigt man die Tasten (Ctrl-Positionsanzeiger nach links), bewegt sich der Positionsanzeiger an den Anfang des Wortes BF:
	`LINE (L1,LOW2)-(MAX,48) ,3 , BF`
	
	Betätigt man diese Tasten erneut, bewegt sich der Positionsanzeiger zum vorherigen Wort, diesmal zur Zahl 3:
	`LINE (L1,LOW2)-(MAX,48) ,3 , BF`
	
	Betätigt man diese Tasten noch zweimal, bewegt sich der Positionsanzeiger zuerst zur Zahl 48, danach zum Wort MAX:
	`LINE (L1,LOW2)-(MAX,48) ,3 , BF`

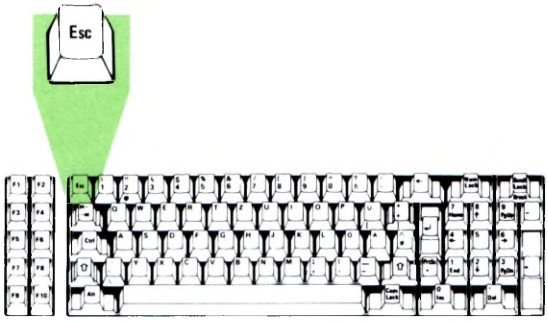
Taste(n)	Funktion
End	 Bewegt den Positionsanzeiger zum Ende einer logischen Zeile. Zeichen, die von dieser Stelle an eingegeben werden, werden an das Ende der Zeile angefügt.
Ctrl-End	 Löscht ab der Position des Positionsanzeigers bis zum Ende der logischen Zeile. Alle physischen Bildschirmzeilen bis zum beendenden Eingabezeichen werden gelöscht.

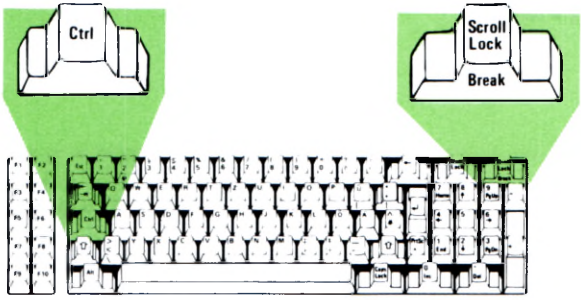
Taste(n)	Funktion
Ins	 Setzt den Einfügmodus. Ist der Einfügmodus ausgeschaltet, wird er eingeschaltet, sobald diese Taste gedrückt wird. Ist der Einfügmodus schon eingeschaltet, wird er ausgeschaltet, sobald diese Taste gedrückt wird. Der Einfügmodus wird dadurch angezeigt, daß die untere Hälfte der Zeichenposition vom Positionsanzeiger verdeckt wird. Steht der Einfügmodus auf ein, werden die Zeichen über dem Positionsanzeiger und die folgenden nach rechts verschoben, sobald Zeichen an der Stelle des Positionsanzeigers eingegeben werden. Nach jedem Tastendruck wird der Text um eine Stelle nach rechts verschoben. Ein Zeilenüberlauf wird beachtet. Das bedeutet, sobald ein Zeichen rechts aus dem Bildschirm verschwindet, erscheint es links wieder auf der nachfolgenden Zeile.

Taste(n)	Funktion
Ins	Steht der Einfügmodus auf aus, so werden existierende Zeichen einer Zeile durch neu eingegebene ersetzt. Der Einfügmodus wird nicht nur durch das erneute Drücken der Taste Ins ausgeschaltet, sondern auch durch das Drücken der Tasten, die den Positionsanzeiger bewegen, oder die Eingabetaste.

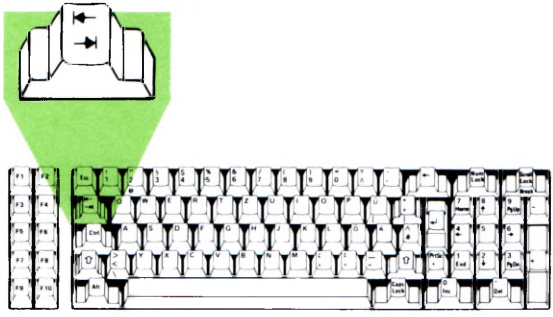
Taste(n)	Funktion
Del	 Löscht das Zeichen an der momentanen Stelle des Positionsanzeigers. Danach werden alle Zeichen rechts des gelöschten um eine Position nach links verschoben, um den leeren Raum auszufüllen. Zeilenüberlauf wird beachtet. Das heißt, sobald eine logische Zeile größer ist als eine physische Zeile, werden auch die Zeichen der folgenden Zeilen um eine Position nach links verschoben, um den vorherigen Platz auszufüllen, und die Zeichen der ersten Spalte aller nachfolgenden Zeilen werden an das Ende der vorherigen Zeile verschoben.

Taste(n)	Funktion
← (Rückschritt)	 Dadurch wird das letzte eingegebene Zeichen gelöscht. Das heißt, das Zeichen links des Positionsanzeigers wird gelöscht. Alle Zeichen rechts des gelöschten Zeichens werden um eine Stelle nach links verschoben, um den Leerraum aufzufüllen. Nachfolgende Zeichen und Zeilen innerhalb einer logischen Zeile werden bei der Taste Del nach rechts aufgefüllt.

Taste(n)	Funktion
Esc	 Wird diese Taste irgendwo in einer Zeile gedrückt, wird der ganze logische Satz vom Bildschirm gelöscht. Die Zeile wird nicht an BASIC zur Verarbeitung übergeben. Handelt es sich um eine Programmzeile, wird die Zeile des Programms nicht im Hauptspeicher gelöscht.

Taste(n)	Funktion
Ctrl-Break	 Es wird zur Befehlsebene zurückverzweigt, *ohne* daß Änderungen, die in der laufenden Zeile vorgenommen wurden, gespeichert werden. Die Zeile wird nicht wie bei Esc vom Bildschirm gelöscht.

Taste(n)	Funktion

(Tab)	
	Bringt den Positionsanzeiger zum nächsten Tabulatorstopp. Tabulatorstopps stehen nach jeweils acht Zeichenpositionen, d.h. an Stelle 1, 9, 17 usw.
	
	Ist der Einfügmodus ausgeschaltet, wird beim Betätigen der Tabulatortaste der Positionsanzeiger über alle Zeichen hinweg bis zum nächsten Tabulatorstopp geführt.
	
	Beispiel:
	
	`10 REM Bemerkung`
	
	Wird die Tabulatortaste betätigt, geht der Positionsanzeiger an die Stelle 9.
	
	`10 REM Bemerkung`
	
	Wird die Taste erneut betätigt, geht der Positionsanzeiger an die Stelle 17.
	
	`10 REM Bemerkung_`

Taste(n)	Funktion

(Tab)	(Fortsetzung)
	
	Ist der Einfügmodus eingeschaltet und wird die Tabulatortaste betätigt, werden ab dem Positionsanzeiger bis zum nächsten Tabulatorstopp Leerstellen eingefügt. Zeilenüberlauf wird wie bei **Ins** ausgeführt.
	
	Beispiel:
	
	`10 REM Bemerkung`
	
	Wird zuerst die Taste Ins (einfügen) und danach die Tabulatortaste betätigt, werden bis zur Stelle 17 Leerzeichen eingefügt.
	
	`10 REM Be          merkung`

Korrektur der laufenden Eingabezeile

Während sich BASIC in der Befehlsebene befindet, wird jede eingegebene Textzeile vom Korrekturprogramm verarbeitet. Dabei können alle im vorherigen Abschnitt unter "Spezielle Programmkorrekturtasten" beschriebenen Tasten benutzt werden. BASIC befindet sich immer 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 möglich, eine logische Zeile länger als eine physische Bildschirmzeile zu machen, indem man einfach über das Ende der Bildschirmzeile eingibt. Der Positionsanzeiger bewegt sich automatisch in die nächste Zeile. Dies geht auch mit einem Zeilenvorschub (Ctrl-Eingabetaste). Wird ein Zeilenvorschub eingegeben, so wird der nachfolgende Text auf die nächste Bildschirmzeile geschrieben, ohne daß Leerstellen eingegeben werden müssen, um den Positionsanzeiger dorthin zu bringen. Die Zeile wird nicht verarbeitet. Dies geschieht erst, wenn die Eingabetaste gedrückt wird.

Dabei muß beachtet werden, daß durch den Zeilenvorschub der Rest der physischen Bildschirmzeile mit Leerstellen aufgefüllt wird. Das Zeichen für Zeilenvorschub wird an den Text nicht angefügt. Diese Leerstellen sind in den 255 Zeichen enthalten, die für eine BASIC-Zeile erlaubt sind.

Wird schließlich die Eingabetaste gedrückt, so wird die gesamte logische Zeile für die Verarbeitung an BASIC weitergegeben.

Zeichen ändern: Wird eine Zeile eingegeben und dabei entdeckt, daß etwas falsch eingegeben wurde, kann dies verbessert werden. Mit der Taste Positionsanzeiger links oder einer anderen Taste, die den Positionsanzeiger bewegt, wird dieser an die Position gebracht, an der der Fehler auftrat, und der korrekte Buchstabe über den falschen geschrieben. Jetzt kann der Positionsanzeiger mit Hilfe der Tasten Positionsanzeiger 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 für Vorheriges Wort (Ctrl-Positionsanzeiger links) werden zweimal betätigt, bis sich der Positionsanzeiger unter V befindet:

```
LOAD "V;PROG
```

Dann wird B: eingegeben:

```
LOAD"B:PROG
```

Danach wird die Taste End gedrückt:

```
LOAD "B:PROG_
```

Der Fehler ist behoben und die Eingabe kann fortgesetzt werden:

```
LOAD "B:PROGRAM1"_
```

Zeichen löschen: Wird bei der Eingabe ein Zeichen bemerkt, das nicht in der Zeile stehen soll, kann man es mit Hilfe der Taste Del löschen. Mit Hilfe der Taste Positionsanzeiger nach links oder einer anderen Taste für Positionsanzeigerbewegungen muß der Positionsanzeiger an das Zeichen bewegt werden, das gelöscht werden soll. Betätigt man jetzt die Taste Del, so wird es gelöscht. Danach wird mit der Taste Positionsanzeiger nach rechts oder End der Positionsanzeiger zurück an das Ende der Zeile gebracht und die Eingabe fortgesetzt.

Beispiel:

```
DEELETE_
```

Um das überzählige E zu löschen, muß der Positionsanzeiger mit der Taste Positionsanzeiger nach links unter das überzählige E gestellt werden:

```
DEELETE
```

Jetzt betätigt man die Taste Del:

```
DELETE
```

Jetzt muß die Taste End betätigt werden:

```
DELETE_
```

und die Eingabe fortgesetzt werden:

```
DELETE 20_
```

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

Beispiel:

```
DELETT_
```

Jetzt die Rücktaste betätigten:

```
DELET_
```

Jetzt kann die Eingabe fortgesetzt werden:

```
DELETE 2Ø_
```

Zeichen hinzufügen: Falls man bemerkt, daß man Zeichen in einer Zeile, die gerade eingegeben wird, vergessen hat, bringt man den Positionsanzeiger an die Stelle, an der die neuen Zeichen eingegeben werden sollen. Danach muß die Taste Ins gedrückt werden, um in den Einfügmodus zu kommen. Jetzt gibt man die Zeichen ein, die hinzugefügt werden sollen. Die Zeichen, die man eingibt, werden an der Stelle des Positionsanzeigers eingefügt, und die Zeichen über und neben ihm werden nach rechts verschoben. Wenn man wie zuvor die Eingabe fortsetzen möchte, bringt man den Positionsanzeiger mit Hilfe der Tasten Positionsanzeiger nach rechts oder End an das Ende der Zeile und setzt die Eingabe fort. Der Einfügmodus wird automatisch ausgeschaltet, wenn eine dieser Tasten benutzt wird.

Beispiel:

```
LIS 1Ø_
```

Das T in **LIST** wurde vergessen. So wird mit der Taste Positionsanzeiger nach links der Positionsanzeiger unter die Leerstelle gebracht.

```
LIS_10
```

Jetzt die Taste Ins betätigten und den Buchstaben **T** eingeben:

```
LIST_10
```

Löschen eines Teils einer Zeile: Soll eine Zeile ab der Stelle des Positionsanzeigers gelöscht werden, betätigt man die Tasten Ctrl-End.

Beispiel:

```
1Ø REM *** Mist Mist Mist
```

Der Positionsanzeiger wurde unter das **M** des ersten Wortes **Mist** gesetzt. Zum Löschen müssen nur noch die Tasten Ctrl-End betätigt werden:

```
1Ø REM *** _
```

Löschen einer Zeile: Soll eine Zeile, die gerade eingegeben wird, gelöscht werden, betätigt man die Taste Esc irgendwo auf der Zeile. Man muß danach nicht die Eingabetaste betätigen. Dadurch wird die gesamte logische Zeile gelöscht.

Beispiel:

```
DIESE ZEILE HAT KEINE BEDEUTUNG_
```

Auch wenn sich der Positionsanzeiger am Ende der Zeile befindet, wird die gesamte Zeile durch die Taste Esc gelöscht.

Eingeben oder Ändern 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 Eingabetaste und kann maximal 255 Zeichen einschließlich des Zeichens für die Eingabetaste enthalten. Enthält eine Zeile mehr als 255 Zeichen, werden die überzähligen Zeichen abgeschnitten, sobald die Eingabetaste betätigt wird. Obwohl die überzähligen Zeichen noch auf dem Bildschirm stehen, werden sie von BASIC nicht verarbeitet.

BASIC-Schlüsselwörter und Variablennamen müssen in Großbuchstaben sein. Jedoch kann man sie in jeder Kombination von Groß- und Kleinbuchstaben eingeben. Das Korrekturprogramm wandelt alles außer Bemerkungen, DATA-Anweisungen und Zeichenketten, die in Anführungszeichen eingeschlossen sind, in Großbuchstaben um.

Manchmal ändert BASIC eine Eingabe etwas ab. Nimmt man z.B. das Fragezeichen (?) anstatt des Wortes PRINT in einer Programmzeile, wird, wenn die Zeile mit LIST aufgelistet wird, das Fragezeichen in PRINT mit einer Leerstelle danach abgeändert, da das Fragezeichen eine Abkürzung für die Eingabe von PRINT ist. Durch diese Erweiterung kann das Ende einer Zeile abgeschnitten werden, falls die Zeilenlänge nahe bei 255 Zeichen liegt.

Warnung:
Erreicht die Zeile die Maximallänge, muß das 255. Zeichen die Eingabetaste sein.

Hinzufügen einer neuen Zeile in ein Programm:
Eingeben einer gültigen Zeilennummer (im Bereich 0 bis 65529), gefolgt von der Eingabetaste. 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 gültige BASIC-Anweisung. Es wird beim Eingeben der Zeile kein Fehler angezeigt. Programme werden *nicht* auf richtige Syntax geprüft, bevor sie dem Programm hinzugefügt werden. Dies geschieht erst, wenn die Programmzeile ausgeführt wird.

Ist schon eine Zeile mit der gleichen Zeilennummer vorhanden, wird sie gelöscht und durch die neue ersetzt.

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

Ersetzen oder Ändern einer existierenden Programmzeile: Eine schon vorhandene Zeile wird wie oben beschrieben verändert, falls die eingegebene Zeilennummer schon im Programm vorhanden ist. Die alte Zeile wird durch den Text der neuen ersetzt.

Beispiel:

```
10 Dies ist eine neue Zeile 10
```

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

Löschen einer Programmzeile: Zum Löschen einer Programmzeile die Zeilennummer eingeben und die Eingabetaste betätigen.

Beispiel:

```
10
```

Dadurch würde die Zeile 10 im Programm gelöscht.

Man kann auch eine Zeilengruppe im Programm mit Hilfe des Befehls DELETE löschen. Der Befehl wird in Kapitel 4 unter "Befehl DELETE" beschrieben.

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

Zum Löschen von Pogrammzeilen darf nicht die Taste Esc verwendet werden. Durch Esc wird die Zeile nur auf dem Bildschirm gelöscht. Befindet sich die Zeile in einem BASIC-Programm, bleibt sie dort erhalten.

Löschen eines ganzen Programms: Soll ein Programm ganz gelöscht werden, das sich im Hauptspeicher befindet, muß der Befehl NEW eingegeben werden (siehe "Befehl NEW" in Kapitel 4). Normalerweise wird NEW benutzt, um den Hauptspeicher zu löschen, bevor ein neues Programm eingegeben wird.

Ändern von Zeilen auf dem gesamten Bildschirm

Man kann jede Zeile auf dem Bildschirm verändern, indem man die Tasten für das Bewegen des Positionsanzeigers benutzt (beschrieben unter "Spezielle Programmkorrekturtasten"). Danach können eine oder alle der vorher beschriebenen Techniken benutzt werden, um Zeilen zu veränden, zu löschen oder Zeichen hinzuzufügen.

Sollen Programmzeilen verändert werden, die nicht angezeigt werden, kann man diese mit Hilfe des Befehls LIST anzeigen. Man listet sich die Zeile oder den Bereich der Zeilen, die verändert werden sollen, auf (siehe "Befehl LIST" in Kapitel 4). Danach bringt man den Positionsanzeiger auf die Zeile, die korrigiert werden soll und verändert die Zeile mit den schon beschriebenen Techniken. Mit Hilfe der Eingabetaste wird die veränderte Zeile im Programm gespeichert. Auch mit Hilfe des Befehls EDIT kann man die gewünschte Zeile anzeigen. Siehe "Befehl EDIT" in Kapitel 4.

Eine Programmzeile kann z.B. auf folgende Art dupliziert werden: Man bringt den Positionsanzeiger auf die Zeile, die dupliziert werden soll. Danach verändert man die Zeilennummer auf eine neue Zeilennummer, indem man einfach über die alte Zahl die neue Zahl eingibt. Wird danach die Eingabetaste betätigt, sind die alte und die neue Zeile im Programm vorhanden.

Man kann auch die Zeilennummer einer Programmzeile ändern, indem man die Zeile wie zuvor beschrieben dupliziert und danach die alte Zeile löscht.

Die Programmzeile eines BASIC-Programms wird immer erst geändert, wenn die Eingabetaste betätigt wird. Müssen mehrer Zeilen verändert werden, kann es leichter sein, mit dem Positionsanzeiger auf dem Bildschirm an all die Stellen zu gehen, an denen Korrekturen ausgeführt werden sollen, und dann zur ersten veränderten Zeile zurückzukehren und die Eingabetaste am Beginn jeder Zeile betätigen. Dadurch wird jede veränderte Zeile im Programm gespeichert.

Es ist nicht nötig, den Positionsanzeiger ans Ende der logischen Zeile zu bringen, bevor die Eingabetaste betätigt wird. Das Korrekturprogramm weiß, wo jede logische Zeile endet und verarbeitet die gesamte Zeile, auch wenn die Eingabetaste am Beginn einer Zeile betätigt wird.

> **Hinweis:** Der Befehl AUTO kann sehr hilfreich sein, wenn ein Programm eingegeben wird. Der AUTO-Modus muß aber durch Betätigen der Tasten Ctrl-Break verlassen werden, bevor irgendeine Zeile außer der laufenden Eingabezeile geändert wird.

Dabei muß daran erinnert werden, daß Veränderungen mit Hilfe dieser Techniken nur das Programm im Hauptspeicher verändern. Soll das Programm mit den neuen Veränderungen gespeichert werden, kann man dies mit Hilfe des Befehls SAVE tun (siehe "Befehl SAVE" in Kapitel 4), bevor man Befehl NEW eingibt oder BASIC verläßt.

Syntaxfehler

Wird während der Programmausführung ein Syntaxfehler entdeckt, zeigt BASIC automatisch die Zeile an, die den Fehler verursachte, so daß man ihn verbessern 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 Positionsanzeiger unter die Ziffer 1 gestellt.

Man kann den Positionsanzeiger nach rechts unter das Dollarzeichen ($) bringen, es auf ein Pluszeichen (+) ändern und die Eingabetaste betätigen. Dadurch wird die verbesserte Zeile in das Programm zurückgespeichert.

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

- Alle Variablen und Bereiche werden gelöscht. Das heißt, sie werden zurück auf Null oder auf die Länge Null gebracht.

- Alle eröffneten Dateien werden geschlossen.

- Das Programm kann nicht durch die Anweisung CONT weiter ausgeführt werden.

Sollen vor den Änderungen die Inhalte einiger Variablen geprüft werden, müssen die Tasten Ctrl-Break betätigt werden, um in die Befehlsebene zu gelangen. Der Inhalt der Variablen ist noch vorhanden, da keine Programmzeile verändert wurde. Jetzt können alle Prüfungen durchgeführt werden. Danach verändert man die Zeile und führt das Programm erneut aus.

Notizen

KAPITEL 3. ALLGEMEINE INFORMATIONEN ÜBER DIE PROGRAMMIERUNG IN BASIC

Inhalt

Zeilenformat

Programmzeilen eines BASIC-Programms haben das folgende Format:

nnnnn BASIC Anweisung[:BASIC Anweisung...][' Kommentar]

Sie werden durch die Eingabetaste beendet. Das Format wird nachfolgend genau erklärt.

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

BASIC-Anweisungen: Eine BASIC-Anweisung ist entweder *ausführbar* oder *nicht ausführbar.* Ausführbare Anweisungen sind Programminstruktionen, die dem BASIC mitteilen, was als nächstes zu tun ist, wenn ein Programm ausgeführt wird. Zum Beispiel ist **PRINT X** eine ausführbare Anweisung. Nicht ausführbare Anweisungen wie z.B. DATA oder REM verusachen keine Aktion, wenn sie BASIC erkennt. Alle BASIC-Anweisungen werden im nächsten Kapitel genau erklärt.

Man kann, falls man möchte, mehrere BASIC-Anweisungen in einer Zeile haben, jedoch muß jede Anweisung in einer Zeile von der letzten durch einen Doppelpunkt getrennt sein, und die Gesamtanzahl der Zeichen darf 255 nicht überschreiten.

Beispiel:

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

Kommentare: Mit Hilfe des Apostrophs (') können an das Ende einer Zeile Kommentare angefügt werden. Durch das Apostroph wird der Kommentar vom Rest der Zeile getrennt.

Zeichensatz

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

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.

Die folgenden Sonderzeichen haben in BASIC eine besondere Bedeutung:

Zeichen	*Name*
	Leerstelle
=	Gleichheitszeichen oder Zuordnungssymbol
+	Pluszeichen oder Verkettungssymbol
–	Minuszeichen
*	Stern oder Multiplikationssymbol
/	Schrägstrich oder Divisionssymbol
\	Umgekehrter Schrägstrich oder Symbol für ganzzahlige Division
^	Fehlerzeichen oder Potenzierungssymbol
(	Linke Klammer
)	Rechte Klammer
%	Prozentzeichen oder Definitionszeichen für ganzzahligen Typ
#	Nummernzeichen oder Definitionszeichen für den Typ doppelte Genauigkeit
$	Dollarzeichen oder Definitionszeichen für den Typ Zeichenkette
!	Ausrufezeichen oder Definitionszeichen für den Typ einfache Genauigkeit
&	Ampersand
,	Komma
.	Punkt oder Dezimalpunkt
'	Apostroph oder Trennzeichen für Bemerkungen
;	Semikolon
:	Doppelpunkt oder Trennzeichen für Anweisungen
?	Fragezeichen (Abkürzung für PRINT)
<	kleiner als
>	größer als
"	Anführungszeichen oder Trennzeichen für Zeichenketten
_	Unterstreichungszeichen

Es gibt viele Zeichen, die gedruckt oder angezeigt werden können, obwohl sie im BASIC keine spezielle Bedeutung haben. Siehe Anhang G. "ASCII-Zeichencodes". Er enthält eine vollständige Liste aller Zeichen.

Reservierte Wörter

Einige Wörter bei BASIC haben eine bestimmte Bedeutung. Diese Wörter nennt man *Reservierte Wörter*. Reservierte Wörter beinhalten alle BASIC-Befehle, -Anweisungen, -Funktionsnamen und -Operatornamen. Reservierte Wörter dürfen nicht als Variablennamen benutzt werden.

Reservierte Wörter müssen 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 Wörter immer getrennt stehen müssen, so daß sie von BASIC erkannt werden können.

Die folgende Liste zeigt alle reservierten Wörter.

```
ABS
AND
ASC
ATN
AUTO
BEEP
BLOAD
BSAVE
CALL
CDBL
CHAIN
CHR$
CINT
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COM
COMMON
CONT
COS
```

CSNG
CSRLIN
CVD
CVI
CVS
DATA
DATE$
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DIM
DRAW
EDIT
ELSE
END
EOF
EQV
ERASE
ERL
ERR
ERROR
EXP
FIELD
FILES
FIX
FNxxxxxxxx
FOR
FRE
GET
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP
INPUT
INPUT#
INPUT$
INSTR
INT
KEY
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MERGE
MID$
MKD$
MKI$
MKS$
MOD
MOTOR
NAME
NEW
NEXT
NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
PAINT
PEEK
PEN
PLAY
POINT
POKE
POS
PRESET
PRINT
PRINT#
PSET
PUT

```
RANDOMIZE
READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SPC(
SQR
STEP
STICK
STOP
STR$
STRIG
STRING$
SWAP
SYSTEM
TAB(
TAN
THEN
TIME$
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VARPTR$
WAIT
WEND
WHILE
WIDTH
WRITE
WRITE#
XOR
```

Konstanten

Konstanten sind die aktuellen Werte, die BASIC während der Ausführung benutzt. Es gibt zwei Arten von Konstanten: Zeichenketten-(oder Zeichen-) Konstanten und numerische Konstanten.

Eine Zeichenkettenkonstante ist eine Folge von bis zu 255 Zeichen, die in Anführungszeichen stehen. Beispiele von Zeichenkettenkonstanten:

```
"HALLO"
"DM25,000.00"
"Anzahl der Angestellten"
```

Numerische Konstanten sind positive oder negative Zahlen. Ein Pluszeichen (+) ist für eine positive Zahl wahlfrei. Numerische Konstanten dürfen in BASIC keine Kommas enthalten. Es gibt fünf verschiedene Arten von numerischen Konstanten:

Ganzzahlig Ganze Zahlen zwischen −32768 und +32767 je einschließlich. Ganzzahlige Konstanten haben keinen Dezimalpunkt.

Festkomma Positive und negative reelle Zahlen, d.h. Zahlen, die einen Dezimalpunkt enthalten.

Gleitkomma Positive oder negative Zahlen, die in Exponentialform dargestellt sind (ähnlich einer wissenschaftlichen Darstellung). 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 nächster 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 regulären Festkommadarstellung könnte 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.

Hex

Hexadezimale Zahlen mit bis zu vier Zeichen und dem Vorsatz &H. Hexadezimale Zeichen sind die Zahlen 0 bis 9, A, B, C, D, E und F. Beispiele:

```
&H76
&H32F
```

Oktal

Oktale Zahlen mit bis zu sechs Ziffern mit dem Vorsatz &O oder nur &. Oktale Ziffern gehen von 0 bis 7. Beispiel:

```
&0349
&1234
```


Genauigkeit numerischer Werte

Numerische Konstanten können 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 und sind ganzzahlig. Reelle Zahlen doppelter Genauigkeit werden mit 17 Stellen Genauigkeit gespeichert und mit bis zu 16 Stellen ausgegeben. Mit 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 angefügten Ausrufezeichen (!).

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

- Acht oder mehr Ziffern oder
- Exponentialform mit D oder
- einem angefügten Nummernzeichen (#)

Beispiele für Konstanten einfacher und doppelter Genauigkeit:

Einfache Genauigkeit	**Doppelte Genauigkeit**
46.8	345692811
−1.09E−06	−1.0943D−06
3489.0	3489.0#
22.5!	7654321.1234

Variablen

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

Variablennamen

Die Variablennamen im BASIC dürfen jede Länge annehmen. Ist der Name länger als 40 Zeichen, werden nur die ersten 40 Zeichen geprüft.

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 über Typen siehe nächsten Abschnitt "Definition von Variablentypen".

Ein Variablenname darf kein reserviertes Wort sein, darf aber reservierte Wörter eingeschlossen haben. (Siehe unter "Reservierte Wörter", wo eine vollständige Liste der reservierten Wörter enthalten ist.) Ein Variablenname darf auch nicht ein reserviertes Wort mit einem der Zeichen am Ende sein, die seinen Typ deklarieren ($,%,!,#). Beispiel:

```
1Ø EXP = 5
```

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

```
1Ø EXPONENT = 5
```

erlaubt, da EXP nur der Teil eines 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.)

Definition von Variablentypen

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

Zeichenkettenvariablen besitzen als letztes Zeichen das Dollarzeichen ($). Beispiel:

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

Das Dollarzeichen ist ein Definitionszeichen für den Variablentyp. Es definiert eine Zeichenkettenvariable. Sonderzeichen, die den Typ der Variablen kennzeichnen, sind als letztes Zeichen des Namens erlaubt - siehe nächster Abschnitt. Weiter Informationen über Typen siehe nächsten Abschnitt "Definition von Variablentypen".

Numerische Variablennamen können ganze Zahlen und Werte einfacher oder doppelter Genauigkeit definieren. Obwohl man bei der Berechnung mit ganzen Zahlen oder Zahlen einfacher Genauigkeit eine geringere Genauigkeit bekommt, gibt es Ursachen, weshalb man eine bestimmte Genauigkeit auswählt.

- Variablen größerer Genauigkeit benötigen mehr Platz im Hauptspeicher. Dies ist wichtig, wenn der Platz eine Rolle spielt.

- Der Computer benötigt länger für Berechnungen mit Zahlen größerer Genauigkeit. Ein Programm mit wiederholten Berechnungen läuft mit ganzzahligen Variablen schneller.

Die Definitionszeichen für den Typ der numerischen Variablen und die Anzahl der Bytes, die für jeden Typ benötigt werden, sind wie folgt:

% Ganzzahlige Variable (2 Bytes)

! Variable einfacher Genauigkeit (4 Bytes)

Variable doppelter Genauigkeit (8 Bytes)

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

Es folgen 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 von Variablen definiert werden kann. Die BASIC-Anweisungen DEFINT, DEFSNG, DEFDBL und DEFSTR können in ein Programm eingefügt werden, um den Typ bestimmter Variablennamen zu definieren. Diese Anweisungen werden unter "DEFtyp-Anweisungen" in Kapitel 4 beschrieben. Bei allen Beispielen, die in dieser Broschüre folgen, wird angenommen, daß keine dieser Definitionen angegeben wurde, außer die Anweisungen werden explizit im Beispiel gezeigt.

Bereiche

Ein Bereich ist eine Gruppe oder eine Tabelle aus Werten, die über denselben Namen angesprochen werden können. Jeder einzelne Wert des Bereichs wird *Element* genannt. Bereichselemente sind Variablen und können in Ausdrücken und in jeder BASIC-Anweisung oder Funktion, die diese Variablen benutzt, verwendet werden.

Die Definition des Namens und des Typs eines Bereichs und das Setzen der Anzahl der Elemente und ihr Aufeinanderfolgen in dem Bereich wird *Definieren* oder *Dimensionieren* des Bereichs genannt. Dies wird gewöhnlich mit Hilfe der Anweisung DIM getan. Beispiel:

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

Damit wird ein eindimensionaler Bereich mit dem Namen B$ erzeugt. Alle seine Elemente sind Zeichenketten variabler Länge und den Elementen ist nichts zugeordnet.

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

Damit wird ein zweidimensionaler Bereich mit dem Namen A erzeugt. Da sich an dem Namen kein Definitionszeichen für den Typ befindet, besteht der Bereich aus Werten einfacher Genauigkeit. Zu Anfang sind alle Bereichselemente auf Null gesetzt.

Jedes Bereichselement wird mit dem Bereichsnamen, *indiziert* mit einer Anzahl von Zahlen, angesprochen. Ein Bereichsvariablenname hat so viele Indizes, wie der Bereich Dimensionen hat.

Der Index zeigt die Position des Elements im Bereich an. Null ist die niedrigste Position, wenn sie nicht explizit geändert wird (siehe "Anweisung OPTION BASE" in Kapitel 4). Die maximale Anzahl der Dimensionen eines Bereichs 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).

Den Bereich A kann man sich als Tabelle aus Zeilen und Spalten wie folgt vorstellen:

	Spalten			
Zeilen	A(0,0)	A(0,1)	A(0,2)	A(0,3)
	A(1,0)	A(1,1)	A(1,2)	A(1,3)
	A(2,0)	A(2,1)	A(2,2)	A(2,3)

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

Wird ein Bereichselement benutzt, bevor der Bereich definiert wurde, ist der Standardwert seiner Dimensionierung der Index 10.

Findet BASIC z.B. die Anweisung

```
5Ø SIS(3) = 500
```

und der Bereich SIS wurde zuvor nicht definiert, so wird er auf einen eindimensionalen Bereich mit 11 Elementen gesetzt, numeriert SIS(0) bis SIS(10). Dies ist die einzige Methode, um einen *eindimensionalen Bereich* implizit zu definieren.

Letztes Beispiel:

```
Ok
1Ø DIM YEARS(3,4)
2Ø YEARS(2,3)=1982
3Ø FOR ROW=Ø TO 3
4Ø FOR COLUMN=Ø TO 4
5Ø PRINT YEARS(ROW,COLUMN);
6Ø NEXT COLUMN
7Ø PRINT
8Ø NEXT ROW
RUN
 Ø  Ø  Ø  Ø  Ø
 Ø  Ø  Ø  Ø  Ø
 Ø  Ø  Ø  1982  Ø
 Ø  Ø  Ø  Ø  Ø
Ok
```

Konvertierung von Zahlen einer Genauigkeit in eine andere in BASIC

Falls nötig, wandelt BASIC Zahlen einer Genauigkeit in eine andere um. Die folgenden Regeln und Beispiele sollte man in Erinnerung halten.

1. Wird der numerische Wert einer Genauigkeit einer numerischen Variablen einer andern 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, als Gegenteil von Abschneiden, passiert, wenn ein Wert höherer Genauigkeit einer Variablen mit niedrigerer Genauigkeit zugeordnet wird (Beispiel: Ändern von doppelter in einfache Genauigkeit).

   Beispiel:

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

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

3. Wird eine Zahl kleinerer Genauigkeit in eine Zahl höherer Genauigkeit umgewandelt, kann die sich ergebende Zahl höherer Genauigkeit nicht genauer sein als die Zahl mit niedrigerer Genauigkeit. Wird z.B. (A), ein Wert einfacher Genauigkeit, B#, einer Variablen doppelter Genauigkeit, 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 ursprünglichen Wert einfacher Genauigkeit ist kleiner als 6.3E-8 mal dem ursprünglichen Wert einfacher Genauigkeit.

Beispiel:

```
Ok
1Ø A = 2.Ø4
2Ø B# = A
3Ø PRINT A;B#
RUN
 2.Ø4  2.Ø39999961853Ø27
Ok
```

4. Während der Berechnung der Ausdrücke 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 übertragen.

   Beispiele:

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

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

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

   Die Berechnung wurde in doppelter Genauigkeit durchgeführt. Das Ergebnis wurde an D übergeben (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 übergeben ein ganzzahliges Ergebnis. Die Operanden müssen im Bereich −32768 bis 32767 liegen, sonst wird der Fehler Overflow (Überlauf) angezeigt.

Numerische Ausdrücke 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 führen 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:

- Arithmetisch
- Vergleich
- Logisch
- Funktionen

Arithmetische Operatoren

Mit den arithmetischen Operationen werden die gewöhnlichen Rechenoperationen durchgeführt, wie Addition und Subtraktion. In ihrer Rangordnung aufgeführt sind es:

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

(Wenn ein mathematisches Wissen vorliegt, fällt auf, daß dies die Standardrangfolge dieser Operatoren ist.) Obwohl die meisten Operationen bekannt aussehen, sehen zwei ein wenig unbekannt aus - ganzzahlige Division und Modulo-Arithmetik.

Ganzzahlige Division: Die ganzzahlige Division wird durch den umgekehrten Schrägstrich (\ ) dargestellt. Die Operanden werden zu ganzen Zahlen gerundet (müssen sich in dem Bereich –32768 bis 32767 befinden), bevor die Division ausgeführt wird, und der Quotient wird zu einer ganzen Zahl abgebrochen.

Beispiel:

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

Modulo-Arithmetik: Modulo-Arithmetik wird durch den Operator MOD angegeben. Dabei wird ein ganzzahliger Wert, und zwar der Rest einer ganzzahligen Division, zurückgegeben.

Beispiel:

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

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

```
Ok
PRINT 25.68 MOD 6.99
 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.)

Vergleichsoperatoren

Vergleichsoperatoren werden dazu benutzt, zwei Werte zu vergleichen. Die Werte können 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 ändern. (Siehe "Anweisung IF" in Kapitel 4.)

Operator	Getesteter Vergleich	Beispiel-Ausdrücke
=	Gleich	X=Y
<> oder ><	Ungleich	X<>Y X><Y
<	Kleiner als	X<Y
>	Größer als	X>Y
<= oder =<	Kleiner als oder gleich	X<=Y X=<Y
>= oder =>	Größer als oder gleich	X>=Y 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 ausgeführt. Zum Beispiel 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 ausgeführt.

```
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 größer als ihre zugehörige 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 "Anhang G. ASCII-Zeichencodes" für eine vollständige 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 höheren Code-Nummer. Wird während eines Zeichenkettenvergleichs das Ende einer Zeichenkette erreicht, ist die kürzere Zeichenkette kleiner.

Führende und nachfolgende Leerzeichen werden beachtet. Zum Beispiel sind alle folgenden Vergleichsausdrücke wahr (d.h. das Ergebnis der Vergleichsoperation ist −1).

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

Alle Zeichenkettenkonstanten, die in Vergleichsausdrücken benutzt werden, müssen in Anführungszeichen eingeschlossen sein.

Logische Operatoren

Mit logischen Operatoren werden logische oder *Bool'sche* Operationen mit numerischen Werten durchgeführt. So wie gewöhnlich mit Vergleichsoperatoren Entscheidungen über den Programmablauf getroffen werden, werden logische Operatoren normalerweise dazu benutzt, zwei oder mehr Beziehungen zu verbinden und einen Wert für wahr oder falsch zurückzugeben, 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 zurück. Ein Operand eines logischen Operators wird als wahr angenommen, wenn er nicht Null ist (wie −1, die von einem Vergleichsoperator übergeben 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 für Bit durchgeführt wird. Dies wird nachfolgend in allen Einzelheiten erklärt.

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

Jeder Operator gibt die Werte zurück, die in den folgenden Tabellen angezeigt sind ("T" steht für wahr oder einen Wert ungleich Null. "F" steht für 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 für Entscheidungen benutzt werden:

```
IF ER>6Ø AND SIE<2Ø THEN 1ØØØ
```

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

```
IF I>1Ø OR K<Ø THEN 5Ø
```

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

```
5Ø IF NOT (P=-1) THEN 1ØØ
```

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

```
1ØØ FLAG% = NOT FLAG%
```

Dies ist eine Möglichkeit, einen Wert laufend von wahr auf falsch und falsch auf wahr umzudrehen.

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 (Überlauf) 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 ausgeführt. Das bedeutet, jedes Bit des Ergebnisses ist das Ergebnis der zwei korrespondierenden Bits in den beiden Operanden, gemäß der Tabellen für 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 = binär 111111 und 16 = binär 10000 ist, ergibt 63 AND 16 010000 binär, was wiederum 16 ist.

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

B wird auf den Wert 8 gesetzt. Da −1 = binär 11111111 11111111 und 8 = binär 1000 ist, ergibt −1 AND 8 binär 00000000 0001000 oder 8.

```
C = 4 OR 2
```

Hier wird C auf den Wert 6 gesetzt. Da 4 = binär 100 und 2 = binär 010 ist, ergibt 4 OR 2 = binär 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 binär. NOT X ist dann binär 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 übergibt.

Numerische Funktionen

Eine Funktion wird wie eine Variable in einem Ausdruck benutzt, um eine vorbestimmte Operation mit einem oder mehreren Operanden auszuführen. BASIC besitzt eingebaute Funktionen, die im System enthalten sind, wie z.B. SQR (Quadratwurzel) oder SIN (Sinus). Alle eingebauten BASIC-Funktionen sind im Kapitel 4. "BASIC-Befehle, -Anweisungen, -Funktionen und -Variablen" beschrieben. Detaillierte Beschreibungen befinden sich auch im alphabetischen Teil des Kapitels 4.

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

Reihenfolge der Ausführung

Die Kategorien der numerischen Operationen wurden in der Rangfolge der Operationen beschrieben und die Rangfolge jeder Operation in der Kategorie innerhalb der Kategorie. Zusammenfassung:

1. Funktionsaufrufe werden zuerst berechnet.

2. Arithmetische Operationen werden danach in dieser Reihenfolge ausgeführt:

   a. ^
   b. – (Minusvorzeichen)
   c. *, /
   d. /
   e. MOD
   f. +,–

3. Vergleichsoperationen werden danach ausgeführt.

4. Logische Operationen werden zuletzt in dieser Reihenfolge ausgeführt:

   a. NOT
   b. AND
   c. OR
   d. XOR
   e. EQV
   f. IMP

Operationen auf derselben Ebene in der Liste werden von links nach rechts ausgeführt. Soll die Reihenfolge der Ausführung einer Operation geändert werden, müssen Klammern benutzt werden. Operationen in Klammern werden zuerst ausgeführt. Innerhalb der Klammern gilt die gewöhnliche Rangordnung der Ausführung von Operationen.

Es folgen einige algebraische Ausdrücke und ihre Darstellung in BASIC.

Algebraischer Ausdruck	BASIC-Ausdruck
$X+2Y$	X+Y*2
$\frac{X-Y}{Z}$	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 müssen, wie im letzten Beispiel gezeigt, durch Klammern getrennt werden.

Zeichenkettenausdrücke 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 aneinanderzuhängen. Es gibt nur zwei Zeichenkettenoperatoren:

- Verkettung
- Funktionen

Dabei muß beachtet werden, daß, obwohl man die Vergleichsoperatoren =, <>, <, >, <=, und >= benutzen kann, um Zeichenketten zu vergleichen, sie nicht als Zeichenkettenoperatoren betrachtet werden, weil sie ein numerisches Ergebnis ergeben und nicht ein Zeichenkettenergebnis. Der Abschnitt "Vergleichsoperatoren" in diesem Kapitel gibt eine Erklärung, wie Zeichenketten mit Hilfe von Vergleichsoperatoren verglichen werden können.

Verkettung

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

```
Ok
10 COMPANY$ = "IBM"
20 TYPE$ = " Personal"
30 FULLNAME$ = TYPE$ + " Computer"
40 PRINT COMPANY$+FULLNAME$
RUN
IBM Personal Computer
Ok
```


Zeichenkettenfunktionen

Eine Zeichenkettenfunktion funktioniert wie eine numerische Funktion, nur daß sie als Ergebnis eine Zeichenkette übergibt. Eine Zeichenkettenfunktion wird in einem Ausdruck benutzt, um eine vorbestimmte Operation aufzurufen, die mit einer oder mehreren Operanden ausgeführt werden soll. BASIC besitzt "eingebaute" Funktionen, die sich im System befinden, wie z.B. MID$, die eine Zeichenkette aus der Mitte einer anderen Zeichenkette übergibt, oder CHR$, die das Zeichen für den angegebenen ASCII-Code übergibt. Alle eingebauten BASIC-Funktionen werden in Kapitel 4 unter "Funktionen und Variablen" beschrieben.

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

Ein- und Ausgabe

Der Rest dieses Kapitels enthält Informationen über Ein- und Ausgabe (E/A) in BASIC. Die folgenden Themen werden behandelt:

- Dateien - Wie BASIC Dateien anspricht, Namensgebung für Dateien und Einheitennamen.
- Bildschirm - Anzeigen auf dem Bildschirm mit Betonung der Grafik.
- Andere Einrichtungen - Zeituhr, Ton, Lichtstift und Spielpulte

Dateien

Eine Datei ist eine Sammlung von Informationen, die irgendwo anders als im Hauptspeicher des IBM Personalcomputers gespeichert ist. Zum Beispiel kann die Information in einer Datei auf Diskette oder Kassette gespeichert sein. Um die Information benutzen zu können, muß die Datei *eröffnet* werden, um BASIC mitzuteilen, wo die Information steht. Dann kann die Datei für Eingabe und/oder Ausgabe benutzt werden.

BASIC unterstützt 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 tatsächlich um die Benutzung einer Kassetten- oder Diskettendatei handelt oder um die Verbindung mit einem anderen Computer.

Dateinummer: BASIC führt Ein-/Ausgabeoperationen mit Hilfe einer Dateinummer aus. Die Dateinummer ist die Verbindung zur aktuellen physischen Datei, wenn sie eröffnet ist. Sie gibt den Pfad für die Datensammlung. Eine Datei kann irgendeine Nummer, Variable oder ein Ausdruck zwischen 1 und *n* sein, wobei *n* die maximale Anzahl der erlaubten Dateien darstellt. *n* ist 4 im Kassetten-BASIC und standardmäßig 3 im Disketten- oder erweiterten BASIC. Sie kann bis zu einem Maximum von 15 mit Hilfe der Angabe /F: des BASIC-Befehls für Disketten- und erweitertes BASIC abgeändert werden.

Namensgebung für Dateien

Die physische Datei wird durch ihre *Dateiangabe* beschrieben.

Die Dateiangabe ist eine Zeichenkette der folgenden Form:

einheit:dateiname

Der Einheitenname sagt BASIC, *wo* nach der Datei zu suchen ist. Der Dateiname sagt BASIC, *welche* Datei auf einer bestimmten Einheit gesucht werden soll. Manchmal werden nicht beide Informationen benötigt. Deshalb ist die Angabe der Einheit und des Dateinamens wahlfrei. Der oben angezeigte Doppelpunkt (:) muß beachtet werden. Sobald eine Einheit angegeben ist, muß der Doppelpunkt folgen, selbst wenn nicht unbedingt auch ein Dateiname angegeben ist. Von nun an wird der Doppelpunkt als Teil des Einheitennamens angesehen.

> **Hinweis:** Die Dateispezifikation für Datenfernverarbeitungseinheiten ist ein wenig verschieden. Der Dateiname wird durch eine Liste von Auswahlen ersetzt, die z.B. Dinge wie Leitungsgeschwindigkeit spezifizieren. Siehe "Anweisung OPEN COM..." in Kapitel 4.

Wird eine Zeichenkettenkonstante für die *Dateiangabe* verwendet, muß sie in Anführungszeichen eingeschlossen werden. Beispiel:

```
LOAD "B:ROTHERM.ARK"
```

Einheitenname: Der Einheitenname besteht aus zwei bis vier Zeichen, gefolgt von einem Doppelpunkt (:). Im folgenden wird eine Liste aller Einheitennamen gezeigt, die aussagt, welche Einheit angesprochen wird , wozu die Einheit benutzt werden kann (Eingabe oder Ausgabe) und welche Versionen von BASIC die Einheit unterstützen.

Tabelle der Einheitennamen

KYBD:	Eingabetastatur. Nur Eingabe, alle Versionen des BASIC.
SCRN:	Bildschirm. Nur Ausgabe, alle Versionen des BASIC.
LPT1:	Erster Drucker. Ausgabe, alle Versionen; oder wahlfreier Zugriff, Disketten- und erweitertes BASIC.
LPT2.	Zweiter Drucker. Ausgabe oder wahlfreier Zugriff, Disketten- und erweitertes BASIC.
LPT3:	Dritter Drucker. Ausgabe oder wahlfreier Zugriff, Disketten- und erweitertes BASIC.

DATENFERNVERARBEITUNGSEINHEITEN

COM1:	Erster Anschluß für asynchrone Übertragung. Eingabe und Ausgabe, Disketten- und erweitertes BASIC.
COM2:	Zweiter Anschluß für asynchrone Übertragung. Ein- und Ausgabe, Disketten- und erweitertes BASIC.

SPEICHEREINHEITEN

CAS1:	Kassettenrecorder. Eingabe und Ausgabe, alle Versionen.
A:	Erste Disketteneinheit. Eingabe und Ausgabe, Disketten- und erweitertes BASIC.
B:	Zweite Disketteneinheit. Eingabe und Ausgabe, Disketten- und erweitertes BASIC.

Siehe unter "Suchfolge für Anschlüsse" in "Anhang I. Technische Informationen und Tips." Dort stehen Informationen darüber, welche Anschlüsse sich auf die Drucker- und Datenfernverarbeitungsnamen beziehen.

Dateiname: Der Dateiname muß nach den folgenden Regeln aufgebaut sein.

Für Kassettendateien:

- Der Name darf nicht länger als acht Zeichen sein.

- Der Name darf keine Doppelpunkte, hex '00' oder hex 'FF' (dezimal 255) enthalten.

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.erweiterung

  Der Name kann ein bis acht Zeichen lang sein. Die *erweiterung* darf nicht länger als drei Zeichen sein.

  Ist die *erweiterung* länger als drei Zeichen, werden die Extrazeichen unterdrückt. Ist der *name* länger als acht Zeichen und die *erweiterung* ist nicht eingeschlossen, fügt BASIC einen Punkt nach dem achten Zeichen ein und nimmt die Extrazeichen (bis zu drei) als *erweiterung*. Ist der *name* länger als acht Zeichen und eine *erweiterung* ist eingeschlossen, wird ein Fehler angezeigt.

- Nur die folgenden Zeichen sind für *name* und *erweiterung* erlaubt.

  A bis Z
  0 bis 9
  < > ( ) { }
  # $ % ^ & !
  @
  - _ ‘ ’ \ ~ |

Beispiele für Dateinamen für Disketten- und erweitertes BASIC sind:

```
27HAL.DAD

VDL

PROGRAM1.BAS

$$@(!).123
```

Die folgenden Beispiele zeigen, wie BASIC Namen und Erweiterungen abbricht, wenn sie, wie oben erklärt, zu lang sind.

A23456789JKLMN	wird zu:	A2345678.9JK
@HOME.TRUM1Ø	wird zu:	@HOME.TRU
SHERRYLYNN.BAS	erzeugt einen Fehler	

Benutzung des Bildschirms

Mit BASIC kann man Texte, Sonderzeichen, Punkte, Linien oder komplexere Dinge farbig oder in schwarz-weiß anzeigen. Wieviel man machen kann, hängt davon ab, welchen Bildschirmanschluß der IBM Personalcomputer hat.

Bildschirmanschlüsse

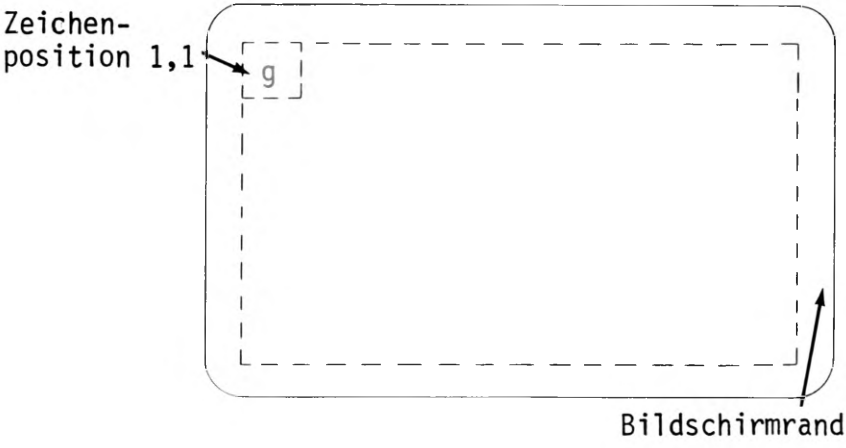
Der IBM Personalcomputer besitzt zwei Bildschirmanschlüsse: den IBM Schwarz/Weiß-Bildschirm- und Parallel-Druckeranschluß und den Farb-/Grafik-Bildschirmanschluß.

Mit dem IBM Schwarz/Weiß-Bildschirm- und Parallel-Druckeranschluß ist eine Textanzeige in schwarz und weiß möglich. *Text* bezieht sich auf Buchstaben, Zahlen und alle Sonderzeichen des normalen Zeichensatzes. Es gibt eine geringe Möglichkeit, Bilder mit Hilfe von Speziallinien und Blockzeichen zu bilden. Blinken, Umkehranzeige, Nicht-anzeigen, Intensivanzeige und Unterstreichen werden durch Setzen von Parametern in der Anweisung COLOR unterstützt.

Der Farb-/Grafik-Bildschirmanschluß arbeitet auch im Textmodus. Er erlaubt aber, Texte in 16 verschiedenen Farben anzuzeigen. (Die Anzeige kann auch nur in schwarz-weiß erfolgen, wenn die Parameter in den Anweisungen SCREEN oder COLOR gesetzt werden.) Hier sind auch alle grafischen Möglichkeiten gegeben, komplexe Bilder zu zeichnen. In dieser grafischen Fähigkeit kann man den Bildschirm mit allen Punkten in mittlerer und hoher Auflösung adressieren. Das ist wesentlich mehr als die Fähigkeit, nur mit den Speziallinien und Blockzeichen zu zeichnen, die im Textmodus *verfügbar* sind. Von jetzt an wird der Ausdruck *Grafiken* nur für die spezielle Fähigkeit des Farb-/Grafik-Bildschirmanschlusses benutzt. Die Benutzung des erweiterten Zeichensatzes mit speziellen Linien und Blockzeichen wird nicht als Grafik angesehen.

Textmodus

Der Bildschirm kann wie folgt dargestellt werden:



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, abhängig vom Wert der Anweisung WIDTH). Sie sind von links nach rechts von 1 bis 40 (oder 80) numeriert. Die Positionsnummern werden von der Anweisung LOCATE benutzt und die Werte werden von den Funktionen POS(0) und CSRLIN übergeben. Zum Beispiel steht das Zeichen in der obersten linken Ecke des Bildschirms auf Zeile 1, Position 1.

Zeichen werden normalerweise mit der Anweisung PRINT auf dem Bildschirm angezeigt. Die Zeichen werden an der Stelle des Positionsanzeigers angezeigt. Die Zeichen werden auf jeder Zeile von links nach rechts von Zeile 1 bis 24 angezeigt. Würde der Positionsanzeiger normalerweise zu Zeile 25 gehen, würden die Zeilen 1 bis 24 um eine Zeile nach oben geschoben, so daß die Zeile 1 vom Bildschirm verschwindet. Jetzt ist die Zeile 24 leer und der Positionanzeiger bleibt auf Zeile 24, um dort mit der Ausgabe fortzufahren.

Die Zeile 25 wird normalerweise benutzt, die "Funktionstasten" anzuzeigen (siehe "Anweisung KEY" in Kapitel 4), aber es ist möglich, diesen Bereich des Bilschirms zu überschreiben, wenn die Anzeige der "Funktionstasten" augeschaltet 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 für jedes Zeichen setzen. Es ist auch möglich, die Zeichen blinken zu lassen.

Es gibt 16 Farben, wenn der Farb-/ Grafikbildschirmanschluß benutzt wird:

0	schwarz	8	grau
1	blau	9	hellblau
2	grün	10	hellgrün
3	kobaltblau	11	hellkobaltblau
4	rot	12	hellrot
5	violett	13	hellviolett
6	braun	14	gelb
7	weiß	15	hochintensives Weiß

Die Farben können abhängig vom benutzten Bildschirm variieren.

Die meisten Fernseher oder Monitoren haben einen "Rand," der außerhalb des Bereiches liegt, der für die Zeichen benutzt wird. Mit der Anweisung COLOR kann auch der Rand farbig angezeigt werden.

Es folgt eine Auflistung aller Anweisungen, die benutzt werden können, um im Textmodus Informationen anzuzeigen:

```
CLS             SCREEN
COLOR           WIDTH
LOCATE          WRITE
PRINT
```

Die folgenden Funktionen und Systemvariablen können im Textmodus benutzt werden:

```
CSRLIN          SPC
POS             TAB
SCREEN
```

Es gibt eine weitere Sondereinrichtung, die im Textmodus möglich ist, wenn der Farb-/Grafikbildschirmanschluß vorhanden ist: mehrere Bildschirmseiten. Der Farb-/Grafikbildschirmanschluß hat einen 16K-Bildschirmpuffer. Im Textmodus werden aber nur 2K benötigt (oder 4K für eine Breite von 80 Spalten). So ist der Puffer in verschiedene *Seiten* aufgeteilt, die individuell beschrieben und/oder angezeigt werden können. Bei der Breite von 40 Spalten gibt es acht Seiten, numeriert von 0 bis 7, bei der Breite von 80 Spalten 4 Seiten, numiert von 0 bis 3. Einzelheiten siehe Kapitel 4 unter "Anweisung SCREEN."

Grafischer Modus

Der grafische Modus ist nur verfügbar, wenn der Farb-/Grafikbildschirmanschluß vorhanden ist.

Die BASIC-Anweisungen erlauben es, in zwei graphischen Auflösungen zu arbeiten:

- mittlere Auflösung: 320 x 200 Punkte und 4 Farben
- hohe Auflösung: 640 x 200 Punkte und 2 Farben

Die Auflösung kann mit Hilfe der Anweisung SCREEN ausgewählt werden.

Die in BASIC verfügbaren Anweisungen für Grafiken sind folgende:

```
CIRCLE      PAINT
COLOR       PRESET
DRAW        PSET
GET         PUT
LINE        SCREEN
```

Die einzige graphische Funktion ist:

```
POINT
```

Mittlere Auflösung: Bei der mittleren Auflösung 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 Bildschirms die Koordinaten (0,0) und die untere rechte Ecke des Bildschirms die Koordinaten (319,199) hat. (Falls die normale mathematische Methode bekannt ist, um Koordinaten zu numerieren, sieht das hier umgekehrt aus.)

Die mittlere Auflösung ist ungewöhnlich wegen ihrer Farbeinrichtungen. Wird am Bildschirm etwas in mittlerer Auflösung angezeigt, kann man eine Farbnummer von 0, 1, 2 oder 3 angeben. Diese Farben sind nicht fest wie bei den 16 Farben des Textmodus. Man wählt die aktuelle Farbe für die Farbnummer 0 aus und wählt eine oder zwei "Farbpaletten" für die anderen drei Farben mit Hilfe der Anweisung COLOR aus. Eine Farbpalette besteht aus drei Farben, die den Farbnummern 1, 2 und 3 zugeordnet werden. Wird eine Palette mit Hilfe der Anweisung COLOR geändert, werden alle Farben des Bildschirms geändert, um sich der neuen Farbpalette anzugleichen.

Auch wenn man sich im grafischen Modus befindet, ist es möglich, Textzeichen am Bildschirm anzuzeigen. Die Größe der Zeichen ist die gleiche wie im Textmodus. Das bedeutet 25 Zeilen zu je 40 Zeichen. Bei der mittleren Auflösung hat der Vordergrund die Farbe Nr. 3, der Hintergrund die Farbe Nr. 0.

Wird keine U.S.-Tastatur benutzt, siehe "Farb-/Grafikadapterzeichen" im Abschnitt DOS des Bedienerhandbuchs.

Hohe Auflösung: Bei der hohen Auflösung gibt es horizontal 640 Punkte und vertikal 200 Punkte. Wie bei der mittleren Auflösung werden diese Punkte beginnend mit Null numeriert, so daß der unterste rechte Bildschirmpunkt die Koordinaten (639,199) hat.

Die hohe Auflösung ist ein wenig einfacher zu beschreiben als die mittlere Auflösung, da sie nur aus zwei Farben besteht: 0 (Null) und 1 (Eins). Null ist immer schwarz und Eins ist immer weiß.

Werden Textzeichen in hoher Auflösung 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.

Wird keine U.S.-Tastatur benutzt, siehe "Farb-/Grafikadapterzeichen" im Abschnitt DOS des Bedienerhandbuchs.

Angabe von Koordinaten: Die graphischen Anweisungen benötigen 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 einen anderen Weg, Koordinaten anzugeben, bekannt als *relative Form*. Mit Hilfe dieser Form wird BASIC angezeigt, wo sich der Punkt relativ zum letzten angesprochenen Punkt befindet. Die Form sieht wie folgt aus:

```
STEP  (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. Wenn diese Anweisungen in Kapitel 4."BASIC-Befehle, -Anweisungen und -Variablen" besprochen werden, wird auch erklärt, was jede Anweisung als letzten angesprochenen Punkt setzt.

> **Hinweis:** Mit den Grafikanweisungen darf nicht über die Grenzen des Bildschirms gezeichnet werden.

Das Beispiel zeigt die Benutzung der beiden Koordinatenformen:

```
100 SCREEN 1
110 PSET (200,100) 'absolute form
120 PSET STEP (10,-20) 'relative form
```

Hierbei werden zwei Punkte auf dem Bildschirm gesetzt. Ihre aktuellen Koordinaten sind (200,100) und (210,80).

Andere Ein-/Ausgabeeinrichtungen

Zeituhr

Es ist möglich, die folgenden Systemvariablen zu setzen und zu lesen:

DATE$ Zeichenfolge aus zehn Zeichen, die das Systemdatum in der Form *mm-dd-jjjj* beinhaltet.

TIME$ Zeichenkette aus acht Zeichen, die die Zeit wie folgt anzeigt: *hh:mm:ss.*

Ton und Musik

Mit den folgenden Anweisungen können mit dem IBM Personalcomputer Töne erzeugt werden:

BEEP Der Lautsprecher erzeugt ein kurzes akustisches Signal.

SOUND Ergibt einen einfachen Ton einer vorgegebenen Frequenz und Dauer.

PLAY Spielt nach einer vorgegebenen Zeichenkette Musik.

Lichtstift

Mit den folgenden Anweisungen und Funktionen kann mit Hilfe eines Lichtstiftes eine Eingabe in BASIC erfolgen.

PEN	Funktion, die anzeigt, ob der Lichtstift ausgelöst wurde und seine Koordinaten angibt.
PEN	Anweisung, die die Funktion des Lichtstiftes aktiviert/inaktiviert.
ON PEN	Anweisung, die eine Aktivität des Lichtstiftes ausliest.

Spielpulte

Spielpulte können in einer interaktiven Umgebung hilfreich sein. BASIC unterstützt zwei zweidimensionale (x- und y-Koordinaten) Spielpulte oder vier eindimensionale Spielkonsolen, jeweils mit einem Knopf. (Vier Knöpfe sind nur im erweiterten BASIC unterstützt.) Dazu können die folgenden Anweisungen und Funktionen benutzt werden:

STICK	Funktion, die die Koordinaten eines Spielpults angibt.
STRIG	Funktion, die den Status des Knopfes eines Spielpultes angibt (oben oder unten).

STRIG Anweisung, die die Funktion STRIG aktiviert/inaktiviert.

ON STRIG Anweisung, die dazu benutzt wird, den betätigten Knopf auszulesen.

STRIG(n) Anweisung, die die Unterbrechung für den Spielpultknopf aktiviert/inaktiviert.

Hinweis: Der Lichtstift kann nur mit dem Farb-/Grafikbildschirmanschluß verwendet werden. Spielpulte können nur mit einem Spielsteueranschluß benutzt werden.

Notizen:

KAPITEL 4. BASIC-BEFEHLE, -ANWEISUNGEN, -FUNKTIONEN UND -VARIABLEN

Inhalt

Benutzen dieses Kapitels

Dieses Kapital enthält eine vollständige Beschreibung aller BASIC-Befehle, -Anweisungen, -Funktionen und -Variablen. Die eingebauten BASIC-Funktionen und -Variablen können in jedem Programm ohne weitere Definition benutzt werden. Die ersten Seiten enthalten eine Liste aller Befehle, Anweisungen, Funktionen und Variablen. Diese Liste kann für einen schnellen Überlick nützlich sein. Der Rest des Kapitels enthält detaillierte Beschreibungen jedes Befehls, jeder Anweisung, jeder Funktion und Variablen, alphabetisch angeordnet.

Die Unterscheidung zwischen Befehl und Anweisung ist im großen und ganzen eine Sache der Tradition. Befehle werden gewöhnlich im direkten Modus eingegeben, weil sie im allgemeinen an Programmen arbeiten. Anweisungen übernehmen im allgemeinen innerhalb eines Programms den Programmablauf und werden deshalb gewöhnlich im indirekten Modus als Teil einer Programmzeile eingegeben. Nun ist es aber so, daß die meisten BASIC-Befehle und -Anweisungen direkt oder indirekt eingegeben werden können.

Die Beschreibung der Befehle, Anweisungen, Funktionen oder Variablen in diesem Kapitel ist wie folgt aufgeteilt:

Zweck: Beschreibt, was Befehl, Anweisung, Funktion oder Variable tut.

Version: Zeigt an, in welcher Version des BASIC der Befehl, die Anweisung, Funktion oder Variable erlaubt ist. Zum Beispiel steht unter der "Anweisung CHAIN" in diesem Kapitel unter Version:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	(**)

Die Sterne zeigen an, welche BASIC-Versionen die Anweisung unterstützen. Das Beispiel zeigt, daß die Anweisung CHAIN in Programmen benutzt werden kann, die für die BASIC-Version Diskette und Erweitert geschrieben werden.

In diesem Beispiel stehen die Sterne unter "Umwandlungsprogramm" in Klammern. Dies bedeutet, daß die Anweisung mit dem BASIC-Interpretierer anders arbeitet als mit dem BASIC-Umwandlungsprogramm des IBM Personalcomputers. Das Umwandlungsprogramm ist eine zusätzlich von der IBM verfügbare Software.

Format: Zeigt das richtige Format für einen Befehl, eine Anweisung, Funktion oder Variable. Eine vollständige Beschreibung der Formatsyntax steht im Vorwort. Die folgenden Regeln muß man im Gedächtnis behalten.

- Wörter in Großbuchstaben sind Schlüsselwörter und müssen genauso wie gezeigt eingegeben werden. Sie können in jeder Kombination von Groß- und Kleinbuchstaben eingegeben werden. BASIC wandelt Schlüsselwörter immer in Großbuchstaben um (außer sie sind Teil einer in Anführungszeichen stehenden Zeichenkette, Bemerkung oder DATA-Anweisung).

- Alle Angaben in kursiven Kleinbuchstaben müssen angegeben werden.

- Angaben in eckigen Klammern ([]) sind wahlfrei.

- Hintereinanderstehende Punkte (...) zeigen an, daß eine Angabe so oft wie gewünscht wiederholt werden kann.

- Jede Interpunktion, außer eckigen Klammern, wie z.B. Kommas (,), Klammern (), Semikolon (;), Bindestrich (-) oder Gleichheitszeichen (=) müssen, wo angezeigt, eingefügt werden.

Bemerkungen: Beschreibt in Einzelheiten, wie der Befehl, die Anweisung, die Funktion oder die Variable benutzt wird.

Beispiel: Zeigt Anweisungen im direkten Modus, Beispielprogramme oder einen Programmabschnitt, in dem die Benutzung des Befehls, der Anweisung, der Funktion oder der Variablen gezeigt wird.

In den in diesem Kapitel angegebenen Formaten wurden einige Parameter wie folgt abgekürzt:

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

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

Funktionen und Variablen: Dieses Kapitel enthält auch alle in BASIC eingebauten Funktionen und Variablen. Sie können in jedem Programm ohne vorherige Definition benutzt werden.

In der Formatbeschreibung werden die meisten Funktionen und Variablen auf der rechten Seite einer Zuordnungsanweisung gezeigt. Dies soll daran erinnern, daß sie nicht wie Anweisungen oder Befehle benutzt werden. Dabei ist nicht gemeint, daß sie nur in Zuordnungsanweisungen benutzt werden dürfen. Man kann sie überall wie eine normale Variable benutzen - *außer* auf der linken Seite einer Zuordnungsanweisung. Alle Ausnahmen für eine Funktion oder Variable stehen in dem Abschnitt, in dem die Funktion oder Variable beschrieben wird. Einige Funktionen können nur in den Anweisungen PRINT benutzt werden. Sie werden als Teil der Anweisung PRINT gezeigt.

> **Hinweis:** Numerische Funktionen übergeben nur ganze Zahlen oder Werte einfacher Genauigkeit, außer es wird etwas anderes angezeigt.

Befehle

Die folgende Liste enthält alle Befehle, die in BASIC benutzt werden. Die Syntax jedes Befehls wird angezeigt, jedoch nicht immer in seiner Ganzheit. Detaillierte Informationen über jeden Befehl sind im alphabetischen Teil dieses Kapitels zu finden. Der nächste Abschnitt dieses Kapitels enthält unter "Anweisungen" eine Liste aller BASIC-Anweisungen.

Befehl	Aktion
AUTO nummer, schrittweite	Erzeugt automatisch Zeilennummern.
BLOAD dateiangabe, relativzeiger	Lädt binäre Daten (wie z.B. Programme in Maschinensprache) in den Hauptspeicher.
BSAVE dateiangabe, relativzeiger, länge	Speichert binäre Daten.
CLEAR ,n,m	Löscht Programmvariablen und setzt wahlfrei einen Hauptspeicherbereich.
CONT	Fährt mit der Programmausführung fort.
DELETE zeile1-zeile2	Löscht angegebene Programmzeilen.

Befehl	Aktion
EDIT zeile	Zeigt eine Programmzeile für eine Änderung an.
FILES dateiangabe	Listet Dateien aus dem Disketteninhaltsverzeichnis an, die mit der Dateiangabe übereinstimmen.
KILL dateiangabe	Löscht eine Diskettendatei.
LIST zeile1-zeile2,dateiangabe	Listet Programmzeilen auf dem Bildschirm oder auf einer anderen angegebenen Einheit auf.
LLIST zeile1-zeile2	Listet Programmzeilen auf dem Drucker auf.
LOAD dateiangabe	Lädt eine Programmdatei. Durch Einfügen der Auswahl R kann sie sofort ausgeführt werden.
MERGE dateiangabe	Mischt ein gespeichertes Programm mit dem Programm im Hauptspeicher.
NAME dateiangabe AS dateiname	Gibt einer Diskettendatei einen neuen Namen.
NEW	Löscht das laufende Programm und seine Variablen im Hauptspeicher.

Befehl	Aktion
RENUM neue nummer, alte nummer, schrittweite	Numeriert Programmzeilen neu.
RESET	Initialisiert Disketteninformationen neu. Ähnlich wie CLOSE.
RUN dateiangabe	Führt ein Programm aus. Durch die Auswahl R können Dateien offen gehalten werden.
RUN zeile	Führt ein im Hauptspeicher befindliches Programm beginnend mit der angegebenen Zeile aus.
SAVE dateiangabe	Speichert ein im Hauptspeicher befindliches Programm unter dem angegebenen Dateinamen. Durch Auswahl A wird es im ASCII-Code, durch Auswahl P im geschützten Format gespeichert.
SYSTEM	Beendet BASIC. Schließt alle Dateien ab und kehrt zu DOS zurück.
TRON, TROFF	Schaltet die Programmablaufverfolgung ein oder aus.

Anweisungen

Dieser Abschnitt enthält alle BASIC-Anweisungen alphabetisch geordnet in zwei Kategorien: Ein-/Ausgabeanweisungen und Nicht-Ein-/Ausgabeanweisungen. In der Liste wird angegeben, was jede Anweisung tut, und ihre Syntax wird gezeigt. Für komplexe Anweisungen ist die gezeigte Syntax nicht immer vollständig. Detaillierte Informationen über jede Anweisung stehen im alphabetischen Teil dieses Kapitels.

Im vorherigen Abschnitt befindet sich eine Liste aller BASIC-Befehle.

Nicht-Ein-/Ausgabeanweisungen

Anweisung **Aktion**

CALL numerische variable(variablenliste)
Ruft ein Programm in Maschinensprache auf.

CHAIN dateiangabe
Ruft ein Programm auf und übergibt an dieses Programm Variablen. Durch andere Auswahlen ist es erlaubt, mit Überlagerungen zu arbeiten, mit der Ausführung des Programms mit einer anderen als der ersten Zeile zu beginnen, alle Variablen zu übergeben oder eine Überlagerung zu löschen.

Anweisung	Aktion
COM(n) ON/OFF/STOP	Aktiviert und inaktiviert die Unterbrechung für Datenfernverarbeitungstätigkeiten.
COMMON variablenliste	Übergibt Variablen an das durch CHAIN aufgerufene Programm.
DATE$ =x$	Setzt das Datum.
DEF FNname(argumentenliste)=ausdruck	Definiert eine numerische oder Zeichenkettenfunktion.
DEFtyp buchstabenbereich	Definiert die Standardannahmen für Variablentypen, wobei *typ* INT, SNG, DBL oder STR ist.
DEF SEG=adresse	Definiert das laufende Segment des Hauptspeichers.
DEF USRn=relativzeiger	Definiert die Startadresse für ein in Maschinensprache geschriebenes Unterprogramm n.
DIM liste der indizierten variablen	Definiert die maximalen Indexwerte für Bereiche und ordnet den Bereichen Platz zu.

Anweisung	Aktion
END	Stoppt ein Programm, schließt alle Dateien ab und kehrt zur Befehlsebene zurück.
ERASE bereichsnamen	Löscht Bereiche aus einem Programm.
ERROR n	Simuliert Fehlernummer n.
FOR variable=x TO y STEP z	Wiederholt die Ausführung von Programmzeilen. Die Anweisung NEXT schließt die Schleife ab.
GOSUB zeile	Ruft ein Unterprogramm durch Verzweigung zu einer angegeben Zeile auf. Durch die Anweisung RETURN wird aus dem Unterprogramm zurückverzweigt.
GOTO zeile	Vezweigt zu einer angegeben Zeile.
IF ausdruck THEN angabe ELSE angabe	Führt die Anweisung(en) in der THEN-Angabe aus, wenn der Ausdruck wahr ist (nicht Null). Im anderen Fall wird die ELSE-Angabe ausgeführt oder es wird zur nächsten Zeile verzweigt.
KEY ON/OFF/LIST	Zeigt den Inhalt von Funktionstasten an oder schaltet den Bildschirm aus.

Anweisung	Aktion
KEY n, x$	Setzt die Funktionstaste n auf den Wert der Zeichenkette x$.
KEY(n) ON/OFF/STOP	Aktiviert/inaktiviert eine Unterbrechung durch die Funktionstasten oder Positionsanzeigersteuertasten.
LET variable=ausdruck	Ordnet den Wert des Ausdrucks der Variablen zu.
MID$(v$,n,m)=y$	Ersetzt einen Teil der Variablen v$ durch die Zeichenkette y$, beginnend mit der Stelle n, m Zeichen ersetzend.
MOTOR status	Schaltet den Kassettenmotor ein, wenn der Status nicht Null ist und aus, wenn der Status = Null ist.
NEXT variable	Schließt eine FOR...NEXT-Schleife ab (siehe FOR).
ON COM(n) GOSUB zeile	Aktiviert eine Unterbrechungsroutine für Datenfernverarbeitungsaktivität.
ON ERROR GOTO zeile	Aktiviert eine Unterbrechungsroutine, beginnend mit der angegeben Zeile.

Anweisung	Aktion
ON n GOSUB zeile liste	Verzweigt zu einem durch n angegeben Unterprogramm.
ON n GOTO zeile liste	Verzweigt zu einer durch n angegeben Anweisung.
ON KEY(n) GOSUB zeile	Aktiviert ein Unterbrechungsunterprogramm für die angegebene Funktionstaste oder die Positionsanzeigersteuertaste.
ON PEN GOSUB zeile	Aktiviert die Unterbrechungsroutine für den Lichtstift.
ON STRIG(n) GOSUB zeile	Aktiviert die Unterbrechungsroutine für den Spielpultknopf.
OPTION BASE n	Gibt den kleinsten Wert für Bereichsindizes an.
PEN ON/OFF/STOP	Aktiviert/inaktiviert die Lichtstiftfunktion.
POKE n,m	Setzt das Byte m im Hauptspeicher an die durch n angegebene Adresse.
RANDOMIZE n	Gibt dem Zufallszeilengenerator einen neuen Anfangswert

Anweisung	Aktion
REM bemerkung	Fügt in das Programm eine Bemerkung ein.
RESTORE zeile	Setzt den DATA-Zeiger zurück, so daß DATA-Anweisungen neu gelesen werden können.
RESUME zeile/NEXT/0	Verzweigt aus der Fehlerunterbrechungsroutine zurück.
RETURN zeile	Verzweigt aus einem Unterprogramm zurück.
STOP	Stoppt die Programmausführung, zeigt eine Unterbrechungsnachricht an und verzweigt zur Befehlsebene zurück.
STRIG ON/OFF	Aktiviert/inaktiviert die Funktion des Spielpultknopfes.

Anweisung	Aktion
STRIG(n) ON/OFF/STOP	Aktiviert/inaktiviert eine Unterbrechung durch den Spielpultknopf.
SWAP variable1,variable2	Tauscht die Werte zweier Variablen aus.
TIME$ = x$	Setzt die Zeit.
WAIT anschluß,n,m	Verläßt die Programmausführung, bis ein angegebener Anschluß eine angegebene Bit-Schablone erzeugt.
WEND	Schließt eine WHILE...WEND-Schleife ab (siehe WHILE).
WHILE ausdruck	Beginnt eine Schleife, die so lange ausgeführt wird, wie der Ausdruck wahr ist.

Ein-/Ausgabeanweisungen

Anweisung	Aktion
BEEP	Der Lautsprecher erzeugt einen kurzen Ton.
CIRCLE (x,y),r	Zeichnet einen Kreis mit dem Mittelpunkt (x,y) und dem Radius r. Durch andere Auswahlen kann man angeben, daß nur ein Teil eines Kreises gezeichnet wird, oder daß das Bildverhältnis geändert wird, um eine Ellipse zu zeichnen.
CLOSE #f	Schließt eine Datei ab.
CLS	Löscht den Bildschirm.
COLOR vordergrund,hintergrund,rand	Setzt im Textmodus die Farben für Vordergrund, Hintergrund und den Bildschirmrand.
COLOR hintergrund,palette	Setzt im grafischen Modus die Hintergrundfarbe und die Palette der Vordergrundfarben.
DATA konstantenliste	Erstellt eine Datentabelle, die von READ-Anweisungen benutzt wird.

Anweisung	Aktion
DRAW zeichenkette	Zeichnet eine durch eine Zeichenkette angegebene Figur.
FIELD #f,breit AS zeichenkettenvariable...	Definiert Felder in einem Puffer für eine Datei mit wahlfreiem Zugriff.
GET #f,nummer	Liest einen Satz aus einer Datei für wahlfreien Zugriff.
GET (x1,y1)-(x2,y2),bereichsname	Liest grafische Informationen vom Bildschirm.
INPUT "anfrage";variablenliste	Liest Daten von der Tastatur.
INPUT #f,variablenliste	Liest Daten aus der Datei f.
LINE (x1,y1)-(x2,y2)	Zeichnet auf dem Bildschirm eine Linie. Andere Parameter erlauben es, ein Rechteck auszufüllen.
LINE INPUT"anfrage";zeichenkettenvariable	Liest eine ganze Zeile von der Tastatur und ignoriert dabei Kommas und andere Trennzeichen.
LINE INPUT #f,zeichenkettenvariable	Liest eine ganze Zeile aus einer Datei.

Anweisung	Aktion
LOCATE zeile,spalte	Positioniert den Positionsanzeiger. Andere Parameter erlauben es, die Größe des Positionsanzeigers zu definieren und ob er sichtbar oder unsichtbar sein soll.
LPRINT ausdrucksliste	Gibt Daten auf dem Drucker aus.
LPRINT USING v$;ausdrucksliste	Gibt Daten auf dem Drucker mit Hilfe des durch v$ angegebenen Formats aus.
LSET zeichenkettenvariable=x$	Legt eine Zeichenkette in einem Feld linksbündig an.
OPEN dateiangabe FOR modus AS #f	Eröffnet die Datei für den angegebenen Modus. Eine andere Auswahl setzt die Satzlänge für Dateien mit direktem Zugriff.
OPEN modus,#f,dateiangabe,satzlänge	Alternative Form des vorhergehenden OPEN.
OPEN "COMn:auswahlen" AS #f	Öffnet eine Datei für Datenfernverarbeitung.

Anweisung	Aktion
OUT n,m	Gibt das Byte m am Maschinenanschluß n aus.
PAINT (x,y),malen,grenzen	Füllt einen durch Grenzen definierten Bereich auf dem Bildschirm mit einer Farbe aus.
PLAY zeichenkette	Spielt die in einer Zeichenkette angegebene Musik.
PRINT ausdrucksliste	Zeigt Daten auf dem Bildschirm an.
PRINT USING v$,ausdrucksliste	Zeigt Daten mit Hilfe des durch v$ angegebenen Formats an.
PRINT #f,ausdrucksliste	Schreibt die Liste der Ausdrücke in die Datei f.
PRINT #f,USING v$;ausdrucksliste	Schreibt Daten mit Hilfe des durch v$ angegebenen Formats in die Datei f.
PRESET (x,y)	Zeichnet einen Punkt auf dem Bildschirm in der Hintergrundfarbe. Siehe PSET.

Anweisung	Aktion
PSET (x,y),farbe	Zeichnet einen Punkt auf dem Bildschirm in der Vordergrundfarbe, wenn die Farbe nicht angegeben ist.
PUT #f,nummer	Schreibt Daten aus einem Puffer für eine Datei mit wahlfreiem Zugriff in die Datei.
PUT (x,y),bereich,aktion	Schreibt grafische Informationen auf den Bildschirm.
READ variablenliste	Liest Informationen aus der durch DATA-Anweisungen erstellten Datentabelle.
RSET zeichenkettenvariable=x$	Legt eine Zeichenkette in einem Feld rechtsbündig an. Siehe LSET.
SCREEN modus,farbe,aktive seite,visuelle seite	Setzt den Bilschirmmodus, die Farbe an oder aus, die aktive Seite und die angezeigte Seite.

Anweisung	Aktion
SOUND frequenz,dauer	Erzeugt einen Ton durch den Lautsprecher.
WIDTH größe	Setzt die Bildschirmbreite. Andere Auswahlen erlauben es, die Breite eines Druckers oder einer Datenfernverarbeitungsdatei zu setzen.
WRITE ausdrucksliste	Gibt Daten auf dem Bildschirm aus.
WRITE #f,ausdrucksliste	Gibt Daten auf eine Datei aus.

Funktionen und Variablen

Die in BASIC verfügbaren eingebauten Funktionen und Variablen sind hier aufgelistet, gruppiert in zwei allgemeine Kategorien: numerische Funktionen oder solche, die ein numerisches Ergebnis übergeben, und Zeichenkettenfunktionen oder solche, die eine Zeichenkette als Ergebnis übergeben.

Jede Kategorie ist weiter unterteilt, gemäß der Benutzung der Funktionen. Die numerischen Funktionen sind unterteilt in allgemeine arithmetische Funktionen, in zeichenkettenbezogene Funktionen, die mit Zeichenketten arbeiten und in Ein-/Ausgabe- und verschiedene Funktionen. Die Zeichenkettenfunktionen sind in allgemeine Zeichenkettenfunktionen und Ein-/Ausgabe- und verschiedene Zeichenkettenfunktionen aufgeteilt.

> **Hinweis:** Numerische Funktionen übergeben nur ganzzahlige Werte oder Werte einfacher Genauigkeit, außer es ist etwas anderes angegeben.

Numerische Funktionen (Übergeben numerischen Wert)

ARITHMETIK

Funktion	Ergebnis
ABS(x)	Übergibt den absoluten Wert von x.
ATN(x)	Übergibt den Arcustangens (im Bogenmaß) von x.
CDBL(x)	Wandelt x in eine Zahl doppelter Genauigkeit um.
CINT(x)	Wandelt x durch Runden in eine ganze Zahl um.
COS(x)	Übergibt den Cosinus eines Winkels x (x im Bogenmaß).
CSNG(x)	Wandelt x in eine Zahl einfacher Genauigkeit um.
EXP(x)	Errechnet e^x.
FIX(x)	Bricht x zu einer ganzen Zahl ab.
INT(x)	Übergibt die größte ganze Zahl kleiner als oder gleich x.
LOG(x)	Übergibt den natürlichen Logarithmus von x.
RND(x)	Übergibt eine Zufallszahl.

Funktion	Ergebnis
SIN(x)	Übergibt den Sinus des Winkels x (x im Bogenmaß).
SGN(x)	Übergibt das Vorzeichen von x.
SQR(x)	Übergibt die Quadratwurzel von x.
TAN(x)	Übergibt den Tangens des Winkels x (x im Bogenmaß).

Informationen, wie mathematische Funktionen zu berechnen sind, die in dieser Liste nicht aufgeführt sind, sind in "Anhang E. Mathematische Funktionen" enthalten.

ZEICHENKETTENBEZOGEN

Funktion	Ergebnis
ASC(x$)	Übergibt den ASCII-Code für das erste Zeichen in x$.
CVI(x$), CVS(x$), CVD(x$)	Wandelt x$ in eine Zahl der angegebenen Genauigkeit um.
INSTR(n,x$,y$)	Übergibt die Position des ersten Auftretens von y$ in x$, beginnend mit Position n.
LEN(x$)	Übergibt die Länge von x$.
VAL(x$)	Übergibt den numerischen Wert von x$.

EIN-/AUSGABE und VERSCHIEDENES

Funktion	Ergebnis
CSRLIN	Übergibt die senkrechte Zeilenposition des Positionsanzeigers.
EOF(f)	Zeigt die Dateiendebedingung der Datei f an.
ERL	Übergibt die Zeilennummer, wo der letzte Fehler auftrat (siehe ERR).
ERR	Übergibt die Fehlercodenummer des letzten Fehlers.
FRE(x$)	Übergibt die Summe des freien Platzes im Hauptspeicher, der gerade von BASIC nicht benutzt wird.
INP(n)	Liest ein Byte vom Anschluß n.

Funktion	Ergebnis
LOC(f)	Übergibt den "Platz" der Datei f: • Nächste Satznummer einer Datei für wahlfreien Zugriff. • Anzahl der gelesenen oder geschriebenen Sektoren einer sequentiellen Datei. • Anzahl der Zeichen im Eingabepuffer für Datenfernverarbeitung.
LOF(f)	Übergibt die Länge der Datei f: • Anzahl der Bytes (in Mehrfachem von 128) in einer sequentiellen Datei oder einer Datei für wahlfreien Zugriff. • Anzahl der freien Bytes in einem Eingabepuffer für Datenfernverarbeitung.
LPOS(n)	Übergibt die Druckkopfposition des Druckers.
PEEK(n)	Liest das Byte in Hauptspeicheradresse n.
PEN(n)	Liest den Lichtstift.

Funktion	Ergebnis
POINT(x,y)	Übergibt die Farbe des Punktes (x,y) (grafischer Modus).
POS(n)	Übergibt die Spaltenposition des Positionsanzeigers.
SCREEN(zeile,spalte,z)	Übergibt das Zeichen oder die Farbe an Position (zeile,spalte).
STICK(n)	Übergibt die Koordinaten eines Spielpults.
STRIG(n)	Übergibt den Status eines Spielpultknopfes.
USRn(x)	Ruft ein Unterprogramm in Maschinensprache mit dem Argument x auf.
VARPTR(variable)	Übergibt die Adresse einer Variablen im Hauptspeicher.
VARPTR(#f)	Übergibt die Adresse des Dateisteuerblocks für die Datei f.

Zeichenkettenfunktionen (Übergeben eines Zeichenkettenwertes)

ALLGEMEIN

Funktion	Ergebnis
CHR$(n)	Übergibt das Zeichen mit dem ASCII-Code n.
LEFT$(x$,n)	Übergibt die am weitesten links stehenden n Zeichen von x$.
MID$(x$,n,m)	Übergibt m Zeichen aus x$, beginnend an Stelle n.
RIGHT$(x$,n)	Übergibt die am weitesten rechts stehenden n Zeichen von x$.
SPACE$(n)	Übergibt eine Zeichenkette von n Leerstellen.
STRING$(n,m)	Übergibt das Zeichen mit dem ASCII-Wert m, n-mal wiederholt.
STRING$(n,x$)	Übergibt das erste Zeichen von x$, n-mal wiederholt.

EIN-/AUSGABE und VERSCHIEDENES

Funktion	Ergebnis
DATE$	Übergibt das Systemdatum.
HEX$(n)	Wandelt n in eine hexadezimale Zeichenkette um.

Funktion	Ergebnis
INKEY$	Liest ein Zeichen von der Tastatur.
INPUT$(n,#f)	Liest n Zeichen aus der Datei f.
MKI$(x), MKS$(x), MKD$(x)	Wandelt x in der angegebenen Genauigkeit in eine Zeichenkette der richtigen Länge um.
OCT$(n)	Wandelt n in eine oktale Zeichenkette um.
SPC(n)	Druckt in einer PRINT- oder LPRINT-Anweisung n Leerstellen.
STR$(x)	Wandelt x in einen Zeichenkettenwert um.
TAB(n)	Geht in einer PRINT- oder LPRINT-Anweisung zur Position n.
TIME$	Übergibt die Systemuhrzeit.
VARPTR$(v)	Übergibt eine Drei-Byte-Zeichenkette, die den Typ der Variablen und die Adresse der Variablen im Hauptspeicher enthält.

ABS Funktion

Zweck: Übergibt den absoluten Wert des Ausdrucks *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = ABS (*x*)

Bemerkung: *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.

ASC Funktion

Zweck: Übergibt den ASCII-Code für das erste Zeichen der Zeichenkette *x$*.

Version:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *x$* ist ein Zeichenkettenausdruck.

Das Ergebnis der Funktion ASC ist ein numerischer Wert, der den ASCII-Code für das erste Zeichen der Zeichenkette *x$* repräsentiert (siehe Anhang G. "ASCII-Zeichencodes" für ASCII-Codes). Enthält *x$* keinen Wert, so wird die Fehlermeldung "Illegal function call" (Ungültiger Funktionsaufruf) übergeben.

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 für das große T den Wert 84 hat. Print ASC("Test") gibt dasselbe Ergebnis.

ATN
Funktion

Zweck: Übergibt den Arcustangens von *x*.

Version:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = ATN(*x*)

Bemerkungen: *x* ist ein numerischer Ausdruck jedes numerischen Typs. Die Errechnung des ATN wird jedoch immer in einfacher Genauigkeit ausgeführt.

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.

ATN Funktion

Beispiel:

```
Ok
PRINT ATN(3)
 1.249Ø46
Ok

1Ø PI=3.141593
2Ø BOGENMASS=ATN(1)
3Ø GRAD=BOGENMASS*18Ø/PI
4Ø 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.

AUTO Befehl

Zweck: Automatische Zeilennummergenerierung, jedesmal nachdem die Eingabetaste gedrückt wurde.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	

Format: AUTO [*nummer*] [, [*Schrittweite*]]

Bemerkungen: *Nummer* Nummer, die für den Beginn der Zeilennumerierung benutzt werden soll. Ein Punkt (.) kann anstelle der Zeilennummer benutzt werden, um die gerade einzugebende Zeile anzuzeigen.

Schrittweite:
Der jeder Zeilennummer hinzugefügte Wert, um die nächstfolgende Zeilennummer zu erhalten.

AUTO Befehl

Die Zeilennumerierung beginnt mit *Nummer* und wird für jede folgende Zeilennummer mit der *Schrittweite* erhöht. Werden beide Werte weggelassen, ist die Standardannahme 10,10. Folgt der *Nummer* ein Komma (,), aber die *Schrittweite* ist weggelassen, wird die letzte für 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 nötig, jede Zeilennummer einzugeben.

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

Drückt man jedoch direkt nach dem * die Eingabetaste, wird die bestehende Zeile nicht ersetzt und AUTO erzeugt die nächste Zeilennummer.

AUTO Befehl

AUTO wird durch Betätigen der Taste Ctrl und der Taste Break beendet. Die Zeile, bei der diese Tasten betätigt wurden, wird nicht gespeichert. Danach kehrt BASIC in die Befehlsebene zurück.

Hinweis: Im AUTO-Modus können nur Änderungen der Eingabezeile vorgenommen werden. Sollen andere Zeilen des Bildschirms geändert werden, muß man vorher AUTO durch Betätigen der Tasten Ctrl-Break verlassen.

Beispiel:

```
AUTO
```

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

```
AUTO 1ØØ,5Ø
```

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

```
AUTO 5ØØ,
```

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

```
AUTO ,2Ø
```

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

BEEP
Anweisung

Zweck: Der Lautsprecher erzeugt einen Ton.

Version:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: BEEP

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

Beispiel:

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

In diesem Beispiel prüft das Programm, ob sich X außerhalb eines gegebenen Bereichs befindet. Ist dies der Fall, ertönt der Lautsprecher.

BLOAD Befehl

Zweck: Lädt eine Datei mit einem Hauptspeicherabbild in den Hauptspeicher.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: BLOAD *Dateiangabe* [,*Relativzeiger*]

Bemerkungen: *Dateiangabe*

Zeichenkettenausdruck, wie in "Namensgebung für Dateien" in Kapitel 3 beschrieben. Entspricht er 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 ausgeführt, wird die durch einen Namen angegebene Datei, beginnend mit dem angegebenen Platz, in den Hauptspeicher geladen. Wird die Datei von der Einheit CAS1: geladen, wird der Kassettenmotor automatisch eingeschaltet.

Wird Kassetten-BASIC benutzt und ist der Einheitenname weggelassen, wird CAS1: angenommen. CAS1: ist die einzige erlaubte Einheit für BLOAD im Kassetten-BASIC.

Wird mit dem Disketten- oder Erweiterten BASIC gearbeitet und der Einheitenname ist weggelassen, wird die DOS-Standarddisketteneinheit benutzt.

BLOAD Befehl

BLOAD und BSAVE sind für das Laden und Speichern von Programmen in Maschinensprache sehr nützlich (siehe Anweisung CALL in diesem Kapitel), jedoch sind BLOAD und BSAVE nicht nur auf Programme in Maschinensprache beschränkt. Jedes Segment kann mit Hilfe der Anweisung DEF SEG als Quelle oder Ziel dieser Anweisungen angegeben werden. Dies ist ein nützlicher Weg, Bildschirmabbildungen zu speichern oder anzuzeigen, indem man sie aus dem Bildschirmpuffer speichert oder in den Bildschirmpuffer lädt:

Warnung:
BLOAD führt keine Adreßbereichsprüfung durch. Das heißt, es ist möglich, irgendwohin in den Hauptspeicher zu laden. Es darf nicht über den BASIC-Stapelbereich, über den BASIC-Variablenbereich oder das BASIC-Programm geladen werden.

Hinweise:

1. Wird der Befehl BLOAD im Direktmodus eingegeben, werden die Dateinamen auf dem Band am Bildschirm angezeigt, gefolgt von einem Punkt (.) und einem Buchstaben, der den Dateityp angibt. Danach folgt die Nachricht "Skipped" (Übersprungen), für die Dateien, die dem angegebenen Namen nicht entsprechen, und "Found" (Gefunden), sobald die angegebene Datei gefunden ist.

   Dateitypen und der dazugehörige Buchstabe:

   .B für BASIC-Programme im internen Format (mit dem Befehl SAVE erstellt)

   .P für geschüzte BASIC-Programme im internen Format (mit dem Befehl SAVE ,P erstellt)

   .A für BASIC-Programme im ASCII-Format (mit dem Befehl SAVE ,A erstellt)

   .M für Dateien im Hauptspeicherabbild (mit dem Befehl BSAVE erstellt)

   .D für Datendateien (erstellt durch OPEN, gefolgt von Ausgabeanweisungen)

BLOAD Befehl

Wird der Befehl BLOAD in einem BASIC-Programm ausgeführt, werden die übergangenen und gefundenen Dateinamen nicht am Bildschirm angezeigt.

2. Während BLOAD können zu jeder Zeit die Tasten Ctrl-Break betätigt werden. Dadurch beendet BASIC das Suchen und kehrt zwischen den Dateien oder nach einer bestimmten Zeitperiode in den Direktmodus zurück. Vorherige Hauptspeicherinhalte bleiben unverändert.

3. Ist CAS1: als Einheit angegeben und der Dateiname weggelassen, wird die nächste Hauptspeicherabbilddatei (.M) von dem Band geladen.

Beispiel:

```
1Ø 'Laden des Bildschirmpuffers
2Ø 'Mit SEG auf den Bildschirmpuffer zeigen
3Ø DEF SEG= $HB8ØØ
4Ø 'BILD in den Bildschirmpuffer laden
5Ø 'BLOAD "BILD",Ø
```

In diesem Beispiel wird der Bildschirmpuffer für den Farb-/Grafikbildschirmanschluß geladen, der sich an der absoluten Adresse hexadezimal B8000 befindet. Sollte der Bildschirmpuffer für den IBM Schwarz/-Weiß-Bildschirm und Paralleldruckeranschluß geladen werden, müßte die Zeile 30 in &HB000 (die aktuelle Adresse ist hexadezimal B0000) geändert 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 für BSAVE im nächsten Abschnitt zeigt, wie BILD gespeichert wurde.

BSAVE Befehl

Zweck: Damit werden Teile des Hauptspeichers in der angegebenen Einheit gespeichert.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: BSAVE *Dateiangabe, Relativzeiger, Länge*

Bemerkungen: *Dateiangabe:*
Zeichenkettenausdruck für eine Dateispezifikation; sie muß den Regeln der "Namensgebung für Dateien" in Kapitel 3 entspechen; andernfalls wird der Fehler "Bad File Name" (falscher Dateiname) angezeigt und das Speichern wird abgebrochen.

Relativzeiger:
Numerischer Ausdruck im Bereich 0 bis 65535. Das ist der Relativzeiger für das Segment, das in der letzten Anweisung DEF SEG deklariert wurde. Ab dieser Position wird gespeichert.

Länge: Numerischer Ausdruck im Bereich 0 bis 65535. Das ist die Länge des Hauptspeicherabbilds, das gespeichert werden soll.

Wird *Relativzeiger* oder *Länge* weggelassen, tritt ein "Syntax Error" (Syntaxfehler) auf, und das Speichern wird abgebrochen.

Fehlt der Einheitenname im Kassetten-BASIC, wird CAS1: angenommen. CAS1: ist die einzige erlaubte Einheit für BSAVE im Kassetten-BASIC. Wird im Disk- oder erweiterten BASIC der Einheitenname unterdrückt, wird die DOS-Standarddisketteneinheit benutzt.

BSAVE Befehl

Wird unter CAS1: gespeichert, wird der Motor der Bandkassette eingeschaltet, und die Datei mit dem Hauptspeicherabbild wird sofort auf das Band geschrieben.

BLOAD und BSAVE sind nützlich für das Laden und Speichern von Programmen in Maschinensprache (die mit Hilfe der Anweisung CALL aufgerufen werden können). Jedoch sind BLOAD und BSAVE nicht nur auf Programme in Maschinensprache beschränkt; 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.

BSAVE Befehl

Beispiel:

```
10 'Speichern des Puffers eines Farbbildschirms
15 'Mit SEG die Adresse des Bildschirmpuffers angeben
20 DEF SEG= &HB800
25 'Speichern des Puffers in die Datei BILD
30 BSAVE "BILD",0,&H4000
```

Wie schon unter "Befehl BLOAD" im vorherigen Abschnitt beschrieben, ist die Adresse des 16K-Bildschirmpuffers für den Farb-/Grafikbildschirmanschluß hexadezimal B8000. Die Adresse des 4K-Bildschirmpuffers für den IBM 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 Länge &H4000 geben an, daß der gesamte 16K-Bildschirmpuffer gespeichert werden soll.

CALL Anweisung

Zweck: Ruft ein Unterprogramm in Maschinensprache auf.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format: CALL *numerische Variable* [(*Variable* [,*Variable*]...)]

Bemerkungen: *Numerische Variable:* Name einer numerischen Variablen. Der Wert der Variablen beinhaltet die Startadresse des Unterprogramms 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 Maschinenspracheunterprogramm übergeben wird.

CALL Anweisung

Die Anweisung CALL ist ein Weg, eine Verbindung zwischen einem Unterprogramm in Maschinensprache und BASIC herzustellen. Der andere Weg ist die Funktion USR (siehe Anhang C. "Unterprogramme in Maschinensprache", der spezielle Betrachtungen über Unterprogramme in Maschinensprache enthält).

Beispiel:

```
1ØØ DEF SEG=&H8ØØØ
11Ø OZ=Ø
12Ø CALL OZ(A,B$,C)
```

Zeile 100 setzt das Segment auf die Adresse hexadezimal 80000. OZ wird auf Null gesetzt, so daß der Aufruf von OZ das Unterprogramm ab Adresse hexadezimal 80000 ausführt. Die Variablen A, B$ und C werden als Argumente an das Unterprogramm in Maschinensprache übergeben.

CDBL
Funktion

Zweck: Wandelt *x* in eine Zahl doppelter Genauigkeit um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *v* = CDBL(*x*)

Bemerkungen: *x* kann jeder numerische Ausdruck sein.

Regeln für die Umwandlung von einer numerischen Genauigkeit in eine andere sind in Kapitel 3, "Konvertierung von Zahlen einer Genauigkeit in eine andere in BASIC" erklärt. Die Funktionen CINT und CSNG wandeln Zahlen in ganze Zahlen und in Zahlen einfacher Genauigkeit um.

CDBL Funktion

Beispiel:

```
Ok
1Ø A = 454.67
2Ø PRINT A;CDBL(A)
RUN
 454.67  454.6699829101563
Ok
```

Der Wert von CDBL(A) ist nach dem Runden nur auf die zweite Dezimalstelle genau. Die andern Ziffern haben keine Bedeutung, weil A nur zwei Dezimalstellen Genauigkeit liefert.

CHAIN
Anweisung

Zweck: Überträgt die Steuerung an ein anderes Programm und übergibt Variablen aus dem laufenden Programm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	(**)

Format: CHAIN [MERGE] *Dateiangabe* [,[*Zeile*] . [,[ALL] [,DELETE *Bereich*]]]

Bemerkungen: *Dateiangabe* folgt den Regeln für Dateispezifikationen, wie in Kapitel 3, "Namensgebung für Dateien" beschrieben. Der Dateiname ist der Name des Programms, an das die Steuerung übergeben wird. Beispiel:

```
CHAIN "A:PROG1"
```

Zeile: Zeilennummer oder Ausdruck, der eine Zeilennummer in dem Programm berechnet, an das die Steuerung übergeben wird. Damit wird die Zeile angegeben, bei der in dem aufgerufenen Programm die Ausführung beginnen soll. Ist sie weggelassen, beginnt die Ausführung mit der ersten Zeile des aufgerufenen Programms. Beispiel:

```
CHAIN "A:PROG1",1ØØØ
```

Zeile (1000 in diesem Beispiel) wird mit dem Befehl RENUM nicht verändert. Wird PROG1 neu numeriert, müßte die in dem Beispiel angegebene Anweisung CHAIN geändert werden, um auf die neue Zeilennummer zu deuten.

ALL bedeutet, daß jede Variable des laufenden Programms an das aufgerufene übergeben werden soll. Wird die Angabe ALL weggelassen, muß in das aufgerufene Programm eine Anweisung COMMON eingefügt werden, um Variablen an das aufgerufene Programm zu übergeben. Siehe "Anweisung COMMON" in diesem Kapitel. Beispiel:

```
CHAIN "A:PROG1",1ØØØ,ALL
```

CHAIN
Anweisung

MERGE übergibt den Programmcode an das BASIC-Programm als Überlagerung. Das bedeutet, daß die Operation MERGE mit dem aufrufenden und dem aufgerufenen Programm durchführt wird. Das aufgerufene Programm muß eine Datei im ASCII-Code sein, falls es eingemischt werden soll. Beispiel:

```
CHAIN MERGE "A:OVRLAY",1000
```

Nachdem eine Überlagerung stattgefunden hat, möchte man sie gewöhnlich löschen, damit eine neue Überlagerung eingelesen werden kann. Dies erfolgt mit Hilfe der Auswahl DELETE, die wie der Befehl DELETE arbeitet. Wie beim Befehl DELETE müssen die Zeilennummern der ersten und letzten angegebenen Zeile existieren oder es wird der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf) angezeigt. Beispiel:

```
CHAIN MERGE "A:OVRLAY2",1000,DELETE 1000-5000
```

In diesem Beispiel werden die Zeilen 1000 bis 5000 des aufrufenden Programms gelöscht, bevor die Überlagerung (aufgerufenes Programm) geladen wird. Die Zeilennummern in *Bereich* werden von dem Befehl RENUM auch neu numeriert.

Hinweise:

1. Die Anweisung CHAIN läßt Dateien eröffnet.

2. Die Anweisung CHAIN mit der Angabe MERGE erhält den Status des letzten Setzens von OPTION BASE.

3. Wird die Angabe MERGE weggelassen, bleibt der Status des Setzens der letzten Anweisung OPTION BASE im aufgerufenen Programm nicht erhalten. CHAIN ohne MERGE erhält auch nicht die Variablen oder benutzerdefinierten Funktionen für 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 müssen.

CHR$ Funktion

Zweck: Wandelt einen ASCII-Code in sein entsprechendes Zeichen um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = CHR$(*n*)

Bemerkungen: *n* muß im Bereich 0 bis 255 liegen.

Die Funktion CHR$ übergibt die Zeichenkette, bestehend aus einem Zeichen für den ASCII-Code *n* (ASCII-Codes sind im Anhang G. ASCII-Zeichencodes aufgelistet). CHR$ wird normalerweise dazu benutzt, ein Sonderzeichen auf den Bildschirm oder Drucker zu übertragen. Das Zeichen BEL, durch das der Lautsprecher ertönt, kann z.B. als Anfang einer Fehlermeldung mit CHR$(7) (statt BEEP) eingefügt werden. Die Funktion ASC zeigt, wie ein Zeichen zurück in seinen ASCII-Code umgewandelt werden kann.

Beispiel:

```
Ok
PRINT CHR$(66)
B
Ok
```

Im nächsten Beispiel wird der Funktionstaste F1 die Zeichenkette "AUTO", verbunden mit dem Zeichen für Eingabe, zugeordnet. Dies ist eine gute Lösung für das Setzen der Funktionstasten. Die Funktion der Eingabetaste erfolgt automatisch, wenn die Funktionstaste gedrückt wird.

```
Ok
KEY 1,"AUTO"+CHR$(13)
Ok
```

Das folgende Beispiel ist ein Programm, das alle anzeigbaren Zeichen zusammen mit ihren ASCII-Codes auf einem Bildschirm mit 80 Zeilen zeigt. Es kann mit dem IBM Schwarz/Weiß-Bildschirm und Paralleldruckeranschluß oder dem Farb-/Grafikbildschirmanschluß benutzt werden.

```
10 CLS
20 FOR I=1 TO 255
30 ' Ignorieren nichtanzeigbares Zeichen
40 IF (I>6 AND I<14) OR (I>27 AND I<32) THEN 100
50 COLOR 0,7 ' schwarz auf weiß
60 PRINT USING "###"; I ; ' dreistelliger ASCII-Code
70 COLOR 7,0 ' weiß auf schwarz
80 PRINT " "; CHR$(I); " ";
90 IF POS(0)>75 THEN PRINT ' zur nächsten Zeile weitergehen
100 NEXT I
```

CINT
Funktion

Zweck: Wandelt *x* in eine ganze Zahl um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = CINT(*x*)

Bemerkungen: *x* kann jeder numerische Ausdruck sein. Liegt *x* nicht im Bereich -32768 bis 32767, wird der Fehler "Overflow" (Überlauf) angezeigt.

x wird in eine ganze Zahl umgewandelt, indem die Zahlen hinter dem Dezimalpunkt gerundet werden.

Auch die Funktionen FIX und INT übergeben 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
```

Es sollte beachtet werden, wie in den beiden Beispielen gerundet wird.

CIRCLE
Anweisung

Zweck: Zeichnen einer Ellipse auf den Bildschirm mit dem Mittelpunkt (*x,y*) und dem Radius *r*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	***

Nur im Grafikmodus.

Format: CIRCLE (*x,y*),*r* [,*Farbe* [,*Start,End* [,*Bild*]]]

Bemerkungen:

(*x,y*) sind die Koordinaten des Mittelpunkts der Ellipse. Die Koordinaten können entweder in absoluter oder relativer Form angegeben werden. Siehe Spezifizieren von Koordinaten unter Grafikmodus in Kapitel 3.

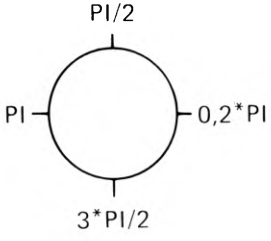
r ist der Radius (Hauptachse) der Ellipse in Punkten.

Farbe ist eine Zahl, die die Farbe der Ellipse im Bereich 0 bis 3 angibt. *Farbe* wählt die Farbe aus der laufenden Palette, die in der Anweisung COLOR definiert ist. 0 ist die Hintergrundfarbe. Die Standardannahme ist die Vordergrundfarbe, Farbe Nummer 3. In der hohen Auflösung zeigt 0 (Null) für *Farbe* schwarz an und die Standardannahme 1 (eins) weiß.

Start, End sind Winkel im Bogenmaß und können 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 0 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 (dabei muß beachtet werden, daß dies nicht das gleiche ist, wie die Addition von 2*PI). Der Anfangswinkel kann kleiner oder größer sein als der Endwinkel. Beispiel:

```
1Ø PI=3.141593
2Ø SCREEN 1
3Ø CIRCLE (16Ø,1ØØ),6Ø,,-PI,-PI/2
```

Dieses Beispiel zeichnet den Teil eines Kreises, ähnlich dem folgenden:

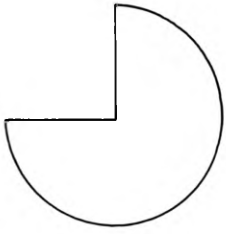
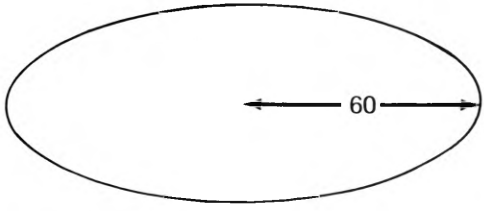


Bild hat Einfluß auf das Verhältnis des x-Radius zum y-Radius. Die Standardannahme für *Bild* ist 5/6 in der mittleren Auflösung und 5/12 in der hohen Auflösung. Diese Werte geben visuell einen Kreis in der Annahme, daß das Bildverhältnis des Standardbildschirms 4/3 ist.

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* größer als 1, so ist *r* der y-Radius. Beispiel:

```
1Ø SCREEN 1
2Ø CIRCLE (16Ø,1ØØ),60,,,,5/18
```

Dieses Beispiel zeichnet eine Ellipse wie folgt:



In vielen Fällen gibt *Bild* gleich 1 (eins) einen besser aussehenden Kreis in der mittleren Auflösung. 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.

Beispiel: Das folgende Beispiel zeichnet ein Gesicht.

```
1Ø PI=3.141593
2Ø SCREEN 1 ' mittlere Auflösung Grafik
3Ø COLOR Ø,1 ' schwarzer Hintergrund, Palette 1
4Ø 'zwei Kreise in Farbe 1 (kobaltblau)
5Ø CIRCLE (12Ø,5Ø),1Ø,1
6Ø CIRCLE (2ØØ,5Ø),1Ø,1
7Ø 'zwei horizontale Ellipsen
8Ø CIRCLE (12Ø,5Ø),3Ø,,,,5/18
9Ø CIRCLE (2ØØ,5Ø),3Ø,,,,5/18
1ØØ 'Bogen in Farbe 2 (violett)
11Ø CIRCLE (16Ø,Ø),15Ø,2, 1.3*PI, 1.7*PI
12Ø 'Bogen, eine Seite mit dem Mittelpunkt verbunden
13Ø CIRCLE (16Ø,52),5Ø,, 1.4*PI, -1.6*PI
```

CLEAR Befehl

Zweck: Setzt alle numerischen Variablen auf Null und löscht den Inhalt aller Zeichenkettenvariablen. Mit den Auswahlen kann die Größe des Arbeitsspeichers und die Größe eines Stapelbereichs definiert werden.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: CLEAR [,[*n*] [,*m*]]

Bemerkungen: *n* ist ein Bytezähler, der - falls angegeben - die maximale Anzahl der Bytes für einen BASIC-Arbeitsbereich definiert (in dem das Programm und die Daten zusammen mit einem Arbeitsbereich für den Interpretierer gespeichert sind). *n* wird dann eingefügt, wenn Hauptspeicherplatz für Programme in Maschinensprache reserviert werden soll.

m setzt die Größe des Stapelbereichs für BASIC. Der Standardwert ist 512 Bytes oder ein Achtel des verfügbaren Hauptspeichers (der kleinere der beiden Werte). *m* wird eingefügt, 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 löscht den gesamten für Daten benutzten Hauptspeicher, ohne das gerade im Hauptspeicher befindliche Programm zu löschen. 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 gelöscht. (Dies beinhaltet DEF FN, DEF SEG, DEF USR, DEFINT, DEFDBL, DEFSNG und DEFSTR.)

Die Ausführung des Befehls CLEAR schaltet jeden Ton ab und setzt auf Musikvordergrund zurück. Auch PEN und STRIG werden auf OFF (AUS) gesetzt.

CLEAR Befehl

Die Anweisung ERASE kann nützlich sein, um einigen Hauptspeicherplatz freizumachen, ohne damit alle Daten des Programms zu löschen. Es werden nur angegebene Bereiche aus dem Arbeitsbereich gelöscht. Siehe "Anweisung ERASE" in diesem Kapitel.

Beispiel: Im nächsten Beispiel werden alle Daten aus dem Hauptspeicher gelöscht (ohne das Programm zu löschen):

```
CLEAR
```

Im nächsten Beispiel werden alle Daten gelöscht und der maximale Arbeitsbereich auf 32K Bytes gesetzt:

```
CLEAR ,32768
```

Im nächsten Beispiel werden die Daten gelöscht und die Größe des Stapelbereichs auf 2000 Bytes gesetzt:

```
CLEAR ,,2000
```

Im letzten Beispiel werden alle Daten gelöscht, der maximale Arbeitsbereich für BASIC wird auf 32K Bytes und der Stapelbereich auf 2000 Bytes gesetzt:

```
CLEAR ,32768,2000
```

CLOSE Anweisung

Zweck: Schließt die Ein-/Ausgabe für eine Einheit oder Datei ab.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: 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 ausgeführt wird. Nachfolgende Ein-/Ausgabeoperationen, die diese Dateinummer ansprechen, sind nicht mehr gültig. Die Datei oder Einheit kann unter derselben oder einer anderen Dateinummer erneut eröffnet werden oder die Dateinummer kann erneut benutzt werden, um eine andere Einheit oder Datei zu eröffnen.

Durch das Abschließen einer Datei oder einer Einheit, die für sequentielle Ausgabe eröffnet wurde, wird der Inhalt des letzten Puffers an die Datei oder Einheit übergeben.

CLOSE
Anweisung

Durch CLOSE ohne angegebene Dateinummern werden alle Einheiten und Dateien, die eröffnet sind, abgeschlossen.

Durch die Ausführung von EN, NEW, RESET, SYSTEM oder RUN ohne die Angabe **R** werden alle eröffneten Dateien und Einheiten automatisch abgeschlossen. STOP schließt keine Datei oder Einheit ab.

Siehe "Anweisung OPEN" in diesem Kapitel für Informationen über das Eröffnen von Dateien.

Beispiel:

```
1ØØ CLOSE 1,#2,#3
```

Dadurch werden die Dateien und Einheiten, denen die Dateinummern 1, 2 oder 3 zugeordnet sind, abgeschlossen.

```
2ØØ CLOSE
```

Alle eröffneten Einheiten und Dateien werden abgeschlossen.

CLS
Anweisung

Zweck: Löscht den Bildschirm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: CLS

Bemerkungen: Im Textmodus wird die aktive Seite (siehe "Anweisung SCREEN" in diesem Kapitel) auf die Hintergrundfarbe gelöscht (siehe "Anweisung COLOR" in diesem Kapitel).

Im Grafikmodus (mittlere oder hohe Auflösung) wird der gesamte Bildschirmpuffer auf die Hintergrundfarbe gelöscht.

Durch die Anweisung CLS wird der Positionsanzeiger auf seine Heimposition gebracht. Im Textmodus bedeutet das, daß der Positionsanzeiger in die obere linke Ecke des Bildschirms gebracht wird. Im grafischen Modus bedeutet dies, daß der "letzte angesprochene Punkt" für neue Grafikanweisungen der Mittelpunkt des Bildschirms ist ((160,100) für mittlere Auflösung, (320,100) für hohe Auflösung).

CLS
Anweisung

Wird der Bildschirmmodus oder die Breite durch die Anweisungen SCREEN oder WIDTH geändert, wird der Bildschirm auch gelöscht. Der Bildschirm kann auch durch Betätigen der Tasten Ctrl-Home gelöscht werden.

Beispiel:

```
10 COLOR 10,1
20 CLS
```

Mit Farb-/Grafikbildschirmanschluß wird der Bildschirm in diesem Beispiel auf blau gelöscht.

COLOR
Anweisung

Zweck: Setzt die Farben für den Vordergrund, Hintergrund und den Bildschirmrand. Die Ausdrücke sind unter "Textmodus" in Kapitel 3 erklärt.

Die Syntax der Anweisung COLOR hängt davon ab, ob sich das System im Textmodus oder Grafikmodus befindet, wie durch die Anweisung SCREEN gesetzt.

Im Textmodus kann folgendes gesetzt werden:

Vordergrund -	1 von 16 Farben. Zeichen blinken, falls gewünscht
Hintergrund -	1 von 8 Farben
Bildschirmrand -	1 von 16 Farben

Das Folgende kann in mittlerer Auflösung für 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.

COLOR
Anweisung

Die Anweisung COLOR im Textmodus

Versionen:

Kassette	Disk	Erweitert	Umwandlungprogramm
***	***	***	***

Nur Textmodus.

Format: 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 für die Hintergrundfarbe.

Bildschirmrand: Numerischer Ausdruck im Bereich 0 bis 15. Er gibt die Farbe für den Bildschirmrand an.

Mit dem Farb-/Grafikbildschirmanschluß sind die folgenden Farben für den Vordergrund erlaubt:

0 Schwarz	8 Grau
1 Blau	9 Hellblau
2 Grün	10 Hellgrün
3 Kobaltblau	11 Hellkobaltblau
4 Rot	12 Hellrot
5 Violett	13 Hellviolett
6 Braun	14 Gelb
7 Weiß	15 Hochintensives Weiß

Farbe und Intensität können an jedem Bildschirm variieren.

Man kann sich die Farben 8 bis 15 als helle oder hochintensive Werte der Farben 0 bis 7 vorstellen.

COLOR
Anweisung (Text)

Es ist möglich, die Zeichen blinken zu lassen, indem man den Vordergrund gleich 16 plus der Zahl der gewünschten Farbe setzt. Das bedeutet, ein Wert von 16 bis 31 erzeugt blinkende Zeichen.

Für den Hintergrund können nur die Farben 0 bis 7 gewählt werden.

Ist der IBM Schwarz/Weiß-Bildschirm- und Paralleldruckeranschluß vorhanden, können die folgenden Farben für den *Vordergrund* gewählt werden:

0 Schwarz
1 Unterstrichenes Zeichen mit weißem Vordergrund
2 bis 7 Weiß

Ähnlich wie beim Farb-/Grafikbildschirmanschluß kann hier durch Addition der Zahl 8 zur gewünschten Farbe die 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 gewünschten Farbe addiert. 16 ergibt schwarz blinkende Zeichen, 31 ergibt hochintensive weiß blinkende Zeichen.

Für den Hintergrund des IBM Schwarz/Weiß-Bildschirm- und Paralleldruckeranschlusses können die folgenden Werte gewählt werden:

0 bis 6 Schwarz
7 Weiß

Hinweis: Weiß (Farbe 7) als Hintergrundfarbe zeigt sich beim IBM 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 für Vordergrund- und Hintergrundfarben ergeben die Standardanzeige (weiß auf schwarz auf dem IBM Schwarz/Weiß-Bildschirm.

COLOR
Anweisung (Text)

Hinweise für jeden Anschluß:

1. Die Vordergrundfarbe kann gleich der Hintergrundfarbe sein. Durch diesen Effekt werden Zeichen unsichtbar gemacht. Das Ändern der Vordergrund- oder Hintergrundfarbe läßt die Zeichen wieder sichtbar werden.

2. Parameter können weggelassen werden. Für weggelassene Parameter wird der alte Wert angenommen.

3. Endet die Anweisung COLOR mit einem Komma (,), erhält man die Fehlernachricht "Missing Operand" (Fehlender Operand), aber die Farbe wird verändert. Beispiel:

   COLOR,7,

   ist ungültig.

4. Durch jeden eingegebenen Wert außerhalb des Bereichs 0 bis 255 erhält man den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Frühere Werte werden beibehalten.

Beispiel:

```
1Ø COLOR 14,1,Ø
```

Dadurch wird der Vordergrund auf gelb, der Hintergrund auf blau, und der Bildschirmrand auf schwarz gesetzt.

Das folgende Beispiel kann entweder mit dem Farb-/Grafikbildschirmanschluß oder dem IBM Schwarz/Weiß-Bildschirm- und Paralleldruckeranschluß benutzt werden:

```
10 PRINT "Eingabe";
20 COLOR 15,0 'Intensivanzeige nächstes Wort
30 PRINT "Paßwort";
40 COLOR 7    'Rückkehr zum Standard (weiß auf schwarz)
50 PRINT " hier: ";
60 COLOR 0    'unsichtbar (schwarz auf schwarz)
70 INPUT PASSWORD$
80 IF PASSWORD$="geheim" THEN 120
90 ' Blinken und intensive Fehlermeldung
100 COLOR 31: PRINT "Falsches Paßwort": COLOR 7
110 GOTO 10
120 COLOR 0,7   'Anzeigenumkehrung (schwarz auf weiß)
130 PRINT "Programm läuft weiter...";
140 COLOR 7,0  'Rückkehr zum Standard (weiß auf schwarz)
```

COLOR
Anweisung (Grafik)

Die Anweisung COLOR im Grafikmodus

Versionen:	Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***	***

Grafikmodus, nur mittlere Auflösung.

Format: COLOR [*Hintergrund*] [,[*Palette*]]

Bemerkungen: *Hintergrund*
Numerischer Ausdruck, der die Hintergrundfarbe angibt. Die für *Hintergrund* erlaubten Farben sind 0 bis 15, wie im Abschnitt "Die Anweisung COLOR im Textmodus" beschrieben.

Palette Numerischer Ausdruck, der die Palette der Farben auswählt.

Die für jede Palette ausgewählten Farben sind die folgenden:

Farbe	Palette 0	Palette 1
1	Grün	Kobaltblau
2	Rot	Violett
3	Braun	Weiß

Ist *Palette* eine gerade Zahl, wird Palette 0 ausgewählt. Dadurch werden die Farben Grün, Rot und Braun den Farbennummern 1, 2 und 3 zugeordnet. Palette 1 wird ausgewählt (Kobaltblau/Violett/Weiß), wenn *Palette* eine ungerade Zahl ist.

Die für den Hintergrund ausgewählte Farbe kann die gleiche sein, wie eine der Palettenfarben.

COLOR
Anweisung (Grafik)

Parameter der Anweisung COLOR können weggelassen werden. Für 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 für die Anzeige mit den Anweisungen PSET, PRESET, LINE, CIRCLE, PAINT und DRAW ausgewählt werden. Dies gilt nur für die mittlere Auflösung (gesetzt durch die Anweisung SCREEN 1). Wird COLOR für hohe Auflösung benutzt, erhält man den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

Jeder eingegebene Wert außerhalb des Bereichs 0 bis 255 erzeugt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Die früheren Werte werden beibehalten.

Biespiel:

```
5 SCREEN 1
1Ø COLOR 9,Ø
```

Setzt den Hintergrund auf hellblau und wählt die Palette 0 aus.

```
2Ø COLOR ,1
```

Der Hintergrund bleibt hellblau, und die Palette 1 wird ausgewählt.

COM(n)
Anweisung

Zweck: Aktiviert oder inaktiviert das Abfragen einer Unterbrechung durch Datenfernverarbeitungsaktivitäten mit dem angegebenen Datenfernverarbeitungsanschluß.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format:

COM(*n*) ON

COM(*n*) OFF

COM(*n*) STOP

Bemerkungen: *n:* Nummer des Datenfernverarbeitungsanschlusses (1 oder 2).

Die Anweisung COM(*n*) ON muß ausgeführt 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, prüft BASIC nach Ausführung von COM(*n*) ON nach jeder Ausführung einer neuen Anweisung, ob ein Zeichen über den Datenfernverarbeitungsanschluß gekommen ist.

Ist die Anweisung COM(*n*) OFF ausgeführt, findet keine Unterbrechung statt, und es wird keine Datenfernverarbeitungsaktivität bemerkt – auch wenn sie stattgefunden hat.

Wurde die Anweisung COM(*n*) STOP ausgeführt, kann keine Unterbrechung stattfinden, jedoch wird jede Datenfernverarbeitungsaktivität, die stattgefunden hatte, bemerkt, sobald die Anweisung COM(*n*) ON ausgeführt wurde, und eine sofortige Unterbrechung findet statt.

COMMON Anweisung

Zweck: Übergibt Variablen an ein aufgerufenes Programm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	(**)

Format: COMMON *Variable*[,*Variable*]...

Bemerkungen: *Variable* Name einer Variablen, die an das aufgerufene Programm übergeben werden soll. Bereichsvariablen werden definiert, indem sie an den Variablennamen "( )" angehängt werden.

Die Anweisung COMMON wird in Verbindung mit der Anweisung CHAIN benutzt. Die Anweisungen COMMON können 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 übergeben werden, benutzt man am besten CHAIN mit der Auswahl ALL und unterdrückt die Anweisung COMMON.

Die Bereiche, die übergeben werden sollen, müssen im aufgerufenen Programm nicht dimensioniert werden.

Beispiel:

```
100 COMMON A,BEE1,C,D(),G$
110 CHAIN "A:PROG3"
```

Dieses Programm ruft das Programm PROG3 auf der Diskette im Laufwerk A: auf und übergibt den Bereich D und die Variablen A, BEE1, C und G$.

CONT Befehl

Zweck: Nimmt die Programmausführung nach einer Unterbrechung wieder auf.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: CONT

Bemerkungen: Der Befehl CONT kann dazu benutzt werden, die Programmausführung wieder aufzunehmen, nachdem die Tasten Ctrl-Break betätigt wurden, die Anweisung STOP oder END ausgeführt wurde oder ein Fehler auftrat. Die Ausführung wird an dem Punkt wieder aufgenommen, wo die Unterbrechung auftrat. Geschah die Unterbrechung nach dem Warten auf eine Eingabe für eine Anweisung INPUT, wird mit der Neuanzeige der Anfrage fortgefahren.

CONT wird normalerweise in Verbindung mit STOP für das Testen benutzt. Wird die Ausführung angehalten, können die Werte von Variablen mit Hilfe von Direktmodusanweisungen geprüft oder geändert werden. Danach kann mit CONT die Ausführung erneut aufgesetzt, oder mit dem Direktmodusbefehl GOTO die Ausführung an einer bestimmten Zeilennummer wieder aufgesetzt werden.

CONT ist ungültig, falls das Programm während der Unterbrechung verändert wurde.

Beispiel: Im folgenden Beispiel wird eine lange 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 Betätigen der Tasten Ctrl-Break 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
```

COS
Funktion

Zweck: Errechnet und übergibt den Cosinuswert.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: $v = \text{COS}(x)$

Bemerkungen: x ist der Winkel, dessen Cosinus errechnet werden soll. Der Wert von x muß im Bogenmaß vorliegen. Um vom Grad in das Bogenmaß umzuwandeln, müssen die Grade mit PI/180 multipliziert werden, wobei PI = 3.141593 ist.

Die Berechnung des COS(x) wird in einfacher Genauigkeit durchgeführt.

COS
Funktion

Beispiel:

```
Ok
1Ø PI=3.141593
2Ø PRINT COS(PI)
3Ø GRAD=18Ø
4Ø BOGEN=GRAD*PI/18Ø
5Ø 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).

CSNG
Funktion

Zweck: Wandelt *x* in eine Zahl einfacher Genauigkeit um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *v* = CSNG(*x*)

Bemerkungen: *x* ist ein numerischer Ausdruck, der in eine Zahl einfacher Genauigkeit umgewandelt wird.

Die Regeln, die in Kapitel 3 "Konvertierung von Zahlen einer Genauigkeit in eine andere in BASIC" beschrieben sind, werden für diese Umwandlung benutzt.

Siehe auch die Funktionen CINT und CDBL für die Umwandlung von Zahlen in ganzzahlige Werte und Werte doppelter Genauigkeit.

Beispiel:

```
Ok
1Ø A# = 975.3421222#
2Ø 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#) übergeben.

CSRLIN
Variable

Zweck: Übergibt die vertikale Koordinate des Positionsanzeigers.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = CSRLIN

Bemerkungen: Die Variable CSRLIN übergibt die laufende Zeilenposition (Zeile) des Positionsanzeigers auf der aktiven Seite (die aktive Seite ist unter "Anweisung SCREEN in diesem Kapitel erklärt). Der übergebene Wert liegt im Bereich 1 bis 25.

Die Funktion POS übergibt die Spalte, an der der Positionsanzeiger steht. Siehe "Funktion POS" in diesem Kapitel.

Unter "Anweisung LOCATE" steht, wie der Positionsanzeiger an eine bestimmte Stelle gesetzt werden kann.

Beispiel:

```
10 Y = CSRLIN  'laufende Zeile aufnehmen
20 X = POS(0)  'laufende Spalte aufnehmen
29 'GUTEN TAG auf Zeile 24 schreiben
30 LOCATE 24,1: PRINT "GUTEN TAG"
40 LOCATE Y,X  'auf alte Position setzen
```

In diesem Beispiel werden die Koordinaten des Positionsanzeigers in den Variablen X und Y gespeichert. Danach wird der Positionsanzeiger auf die Zeile 24 gebracht, und die Wörter "GUTEN TAG" auf diese Zeile geschrieben. Dann wird der Positionsanzeiger zurück in seine alte Position gebracht.

CVI, CVS, CVD Funktionen

Zweck: Wandelt den Inhalt von Zeichenkettenvariablen in numerische Werte um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format:

v = CVI(*2-Byte-Zeichenkette*)

v = CVS(*4-Byte-Zeichenkette*)

v = CVD(*8-Byte-Zeichenkette*)

Bemerkungen: Numerische Werte, die aus einer Datei für wahlfreien Zugriff gelesen wurden, müssen 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.

CVI, CVS, CVD Funktionen

Die Funktionen CVI, CVS und CVD ändern *nicht* die Bytes der aktuellen Daten. Sie ändern nur den Weg, wie BASIC diese Bytes interpretiert.

Siehe auch "Funktionen MKI$, MKS$, MKD$" in diesem Kapitel und "Anhang B. BASIC-Diskettenein- und -ausgabe."

Beispiel:

```
7Ø FIELD #1,4 AS N$, 12 AS B$
8Ø GET #1
9Ø Y=CVS(N$)
```

In diesem Beispiel wird eine Datei mit wahlfreiem Zugriff (#1) benutzt, die Felder, wie in Zeile 70 definiert, enthält. 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.

DATA
Anweisung

Zweck: In dieser Anweisung werden numerische und Zeichenkonstanten gespeichert, die mit der Anweisung READ im Programm angesprochen werden können.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: DATA *Konstante*[,*Konstante*]...

Bemerkungen: *Konstante* kann eine numerische oder Zeichenkettenkonstante sein. In der Liste sind keine Ausdrücke erlaubt. Die numerische Konstante kann jedes Format haben - ganzzahlig, Festkomma, Gleitkomma, hexadezimal oder oktal. Zeichenkettenkonstanten in der Anweisung DATA müssen nicht in Anführungszeichen eingeschlossen sein, außer die Zeichenkette enthält Kommas (,), Doppelpunkte (:) oder signifikante führende oder nachstehende Leerstellen.

DATA
Anweisung

Anweisungen DATA sind nicht ausführbar und können überall 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, können als fortlaufende Liste von Angaben betrachtet werden, unabhängig davon, wie viele Angaben in einer Zeile stehen, 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 übereinstimmen; andernfalls 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 Beispiele unter "Anweisung READ" in diesem Kapitel.

DATE$
Variable und Anweisung

Zweck: Setzt und liest das Datum.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: Als Variable:

*v$* = DATE$

Als Anweisung:

DATE$ = *x$*

Bemerkungen: **Für die Variable (v$ = DATE$):**

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

DATE$
Variable und Anweisung

Für 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 für den Monat oder Tag nur eine Ziffer angegeben, wird davor eine Null (0) angenommen. Wird für das Jahr nur eine Ziffer angegeben, wird eine Null (0) angehängt, um daraus zwei Ziffern zu machen. Werden nur zwei Ziffern für das Jahr gegeben, wird das Jahr als 19*jj* angenommen.

DATE$
Variable und Anweisung

Beispiel:

```
Ok
1Ø DATE$= "8/29/82"
2Ø PRINT DATE$
RUN
Ø8-29-1982
Ok
```

In dem Beispiel wird das Datum auf den 29. August 1982 gesetzt. Wird das Datum mit Hilfe der Funktion DATE$ eingelesen, so wird vor den Monat eine Null (0) gesetzt, um daraus zwei Ziffern zu machen, und aus dem Jahr wurde 1982. Auch wurde Monat, Tag und Jahr durch Bindestriche (-) getrennt, obwohl Schrägstriche (/) eingegeben wurden.

DEF FN Anweisung

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

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

Bemerkungen:

Name ist ein gültiger 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 übergibt. Der Typ des *Ausdrucks* (numerisch oder Zeichenkette) muß mit dem Typ übereinstimmen, der mit *Name* definiert wird.

DEF FN
Anweisung

Die Definition einer Funktion ist auf eine Anweisung beschränkt. Argumente (*Arg*), die in der Funktionsdefinition erscheinen, dienen nur zur Definition der Funktion; sie haben keinen Einfluß auf Programmvariablen mit dem gleichen Namen. Ein Variablenname, der in *Ausdruck* benutzt wird, muß nicht in der Liste der Argumente erscheinen. Erscheint er in der Liste, wird der Wert für das Argument mitgegeben, 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 Zeichenwert übergibt. Der Typ der Funktion wird mit *Name* deklariert, und zwar auf dieselbe Art, wie Variablen deklariert werdem (siehe "Definition von Variablentypen" in Kapitel 3). Stimmt der Typ des *Ausdrucks* (Zeichenkette oder numerisch) nicht mit dem Funktionstyp überein, erhält man den Fehler "Type Mismatch" (Keine Typübereinstimmung). Ist die Funktion numerisch, wird der Wert für den Ausdruck in die durch *Name* angegebene Genauigkeit umgewandelt, bevor er an die aufrufende Anweisung übergeben wird.

DEF FN Anweisung

Die Anweisung DEF FN muß ausgeführt werden, um die Funktion zu definieren, bevor sie aufgerufen wird. Wird eine Funktion aufgerufen, bevor sie definiert ist, so erhält man den Fehler "Undefined User Function" (Nicht definierte Benutzerfunktion).

Zum anderen darf die Funktion mehr als einmal definiert werden. Die zuletzt ausgeführte 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 im direkten Modus ungültig.

DEF FN
Anweisung

Beispiel:

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

(Mit 2 wird geantwortet.)

```
 Radius? 2
 Fläche ist 12.56637
 Ok
```

In Zeile 20 wird die Funktion FNAREA definiert, die die Fläche 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
```

DEF SEG Anweisung

Zweck: Definiert das gerade angesprochene "Segment" des Hauptspeichers. Eine nachfolgende BLOAD-, BSAVE-, CALL-, PEEK-, POKE- oder USR-Definition definiert die aktuelle physische Adresse seiner Operation als einen Relativzeiger in dieses Segment.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: DEF SEG [=*Adresse*]

Bemerkungen: *Adresse* ist ein numerischer Ausdruck im Bereich 0 bis 65535.

Das erste Setzen eines Segments, wenn BASIC gestartet wird, ist das Datensegment des BASIC. Das Datensegment von BASIC ist der Anfang des Benutzerbereichs im Hauptspeicher. Wird eine Anweisung DEF SEG ausgeführt, die das Segment ändert, wird der Wert *nicht* auf das Datensegment des BASIC zurückgesetzt, wenn der Befehl RUN ausgeführt wird.

Ist *Adresse* in der Anweisung DEF SEG weggelassen, wird das Segment auf das Datensegment des BASIC gesetzt.

DEF SEG Anweisung

Ist *Adresse* angegeben, muß sein Wert eine 16-Byte-Grenze haben. Der Wert wird um vier Bits nach links verschoben (mit 16 multipliziert), um die Segmentadresse für nachfolgende Operationen zu formen. Das heißt, falls die *Adresse* hexadezimal ist, wird eine Null (0) angefügt, um die aktuelle Segmentadresse zu erhalten. BASIC führt keine Prüfung durch, um zu sehen, ob der Segmentwert gültig ist.

DEF und SEG müssen durch eine Leerstelle getrennt werden. Im anderen Fall wird BASIC die Anweisung DEFSEG=100 so interpretieren:

"Übergib den Wert 100 an die Variable DEFSEG"

Jeder eingegebene Wert, der sich außerhalb des angegebenen Bereichs befindet, erzeugt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Der vorherige Wert wird beibehalten.

Im Anhang C. "Unterprogramme in Maschinensprache" sind weitere Informationen über die Benutzung von DEF SEG enthalten.

DEF SEG Anweisung

Beispiel:

```
100 DEF SEG ' wieder auf Datensegment des BASIC setzen
200 ' Segment auf Farbbildschirmpuffer
210 DEF SEG=&HB800
```

Im zweiten Beispiel liegt der Bildschirmpuffer für den Farb-/Grafikbildschirmanschluß auf der absoluten Adresse B8000 hexadezimal. Da Segmente in 16-Byte-Grenzen angegeben werden, wird die letzte Hexadezimalziffer in der Anweisung DEF SEG weggelassen.

DEFTyp Anweisungen

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format: DEF*Typ* *Buchstabe*[-*Buchstabe*] [,*Buchstabe* [-*Buchstabe*]]...

Bemerkungen: *Typ* INT, SNG, DBL oder STR.

Buchstabe Ein Buchstabe des Alphabets (A bis Z).

Die Anweisung DEF*Typ* 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 DEF*Typ* für die Typgebung einer Variablen. Siehe "Definitionen von Variablentypen" in Kapitel 3.

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

DEFTyp Anweisungen

Werden Typdefinitionsanweisungen benutzt, müssen sie am Beginn des Programms stehen. Die Anweisung DEF*Typ* muß ausgeführt 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 ANIMAL = "KATZE": PRINT ANIMAL
60 X=10/3: PRINT X
RUN
 .3333333333333333
KATZE
 3
Ok
```

In Zeile 10 wird definiert, daß alle Variablen, die mit dem 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.

DEF USR
Anweisung

Zweck: Spezifiziert die Startadresse eines Unterprogramms in Maschinensprache, das später durch die Funktion USR aufgerufen wird.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: 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.

DEF USR Anweisung

Es ist möglich, die Adresse für die USR-Routine neu zu definieren. Die Zahl der Anweisungen DEF USR in einem Programm sind nicht begrenzt, so daß auf so viele Unterprogramme wie nötig zugegriffen werden kann. Der zuletzt ausgeführte Wert wird als Relativzeiger benutzt.

In Anhang C. "Unterprogramme in Maschinensprache" stehen genaue Informationen.

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.

DELETE Befehl

Zweck: Löscht Programmzeilen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

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

Bemerkungen: *Zeile1* Zeilennummer der ersten zu löschenden Zeile.

Zeile2 Zeilennummer der letzten zu löschenden Zeile.

Der Befehl DELETE löscht den angegebenen Zeilenbereich aus dem Programm. BASIC kehrt immer zur Befehlsebene zurück, nachdem DELETE ausgeführt wurde.

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

Beispiel:

In diesem Beispiel wird die Zeile 40 gelöscht:

```
DELETE 40
```

Im nächsten Beispiel werden die Zeilen 40 bis 100 je einschließlich gelöscht:

```
DELETE 40-100
```

Im letzten Beispiel werden alle Zeilen bis zu und einschließlich Zeile 40 gelöscht:

```
DELETE -40
```

DIM Anweisung

Zweck: Definiert die maximalen Werte für die Indizes von Bereichsvariablen und belegt danach den erforderlichen Speicherplatz.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

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

Bemerkungen:

Variable Der für den Bereich benutzte Name.

Indizes Eine Liste numerischer Ausdrücke, getrennt durch Kommas, die die Dimensionen des Bereichs angeben.

Bei der Ausführung setzt die Anweisung DIM alle Elemente eines angegebenen numerischen Bereichs auf Null (0). Bereichselemente für Zeichenketten haben alle eine variable Länge und zu Beginn keinen Inhalt (Länge 0).

Wird ein Bereichsvariablenname ohne Anweisung DIM benutzt, wird der höchste Wert für seinen Index mit 10 angenommen. Wird ein Index benutzt, der größer als das angegebene Maximum ist, erhält man den Fehler "Subscript out of Range" (Index außerhalb des Bereichs).

Der kleinste Wert für einen Index ist immer Null (0), wenn nicht durch die Anweisung OPTION BASE anders angegeben (siehe "Anweisung OPTION BASE" in diesem Kapitel). Die größte Anzahl von Dimensionen für einen Bereich ist 255. Die maximale Anzahl von Elementen pro Dimension ist 32767. Beide Angaben sind auch durch die Größe des Hauptspeichers und die Länge der Anweisungen begrenzt.

Wird versucht, einen Bereich mehr als einmal zu dimensionieren, tritt der Fehler "Duplicate Definition" (Doppelte Definition) auf. Es ist jedoch möglich, mit der Anweisung ERASE einen Bereich zu löschen, so daß er neu dimensioniert werden kann. Weitere Informationen über Bereiche siehe Kapitel 3. "Bereiche".

DIM Anweisung

Beispiel:

```
Ok
10 WRRMAX=2
20 DIM SIS(12), WRR$(WRRMAX,2)
30 DATA 26.5, 37, 8,29,80, 9.9, &H800
40 DATA 7, 18, 55, 12, 5, 43
50 FOR I=0 TO 12
60 READ SIS(I)
70 NEXT I
80 DATA SHERRY, ROBERT, "A:"
90 DATA "HI, SCOTT", HELLO, GOOD-BYE
100 DATA BOCA RATON, DELRAY, MIAMI
110 FOR I=0 TO 2: FOR J=0 TO 2
120 READ WRR$(I,J)
130 NEXT J,I
140 PRINT SIS(3); WRR$(2,0)
RUN
 29 BOCA RATON
Ok
```

In diesem Beispiel werden zwei Bereiche erstellt: ein eindimensionaler numerischer Bereich mit dem Namen SIS mit 13 Elementen - SIS (0) bis SIS(12) - und ein zweidimensionaler Bereich für Zeichenkette mit dem Namen WRR$, bestehend aus drei Zeilen und drei Spalten.

DRAW
Anweisung

Zweck: Zeichnet ein Objekt, das durch eine Zeichenkette angegeben ist.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Nur im Grafikmodus.

Format: 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 ausführt. Während der Ausführung prüft BASIC den Wert der Zeichenkette und interpretiert die einzelnen Buchstabenbefehle aus dem Inhalt der *Zeichenkette*. Die Befehle werden unten ausführlich erklärt.

DRAW
Anweisung

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 Länge 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.

Das Bildschirmverhältnis bestimmt den Abstand der horizontalen, vertikalen und diagonalen Punkte. Das Standardbildverhältnis von 4/3 bedeutet z. B., daß die horizontale Achse des Bildschirms 4/3 mal so lang ist wie die vertikale Achse. Mit dieser Information kann bestimmt werden, wieviele vertikale Punkte die gleiche Länge ergeben wie die Anzahl von Horizontalpunkten.

Die mittlere Auflösung besitzt 320 Horizontalpunkte und 200 Vertikalpunkte. Das bedeutet, daß acht Horizontalpunkte die gleiche Länge ergeben wie fünf vertikale Punkte, wenn das Bildschirmverhältnis 1 : 1 ist. Ist das Bildschirmverhältnis verschieden, muß die Anzahl der Vertikalpunkte mit dem Bildschirmverhältnis multipliziert werden. Nimmt man z. B. das Standardbildschirmverhältnis von 4/3, sind bei der mittleren Auflösung acht Horizontalpunkte in der Länge gleich 20/3 Vertikalpunkte oder 24 Horizontalpunkte sind gleich 20 Vertikalpunkete. Das bedeutet, daß:

```
DRAW "U8Ø R96 D8Ø L96"
```

ein Quadrat in mittlerer Auflösung zeichnet. Mit dem Standardbildschirmverhältnis von 4/3 haben in der hohen Auflösung 48 Horizontalpunkte die gleiche Länge wie 20 Vertikalpunkte.

DRAW
Anweisung

Die folgenden zwei Befehle können jedem der oben erwähnten Bewegungsbefehle vorangestellt werden.

B Bewegen, aber keine Punkte zeichnen.

N Bewegen, aber nach der Ausführung auf die alte Position zurückkehren.

Die folgenden Befehle sind ebenfalls verfügbar:

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 Größe auf dem Bildschirm erscheinen wie mit 0 oder 180 Grad im Standardbildschirmverhältnis 4/3.

C n Setzt die Farbe *n*. *n* darf sich für mittlere Auflösung im Bereich 0 bis 3 befinden, für hohe Auflösung im Bereich 0 bis 1. Für die mittlere Auflösung wählt *n* die Farbe aus der laufenden Palette aus, die mit der Anweisung COLOR definiert wurde. 0 ist die Hintergrundfarbe. Der Standardwert ist die Vordergrundfarbe - Farbe Nummer 3. Für die hohe Auflösung 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 Abständen, die mit den Befehlen U, D, L, R, E, F, G, H und dem relativen Befehl M gegeben werden, ergibt die Länge, um die bewegt werden soll. Der Standardwert ist 4, so daß der Skalenfaktor 1 ist.

X Variable;
Führt den Teil einer Zeichenkette aus. Dies erlaubt die Ausführung einer zweiten Zeichenkette innerhalb einer Zeichenkette.

In all diesen Befehlen können 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 benötigt, 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. Variablen in einer Bewegungsanweisung auf die folgende Weise verwenden:

```
M+=X1;.-=X2;
```

Man kann die Variablen auch in der Form VARPTR$ (*Variable*) anstelle von =*Variable;* definieren. Dies ist für Programme nützlich, die später umgewandelt werden sollen. Beispiel:

1. Methode

```
DRAW "XA$;"
DRAW "S=SCALE;"
```

2. Methode

```
DRAW "X"+VARPTR$(A$)
DRAW "S="+VARPTR$(SCALE)
```

Der Befehl X kann ein sehr nützlicher Befehl als Teil von DRAW sein, weil man damit Teile eines Objekts separat vom ganzen Objekt definieren kann. (Ein Fuß ist z.B. Teil eines Menschen.) Man kann mit X auch Zeichenketten von Befehlen zeichnen, die länger als 255 Zeichen lang sind.

Werden Koordinaten an DRAW übergeben, die sich nicht im Bereich befinden, wird der Koordinate, die sich außerhalb des Bereichs befindet, der nächstgültige Wert zugeordnet. Das bedeutet, die negativen Werte werden zu Null und X-Werte, größer als 199, werden 199. X-Werte, größer als 639, werden zu 639, X-Werte, größer als 319 in mittlerer Auflösung, gehen zur nächsten horizontalen Zeile.

Beispiel: Zeichnen eines Rechtecks:

```
5 SCREEN 1
1Ø A=2Ø
2Ø DRAW "U=A;R=A;D=A;L=A"
```

Zeichnen eines Dreiecks:

```
1Ø SCREEN 1
2Ø DRAW "E15 F15 L3Ø"
```

Erzeugen einen sich bewegenden Sterns:

```
1Ø SCREEN 1,Ø: COLOR Ø,Ø: CLS
2Ø DRAW "BM3ØØ,25" ' Anfangspunkt
3Ø STAR$="M+7,17 M-17,-12 M+2Ø,Ø M-17,12 M+7,-17"
4Ø FOR SCALE=1 TO 4Ø STEP 2
5Ø DRAW "C1;S=SCALE; BM-2,Ø;XSTAR$;"
6Ø NEXT
```

EDIT
Befehl

Zweck: Zeigt eine Zeile für eine Änderung an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: EDIT *Zeile*

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

Die Anweisung EDIT zeigt einfach die angegebene Zeile an und stellt den Positionsanzeiger unter die erste Ziffer der Zeilennummer. Jetzt kann die Zeile, wie unter "BASIC-Korrekturprogramm" in Kapitel 2 beschrieben, verändert werden.

Ein Punkt (.) kann benutzt werden, um die laufende Eingabezeile anzugeben. Wurde z. B. gerade eine Zeile eingegeben und diese Zeile soll geändert werden, zeigt der Befehl **EDIT.** diese Zeile für eine Änderung an.

Mit LIST können auch Programmzeilen für Änderungen angegeben werden. Siehe "Befehl LIST" in diesem Kapitel.

END
Anweisung

Zweck: Beendet die Programmausführung, schließt alle Dateien ab und kehrt zur Befehlsebene zurück.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: END

Bemerkungen: Anweisungen END können überall im Programm stehen und beenden die Ausführung. END und STOP unterscheiden sich in zwei Punkten:

- Bei END wird die Nachricht "Break" (Unterbrechung) nicht gedruckt.
- END schließt alle Dateien ab.

Die Anweisung END am Ende eines Programms ist wahlfrei. BASIC kehrt immer zur Befehlsebene zurück, nachdem END ausgeführt wurde.

END Anweisung

Beispiel:

```
52Ø IF K>1ØØØ THEN END ELSE GOTO 2Ø
```

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

EOF
Funktion

Zweck: Zeigt die Bedingung Dateiende an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = EOF (*Dateinummer*)

Bemerkungen: *Dateinummer* Die in der Anweisung OPEN angegebene Nummer.

Die Funktion EOF ist nützlich, um den Fehler "Input past End" (Eingabe nach logischem Ende) zu vermeiden. EOF übergibt -1 (wahr), wenn das Dateiende bei einer angegebenen Datei erreicht wurde. Eine 0 (Null) wird übergeben, wenn das Dateiende nicht erreicht wurde.

EOF ist nur dann von Bedeutung, wenn eine Datei für sequentielle Eingabe von Diskette oder Kassette eröffnet wurde oder für eine Datenfernverarbeitungsdatei. Eine -1 für eine Datenfernverarbeitungsdatei bedeutet, daß der Puffer leer ist.

EOF
Funktion

Beispiel:

```
1Ø OPEN "DATA" FOR INPUT AS #1
2Ø C=Ø
3Ø IF EOF(1) THEN END
4Ø INPUT #1,M(C)
5Ø C=C+1: GOTO 3Ø
```

In diesem Beispiel werden die Informationen von der sequentiellen Datei DATA gelesen. Dateien werden in den Bereich M gelesen, bis das Dateiende erreicht ist.

ERASE
Anweisung

Zweck: Löscht Bereiche aus dem Programm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: ERASE *Bereichsname*[,*Bereichsname*]...

Bemerkungen: *Bereichsname* Name eines Bereichs, der gelöscht werden soll.

Es kann nützlich sein, die Anweisung ERASE aufzurufen, wenn der Speicherplatz sehr klein wird, während ein Programm abläuft. Nachdem Bereiche gelöscht sind, kann der Platz im Hauptspeicher, dem die Bereiche zugeordnet waren, für andere Zwecke benutzt werden.

ERASE kann auch benutzt werden, um Bereiche im Programm neu zu dimensionieren. Wird versucht, einen Bereich neu zu dimensionieren, bevor er gelöscht wurde, erhält man den Fehler "Duplicate Definition" (Doppelte Definition).

Mit dem Befehl CLEAR können *alle* Variablen aus dem Arbeitsbereich gelöscht werden.

Beispiel:

```
Ok
1Ø START=FRE('''')
2Ø DIM BIG(1ØØ,1ØØ)
3Ø MIDDLE=FRE('''')
4Ø ERASE BIG
5Ø DIM BIG(1Ø,1Ø)
6Ø FINAL=FRE('''')
7Ø PRINT START, MIDDLE, FINAL
RUN
 62808          21980          62289
Ok
```

In diesem Beispiel wird die Funktion FRE benutzt, um zu zeigen, wie mit ERASE der Hauptspeicher geleert werden kann. Der Bereich BIG benutzte bis zu 40K Bytes des Hauptspeichers (62808 bis 21980), als er mit BIG(100,100) dimensioniert war. Nach dem Löschen konnte er auf BIG(10,10) neu dimensioniert werden und benötigte dafür nur wenig mehr als 500 Bytes (62808 bis 62289).

Die aktuellen Werte, die die Funktion FRE übergibt, können an den einzelnen Computern unterschiedlich sein.

ERR und ERL Variablen

Zweck: Übergibt den Fehlercode und die Zeilennummer, in der der Fehler aufgetreten ist.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = ERR

v = ERL

Bemerkungen: Die Variable ERR enthält den Fehlercode für den letzten Fehler, die Variable ERL enthält die Zeilennummer der Zeile, in der der Fehler entdeckt wurde. Die Variablen ERR und ERL werden gewöhnlich 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 für den direkten Modus, enthält ERL den Wert 65535. Da man nicht möchte, daß diese Nummer während RENUM verändert wird, wenn man prüfen möchte, ob sich ein bestimmter Fehler in einer Anweisung im direkten Modus ereignete, muß man folgende Form wählen:

IF 65535 = ERL THEN ...

ERR und ERL können mit Hilfe der Anweisung ERROR gesetzt werden (siehe nächsten Abschnitt).

BASIC-Fehlercodes sind in Anhang A "Nachrichten" aufgelistet.

ERR und ERL Variablen

Beispiel:

```
1Ø ON ERROR GOTO 1ØØ
2Ø LPRINT "Auf dem Drucker wird gedruckt"
3Ø END
1ØØ IF ERR=27 THEN LOCATE 23,1:
    PRINT "Drucker überprüfen": RESUME
```

In diesem Beispiel wird ein allgemeines Problem getestet: Papier fehlt im Drucker oder er ist nicht eingeschaltet.

ERROR Anweisung

Zweck:

- Simuliert den Auftritt eines BASIC-Fehlers oder
- erlaubt die Definition eigener Fehlercodes.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: ERROR *n*

Bemerkungen: *n* muß ein ganzzahliger Ausdruck zwischen 0 und 255 sein.

Ist der Wert von *n* der gleiche wie ein von BASIC benutzter Fehlercode (siehe Anhang A. Nachrichten), 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 Ausführung wird angehalten (siehe erstes Beispiel unten).

ERROR
Anweisung

Soll ein eigener Fehlercode definiert werden, muß ein Wert benutzt werden, der in BASIC nicht benutzt wird. (Es wird vorgeschlagen, die höchsten verfügbaren Werte zu benutzen; z.B. Werte über 200.) Die neuen Fehlercodes können dann in Fehlerbehandlungsroutinen getestet werden, genau wie andere Fehler (siehe zweites Beispiel unten).

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 Ausführung stoppt.

Beispiele: Das erste Beispiel simuliert den Fehler "String too long" (Zeichenkette zu lang):

```
Ok
10 T = 15
20 ERROR T
RUN
String too long in Line 20
Ok
```

Das nächste Beispiel ist Teil eines Spielprogramms, in dem Werte gesetzt werden können. Mit Hilfe des Fehlercodes 210, der in BASIC nicht benutzt wird, fängt das Programm diesen Fehler ab, sobald das Hauslimit überschritten 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
```

EXP
Funktion

Zweck: Errechnet die Exponentialfunktion.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v = EXP(x)*

Bemerkungen: *x* Kann irgendein numerischer Ausdruck sein.

Die Funktion übergibt die mathematische Zahl *e* hoch *x*, wobei *e* die Basis für den natürlichen Logarithmus ist. Ein Überlauf geschieht, falls *x* größer 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.

FIELD Anweisung

Zweck: Legt Platz für Variablen in einem Puffer für Dateien mit wahlfreiem Zugriff an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: FIELD [#]*Dateinummer, Länge* AS *Zeichenkettenvariable* [,*Länge* AS *Zeichenkettenvariable*]...

Bemerkungen: *Dateinummer*
Nummer, mit der die Datei eröffnet wurde.

Länge Numerischer Ausdruck. Gibt die Anzahl der Zeichenpositionen an, die einer *Zeichenkettenvariablen* zugeordnet werden sollen.

Zeichenkettenvariable
Zeichenkettenvariable, die für den Zugriff auf Dateien mit wahlfreiem Zugriff benutzt wird.

Die Anweisung FIELD definiert Variablen, die benutzt werden, um mit Hilfe von GET Daten aus dem wahlfreien Puffer zu lesen oder mit PUT Daten in den Puffer einzugeben.

FIELD
Anweisung

Die Anweisung:

```
FIELD 1, 2Ø AS N$, 1Ø AS ID$, 4Ø AS ADD$
```

übergibt die ersten 20 Positionen (Bytes) aus dem Puffer für die Datei mit wahlfreiem Zugriff an die Zeichenkettenvariable N$, die nächsten 10 Positionen an ID$ und die nächsten 40 Positionen an ADD$. FIELD stellt *nicht* selbst die Daten in den Puffer für eine Datei mit wahlfreiem Zugriff. 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 für eine Datei mit wahlfreiem Zugriff 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, dürfen die Satzlänge, die bei der Eröffnung der Datei angegeben wurde, nicht überschreiten; andernfalls erhält man den Fehler "Field Overflow" (Feldüberlauf).

Es können beliebig viele Anweisungen FIELD für die gleiche Dateinummer ausgeführt werden, und alle Anweisungen FIELD, die ausgeführt wurden, sind zur gleichen Zeit gültig. Jede neue Anweisung FIELD fängt wieder mit der ersten Zeichenposition im Puffer an. Das hat den Effekt, daß man mehrere Felddefinitionen für 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 sie auf den richtigen Platz in dem Puffer für Dateien mit wahlfreiem Zugriff. Wird eine nachfolgende Eingabeanweisung oder Anweisung LET mit diesem Variablennamen auf der linken Seite des Gleichheitszeichens ausgeführt, wird die Variable in dem Platz für Zeichenkettenvariablen angelegt und steht nicht mehr länger im Dateipuffer.

Siehe Anhang B. "BASIC-Diskettenein- und -ausgabe." Dort ist eine vollständige Beschreibung gegeben, wie Dateien für wahlfreien Zugriff angesprochen werden.

FIELD
Anweisung

Beispiel:

```
1Ø OPEN "A:CUST" AS #1
2Ø FIELD 1, 2 AS CUSTNO$, 3Ø AS CUSTNAME$,
      35 AS ADDR$
3Ø LSET CUSTNAME$+"O'NEIL INC"
4Ø LSET ADDRES+"5Ø SE 12TH ST, NY, NY"
5Ø LSET CUSTNO$=MKI$(785Ø)
6Ø PUT 1,1
7Ø GET 1,1
8Ø CNUM%= CVI(CUSTNO$): N$ = CUSTNAME$
9Ø PRINT CNUM$, N$, ADDR$
```

In diesem Beispiel wird eine Datei mit Namen CUST als Datei mit wahlfreiem Zugriff eröffnet. Die Variable CUSTNO$ wird den ersten zwei Stellen jedes Satzes zugeordnet, CUSTNAME$ den nächsten 30 Positionen und ADDR$ den nächsten 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 90 werden die drei Felder angezeigt. In Zeile 80 beachten, daß es möglich ist, einen Variablennamen zu benutzen, der auf der *rechten* Seite einer Zuordnungsanweisung in einer Anweisung FIELD definiert war.

FILES Befehl

Zweck: Zeigt die Dateinamen auf einer Diskette an. Der Befehl FILES in BASIC ist ähnlich dem Befehl DIR in DOS.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: FILES [*Dateiangabe*]

Bemerkungen: *Dateiangabe*
Zeichenkettenausdruck für eine Dateispezifikation, wie in "Namensgebung für Dateien" in Kapitel 3 erklärt. Ist *Dateiangabe* weggelassen, werden alle Dateien im DOS-Standardlaufwerk angelistet.

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

FILES Befehl

Ist ein Laufwerk als Teil der *Dateiangabe* spezifiziert, werden die Dateien auf der Diskette in diesem Laufwerk, die dem angegebenen Dateinamen entsprechen, angelistet; andernfalls wird die DOS-Standardeinheit angenommen.

Beispiele:

```
FILES
```

Damit werden alle Dateien im DOS-Standarddiskettenlaufwerk angezeigt.

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

Damit werden alle Dateien mit der Erweiterung .**BAS** auf der DOS-Standarddisketteneinheit angezeigt.

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

Damit werden alle Dateien im Laufwerk **B:** angezeigt.

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

Damit werden alle Dateien im DOS-Standardlaufwerk angezeigt, die einen Dateinamen haben, der mit TEST, gefolgt von zwei oder weniger anderen Zeichen beginnt, und in der Erweiterung .**BAS** hat.

FIX
Funktion

Zweck: Schneidet *x* auf eine ganze Zahl ab.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *v* = FIX(*x*)

Bemerkungen: *x* kann jeder numerische Ausdruck sein.

FIX schneidet alle Ziffern rechts des Dezimalpunkts ab und übergibt den Wert der Ziffern, die links des Dezimalpunkts stehen.

Der Unterschied zwischen FIX und INT ist der, daß FIX nicht die nächstkleinere Zahl übergibt, falls *x* negativ ist.

Siehe auch die Funktionen INT und CINT, die ebenfalls ganze Zahlen übergeben.

Beispiel:

```
Ok
PRINT FIX(45.67)
 45
Ok
PRINT FIX(-2.89)
-2
Ok
```

In diesen Beispielen muß beachtet werden, daß FIX *nicht* rundet, wenn es in eine ganze Zahl umwandelt.

FOR und NEXT Anweisungen

Zweck: Führt eine Folge von Instruktionen in einer Schleife aus, wobei die Zahl der Ausführungen angegeben ist.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format:

FOR *Variable*=*x* TO *y* [STEP *z*]

.
.
.

NEXT [*Variable*][,*Variable*]...

Bemerkungen:

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

x ist ein numerischer Ausdruck, der den Anfangswert des Zählers enthält.

y ist ein numerischer Ausdruck, der den Endwert des Zählers enthält.

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

FOR und NEXT Anweisungen

Die Programmzeilen, die der Anweisung FOR folgen, werden ausgeführt, bis eine Anweisung NEXT erreicht wird. Der Zähler wird um die Summe erhöht, die im STEP-Wert (*z*) angegeben ist. Wird der Wert für *z* nicht angegeben, wird als Schrittweite 1 (eins) angenommen.
Eine Prüfung wird ausgeführt, ob der Wert des Zählers jetzt größer ist als der Endwert *y*. Ist er nicht größer, verzweigt BASIC zurück zur Anweisung nach der Anweisung FOR, und der Prozeß wird wiederholt. Ist er größer, so geht die Ausführung mit der Anweisung weiter, die der Anweisung NEXT folgt. Dies ist eine FOR...NEXT-Schleife.

Ist der Wert von *z* negativ, läuft der Test umgekehrt ab. Der Zähler wird jedesmal, wenn die Schleife beendet ist, erniedrigt, und die Schleife wird so lange ausgeführt, bis der Zähler kleiner als der Endwert ist.

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

Der Programmdurchsatz wird verbessert, falls - wenn möglich - ganzzahlige Zähler verwendet werden.

FOR und NEXT Anweisungen

Geschachtelte Schleifen

FOR...NEXT-Schleifen können geschachtelt werden, d. h. eine FOR...NEXT-Schleife kann in eine andere FOR...NEXT-Schleife gestellt werden. Sind die Schleifen geschachtelt, muß jede Schleife als Zähler eigene Variablennamen haben. Die Anweisung NEXT der inneren Schleife muß vor der für die äußere Schleife erscheinen. Haben geschachtelte Schleifen das gleiche Ende, kann für 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 kann weggelassen werden. In diesem Fall bezieht sich die Anweisung NEXT auf die letzte Anweisung FOR. Werden geschachtelte FOR...NEXT-Schleifen benutzt, sollte(*n*) die Variable(*n*) für alle Anweisungen NEXT eingefügt werden, um Konfusionen zu vermeiden. Erforderlich ist dies z.B., wenn man aus einer geschachtelten Schleife heraus verzweigt. (Werden Variablennamen in den Anweisungen NEXT benutzt, wird das Programm etwas langsamer ausgeführt.)

Wird eine Anweisung NEXT vor seiner zugehörigen Anweisung FOR ausgeführt, erhält man den Fehler "NEXT without FOR" (NEXT ohne FOR).

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

```
Ok
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
```

FOR und NEXT Anweisungen

Im nächsten Beispiel wird die Schleife nicht ausgeführt, weil der Anfangswert der Schleife größer 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 ausgeführt. Der Endwert der Schleifenvariablen wird immer vor dem Anfangswert gesetzt. (Dies ist von einigen anderen Versionen des BASIC verschieden, bei denen der Anfangswert des Zählers vor dem Endwert gesetzt wird. In einem anderen BASIC kann die Schleife dieses Beispiels sechsmal ausgeführt werden.)

```
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
```

FRE
Funktion

Zweck: Übergibt die Anzahl der Bytes im Hauptspeicher, die nicht von BASIC benutzt werden. Die Zahl beinhaltet nicht die Größe des reservierten Teils für den Arbeitsbereich des Interpretierers (normalerweise 2,5K bis 4K Bytes).

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: *v* = FRE(*x*)

v = FRE(*x$*)

Bemerkungen: *x* und *x$* sind Scheinargumente.

Da die Zeichenketten in BASIC eine variable Länge haben können (nach jeder Zuweisung zu einer Zeichenkettenvariablen kann sich ihre Länge ändern), werden Zeichenketten dynamisch behandelt. Deshalb kann sich der Bereich für Zeichenketten aus mehreren Teilen zusammensetzen, die durch Zwischenräume getrennt sind.

FRE
Funktion

Wird FRE mit einer Zeichenkette aufgerufen, sammelt BASIC alle Daten, die noch benötigt werden, und löscht alle unbenutzten Bereiche des Hauptspeichers, die einmal für Zeichenketten benutzt wurden. Die Daten werden alle aneinandergehängt, so daß das Programm seine Ausführung fortsetzen kann, bis der Platz im Hauptspeicher wirklich nicht mehr ausreicht.

Dieses Säubern des Hauptspeichers geschieht mit BASIC automatisch, wenn der benutzte Arbeitsbereich nicht mehr ausreicht. Wird dies durch FRE("") periodisch vorgenommen, dauert diese Säuberung nicht solange und ergibt nur kleinere Verzögerungen.

CLEAR ,*n* setzt die maximale Anzahl von Bytes für einen BASIC-Arbeitsbereich. FRE übergibt die Anzahl der freien Plätze in diesem BASIC-Arbeitsbereich. Steht nichts im Arbeitsbereich, ist der Wert, der von FRE übergeben wird, zwischen 2,5K und 4K Bytes (die Größe des reservierten Arbeitsbereichs für den Interpretierer) kleiner als die Anzahl der Bytes, die von CLEAR gesetzt wurde.

FRE
Funktion

Beispiel:

```
Ok
PRINT FRE(Ø)
 14542
Ok
```

Der von FRE übergebene Wert in diesem Beispiel kann an den einzelnen Computern unterschiedlich sein.

GET
Anweisung (Dateien)

Zweck: Liest einen Satz aus einer Datei mit wahlfreiem Zugriff in einen Puffer für wahlfreien Zugriff.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: GET [#] *Dateinummer* [, *Nummer*]

Bemerkungen: *Dateinummer*
Nummer, mit der die Datei eröffnet wurde.

Nummer Nummer des zu lesenden Satzes im Bereich 1 bis 32767. Ist *Nummer* nicht angegeben, wird der nächste Satz (nach dem letzten GET) in den Puffer gelesen.

Nach der Anweisung GET können INPUT#, LINE INPUT # oder Bezugnahmen auf Variablen, die in der Anweisung FIELD definiert sind, benutzt werden, um Zeichen aus dem Puffer der Datei mit wahlfreiem Zugriff zu lesen. Siehe Anhang B. "BASIC-Diskettenein- und -ausgabe." Dort stehen weitere Einzelheiten über die Benutzung von GET.

GET
Anweisung (Dateien)

Da BASIC und DOS so viele Sätze wie möglich in 512 Byte-Sektoren blocken, führt die Anweisung GET nicht notwendigerweise ein physisches Lesen von der Diskette aus.

GET kann auch für 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 überschreiten.

Beispiel:

```
10 OPEN "A:CUST" AS #1
20 FIELD 1, 30 AS CUSTNAME$, 30 AS ADDR$,
      35 AS CITY$
30 GET 1
40 PRINT CUSTNAME$, ADDR$, CITY$
```

In diesem Beispiel wird die Datei CUST für wahlfreien Zugriff eröffnet. 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.

GET
Anweisung (Grafik)

Zweck: Liest Punkte aus dem Bereich des Bildschirms.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	***

Nur im Grafikmodus.

Format: GET (*x1,y1*)–(*x2,y2*),*Bereichsname*

Bemerkungen: (*x1,y1*), (*x2,y2*)
sind Koordinaten in absoluter oder relativer Form. Siehe unter "Spezifizieren von Koordinaten unter Grafikmodus" in Kapitel 3. Dort sind weitere Informationen über Koordinaten enthalten.

Bereichsname
Name eines Bereichs, in dem die Information gespeichert werden soll.

GET
Anweisung (Grafik)

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 gegenüberliegende Ecken. (Dies ist das gleiche wie das Viereck, das unter der Anweisung LINE mit der Auswahl B gezeichnet wurde.)

Mit GET und PUT können 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 einfach dazu benutzt, das Abbild zu speichern, und muß numerisch sein. Jedoch ist jede Genauigkeit erlaubt. Die benötigte Größe des Bereichs in Bytes ist:

4+INT((*x***Bit-Anzahl*+7)/8)**y*

wobei *x* und *y* die Längen der horizontalen und vertikalen Seiten des Rechtecks sind. Der Wert für die *Bit-Anzahl* ist 2 für die mittlere Auflösung und 1 für die hohe Auflösung.

GET
Anweisung (Grafik)

Soll die Anweisung GET dazu benutzt werden, ein 10x12-Abbild in mittlerer Auflösung zu bekommen, ist die Anzahl der benötigten Bytes 4+INT((10*2+7)*12 oder 40 Bytes. Die Anzahl von Bytes pro Element eines Bereichs sind:

- 2 für ganzzahlig
- 4 für einfache Genauigkeit
- 8 für doppelte Genauigkeit

Deshalb reicht ein ganzzahliger Bereich mit mindestens 20 Elementen aus.

Die Information vom Bildschirm wird im Bereich wie folgt gespeichert:

1. Zwei Bytes enthalten die x-Dimension in Bits
2. Zwei Bytes enthalten die y-Dimension in Bits
3. Die Daten

Es ist möglich, die x- und y-Dimensionen und selbst die Daten zu prüfen, wenn ein ganzzahliger Bereich benutzt wurde. Die x-Dimension steht in Element 0 des Bereichs, die y-Dimension steht in 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. Jedoch werden die Daten so übertragen, daß zuerst das Byte mit hohem Stellenwert, danach das mit niedrigem Stellenwert übertragen wird.

Die Daten für Punktzeile des Rechtecks werden linksbündig an einer Byte-Grenze gespeichert, sodaß, falls weniger als ein Vielfaches von acht Bits gespeichert wird, der Rest des Bytes mit Nullen aufgefüllt wird.

PUT und GET arbeiten in mittlerer Auflösung wesentlich schneller, wenn *x1* MOD 4 gleich Null ist, und in hoher Auflösung, wenn *x1* MOD 8 gleich Null ist. In diesem Spezialfall fallen die Rechteckgrenzen mit Byte-Grenzen zusammen.

GOSUB und RETURN Anweisungen

Zweck: Verzweigen in ein Unterprogramm und Rückkehr von einem Unterprogramm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: GOSUB *Zeile*

.
.
.

RETURN

Bemerkungen: *Zeile* Zeilennummer der ersten Zeile des Unterprogramms.

Ein Unterprogramm kann so oft wie nötig in einem Programm aufgerufen werden, und ein Unterprogramm kann in einem anderen Unterprogramm aufgerufen werden. Diese Verschachtelung von Unterprogrammen wird nur vom verfügbaren Hauptspeicher begrenzt.

GOSUB und RETURN Anweisungen

Durch die Anweisung RETURN verzweigt BASIC zu der Anweisung zurück, die der letzten Anweisung GOSUB folgt. Ein Unterprogramm kann mehrere Anweisungen RETURN enthalten, falls man von verschiedenen Punkten des Unterprogramms zurückverzweigen möchte. Unterprogramme können überall 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 übergehen.

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

Beispiel:

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

GOSUB und RETURN Anweisungen

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 Ausführung der Anweisungen, bis die Anweisung RETURN in Zeile 70 erscheint. An diesem Punkt verzweigt das Programm zu der Anweisung nach dem Unterprogrammaufruf zurück. Das ist die Zeile 20. Die Anweisung END in Zeile 30 verhindert, daß das Unterprogramm ein zweites Mal ausgeführt wird.

GOTO Anweisung

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: GOTO *Zeile*

Bemerkungen: *Zeile* Zeilennummer einer Programmzeile.

Ist *Zeile* die Zeilennummer einer ausführbaren Anweisung, werden diese Anweisung und die nachfolgenden ausgeführt. Ist *Zeile* eine nicht ausführbare Anweisung (wie z.B. REM oder DATA), fährt das Programm mit der ersten ausführbaren Anweisung nach *Zeile* fort.

Die Anweisung GOTO kann auch im direkten Modus benutzt werden, um in einem Programm zu einem bestimmten Punkt zu verzweigen. Dies kann beim Testen nützlich sein.

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

GOTO
Anweisung

Beispiel:

```
Ok
5 DATA 5,7,12
1Ø READ R
2Ø PRINT "R =";R,
3Ø A = 3.14*R^2
4Ø PRINT "FLÄCHE =";A
5Ø GOTO 5
RUN
R = 5              FLÄCHE = 78.5
R = 7              FLÄCHE = 153.86
R =12              FLÄCHE = 452.16
Out of Data in 1Ø
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. (Beachten, daß die Verzweigung zur Anweisung DATA keine zusätzlichen Werte zur internen Datentabelle hinzufügt.)

HEX$ Funktion

Zweck: Übergibt eine Zeichenkette, die den hexadezimalen Wert eines dezimalen Arguments darstellt.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = HEX$(*n*)

Bemerkungen: *n* ist ein numerischer Ausdruck im Bereich −32768 bis 65535.

Ist *n* negativ, wird das Zweier-Komplement benutzt. Das heißt, HEX$(-*n*) ist das gleiche wie HEX$(65536-*n*).

Die Funktion OCT$ wird für die Oktalumwandlung benutzt.

HEX$
Funktion

Beispiel: Im folgenden Beispiel wird die Funktion HEX$ benutzt, um die hexadezimale Darstellung von zwei eingegebenen dezimalen Werten zu errechnen.

```
Ok
1Ø INPUT X
2Ø A$ = HEX$(X)
3Ø PRINT X "DEXIMAL IST " A$ " HEXADEXIMAL"
RUN
? 32
 32 DEXIMAL IST 2Ø HEXADEXIMAL
Ok
RUN
? 1Ø23
 1023 DEXIMAL IST 3FF HEXADEXIMAL
Ok
```

IF Anweisung

Zweck: Ändert den Programmablauf, abhängig vom Ergebnis eines Ausdrucks.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: IF *Ausdruck* [,]THEN *Angabe* [ELSE *Angabe*]

IF *Ausdruck* [,]GOTO *Zeile* [[,]ELSE *Angabe*]

Bemerkungen:

Ausdruck kann irgendein numerischer Ausdruck sein.

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

Zeile Zeilennummer einer im Programm existierenden Zeile.

IF
Anweisung

Ist *Ausdruck* wahr (nicht Null), wird die Angabe THEN oder GOTO ausgeführt. THEN kann entweder einer Zeilennummer zur Verzweigung oder einer oder mehreren Anweisungen, die ausgeführt werden sollen, folgen. Hinter GOTO steht immer eine Zeilennummer.

Ist das Ergebnis von *Ausdruck* falsch (Null), wird die Angabe THEN oder GOTO ignoriert, und die Angabe ELSE wird - falls sie vorhanden ist - ausgeführt. Die Ausführung geht mit der nächsten ausführbaren Anweisung weiter.

Wird die Anweisung IF...THEN im Direktmodus eingegeben und es wird eine Zeilennummer angesprochen, erhält 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 für einfache oder doppelte Genauigkeit sind, muß man sich daran erinnern, daß die interne Darstellung der Werte nicht immer genau ist. (Dies kommt daher, weil die Werte einfacher und doppelter Genauigkeit intern in binärer Gleitkommadarstellung gespeichert sind.) Deshalb muß dieser Test über 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 übergibt ein richtiges Ergebnis, falls der Wert von A gleich 1.0 mit einem relativen Fehler kleiner als 1.0E-6 ist.

IF Anweisung

Auch muß beachtet werden, daß IT... THEN...ELSE eine Anweisung ist. Das heißt die Angabe ELSE kann keine eigene Programmzeile sein. Beispiel:

```
1Ø IF A=B THEN X=4
2Ø ELSE P=Q
```

Dieses Beispiel ist falsch; es muß wie folgt aussehen:

```
1Ø IF A=B THEN X=4 ELSE P=Q
```

Verschachtelung von Anweisungen IF: Anweisungen IF...THEN...ELSE können verschachtelt werden. Die Verschachtelung ist nur durch die Länge der Zeile begrenzt. Beispiel:

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

Es handelt sich um eine gültige Anweisung. Enthält die Anweisung nicht die gleiche Anzahl von ELSE- und THEN-Angaben, wird ELSE mit dem nächsten 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.

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

```
2ØØ IF I THEN GET #1,I
```

Liegt im nächsten 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 gedruckt. Es sollte beachtet werden, daß in der Angabe THEN zwei Anweisungen enthalten sind.

```
1ØØ IF (I>1Ø) AND ( I<2Ø) THEN
       DB=1982-1: GOTO 3ØØ
  ELSE PRINT "AUSSERHALB DES BEREICHS"
```

Die nächte Anweisung bewirkt, daß die Ausgabe entweder zum Bildschirm oder zum Drucker geht, abhängig vom Wert der Variablen (IOFLAG). Ist IOFLAG falsch (Null), geht die Ausgabe zum Drucker; andernfalls geht sie zum Bildschirm:

```
21Ø IF IOFLAG THEN PRINT A$ ELSE LPRINT A$
```

INKEY$ Variable

Zweck: Liest ein Zeichen von der Tastatur.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = INKEY$

Bemerkungen: INKEY$ liest nur ein Zeichen, auch wenn sich mehrere Zeichen im Tastaturpuffer befinden. Der übergebene Wert ist eine Zeichenkette aus Null, einem oder zwei Zeichen.

- Eine Null-Zeichenkette (Länge Null) zeigt an, daß sich im Puffer kein Zeichen befindet.
- Eine Zeichenkette aus einem Zeichen enthält 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 enthält Anhang G. "ASCII-Zeichencodes."

INKEY$ Variable

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 Bildschirm kein Zeichen angezeigt, und alle Zeichen werden an das Programm übergeben. Ausnahmen (Tastatur):

- Ctrl-Break stoppt das Programm.
- Ctrl-Num sendet das System in den Pausenstatus.
- Alt-Ctrl-Del setzt das System zurück.
- PrtSc druckt den Inhalt des Bildschirms.

Wird die Eingabetaste als Antwort auf INKEY$ gedrückt, wird das Zeichen für Wagenrücklauf an das Programm übergeben.

Hinweis: Um Komplikationen mit dem Eingabepuffer des Kassetten-BASIC zu vermeiden, muß folgendes ausgeführt werden:

```
DEF SEG: POKE 1Ø6,Ø
```

Dies sollte ausgeführt werden, nachdem INKEY$ das letzte Zeichen einer Tastenfunktion erhalten hat. POKE wird im Disk- oder erweiterten BASIC nicht benötigt.

INKEY$ Variable

Beispiel: Der folgende Programmabschnitt hält das Programm an, bis irgendeine Taste der Tastatur gedrückt wird:

```
11Ø PRINT "Zum Weitermachen eine Taste drücken"
12Ø A$=INKEY$: IF A$="" THEN 12Ø
```

Im nächsten Beispiel wird ein übergebener Zwei-Zeichen-Code getestet:

```
21Ø KB$=INKEY$
22Ø IF LEN(KB$)=2 THEN KB$=RIGHT$(KB$,1)
```

INP
Funktion

Zweck: Übergibt ein gelesenes Byte von Anschluß *n*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = INP(*n*)

Bemerkungen: *n* muß im Bereich 0 bis 65535 liegen.

INP ist die Gegenfunktion zur Anweisung OUT (siehe "Anwendung OUT" in diesem Kapitel).

INP führt die gleiche Funktion aus, wie die Instruktion IN in Assembler-Sprache. In der Broschüre *IBM Personalcomputer Technisches Handbuch* ist eine Beschreibung der gültigen Anschlüsse (Ein-/Ausgabeadressen) enthalten.

Beispiel:

```
1ØØ A=INP(225)
```

Diese Instruktion liest ein Byte vom Anschluß 255 und ordnet seinen Inhalt der Variablen A zu.

INPUT
Anweisung

Zweck: Erhält die Eingabe von der Tastatur während der Programmausführung.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: INPUT[;][*"Anfrage"*;] *Variable*[,*Variable*]...

Bemerkungen: *"Anfrage"* Zeichenkettenkonstante, die als Anfrage für eine gewünschte Eingabe benutzt wird.

Variable Name einer numerischen oder Zeichenkettenvariablen oder eines Zeichenelements, an das die Eingabe übergeben wird.

Wenn im Programm die Anweisung INPUT erscheint, zeigt es mit einem Fragezeichen (?) am Bildschirm an, daß auf Dateneingabe gewartet wird. Ist *"Anfrage"* angegeben, wird diese Zeichenkette vor dem Fragezeichen angezeigt. Danach können die benötigten Daten von der Tastatur eingegeben werden.

Falls statt des Semikolons (;) ein Komma (,) nach Anfrage eingegeben wird, wird das Fragezeichen unterdrückt. Wird die Anweisung INPUT "GEBURTSTAG EINGEBEN",B$ eingegeben, wird die Anfrage ohne Fragezeichen angezeigt.

Die eingegebenen Daten werden der oder den *Variablen* der Variablenliste übergeben. Die angegebenen Daten müssen durch Kommas (,) getrennt sein, und die Anzahl der Daten muß dieselbe sein, wie die Anzahl der Variablen in der Liste.

Der Typ der Daten muß dem Typ der dazugehörigen Variablen übereinstimmen. (Zeichenketten, die auf Anfrage einer Anweisung INPUT eingegeben werden, müssen nicht in Anführungszeichen (") eingeschlossen sein, außer sie enthalten Kommas oder signifikante führende oder nachstehende Leerstellen.)

Werden als Anwort auf INPUT zu viele oder zu wenige Daten eingegeben oder wird ein falscher Typ für einen Wert eingegeben (Buchstaben statt Zahlen usw.), zeigt BASIC die Nachricht "? Redo from Start" (Von Anfang an neu eingeben) an. Wird nur eine einzelne Variable verlangt, kann man die Eingabetaste betätigen, um anzuzeigen, daß als Standardwert für numerische Eingabe 0 oder ein leerer Satz für Zeichenketteneingabe benutzt werden soll. Wird jedoch mehr als eine Variable verlangt, wird durch Betätigen der Eingabetaste die Anweisung "? Redo from Start" angezeigt, weil zu wenige Eingaben erfolgten. BASIC ordnet die Eingabewerte erst dann den Variablen zu, wenn alle Eingaben richtig sind.

Folgt der Anweisung INPUT im Disk- und erweiterten BASIC sofort ein Semikolon (;), wird nach dem Betätigen der Eingabetaste nach der Eingabe der Eingabedaten nicht die Zeichenfolge Wagenrücklauf, Zeilenvorschub auf dem Bildschirm erzeugt. Das bedeutet, der Positionsanzeiger bleibt auf derselben Zeile wie die Eingabe.

Beispiele:

```
Ok
10 INPUT X
20 PRINT X "QUADRIERT IST" X^2
30 END
RUN
?
```

In diesem Beispiel zeigt das Fragenzeichen (?) an, daß etwas eingegeben werden soll. Wird z.B. eine 5 eingegeben, fährt das Programm fort:

```
   .
   .
   .
   ? 5
    5 QUADRIERT IST 25
  Ok
```

```
Ok
10 PI=3.14
20 INPUT "RADIUS";R
30 A=PI*R^2
40 PRINT "DIE FLÄCHE DES KREISES IST";A
50 END
RUN
RADIUS?
```

In diesem zweiten Beispiel wurde eine Frage in Zeile 20 eingefügt, so daß der Computer mit der Frage antwortet: "RADIUS?". Es wird mit 7.4 geanwortet. Das Programm fährt fort:

```
.
.
.
RADIUS?  7.4
DIE FLÄCHE DES KREISES IST 171.9464
Ok
```

INPUT # Anweisung

Zweck: Liest Daten aus einer sequentiellen Einheit oder Datei und weist sie den Programmvariablen zu.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: INPUT *#Dateinummer, Variable* [,*Variable*]...

Bemerkungen: *Dateinummer*
Nummer, die für die Eingabedatei benutzt wurde.

Variable Name einer Variablen, dem Daten aus der Datei zugeordnet werden. Es kann sich um eine Zeichenketten- oder numerische Variable handeln oder um ein Bereichselement.

INPUT
Anweisung

Die sequentielle Datei kann sich auf Diskette oder Kassette befinden. Es kann sich um sequentielle Daten des Datenfernverarbeitungsanschlusses handeln. Die Daten können auch von der Tastatur kommen (KYBD:).

Der Typ der Dateidaten muß mit dem Typ der angegebenen Variablennamen übereinstimmen. Mit INPUT # wird nicht wie mit INPUT ein Fragezeichen (?) angezeigt.

Die Daten in der Datei müssen so aussehen, als wären sie als Antwort auf eine Anweisung INPUT eingegeben worden. Bei numerischen Werten werden führende Leerstellen, Wagenrückläufe und Zeichen für neue Zeilen ignoriert. Das erste Zeichen, das nicht gleich einer Leerstelle, einem Wagenrücklauf oder einem Zeilenvorschub ist, wird als Anfang der Zahl angenommen. Die Zahl endet mit einer Leerstelle, einem Wagenrücklauf, einem Zeilenvorschub oder einem Komma.

INPUT # Anweisung

Sucht sich BASIC die Daten für eine Zeichenkette, werden führende Leerstellen, Wagenrückläufe und Zeilenvorschübe auch ignoriert. Alle anderen Zeichen werden als Beginn der Zeichenkette angenommen. Ist das erste Zeichen ein Anführungszeichen ("), besteht die Zeichenkette aus allen Zeichen, die zwischen dem ersten und dem zweiten Anführungszeichen stehen. Das bedeutet, daß eine in Anführungszeichen eingeschlossene Zeichenkette kein Anführungszeichen als Zeichen enthalten darf. Ist das erste Zeichen einer Zeichenkette kein Anführungszeichen ("), dann handelt es sich um eine Zeichenkette ohne Anführungszeichen; das Lesen wird durch ein Komma (,),einen Wagenrücklauf, einen Zeilenvorschub oder nach 255 gelesenen Zeichen beendet. Wird bei der Eingabe eines numerischen Wertes oder einer Zeichenkette das Dateiende erreicht, wird diese Eingabe unterdrückt.

INPUT # kann auch mit einer Datei mit wahlfreiem Zugriff benutzt werden.

Beispiel: Siehe Anhang B. "BASIC-Diskettenein- und -ausgabe."

INPUT$
Funktion

Zweck: Übergibt eine Zeichenkette von *n* Zeichen, die von der Tastatur oder von einer Datei mit der Nummer *Dateinummer* gelesen wurde.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = INPUT$(*n*[,[#]*Dateinummer*])

Bemerkungen: *n* ist die Anzahl der Zeichen, die von der Datei gelesen werden sollen.

Dateinummer
Dateinummer, die in der Anweisung OPEN benutzt wird. Wird *Dateinummer* weggelassen, wird von der Tastatur gelesen.

INPUT$ Funktion

Wird die Tastatur für die Eingabe benutzt, wird kein Zeichen auf dem Bildschirm angezeigt. Alle Zeichen (einschließlich der Steuerzeichen) werden an das Programm übergeben, außer Ctrl-Break, das benutzt wird, um die Ausführung der Funktion INPUT$ zu unterbrechen. Wird von der Tastatur auf die Funktion INPUT$ geantwortet, ist es nicht nötig, die Eingabetaste zu betätigen.

Mit der Funktion INPUT$ können Zeichen über Tastatur eingelesen werden, die für das BASIC-Korrekturprogramm wichtig sind, wie z.B. Rücksetzen (ASCII-Code 8). Sollen diese Sonderzeichen gelesen werden, muß INPUT$ oder INKEY$ benutzt werden (*nicht* INPUT oder LINE INPUT).

Für Datenfernverarbeitungsdateien ist es besser, die Funktion INPUT$ zu benutzen, als INPUT # und LINE INPUT #, da in der Datenfernverarbeitung alle ASCII-Zeichen von Bedeutung sein können. Siehe Anhang F. "Datenfernverarbeitung."

INPUT$
Funktion

Beispiel: Das folgende Programm listet den Inhalt einer sequentiellen Datei in hexadezimal 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 nächsten Beispiel wird ein einzelnes Zeichen als Antwort auf eine Frage von der Tastatur gelesen:

```
100 PRINT "EINGABE P FÜR FORTFAHREN, S FÜR STOPP"
110 X$=INPUT$(1)
120 IF X$="P" THEN 500
130 IF X$="S" THEN 700 ELSE 100
```

INSTR Funktion

Zweck: Sucht das erste Auftreten der Zeichenkette *y$* in *x$* und übergibt die Position an der die Zeichenkette gefunden wurde. Der wahlfreie Relativzeiger *n* setzt die Position fest, an deren Stelle mit dem Suchen in *x$* begonnen werden soll.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = INSTR([*n*,]*x$,y$*)

Bemerkungen: *n* ist ein numerischer Ausdruck im Bereich 1 bis 255.

x$,y$ können Zeichenkettenvariablen, Zeichenkettenausdrücke oder Zeichenkettenkonstanten sein.

Ist *n* größer LEN(*x$*) oder ist *x$* leer oder kann *y$* nicht gefunden werden, übergibt INSTR eine 0 (Null). Ist *y$* leer, übergibt INSTR ein *n* (oder 1, falls *n* nicht angegeben ist).

Liegt *n* nicht im vorgegebenen Bereich, wird der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf) übergeben.

INSTR
Funktion

Beispiel:

```
Ok
1Ø A$ = "ABCDEB"
2Ø B$ = "B"
3Ø 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.

INT
Funktion

Zweck: Übergibt die größte ganze Zahl, die kleiner oder gleich *x* ist.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = INT(*x*)

Bemerkungen: *x* ist ein numerischer Ausdruck.

Siehe Funktionen FIX und CINT; diese übergeben ebenfalls ganzzahlige Werte.

Beispiel:

```
Ok
PRINT INT(45.67)
 45
PRINT INT(-2.89)
-3
Ok
```

In diesem Beispiel wird gezeigt, daß INT positive ganzzahlige Werte abschneidet, aber negative Werte aufrundet (in negativer Richtung).

KEY
Anweisung

Zweck: Setzt und zeigt die Funktionstasten an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format:

KEY *n, x$*

KEY LIST

KEY ON

KEY OFF

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

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

KEY
Anweisung

Durch die Anweisung KEY ist es möglich, die Funktionstasten so zu setzen, 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 betätigt, ist die Zeichenkette eine Eingabe in BASIC.

Zu Anfang 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 Eingabetaste an.

Durch KEY ON wird der Inhalt dieser Tasten in der 25. Zeile angezeigt. Ist die Breite 40, werden fünf 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 für die Anzeige der Funktionstasten.

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

KEY
Anweisung

KEY LIST zeigt den Inhalt aller zehn Funktionstasten auf dem Bildschirm 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 Länge von einem bis 15 Zeichen haben. Ist der Inhalt länger als 15 Zeichen, werden nur die ersten 15 Zeichen zugeordnet.

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

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

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

Hinweis: Um Komplikationen für Eingabepuffer des Kassetten-BASIC zu vermeiden, muß die folgende Anweisung ausgeführt werden:

```
DEF SEG: POKE 1Ø6,Ø
```

Dies muß nach jeder Neuzuordnung für eine Funktionstaste und nachdem INKEY$ das letzte Zeichen, das aus einer Zeichenkette für eine Funktionstaste gelesen werden soll, übergeben hat, ausgeführt werden. POKE ist im Disk- und erweiterten BASIC nicht erforderlich.

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

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

KEY
Anweisung

Beispiel:

```
5Ø KEY ON
```

zeigt die Funktionstasten auf der 25. Zeile an.

```
2ØØ KEY OFF
```

löscht die Anzeige für die Funktionstasten. Die Funktionstasten sind immer noch aktiv, werden aber nicht angezeigt.

```
1Ø KEY 1,"FILES"+CHR$(13)
```

ordnet die Zeichenkette "FILES" und die Eingabetaste der Funktionstaste 1 zu. Dies ist ein Weg, um einen häufig benutzten Befehl einer Funktionstaste zuzuordnen.

```
2Ø KEY 1,""
```

inaktiviert die Funktionstaste 1 als Funktionstaste.

KEY(n) Anweisung

Zweck: Aktiviert und inaktiviert die Unterbrechung für eine angegebene Taste in einem BASIC-Programm. Siehe "Anweisung ON KEY(n)" in diesem Kapitel.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format:

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	Positionsanzeiger nach oben
12	Positionsanzeiger nach links
13	Positionsanzeiger nach rechts
14	Positionsanzeiger nach unten

KEY(n) Anweisung

KEY(*n*) ON muß ausgeführt werden, um das Unterbrechen für eine Funktionstaste oder für eine Taste zur Steuerung des Positionsanzeigers zu aktivieren. Nach der Ausführung der Anweisung KEY(*n*) ON wird - falls eine Zeilennummer ungleich Null in dieser Anweisung angegeben wurde - jedesmal wenn BASIC eine neue Anweisung ausführt, geprüft, ob die angegebene Taste betätigt wurde. Wurde eine dieser Tasten betätigt, wird ein GOSUB zu der Zeilennummer ausgeführt, die in der Anweisung ON KEY(*n*) angegeben wurde.

Wird KEY(*n*) OFF ausgeführt, findet keine Unterbrechung statt, und wenn die Taste betätigt wurde, wird dies nicht bemerkt.

Wurde die Anweisung KEY(*n*) STOP ausgeführt, findet keine Unterbrechung statt. Wird jedoch eine der angegebenen Tasten betätigt, findet eine sofortige Unterbrechung statt, wenn die Anweisung KEY(*n*) ON ausgeführt wird.

KEY(n) Anweisung

KEY(*n*) ON hat keinen Einfluß auf die Anzeige der Funktionstastenwerte auf der untersten Bildschirmzeile.

Wird die Anweisung KEY(*n*) im Kassetten-oder Disk-BASIC benutzt, erhält man einen "Syntax Error" (Syntaxfehler). Im vorherigen Abschnitt "Anweisung KEY" steht die Erklärung über die Anweisung KEY ohne (*n*).

KILL Befehl

Zweck: Löscht eine Datei von einer Diskette. Der Befehl KILL in BASIC ist ähnlich dem Befehl ERASE im DOS.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: KILL *Dateiangabe*

Bemerkungen: *Dateiangabe*

Gültige Dateiangabe, wie unter "Namensgebung für Dateien" in Kapitel 3 beschrieben. Der Einheitenname muß das Laufwerk einer Diskette sein. Fehlt der Name der Einheit, wird das Standardlaufwerk von DOS benutzt.

KILL Befehl

KILL kann für alle Diskettendateientypen verwendet werden. Der Name muß die Erweiterung enthalten, falls eine solche existiert. Man kann z. B. ein BASIC-Programm mit Hilfe des Befehls

```
SAVE "TEST"
```

speichern. BASIC liefert dafür die Erweiterung **.BAS** für den Befehl SAVE, aber nicht für den Befehl KILL.

Soll die Programmdatei später gelöscht werden, muß eingegeben werden

KILL "TEST.BAS"

und nicht

KILL "TEST"

Wird der Befehl KILL für eine Datei gegeben, die gerade eröffnet ist, erhält man den Fehler "File Already Open" (Datei schon eröffnet).

Beispiel:

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

In diesem Beispiel wird die Datei DATA1 auf der Einheit A: gelöscht.

LEFT$ Funktion

Zweck: Übergibt die am weitesten links liegenden *n*-Zeichen aus *x$*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *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* größer als LEN(*x$*), wird die gesamte Zeichenkette (*x$*) übergeben. Ist *n* gleich Null, wird eine leere Zeichenkette (Länge Null) übergeben.

Siehe auch die Funktionen MID$ und RIGHT$.

LEFT$ Funktion

Beispiel:

```
Ok
1Ø A$ = "BASIC PROGRAM"
2Ø B$ = LEFT$(A$,5)
3Ø PRINT B$
RUN
BASIC
Ok
```

In diesem Beispiel werden mit der Funktion LEFT$ die ersten fünf Zeichen aus der Zeichenkette "BASIC PROGRAM" herausgenommen.

LEN
Funktion

Zweck: Übergibt die Anzahl der Zeichen in *x$*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = LEN(*x$*)

Bemerkungen: *x$* ist ein Zeichenkettenausdruck.

Nicht druckbare Zeichen und Leerzeichen sind in der Anzahl der Zeichen enthalten.

Beispiel:

```
10 X$ = "BOCA RATON, FL"
20 PRINT LEN(X$)
RUN
 14
Ok
```

In der Zeichenkette "BOCA RATON, FL" sind 14 Zeichen enthalten, weil das Komma und die Leerstelle mitgezählt werden.

LET Anweisung

Zweck: Ordnet den Wert eines Ausdrucks einer Variablen zu.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: [LET] *Variable=Ausdruck*

Bemerkungen: *Variable* Name einer Variablen oder eines Bereichselements, das einen Wert erhalten soll. Es kann sich um numerische Variablen und Bereichselemente und um alphanumerische Variablen und Bereichselemente handeln.

Ausdruck Ausdruck, dessen Wert einer *Variablen* zugeordnet werden soll. Der Typ des Ausdrucks (Zeichenkette oder numerisch) muß mit dem Typ der Variablen übereinstimmen; andernfalls erhält man den Fehler "Typ Mismatch" (Keine Typübereinstimmung).

Das Wort LET ist wahlfrei. Das bedeutet, daß es genügt, ein Gleichheitszeichen anzugeben, wenn Ausdrücke einem Variablennamen zugeordnet werden sollen.

LET Anweisung

Beispiel:

```
11Ø LET DORI=12
12Ø LET E=DORI+2
13Ø LET FDANCE$="HORA"
```

In diesem Beispiel wird der Wert 12 der Variablen DORI zugeordnet. Dann wird der Wert 14, d.h. der Wert aus dem Ausdruck DORI+2, der Variablen E zugeordnet. Die Zeichenkette "HORA" wird der Variablen FDANCE$ zugeordnet.

Diese Anweisungen hätte man auch wie folgt schreiben können:

```
11Ø DORI= 12
12Ø E =DORI+2
13Ø FDANCE$ = "HORA"
```

LINE Anweisung

Zweck: Zeichnet eine Linie oder ein Viereck auf dem Bildschirm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Nur im Grafikmodus.

Format: LINE [(*x1,y1*)] –(*x2,y2*) [,[*Farbe*] [,B[F]]]

Bemerkungen: *(x1,y1), (x2,y2)*
sind die Koordinaten entweder in absoluter oder relativer Form. (Siehe "Spezifikation von Koordinaten" unter "Grafischer Modus" in Kapitel 3.)

LINE
Anweisung

Farbe Farbnummer im Bereich 0 bis 3. In der mittleren Auflösung wählt *Farbe* die Farbe aus der laufenden Palette aus, wie in der Anweisung COLOR definiert. 0 ist die Hintergrundfarbe. Die Standardannahme ist die Vordergrundfarbe, Farbe Nummer 3. In der hohen Auflösung zeigt *Farbe* gleich 0 schwarz und die Standardannahme 1 weiß an.

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.

Man kann auch einen Anfangspunkt einfügen:

```
LINE (Ø,Ø)-(319,199) 'Bildschirmdiagonale nach unten
LINE (Ø,1ØØ)-(319,1ØØ) 'Bildschirmquerstrich
```

Im nächsten Beispiel wird die Farbe angegeben, in der die Linie gezeichnet werden soll:

```
LINE (1Ø,1Ø)-(2Ø,2Ø),2 'zeichnen in Farbe 2

1 'zeichnen Zufallszeilen in Zufallsfarben
1Ø SCREEN 1,Ø,Ø,Ø: CLS
2Ø LINE -(RND*319,RND*199),RND*4
3Ø GOTO 2Ø

1 'wechselndes Bild - Zeile ein, Zeile aus
1Ø SCREEN 1,Ø,Ø,Ø: CLS
2Ø FOR X=0 to 319
3Ø LINE (X,0)+(X,199),X AND 1
4Ø NEXT
```

Das letzte Argument für LINE ist B - Viereck oder BF - gefülltes Viereck. Man kann *Farbe* weglassen und das letzte Argument einfügen:

```
LINE (Ø,Ø)-(1ØØ,1ØØ),,B 'Viereck im Vordergrund
```

Man kann auch die Farbe einfügen:

```
LINE (Ø,Ø)-(1ØØ,1ØØ),2,BF 'Viereck in Farbe 2
```

B sagt BASIC, daß ein Rechteck mit den Punkten (*x1,y1*) und (*x2,y2*) als gegenüberliegende Ecken gezeichnet werden soll. Dadurch vermeidet man, die folgenden vier Befehle LINE angeben zu müssen:

```
LINE (X1,Y1)-(X2,Y1)
LINE (X1,Y1)-(X1,Y2)
LINE (X2,Y1)-(X2,Y2)
LINE (X1,Y2)-(X2,Y2)
```

Damit wird die gleiche Funktion ausgeführt.

Durch die Angabe **BF** wird das gleiche Rechteck wie mit **B** gezeichnet; zusätzlich wird der Inhalt des Rechtecks mit der ausgewählten Farbe aufgefüllt.

Falls Koordinaten, die sich nicht im Bereich befinden, der Anweisung LINE zugeordnet werden, wird der Koordinaten, die sich nicht innerhalb des Bereichs befindet, der nächste gültige Wert zugeordnet. Das heißt negative Werte werden zu Null und Y-Werte größer als 199 werden auf 199 gesetzt. X-Werte größer als 639 werden auf 639 gesetzt. X-Werte größer als 319 in der mittleren Auflösung gehen auf die nächste horizontale Zeile.

Der letzte angesprochene Punkt nach einer Anweisung LINE ist der Punkt (*x2,y2*). Wird für die zweite Koordinate die relative Form gewählt, ist sie relativ zur ersten Koordinaten. Beispiel:

```
LINE (1ØØ,1ØØ)-STEP (1Ø,-2Ø)
```

Damit wird eine Linie von (100,100) zu (110,80) gezeichnet.

Beispiel: In diesem Beispiel werden gefüllte Zufallsrechtecke mit Zufallszahlen gezeichnet:

```
1Ø CLS
2Ø SCREEN 1,Ø: COLOR Ø,Ø
3Ø LINE -(RND*319,RND*199),RND*2+1,BF
4Ø GOTO 3Ø 'Rechtecke überlappen sich
```

LINE INPUT
Anweisung

Zweck: Liest eine gesamte Zeile (bis zu 254 Zeichen) von der Tastatur in eine Zeichenkettenvariable und ignoriert Trennzeichen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: 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 Bereichselements für Zeichenketten, dem die Zeile zugeordnet wird. Die ganze Eingabe ab Ende der Anfrage bis zum Betätigen der Eingabetaste wird der *Zeichenkettenvariablen* zugeordnet. Nachführende Leerstellen werden ignoriert.

LINE INPUT
Anweisung

Im Disk- und erweiterten BASIC wird -falls dem LINE INPUT sofort ein Semikolon (;) folgt und danach die Eingabetaste betätigt wird - dem Ende der Eingabezeile nicht die Zeichenfolge Wagenrücklauf/Zeilenvorschub angefügt. Das bedeutet, daß der Positionsanzeiger auf der gleichen Zeile bleibt wie die Eingabe.

Man kommt aus der Anweisung LINE INPUT durch Betätigen der Tasten Ctrl-Break heraus. BASIC kehrt zur Befehlsebene zurück und zeigt **Ok** an. Durch die Eingabe CONT kann man die Ausführung von LINE INPUT wieder aufnehmen.

Beispiel: Ein Beispiel steht im nächsten Abschnitt unter "Anweisung LINE INPUT #."

LINE INPUT
Anweisung

Zweck: Liest eine ganze Zeile (bis zu 254 Zeichen), Trennzeichen ignorierend, von einer sequentiellen Datei in eine Zeichenkettenvariable.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: LINE INPUT *#Dateinummer, Zeichenkettenvariable*

Bemerkungen: *Dateinummer*
Nummer, unter der die Datei eröffnet wurde.

Zeichenkettenvariable
Name einer Zeichenkettenvariablen oder eines Bereichselements, dem die Zeile zugeordnet werden soll.

LINE INPUT # Anweisung

LINE INPUT # liest alle Zeichen aus einer sequentiellen Datei bis zum Zeichen Wagenrücklauf. Dann wird die Zeichenfolge Wagenrücklauf/Zeilenvorschub übersprungen, und mit der nächsten Anweisung LINE INPUT # werden die nächsten Zeichen bis zum nächsten Wagenrücklauf gelesen. (Wird die Zeichenfolge Zeilenvorschub/Wagenrücklauf gefunden, wird sie konserviert; das bedeutet, die Zeichen für Zeilenvorschub/Wagenrücklauf werden als Teil der Zeichenkette übergeben.)

LINE INPUT # ist speziell dann nützlich, 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 für Dateien mit wahlfreiem Zugriff benutzt werden. Siehe Anhang B. "BASIC-Diskettenein- und -ausgabe."

LINE INPUT
Anweisung

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.

```
Ok
10 OPEN "LIST" FOR OUTPUT AS #1
20 LINE INPUT "Adresse?";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

Adresse?
```

Wird z.B. mit DELRAY BEACH, FL 33445, geantwortet, macht das Programm wie folgt weiter.

```
.
.
.
Adresse? DELRAY BEACH, FL      33445

DELRAY BEACH, FL      33445
Ok
```

LIST Befehl

Zweck: Listet das sich gerade im Hauptspeicher befindliche Programm auf dem Bildschirm oder auf einer anderen angegebenen Einheit an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	

Format: LIST [*Zeile1*] [-[*Zeile2*]] [,*Dateiangabe*]

Bemerkungen: *Zeile1, Zeile2*

Gültige 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 für die Dateispezifikation, wie unter "Namensgebung für Dateien" in Kapitel 3 erklärt. Wird *Dateiangabe* weggelassen, werden die angegebenen Zeilen auf dem Bildschirm angezeigt.

LIST Befehl

Im Kassetten-BASIC kann das Anzeigen auf dem Bildschirm durch Weglassen der Einheitenspezifikation dadurch gestoppt werden, daß man die Tasten Ctrl-Break betätigt. Ein Auflisten auf einer angegebenen Einheit kann nicht unterdrückt werden. Das bedeutet, LIST *Bereich* kann unterbrochen werden, aber LIST *Bereich*, "SCRN:" nicht.

Im Disk- und erweiterten BASIC kann das Auflisten auf dem Bildschirm oder dem Drucker mit Hilfe der Tasten Ctrl-Break unterbrochen werden.

Wird der Zeilenbereich nicht angegeben, wird das gesamte Programm aufgelistet.

Wird ein Bindestrich (-) im Zeilenbereich angegeben, sind die folgenden drei Auswahlen verfügbar:

- Wird nur *Zeile1* angegeben, werden diese Zeile und alle Zeilen mit höheren Nummern aufgelistet.
- Wird nur *Zeile2* angegeben, werden alle Zeilen vom Beginn des Programms bis zur *Zeile2* aufgelistet.
- Werden beide Zeilennummern angegeben, werden alle Zeilen von *Zeile1* bis *Zeile2* je einschließlich aufgelistet.

LIST Befehl

Erfolgt die Auflistung in eine Datei auf Kassette oder Diskette, wird der angegebene Teil des Programms im ASCII-Format gespeichert. Diese Datei kann später mit MERGE benutzt werden.

Beispiel:

```
LIST
```

zeigt das gesamte Programm auf dem Bildschirm an.

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

zeigt Zeile 35 auf dem Bildschirm an.

```
LIST 1Ø-2Ø, "LPT1:"
```

listet die Zeilen 10 bis 20 auf dem Drucker auf.

```
LIST 1ØØ- ,"COM1:12ØØ,N,8"
```

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

```
LIST -2ØØ,"CAS1:BOB"
```

überträgt die erste Zeile bis zur Zeile 200 in eine Datei mit dem Namen BOB auf Kassette.

LLIST Befehl

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: LLIST [*Zeile1*][– [*Zeile2*]]

Bemerkungen: Der Zeilennummernbereich für LLIST arbeitet wie LIST.

Im Kassetten-BASIC kann LLIST nicht durch Ctrl-Break unterbrochen werden. Das Auflisten kann nur durch Ausschalten des Druckers gestoppt werden.

BASIC steht in der Befehlsebene, nachdem LLIST ausgeführt wurde.

LLIST Befehl

Beispiel:

```
LLIST
```

druckt eine Liste des gesamten Programms.

```
LLIST 35
```

druckt die Zeile 35.

```
LLIST 1Ø-2Ø
```

listet die Zeilen 10 bis 20 auf dem Drucker auf.

```
LLIST 1ØØ-
```

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

```
LLIST -2ØØ
```

druckt ab der ersten Zeile bis zur Zeile 200.

LOAD Befehl

Zweck: Lädt ein Programm von einer angegebenen Einheit in den Hauptspeicher und führt es wahlweise aus.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: LOAD *Dateiangabe*[,R]

Bemerkungen: *Dateiangabe*
Zeichenkettenausdruck für die Dateispezifikation, der den Regeln entsprechen muß, die unter "Namensgebung für Dateien" in Kapitel 3 beschrieben sind; anderfalls wird ein Fehler angezeigt und das Laden abgebrochen.

LOAD schließt alle eröffneten Dateien ab und löscht alle Variablen und Programmzeilen, die sich im Hauptspeicher befinden, bevor das angegebene Programm geladen wird. Wird die Angabe R weggelassen, geht BASIC in den Direktmodus, nachdem das Programm geladen wurde.

Wird jedoch die Angabe **R** im Befehl LOAD angegeben, wird das Programm nach dem Laden ausgeführt. In diesem Fall bleiben alle eröffneten Dateien offen. Das bedeutet, daß LOAD mit der Angabe **R** dazu benutzt werden kann, mehrere Programme zu verketten (oder Segmente des gleichen Programms). Informationen können zwischen den Programmen mit Hilfe von Datendateien übergeben werden.

LOAD *Dateiangabe*,R ist gleichbedeutend mit RUN *Dateiangabe*.

Wird Kassetten-BASIC benutzt und der Einheitenname weggelassen, wird CAS1: angenommen. CAS1: ist die einzige erlaubte Einheit für LOAD im Kassetten-BASIC.

Wird mit Disk- oder erweitertem BASIC gearbeitet, wird – falls die Einheit nicht angegeben ist - die DOS-Standarddisketteneinheit genommen. Die Erweiterung **.BAS** wird an den Dateinamen angefügt, falls keine Erweiterung angegeben ist und der Dateiname aus acht oder weniger Buchstaben besteht.

LOAD
Befehl

Hinweise für die Benutzung von CAS1:

1. Wird der Befehl LOAD im direkten Modus eingegeben, werden die Dateinamen auf dem Band, gefolgt von einem Punkt (.) und einem einzelnen Buchstaben, der den Dateityp angibt, auf dem Bildschirm angezeigt. Darauf folgt die Nachricht "Skipped" (Übergangen) für die Dateien, die nicht gleich dem angegebenen Dateinamen sind, und "Found" (Gefunden), falls der Dateiname gefunden wird. Dateitypen und ihre zugehörigen Buchstaben sind:

   .B BASIC-Programme im internen Format (erstellt mit dem Befehl SAVE)
   .P Geschützte BASIC-Programme im internen Format (erstellt mit dem Befehl SAVE ,P)
   .A BASIC-Programme im ASCII-Format (erstellt mit dem Befehl SAVE ,A)
   .M Dateien im Hauptspeicherabbildformat (erstellt mit dem Befehl BSAVE)
   .D Datendateien (erstellt mit OPEN, gefolgt von Ausgabeanweisungen)

Falls man sehen will, welche Dateien auf einer Kassette gespeichert sind, spult man das Band an den Anfang zurück und gibt einen Namen ein, von dem man weiß, daß er nicht auf dem Band steht, z.B. **LOAD "CAS1:Nicht"**. Dann werden alle Dateinamen angezeigt.

Wird der Befehl LOAD in einem BASIC-Programm ausgeführt, werden die übergangenen und gefundenen Dateinamen am Bildschirm nicht angezeigt.

2. Man muß beachten, daß die Tasten Ctrl-Break jederzeit während LOAD betätigt werden können. Zwischen Dateien oder nach einer Time-out-Periode verzweigt BASIC aus dem Suchen und kehrt zur Befehlsebene zurück. Vorherige Hauptspeicherinhalte bleiben unverändert.

3. Wird CAS1: als Einheit angegeben und der Dateiname fehlt, wird die nächste Programmdatei auf dem Band geladen.

LOAD Befehl

Beispiel:

```
LOAD "MENU"
```

lädt das Programm mit dem Namen MENU, führt es aber nicht aus.

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

lädt das Programm INVENT und führt es aus.

```
RUN "INVENT"
```

wie LOAD "INVENT",R.

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

lädt die Datei REPORT.BAS von der Disketteneinheit B. (**.BAS** braucht nicht angegeben zu werden.)

```
LOAD "CAS1:"
```

lädt das nächste auf dem Band befindliche Programm.

LOC
Funktion

Zweck: Übergibt die laufende Position in der Datei.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: *v* = LOC(*Dateinummer*)

Bemerkungen: *Dateinummer*
Dateinummer, die zur Eröffnung der Datei benutzt wurde.

Für Dateien mit wahlfreiem Zugriff übergibt LOC die Satznummer des letzten gelesenen oder geschriebenen Satzes.

Bei sequentiellen Dateien übergibt LOC die Anzahl der gelesenen Sätze der Datei seit sie eröffnet wurde. (Ein Satz ist ein Datenblock aus 128 Bytes.) Ist eine Datei für sequentielle Eingabe eröffnet, liest BASIC den ersten Sektor der Datei, so daß LOC eine 1 übergibt, bevor irgendeine Eingabe von der Datei gelesen wurde.

LOC
Funktion

Für eine Datenfernverarbeitungsdatei übergibt LOC die Anzahl der Zeichen im Eingabepuffer, die darauf warten gelesen zu werden. Die Standardgröße für den Eingabepuffer ist 256 Zeichen. Dies kann aber mit der Auswahl /C: in dem Befehl BASIC geändert werden. Stehen mehr als 255 Zeichen im Puffer, übergibt LOC 255. Da eine Zeichenkette auf 255 Zeichen beschränkt ist, erleichtert diese praktische Grenze den Test der Zeichenkettenlänge, bevor Daten eingelesen werden. Bleiben weniger als 255 Zeichen im Puffer, übergibt LOC den aktuellen Zähler.

Beispiele:

```
200 IF LOC(1)>50 THEN STOP
```

Im ersten Beispiel stoppt das Programm, falls ein Satz nach dem 50. Satz der Datei gelesen wurde.

```
300 PUT #1,LOC(1)
```

Das zweite Beispiel kann benutzt werden, um den gerade gelesenen Satz zu überschreiben.

LOCATE
Anweisung

Zweck: Setzt den Positionsanzeiger auf dem aktiven Bildschirm. Wahlfreie Parameter schalten das Blinken des Positionsanzeigers an und aus und definieren die Größe des blinkenden Positionsanzeigers.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: LOCATE [*Zeile*][,[*Spalte*] [,[*Positionsanzeiger*][,[*Start*][,*Stopp*]]]]

Bemerkungen:

Zeile Numerischer Ausdruck im Bereich 1 bis 25. Er zeigt die Bildschirmzeilennummer an, an die der Positionsanzeiger gebracht werden soll.

Spalte Numerischer Ausdruck im Bereich 1 bis 40 oder 1 bis 80, abhängig von der Bildschirmbreite. Er zeigt die Bildschirmspaltennummer an, an die der Positionsanzeiger gebracht werden soll.

LOCATE
Anweisung

Positionsanzeiger
: Wert, der angibt, ob der Positionsanzeiger sichtbar sein soll. Eine 0 (Null) zeigt Aus an, eine 1 (Eins) zeigt Ein an.

Start
: Beginnende Suchzeile für den Positionsanzeiger. Es muß ein numerischer Audruck im Bereich 0 bis 31 sein.

Stopp
: Stoppsuchzeile für den Positionsanzeiger. Es muß sich auch um einen numerischen Ausdruck im Bereich 0 bis 31 handeln.

Positionsanzeiger, Start und *Stopp* können nicht im grafischen Modus angewendet werden.

LOCATE
Anweisung

Mit *Start* und *Stopp* kann der Positionsanzeiger jede gewünschte Größe annehmen. Man gibt damit die beginnenden und endenden Suchzeilen an. Die Suchzeilen werden ab 0 (Null) über der Zeichenposition numeriert. Die unterste Suchzeile ist 7 mit dem Farb-/Grafikbildschirmanschluß, 13 mit dem IBM Schwarz/Weiß-Bildschirm- und Paralleldruckeranschluß. Wird *Start* angegeben und *Stopp* weggelassen, wird für *Stopp* der Wert von *Start* angenommen. Ist *Start* größer als *Stopp*, erhält man einen zweiteiligen Positionsanzeiger. Der Positionsanzeiger geht von der unteren Zeile zurück zur oberen.

Nach der Anweisung LOCATE werden Zeichen in Ein-/Ausgabeanweisungen für den Bildschirm an die angegebene Stelle gebracht.

Läuft ein Programm ab, ist der Positionsanzeiger normalerweise ausgeschaltet - mit LOCATE ,,1 kann er eingeschaltet 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: PRINT... angeben, um Texte in Zeile 25 einzugeben.

LOCATE
Anweisung

Jeder Parameter kann weggelassen werden. Für weggelassene Parameter wird der laufende Wert übernommen.

Jeder Wert, der sich nicht innerhalb der angegebenen Bereiche befindet, ergibt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Die vorherigen Werte bleiben dann erhalten.

Beispiele:

```
10 LOCATE 1,1
```

bringt den Positionsanzeiger in seine Heimposition in der oberen linken Ecke des Bildschirms.

```
20 LOCATE ,,1
```

macht den blinkenden Positionsanzeiger sichtbar. Seine Position bleibt unverändert.

```
30 LOCATE ,,,7
```

Die Position und die Sichtbarkeit des Positionsanzeigers bleiben unverändert. Setzt den Positionsanzeiger so, daß er im unteren Ende des Zeichens auf dem Farb-/Grafikbildschirm angezeigt wird (Start und Ende in der Suchzeile 7).

```
4Ø LOCATE 5,1,1,Ø,7
```

Bringt den Positionsanzeiger in Zeile 5, Spalte 1. Macht den Positionsanzeiger sichtbar und überdeckt das ganze Zeichen auf dem Farb-/Grafikbildschirm, beginnend mit Suchzeile 0 (Null) und endend mit Suchzeile 7.

LOF
Funktion

Zweck: Übergibt die Anzahl der Bytes, die einer Datei zugeordnet sind (Länge der Datei).

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: *v* = LOF(*Dateinummer*)

Bemerkungen: *Dateinummer*
Dateinummer, die benutzt wurde, um die Datei zu eröffnen.

Für Diskettendateien, die mit BASIC erstellt wurden, übergibt LOF ein Mehrfaches von 128. Sind z.B. die aktuellen Daten in einer Datei 257 Bytes lang, wird die Zahl 384 übergeben. Für Diskettendateien, die nicht mit BASIC erstellt wurden (z.B. mit Hilfe von EDLIN), übergibt LOF die aktuelle Anzahl der Bytes, die der Datei zugeordnet wurden.

LOF Funktion

Für Datenfernverarbeitung übergibt LOF die Anzahl der freien Stellen im Eingabepuffer. Das heißt, *Größe*-LOC(*Dateinummer*), wobei *Größe* die Größe des Datenfernverarbeitungspuffers ist, dessen Standardannahme 256 Bytes ist, die aber durch die Auswahl /**C:** im BASIC-Befehl geändert werden kann. Mit Hilfe von LOF kann geprüft werden, wann der Eingabepuffer voll wird. Dies wird in einem Beispiel in Anhang F. "Datenfernverarbeitung" gezeigt.

Beispiel: Mit folgenden Anweisungen wird der letzte Satz der Datei BIG gelesen, wobei angenommen wird, daß BIG mit einer Satzlänge von 128 Bytes erstellt wurde:

```
10 OPEN "BIG" AS #1
20 GET #1,LOF(1)/128
```

LOG
Funktion

Zweck: Übergibt den natürlichen Logarithmus von *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *v* = LOG(*x*)

Bemerkungen: *x* muß ein numerischer Ausdruck sein, der größer als 0 (Null) ist.

Der natürliche 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 und e^2 errechnet:

```
Ok
E= 2.718282
Ok
? LOG(E)
 1
Ok
? LOG(E*E)
 2
Ok
```

LPOS
Funktion

Zweck: Übergibt die laufende Position des Druckkopfs im Druckpuffer für LPT1:.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = LPOS(*n*)

Bemerkungen: *n* ist ein numerischer Ausdruck, der im Kassetten-BASIC aus einem Scheinargument besteht. Im Disk- und erweiterten BASIC wird mit *n* der zu testende Drucker angezeigt:

0 oder 1	LPT1:
2	LPT2:
3	LPT3:

Es wird empfohlen, im Kassetten-BASIC 0 oder 1 zu verwenden, um die Verträglichkeit mit anderenVersionen sicherzustellen.

Mit der Funktion LPOS wird nicht notwendigerweise die physische Position des Druckkopfs auf dem Drucker angezeigt.

Beispiel: Im folgenden Beispiel wird - falls die Zeilenlänge mehr als 60 Zeichen beträgt - ein Zeichen für Wagenrücklauf zum Drucker gesendet, so daß ein Vorschub zur nächsten Zeile ausgeführt wird.

```
100 IF LPOS(0)>60 THEN LPRINT CHR$(13)
```

LPRINT und LPRINT USING Anweisungen

Zweck: Druckt Daten auf dem Drucker (LPT1:).

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: LPRINT [*Liste von Ausdrücken*] [;]

LPRINT USING *v$; Liste von Ausdrücken* [;]

Bemerkungen: *Liste von Ausdrücken*
Liste der numerischen und/oder Zeichenvariablenausdrücke, die gedruckt werden sollen. Die Ausdrücke müssen 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" erklärt.

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

LPRINT und LPRINT USING Anweisungen

LPRINT nimmt eine Druckbreite von 80 Zeichen auf dem Drucker an. Das heißt, BASIC fügt automatisch einen Wagenrücklauf/Zeilenvorschub 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 für die Druckbreite kann mit der Anweisung WIDTH "LPT1:" geändert werden.

Das Drucken läuft asynchron zur Verarbeitung ab. Wird ein Seitenvorschub (LPRINT CHR$(12);), gefolgt von einem anderen LPRINT, ausgeführt und der Drucker benötigt mehr als 10 Sekunden für den Seitenvorschub, kann der Fehler "Device Timeout" (Einheitenzeitunterbrechung) für 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 zu Absicherung noch ERL testen, um zu sehen, ob die Zeitunterbrechung durch die Anweisung LPRINT entstand.

LPRINT und LPRINT USING Anweisungen

Beispiel: In diesem Beispiel werden an den IBM Matrixdrucker mit Hilfe von LPRINT und CHR$ spezielle Steuerzeichen übergeben. Diese Drucksteuerzeichen sind im *IBM Personalcomputer Technisches Handbuch* 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 erhält man die folgende Ausgabe:

```
                Titelzeile
Berichtszeile 2
Berichtszeile 3
Berichtszeile 4
Komprimiertes Drucken; 132 Zeichen/Zeile
Zurueck zu Normaldruck

Dies ist verstärktes Drucken

Wieder zurueck zu Normaldruck
```

LSET und RSET Anweisungen

Zweck: Übergibt Daten in einen Dateipuffer für wahlfreien Zugriff (als Vorbereitung auf die Anweisung PUT (Datei)).

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: LSET *Zeichenkettenvariable* = *x$*

RSET *Zeichenkettenvariable* = *x$*

Bemerkungen: *Zeichenkettenvariable*
Name einer Variablen, die in der Anweisung FIELD definiert wurde.

x$ Zeichenkettenausdruck für die Information, die in das durch die *Zeichenkettenvariable* identifizierte Feld gebracht werden soll.

Benötigt *x$* weniger Bytes als für die *Zeichenkettenvariable* in der Anweisung FIELD angegeben wurden, wird durch LSET die Zeichenkette linksbündig im Feld gespeichert - mit RSET rechtsbündig. (Die Extrastellen werden durch Leerstellen aufgefüllt.) Ist *x$* länger als die *Zeichenkettenvariable*, werden die Zeichen rechtsbündig abgeschnitten.

LSET und RSET Anweisungen

Numerische Werte müssen in Zeichenketten umgewandelt werden, bevor sie mit LSET oder RSET verarbeitet werden können. Siehe "Funktionen MKI$, MKS$, MKD$" in diesem Kapitel.

Anhang B. "BASIC-Diskettenein- und -ausgabe" enthält eine vollständige Erklärung für die Benutzung von Dateien mit wahlfreiem Zugriff.

Hinweis: LSET und RSET können auch mit einer Zeichenkettenvariablen benutzt werden, die nicht in einer Anweisung FIELD definiert war, um eine Zeichenkette in einem gegebenen Feld rechts- oder linksbündig anzuordnen. Beispiel:

```
11Ø A$=SPACE$(2Ø)
12Ø RSET A$=N$
```

Die Programmzeilen übergeben die Zeichenkette N$ rechtsbündig 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 linksbündig in dem Feld A$ zur Vorbereitung für eine Anweisung PUT (Datei) gespeichert.

```
15Ø LSET A$=MKS$(AMT)
```

MERGE Befehl

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	

Format: MERGE *Dateiangabe*

Bemerkungen: *Dateiangabe*
Zeichenkettenausdruck für eine Dateispezifikation, der den Regeln für die Namensgebung von Dateien, wie unter "Namensgebung für Dateien" in Kapitel 3 erklärt, entsprechen muß; andernfalls wird ein Fehler angezeigt und MERGE abgebrochen.

Die Einheit wird nach dem Namen der Datei abgesucht. Sobald die Datei gefunden ist, werden die Programmzeilen in der Einheitendatei mit den Zeilen im Hauptspeicher vermischt. Haben Zeilen aus der Datei, mit der vermischt werden soll, die gleichen Zeilennummern wie Zeilen des Programms im Hauptspeicher, ersetzen die Zeilen der Datei die Zeilen im Hauptspeicher.

MERGE
Befehl

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

Wird im Kassetten-BASIC der Einheitenname weggelassen, wird CAS1: angenommen. CAS1: ist die einzig erlaubte Einheit für MERGE im Kassetten-BASIC. Wird im Disk- oder erweiterten BASIC der Einheitenname weggelassen, wird die Standardeinheit für DOS angenommen.

Wird CAS1: als Einheitenname angegeben und der Dateiname weggelassen, wird mit der nächsten ASCII-Programmdatei, die auf dem Band gefunden wird, vermischt.

Wurde das Programm, mit dem vermischt werden soll, nicht im ASCII-Format gespeichert (mit Hilfe der Angabe A im Befehl SAVE), erhält man den Fehler "Bad File Mode" (Falscher Dateityp). Das Programm im Hauptspeicher wird nicht verändert.

Beispiel:

```
MERGE "A:NUMBRS"
```

Damit wird die Datei dem Namen NUMBRS auf Einheit A: mit dem Programm im Hauptspeicher vermischt.

MID$
Funktion und Anweisung

Zweck: Übergibt den angeforderten Teil einer gegebenen Zeichenkette. Wenn - wie im zweiten Format - als Anweisung benutzt, wird ein Teil einer Zeichenkette durch eine andere Zeichenkette ersetzt.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: Als Funktion:

*v$* = MID$(*x$*,*n*[,*m*])

Als Anweisung:

MID$(*v$*,*n*[,*m*]) = *y$*

Bemerkungen: **Für 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.

MID$
Funktion und Anweisung

Die Funktion übergibt eine Zeichenkette der Länge *m* Zeichen aus *x$*, beginnend mit dem *n*. Zeichen. Wird *m* weggelassen oder stehen weniger als *m* Zeichen rechts des *n*. Zeichens, werden alle rechtsbündigen Zeichen, beginnend mit dem *n*. Zeichen übergeben. Ist *m* gleich Null (0) oder ist *n* größer als LEN(*x$*), übergibt MID$ eine leere Zeichenkette.

Siehe auch Funktionen LEFT$ und RIGHT$.

Für 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.

MID$
Funktion und Anweisung

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$*, die für das Ersetzen benutzt werden. Wird *m* weggelassen, wird der ganze Inhalt von *y$* benutzt.

Jedoch wird, unabhängig davon, ob *m* weggelassen oder benutzt wird, die Länge von *v$* nicht verändert. Ist z.B. *v$* vier Zeichen lang und *y$* fünf Zeichen lang, enthält *v$* nach dem Ersetzen nur die ersten vier Zeichen von *y$*.

> **Hinweis:** Befinden sich *n* oder *m* nicht im angegebenen Bereich, erhält man den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

MID$
Funktion und Anweisung

Beispiel: Im ersten Beispiel wird mit der Funktion MID$ der mittlere Teil der Zeichenkette B$ ausgewählt.

```
Ok
10 A$="GUTEN"
20 B$=MORGEN TAG ABEND"
30 PRINT A$;MID$(B$,8,3)
RUN
GUTEN TAG
Ok
```

Im nächsten Beispiel werden mit Hilfe der Anweisung MID$ Zeichen in der Zeichenkette A$ ersetzt.

```
Ok
10 A$="MARATHON, GREECE"
20 MID$(A$,11)="FLORIDA"
30 PRINT A$
RUN
MARATHON, FLORID
Ok
```

Man beachte, daß im zweiten Beispiel die Länge von A$ nicht verändert wurde.

MKI$, MKS$, MKD$ Funktionen

Zweck: Wandelt numerische Werte in Zeichenkettenwerte um.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format:

*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 für wahlfreien Zugriff 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 ändern, sondern dadurch, wie BASIC diese Bytes interpretiert.

MKI$, MKS$, MKD$ Funktionen

Siehe auch "Funktionen CVI, CVS, CVD" in diesem Kapitel und Anhang B. "BASIC-Diskettenein- und -ausgabe".

Beispiel:

In diesem Beispiel wird eine Datei mit wahlfreiem Zugriff (#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 ungewandelt und mit LSET der aktuelle Wert von AMT in den Dateipuffer für wahlfreien Zugriff gespeichert. In Zeile 120 wird eine Zeichenkette in den Puffer gebracht. In Zeile 130 werden die Daten aus dem Dateipuffer für wahlfreien Zugriff in die Datei geschrieben.

```
100 FIELD #1, 4 AS D$, 20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
```

MOTOR
Anweisung

Zweck: Schaltet den Motor des Kassettengeräts über ein Programm ein oder aus.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	

Format: MOTOR [*Status*]

Bemerkungen: *Status* Numerischer Ausdruck, der Ein oder Aus anzeigt.

Ist *Status* nicht Null (0), wird der Kassettenmotor eingeschaltet. Ist *Status* gleich Null (0), wird der Kassettenmotor ausgeschaltet.

Wird *Status* weggelassen, wird der Status für den Kassettenmotor umgedreht, d. h. ist der Motor aus, wird er eingeschaltet und umgekehrt.

Beispiel: In der folgenden Anweisungsfolge wird der Kassettenmotor eingeschaltet, ausgeschaltet, danach wieder eingeschaltet:

```
1Ø MOTOR 1
2Ø MOTOR Ø
3Ø MOTOR
```

NAME Befehl

Zweck: Ändert den Namen einer Diskettendatei. Der Befehl NAME in BASIC ist ähnlich dem Befehl RENAME in DOS.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: NAME *Dateiangabe* AS *Dateiname*

Bemerkungen: *Dateiangabe*

Dateispezifikation, wie unter "Namensgebung für Dateien" in Kapitel 3 beschrieben.

Dateiname

Neuer Dateiname. Es muß sich um einen gültigen Dateinamen handeln, wie in diesem Abschnitt beschrieben.

Die in *Dateiangabe* angegebene Datei muß existieren, und *Dateiname* darf nicht auf der Diskette vorhanden sein, sonst erhält man einen Fehler. Wird der Einheitenname weggelassen, wird für DOS die Standardeinheit angenommen. Es muß beachtet werden, daß für die Dateierweiterung nicht standardmäßig **.BAS** angehängt wird.

Nach der Ausführung des Befehls NAME steht die Datei auf derselben Diskette im gleichen Bereich unter einem neuen Namen.

Beispiel:

```
NAME "A:ACCT.BAS" AS "LEDGER.BAS"
```

Nach der Ausführung dieses Beispiels hat die Datei ACCTS.BAS auf der Diskette in Laufwerk A den Namen LEDGER.BAS.

NEW Befehl

Zweck: Löscht das im Hauptspeicher befindliche Programm und alle Variablen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: NEW

Bemerkungen: NEW: Wird normalerweise benutzt, um den Hauptspeicher zu löschen, bevor ein neues Programm eingegeben wird. BASIC kehrt immer in die Befehlsebene zurück, nachdem NEW ausgeführt wurde. Durch NEW werden alle Dateien geschlossen und die Programmablaufverfolgung ausgeschaltet, falls sie eingeschaltet war (siehe "Befehle TRON und TROFF" in diesem Kapitel).

Beispiel:

```
Ok
NEW
Ok
```

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

OCT$ Funktion

Zweck: Übergibt eine Zeichenkette, die den oktalen Wert eines dezimalen Arguments darstellt.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *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$" für hexadezimale Umwandlung.

Beispiel:

```
Ok
PRINT OCT$(24)
3Ø
Ok
```

Dieses Beispiel zeigt, daß 24 dezimal 30 in oktaler Schreibweise ist.

ON COM(n) Anweisung

Zweck: Setzt für BASIC eine Zeilennummer, zu der verzweigt werden soll, sobald eine Information im Datenfernverarbeitungspuffer steht.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format: 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 für die Datenfernverabeitungsaktivität für den angegebenen Anschluß inaktiviert.

ON COM(n) Anweisung

Die Anweisung COM(*n*) ON muß ausgeführt werden, um diese Anweisung für den Anschluß *n* zu aktivieren. Nach der Ausführung dieser Anweisung - falls die Zeilennummer nicht 0 (Null) war - prüft BASIC vor der Ausführung jeder neuen Anweisung, ob ein Zeichen in dem angegebenen Datenfernverabeitungsanschluß angekommen ist. Steht ein neues Zeichen zur Verfügung, verzweigt BASIC mit GOSUB zur angegebenen *Zeile.*

Wurde COM(*n*) OFF ausgeführt, findet keine Unterbrechung für den Anschluß statt. Falls über die Datenfernverarbeitungsleitung Zeichen übertragen werden, wird dies nicht notiert.

Wird die Anweisung COM(*n*) STOP ausgeführt, findet für den Anschluß keine Unterbrechung statt. Es werden jedoch alle empfangenen Zeichen gespeichert, so daß sofort eine Unterbrechung stattfindet, wenn COM(*n*) ON ausgeführt wird.

Erfolgt eine Unterbrechung, wird automatisch COM(*n*) STOP ausgeführt, so daß niemals eine rekursive Unterbrechung auftreten kann.

ON COM(n) Anweisung

Durch RETURN aus der Unterbrechungsroutine wird automatisch COM(*n*) ON ausgeführt, außer es wird explizit COM(*n*) OFF innerhalb der Unterbrechungsroutine ausgeführt.

Eine Unterbrechung kann nicht stattfinden, wenn BASIC kein Programm ausführt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung ON ERROR), werden alle Unterbrechungen automatisch inaktiviert (einschließlich ERROR, STRIG(n), PEN, COM(n) und KEY(n)).

Die Datenfernverarbeitungs-Unterbrechungsroutine liest eine gesamte Nachricht von der Datenfernverarbeitungsleitung, bevor zurückverzweigt wird. Es wird nicht empfohlen, die Datenfernverarbeitungsunterbrechung für Nachrichten aus einzelnen Buchstaben zu benutzen, da hohe Baud-Raten zu viel Zeit für die Unterbrechung und das Lesen jedes einzelnen Zeichens benötigen, so daß vielleicht der Unterbrechungspuffer für Datenfernverarbeitungen überlaufen kann.

ON COM(n) Anweisung

Man kann mit RETURN *Zeile* in einem BASIC-Programm zu einer festen Zeilennummer zurückverzweigen. Die Benutzung dieser Zurückverzweigung muß sehr vorsichtig gehandhabt werden, da andere GOSUBs, WHILEs oder FORs zur Zeit der Unterbrechung aktiv sind und aktiv bleiben.

Beispiel:

```
15Ø ON COM(1) GOSUB 5ØØ
16Ø COM(1) ON
 .
 .
 .
5ØØ REM ankommende Zeichen
 .
 .
 .
59Ø RETURN 3ØØ
```

In diesem Beispiel wird für den ersten Datenfernverarbeitungsanschluß eine Unterbrechungsroutine in Zeile 500 angelegt.

ON ERROR Anweisung

Zweck: Aktiviert die Fehlerunterbrechung und gibt die erste Zeile des Fehlerbehandlungsunterprogramms an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format: ON ERROR GOTO *Zeile*

Bemerkungen: *Zeile* Zeilennummer der ersten Zeile der Fehlerunterbrechungsroutine. Fehlt diese Zeilennummer, erhält man den Fehler "Undefined Line Number" (Nicht definierte Zeilennummer).

Wurde die Fehlerunterbrechung aktiviert, wird für jeden entdeckten Fehler (einschließlich Fehler im direkten Modus) in das angegebene Fehlerbehandlungsunterprogramm verzweigt.

Um die Fehlerunterbrechung zu inaktivieren, muß ON ERROR GOTO 0 ausgeführt werden. Für nachfolgende Fehler wird ein Fehler angezeigt und die Ausführung angehalten. Steht diese Anweisung im Unterprogramm für Fehlerbehandlung, hält BASIC an. Die Fehlernachricht für den Fehler, der die Unterbrechung verursachte, wird angezeigt. Es wird empfohlen, daß alle Fehlerbehandlungsunterprogramme den Befehl ON ERROR GOTO 0 ausführen, falls ein Fehler auftritt, für den kein Wiederherstellungsverfahren existiert.

> **Hinweis:** Tritt ein Fehler während der Ausführung des Fehlerbehandlungsunterprogramms auf, wird die BASIC-Fehlernachricht angezeigt und die Ausführung abgebrochen. Im Fehlerbehandlungsunterprogramm erfolgt keine Fehlerunterbrechung.

Mit Hilfe der Anweisung RESUME verzweigt man aus der Fehlerbehandlungsroutine. Siehe "Anweisung RESUME" in diesem Kapitel.

ON ERROR
Anweisung

Beispiel:

```
1Ø ON ERROR GOTO 1ØØ
2Ø LPRINT "Dies geht zum Drucker."
3Ø END
1ØØ IF ERR=27 THEN PRINT "Drucker prüfen"
    : RESUME
```

Dieses Beispiel zeigt, wie man einen häufigen Fehler abfangen kann, z. B. vergessen, Papier in den Drucker einzulegen, oder vergessen, ihn einzuschalten.

ON...GOSUB und ON...GOTO Anweisungen

Zweck: Verzweigt zu einer von mehreren angegebenen Zeilennummern, abhängig vom Wert des Ausdrucks.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

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

ON *n* GOSUB *Zeile*[,*Zeile*]...

Bemerkungen: *n* ist ein numerischer Ausdruck, der – falls nötig – zu einer ganzen Zahl gerundet wird. Er muß sich im Bereich 0 bis 255 befinden, oder man erhält den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

Zeile Zeilennummer einer Zeile, zu der verzweigt werden soll.

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

ON...GOSUB und ON...GOTO Anweisungen

In der Anweisung ON...GOSUB muß jede Zeilennummer in der Liste die erste Zeilennummer eines Unterprogramms sein. Eventuell benötigt man eine Anweisung RETURN, um das Programm zu der Zeile zurückzubringen, die der Anweisung ON...GOSUB folgt.

Ist der Wert von *n* 0 (Null) oder größer als die Anzahl der angegebenen Zeilennummern in der Liste (aber kleiner oder gleich 255), fährt BASIC mit der nächsten ausführbaren Anweisung fort.

Beispiele: Im ersten Beispiel wird zur Zeile 150 verzweigt, falls L-1 gleich 1, zur Zeile 300 – falls L-1 gleich 2, zur Zeile 320 – falls L-1 gleich 3, zur Zeile 390 – falls L-1 gleich 4 ist. Ist L-1 gleich 0 (Null) oder größer als 4, geht das Programm zur nächsten Anweisung.

```
100 ON L-1 GOTO 150,300,320,390
```

Im nächsten Beispiel wird gezeigt, wie die Anweisung ON...GOSUB benutzt wird:

```
1200 ON A GOSUB 1300,1400
  .
  .
  .
1300 REM Start des Unterprogramms für A=1
  .
  .
  .
1390 RETURN
```

ON KEY (n) Anweisung

Zweck: Setzt für BASIC eine Zeilennummer, zu der bei einer Unterbrechung verzweigt werden soll, falls eine angegebene Funktionstaste oder Positionssteuertaste betätigt wurde.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format: ON KEY(*n*) GOSUB *Zeile*

Bemerkungen: *n* ist ein numerischer Ausdruck im Bereich 1 bis 14, der die Taste anzeigt, die eine Unterbrechung erzeugen soll:

1 bis 10	Funktionstasten F1 bis F10
11	Positionsanzeiger nach oben
12	Positionsanzeiger nach links
13	Positionsanzeiger nach rechts
14	Positionsanzeiger nach unten

Zeile Zeilennummer des Beginns der Unterbrechungsroutine für die angegebene Taste. Wird *Zeile* auf 0 (Null) gesetzt, wird die Unterbrechung für diese Taste inaktiviert.

ON KEY (n)
Anweisung

Die Anweisung KEY(*n*) ON muß ausgeführt werden, um diese Anweisung zu aktivieren. Nach KEY(*n*) ON wird - falls in ON KEY(*n*) eine Zeilennummer ungleich 0 (Null) angegeben wurde - mit BASIC vor dem Beginn jeder neuen Anweisung geprüft, ob die angegebene Taste betätigt wurde. War dies der Fall, führt BASIC ein GOSUB zur angegebenen *Zeile* aus.

Wird KEY(*n*) OFF ausgeführt, findet für die angegebene Taste keine Unterbrechung statt. Auch wenn die Taste betätigt wurde, wird dies nicht gespeichert.

Wurde die Anweisung KEY(*n*) STOP ausgeführt, findet für die Taste keine Unterbrechung statt. Wurde die Taste jedoch betätigt, wird dies gespeichert, und sobald KEY(*n*) ON ausgeführt wird, tritt eine sofortige Unterbrechung ein.

Nach der Unterbrechung wird automatisch KEY(*n*) STOP ausgeführt, so daß eine rekursive Unterbrechung nicht auftreten kann. Die Anweisung RETURN aus der Unterbrechungsroutine führt automatisch ein KEY(*n*) ON aus, außer es wurde explilzit innerhalb der Unterbrechungsroutine KEY(*n*) OFF ausgeführt.

Unterbrechungen können nicht auftreten, wenn BASIC kein Programm ausführt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung ON ERROR), wird jede Unterbrechung automatisch inaktiviert (einschließlich ERROR, STRIG(n), PEN, COM(n) und KEY(n)).

Es ist möglich, daß die Tastenunterbrechung nicht arbeitet, wenn andere Tasten vor der angegebenen betätigt wurden. Die Taste, die die Unterbrechung verursachte, kann nicht mit Hilfe von INPUT$ oder INKEY$ getestet werden, so daß die Unterbrechungsroutine für jede Taste unterschiedlich sein muß, falls eine andere Funktion gewünscht wird.

Es ist möglich, mit RETURN *Zeile* im BASIC-Programm zu einer festen Zeilennummer zurückzuverzweigen. 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.

ON KEY (n) Anweisung

Beispiel: Dieses Beispiel zeigt eine Unterbrechungsroutine für die Funktionstaste 5:

```
100 ON KEY(5) GOSUB 200
110 KEY(5) ON
 .
 .
 .
200 REM Funktionstaste 5 betätigt
 .
 .
 .
290 RETURN 140
```

ON PEN
Anweisung

Zweck: Setzt in BASIC eine Zeilennummer, zu der verzweigt wird, sobald der Lichtstift aktiviert wurde.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format: ON PEN GOSUB *Zeile*

Bemerkungen: *Zeile* Zeilennummer des Beginns einer Unterbrechungsroutine für den Lichtstift. Mit Hilfe der Zeilennummer 0 (Null) wird die Unterbrechung für den Lichtstift inaktiviert.

Die Anweisung PEN ON muß ausgeführt werden, um diese Anweisung zu aktivieren. Nach der Ausführung von PEN ON wird – falls eine Zeilennummer ungleich 0 (Null) für die Anweisung ON PEN angegeben wurde – jedesmal, sobald BASIC eine neue Anweisung ausführt, geprüft, ob der Lichtstift aktiviert worden war. Falls ja, führt BASIC die Anweisung GOSUB *Zeile* aus.

ON PEN
Anweisung

Wird PEN OFF ausgeführt, findet keine Unterbrechung statt. Falls der Lichtstift aktiviert wird, wird dies nicht gespeichert.

Wird die Anweisung PEN STOP ausgeführt, findet keine Unterbrechung statt, aber eine Lichtstiftaktivierung wird gespeichert, so daß eine sofortige Unterbrechung stattfindet, sobald PEN ON ausgeführt wird.

Findet eine Unterbrechung statt, wird automatisch PEN STOP ausgeführt, um keine rekursive Unterbrechung stattfinden zu lassen. Die Anweisung RETURN aus der Unterbrechungsroutine führt automatisch PEN ON aus, außer es wird explizit PEN OFF innerhalb des Unterbrechungsprogramms ausgeführt.

Eine Unterbrechung findet nicht statt, wenn BASIC kein Programm ausführt. Findet eine Fehlerunterbrechung statt (resultierend aus der Anweisung ON ERROR), werden alle Unterbrechungen automatisch inaktiviert (einschließlich ERROR, STRIG(n), PEN, COM(n) und KEY(n)).

PEN(0) wird nicht gesetzt, wenn die Lichtstiftaktivität eine Unterbrechung verursacht.

ON PEN Anweisung

Es ist möglich, mit der Anweisung RETURN *Zeile* im BASIC-Programm zu einer festen Zeilennummer zu verzweigen. Dies sollte jedoch mit Vorsicht benutzt werden, weil alle anderen GOSUBs, WHILEs oder FORs, die zur Zeit der Unterbrechung aktiv waren, aktiv bleiben.

> **Hinweis:** Es ist nicht erlaubt, irgendeine Kassettenein-/-ausgabe auszuführen, solange PEN auf ON (ein) steht.

Beispiel: In diesem Programm wird für den Lichtstift zu einer Unterbrechungsroutine verzweigt:

```
1Ø ON PEN GOSUB 5ØØ
2Ø PEN ON
 .
 .
 .
5ØØ REM Unterprogramm für Lichtstift
 .
 .
 .
65Ø RETURN 3Ø
```

ON STRIG(n) Anweisung

Zweck: Übergibt BASIC eine Zeilennummer, zu der verzweigt werden soll, falls einer der Spielpultknöpfe gedrückt wurde.

Versionen:	Kassette	Disk	Erweitert	Umwandlungsprogramm
			***	(**)

Format: ON STRIG(*n*) GOSUB *Zeile*

Bemerkungen: *n* kann 0, 2, 4 oder 6 sein und zeigt den Knopf an, für den eine Unterbrechung erfolgen soll:

0 - Knopf A1

2 - Knopf B1

4 - Knopf A2

6 - Knopf B2

Zeile Zeilennummer der Unterbrechungsroutine. Ist *Zeile* gleich 0 (Null), wird die Unterbrechung der Spielpultknöpfe inaktiviert.

ON STRIG(n) Anweisung

Die Anweisung STRIG(*n*) ON muß ausgeführt werden, um diese Anweisung für den Knopf *n* zu aktivieren. Ist STRIG(*n*) ON ausgeführt und der Anweisung ON STRIG(*n*) wurde eine Zeilennummer ungleich 0 (Null) zugeordnet, prüft BASIC jedesmal, bevor eine neue Anweisung ausgeführt wird, ob der angegebene Knopf gedrückt wurde. War dies der Fall, führt BASIC die Anweisung GOSUB zu der angegebenen *Zeile* aus.

Wurde die Anweisung STRIG(*n*) OFF ausgeführt, findet für den Knopf *n* keine Unterbrechung statt. Auch wenn der Knopf gedrückt wurde, wird dies nicht gespeichert.

Nachdem die Anweisung STRIG(*n*) STOP ausgeführt wurde, findet für den Knopf *n* keine Unterbrechung statt. Wurde aber der Knopf gedrückt, findet eine sofortige Unterbrechung statt, nachdem STRIG(*n*) ON ausgeführt wurde.

ON STRIG(n) Anweisung

Tritt eine Unterbrechung auf, wird automatisch die Anweisung STRIG(*n*) STOP ausgeführt, um keine rekursive Unterbrechung stattfinden zu lassen. Die Anweisung RETURN aus der Unterbrechungsroutine führt automatisch ein STRIG(*n*) ON aus, außer es wird explizit STRIG(*n*) OFF innerhalb der Unterbrechungsroutine ausgeführt.

Eine Unterbrechung findet nicht statt, wenn BASIC kein Programm ausführt. Tritt eine Fehlerunterbrechung auf (resultierend aus der Anweisung ON ERROR), werden alle Unterbrechungen automatisch inaktiviert (einschließlich ERROR, STRIG(n), PEN, COM(n) und KEY(n)).

Mit STRIG(*n*) ON wird die Unterbrechungsroutine aktiviert, die den Status des angegebenen Spielpultknopfes prüft. Das Niederdrücken, das die Unterbrechung verursacht, setzt nicht die Funktionen STRIG(0), STRIG(2), STRIG(4) oder STRIG(6).

Mit RETURN *Zeile* kann im BASIC-Programm zu einer festen Zeilennummer verzweigt werden. Dies sollte mit Vorsicht getan werden, da alle anderen GOSUBs, WHILEs oder FORs, die zur Zeit der Unterbrechung aktiv waren, aktiv bleiben.

ON STRIG(n) Anweisung

Beispiel: Dieses Beispiel zeigt eine Unterbrechungsroutine für den Knopf des ersten Spielpults:

```
100 ON STRIG(0) GOSUB 2000
110 STRIG(0) ON
 .
 .
 .
2000 REM Unterprogramm für 1. Knopf
 .
 .
 .
2100 RETURN
```

OPEN
Anweisung

Zweck: Erlaubt die Ein-/Ausgabe zu und von einer Datei oder Einheit.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: Erste Form:

OPEN *Dateiangabe* [FOR *Modus*] AS [#] *Dateinummer* [LEN=*Satzlänge*]

Alternative Form:

OPEN *Modus2,* [#] *Dateinummer, Dateiangabe* [,*Satzlänge*]

Bemerkungen: *Modus* ist in der ersten Form eines der folgenden Wörter:

OUTPUT Spezifiziert sequentielle Ausgabe

INPUT Spezifiziert sequentielle Eingabe

APPEND Spezifiziert Hinzufügen zu einer sequentiellen Datei, wobei die Datei bei der Eröffung auf das Datenende positioniert wird.

Man muß beachten, daß *Modus* eine Zeichenkettenkonstante sein muß, die nicht innerhalb Anführungszeichen (") stehen darf. Wird *Modus* weggelassen, wird wahlfreier Zugriff 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 für wahlfreien Zugriff

Für beide Formate:

Dateinummer

Ganzzahliger Ausdruck, dessen Wert zwischen 1 und der maximalen Anzahl der erlaubten Dateien liegt. Im Kassetten-BASIC ist die maximale Anzahl 4, im Disk- und erweiterten BASIC ist das Standardmaximum 3, kann aber mit der Auswahl /F: im BASIC-Befehl geändert werden.

Dateiangabe

Zeichenkettenausdruck für die Dateispezifikation, wie unter "Namensgebung für Dateien" in Kapitel 3 beschrieben.

Satzlänge

Ganzzahliger Ausdruck, der - falls eingefügt - die Satzlänge für Dateien mit wahlfreiem Zugriff setzt. Er kann sich im Bereich 1 bis 32767 befinden. *Satzlänge* ist für sequentielle Dateien nicht gültig. Die Standardsatzlänge ist 128 Bytes. Sie darf den Wert, der durch die Auswahl /S: im BASIC-Befehl gesetzt wurde, nicht überschreiten.

OPEN ordnet der Datei oder Einheit für die Ein-/Ausgabe einen Puffer zu und bestimmt den Zugriffsmodus, der für den Puffer benutzt wird.

Dateinummer ist die Nummer, die der Datei zugeordnet ist, solange sie eröffnet ist, und die von anderen Ein-/Ausgabeanweisungen benutzt wird, um die Datei oder Einheit anzusprechen.

OPEN Anweisung

OPEN muß ausgeführt werden, bevor irgendeine Ein-/Ausgabe zu einer Einheit oder Datei mit Hilfe einer der folgenden Anweisungen oder irgendeiner Anweisung oder Funktion, die eine Dateinummer benötigt, erfolgen kann:

PRINT #	INPUT #
PRINT # USING	LINE INPUT #
WRITE #	GET
INPUT$	PUT

GET und PUT sind für Dateien mit wahlfreiem Zugriff gültig (oder Datenfernverarbeitungsdateien - siehe nächster Abschnitt). Eine Diskettendatei kann entweder wahlfrei oder sequentiell sein, und ein Drucker kann entweder in wahlfreiem oder sequentiellem Modus eröffnet werden. Alle anderen Einheiten jedoch können nur für sequentielle Operationen eröffnet werden.

BASIC fügt normalerweise nach jedem Wagenrücklauf, der zum Drucker gesendet wird, einen Zeilenvorschub (CHR$(13)) an. Wird jedoch der Drucker (LPT1:, LPT2: oder LPT3:) als Datei mit wahlfreiem Zugriff mit einer Breite von 255 eröffnet, wird der Zeilenvorschub unterdrückt.

APPEND ist nur für Diskettendateien gültig. Der Dateizeiger wird auf das Ende der Datei gesetzt und die Satznummer auf den letzten Satz der Datei. PRINT # oder WRITE # fügen Sätze an die Datei an.

> **Hinweis:** Es ist jederzeit möglich, eine bestimmte Datei für mehr als eine Dateinummer zu eröffnen. Dies erlaubt, für verschiedene Zwecke in verschiedenen Modi zu arbeiten, d. h. daß mit verschiedene 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 für sequentielle Ausgabe oder Hinzufügen (append) eröffnet werden, wenn die Datei schon eröffnet ist.

Ist der Einheitenname weggelassen, wenn Kassetten-BASIC benutzt wird, wird CAS1: angenommen. Wird mit Disk- oder erweitertem BASIC gearbeitet, wird die Standardeinheit für DOS angenommen.

Ist CAS1: als Einheit angegeben und der Dateiname fehlt, wird die nächste Datendatei auf der Kassette eröffnet.

OPEN Anweisung

Im Kassetten-BASIC können zu einer Zeit maximal vier Dateien eröffnet werden (Kassette, Drucker, Tastatur und Bildschirm). Man beachte, daß zu einer Zeit nur eine Kassettendatei eröffnet werden kann. Für Disk- und erweitertes BASIC ist das Standardmaximum drei Dateien. Dieser Wert kann mit der Auswahl /F: im BASIC-Befehl geändert werden.

Existiert eine Eingabedatei, die eröffnet werden soll, nicht, erhält man den Fehler "File not Found" (Datei nicht gefunden). Wird eine Datei, die nicht existiert, für Ausgabe, Hinzufügen oder wahlfreien Zugriff eröffnet, wird eine Datei erstellt.

Jeder angegebene Wert, der sich außerhalb der angegebenen Bereiche befindet, ergibt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Die Datei wird nicht eröffnet.

Eine vollständige Erklärung der Benutzung von Diskettendateien ist in Anhang B. "BASIC-Disketteneinein- und -ausgabe" enthalten. Im nächsten Abschnitt folgen Informationen über die Eröffnung von Datenfernverarbeitungsdateien mit der Anweisung OPEN "COM....

OPEN
Anweisung

Beispiel:

```
1Ø OPEN "DATA" FOR OUTPUT AS #1
```
oder
```
1Ø OPEN "O",#1,"DATA"
```

Jede dieser Anweisungen eröffnet die Datei "DATA" für sequentielle Ausgabe auf der Standardeinheit (CAS1: für Kassetten-BASIC, Standardeinheit für Disk- oder erweitertes BASIC). Dabei muß beachtet werden, daß die Eröffnung für Ausgabe aller Daten, die sich in der Datei befinden, zerstört. Sollen die Daten nicht zerstört werden, muß mit APPEND eröffnet werden.

```
2Ø OPEN "B:SSFILE" AS 1 LEN=256
```
oder
```
2Ø OPEN "R",1,"B:SSFILE",256
```

Jede der zwei vorhergehenden Anweisungen eröffnet die Datei "SSFILE" auf der Diskette in Laufwerk B für wahlfreie Ein- und Ausgabe. Die Satzlänge ist 256.

```
25 FILE$ = "A:DATA.ART"
3Ø OPEN FILE$ FOR APPEND AS 3
```

In diesem Beispiel wird die Datei "DATA.ART" auf der Diskette in Laufwerk A eröffnet und der Dateizeiger an das Ende der Datei gebracht, so daß jede Ausgabe zu dieser Datei an das Ende der Datei angefügt wird.

```
Ok
10 OPEN "LPT1:" AS #1' wahlfreier Zugriff
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$(30,"_")
80 PRINT #1,"Druckbreite 80 mit CR/LF"
RUN
Druckbreite 80
Jetzt Druckbreite 255
Diese Zeile wird unterstrichen
Druckbreite 80 mit CR/LF
Ok
```

OPEN
Anweisung

In Zeile 10 dieses Beispiels wird der Drucker für wahlfreien Modus eröffnet. Da die Standardbreite 80 ist, enden die mit den Zeilen 20 und 30 gedruckten Zeilen mit einem Wagenrücklauf/Zeilenvorschub. In Zeile 40 wird die Druckbreite auf 255 geändert. Dadurch wird der Zeilenvorschub nach dem Wagenrücklauf unterdrückt.
Deshalb endet die mit Zeile 50 gedruckte Zeile nur mit einem Wagenrücklauf und nicht mit einem Zeilenvorschub. Dadurch überdruckt 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 geändert, so daß das Unterstreichen und die folgenden Zeilen wieder mit einem Zeilenvorschub enden.

OPEN "COM... Anweisung

Zweck: Eröffnet eine Datei für Datenfernverarbeitung.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	(**)

Nur gültig mit dem Anschluß für asynchrone Datenfernverarbeitung.

Format: OPEN "COM*n*: [*Geschwindigkeit*] [,*Parität*] [,*Daten*] [,*Stopp*] [,RS] [,CS[*n*]] [,DS[*n*]] [,CD[*n*]] [,LF]" AS [#] *Dateinummer* [LEN=*Nummer*]

Bemerkungen: *n* ist 1 oder 2 und bedeutet die Nummer des Anschlusses für asynchrone Datenfernverarbeitung.

Geschwindigkeit
Ganzzahlige Konstante, die die Sende-/Empfangs-Bit-Rate in Bits pro Sekunde (bps) angibt. Gültige Geschwindigkeiten sind 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 und 9600 bps. Die Standardannahme ist 300 bps.

Parität Konstante, bestehend aus einem Zeichen, die die Parität für Senden und Empfangen wie folgt angibt:

S SPACE: Das Paritäts-Bit wird immer als Leerstelle (0 Bit) gesendet oder empfangen.

O ODD: Für Senden und Empfangen wird auf ungerade Bit-Parität geprüft.

M MARK: Das Paritäts-Bit wird immer als Einer-Bit gesendet und empfangen.

E EVEN: Beim Senden und Empfangen wird auf gerade Bit-Parität geprüft.

N NONE: Beim Senden und Empfangen wird nicht auf Bit-Parität geprüft.

Die Standardannahme ist EVEN (E) (gerade).

Daten Ganzzahlige Konstante, die angibt, wie viele Daten-Bits gesendet oder empfangen werden sollen. Gültige Werte sind 4, 5, 6, 7 oder 8. Die Standardannahme ist 7.

Stopp Ganzzahlige Konstante, die die Anzahl der Stopp-Bits angibt. Gültige Werte sind 1 oder 2. Die Standardannahme sind zwei Stopp-Bits für 75 und 110 bps, ein Stopp-Bit für alle anderen. Wird für *Daten* 4 oder 5 ausgewählt, bedeutet hier eine 2 (zwei) 1 1/2 Stopp-Bits.

Dateinummer
Ganzzahliger Ausdruck, der eine gültige Dateinummer errechnet. Diese Zahl ist so lange einer Datei zugeordnet, wie sie eröffnet ist, und wird mit anderen Ein-/Ausgabeanweisungen für Datenfernverarbeitung benutzt, um diese Datei anzusprechen.

Nummer Maximale Anzahl von Bytes, die aus dem Datenfernverarbeitungspuffer mit Hilfe von GET oder PUT gelesen oder geschrieben werden können. Die Standardannahme ist 128 Bytes.

OPEN "COM...
Anweisung

OPEN "COM... legt in der gleichen Art und Weise für Ein-/Ausgabe einen Puffer an, wie OPEN für Diskettendateien. Die Schnittstelle RS232 für asynchrone Datenfernverarbeitung mit anderen Computern und peripheren Einheiten wird unterstützt.

Die Datenfernverarbeitungseinheit kann zu einer Zeit nur für eine Dateinummer eröffnet werden.

Die Angaben **RS, CS, DS, CD** und **LF** beeinflussen die Leitungssignale wie folgt:

RS unterdrückt 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 Wagenrücklauf einen Zeilenvorschub

Hinweis: Die Parameter *Geschwindigkeit, Parität*, Daten und Stopp müssen in dieser Reihenfolge angegeben sein. Nicht aber **RS, CS, DS, CD** und **LF**.

Die Leitung RTS (Sendeteil einschalten) wird eingeschaltet, wenn die Anweisung OPEN "COM... ausgeführt 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 geprüft.

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) zurücksendet. 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) ignoriert, wenn die Anweisung OPEN "COM... ausgeführt wird. Die Auswahl **CD** erlaubt es, diese Leitung zu testen, indem man den Parameter in der gleichen Weise wie für **CS** und **DS** einfügt. Wird *n* weggelassen oder ist *n* gleich 0 (Null), wird der Empfangssignalpegel überhaupt nicht getestet (dies ist das gleiche, als ob die Auswahl **CD** weggelassen würde).

Der Parameter **LF** (Zeilenvorschub) ist vorgesehen, wenn Datenfernverarbeitungsdateien verwendet werden sollen, um eine Ausgabe auf einen seriellen Zeilendrucker zu übertragen. Wird **LF** angegeben, wird ein Zeilenvorschubzeichen (Hex 0A) automatisch nach jedem Wagenrücklaufzeichen (Hex 0C) gesendet. (Dies schließt einen gesendeten Wagenrücklauf als Ergebnis der Druckbreitensetzung ein.) Es ist zu beachten, daß INPUT # und LINE INPUT # stoppen, sobald ein Wagenrücklaufzeichen erkannt wird, wenn sie dazu benutzt werden, von einer mit Auswahl **LF** eröffneten Datenfernverarbeitungsdatei zu lesen. Das Zeilenvorschubzeichen wird immer ignoriert.

Jeder Codierfehler innerhalb des Zeichenkettenausdrucks, beginnend mit *Geschwindigkeit*, resultiert in dem Fehler "Bad File Name" (Falscher Dateiname). Es wird nicht angegeben, welcher Parameter falsch ist.

In Anhang F. "Datenfernverarbeitung" sind weitere Informationen über die Steuerung von Ausgabesignalen und weitere technische Informationen über die Datenfernverarbeitungsunterstützung enthalten.

Wenn acht Daten-Bits angegeben werden, muß die Parität N spezifiziert werden. Werden vier Daten-Bits angegeben, muß eine Parität angegeben werden; das heißt N ist hier ungültig. BASIC benutzt alle acht Bits in einem Byte, um Zahlen zu speichern. Das bedeutet, daß acht Daten-Bits angegeben werden müssen, wenn numerische Daten gesendet oder empfangen werden (z. B. mit Hilfe von PUT). (Dies ist nicht der Fall, wenn numerische Daten als Text gesendet werden.)

OPEN "COM... Anweisung

Beispiele:

```
1Ø OPEN "COM1:" AS 1
```

Die Datei 1 wird für Datenfernverarbeitung mit allen Standardannahmen eröffnet. Die Geschwindigkeit ist 300 bps mit gerader Parität, sieben Daten-Bits und einem Stopp-Bit.

```
1Ø OPEN "COM1"24ØØ" AS #2
```

Die Datei 2 wird für Datenfernverarbeitung mit 2400 bps eröffnet. Die Parität, die Anzahl der Daten-Bits und die Anzahl der Stopp-Bits sind Standardannahmen.

```
2Ø OPEN "COM2:12ØØ,N,8" AS #1
```

Die Dateinummer 1 wird für asynchrone Ein-/Ausgabe mit 1200 bps eröffnet, wobei keine Parität erzeugt oder geprüft wird. Acht Bit-Bytes werden gesendet und empfangen, und ein Stopp-Bit wird gesendet.

```
1Ø OPEN "COM1:96ØØ,N,8,,CS,DS,CD" AS #1
```

Eröffnet COM1: mit 9600 bps, keiner Parität und acht Daten-Bits. CTS, DSR und CD werden nicht geprüft.

OPEN "COM... Anweisung

```
5Ø OPEN "COM1:12ØØ,,,,CS,DS2ØØØ" AS #1
```

Eröffnet COM1: mit 1200 bps und der Standardannahme gerade Parität und sieben Daten-Bits. RTS wird gesendet, CTS wird nicht geprüft, und "Device Timeout" (Einheitenzeitsperre) wird angezeigt, falls DSR nicht innerhalb zwei Sekunden angezeigt wird.

Man muß beachten, daß die Kommata benötigt werden, um die Positionen für die Parameter *Parität, Start* und *Stopp* anzuzeigen, auch wenn diese Werte nicht angegeben werden.

OPEN "COM... Anweisung

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 geprüft, daß CD (Empfangssignalpegel) vom Modem gesendet wird, bevor mit der Übertragung begonnen wird. In Zeile 20 wird eine Zeitsperre nach 10 Sekunden gesetzt. TRIES 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 kürzeren Verzögerung bis zur Zeitsperre neu eröffnet..

```
5 TRIES=6
1Ø ON ERROR GOTO 1ØØ
2Ø OPEN "COM1:3ØØ,N,8,2,CS,DS,CD1ØØØØ" AS #1
3Ø ON ERROR GOTO Ø
4Ø CLOSE #1 ' arbeitet, deshalb fortfahren
5Ø GOTO 1ØØØ
 .
 .
 .
1ØØ TRIES=TRIES-1
11Ø IF TRIES=Ø THEN ON ERROR GOTO Ø ' abbrechen
12Ø RESUME
 .
 .
 .
1ØØØ OPEN "COM1:3ØØ,N,8,2,CS,DS,CD2ØØØ" AS #1
```

OPEN "COM... Anweisung

Das nächste Beispiel zeigt einen typischen Weg, wie eine Datenfernverarbeitungsdatei benutzt werden kann, um einen seriellen Zeilendrucker zu steuern. Durch den Parameter LF in der Anweisung OPEN wird verhindert, daß die Ausgabezeilen übereinander gedruckt werden.

```
1Ø WIDTH "COM1:", 132
2Ø OPEN "COM1:12ØØ,N,8,,CS1ØØØØ,DS1ØØØØ,
      CD1ØØØØ,LF" AS #1
```

OPTION BASE
Anweisung

Zweck: Definiert den kleinsten Wert für Bereichsindizes.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: OPTION BASE *n*

Bemerkungen: *n* ist 1 oder 0.

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

```
OPTION BASE 1
```

ausgeführt, darf der kleinste Wert für den Bereichsindex 1 sein.

Die Anweisung OPTION BASE muß vor der Definition oder Benutzung irgendeines Bereichs stehen.

OUT Anweisung

Zweck: Sendet ein Byte zu einem Maschinenausgabeanschluß

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: OUT *n,m*

Bemerkungen: *n* ist ein numerischer Ausdruck für eine Anschlußnummer im Bereich 0 bis 65535.

m ist ein numerischer Ausdruck für zu sendende Daten im Bereich 0 bis 255.

Die Broschüre *IBM Personalcomputer Technisches Handbuch* enthält eine Beschreibung über gültige Anschlußnummern (Ein-/Ausgabeadressen).

OUT ist die Gegenanweisung zur Funktion INP. Siehe "Funktion INP" in diesem Kapitel.

OUT
Anweisung

Mit OUT kann man die Videoausgabe beeinflussen. Bei manchen Bildschirmen, die an den Farb-/Grafikbildschirmanschluß angeschlossen sind, ist es möglich, 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 98Ø,2: OUT 981,43
```

Mit diesem Beispiel wird die Anzeige um zwei Zeichen nach rechts für eine Zeichenbreite von 40 verschoben (oder 16 Punkte im grafischen Modus für mittlere Auflösung oder 32 Punkte im grafischen Modus für hohe Auflösung).

```
OUT 98Ø,2: OUT 981,85
```

In diesem Beispiel wird die Bildschirmanzeige um fünf Zeichen nach rechts für eine Zeichenbreite von 80 verschoben.

OUT
Anweisung

Die Verschiebung durch diese OUT-Anweisung bleibt so lange aktiviert, bis eine Anweisung WIDTH oder SCREEN ausgeführt wird. Auch mit dem Befehl MODE in DOS kann diese Verschiebung der Bildschirmanzeige vorgenommen werden. Dieser Befehl hat den Vorteil, daß er gültig bleibt, bis das System in Grundstellung gebracht wird.

Beispiel:

```
100 OUT 32,100
```

Damit wird der Wert 100 zum Ausgabeanschluß 32 gesendet.

PAINT
Anweisung

Zweck: Füllt einen Bildschirmbereich mit einer ausgewählten Farbe.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	***

Nur im grafischen Modus gültig.

Format: PAINT (*x,y*) [,*malen*[,*Grenze*]]

Bemerkungen:

(*x,y*) sind die Koordinaten eines Punkts innerhalb des Bereichs, der mit Farbe gefüllt werden soll. Die Koordinaten des Bereichs können in absoluter oder relativer Form angegeben werden (siehe "Spezifikation von Koordinaten" unter "Grafischer Modus" in Kapital 3). Dieser Punkt wird als Anfangspunkt benutzt.

malen Farbe, mit der gemalt werden soll, im Bereich 0 bis 3. In der mittleren Auflösung ist dies die Farbe der laufenden Palette, wie in der Anweisung COLOR definiert. 0 ist die Hintergrundfarbe, die Standardannahme ist die Vordergrundfarbe, Farbe Nummer 3.

Ist in der hohen Auflösung *malen* gleich 0, bedeutet dies schwarz, die Standardannahme von 1 bedeutet weiß.

Grenze Farbe der Seiten einer Figur, deren Inhalt bemalt werden soll, im Bereich 0 bis 3, wie oben beschrieben.

Die Figur, die bemalt werden soll, ist die Figur, deren Seiten die Farben haben, die in *Grenze* angegeben sind. Die Figur wird mit der Farbe, die in *malen* angegeben ist, bemalt.

PAINT
Anweisung

Da es für die hohe Auflösung nur zwei Farben gibt, ergibt es keinen Sinn, wenn *malen* und *Grenze* verschieden sind. Da *Grenze* die Standardannahme für *malen* ist, benötigt man den dritten Parameter in der hohen Auflösung nicht.

In der hohen Auflösung bedeutet das, daß der Bereich geschwärzt wird, bis schwarz angetroffen wird, oder geweißt wird, bis weiß angetroffen wird.

In der mittleren Auflösung kann man mit der Farbe 1 auffüllen, wenn die Grenze die Farbe 2 hat. Visuell ist dies eine grüne Fläche mit einem roten Rand.

Der Anfangspunkt von PAINT muß innerhalb der auszufüllenden Figur sein. Hat der angegebene Punkt schon die Farbe *Grenze,* hat PAINT keinen Effekt. Wird *malen* weggelassen, wird die Vordergrundfarbe benutzt (3 in der mittleren Auflösung, 1 in der hohen Auflösung). Mit PAINT kann jeder Figurtyp bemalt werden, aber gezackte Ränder in einer Figur erhöhen die Größe des Stapelspeichers, der von PAINT benötigt wird. Soll eine komplexe Zeichnung hergestellt werden, ist es ratsam, am Anfang des Programms CLEAR auszuführen, um den verfügbaren Stapelspeicher zu vergrößern.

PAINT
Anweisung

Die Anweisung PAINT erlaubt es, Szenen mit sehr wenigen Anweisungen darzustellen.

Beispiel:

```
5 SCREEN 1
1Ø LINE (Ø,Ø)-(1ØØ,15Ø),2,B
2Ø PAINT (5Ø,5Ø),1,2
```

Die Anweisung PAINT in Zeile 20 bemalt das Rechteck, das in Zeile 10 gezeichnet wird, mit der Farbe 1.

PEEK
Funktion

Zweck: Übergibt ein gelesenes Byte von der angegebenen Hauptspeicherposition.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *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 übergebene Wert ist eine ganze Zahl im Bereich 0 bis 255.

PEEK ist die Gegenfunktion der Anweisung POKE (siehe "Anweisung POKE" im weiteren Verlauf dieses Kapitels).

Beispiel:

Das folgende Beispiel kann dazu benutzt werden, in einem Programm zu testen, welcher Bildschirmanschluß sich am System befindet. Nachdem die Zeile 30 ausgeführt ist, enthält die Variable IBMMONO den Wert 0, wenn der Farb-/Grafikbildschirmanschluß benutzt wird, oder eine 1, wenn der IBM Schwarz/Weiß-Bildschirm- und Paralleldruckeranschluß benutzt wird.

```
10 'Testen Bildschirmanschluß
20 DEF SEG=0
30 IF (PEEK(&410) AND &H30)=&H30
      THEN IBMMONO=1
      ELSE IBMMONO=0
```

PEN
Anweisung und Funktion

Zweck: Liest die Koordinaten des Lichtstifts.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

PEN STOP ist nur im erweiterten BASIC und im BASIC-Unwandlungsprogramm gültig.

Format: Als Anweisung:

PEN ON

PEN OFF

PEN STOP

Als Funktion:

v = PEN(n)

PEN
Anweisung und Funktion

Bemerkungen: Die Funktion PEN, *v*=PEN(*n*) liest die Koordinaten des Lichtstifts.

n ist ein numerischer Ausdruck im Bereich 0 bis 9 und beeinflußt den übergebenen Wert durch die Funktion wie folgt:

0 Ein Zeichen, das angibt, ob der Lichtstift seit der letzten Anfrage unten war. Falls der Stift unten war, wird -1 übergeben, sonst 0.

1 Übergibt die x-Koordinate der Position, in der der Lichtstift zuletzt aktiviert wurde. Bereich 0 bis 319 in der mittleren Auflösung, 0 bis 639 in der hohen Auflösung.

2 Übergibt die Position der y-Koordinate, in der der Lichtstift zuletzt aktiviert wurde. Bereich ist 0 bis 199.

3 Übergibt den laufenden Schalterwert des Lichtstifts -1, falls er unten ist, 0, falls er oben ist.

PEN
Anweisung und Funktion

4 Übergibt die letzte bekannte gültige x-Koordinate. Der Bereich ist 0 bis 319 in der mittleren Auflösung oder 0 bis 639 in der hohen Auflösung.

5 Übergibt die letzte bekannte gültige y-Koordinate. Der Bereich liegt zwischen 0 und 199.

6 Übergibt die Zeilenposition eines Zeichens, in der der Lichtstift zuletzt aktiviert wurde. Der Bereich liegt zwischen 1 und 24.

7 Übergibt die Spaltenposition des Zeichens, in der der Lichtstift zuletzt aktiviert wurde. Der Bereich liegt zwischen 1 und 40 oder 1 und 80, abhängig von WIDTH (Breite).

8 Übergibt die letzte bekannte gültige Zeichenzeile. Der Bereich liegt zwischen 1 und 24.

9 Übergibt die letzte bekannte gültige Zeichenspaltenposition. Der Bereich liegt zwischen 1 und 40 oder 1 und 80, abhängig von WIDTH (Breite).

PEN
Anweisung und Funktion

Durch PEN ON wird die Lesefunktion des Lichtstifts aktiviert. Normalerweise ist die Lichtstiftfunktion ausgeschaltet. Die Anweisung PEN ON muß ausgeführt werden, bevor irgendeine Lesefunktion für den Lichtstift ausgeführt werden kann. Ist die Funktion des Lichtstifts ausgeschaltet und eine Lichtstiftfunktion wird aufgerufen, ergibt sich der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

Um die Ausführungsgeschwindigkeit eines Programms zu verbessern, sollte man die Anweisung PEN OFF ausführen, wenn der Lichtstift nicht benutzt wird.

Im erweiterten BASIC erlaubt die Ausführung der Anweisung PEN ON auch Unterbrechungen mit der Anweisung ON PEN. Nach der Ausführung von PEN ON wird, falls der Anweisung ON PEN eine Zeilennummer ungleich 0 zugeordnet wurde, jedesmal wenn eine neue Anweisung ausgeführt wird, von BASIC geprüft, ob der Lichtstift aktiviert worden war. Siehe "Anweisung ON PEN" in diesem Kapitel.

PEN OFF inaktiviert die Lesefunktion für den Lichtstift. Im erweiterten BASIC findet für den Lichtstift keine Unterbrechung statt, und eine Aktion für den Lichtstift wird nicht gespeichert.

PEN
Anweisung und Funktion

PEN STOP ist nur im erweiterten BASIC verfügbar. Die Unterbrechung der Lichtstiftaktivität wird inaktiviert, aber jede Aktivität wird gespeichert, so daß nach der Ausführung von PEN ON eine sofortige Unterbrechung erfolgt.

Befindet sich der Lichtstift in dem Bereich des Bildschirmrandes, sind die übergebenen Werte ungenau.

Es sollte keine Ein-/Ausgabe zur Kassette durchgeführt werden, solange der Lichtstift durch PEN ON aktiviert ist.

Beispiel:

```
5Ø PEN ON
6Ø FOR I=1 TO 5ØØ
7Ø X=PEN(Ø): X1=PEN(3)
8Ø PRINT X, X1
9Ø NEXT
1ØØ PEN OFF
```

In diesem Beispiel werden der Lichtstiftwert seit der letzten Abfrage und sein laufender Wert angezeigt.

PLAY
Anweisung

Zweck: Spielt Musik mit Hilfe einer *Zeichenkette.*

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format: PLAY *Zeichenkette*

Bemerkungen: PLAY arbeitet nach einem ähnlichen Konzept wie DRAW, indem einer Zeichenkette eine "Tondefinitionssprache" zugeordnet wird.

Zeichenkette
Zeichenkettenausdruck, der aus einzelnen Zeichen für Musikbefehle besteht.

Die einzelnen Zeichenbefehle für PLAY:

A bis G mit wahlfreiem #, + oder –
Spielt die angegebenen Noten in der laufenden Oktave. Ein Nummerzeichen (#) oder Pluszeichen (+) zeigt einen Halbton höher, 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 ungültige Note.

PLAY
Anweisung

O n Oktave. Setzt die laufende Oktave für 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 n Spielt die Note n. n kann sich im Bereich 0 bis 84 befinden. In den sieben möglichen Oktaven befinden sich 84 Noten. n=0 bedeutet Pause. Dies ist eine weitere Alternative, um Noten auszuwählen.

L n setzt die Länge der folgenden Noten. Die aktuelle Notenlänge ist 1/n. n kann sich im Bereich 1 bis 64 befinden. In der folgenden Tabelle wird dies verdeutlicht.

Länge	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 Längenangabe kann auch hinter einer Note stehen, wenn man die Länge nur für diese Note ändern möchte, z.B. ist A16 gleichbedeutend mit L16A.

P n Pause. n kann sich im Bereich 1 bis 64 befinden und zeigt die Länge der Pause in der gleichen Art und Weise an wie L (Länge).

(Punkt). Ein Punkt nach einer Note bedeutet eine punktierte Note, d. h. seine Länge wird mit 3/2 multipliziert. Nach einer Note kann mehr als ein Punkt auftreten, und die Länge 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 können auch nach einer Pause (P) erscheinen, um die Pausenlänge auf die gleiche Art zu berechnen.

T n Tempo. Setzt die Anzahl der Viertelnoten in einer Minute. n kann im Bereich 32 bis 255 liegen. Die Standardannahme ist 120. Unter "Anweisung SOUND" im weiteren Verlauf dieses Kapitels befindet sich eine Tabelle, die häufige Tempi und die zugehörigen Schläge pro Minute enthält.

MF Musikvordergrund. Musik (erstellt durch SOUND oder PLAY) läuft im Vordergrund. Das bedeutet, jede nachfolgende Note oder Ton kann erst beginnen, nachdem die vorherige Note oder Ton beendet ist. PLAY kann durch Ctrl-Break beendet werden.
Der Standardstatus ist Musik im Vordergrund.

MB Musikhintergrund. Musik (erstellt durch SOUND oder PLAY) läuft im Hintergrund statt im Vordergrund ab. Das bedeutet, daß jede Note oder jeder Ton in einen Puffer gesetzt wird, was dem BASIC-Programm erlaubt, weiter in der Ausführung abzulaufen, während im Hintergrund Musik spielt. Bis zu 32 Noten (oder Pausen) können gleichzeitig im Hintergrund gespielt werden.

MN Normale Musik. Jede Note wird 7/8 der Zeit, die in L (Länge) angegeben ist, gehalten. Dies ist die Standardannahme für **MN**, **ML** und **MS**.

ML Musik legato. Jede Note wird die volle Periode gehalten, die in L (Länge) gesetzt wurde.

MS Musik-Staccato. Jede Note wird 3/4 der Zeit, die in L angegeben wurde, gehalten.

X Variable;
führt 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 benötigt, 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 können auch in der Form VARPTR**$**(*Variable*) statt **=Variable;** angegeben werden. Dies ist für Programme nützlich, die später umgewandelt werden. Beispiel:

Eine Methode **Alternative Methode**

```
PLAY "XA$;"          PLAY "X"+VARPTR$(A$)
PLAY "O=I;"          PLAY "O="+VARPTR$(I)
```

X kann dazu benutzt werden, einen "Unterton" in einer Zeichenkette zu speichern und ihn wiederholt in verschiedenen Tempi oder Oktaven von einer anderen Zeichenkette aufzurufen.

Beispiel: Das folgende Beispiel spielt einen Ton:

```
1Ø REM little lamb
2Ø MARY$="GFE-FGGG"
3Ø PLAY "MB T1ØØ O3 L8;XMARY$;P8 FFF4"
4Ø PLAY "GB-B-4; XMARY$; GFFGFE-."
```

POINT
Funktion

Zweck: Übergibt die Farbe eines angegebenen Punkts auf dem Bildschirm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Nur im Grafikmodus.

Format: *v* = POINT (*x,y*)

Bemerkungen: (*x,y*) sind die Koordinaten des benutzten Punktes. Die Koordinaten müssen in absoluter Form verfügbar sein (siehe "Spezifizieren von Koordinaten" unter "Grafischer Modus" in Kapitel 3).

Liegt der angegebene Punkt nicht im Bereich, wird der Wert -1 übergeben. In der mittleren Auflösung sind gültige übergebene Wert 0, 1, 2 und 3, in der hohen Auflösung 0 und 1.

Beispiel: Im folgenden Beispiel wird der laufende Status des Punktes (I,I) umgekehrt:

```
5 SCREEN 2
1Ø IF POINT (I,I)<>Ø THEN PRESET(I,I)
                     ELSE PSET(I,I)
```

oder

```
1Ø PSET(I,I),1-POINT(I,I)
```

POKE
Anweisung

Zweck: Schreibt ein Byte in eine Hauptspeicherstelle.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: 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 müssen sich im Bereich 0 bis 255 befinden.

POKE
Anweisung

Die Gegenfunktion zu POKE ist PEEK.

(Siehe "Funktion 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 übergeben.

Warnung:
BASIC macht keine Adreßprüfung. Deshalb ist es nicht erlaubt, mit POKE etwas in BASIC-Stapelbereiche, in den BASIC-Variablenbereich oder in das BASIC-Programm zu schreiben.

Beispiel:

```
1Ø DEF SEG: POKE 1Ø6,Ø
```

Unter "Variable INKEY$" in diesem Kapitel wird dieses Beispiel erklärt.

POS
Funktion

Zweck: Übergibt die Spaltenposition, in der sich gerade der Positionsanzeiger befindet.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = POS(*n*)

Bemerkungen: *n* ist ein Scheinargument.

Die laufende Horizontalposition (Spalte) des Positionsanzeigers wird übergeben. Der übergebene Wert befindet sich im Bereich 1 bis 40 oder 1 bis 80, abhängig vom Setzen von WIDTH (Breite). Mit CSRLIN kann die vertikale Position (Zeile) des Positionsanzeigers gefunden werden (siehe "Variable CSRLIN" in diesem Kapitel).

Siehe auch "Funktion LPOS".

Beispiel:

```
IF POS(Ø)>6Ø THEN PRINT CHR$(13)
```

Dieses Beispiel erzeugt einen Wagenrücklauf (bringt den Positionsanzeiger an den Beginn der nächsten Zeile), falls sich der Positionsanzeiger hinter Position 60 des Bildschirms befindet.

PRINT Anweisung

Zweck: Zeigt Daten auf dem Bildschirm an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format:

PRINT [*Liste der Ausdrücke*] [;]
? [*Liste der Ausdrücke*] [;]

Bemerkungen *Liste der Ausdrücke*
Liste der numerischen und/oder Zeichenkettenausdrücke, die durch Kommata (,), Leerstellen oder Semikolons (;) getrennt sind. Jede Zeichenkonstante in dieser Liste muß in Anführungszeichen eingeschlossen sein.

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

Hinweis: Das Fragezeichen (?) kann als Kurzform für die Eingabe von PRINT nur in Verbindung mit dem BASIC-Korrekturprogramm benutzt werden.

Druckpositionen

Die Position jeder angezeigten Angabe wird durch die Zeichen bestimmt, die die Angaben in der Liste trennen. BASIC teilt die Zeile in Druckzonen zu je 14 Zeichen ein. Ein Komma in der Liste der Ausdrücke bewirkt, daß der nächste Wert angezeigt wird. Die Angabe von einer oder mehreren Leerstellen zwischen den Ausdrücken hat den gleichen Effekt wie die Angabe eines Semikolons.

Endet die Liste der Ausdrücke mit einem Komma, Semikolon oder einer Funktion SPC oder TAB, beginnt die nächste Anweisung PRINT auf derselben Zeile mit denselben Zwischenräumen. Endet die Liste der Ausdrücke ohne Komma, Semikolon oder einer Funktion SPC oder TAB, wird ein Wagenrücklauf an das Ende der Zeile angesetzt, d.h. BASIC setzt den Positionsanzeiger an den Beginn der nächsten Zeile.

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

Das Verschieben nach oben läuft ab, wie unter "Textmodus" in Kapitel 3 beschrieben.

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 Festpunktformat dargestellt werden können ohne an Genauigkeit zu verlieren, werden im Festpunktformat oder ganzzahligen Format ausgegeben. Z. B. wird 10^(−7) als .0000001 und 10^(−8) als 1E-8 ausgegeben.

BASIC fügt automatisch ein Wagenrücklauf-/Zeilenvorschubzeichen ein, nachdem x gleich 40 oder 80 ist, wie in der Anweisung WIDTH definiert. Dadurch werden zwei Zeilen übersprungen, wenn genau 40 (oder 80) Zeichen angezeigt werden, außer die Anweisung PRINT endet mit einem Semikolon (;).

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

Beispiel:

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

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

```
Ok
10 INPUT
20 PRINT X "QUADRIERT IST" X^2 "UND";
30 PRINT X "HOCH 3 IST" X^3
RUN
? 9
 9 QUADRIERT IST 81 UND 9 HOCH 3 IST 729
Ok
RUN
? 21
 21 QUADRIERT IST 441 UND 21 HOCH 3 IST 9261
Ok
```

PRINT
Anweisung

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

```
Ok
10 FOR X = 1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT X
RUN
 5  10  10  20  15  30  20  40  25  50
Ok
```

Hier werden durch die Semikolons in der Anweisung PRINT alle Werte sofort hinter dem vorherigen Wert ausgegeben. (Nicht vergessen, daß jeder Zahl immer eine Leerstelle folgt und positive Zahlen mit einer Leerstelle beginnen.) In Zeile 40 wird statt des Wortes PRINT ein ? benutzt.

PRINT USING
Anweisung

Zweck: Gibt Zeichenketten und Zahlen mit Hilfe eines Formats aus.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: PRINT USING *v$; Liste der Ausdrücke* [;]

Bemerkungen: *v$* ist eine Zeichenkettenkonstante oder Variable, die aus speziellen Formatierungszeichen besteht. Diese Formatierungszeichen (siehe unten) bestimmen das Feld und das Format der angezeigten Zeichenketten und Zahlen.

Liste der Ausdrücke
Zeichenkettenausdrücke oder numerische Ausdrücke, die getrennt durch Semikolons (;) oder Kommata (,) angezeigt werden.

PRINT USING
Anweisung

Zeichenkettenfelder

Wenn mit PRINT USING Zeichenketten angezeigt werden, kann für das Format der Zeichenkettenfelder eines von drei Formatierungszeichen 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 Schrägstriche ohne Leerstellen eingegeben, werden zwei Zeichen angezeigt, mit einer Leerstelle werden drei Zeichen angezeigt usw.

Ist die Zeichenkette länger als das Feld, werden die überzähligen Zeichen ignoriert. Ist das Feld länger als die Zeichenkette, wird die Zeichenkette linksbündig im Feld gespeichert und rechts mit Leerstellen aufgefüllt.

Beispiel:

```
10 A$="LOOK": B$="OUT"
30 PRINT USING "!";A$;B$
40 PRINT USING "\  \";A$;B$
50 PRINT USING "\   \".A$,B$/"!!"
RUN
LO
LOOKOUT
LOOK OUT  !!
```

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

```
10 A$="LOOK": B$="OUT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
LOUT
```

PRINT USING
Anweisung

Numerische Felder

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

#	Mit dem Nummernzeichen kann jede Ziffernposition dargestellt werden. Ziffernpositionen werden immer aufgefüllt. Hat die darzustellende Zahl weniger Ziffern als angegebene Positionen, wird die Zahl rechtsbündig 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 (0, falls notwendig). Zahlen werden falls nötig gerundet.

```
PRINT USING "##.##";.78
 Ø.78

PRINT USING "###.##";987.654
987.65

PRINT USING "##.##    ";1Ø.2,5.3,66.789,.234
1Ø.2Ø    5.3Ø   66.79    Ø.23
```

Im letzten Beispiel wurden am Ende der Formatzeichenkette drei Leerstellen eingefügt, 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.

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

```
PRINT USING "+##.##    ";-68.95,2.4,55.6,-.9
-68.95     +2.40    +55.6Ø     -Ø.9Ø

PRINT USING "##.##-    ";-68.95,22.449,-7.Ø1
68.95-   22.45     7.Ø1-
```

**

Durch zwei Sterne am Beginn der Formatzeichenkette werden führende Leerstellen eines numerischen Feldes mit Sternen aufgefüllt.

Die zwei Sterne definieren auch die Positionen für zwei weitere Ziffern.

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

$$

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

```
PRINT USING "$$###.##";456.78
 $456.78
```

**$

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

```
PRINT USING "**$##.##";2.34
***$2.34
```

PRINT USING
Anweisung

,

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

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

```
PRINT USING "####,.##";1234.5
1,234.5Ø

PRINT USING "####.##,";1234.5
1234.5Ø,
```

^ ^ ^ ^ Vier Fehlzeichen können hinter die Ziffernpositionszeichen geschrieben werden, um das Exponentialformat zu definieren. Durch die vier Fehlzeichen wird Platz gelassen, um E±nn oder D±nn darzustellen. Jede Dezimalpunktposition kann angegeben werden. Die signifikanten Ziffern werden linksbündig angelegt und der Exponent errechnet. Ist kein führendes + oder nachstehendes + oder – angegeben, wird eine Ziffernposition links des Dezimalpunkts benutzt, um eine Leerstelle oder ein Minuszeichen auszugeben.

```
Ok
PRINT USING "##.##^^^^";234.56
 2.35E+Ø2
Ok
PRINT USING ".###^^^^-";-88888
.889E+Ø5-
Ok
PRINT USING "+.##^^^^";123
+.12E+Ø3
Ok
```

_ Durch ein Unterstreichungszeichen in der Formatzeichenkette wird das nächste 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 größer 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.ØØ
Ok
```

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

Beispiel: In diesem Beispiel wird gezeigt, wie Zeichenkettenkonstanten in eine Formatzeichenkette eingefügt werden können:

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

PRINT # und PRINT # USING Anweisungen

Zweck: Schreibt Daten sequentiell in eine Datei.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: PRINT *#Dateinummer,* [USING *v$*;] *Liste der Ausdrücke*

Bemerkungen: *Dateinummer*
Nummer, die benutzt wurde, als die Datei für Ausgabe eröffnet wurde.

v$ ist ein Zeichenkettenausdruck, der aus den Formatierungszeichen besteht, die im vorherigen Abschnitt "Anweisung PRINT USING" beschrieben wurden.

Liste der Ausdrücke
Liste der numerischen und/oder Zeichenkettenausdrücke, 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 würden. Deshalb müssen die Daten in der Datei unbedingt mit Trennzeichen versehen werden, so daß sie als Eingabe korrekt aus der Datei gelesen werden können:

In der Liste der Ausdrücke müssen numerische Ausdrücke immer durch Semikolon (;) getrennt werden. Beispiel:

```
PRINT #1,A;B;C;X;Y;Z
```

(Werden Kommata (,) als Trennzeichen benutzt, werden die extra Leerzeichen, die zwischen den Druckfeldern eingefügt werden, auch in die Datei geschrieben.)

Zeichenkettenausdrücke müssen mit Hilfe von Semikolons in der Liste getrennt werden. Um die Zeichenkettenausdrücke richtig in der Datei zu formatieren, müssen explizite Trennzeichen in der Liste der Ausdrücke 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, könnte dies nicht als Ausgabe für zwei separate Zeichenketten benutzt werden. Um dieses Problem zu umgehen, müssen explizite Trennzeichen in die Anweisung PRINT# wie folgt eingefügt werden:

```
PRINT #1,A$;",";B$
```

Das Abbild, das jetzt in die Datei geschrieben wird, ist wie folgt:

```
CAMERA,936Ø4-1
```

Dies kann wieder in zwei Zeichenkettenvariablen zurückgelesen werden.

Enthalten die Zeichenketten Kommata, Semikolons, signifikante führende Leerstellen, Wagenrücklauf oder Zeilenvorschub, müssen sie in die Datei in eingeschlossenen Anführungszeichen mit Hilfe von CHR$(34) geschrieben werden.

Beispiel: A$="CAMERA, AUTOMATIC und B$=" 93604-1".

Die Anweisung

```
PRINT #1,A$;B$
```

schreibt das folgende Abbild in die Datei:

```
CAMERA, AUTOMATIC 936Ø4-1
```

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, müssen in das Abbild für die Datei mit Hilfe von CHR$(34) doppelte Anführungszeichen 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""    936Ø4-1"
```

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 #" am Ende dieses Kapitels).

Beispiel: Weitere Beispiele über PRINT # und WRITE # sind in Anhang B. "BASIC-Diskettenein- und -ausgabe" enthalten.

PSET und PRESET Anweisungen

Zweck: Zeichnet einen Punkt an eine angegebene Stelle auf dem Bildschirm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Nur im Grafikmodus.

Format: PSET (*x,y*) [,*Farbe*]

PRESET (*x,y*) [,*Farbe*]

Bemerkungen: (*x,y*) sind die Koordinaten des zu setzenden Punktes. Sie können in absoluter oder relativer Form angegeben werden, wie im Abschnitt "Spezifikation von Koordinaten" unter "Grafischer Modus" in Kapitel 3 beschrieben.

PSET und PRESET Anweisungen

Farbe Definiert die zu benutzende Farbe im Bereich 0 bis 3. In der mittleren Auflösung wählt *Farbe* die Farbe der laufenden Palette aus, wie mit der Anweisung COLOR definiert. 0 (Null) ist die Hintergrundfarbe.
Die Standardfarbe ist die Vordergrundfarbe, Farbe Nummer 3. In der hohen Auflösung kennzeichnet *Farbe* gleich 0 schwarz, die Standardannahme von 1 kennzeichnet weiß. In der hohen Auflösung 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 für PRESET nicht der Parameter *Farbe* angegeben wird - die Hintergrundfarbe (0) ausgewählt wird. Ist *Farbe* angegeben, ist PRESET gleich PSET. Die Zeile 70 im unteren Beispiel könnte auch wie folgt aussehen:

```
7Ø PSET(I,I),Ø
```

Wird für PSET oder PRESET eine Koordinate angegeben, die sich nicht im Bereich befindet, wird nichts ausgeführt und auch kein Fehler angezeigt. Ist *Farbe* größer als 3, wird der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf) angezeigt.

Beispiel:

Mit den Zeilen 20 bis 40 dieses Beispiels wird eine Diagonale von dem Punkt (0,0) zum Punkt (100,100) gezeichnet. Die Zeilen 60 bis 80 löschen diese Diagonale, indem sie jeden Punkt auf die Farbe 0 setzen.

```
10 SCREEN 1
20 FOR I=0 TO 100
30 PSET (I,I)
40 NEXT
50 'Löschen Linie
60 FOR I=100 TO 0 STEP -1
70 PRESET(I,I)
80 NEXT
```

PUT
Anweisung (Dateien)

Zweck: Schreibt einen Satz aus dem Puffer für wahlfreien Zugriff in eine Datei mit wahlfreiem Zugriff.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: PUT [#] *Dateinummer* [,*Nummer*]

Bemerkungen: *Dateinummer*
Nummer, unter der die Datei eröffnet wurde.

Nummer Satznummer für einen zu schreibenden Satz im Bereich 1 bis 32767.

Wird *Nummer* weggelassen, bekommt der Satz die nächste verfügbare Satznummer (nach dem letzten PUT).

Bevor die Anweisung PUT ausgeführt wird, können mit PRINT #, PRINT # USING, WRITE #, LSET und RSET Zeichen in den Dateipuffer für wahlfreien Zugriff gebracht werden. Im Falle von WRITE # füllt BASIC den Puffer mit Leerstellen bis zum Wagenrücklauf auf.

PUT
Anweisung (Dateien)

Jeder Versuch, über das Ende des Puffers hinauszuschreiben oder zu lesen, ergibt den Fehler "Field Overflow" (Feldüberlauf). Siehe Anhang B. "BASIC-Diskettenein- und -ausgabe".

Da BASIC und DOS soviel Sätze wie möglich in 512-Byte-Sektoren blockt, führt die Anweisung PUT nicht notwendigerweise ein physisches Schreiben auf die Diskette aus.

PUT kann für eine Datenfernverarbeitungsdatei benutzt werden. In diesem Fall bedeutet *Nummer* die Anzahl der Bytes, die in die Datenfernverarbeitungsdatei geschrieben werden soll. Diese Zahl muß kleiner oder gleich dem Wert sein, der mit der Auswahl LEN in der Anweisung OPEN "COM... gesetzt wurde.

Beispiel: Siehe Anhang B."BASIC-Diskettenein- und -ausgabe".

PUT
Anweisung (Grafik)

Zweck: Färbt einen angegebenen Bereich des Bildschirms.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	***

Nur Grafikmodus.

Format: PUT (*x,y*) ,*Bereich* [,*Aktion*]

Bemerkungen:

(*x,y*) sind die Koordinaten der linken oberen Ecke des zu übertragenden Bildes.

Bereich Name eines numerischen Bereichs, der die zu übertragende Information enthält. Im Abschnitt "Anweisung (Grafik) GET" in diesem Kapitel sind weitere Informationen über diesen Bereich enthalten.

PUT
Anweisung (Grafik)

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, ein Wert von 0 (Null) im Bereich bewirkt, daß der zugehörige Punkt die Farbe Nummer 3 bekommt und umgekehrt. Ein Wert von 1 im Bereich bewirkt, daß der korrespondierende Punkt die Farbe Nummer 2 bekommt und umgekehrt.

PUT
Anweisung (Grafik)

AND wird benutzt, wenn ein Abbild nur übertragen werden soll, falls das Abbild schon unter dem übertragenen Abbild existiert.

OR wird benutzt, um ein Abbild über ein schon existierendes Abbild zu legen.

XOR ist ein Spezialmodus, der für 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 Veränderung neu dargestellt. Dadurch kann ein Objekt bewegt werden, ohne daß der Hintergrund gelöscht wird.

In der mittleren Auflösung 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

PUT
Anweisung (Grafik)

Bewegung eines Objekts kann wie folgt durchgeführt 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 Schritt 1 zurückgehen und diesmal das Objekt an den neuen Platz stellen.

Eine solche Bewegung läßt den Hintergrund unverändert. Das Flackern des Bildschirms kann durch Verkürzung der Zeit zwischen den Schritten 4 und 1 reduziert werden. Auch sollte zwischen den Schritten 1 und 3 genügend Zeitverzögerung bestehen. Soll mehr als ein Objekt bewegt werden, sollten alle Objekte zusammen schrittweise zur gleichen Zeit verarbeitet werden.

PUT
Anweisung (Grafik)

Falls es nicht wichtig ist, den Hintergrund zu erhalten, kann Bewegung auch durch das Aktionsverb in PSET erzielt werden. Man benötigt jedoch einen Abbildungsbereich, der die Abbildungen des Objekts zuvor und danach enthält. Auf diese Weise löscht dieser Extrabereich das alte Abbild. Diese Methode ist etwas schneller als die Methode mit Hilfe von XOR wie oben beschrieben, da nur ein PUT benötigt wird, um das Objekt zu bewegen (obwohl mit einem größeren Abbild gearbeitet wird).

Ist die zu übertragende Abbildung zu groß, um auf den Bildschirm zu passen, erscheint der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

RANDOMIZE Anweisung

Zweck: Übergibt einen neuen Anfangswert für den Zufallszahlengenerator.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: RANDOMIZE [*n*]

Bemerkungen: *n* ist ein ganzzahliger Ausdruck, der als Anfangswert für Zufallszahlen benutzt wird.

Wird *n* weggelassen, verläßt BASIC die Programmausführung und fordert durch folgende Anzeige einen Wert:

```
Random Number Seed (Zufallszahlenwert) (-32768 to (bis) 32767)?
```

Danach wird RANDOMIZE ausgeführt.

RANDOMIZE Anweisung

Erhält der Zufallszahlengenerator keinen neuen Anfangswert, übergibt die Funktion RND jedesmal, wenn das Programm ausgeführt wird, die gleiche Folge von Zufallszahlen. Damit diese Folge bei jedem Programmlauf geändert wird, muß die Anweisung RANDOMIZE am Anfang des Programms stehen, und der Anfangswert für jeden Lauf muß geändert werden.

Im Disk- und erweiterten BASIC kann die Zeituhr verwendet werden, um einen Anfangswert für den Zufallszahlengenerator zu erhalten. Man kann z. B. mit VAL die letzten zwei Ziffern von TIME$ in eine Zahl umwandeln und diese Zahl für *n* benutzen.

RANDOMIZE
Anweisung

Beispiel:

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

Die Antwort soll 3 sein. Das Programm fährt wie folgt fort:

```
Random Number Seed (-32768 to 32767)? 3
 .7655695   .3558607   .3742327   .1388798
Ok
RUN
Random Number Seed (-32768 to 32767)?
```

Diesmal soll mit 4 geantwortet werden. Das Programm fährt fort:

```
Random Number Seed (-32768 to 32767)? 4
 .1719568   .5273236   .6879686   .713297
Ok
RUN
Random Number Seed (-32768 to 32767)?
```

Wird wieder die 3 versucht, bekommt man die gleiche Zahlenfolge wie beim ersten Lauf:

```
Random Number Seed (-32768 to 32767)? 3
 .7655695   .3558607   .3742327   .1388798
Ok
```

READ
Anweisung

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: READ *Variable* [,*Variable*]...

Bemerkungen: *Variable* Numerische oder Zeichenkettenvariable oder ein Bereichselement, 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 auf 1 : 1-Basis zugeordnet. Die Variablen der Anweisung READ müssen numerische oder Zeichenkettenvariablen sein, und die gelesenen Werte müssen mit dem Variablentyp übereinstimmen. Stimmen sie nicht überein, wird ein "Syntax Error" (Syntaxfehler) angezeigt.

READ
Anweisung

Mit READ können eine oder mehrere Anweisungen DATA angesprochen werden (sie werden immer in Reihenfolge abgearbeitet). Auch können mehrere Anweisungen READ die gleiche Anweisung DATA ansprechen. Ist die Anzahl der Variablen in der Liste größer 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:

```
8Ø FOR I=1 TO 1Ø
9Ø READ A(I)
1ØØ NEXT I
11Ø DATA 3.Ø8,5.19,3.12,3.98,4.24
12Ø DATA 5.Ø8,5.55,4.ØØ,3.16,3.37
```

Dieser Programmteil liest Werte der Anweisungen DATA in den Bereich A. Nach der Ausführung enthält A(1) den Wert 3.08 usw.

```
Ok
1Ø PRINT "STADT", "STAAT", " PLZ"
2Ø READ C$,S$,Z
3Ø DATA "DENVER,", COLORADO, 8Ø211
4Ø PRINT C$,S$,Z
RUN
STADT      STAAT              PLZ
DENVER,    COLORADO           8Ø211
Ok
```

In diesem Programm werden Zeichenketten und numerische Daten aus der Anweisung DATA in Zeile 30 gelesen. COLORADO muß nicht in Anführungszeichen (") stehen, da es keine Kommata, Semikolon oder signifikante führende oder nachstehende Leerstellen enthält. "DENVER," muß in Anführungszeichen stehen, da es ein Komma enthält.

REM
Anweisung

Zweck: Einfügen erklärender Bemerkungen in ein Programm.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format: REM *Bemerkung*

Bemerkungen: *Bemerkung*
kann irgendeine Zeichenfolge sein.

Anweisungen REM werden nicht ausgeführt, werden aber ausgegeben, wenn das Programm aufgelistet wird. Sie verlängern jedoch die Ausführungszeit und benötigen Platz im Hauptspeicher.

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

REM Anweisung

An das Ende einer Zeile können auch Bemerkungen hinzugefügt werden, indem man vor die Bemerkungen ein Apostroph setzt. Soll die Bemerkung in die gleiche Zeile kommen wie andere BASIC-Anweisungen, muß die Bemerkung die letzte Anweisung der Zeile sein.

Beispiel:

```
100 REM Berechnen mittlere Geschwindigkeit
110 SUM=0: REM Initialisieren SUM
120 FOR I=1 TO 20
130 SUM=SUM + V(I)
```

In Zeile 10 könnte auch stehen:

```
110 SUM=0 ' Initialisieren SUM
```

RENUM Befehl

Zweck: Neue Numerierung von Programmzeilen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: 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 ändert auch alle Zeilennummerreferenzen, die auf GOTO-, GOSUB-, THEN-, ELSE-, ON...GOTO-, ON...GOSUB-, RESTORE-, RESUME- und ERL-Testanweisungen folgen, um den neuen Zeilennummern zu entsprechen. Wird dabei eine Zeilennummer gefunden, die nicht existiert, wird die Nachricht "Undefined Line Number xxxxx in yyyyy" (Nicht definierte Zeilennummer xxxxx in yyyyy) gedruckt. Die ungültige Zeilennummerreferenz (xxxxx) wird von RENUM nicht geändert. Die Zeilennummer yyyyy kann aber geändert sein.

Hinweis: Mit RENUM kann die Reihenfolge der Programmzeilen nicht verändert werden (z. B. RENUM 15,30, wenn das Programm drei Zeilen mit den Nummern 10, 20 und 30 hat) oder Zeilennummern größer als 65529 erzeugt werden. Es ergibt sich der Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

RENUM Befehl

Beispiel:

```
RENUM
```

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

```
RENUM 3ØØ,,5Ø
```

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

```
RENUM 1ØØØ,9ØØ,2Ø
```

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

RESET Befehl

Zweck: Schließt alle Diskettendateien ab und löscht den Systempuffer.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: RESET

Bemerkungen: Sind alle Dateien auf der Diskette eröffnet, wirkt RESET wie CLOSE ohne Angabe von Dateinummern.

RESTORE
Anweisung

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

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: RESTORE [*Zeile*]

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

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

RESTORE
Anweisung

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.

RESUME
Anweisung

Zweck: Fährt mit der Programmausführung fort, nachdem eine Fehlerbehandlungsroutine ausgeführt wurde.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format:

RESUME [0]

RESUME NEXT

RESUME *Zeile*

Bemerkungen: Jedes der oben gezeigten Formate kann benutzt werden, abhängig davon, wo die Ausführung wieder aufsetzen soll:

RESUME oder RESUME 0

Die Ausführung 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 enthält, tritt der Fehler "Undefined Line Number" (Nicht definierte Zeilennummer) auf. Die Anweisung heißt weiter RESUME 0 und ist damit richtig.

RESUME NEXT

Die Ausführung geht mit der Anweisung weiter, die nach der Anweisung steht, die den Fehler verursachte.

RESUME Zeile

Die Ausführung geht an der angegebenen Zeilennummer weiter.

RESUME
Anweisung

Steht die Anweisung RESUME nicht in einer Fehlerbehandlungsroutine, erscheint die Nachricht "RESUME without Error" (RESUME ohne Fehler).

Beispiel:

```
1Ø ON ERROR GOTO 9ØØ
 .
 .
 .
9ØØ IF (ERR=23Ø)AND(ERL=9Ø) THEN PRINT
            "NEUER VERSUCH": RESUME 8Ø
```

Zeile 900 ist der Beginn der Fehlerbehandlungsroutine. Durch die Anweisung RESUME wird zu Zeile 80 verzweigt, falls der Fehler 230 in Zeile 90 auftrat.

RETURN Anweisung

Zweck: Verzweigt aus einem Unterprogramm zurück. Siehe "Anweisungen GOSUB und RETURN" in diesem Kapitel.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: RETURN [*Zeile*]

Zeile ist nur im erweiterten BASIC und im Umwandlungsprogramm gültig.

RETURN
Anweisung

Bemerkungen: *Zeile* Zeilennummer des Programms, zu der zurückverzweigt werden soll.

Obwohl die Benutzung von *Zeile* in RETURN in jedem Unterprogramm verwendet werden kann, wurde diese Erweiterung für das Zurückverzweigen aus den nicht lokalen Ereignisunterbrechungsroutinen hinzugefügt. Aus einer dieser Routinen will man im BASIC-Programm oft an eine feste Zeilennummer zurückverzweigen 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.

RIGHT$ Funktion

Zweck: Übergibt die rechtsliegenden *n* Zeichen der Zeichenkette *x$*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *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* größer oder gleich LEN(*x$*), wird *x$* übergeben. Ist *n* gleich Null, wird eine leere Zeichenkette (Länge Null) übergeben.

Siehe auch die Funktionen MID$ und LEFT$.

RIGHT$
Funktion

Beispiel:

```
Ok
1Ø A$="BOCA RATON, FLORIDA"
2Ø PRINT RIGHT$(A$,7)
RUN
FLORIDA
Ok
```

Die am weitesten rechts stehenden sieben Zeichen der Zeichenkette A$ werden übergeben.

RND Funktion

Zweck: Übergibt eine Zufallszahl zwischen 0 und 1.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: v = RND[(x)]

Bemerkungen: x ist ein numerischer Ausdruck, der den übergebenen Wert wie unten beschrieben beeinflußt.

Bei jeder Programmausführung wird die gleiche Folge von Zufallszahlen generiert, wenn der Anfangswert für den Zufallszahlengenerator nicht neu gesetzt wird. Dies kann sehr leicht mit Hilfe der Anweisung RANDOMIZE geschehen (siehe "Anweisung RANDOMIZE" in diesem Kapitel). 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 für ein gegebenes x erzeugt.Auf diese Folge hat RANDOMIZE keinen Einfluß. Soll also für jede Programmausführung eine andere Folge generiert werden, muß dazu jedesmal für x ein neuer Wert genommen werden.

RND
Funktion

Ist *x* positiv oder nicht eingefügt, erzeugt RND(*x*) die nächste Zufallszahl in Folge.

RND(0) wiederholt die letzte generierte Zahl.

Mit Hilfe der Formel:

INT(RND * (*n*+1))

erhält man Zufallszahlen im Bereich 0 bis *n*.

Beispiel:

```
Ok
10 FOR I=1 TO 3
20 PRINT RND(I);     ' x>0
30 NEXT I
40 PRINT: X=RND(-6) ' x<0
50 FOR I=1 TO 3
60 PRINT RND(I);     ' x>0
70 NEXT I
80 RANDOMIZE 853 'neuer Anfangswert
90 PRINT: X=RND(-6) ' X<0
100 FOR I=1 TO 3
110 PRINT RND;    ' gleich wie x>0
120 NEXT I
130 PRINT: PRINT RND(0)
RUN
 .6291626   .1948297   .6305799
 .6818615   .4193624   .6215937
 .6818615   .4193624   .6215937
 .6215937
Ok
```

In der ersten waagerechten Ergebniszeile stehen drei Zufallszahlen, die mit einem positiven *x* erzeugt wurden.

In Zeile 40 wird mit Hilfe der negativen Zahl für den Zufallszahlengenerator ein neuer Anfangswert gesetzt. Die jetzt erzeugten Zufallszahlen stehen in der zweiten Ergebniszeile.

In Zeile 80 wird dem Zufallszahlengenerator mit Hilfe der Anweisung RANDOMIZE ein neuer Anfangswert zugeordnet. In Zeile 90 wird wieder durch Aufruf von RND mit dem gleichen negativen Wert wie in Zeile 40 ein neuer Anfangswert zugeordnet. Damit wird der mit der Anweisung RANDOMIZE zugeordnete Wert ersetzt. Die dritte Ergebniszeile entspricht der zweiten Zeile.

In Zeile 130 wird RND mit dem Argument 0 aufgerufen, so daß die letzte ausgegebene Zahl gleich der vorletzten Zahl ist.

RUN Befehl

Zweck: Beginnt die Programmausführung.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: RUN [*Zeile*]

RUN *Dateiangabe*[,R]

Bemerkungen: *Zeile* Zeilennummer des Programms im Hauptspeicher, an der mit der Ausführung begonnen werden soll.

Dateiangabe
Zeichenkettenausdruck für eine Dateispezifikation, wie unter "Namensgebung für Dateien" in Kapitel 3 beschrieben. Die Standarderweiterung .BAS wird für Diskettendateien angehängt.

RUN Befehl

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

RUN *Dateiangabe* lädt eine Datei von Diskette oder Kassette in den Hauptspeicher und führt sie aus. Alle eröffneten Dateien werden geschlossen und der laufende Inhalt des Hauptspeichers wird gelöscht, bevor das ausgewählte Programm geladen wird. Mit der Angabe R bleiben jedoch alle Datendateien eröffnet. Siehe auch Anhang B. "BASIC-Diskettenein- und -ausgabe".

Durch die Ausführung des Befehls RUN wird jeglicher Ton abgestellt und die Musik in den Vordergrund gesetzt. Auch PEN und STRIG werden auf OFF (AUS) gesetzt.

RUN Befehl

Beispiel:

```
Ok
10 PRINT 1/7
RUN
 .1428571
Ok
10 PI=3.141593
20 PRINT PI
RUN 20
 0
Ok
```

Im ersten Beispiel wird die erste Form von RUN mit zwei sehr kleinen Programmen benutzt. Das erste Programm wird von Anfang an ausgeführt. Im zweiten Beispiel wird die Angabe *Zeile* in RUN gegeben, um das Programm ab Zeile 20 auszuführen. In diesem Fall wird Zeile 10 nicht ausgeführt, so daß PI nicht den richtigen Wert erhält. Eine 0 (Null) wird ausgegeben, weil alle numerischen Variablen zu Anfang auf 0 gesetzt werden.

```
RUN "CAS1:NEWFIL",R
```

Im vorherigen Beispiel wird das Programm "NEWFIL" vom Band geladen und ausgeführt, wobei die Dateien geöffnet bleiben.

SAVE Befehl

Zweck: Speichert eine BASIC-Programmdatei auf Diskette oder Kassette.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	

Format: SAVE *Dateiangabe* [,A]

SAVE *Dateiangabe* [,P]

Bemerkungen: *Dateiangabe*

Zeichenkettenausdruck für eine Dateispezifikation. Entspricht *Dateiangabe* nicht den Regeln, die unter "Namensgebung für Dateien" in Kapitel 3 beschrieben sind, wird ein Fehler angezeigt und das Speichern abgebrochen.

Das BASIC-Programm wird in die angegebene Einheit geschrieben. Wird nach CAS1: gespeichert, wird der Kassettenmotor eingeschaltet und die Datei sofort auf Band geschrieben.

SAVE Befehl

Enthält bei Diskettendateien der Dateiname acht Zeichen oder weniger und es fehlt eine Erweiterung, wird die Erweiterung .BAS an den Namen angefügt. Steht schon eine Datei mit dem gleichen Namen auf der Diskette, wird sie überschrieben.

Fehlt bei Benutzung von Kassetten-BASIC der Einheitenname, wird CAS1: angenommen. CAS1: ist die einzige erlaubte Einheit für SAVE im Kassetten-BASIC.

Im Disk- oder erweiterten BASIC ist die Standardeinheit die Standardeinheit für DOS.

Mit der Angabe **A** wird das Programm im ASCII-Format gespeichert; andernfalls speichert BASIC die Datei in einem komprimierten Binärformat.
ASCII-Dateien benötigen mehr Platz, aber einige Zugriffsarten benötigen Dateien im ASCII-Format.

Eine Datei, die mit einem Programm gemischt werden soll, muß im ASCII-Format gespeichert sein. Programme, die im ASCII-Format gespeichert sind, können als Datendateien gelesen werden.

Mit der Angabe P können Programme geschützt gespeichert werden. Solche geschützten Programme können später ausgeführt (oder geladen) werden, jeder Versuch, sie mit LIST oder EDIT anzusprechen, ergibt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf). Es ist nicht möglich, ein geschütztes Programm wieder in den ungeschützen Status zu bringen.

> **Hinweis:** Das Inhaltsverzeichnis für Disketten zeigt für BASIC-Programme nicht an, daß sie geschützt oder im ASCII-Format gespeichert sind. In jedem Falle wird die Erweiterung .BAS benutzt.

Siehe auch Anhang B.
"BASIC-Diskettenein- und -ausgabe."

SAVE Befehl

Beispiele:

```
SAVE "INVENT"
```

Das Programm, das sich im Hauptspeicher befindet, wird unter dem Namen INVENT gespeichert. Das Programm wird auf Kassette gespeichert, wenn mit Kassetten-BASIC gearbeitet wird. Wird mit Disk- oder erweitertem BASIC gearbeitet, wird das Programm auf Diskette in die Standardeinheit für DOS gespeichert und die Erweiterung .BAS angefügt.

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

Das Programm PROG.BAS wird in Einheit B im ASCII-Format gespeichert, so daß es später mit Programmen im Hauptspeicher gemischt werden kann.

```
SAVE "A:SECRET.BOZ",P
```

Das Programm SECRET.BOZ wird in Einheit A: geschützt gespeichert, so daß es nicht verändert werden kann.

SCREEN Funktion

Zweck: Übergibt den ASCII-Code (0 bis 255) für ein Zeichen auf dem aktiven Bildschirm an der angegebenen Zeile und Spalte.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *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, abhängig vom Setzen der Breite WIDTH.

z ist ein numerischer Ausdruck, dem der Wert wahr oder falsch zugeordnet wird. *z* ist nur im Textmodus gültig.

Siehe Anhang G. "ASCII-Zeichencodes". Er enthält eine Liste aller ASCII-Codes.

SCREEN Funktion

Ist im Textmodus *z* eingefügt und wahr (nicht Null), wird das Farbattribut für dieses Zeichen statt des Zeichencodes übergeben. 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* - *Vordergrund*)/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 "Anweisung COLOR" befindet sich die Liste aller Farben und ihrer zugehörigen Nummern.

Enthält die angegebene Stelle im grafischen Modus grafische Informationen (Punkte oder Linien im Gegensatz zu einem Zeichen), übergibt die Funktion SCREEN eine 0 (Null).

Jeder eingegebene Wert außerhalb des Bereichs erzeugt den Fehler "Illegal Function Call" (Ungültiger Funktionsaufruf).

Die Anweisung SCREEN wird im nächsten Abschnitt beschrieben.

Beispiel:

```
100 X = SCREEN (10,10)
```

Ist das Zeichen an der Position 10,10 ein A, ist X gleich 65.

```
110 X = SCREEN (1,1,1)
```

Dieses Beispiel übergibt das Farbattribut des Zeichens in der oberen linken Ecke des Bildschirms.

SCREEN
Anweisung

Zweck: Setzt die Bildschirmattribute, die von den nachfolgenden Anweisungen benutzt werden sollen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Nur mit dem Anschluß für den Farb-/Grafikbildschirm von Bedeutung.

Format: SCREEN [*Modus*] [,[*ändern*] [,[*aseite*] [,*vseite*]]]

Bemerkungen: *Modus* Numerischer Ausdruck, dessen Ergebnis ein ganzzahliger Wert von 0, 1 oder 2 ist. Gültige Modi sind:

- **0** Textmodus mit der laufenden Breite (40 oder 80)
- **1** Mittlere Auflösung im grafischen Modus (320 x 320). Darf nur mit dem Anschluß für Farb-/Grafikbildschirm verwendet werden.

2 Grafischer Modus für hohe Auflösung (640 x 200). Darf mit dem Anschluß für Farb-/Grafikbildschirm benutzt werden.

ändern Numerischer Ausdruck, der sich als wahrer oder falscher Wert ergibt. Er aktiviert die Farbe. Im Textmodus (*Modus* = 0) inaktiviert ein falscher (0) Wert die Farbe (nur schwarze oder weiße Bilder), und ein wahrer (nicht 0) Wert aktiviert Farbe (erlaubt farbige Bilder). In mittlerer Auflösung für grafischen Modus (*Modus* = 1) inaktiviert ein wahrer (nicht 0) Wert die Farbe und ein falscher (0) Wert aktiviert Farbe. Da schwarz und weiß die einzigen Farben in der hohen Auflösung für Grafik sind (*Modus* = 2), hat dieser Parameter keinen großen Einfluß auf die hohe Aluflösung.

aseite (aktive Seite): Ganzzahliger Ausdruck im Bereich 0 bis 7 für die Breite 40 oder 0 bis 3 für die Breite 80. Damit wird die Seite ausgewählt, die mit Hilfe einer Ausgabeanweisung auf den Bildschirm geschrieben wird und ist nur im Textmodus (*Modus* = 0) gültig.

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) gültig. Wird der Parameter weggelassen, ist *vseite* Standardwert für *aseite.*

Sind alle Parameter gültig, wird der neue Bildschirmmodus gespeichert, der Bildschirm gelöscht, die Vordergrundfarbe auf weiß und der Hintergrund und die Bildrandfarben auf schwarz gesetzt.

Ist der neue Bildschirmmodus der gleiche wie der vorherige, wird nichts verändert.

Ist der Modus Text und sind nur *aseite* und *vseite* angegeben, werden die Bildschirmseiten zum ansehen abgeändert. Am Anfang ist der Standardwert für die aktiven und visuellen Seiten 0 (Null). Durch Behandeln der aktiven und visuellen Seiten kann man eine Seite anzeigen, während die andere aufgebaut wird. Damit kann man visuelle Seiten umschalten.

Werden Text und Grafiken im 40- oder 80-Zeilen-Grafikmodus vermischt und es wird keine U.S.-Tastatur verwendet, siehe "Farb-/Grafikadapterzeichen" im Abschnitt DOS des Bedienerhandbuchs.

Hinweis: Es gibt nur einen gemeinsamen Positionsanzeiger für alle Seiten. Wird bei aktiven Seiten auf vorherige und nachfolgende umgeschaltet, muß die Stelle des Positionsanzeigers 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 zurück, kann man die Stelle des Positionsanzeigers mit Hilfe der Anweisung LOCATE wieder herstellen.

Jeder Parameter kann weggelassen werden. Für weggelassene Werte, außer *vseite,* wird der alte Wert angenommen.

Alle Werte, die sich außerhalb des angegebenen Bereichs befinden, ergeben den Fehler "Illegal Function Call' (Ungültiger Funktionsaufruf).

Die alten Werte bleiben erhalten.

Soll ein Programm geschrieben werden, das auf einer Maschine laufen soll, die einen dieser Anschlüsse hat, wird vorgeschlagen, die Anweisungen SCREEN 0,0,0 und WIDTH 40 an den Anfang des Programms zu stellen.

SCREEN
Anweisung

Beispiele:

```
1Ø SCREEN Ø,1,Ø,Ø
```

Wählt Textmodus mit Farbe aus und setzt die aktive und visuelle Seite auf 0.

```
2Ø SCREEN ,,1,2
```

Modus und Farbänderung bleiben erhalten. Die aktive Seite wird auf 1 gesetzt und die Anzeigeseite auf 2.

```
3Ø SCREEN 2,,Ø,Ø
```

Schaltet in den grafischen Modus für hohe Auflösung um.

```
4Ø SCREEN 1,Ø
```

Schaltet auf Farbgrafiken in der mittleren Auflösung um.

```
5Ø SCREEN ,1
```

Setzt auf Grafik in mittlerer Auflösung und schaltet die Farbe aus.

SGN Funktion

Zweck: Übergibt das Vorzeichen von *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = SGN(*x*)

Bemerkungen: *x* ist irgendein numerischer Ausdruck.

SGN(*x*) ist die mathematische Signumfunktion:

- Ist *x* positiv, übergibt SGN(*x*) eine 1.
- Ist *x* gleich Null (0), übergibt SGN(*x*) eine 0.
- Ist *x* negativ, übergibt SGN(*x*) eine −1.

Beispiel:

```
ON SGN(X)+2 GOTO 1ØØ,2ØØ,3ØØ
```

In dieser Anweisung wird nach 100 verzweigt, falls X negativ, nach 200, falls X gleich 0 und nach 300, falls X positiv ist.

SIN
Funktion

Zweck: Errechnet den Sinuswert für einen Wert im Bogenmaß.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: v = SIN(x)

Bemerkungen: x ist der Winkel im Bogenmaße.

Um Gradmaße in Bogenmaße umzuwandeln, muß man mit PI/180 multiplizieren, wobei PI = 3.141593 ist.

SIN(x) wird mit einfacher Genauigkeit berechnet.

Beispiel:

```
Ok
10 PI=3.141593
20 DEGREES = 90
30 RADIANS=DEGREES * PI/180 ' PI/2
40 PRINT SIN(RADIANS)
RUN
 1
Ok
```

In diesem Beispiel wird der Sinus von 90 Grad errechnet, nachdem die Grade in Bogenmaß umgewandelt wurden.

SOUND Anweisung

Zweck: Erzeugt Töne über den Lautsprecher.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: SOUND *freq, Dauer*

Bemerkungen:

freq Gewünschte Frequenz in Hertz (Zyklen pro Sekunde). Es muß sich um einen numerischen Ausdruck im Bereich 37 bis 32767 handeln.

Dauer Gewünschte Dauer in Zeitschlägen. Die Zeitschläge kommen 18.2 mal pro Sekunde. *Dauer* muß ein numerischer Ausdruck im Bereich 0 bis 65535 sein.

SOUND
Anweisung

Erzeugt die Anweisung SOUND einen Ton, fährt das Programm mit der Ausführung fort, bis eine neue Anweisung SOUND erreicht wird. Ist *Dauer* der neuen Anweisung SOUND 0 (Null), 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 ausgeführt wird.

Im erweiterten BASIC können die Töne gepuffert werden, so daß die Ausführung nicht stoppt, wenn eine neue Anweisung SOUND von BASIC erkannt wird. Siehe Befehl **MB,** der unter "Anweisung PLAY" in diesem Kapitel in allen Einzelheiten erklärt wird.

Läuft 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 für zwei Oktaven auf jeder Seite des kleinen C 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

*kleines C. Höhere (oder tiefere) Noten können durch Verdoppelung (oder Halbierung) der Frequenz der korrespondierenden Note in der nächsttieferen (höheren) Oktave angenähert werden.

Zur Erstellung von Pausenperioden muß SOUND 32767,*Dauer* ausgeführt werden.

SOUND
Anweisung

Die Dauer eines Schlages kann aus den Schlägen pro Minute errechnet werden, indem man die Schläge pro Minute durch 1092 teilt (die Anzahl der Zeitschläge pro Minute).

Die nächste Tabelle zeigt typische Tempi als Ausdruck von Zeitschlägen:

	Tempo	Schläge/ Minute	Einheiten/ Schlag
Sehr	Larghissimo		
langsam	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: Das folgende Programm erstellt ein Glissando nach oben und nach unten:

```
1Ø FOR I=44Ø TO 1ØØØ STEP 5
2Ø SOUND I, Ø.5
3Ø NEXT
4Ø FOR I=1ØØØ TO 44Ø STEP -5
5Ø SOUND I, Ø.5
6Ø NEXT
```

SPACE$
Funktion

Zweck: Übergibt eine Zeichenkette, die aus *n* Leerstellen besteht.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = SPACES$(*n*)

Bemerkungen: *n* muß sich im Bereich 0 bis 255 befinden.

Siehe auch Funktion SPC.

SPACE$ Funktion

Beispiel:

```
Ok
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
  1
   2
    3
     4
      5
Ok
```

In diesem Beispiel wird die Funktion SPACE$ benutzt, um jede Zahl I in einer Zeile mit vorangestellten I Leerstellen auszugeben. Eine zusätzliche Leerstelle wird eingefügt, weil BASIC vor jede positive Zahl eine Leerstelle setzt.

SPC
Funktion

Zweck: Übergeht *n* Leerstellen in einer Anweisung PRINT.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: PRINT SPC(*n*)

Bemerkungen: *n* muß sich im Bereich 0 bis 255 befinden.

Ist *n* größer als die definierte Breite für die Einheit, wird als Wert *n* MOD *Breite* benutzt. SPC kann nur mit den Anweisungen PRINT, LPRINT und PRINT # benutzt werden.

Befindet sich die Funktion SPC am Ende einer Datenliste, fügt BASIC keinen Wagenrücklauf hinzu, da die Funktion SPC immer ein Semikolon (;) anfügt.

Siehe auch Funktion SPACE$.

Beispiel:

```
Ok
PRINT "HIER" SPC(15) "DORT"
HIER               DORT
Ok
```

Dieses Beispiel druckt HIER und DORT, getrennt durch 15 Leerstellen.

SQR Funktion

Zweck: Übergibt die Quadratwurzel von *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = SQR(*x*)

Bemerkungen: *x* muß größer oder gleich Null sein.

Beispiel:

```
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.

STICK
Funktion

Zweck: Übergibt die *x*- und *y*- Koordinaten von zwei Spielpulten.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = STICK(*n*)

Bemerkungen: *n* ist ein numerischer Ausdruck im Bereich 0 bis 3, der das Ergebnis wie folgt beeinflußt:

0 übergibt die x-Koordinate des Spielpults A.

1 übergibt die y-Koordinate des Spielpults A.

2 übergibt die x-Koordinate des Spielpults B.

3 übergibt die y-Koordinate des Spielpults B.

Hinweis: STICK(0) liest alle vier Werte für die Koordinaten und übergibt den Wert für STICK(0). STICK(1), STICK(2) und STICK(3) enthalten nicht den Wert des Spielpults. Sie erhalten ihre Werte über das vorherige Lesen mit STICK(0).

Der Bereich der Werte für x und y hängt von den Spielpulten ab.

Beispiel:

```
10 PRINT "Spielpult B"
20 PRINT "x Koordinate","y Koordinate"
30 FOR J=1 TO 100
40 TEMP=STICK(0)
50 X=STICK(2): Y=STICK(3)
60 PRINT X,Y
70 NEXT
```

Mit diesem Programm werden 100 Koordinatenbeispiele für das Spielpult B gelesen und angezeigt.

STOP
Anweisung

Zweck: Beendet die Programmausführung und kehrt zur Befehlsebene zurück.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: STOP

Bemerkungen: Anweisungen STOP können überall im Programm verwendet werden, um die Programmausführung zu beenden. Sobald BASIC eine Anweisung STOP ausführt, wird die folgende Nachricht angezeigt:

```
Break in nnnnn
```

nnnnn ist die Zeilennummer, in der STOP auftrat.

Die Anweisung STOP schließt Dateien nicht ab, im Gegensatz zu END.

BASIC kehrt immer zur Befehlsebene zurück, nachdem STOP ausgeführt wurde. Die Ausführung des Programms kann mit dem Befehl CONT wieder aufgenommen werden (siehe "Befehl CONT" in diesem Kapitel).

Beispiel:

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

In diesem Beispiel wird der Wert für TEMP errechnet, danach die Ausführung beendet. Während das Programm steht, kann der Wert von TEMP geprüft werden. Danach kann mit CONT die Programmausführung an Zeile 40 wieder aufgenommen werden.

STR$ Funktion

Zweck: Übergibt die Zeichenkettendarstellung des Wertes *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format: *v$* = STR$(*x*)

Bemerkungen: *x* ist ein numerischer Ausdruck.

Ist *x* positiv, enthält die mit STR$ übergebene Zeichenkette eine führende Leerstelle (die Leerstelle ist für das Plusvorzeichen reserviert). Beispiel:

```
Ok
 ? STR$(321); LEN(STR$(321))
 321 4
Ok
```

Die Funktion VAL ist die Gegenfunktion von STR$.

Beispiel: Das Beispiel verzweigt zu verschiedenen Abschnitten des Programms, abhängig von der Anzahl der Ziffern einer eingegebenen Zahl. Die Ziffern der Zahl werden dadurch gezählt, daß man mit Hilfe von STR$ die Zahl in eine Zeichenkette umwandelt und dann abhängig von der Länge der Zeichenkette verzweigt.

```
5 REM Arithmetik für Kinder
1Ø INPUT "ZAHL EINGEBEN";N
2Ø ON LEN(STR$(N))-1 GOSUB 3Ø,1ØØ,2ØØ,3ØØ
●
●
●
```

STRIG
Anweisung und Funktion

Zweck: Übergibt den Status der Spielpultknöpfe.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: Als Anweisung:

STRIG ON

STRIG OFF

Als Funktion:

v = STRIG(*n*)

Bemerkungen: *n* ist ein numerischer Ausdruck im Bereich 0 bis 3. Er beeinflußt die übergebenen Werte der Funktion wie folgt.

0 übergibt −1, falls der Knopf A1 seit dem letzten STRIG(0)-Funktionsaufruf gedrückt wurde, sonst wird 0 übergeben.

1 übergibt −1, falls der Knopf A1 gerade gedrückt wurde, sonst 0.

2 übergibt −1, falls der Knopf B1 seit dem letzten STRIG(2)-Funktionsaufruf gedrückt wurde, sonst wird 0 übergeben.

3 übergibt −1, falls der Knopf B1 gerade gedrückt wurde, sonst 0.

Im erweiterten BASIC und im BASIC-Umwandlungsprogramm können vier Knöpfe von den Spielpulten gelesen werden. Die zusätzlichen Werte für *n* sind:

4 übergibt −1, falls der Knopf A2 seit dem letzten STRIG(4)-Funktionsaufruf gedrückt wurde, sonst wird 0 übergeben.

5 übergibt −1, falls der Knopf A2 gerade gedrückt wurde, sonst 0.

6 übergibt −1, falls der Knopf B2 seit dem letzten STRIG(6)-Funktionsaufruf gedrückt wurde, sonst wird 0 übergeben.

7 übergibt −1, falls der Knopf B2 gerade gedrückt wurde, sonst 0.

STRIG
Anweisung und Funktion

STRIG ON muß ausgeführt sein, bevor ein STRIG(*n*)-Funktionsaufruf ausgeführt werden kann. Nach der Ausführung von STRIG ON prüft BASIC vor der Ausführung jeder neuen Anweisung, ob ein Knopf gedrückt wurde.

Wurde die Anweisung STRIG OFF ausgeführt, findet kein Test statt.

Siehe auch im nächsten Abschnitt "Anweisung STRIG(*n*)". Er enthält Erweiterungen der Funktion STRIG für das erweiterte BASIC.

STRIG(n) Anweisung

Zweck: Aktiviert und inaktiviert die Unterbrechung der Spielpultknöpfe.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
		***	(**)

Format:

STRIG(*n*) ON

STRIG(*n*) OFF

STRIG(*n*) STOP

Bemerkungen: *n* kann 0, 2, 4 oder 6 sein und ist den Knöpfen für die Unterbrechung wie folgt zugeordnet:

0	Knopf A1
2	Knopf B1
4	Knopf A2
6	Knopf B2

STRIG(*n*) ON muß ausgeführt sein, um eine Unterbrechung durch die Anweisung ON STRIG(*n*) zu aktivieren (siehe "Anweisung ON STRIG(*n*)" in diesem Kapitel). Nach der Ausführung der Anweisung STRIG(*n*) ON prüft das Programm, bevor eine neue Anweisung ausgeführt wird, ob der angegebene Knopf gedrückt wurde.

STRIG(n)
Anweisung

Wurde STRIG(*n*) OFF ausgeführt, findet weder Testen noch eine Unterbrechung statt. Wurde der Knopf gedrückt, wird dies nicht gespeichert.

Wurde die Anweisung STRIG(*n*) STOP ausgeführt, findet keine Unterbrechung statt. Es wird jedoch ein Knopfdruck gespeichert, so daß nach der Ausführung der Anweisung STRIG(*n*) ON eine sofortige Unterbrechung eintritt.

Siehe auch vorherigen Abschnitt "Anweisung und Funktion STRIG."

STRING$ Funktion

Zweck: Übergibt eine Zeichenkette der Länge *n*, deren Zeichen alle aus dem ASCII-Code *m* oder dem ersten Zeichen von *x$* bestehen.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v$* = STRING$(*n,m*)

*v$* = STRING$(*n,x$*)

Bemerkungen: *n,m* liegen im Bereich 0 bis 255.

x$ ist ein Zeichenkettenausdruck.

STRING$ Funktion

Beispiel:

```
Ok
10 X$ = STRING$(10,45)
20 PRINT X$ "MONATSBERICHT" X$
RUN
----------MONATSBERICHT----------
Ok
```

Im ersten Beispiel wird der ASCII-Wert von 45 wiederholt, um eine Zeichenkette aus Bindestrichen auszugeben.

```
Ok
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.

SWAP
Anweisung

Zweck: Tauscht die Werte zweier Variablen aus.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: SWAP *Variable1, Variable2*

Bemerkungen: *Variable1, Variable2*
Namen zweier Variablen oder Bereichselemente.

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

SWAP
Anweisung

Beispiel:

```
Ok
10 A$=" ONE " : B$=" ALL " : C$="FOR"
20 PRINT A$ C$ B$
30 SWAP A$,B$
40 PRINT A$ C$ B$
RUN
 ONE FOR ALL
 ALL FOR ONE
Ok
```

Nachdem Zeile 30 ausgeführt wurde, ist der Wert von A$ " ALL " und der Wert von B$ " ONE ".

SYSTEM Befehl

Zweck: Verzweigt aus BASIC und kehrt zu DOS zurück.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: SYSTEM

Bemerkungen: Durch SYSTEM werden alle Dateien abgeschlossen, bevor zu DOS zurückgekehrt wird. Das BASIC-Programm geht verloren.

Wurde zu BASIC durch eine Stapelverarbeitungsdatei in DOS verzweigt, verzweigt der Befehl SYSTEM zu der Stapelverarbeitungsdatei zurück und fährt mit der Verarbeitung an dem Punkt fort, an dem die Datei verlassen wurde.

TAB
Funktion

Zweck: Setzt an der Position *n* einen Tabulator.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: 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 nächsten 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, fügt BASIC keinen Wagenrücklauf an, da die Funktion TAB automatisch ein Semikolon (;) anfügt.

Beispiel: Mit Hilfe von TAB wird im nächsten Beispiel die Information auf dem Bildschirm in Spalten aufgeteilt:

```
Ok
10 PRINT "NAME" TAB(25) "SUMME" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "L. M. JACOBS","$25.00"
RUN
NAME                      SUMME

L. M. JACOBS              $25.00
Ok
```

TAN
Funktion

Zweck: Übergibt den Tangens von *x*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *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 mit einfacher Genauigkeit berechnet.

Beispiel:

```
Ok
10 PI=3.141593
20 DEGREES=45
30 PRINT TAN(DEGREES*PI/180)
RUN
 1
Ok
```

In diesem Beispiel wird der Tangens für 45 Grad errechnet.

TIME$
Variable und Anweisung

Zweck: Setzt die laufende Zeit und ruft sie ab.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
	***	***	***

Format: Als Variable:

*v$* = TIME$

Als Anweisung:

TIME$ = *x$*

Bemerkungen: **Für die Variable (v$ = TIME$):**

Die laufende Zeit wird als Zeichenkette aus acht Zeichen übergeben. 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 DOS gesetzt worden sein, bevor zu BASIC verzweigt wurde.

Für die Anweisung (TIME$ = x$):

TIME$
Variable und Anweisung

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

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

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

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

Eine führende Null kann für jeden dieser Werte weggelassen werden, aber es muß mindestens eine Ziffer eingesetzt werden. Soll z.B. die Zeit für 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" (Ungültiger Funktionsaufruf) angezeigt. Dabei wird die vorherige Zeitangabe beibehalten. Ist *x$* keine gültige Zeichenkette, wird der Fehler "Type Mismatch" (Keine Typübereinstimmung) angezeigt.

TIME$
Variable und Anweisung

Beispiel: Das folgende Programm zeigt fortlaufend die Zeit in der Mitte des Bildschirms an.

```
1Ø CLS
2Ø LOCATE 1Ø, 15
3Ø PRINT TIME$
4Ø GOTO 3Ø
```

TRON und TROFF Befehle

Zweck: Zeigt eine Ablaufverfolgung der Programmanweisungen bei der Ausführung des Programms an.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	(**)

Format: TRON

TROFF

Bemerkungen: Als Hilfe beim Programmtest aktiviert der Befehl TRON (der im indirekten Modus eingegeben werden kann) eine Programmablaufverfolgung, d.h. jede Zeilennummer des Programms wird angezeigt und die zugehörige Programmzeile ausgeführt. Die Nummer erscheint in eckigen Klammern. Durch den Befehl TROFF wird diese Ablaufverfolgung ausgeschaltet.

Beispiel:

```
Ok
10 K=10
20 FOR J=1 TO 2
30 L=K + 10
40 PRINT J;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 Ausführung 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 für J, K und L, die das Programm anzeigt.

USR Funktion

Zweck: Ruft das angegebene Unterprogramm in Maschinensprache mit dem Argument *arg* auf.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *v* = USR[*n*](*arg*)

Bemerkungen: *n* liegt im Bereich 0 bis 9 und korrespondiert mit der Ziffer, die mit der Anweisung DEF USR für das gewünschte Unterprogramm übergeben 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.

USR Funktion

Die Anweisung CALL ist ein weiterer Weg, Unterprogramme in Maschinensprache aufzurufen. Siehe Anhang C. "Unterprogramme in Maschinensprache". Dort wird die Benutzung von Unterprogrammen in Maschinensprache detailliert beschrieben.

Beispiel:

```
1Ø DEF USRØ = &HFØØØ
5Ø C = USRØ(B/2)
6Ø 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.

VAL
Funktion

Zweck: Übergibt den numerischen Wert der Zeichenkette *x$*.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: *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 können. Beispiel:

```
VAL("    -3")
```

Es wird -3 übergeben.

Sind die ersten Zeichen von *x$* nicht numerisch, übergibt VAL(*x$*) eine 0 (Null).

Siehe auch die "Funktion STR$" für die Umwandlung einer numerischen Variablen in eine Zeichenkette.

Beispiel:

```
Ok
PRINT VAL("34Ø8 SHERWOOD BLVD.")
 34Ø8
Ok
```

In diesem Beispiel wird mit Hilfe von VAL die Hausnummer aus einer Adresse herausgezogen.

VARPTR Funktion

Zweck: Übergibt die Hauptspeicheradresse einer Variablen oder eines Dateisteuerblocks.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	***

Format:

v = VARPTR(*Variable*)

v = VARPTR(*#Dateinummer)*

Bemerkungen: *Variable* Name einer numerischen oder Zeichenkettenvariablen oder eines Bereichselements im Programm. Bevor VARPTR aufgerufen wird, muß der *Variablen* ein Wert zugeordnet werden, sonst erhält man die Fehlernachricht "Illegal Function Call" (Ungültiger Funktionsaufruf).

Dateinummer
Nummer, unter der die Datei eröffnet wurde.

Für beide Formate ist die übergebene Adresse eine ganze Zahl im Bereich 0 bis 65535. Diese Nummer ist der Relativzeiger im BASIC-Datensegment. Die Adresse wird von der Anweisung DEF SEG nicht beeinflußt.

Für das erste Format wird die Adresse des ersten Bytes der Daten übergeben, die mit *Variable* angesprochen wird. Das Format dieser Daten wird in Anhang I. unter "Speicherung von Variablen" beschrieben.

> **Hinweis:** Allen einfachen Variablen sollte ein Wert zugeordnet sein, bevor VARPTR für einen Bereich aufgerufen wird, weil sich die Adressen von Bereichen jedesmal ändern, wenn eine neue einfache Variable eine Zuordnung erhält.

VARPTR wird normalerweise benutzt, um die Adresse einer Variablen oder eines Bereichs zu erhalten, so daß man diese Adresse an ein Unterprogramm in Maschinensprache mit Hilfe von USR übergeben kann. Ein Funktionsaufruf der Form VARPTR(A(0)) wird normalerweise angegeben, wenn ein Bereich übergeben werden soll, so daß das am niedrigsten adressierte Element des Bereichs übergeben wird.

VARPTR
Funktion

Durch das zweite Format wird die Startadresse eines Dateisteuerblocks für eine angegebene Datei übergeben. Dies ist nicht das gleiche wie der DOS-Dateisteuerblock. Siehe "BASIC-Dateisteuerblock" in Anhang I. "Technische Informationen und Tips", für detaillierte Informationen über das Format des Dateisteuerblocks.

VARPTR ist für Kassettendateien bedeutungslos.

Beispiel: In diesem Beispiel wird das erste Byte in den Puffer einer Datei für wahlfreien Zugriff gelesen:

```
10 OPEN "DATA.FIL" AS #1
20 GET #1
30 'holen Adresse des Steuerblocks
40 FCBADR = VARPTR(#1)
50 'errechnen Adresse des Datenpuffers
60 DATADR = FCBADR+188
70 'holen erstes Byte in den Datenpuffer
80 A% = PEEK(DATADR)
```

VARPTR Funktion

Im nächsten Beispiel werden mit Hilfe von VARPTR Daten aus einer Variablen gelesen. In Zeile 30 wird nach P die Adresse der Daten übertragen. Ganzzahlige Daten werden in zwei Bytes gespeichert, das weniger signifikante 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 höherwertigen Bits enthält.

```
10 DEFINT A-Z
20 DATA1=500
30 P=VARPTR(DATA1)
40 V=PEEK(P) + 256*PEEK(P+1)
50 PRINT V
```

VARPTR$ Funktion

Zweck: Übergibt die Zeichenform der Adresse einer Variablen im Hauptspeicher. Sie wird vorzugsweise mit PLAY und DRAW in Programmen benutzt, die später umgewandelt werden.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
	***	***	***

Format: *v$* = VARPTR$(*Variable*)

Bemerkungen: *Variable* Name einer Variablen, die sich im Programm befindet.

Hinweis: Zu allen einfachen Variablen sollte eine Zuordnung erfolgt sein, bevor VARPTR$ für ein Bereichselement aufgerufen wird, da sich die Adressen von Bereichen jedesmal ändern, wenn eine einfache Variable eine Zuordnung erhält.

VARPTR$ übergibt eine Drei-Byte-Zeichenkette der folgenden Form:

Byte 0	Byte 1	Byte 2
Typ	Niedrigwertiges Byte einer Variablenadresse	Höherwertiges Byte einer Variablenadresse

VARPTR$ Funktion

Typ: Zeigt den Variablentyp an:

2	Ganzzahlig
3	Zeichenkette
4	Einfache Genauigkeit
8	Doppelte Genauigkeit

Der übergebene Wert ist der gleiche wie:

CHR$(*Typ*)+MKI$(VARPTR(*Variable*))

Mit Hilfe von VARPTR$ kann man einen Variablennamen in der Befehlszeichenkette für PLAY oder DRAW angeben. Beispiel:

Methode I

```
PLAY "XA$;"
PLAY "O=I;"
```

Alternative Methode

```
PLAY "X"+VARPTR$(A$)
PLAY "O="+VARPTR$(I)
```

Diese Technik wird meistens in Programmen benutzt, die später umgewandelt werden.

WAIT
Anweisung

Zweck: Verläßt die Programmausführung zur Beobachtung des Status eines Maschineneingabeanschlusses.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: WAIT Anschluß, *n*[,*m*]

Bemerkungen: *Anschluß* Anschlußnummer im Bereich 0 bis 65535.

n, m sind ganzzahlige Ausdrücke im Bereich 0 bis 255.

Siehe *"IBM Personalcomputer Technisches Handbuch"*, das die Beschreibung der gültigen Anschlußnummern (Ein-/Ausgabeadressen) enthält.

Durch die Anweisung WAIT wird die Ausführung so lange ausgesetzt, bis der angegebene Maschineneingabeanschluß ein vorgegebenes Bit-Abbild entwickelt.

WAIT Anweisung

Die von dem Anschluß gelesenen Daten werden zuerst durch XOR mit dem ganzzahligen Ausdruck *m* und dann mit *n* durch AND verknüpft. Ist das Ergebnis 0 (Null), verzweigt BASIC zurück und liest die Daten des Anschlusses erneut. Ist das Ergebnis nicht 0, geht die Ausführung mit der nächsten Anweisung weiter. Ist *m* weggelassen, wird 0 angenommen.

Durch die Anweisung WAIT kann man eine oder mehrere Bit-Positionen eines Eingabeanschlusses testen. Man kann die Bit-Positionen entweder auf 0 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 Eingabeanschlusses auf 1 getestet. Ist *m* angegeben, so wird jede 1 in irgendeiner Bit-Position in *m* (für die ein 1-Bit in *n* existiert), durch WAIT auf 0 für dieses Eingabe-Bit geprüft.

WAIT Anweisung

Bei der Ausführung 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 0, falls das korrespondierende Bit in *m* 1 ist), geht das Programm mit der nächsten Anweisung weiter. Das bedeutet, daß WAIT nicht auf das gesamte Bit-Abbild, sondern nur auf ein Bit wartet.

> **Hinweis:** Es ist möglich, durch die Anweisung WAIT eine unendliche Schleife zu erhalten. Aus dieser Schleife kommt man durch Betätigen der Tasten Ctrl-Break oder ein Systemzurücksetzen heraus.

Beispiel: Die Programmausführung geht so lange nicht weiter, bis Anschluß 32 ein Einer-Bit in der zweiten Bit-Position enthält:

```
100 WAIT 32,2
```

WHILE und WEND Anweisung

Zweck: Führt eine Serie von Anweisungen in einer Schleife so lange aus, wie eine angegebene Bedingung wahr ist.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format:

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

WHILE und WEND
Anweisung

Bemerkungen: *Ausdruck* Numerischer Ausdruck.

Ist der *Ausdruck* wahr (nicht Null), werden die Schleifenanweisungen so lange ausgeführt, bis die Anweisung WEND kommt. BASIC kehrt jetzt zur Anweisung WHILE zurück und prüft *Ausdruck.* Ist seine Bedingung immer noch wahr, wird dieser Prozeß wiederholt. Ist die Bedingung nicht wahr, wird die Ausführung mit der Anweisung aufgenommen, die der Anweisung WEND folgt.

WHILE...WEND-Schleifen können in allen Ebenen verschachtelt werden. Jedes WEND gehört zum nächstliegenden WHILE. Zählt zu einer Anweisung WHILE kein WEND, erscheint der Fehler "WHILE without WEND" (WHILE ohne WEND). Zählt zu einer Anweisung WEND keine Anweisung WHILE, erscheint der Fehler "WEND without WHILE" (WEND ohne WHILE).

Beispeil: In diesem Beispiel werden die Elemente des Zeichenkettenbereichs A$ alphabetisch geordnet. A$ wurde mit J-Elementen definiert.

```
90 'Sortierbereich A$
100 FLIPS=1 'ein Lauf durch die Schleife wird forciert
110 WHILE FLIPS
115 FLIPS=0
120 FOR I=1 TO J-1
130  IF A$ (I)>A$ (I+1) THEN
       SWAP A$(I),A$(I+1): FLIPS=1
140 NEXT I
150 WEND
```

WIDTH
Anweisung

Zweck: Setzt die Ausgabezeilenbreite in Anzahl von Zeichen. Nachdem die angegebene Anzahl von Zeichen ausgegeben wurde, fügt BASIC ein Wagenrücklaufzeichen hinzu.

Versionen:

Kassette	Disk	Erweitert	Umwandlungsprogramm
***	***	***	(**)

Format:

WIDTH *Größe*

WIDTH *Dateinummer, Größe*

WIDTH *Einheit, Größe*

Bemerkungen: *Größe* Numerischer Ausdruck im Bereich 0 bis 255. Dies ist eine neue Breite. WIDTH 0 bedeutet das gleiche wie WIDTH 1.

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

Einheit Zeichenkettenausdruck für die Einheitenidentifikation. Gültige Einheiten sind SCRN:, LPT1:, LPT2:, COM1: oder COM2:.

Abhängig von der angegebenen Einheit sind die folgenden Aktionen möglich:

WIDTH *Größe* **oder WIDTH "SCRN:",***Größe*

Setzt die Bildschirmbreite. Nur Spaltenbreite 40 oder 80 ist erlaubt.

Befindet sich der Bildschirm im grafischen Modus für mittlere Auflösung (dies würde durch die Anweisung SCREEN 1 geschehen), wird durch WIDTH 80 der Bildschirm in die hohe Auflösung gebracht (wie durch die Anweisung SCREEN 2).

WIDTH
Anweisung

Befindet sich der Bildschirm im grafischen Modus für hohe Auflösung (wie durch die Anweisung SCREEN 2), wird durch WIDTH 40 der Bildschirm in die mittlere Auflösung gebracht (wie mit der Anweisung SCREEN 1).

Hinweis: Durch die Änderung der Bildschirmbreite wird der Bildschirm gelöscht und der Bildschirmrand auf Schwarz gesetzt.

WIDTH *Einheit, Größe*

Wird für eine verzögerte Breitenzuordnung für eine Einheit benutzt. Diese Form der Breite speichert den neuen Wert für Breite, ohne die aktuelle Breite zu verändern. Ein nachfolgendes OPEN für diese Einheit benutzt diesen Wert für die Breite, solange die Einheit eröffnet ist. Die Breite ändert sich nicht sofort, wenn die Einheit schon eröffnet ist.

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

WIDTH Anweisung

WIDTH *Dateinummer, Größe*

Die Breite der Einheit, der die *Dateinummer* zugeordnet ist, ändert sofort auf die neue angegebene Größe. Dies erlaubt, die Breite zu ändern, während eine Datei eröffnet ist. Diese Form von WIDTH hat nur für LPT1: im Kassetten-BASIC Bedeutung. Disk- und erweitertes BASIC erlauben auch LPT2:, LPT3:, COM1: und COM2:.

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

WIDTH hat auf die Tastatur (KYBD:) oder Kassette (CAS1:) keinen Einfluß.

Die Druckbreite für den Drucker ist standardmäßig 80, wenn mit BASIC begonnen wird. Die Maximalbreite für den IBM Matrixdrucker für 80 Zeichen pro Sekunde ist 132. Es wird jedoch kein Fehler für Werte zwischen 132 und 255 übergeben.

WIDTH
Anweisung

Es ist dem Programmierer überlassen, die physische Breite des Druckers zu setzen. Bei einigen Druckern geschieht dies durch Senden von Sonderzeichen, manche haben Schalter. Der IBM Matrixdrucker für 80 Zeichen pro Sekunde muß mit der Anweisung LPRINT CHR$(15); auf verdichtetes Schreiben gesetzt werden, wenn mit einer Breite größer als 80 gedruckt wird. Mit LPRINT CHR$(18); wird auf die normale Schriftbreite zurückgesetzt. Der IBM Matrixdrucker ist so gesetzt, daß er automatisch ein Wagenrücklaufzeichen hinzufügt, wenn die maximale Zeilenbreite überschritten ist.

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

Durch das Ändern der Breite für die Datenfernverarbeitungsdatei ändert sich weder der Empfangs- noch der Übertragungspuffer. BASIC sendet nur ein Wagenrücklaufzeichen nach jeder angegebenen Anzahl von Zeichen.

Das Ändern des Bildschirmmodus beeinflußt die Bildschirmbreite nur, wenn zwischen SCREEN 2 und SCREEN 1 oder SCREEN 0 gewechselt wird. Siehe "Anweisung SCREEN" in diesem Kapitel.

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 für den Drucker eröffnet und die Druckbreite für nachfolgende Anweisungen PRINT #1,...auf 75 gesetzt. In Zeile 6020 wird die gegenwärtige Druckbreite auf 40 Zeichen pro Zeile geändert.

```
SCREEN 1,0  'Setzen auf Farbgrafik, mittlere Auflösung
WIDTH 80    'Setzen auf Grafik, hohe Auflösung
WIDTH 40    'Zurück zu mittlerer Auflösung

SCREEN 0,1  'Nun 40x25 Text Farbmodus
WIDTH 80    'Nun 80x25 Text Farbmodus
```

WRITE
Anweisung

Zweck: Gibt Daten auf den Bildschirm aus.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: WRITE [*Liste der Ausdrücke*]

Bemerkungen: *Liste der Ausdrücke*
Liste aus numerischen und/oder Zeichenkettenausdrücken, die durch Kommata (,) oder Semikolons (;) getrennt sind.

Wird die Liste der Ausdrücke weggelassen, ist die Ausgabe eine Leerzeile. Wird die Liste der Ausdrücke eingefügt, ist der Wert der Ausdrücke die Ausgabe auf dem Bildschirm.

Werden die Werte der Ausdrücke ausgegeben, wird jede Angabe von der letzten durch ein Komma getrennt ausgegeben. Zeichenketten werden in Anführungszeichen (") eingeschlossen. Ist die letzte Angabe der Liste ausgegeben, fügt BASIC einen Wagenrücklauf/Zeilenvorschub an.

WRITE Anweisung

WRITE ist ähnlich wie PRINT. Der Unterschied zwischen WRITE und PRINT besteht darin, daß WRITE Kommata zwischen die Angaben einfügt, wie sie angezeigt werden, und Zeichenketten in Anführungszeichen 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
```

WRITE#
Anweisung

Zweck: Schreibt Daten in eine sequentielle Datei.

Versionen:

Kassette	Disk	Erweitert	Umwandlungs-programm
***	***	***	***

Format: WRITE *#Dateinummer, Liste der Ausdrücke*

Bemerkungen: *Dateinummer*
Nummer, unter der die Datei für Ausgabe eröffnet wurde.

Liste der Ausdrücke
Liste von Zeichenketten und/oder numerischen Ausdrücken, die durch Kommata (,) oder Semikolons (;) getrennt sind.

Der Unterschied zwischen WRITE # und PRINT # ist der, daß WRITE # Kommata zwischen die Angaben einfügt, wie sie geschrieben werden, und Zeichenketten in Anführungszeichen (") einschließt. Deshalb braucht der Benutzer keine Trennzeichen in die Liste einzufügen. WRITE # setzt auch keine Leerstellen vor eine positive Zahl. Ein Wagenrücklauf/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$,B$
```

würde "CAMERA" der Variablen A$ und "93604-1" der Variablen B$ zuordnen.

Notizen:

ANHÄNGE

Inhalt

Anhang A. Nachrichten

Falls BASIC einen Fehler entdeckt, der das Programm stoppt, wird eine Fehlernachricht angezeigt. Es ist möglich, mit der Anweisung ON ERROR und den Variablen ERR und ERL Fehler in einem BASIC-Programm zu lesen und zu testen. (Eine vollständige Erklärung für ON ERROR, ERR und ERL enthält Kapitel 4. "BASIC-Befehle, -Anweisungen, -Funktionen und -Variablen.")

In diesem Anhang sind alle BASIC-Fehlernachrichten mit ihren zugehörigen Fehlernummern aufgeführt.

Nummer *Nachricht*

1 **NEXT without FOR (NEXT ohne FOR)**
Der Anweisung NEXT fehlt die zugehörige Anweisung FOR. Es kann sein, daß eine Variable in der Anweisung NEXT nicht mit einer zuvor ausgeführten und ungleichen Anweisung FOR-Variablen übereinstimmt.

Das Programm ändern, so daß NEXT eine zugehörige Anweisung FOR besitzt.

2 **Syntax error (Syntaxfehler)**
Eine Zeile enthält eine nicht gültige Folge von Zeichen, wie z.B. ungleiche Anzahl von Klammern, einen falsch geschriebenen Befehl, eine falsch geschriebene Anweisung, eine ungültige Interpunktion, oder die Daten einer Anweisung DATA sind nicht vom gleichen Typ (numerisch oder Zeichenkette) wie die Variable in einer Anweisung READ.

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 benötigt 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 "fällt".

4 **Out of data (Zuwenig Daten)**
Eine Anweisung READ versucht, mehr Daten zu lesen als in den Anweisungen DATA angegeben sind.

Das Programm korrigieren, so daß genug Konstanten in den Anweisungen DATA für alle Anweisungen READ des Programms zu Verfügung stehen.

5 **Illegal function call**
(Ungültiger Funktionsaufruf)
Ein sich nicht im gültigen Bereich befindlicher Parameter wurde an eine Systemfunktion übergeben. 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 für GET oder PUT (Datei).
- Ein ungültiges Argument für eine Funktion oder Anweisung.
- Versuch, ein geschütztes BASIC-Programm aufzulisten oder zu verändern.
- Versuch, Zeilennummern zu löschen, die nicht existieren.

Korrigieren des Programms. Siehe unter Kapital 4. "BASIC-Befehle, -Anweisungen, Funktionen und Variablen" für Informationen über eine bestimmte Anweisung oder Funktion.

6 **Overflow (Überlauf)**
Der Bereich einer Zahl ist zu groß, um im BASIC-Zahlenformat dargestellt werden zu können. Überlauf für ganze Zahlen stoppt die Ausführung, anderenfalls wird die größte in der Maschine darstellbare Zahl mit dem richtigen Vorzeichen als Ergebnis übergeben und die Ausführung fortgesetzt.

Um einen Überlauf für ganze Zahlen zu korrigieren, müssen kleinere Zahlen verwendet werden oder zu Variablen mit einfacher oder doppelter Genauigkeit übergegangen werden.

Hinweis: Ist eine Zahl zu klein, um im Zahlenformat von BASIC dargestellt zu werden, bekommt man die *Unterlaufbedingung*. Wenn dies auftritt, ist das Ergebnis Null, und die Ausführung geht ohne einen Fehler weiter.

7 **Out of memory**
(Hauptspeicher reicht nicht aus)
Ein Programm ist zu groß, enthält zu viele FOR-Schleifen oder GOSUBs, zu viele Variablen, Ausdrücke, die zu kompliziert sind, oder komplexes Malen.

Mit CLEAR am Anfang des Programms kann man sich mehr Stapel- oder Hauptspeicherbereich schaffen.

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 prüfen und die richtige Zeilennummer benutzen.

9 **Subscript out of range**
(Index außerhalb des Bereichs)
Ein Bereichselement wurde mit einem Index außerhalb der Dimensionen des Bereichs oder mit einer falschen Anzahl von Indizes angesprochen.

Benutzung der Bereichsvariablen überprüfen. Vielleicht wurde an eine Variable ein Index angefügt, die keine Bereichsvariable ist, oder eine eingebaute Funktion wurde falsch codiert.

10 Duplicate Definition
(Doppelte Definition)
Es wurde versucht, die Größe des gleichen Bereichs zweimal zu definieren. Dies kann auf verschiedene Weise passieren:

- Der gleiche Bereich wurde in zwei Anweisungen DIM definiert.
- Das Programm führt eine Anweisung DIM für einen Bereich aus, nachdem die Standarddimension von 10 für diesen Bereich gesetzt wurde.
- Das Programm erkennt eine Anweisung OPTION BASE, nachdem ein Bereich entweder durch eine Anweisung DIM oder durch die Standardannahme dimensioniert wurde.

Die Anweisung OPTION BASE muß an eine Stelle im Programm gesetzt werden, so daß die ausgeführt wird, bevor irgendein Bereich benutzt wird; oder das Programm muß korrigiert werden, so daß jeder Bereich 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 nötig, diese Bedingung zu korrigieren, da das Programm in der Ausführung fortfährt. Das Ergebnis der Division ist die größtmögliche darstellbare Zahl mit dem Vorzeichen der Zahl, die dividiert werden sollte (Dividend); oder eine positive größtmögliche darstellbare Zahl als Ergebnis der Potenzierung.

12 **Illegal direct**
(Ungültiger Direktmodus)
Es wurde versucht, eine Anweisung im Direktmodus einzugeben, für die der Direktmodus ungültig ist (wie z.B. DEF FN).

Die Anweisung muß als Teil einer Programmzeile eingegeben werden.

13 **Type mismatch**
(Keine Typübereinstimmung)
Die Zuordung war ein Zeichenkettenwert, ein numerischer Wert wurde aber erwartet; oder es wurde ein numerischer Wert anstelle eines Zeichenkettenwertes angegeben. 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 für Zeichenketten zu Ende)
BASIC ordnet den Platz für Zeichenketten dynamisch zu, bis kein Hauptspeicherplatz mehr zur Verfügung steht. Die Nachricht bedeutet, daß durch Zeichenkettenvariablen BASIC mehr als die verbleibende Anzahl von freiem Hauptspeicher benötigt, nachdem alles gelöscht war, was gelöscht werden konnte.

15 **String too long (Zeichenkette zu lang)**
Es wurde versucht, eine Zeichenkette zu erstellen, die länger als 255 Zeichen ist.

Versuchen, die Zeichenkette in kleinere Zeichenketten aufzuteilen.

16 **String formula too complex (Formel für Zeichenkette zu komplex)**
Ein Zeichenkettenausdruck ist zu lang oder zu komplex.

Der Ausdruck sollte in kleinere Ausdrücke zerlegt werden.

17 **Can't continue (Programmausführung kann nicht fortgesetzt werden)**
Es wurde versucht, mit CONT die Ausführung eines Programms fortzuführen, das:

- durch einen Fehler gestoppt wurde;
- während einer Ausführungsunterbrechung verändert wurde;
- nicht existiert.

Das Programm muß geladen und mit RUN ausgeführt 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 ausführen, bevor die Funktion benutzt werden kann.

19 **No RESUME (Keine Wiederaufnahme des Programms)**
Das Progamm verzweigte 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 eingefügt werden, um mit der Programmausführung fortfahren zu können. Man kann in die Fehlerbehandlungsroutine auch die Anweisung ON ERROR GOTO 0 einfügen, so daß BASIC für 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 ausgeführt wird.

Wahrscheinlich muß eine Anweisung STOP oder END vor der Fehlerbehandlungsroutine eingefügt werden, um zu verhindern, daß das Programm in die Fehlerbehandlungsroutine "fällt".

22 **Missing operand (Fehlender Operand)**
Ein Ausdruck enthält einen Operator, wie z.B. * (Stern) oder OR, dem kein Operand folgt.

Alle benötigten Operanden müssen in den Ausdruck eingefügt werden.

23 **Line buffer overflow (Zeilenpufferüberlauf)**
Es wurde versucht, eine Zeile einzugeben, die aus zu vielen Zeichen besteht.

Einzeilige Mehrfachanweisungen in mehrere Zeilen aufteilen. Es können auch Zeichenkettenvariablen anstelle von Konstanten benutzt werden, wo dies möglich ist.

24 **Device Timeout (Einheitenzeitsperre)**
BASIC erhielt innerhalb einer vorgegebenen Zeitspanne keine Information von einer Ein-/Ausgabeeinheit. Im Kassetten-BASIC tritt dies nur auf, während ein Programm versucht, von einer Kassette zu lesen oder auf dem Drucker zu drucken.

Für Datenfernverarbeitungsdateien bedeutet diese Nachricht, daß ein oder mehrere Signale, die mit OPEN "COM... gesetzt wurden, in einer angegebenen Zeitperiode nicht gefunden wurden.

Operation wiederholen.

25 **Device Fault (Einheitenfehler)**
Ein Hardware-Fehler wurde von einem Anschluß angezeigt.

Im Kassetten-BASIC passiert dies nur, wenn ein falscher Status vom Druckeranschluß übergeben wurde.

Diese Nachricht kann auch auftreten, wenn Daten in eine Datenfernverarbeitungsdatei übertragen 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 Zeitperiode gefunden wurden.

26 **FOR without NEXT (FOR ohne NEXT)**
FOR wurde ohne ein zugehöriges NEXT gefunden. Das heißt, eine FOR-Schleife war aktiv, als das physische Ende des Programms erreicht wurde.

Das Programm durch Einfügen einer Anweisung NEXT verbessern.

27 **Out of Paper (Kein Druckerpapier)**
Im Drucker ist kein Papier oder er ist nicht eingeschaltet.

Papier einlegen (falls nötig), die Verbindung des Druckers überprüfen oder den Drucker einschalten. Danach mit dem Programm fortfahren.

29 **WHILE without WEND**
(WHILE ohne WEND)
Einer Anweisung WHILE fehlt die zugehörige Anweisung WEND. Das heißt, WHILE war noch aktiv, als das physische Ende des Programms erreicht wurde.

Das Programm durch Einfügen einer Anweisung WEND für jedes WHILE korrigieren.

30 **WEND without WHILE**
(WEND ohne WHILE)
Eine Anweisung WEND wurde gefunden, bevor die zugehörige Anweisung WHILE ausgeführt wurde.

Das Programm so verbessern, daß für jede Anweisung WEND eine Anweisung WHILE existiert.

50 **FIELD overflow (FIELD-Überlauf)**
Eine Anweisung FIELD versucht, mehr Bytes zuzuordnen als für die Satzlänge in einer Datei für wahlfreien Zugriff in der Anweisung OPEN angegeben waren, oder es wurde das Ende des FIELD-Puffers gefunden, während eine sequentielle Ein-/Ausgabe (PRINT #, WRITE #, INPUT #) zu einer Datei mit wahlfreiem Zugriff erfolgte.

Die Anweisungen OPEN und FIELD überprüfen und beide auf Übereinstimmung bringen. Bei sequentieller Ein-/Ausgabe zu einer Datei mit wahlfreiem Zugriff darf die Länge der gelesenen oder geschriebenen Daten nicht die Satzlänge der Datei für wahlfreien Zugriff überschreiten.

51 **Internal error (Interner Fehler)**
In BASIC trat ein interner Fehler auf.

Diskette neu kopieren. Die Hardware prüfen und die Operation wiederholen. Im erneuten Fehlerfalle die Verkaufsstelle über die Fehlerbedingungen informieren.

52 **Bad file number (Falsche Dateinummer)**
Eine Anweisung benutzt eine Dateinummer einer Datei, die nicht eröffnet ist, oder die Dateinummer befindet sich nicht in dem möglichen Dateinummernbereich, der bei der Initialisierung angegeben wurde; oder der Einheitenname in der Dateispezifikation ist zu lang oder ungültig oder der Dateiname war zu lang oder ungültig.

Sicherstellen, daß die gewünschte Datei eröffnet war und die Dateinummer richtig in die Anweisung eingefügt wurde. Auch auf gültige Dateispezifikation überprüfen (siehe unter "Namensgebung für Dateien" in Kapitel 3).

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.

Überprüfen, ob sich die richtig 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, Hinzufügen oder wahlfreier Zugriff auszuführen.

Sicherstellen, daß die Anweisung OPEN richtig eingegeben und ausgeführt wurde. GET und PUT benötigen eine Datei für 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 nötig, das Programm laden und mit der Auswahl A speichern.

55 **File already open (Datei schon eröffnet)**
Es wurde versucht, eine Datei für sequentielle Ausgabe oder für sequentielles Hinzufügen zu eröffnen, und die Datei war schon eröffnet; oder es wurde versucht, KILL für eine schon eröffnete Datei zu benutzen.

Sicherstellen, daß nur ein OPEN für eine Datei ausgeführt wird, wenn sie sequentiell beschrieben werden soll. Die Datei abschließen, bevor KILL benutzt wird.

57 **Device I/O Error**
(Einheitenein-/-ausgabefehler)
Ein Fehler passierte bei einer Einheitenein-/-ausgabeoperation. DOS kann diesen Fehler nicht beheben.

Beim Empfang von Datenfernverarbeitungsdaten kann dieser Fehler durch Überlauf, Datenfernverarbeitungsrahmen, Unterbrechung oder Paritätsfehler auftreten. Werden Daten mit sieben oder weniger Daten-Bits empfangen, wird das achte Bit in das fehlerhafte Byte eingeschaltet.

58 **File already exists**
(Die Datei existiert schon)
Ein 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 vollständig benutzt. Dateien werden abgeschlossen, wenn dieser Fehler auftritt.

Befinden sich Dateien auf der Diskette, die nicht mehr benötigt werden, sollten sie gelöscht 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 für eine leere Datei ausgeführt 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 abzufangen.

Der gleiche Fehler wird angezeigt, falls versucht wird, von einer Datei zu lesen, die für Ausgabe oder Hinzufügen eröffnet wurde. Soll von einer sequentiellen Ausgabe- (oder Hinzufügen-) datei gelesen werden, muß sie abgeschlossen und wieder für Eingabe eröffnet werden.

63 **Bad record number**
(Falsche Satznummer)
In einer Anweisung PUT oder GET ist die Satznummer entweder größer als die maximal erlaubte (32767) oder gleich Null.

Die Anweisung PUT oder GET korrigieren, damit eine gültige Satznummer angesprochen wird.

64 **Bad file name (Falscher Dateiname)**
Eine ungültige Form wurde für den Dateinamen mit BLOAD, BSAVE, KILL, NAME, OPEN oder FILES benutzt.

In Kapitel 3 unter "Namensgebung für Dateien" nachsehen, wie gültige Dateinamen aussehen müssen, und den fehlerhaften Dateinamen korrigieren.

66 **Direct Statement in file (Anweisung für direkten Modus in der Datei)**
Eine Anweisung für direkten Modus wurde gefunden, während 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 für Zeilenvorschub im Eingabestrom befindet. Siehe Anhang D. "Programmumwandlung in BASIC des IBM Personalcomputers".

67 **Too many files (Zu viele Dateien)**
Es wurde versucht, eine neue Datei zu erstellen (mit Hilfe von SAVE oder OPEN), als alle Bibliothekseintragungen der Diskette voll waren oder die Dateispezifikation ungültig war.

Ist die Dateispezifikation in Ordnung, muß eine neue formatierte Diskette verwendet und die Operation wiederholt werden.

68 **Device Unavailable**
(Einheit nicht verfügbar)
Es wurde versucht, eine Datei in einer Einheit zu eröffnen, die nicht vorhanden ist. Entweder fehlt die Hardware, um die Einheit zu unterstützen (wie z.B. Druckeranschlüsse für den zweiten oder dritten Drucker) oder die Einheit wurde inaktiviert. (Es wurde z.B. **/C:0** mit dem Befehl BASIC benutzt, um Disketten BASIC zu starten. Dadurch würden die Datenfernverarbeitungseinheiten inaktiviert.)

Die Einheit muß richtig installiert sein. Falls nötig, den folgenden Befehl eingeben:

```
SYSTEM
```

Damit wird zurück zu DOS verzweigt, wo der BASIC-Befehl erneut eingegeben werden kann.

69 **Communication buffer overflow**
(Überlauf des
Datenfernverarbeitungspuffers)
Eine Eingabeanweisung für Datenfernverarbeitung wurde ausgeführt, 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 löschen, außer die Zeichen werden weiterhin schneller empfangen, als das Programm sie verarbeiten kann. Es gibt verschiedene Lösungen:

- Die Größe 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 (siehe Beispiel in Anhang F. "Datenfernverarbeitung").

- Eine niedrigere Übertragungsrate für Senden und Empfangen benutzen.

70 **Disk Write Protect (Diskettenschreibschutz)**
Es wurde versucht, eine Diskette mit Diskettenschreibschutz zu beschreiben.

Überprüfen, 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 möglich, sollte die Diskette nicht mehr benutzt werden.

73 **Advanced Feature**
(Einrichtung für erweitertes BASIC)
Im Programm wurde eine Einrichtung des erweiterten BASIC benutzt, während mit Disketten-BASIC gearbeitet wird.

Das erweiterte BASIC starten und das Programm neu laufen lassen.

— **Unprintable error**
(Nichtdruckbarer Fehler)
Für die existierende Fehlerbedingung ist keine Fehlernachricht verfügbar. Dies passiert gewöhnlich durch eine Anweisung ERROR mit einem nicht definierten Fehlercode.

Im Programm prüfen, ob alle erstellten Fehlercodes auch bearbeitet werden.

Notizen:

Anhang B. BASIC-Diskettenein- und -ausgabe

In diesem Anhang werden Prozeduren beschrieben und besondere Betrachtungen angestellt für die Benutzung der Diskettenein- und -ausgabe. Er enthält eine Liste der Befehle und Anweisungen, die mit Diskettendateien benutzt werden, und Erklärungen, wie sie benutzt werden sollen. Verschiedene Programmbeispiele sind eingefügt, um die Benutzung von Datendateien auf Diskette zu klären. Falls es sich um einen Anfänger in BASIC handelt oder sehr viele Fehler auftreten, die sich auf Diskette beziehen, sollten diese Prozeduren und Programmbeispiele gelesen werden, um sicherzustellen, daß die Diskettenanweisungen richtig angewandt werden.

Weitere Informationen über die Behandlung von Disketten und Diskettendateien siehe *IBM Personalcomputer Betriebssystem DOS (Disk Operating System) Handbuch.*

> **Hinweis:** Die meisten Informationen dieses Anhangs über Programmdateien und sequentielle Dateien beziehen sich auch auf die Kassettenein-/-ausgabe. Eine Kassette kann jedoch nicht für wahlfreien Zugriff eröffnet werden.

Spezifikation von Dateinamen

Dateinamen für Diskettendateien müssen mit den Konventionen über die Namensgebung in DOS übereinstimmen, damit sie mit Hilfe von BASIC gelesen werden können. Siehe "Namensgebung für Dateien" in Kapitel 3. Dort ist beschrieben, wie die Diskettendateien richtig spezifiziert werden.

Befehle für Programmdateien

Die Befehle, die für die BASIC-Programmdateien benutzt werden können, sind unten aufgelistet, zusätzlich eine Kurzbeschreibung. Detaillierte Informationen über jeden dieser Befehle stehen in Kapitel 4. "BASIC Befehle, -Anweisungen, -Funktionen und -Variablen.

SAVE Dateiangabe [,A]

Das sich gerade im Hauptspeicher befindliche Programm wird auf Diskette gespeichert. Durch die Auswahl A wird das Programm als Serie von ASCII-Zeichen geschrieben. (Sonst benutzt BASIC ein komprimiertes Binärformat.)

LOAD Dateiangabe [,R]

Lädt ein Programm von der Diskette in den Hauptspeicher. Durch die Auswahl **R** wird das Programm sofort ausgeführt. Durch LOAD wird immer der Inhalt des Hauptspeichers gelöscht und alle Dateien geschlossen, bevor der Ladevorgang beginnt. Ist jedoch **R** angegeben, bleiben eröffnete Datendateien offen. Dadurch können Programme verkettet oder in Abschnitten geladen werden und haben auf die gleichen Datendateien Zugriff.

RUN Dateiangabe [,R]

Ein Programm wird von der Diskette in den Hauptspeicher geladen und ausgeführt. RUN löscht den Inhalt des Hauptspeichers und schließt alle Dateien, bevor das Programm geladen wird. Ist jedoch die Auswahl **R** eingefügt, bleiben alle eröffneten Datendateien offen.

MERGE Dateiangabe

Lädt ein Programm von Diskette in den Hauptspeicher. Dabei wird der vorherige Inhalt des Hauptspeichers nicht gelöscht. Die Programmzeilennummern auf der Diskette werden mit den Zeilennummern im Hauptspeicher vermischt. Haben zwei Zeilen die gleiche Nummer, überschreibt die Zeile von der Diskette die Zeile im Hauptspeicher. Nach dem Befehl MERGE bleibt das vermischte Programm im Hauptspeicher und BASIC kehrt zur Befehlsebene zurück.

KILL Dateiangabe
Löscht eine Datei von der Diskette.

NAME Dateiangabe AS Dateiname
Verändert den Namen einer Diskettendatei.

Geschützte Dateien

Soll ein Programm geschützt gespeichert werden, muß es unter Zuhilfenahme der Auswahl P (protect = schützen) mit dem Befehl SAVE gespeichert werden. Beispiel:

```
SAVE "MYPROG",P
```

Ein Programm, das auf diese Weise gespeichert wurde, kann nicht aufgelistet, gespeichert oder verändert werden. Da dieser Programmschutz nicht mehr aufgehoben werden kann, sollte man auch ein Programm ohne Schreibschutz für Auflisten und Verändern speichern.

Diskettendatendateien - Sequentielle und wahlfreie Ein-/Ausgabe

Zwei Arten von Diskettendatendateien können mit einem BASIC-Programm erstellt und verarbeitet werden: sequentielle Dateien und Dateien für wahlfreien Zugriff.

Sequentielle Dateien

Sequentielle Dateien sind leichter zu erstellen als Dateien für wahlfreien Zugriff, sind aber in ihrer Flexibilität und Geschwindigkeit begrenzt, wenn die Daten verarbeitet werden sollen. Daten, die in eine sequentielle Datei geschrieben werden, werden sequentiell gespeichert - eine Angabe nach der anderen, in der Reihenfolge, in der jede Angabe übergeben wird. Alle Angaben werden auf die gleiche Weise zurückgelesen - von der ersten Angabe in der Datei bis zur letzten.

Die folgenden Anweisungen und Funktionen werden in Verbindung mit sequentiellen Dateien benutzt:

CLOSE	WRITE #
INPUT #	EOF
LINE INPUT #	INPUT$
OPEN	LOC
PRINT #	LOF
PRINT # USING	

Erstellen und Zugriff auf eine sequentielle Datei

Um eine sequentielle Datei zu erstellen und auf die Daten in der Datei zuzugreifen, müssen folgende Schritte in das Programm eingefügt werden:

1. Die Datei muß mit Hilfe der Anweisung OPEN für Ausgabe oder Hinzufügen eröffnet werden.

2. Die Daten müssen in die Datei mit Hilfe der Anweisungen PRINT #, WRITE # oder PRINT # USING geschrieben werden.

3. Soll auf die Daten in der Datei zugegriffen werden, muß die Datei geschlossen werden (mit Hilfe von CLOSE) und erneut für Eingabe (mit Hilfe von OPEN) eröffnet werden.

4. Mit den Anweisungen INPUT # oder LINE INPUT # können Daten von einer sequentiellen Datei in das Programm gelesen werden.

Im folgenden Beispiel werden diese Schritte angezeigt:

```
100 OPEN "DATA" FOR OUTPUT AS #1 'Schritt 1
200 WRITE #1,A$,B$,C$             'Schritt 2
300 CLOSE #1                      'Schritt 3
400 OPEN "DATA" FOR INPUT AS #1 'auch Schritt 3
500 INPUT #1,X$,Y$,Z$              'Schritt 4
```

Das obige Programm hätte auch wie folgt geschrieben werden können:

```
100 OPEN "O",#1,"DATA"    'Schritt 1
200 WRITE #1,A$,B$,C$     'Schritt 2
300 CLOSE #1              'Schritt 3
400 OPEN "I",#1,"DATA"    'noch Schritt 3
500 INPUT #1,X$,Y$,Z$      'Schritt 4
```

Man beachte, daß beide Anweisungen OPEN zum gleichen Ergebnis kommen. Unter Anweisung OPEN in Kapitel 4 befindet sich die Syntax für jede Form des OPEN.

Das folgende kurze Programm PROGRAM1 erzeugt die sequentielle Datei "DATA" aus Eingaben über die Tastatur.

Programm 1

```
1 REM PROGRAM1 - erzeugt eine sequentielle Datei
1Ø OPEN "DATA" FOR OUTPUT AS #1
2Ø INPUT "NAME";N$
25 IF N$="FERTIG" THEN CLOSE: END
3Ø INPUT "ABTEILUNG";D$
4Ø INPUT "EINSTELLUNGSDATUM";H$
5Ø WRITE #1,N$,D$,H$
6Ø PRINT: GOTO 2Ø
RUN
NAME? MICHELANGELO
ABTEILUNG? VISUELLE HILFEN
EINSTELLUNGSDATUM? Ø1/12/72

NAME? SHERLOCK HOLMES
ABTEILUNG? FORSCHUNG
EINSTELLUNGSDATUM? 12/Ø3/65

NAME? EBENEEZER SCROOGE
ABTEILUNG? RECHNUNGSWESEN
EINSTELLUNGSDATUM? Ø4/27/78

NAME? SUPER MANN
ABTEILUNG? WARTUNG
EINSTELLUNGSDATUM? Ø8/16/78

NAME? FERTIG
Ok
```

PROGRAM2 verarbeitet die Datei "DATA", die mit PROGRAM1 erzeugt wurde, und zeigt die Namen der Personen am Bildschirm an, die 1978 eingestellt wurden:

Programm 2

```
1 REM PROGRAM2 - Verarbeiten einer sequentiellen Datei
1Ø OPEN "DATA" FOR INPUT AS 1
2Ø INPUT #1,N$,D$,H$
3Ø IF RIGHT$(H$,2)="78" THEN PRINT N$
4Ø GOTO 2Ø
RUN
EBENEEZER SCROOGE
SUPER MANN
Input past end in 2Ø
Ok
```

PROGRAM2 liest sequentiell jede Angabe in der Datei. Sind alle Daten gelesen, erzeugt die Zeile 20 den Fehler "Input past end" (Eingabe nach logischem Ende). Um dies zu vermeiden, muß eine Zeile 15 eingefügt werden, die die Funktion EOF benutzt, um das Dateiende zu testen:

```
15 IF EOF(1) THEN CLOSE: END
```

Die Zeile 40 muß auf GOTO 15 geändert werden. Das Dateiende wird durch ein Sonderzeichen in der Datei angezeigt Dieses Zeichen entspricht dem ASCII-Code 26 (Hex 1A). Deshalb darf man nicht CHR$(26) in einer sequentiellen Datei speichern.

Ein Programm, das eine sequentielle Datei erstellt, kann auch formatierte Daten mit Hilfe der Anweisung PRINT # USING auf die Diskette schreiben. Die folgende Anweisung

```
PRINT 1,USING "####.##,";A,B,C,D
```

könnte benutzt werden, numerische Daten ohne explizit angegebene Trennzeichen auf die Diskette zu schreiben. Das Komma am Ende der Formatzeichenkette dient dazu, die Angaben in der Diskettendatei zu trennen.

Die Funktion LOC übergibt, wenn sie beim Verarbeiten einer sequentiellen Datei benutzt wird, die Anzahl der Sätze, die seit der Eröffnung der Datei geschrieben oder gelesen wurden. (Ein Satz besteht aus einem 128 Bytes langen Datenblock.) Die Funktion LOF übergibt die Anzahl der Bytes, die der Datei zugeordnet wurden. Diese Nummer ist immer ein Mehrfaches von 128 (falls nötig aufgerundet).

Hinzufügen von Daten zu einer sequentiellen Datei

Steht eine sequentielle Datei auf Diskette und man möchte Daten an das Ende der Datei hinzufügen, darf man nicht einfach die Datei für Ausgabe eröffnen und beginnen, Daten zu schreiben. Sobald man eine sequentielle Datei für Ausgabe eröffnet, wird ihr bisheriger Inhalt zerstört. Die Datei muß mit APPEND eröffnet werden. Siehe "Anweisung OPEN" in Kapitel 4.

Dateien für wahlfreien Zugriff

Für die Erstellung und Verarbeitung wahlfreier Dateien sind mehr Programmschritte erforderlich als für sequentielle Dateien. Jedoch bringt die Benutzung von wahlfreien Dateien Vorteile. Z.B. werden die Zahlen in wahlfreien Dateien gewöhnlich auf Diskette im Binärformat gespeichert, während Zahlen in sequentiellen Dateien als ASCII-Zeichen gespeichert werden. Deshalb benötigen wahlfreie Dateien normalerweise weniger Platz auf der Diskette als sequentielle Dateien.

Der größte Vorteil wahlfreier Dateien ist, daß auf Daten wahlfrei zugegriffen werden kann. Das bedeutet, die Daten können irgendwo auf der Diskette stehen - es ist nicht nötig, sequentiell alle Informationen zu lesen, bis man die gewünschte Information bekommt, wie das bei sequentiellen Dateien nötig ist. Dies ist deshalb möglich, weil die Informationen in definierten Einheiten, genannt Sätze, gespeichert und verarbeitet werden und jedem Satz eine Nummer zugeordnet ist.

Sätze können jede Länge bis zu 32767 Bytes haben. Die Größe des Satzes ist nicht von der Größe eines Sektors auf der Diskette (512 Bytes) abhängig. BASIC benutzt automatisch alle 512 Bytes eines Sektors für die Speicherung der Information. Dies geschieht durch Satzblockung und Übergreifen über Sektorgrenzen (das bedeutet, daß ein Teil eines Satzes am Ende eines Sektors stehen kann und der andere sich am Beginn des nächsten Sektors befindet).

Die folgenden Anweisungen und Funktionen werden für wahlfreie Dateien benutzt:

CLOSE	CVI
FIELD	CVS
GET	LOC
LSET/RSET	LOF
OPEN	MKD$
PUT	MKI$
CVD	MKS$

Erstellen einer Datei für wahlfreien Zugriff

Die folgenden Programmschritte werden benötigt, um eine wahlfreie Datei zu erstellen:

1. Die Datei muß für wahlfreien Zugriff eröffnet werden. Das folgende Beispiel zeigt diese Schritte, wobei eine Satzlänge von 32 Bytes gewählt wird. Wird die Satzlänge weggelassen, wird als Standardlänge 128 Bytes gewählt.

2. Mit der Anweisung FIELD wird dem Puffer für wahlfreien Zugriff Platz für die Variablen zugeordnet, die in die wahlfreie Datei geschrieben werden.

3. Mit LSET oder RSET werden die Daten in den Puffer für wahlfreien Zugriff gebracht. Numerische Werte müssen in Zeichenketten umgewandelt werden, bevor sie in den Puffer gebracht werden. Dies geschieht mit den folgenden Funktionen: MKI$ verwandelt einen ganzzahligen Wert in eine Zeichenkette, MKS$ tut dies für Werte einfacher Genauigkeit und MKD$ für Werte doppelter Genauigkeit.

4. Die Daten aus dem Puffer werden mit Hilfe der Anweisung PUT auf die Diskette geschrieben.

Die folgenden Zeilen illustrieren diese Schritte:

```
100 OPEN "FILE" AS #1 LEN=32 'Schritt 1
200 FIELD 1,20 AS N$, 4 AS A$, 8 AS P$ 'Schritt 2
300 LSET N$=X$                          'Schritt 3
400 LSET A$=MKS$(AMT)                   'noch Schritt 3
500 LSET P$=TEL$                        'noch Schritt 3
600 PUT #1,CODE%                        'Schritt 4
```

Hinweis: Es darf keine Zeichenkettenvariable benutzt werden, die in einer Anweisung FIELD, in einer Eingabeanweisung oder auf der linken Seite einer Zuordnungsanweisung (LET) definiert wurde. Dadurch würde der Zeiger für diese Variable auf den Platz für Zeichenketten statt auf den Dateipuffer für wahlfreien Zugriff zeigen.

PROGRAM3 übernimmt die Information, die über die Tastatur eingegeben wird, und schreibt sie in eine wahlfreie Datei. Bei jeder Ausführung der Anweisung PUT wird ein Satz in die Datei geschrieben. Der zweistellige Code, der in Zeile 30 eingegeben wird, ist die Satznummer.

Programm 3

```
1 REM PROGRAM3 - Erstellen einer wahlfreien Datei
10 OPEN "FILE" AS #1 LEN=32
20 FIELD #1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "zweistelliger Code";CODE%
35 IF CODE%=99 THEN CLOSE: END
40 INPUT "NAME";X$
50 INPUT "SUMME";AMT
60 INPUT "TEL.-NR.";TEL$: PRINT
70 LSET N$=X$
80 LSET A$=MKS$(AMT)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
```

Zugriff zu einer Datei für wahlfreien Zugriff

Die folgenden Programmschritte werden benötigt, um eine wahlfreie Datei zu verarbeiten:

1. Die Datei für wahlfreien Zugriff eröffnen.

2. Mit Hilfe der Anweisung FIELD dem Puffer für wahlfreien Zugriff Platz für die Variablen zuordnen, die von der Datei gelesen werden.

   Hinweis: Werden in einem Programm Ein- und Ausgaben zu der gleichen wahlfreien Datei vorgenommen, kann gewöhnlich mit einer Anweisung OPEN und einer Anweisung FIELD gearbeitet werden.

3. Mit der Anweisung GET den gewünschten Satz in den Puffer für wahlfreien Zugriff bringen.

4. Die Daten in dem Puffer können jetzt vom Programm verarbeitet werden. Numerische Werte müssen mit Hilfe der Umwandlungsfunktionen in Zahlen umgewandelt werden: CVI für ganze Zahlen, CVS für Werte einfacher Genauigkeit und CVD für Werte doppelter Genauigkeit.

Das folgende Programm illustriert diese Schritte:

```
100 OPEN "FILE" AS 1 LEN=32   'Schritt 1
200 FIELD #1 20 AS N$, 4 AS A$, 8 AS P$
                              'Schritt 2
300 GET #1, CODE%             'Schritt 3
400 PRINT N$                  'Schritt 4
500 PRINT CVS(A$)             'noch Schritt 4
```

Mit PROGRAM4 wird die wahlfreie Datei "FILE", die in PROGRAM3 erstellt wurde, verarbeitet. Durch Eingabe des zweistelligen Codes über die Tastatur wird die Information für diesen Satz von der Datei gelesen und am Bildschirm angezeigt.

Programm 4

```
1 REM PROGRAM4 - Verarbeiten wahlfreie Datei
1Ø OPEN "FILE" AS 1 LEN=32
2Ø FIELD #1, 2Ø AS N$, 4 AS A$, 8 AS P$
3Ø INPUT "zweistelliger Code";CODE%
35 IF CODE%=99 THEN CLOSE: END
4Ø GET #1, CODE%
5Ø PRINT N$
6Ø PRINT USING "$$###.##";CVS(A$)
7Ø PRINT P$: PRINT
8Ø GOTO 3Ø
```

Die Funktion LOC übergibt für wahlfreie Dateien die laufende Satznummer. Die laufende Satznummer ist die letzte Satznummer, die in einer Anweisung GET oder PUT benutzt wurde. Die Anweisung

```
IF LOC(1)>5Ø THEN END
```

beendet z.B. die Programmausführung, wenn die laufende Satznummer in Datei 1 größer ist als 50.

Beispielprogramm

PROGRAM5 ist ein Inventurprogramm, das den Zugriff auf eine wahlfreie Datei zeigt. In diesem Programm wird die Satznummer als Teilenummer benutzt, und es wird angenommen, daß das Lager nicht mehr als 100 verschiedene Teilenummern enthält. Die Zeilen 690 bis 750 initialisieren die Datendatei durch Schreiben des Zeichens CHR$(255) als erstes Zeichen jedes Satzes. Dies wird später dazu benutzt (Zeile 180 und Zeile 320), um zu bestimmen, ob für eine bestimmte Teilenummer schon ein Eintrag besteht.

Die Zeilen 40 bis 120 zeigen die verschiedenen Inventurfunktionen an, die das Programm ausführt. Wird die gewünschte Funktionsnummer eingegeben, verzweigt Zeile 140 zu dem zugehörigen Unterprogramm.

Programm 5

```
10 REM PROGRAM5 - inventory
20 OPEN "inven.dat" AS #1 LEN=39
30 FIELD #1,1 AS F$,30 AS D$,2 AS Q$,2 AS R$,4 AS P$
40 PRINT: PRINT"Options:": PRINT
50 PRINT 1,"Initialize File"
60 PRINT 2,"Create a New Entry"
70 PRINT 3,"Display Inventory for One Part"
80 PRINT 4,"Add to Stock"
90 PRINT 5,"Subtract from Stock"
100 PRINT 6,"List Items Below Reorder Level"
110 PRINT 7,"End Application"
120 PRINT: PRINT: INPUT "Choice";CHOICE
130 IF (CHOICE<1) OR (CHOICE>7) THEN PRINT "Bad Choice Number"
    : GOTO 40
140 ON CHOICE GOSUB 690, 160, 300, 390, 470, 590, 760
150 GOTO 120
160 REM      build new entry
170 GOSUB 670
180 IF ASC(F$)<>255 THEN INPUT "Overwrite";A$;
    IF A$<>"y" AND A$<>"Y" THEN RETURN
190 LSET F$=CHR$(0)
200 INPUT "Description";DESC$
210 LSET D$=DESC$
```

```
220 INPUT "Quantity in stock";Q%
230 LSET Q$=MKI$(Q%)
240 INPUT "Reorder level":R%
250 LSET R$=MKI$(R%)
260 INPUT "Unit price";P
270 LSET P$=MKS$(P)
280 PUT #1,PART%
290 RETURN
300 REM      display entry
310 GOSUB 670
320 IF ASC(F$)=255 THEN PRINT "Null entry"; RETURN
330 PRINT USING "Part number ###";PART%
340 PRINT D$
350 PRINT USING "Quantity on hand #####";CVI(Q$)
360 PRINT USING "Reorder level #####";CVI(R$)
370 PRINT USING "Unit price $$###.##";CVS(P$)
380 RETURN
390 REM      add to stock
400 GOSUB 670
410 IF ASC(F$)=255 THEN PRINT"Null entry":RETURN
420 PRINT D$:INPUT "Quantity to add";A%
430 Q%=CVI(Q$)+A%
440 LSET Q$=MKI$(Q%)
450 PUT #1,PART%
460 RETURN
470 REM      remove from stock
480 GOSUB 670
490 IF ASC(F$)=255 THEN PRINT "Null entry": RETURN
500 PRINT D$
510 INPUT "Quantity to subtract":S%
520 Q%=CVI(Q$)
530 IF (Q%-S%)<0 THEN PRINT"Only";Q%;"in stock": GOTO 510
540 Q%=Q%-S%
550 IF Q%=<CVI(R$) THEN PRINT "Quantity now";Q%;
      ", Reorder level";CVI(R$)
560 LSET Q$=MKI$(Q%)
570 PUT #1,PART%
580 RETURN
590 REM      list items below reorder level
600 FOR I=1 TO 100
610 GET #1,I
620 IF ASC(F$)=255 THEN 640
630 IF CVI(Q$)<CVI(R$) THEN PRINT D$;" Quantity";CVI(Q$)
      TAB(50) "Reorder level";CVI(R$)
640 NEXT I
650 RETURN
660 REM   get part number
670 INPUT "Part number";PART%
680 IF PART%<1 OR PART%>100
      THEN PRINT "Bad part number": GOTO 670
      ELSE GET #1,PART%: RETURN
690 REM initialize file
700 INPUT "Are you sure";B$: IF B$<>"Y" AND B$<>"y"
    THEN RETURN
710 LSET F$=CHR$(255)
720 FOR I=1 TO 100
730 PUT #1,I
740 NEXT I
750 RETURN
760 REM      end application
770 CLOSE: END
```

Tips für den Durchsatz

- Werden keine wahlfreien Dateien benutzt, sollte beim Starten mit BASIC im BASIC-Befehl **/S:0** angegeben werden. Dadurch werden 128 Bytes multipliziert mit der Anzahl der angegebenen Dateien in der Auswahl **/F:** gespart.

- BASIC setzt als Standard drei Dateien. Werden weniger als drei benutzt, sollte beim Starten mit BASIC mit Hilfe des BASIC-Befehls **/F:***Dateien* gesetzt werden. Dabei muß beachtet werden, daß Bildschirm, Tastatur und Drucker nicht als Dateien zählen, solange sie nicht explizit durch OPEN eröffnet werden.

- Sequentielle Dateien benötigen einen Puffer von 128 Bytes. Wahlfreie Dateien benötigen als Standard auch einen Puffer von 128 Bytes. Diese Angabe kann aber durch die Auswahl **/S:** im Befehl BASIC überschrieben werden. Es bringt keinen Vorteil, wenn **/S:** auf eine Zahl gesetzt wird, die größer als die größte Satzlänge im Programm für wahlfreie Dateien ist. Jedoch ergibt eine Kombination einer Satzlänge von 512 und **/S:512** verbesserten Durchsatz, da die Sektorgröße der Disketten 512 Bytes beträgt.

Soll mit sequentieller Ein-/Ausgabe gearbeitet werden, aber der Durchsatz besser sein, kann man mit wahlfreien Dateien arbeiten, die man "pseudosequentiell" verarbeitet. Beispiel:

```
1 ' Beispiel 1A
1Ø OPEN "ABC" FOR OUTPUT AS #1
2Ø FOR I=1 TO 3ØØØ
3Ø PRINT #1,"MELH"
4Ø NEXT
5Ø CLOSE #1: END
```

In diesem Beispiel (1A) wird mit normalen sequentiellen Ein-/Ausgabebefehlen gearbeitet, um eine Datei mit 3000 Sätzen zu erstellen.

```
1 ' Beispiel 1B
1Ø OPEN "ABC" FOR INPUT AS #1
2Ø OPEN "DEF" FOR OUTPUT AS #2
3Ø IF EOF(1) THEN CLOSE: END
4Ø INPUT #1,A$
5Ø PRINT #2,A$
6Ø GOTO 3Ø
7Ø END
```

Im Beispiel 1B wird die sequentielle Datei "ABC", die gerade erstellt wurde, in eine Datei mit dem Namen "DEF" kopiert.

Mit den nächsten Beispielen wird die gleiche Arbeit mit Hilfe von "pseudosequentieller" Ein-/Ausgabe durchgeführt:

```
1 ' Beispiel 2A
1Ø OPEN "PQR" AS #1 LEN=512
15 ON ERROR GOTO 9Ø
2Ø FOR I=1 TO 3ØØØ
3Ø PRINT #1, "MELH"
4Ø NEXT
45 PRINT #1,"/eof"
5Ø ON ERROR GOTO Ø: PUT #1: CLOSE
6Ø END
9Ø PUT #1: RESUME
```

Das Beispiel 2A erstellt eine Datei mit 3000 Sätzen mit Hilfe von wahlfreier Ein-/Ausgabe. Dies ist eine "pseudosequentielle" Datei.

```
1 ' Beispiel 2B
10 OPEN "PQR" AS #1 LEN=512
20 OPEN "XYZ" AS #2 LEN=512
30 ON ERROR GOTO 80
40 GET #1
50 INPUT #1,A$
60 PRINT #2,A$
70 IF A$<>"/eof" THEN 50 ELSE
      ON ERROR GOTO 0: PUT #2: CLOSE: END
80 IF ERL=50 THEN GET #1: RESUME
      ELSE PUT #2: RESUME
```

Dieses letzte Beispiel kopiert die "pseudosequentielle" Datei, die mit dem vorherigen Beispiel erstellt wurde, in eine neue "pseudosequentielle" Datei mit Namen "XYZ". Dies dauert etwa halb solang wie das Beispiel mit der sequentiellen Ein-/Ausgabe.

Die Technik, die in diesem Beispiel benutzt wird, beinhaltet das Abfragen des Fehlers "FIELD overflow" (FIELD-Überlauf) (Fehler 50) und die Ausführung der zugehörigen Anweisungen GET oder PUT, um den Puffer wieder leer zu machen (Zeile 90 im Beispiel 2A und Zeile 80 in Beispiel 2B). Es muß auch beachtet werden, daß ein Schein-Dateiende geschrieben ("/eof" in diesem Beispiel) und während der Eingabe wieder geprüft werden muß. Auch benutzen die Anweisungen INPUT und PRINT nur eine einzige Variable statt einer Variablenliste.

Diese Technik ist nur sinnvoll, wenn mehr als eine Datei gleichzeitig eröffnet ist.

Anhang C: Unterprogramme in Maschinensprache

In diesem Anhang wird beschrieben, wie BASIC Verbindung zu Unterprogrammen in Maschinensprache herstellt. Insbesondere wird beschrieben:

- wie den Unterprogrammen Hauptspeicherplatz zugeordnet wird;
- wie Unterprogramme in Maschinensprache in den Hauptspeicher geladen werden;
- wie das Unterprogramm von BASIC aufgerufen wird und wie Parameter übergeben werden.

Dieser Anhang ist für erfahrene Maschinenprogrammprogrammierer gedacht.

Referenzmaterial

- Rector, Russell and Alexy, George. *The 8086 Book,* Osborne/McGraw-Hill, Berkeley, California, 1980 (includes the 8088).
- Intel Corporation Literature Department. *The 8086 Family User's Manual,* 9800722. 3065 Bowers Avenue, Santa Clara, CA 95051.
- IBM Corporation Personal Computer library. *Macro-Assembler.* Boca Raton, FL 33432.
- IBM Corporation Personal Computer library. *Technical Reference.* Boca Raton, FL 33432.

Hauptspeicherzuordung für eigene Unterprogramme

BASIC benutzt normalerweise den ganzen verfügbaren Hauptspeicher von seinem Startpunkt bis zu einem Maximum von 64K Bytes. Dieser BASIC-Arbeitsbereich enthält das BASIC-Programm und die Daten, auch den Arbeitsbereich für den Interpretierer und einen BASIC-Stapelbereich. Für Unterprogramme in Maschinensprache kann Hauptspeicherplatz innerhalb oder außerhalb dieses BASIC-64K-Arbeitsbereichs angelegt werden. Wo diese Routinen untergebracht werden, hängt davon ab, wieviel Hauptspeicher insgesamt zur Verfügung stehen und wie groß die zu ladenden Anwendungen sind.

Ein System benötigt mehr als einen 64K Hauptspeicher, wenn Unterprogramme in Machinensprache außerhalb dieses BASIC-64K-Arbeitsbereichs gespeichert werden sollen. Wird mit Disketten- oder erweitertem BASIC gearbeitet, benötigt DOS ungefähr 12K Bytes, und BASIC benötigt weitere 10K Bytes, so daß mindeste ein 96K-Byte-System benötigt wird, um außerhalb des BASIC-Arbeitsbereichs Platz für Unterprogramme in Maschinensprache zu haben.

Außerhalb des BASIC-Arbeitsbereichs: Wenn im System genug Hauptspeicher zur Verfügung steht, daß man eigene Unterprogramme außerhalb des BASIC-64K-Bytes-Arbeitsbereichs speichern kann, muß nichts getan werden, um diesen Bereich zu reservieren. Mit Hilfe der Anweisung DEF SEG adressiert man den externen Unterprogrammbereich außerhalb des BASIC-Arbeitsbereichs.

Möchte man z.B. in einem 96K-Byte-System eine Adresse in den oberen 4K Bytes des Hauptspeichers angeben, kann man dies folgendermaßen tun:

```
11Ø DEF SEG=&H17ØØ
```

Mit dieser Anweisung wird ein Segment angegeben, das mit der hexadezimalen Adresse 17000 (92 K) beginnt.

Innerhalb des BASIC-Arbeitsbereichs: Damit BASIC nicht die Unterprogramme in Maschinensprache überschreibt, muß entweder:

- die Anweisung CLEAR benutzt werden, die in allen Versionen von BASIC verfügbar ist;
- oder die Auswahl **/M:** in dem BASIC-Befehl benutzt werden, um das Disketten- oder erweiterte BASIC von DOS zu starten.

Nur die höchsten Hauptspeicherplätze können für Unterprogramme reserviert werden. Will man z.B. die höchsten 4 K Bytes des BASIC-64K-Arbeitsbereichs für Unterprogramme in Maschinensprache reservieren, kann man dies folgendermaßen tun:

```
1Ø CLEAR ,&HFØØØ
```

oder BASIC mit dem DOS-Befehl starten:

```
BASIC /M:&HFØØØ
```

Jede dieser Anweisungen schränkt die Größe des BASIC-Arbeitsbereichs auf hex F000 (60K) Bytes ein, so daß man die höchsten 4K Bytes für Unterprogramme in Maschinensprache benutzen kann.

Laden des Unterprogramms in den Hauptspeicher

Im folgenden werden einige Anregungen gegeben, wie man Unterprogramme in Maschinensprache laden kann. Es werden nicht alle möglichen Situationen beschrieben.

Zwei häufig benutzte Wege, ein Programm in Maschinensprache in den Hauptspeicher zu laden, sind folgende:

- es über das BASIC-Programm mit der Anweisung POKE in den Hauptspeicher laden.
- es von einer Datei auf Diskette oder Kassette in den Hauptspeicher laden.

Laden mit Hilfe der Anweisung POKE

Man kann relativ kurze Unterprogramme in Maschinensprache codieren und sie mit Hilfe der Anweisung POKE in den Hauptspeicher laden. Auf diese Weise wird das Unterprogramm Teil des BASIC-Programms. Ein Weg, dies zu tun, ist der folgende:

1. Bestimmen des Maschinencodes für das Unterprogramm.
2. Den hexadezimalen Wert (&Hxx-Format) jedes Bytes des Codes in Anweisungen DATA bringen.
3. Eine Schleife ausführen, die jedes Daten-Byte liest und es in den Bereich bringt, der für das Unterprogramm ausgewählt wurde (wie oben angegeben).

4. Nachdem die Schleife beendet ist, ist das Unterprogramm geladen. Soll das Unterprogramm mit Hilfe der Funktion USR aufgerufen werden, muß die Anweisung DEF USR ausgeführt werden, um die Eingangsadresse des Unterprogramms zu definieren. Soll das Unterprogramm mit der Anweisung CALL aufgerufen werden, muß der Wert der Unterprogrammvariablen auf die Unterprogrammeingangsadresse gesetzt werden.

Beispiel:

```
Ok
10 DEFINT A-Z
20 DEF SEG=&H1700
30 FOR I=0 TO 21
40 READ J
50 POKE I,J
60 NEXT
70 SUBRT=0
80 A=2:B=3:C=0
90 CALL SUBRT (A,B,C)
100 PRINT C
110 END
120 DATA &H55,&H8B,&HEC,&H8B,&H76,&H0A
130 DATA &H8B,&H04,&H8B,&H76,&H08
140 DATA &H03,&H04,&H8B,&H7E,&H06
150 DATA &H89,&H05,&H5D,&HCA,&H06,&H00
RUN
5
Ok
```

Laden eines Unterprogramms von einer Datei

Mit dem BASIC-Befehl BLOAD kann man eine Hauptspeicherabbilddatei direkt in den Hauptspeicher laden. Dieses Hauptspeicherabbild kann ein Unterprogramm in Maschinensprache sein, das mit Hilfe des Befehls BSAVE gespeichert wurde.

Dies führt natürlich zu der Frage, wie das Unterprogramm zum erstenmal an diesen Platz gebracht wurde. Das Unterprogramm in Maschinensprache kann eine ausführbare Datei sein, die vom Binder in DOS erstellt wurde und in den Hauptspeicher mit DEBUG gespeichert wurde. DEBUG und der Binder sind im *IBM Personalcomputer Betriebssystem DOS (Disk Operating System) Handbuch* beschrieben.

Im folgenden Abschnitt wird ein Weg vorgeschlagen, wie mit Hilfe von BLOAD solch ein Unterprogramm in Maschinensprache in den Hauptspeicher geladen werden kann:

1. Mit dem Binder eine .EXE-Datei der Routine erstellen (hier ASMROUT.EXE genannt), so daß sie an das obere Ende des Hauptspeichers geladen wird.

2. BASIC unter DEBUG durch die folgende Eingabe laden:

   ```
   DEBUG BASIC.COM
   ```

3. Die Register anzeigen (mit Hilfe des Befehls R), um herauszufinden, wo sich BASIC im Hauptspeicher befindet. Die Werte der Register aufschreiben (CS, IP, SS, SP, DS, ES), um sie später benutzen zu können.

4. Mit DEBUG die Datei .EXE (das Unterprogramm) in den oberen Hauptspeicherbereich laden, wo es den Übergangsbereich für COMMAND.COM überlagert.

```
N ASMROUT.EXE
L
```

5. Die Register anzeigen (mit Hilfe des Befehls R), um herauszufinden, wo das Unterprogramm in den Hauptspeicher geladen wurde. Die Werte der Register CS und IP für spätere Benutzung aufschreiben.

6. Die Register (mit Hilfe des Befehls R) zurück auf ihre alten Werte setzen, die sie enthielten, als BASIC.COM ursprünglich geladen war. Dazu die aufgeschriebenen Werte aus Schritt 3 benutzen.

7. Falls erforderlich, können die Parameter zu BASIC mit Hilfe des DEBUG-Befehls N übertragen werden. N initialisiert den Parameter übertragungsbereich

   N /M:40000

   Hinweis: Der Eingang in den Arbeitsbereich /M:max ist nur erforderlich, wenn sich das Unterprogramm innerhalb des BASIC-64K-Arbeitsbereiches befindet.

8. Mit Hilfe des Befehls G zum BASIC-Eingangspunkt verzweigen und Unterbrechungspunkte (falls gewünscht) im Unterprogramm in Maschinensprache setzen.

9. Wenn die BASIC-Anfrage erscheint, das BASIC-Anwendungsprogramm laden und DEF SEG und entweder die Anweisung DEF USR oder den Wert der Variablen CALL ändern, damit er gleich an dem Platz des Unterprogramms ist, der bestimmt wurde, als das Unterprogramm in Schritt 5 geladen wurde.

   - Den vorher ermittelten Wert des Registers CS für DEF SEG verwenden.
   - Den vorher ermittelten Wert des Registers IP für DEF USR oder den Variablenwert für CALL verwenden.

10. Im direkten Modus in BASIC den Befehl BSAVE eingeben, um den Unterprogrammbereich zu speichern. Dazu die Startadresse, definiert in den Registern CS und IP, als das Unterprogramm in Schritt 5 geladen wurde, und die Codelänge, die in der Assembler-Liste oder LINK Hauptspeicherbelegung gedruckt wurde, verwenden (siehe Befehl BSAVE in Kapitel 4).

11. Das BASIC-Anwendungsprogramm so ändern, daß es eine Anweisung BLOAD nach DEF SEG enthält, das den richtigen Wert von CS für das Unterprogramm setzt.

    Hinweis: Ist das Maschinenspracheunterprogramm verschiebbar, kann BLOAD dieses Unterprogramm an einem anderen Platz speichern als es der Binder ursprünglich gespeichert hatte. Wird solch eine Veränderung vorgenommen, muß die gleiche Änderung auch in der Anweisung DEF SEG vorgenommen werden, die mit diesem Aufruf in Verbindung steht, so daß BASIC das Unterprogramm zur Ausführungszeit finden kann.

Einige Vorschläge für andere mögliche Plätze für das Unterprogramm sind:

- ein unbenutzter Bildschirmpuffer;
- ein unbenutzter Dateipuffer (gefunden mit VARPTR(*#f*));
- ein Zeichenkettenvariablenbereich, gefunden durch VARPTR(*Zeichenkettenvariable*).

(Siehe Befehl BLOAD und Funktion VARPTR in Kapitel 4.)

12. Das sich daraus ergebende veränderte BASIC-Programm speichern.

    Beispiel eines BASIC-Programms, das ein Assembler-Unterprogramm aufruft:

```
5  A%=2 : A%=3
1Ø DEF SEG=&H27EØ
15 BLOAD "B:ASMROUT",Ø
2Ø SUBRT=Ø
3Ø CALL SUBRT(A%,B%,C%)
4Ø PRINT C%
```

Hinweise für die Benutzung von DEBUG mit BASIC: Wird BASIC unter DEBUG ausgeführt, so wird BASIC nach DEBUG in den Hauptspeicher geladen, so daß DEBUG nicht überschrieben wird, wenn ein großes BASIC-Programm geladen wird. Werden im Maschinenspracheunterprogramm Unterbrechungspunkte gesetzt, führen diese Unterbrechungen zu DEBUG zurück. Der Befehl SYSTEM führt ebenfalls von BASIC zu DEBUG zurück.

Aufrufen eines Unterprogramms vom BASIC-Programm

Alle Versionen des BASIC beinhalten zwei Wege, Maschinenspracheunterprogramme aufzurufen: die Funktion USR und die Anweisung CALL. In diesem Abschnitt wird die Benutzung von USR und CALL beschrieben.

Gemeinsamkeiten von CALL und USR

Für das Aufrufen von Maschinenspracheunterprogrammen mit CALL oder mit der Funktion USR müssen die folgenden Punkte beachtet werden:

Eingang in das Unterprogramm

- Zur Eingangszeit müssen die Segmentregister DS, ES und SS auf den gleichen Wert gesetzt werden, und zwar auf die Adresse des BASIC-Datenbereichs (Standardannahme für DEF SEG).

- Zur Eingangszeit enthält das Codesegmentregister CS den laufenden Wert, der in der letzten Anweisung DEF SEG angegeben wurde. Wurde DEF SEG nicht spezifiziert oder spezifizierte die letzte DEF SEG keinen Überschreibungswert, ist der Wert in CS der gleiche wie in den anderen drei Segmentregistern.

Zeichenkettenargumente

- Ist das Eingabeargument eine Zeichenkette, ist der Wert, der in dem Argument empfangen wurde, die Adresse eines Drei-Byte-Bereichs, der *Zeichenkettenbeschreibung* genannt wird:

  1. Byte 0 der Zeichenkettenbeschreibung enthält die Länge der Zeichenkette (0 bis 255).

  2. Byte 1 der Zeichenkettenbeschreibung enthält die niedrigen acht Bits des Relativzeigers der Zeichenkette im BASIC-Datenbereich.

  3. Byte 2 der Zeichenkettenbeschreibung enthält die höchsten acht Bits des Relativzeigers der Zeichenkette im BASIC-Datenbereich.

Auf die Zeichenkette selbst wird durch die zwei letzten Bytes der Zeichenkettenbeschreibung gedeutet.

Warnung:
Das Unterprogramm darf keines der drei Bytes der *Zeichenkettenbeschreibung* ändern.

Das Unterprogramm darf den *Inhalt* der Zeichenkette ändern, aber nicht seine *Länge*.

Verändert das Unterprogramm den Inhalt einer Zeichenkette, kann dies das *Programm verändern*. Das folgende Beispiel kann die Zeichenkette "TEXT" in einem BASIC-Programm verändern:

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

Jedoch ändert das nächste Beispiel nicht das Programm, weil durch die Zeichenkettenverbindung BASIC die Zeichenkette in den Zeichenkettenbereich kopiert, wo er geändert werden kann, ohne den Originaltext zu beeinflussen.

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

Rücksprung aus dem Unterprogramm

- Der Rücksprung zu BASIC muß mit der Intersegmentinstruktion RET geschehen. (Das Unterprogramm ist eine FAR-Prozedur.)

- Bevor zurückverzweigt wird, müssen alle Segmentregister und der Stapelbereichszeiger SP auf den alten Stand gebracht werden. Alle anderen Register (und Anzeiger) können geändert bleiben.

- Der Stapelbereichszeiger zeigt zur Zeit des Eingangs in das Unterprogramm auf einen Bereich von nur 16 Bytes (acht Wörter), der für das Unterprogramm verfügbar ist. Wird mehr Platz benötigt, muß das Unterprogramm selbst seinen Stapelsegmentbereich und seinen Stapelbereichszeiger setzen. Die Adresse des laufenden Bereichs muß vermerkt werden, so daß die Adresse des Zeigers wieder auf den alten Stand gesetzt werden kann bevor zurückverzweigt wird.

- Wurden Unterbrechungen durch das Unterprogramm inaktiviert, müssen sie vor dem Rückverzweigen wieder aktiviert werden.

Anweisung CALL

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

CALL *numerische variable* [(*Variablenliste*)]

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

Variablenliste
enthält die Variablen, getrennt durch Kommas (,), die als Argument an das Unterprogramm übergeben werden (die Argumente dürfen keine Konstanten sein).

Die Ausführung der Anweisung CALL bedingt folgende Schrite:

1. Für jede Variable in der Variablenliste wird die Adresse der Variablen in den Stapelbereich gebracht. Die Adresse wird als Zwei-Byte-Relativzeiger im BASIC-Datensegment angegeben (Standardannahme DEF SEG).

2. Die Rücksprungadresse, spezifiziert im Register CS und der Relativzeiger werden in den Stapelbereich gebracht.

3. Die Steuerung wird an das Maschinenspracheunterprogramm mit Hilfe der Segmentadresse, die in der letzten Anweisung DEF SEG angegeben wurde, und dem Relativzeiger, der durch den Wert von *numerische Variable* angegeben wurde, übertragen.

Hinweise für die Anweisung CALL

- Es können Werte an BASIC durch die Argumente übergeben werden, indem die Werte der Variablen in der Argumentliste geändert werden.

- Ist das Argument eine Zeichenkette, zeigt der Relativzeiger für das Argument auf eine drei Bytes lange *Zeichenkettenbeschreibung*, wie vorher erklärt wurde.

- Das aufgerufene Unterprogramm muß wissen, wie viele Argumente übergeben wurden. Die Parameter werden dadurch angesprochen, daß ein positiver Relativzeiger zu BP addiert wird, nachdem das aufgerufene Unterprogramm den laufenden Stapelbereichzeiger nach BP überträgt. Die ersten Instruktionen im Unterprogramm müssen wie folgt aussehen:

```
PUSH BP      :RETTEN BP
MOV BP,SP    :OBERGEBEN SP NACH BP
```

Der Relativzeiger in dem Stapelbereich irgendeines Arguments wird wie folgt berechnet:

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

Beschreibung:

n ist die Gesamtanzahl der übergebenen Argumente.

m ist die Position eines spezifischen Arguments in der Argumentliste der BASIC-Anweisung CALL (*m* muß sich im Bereich 1 bis *n* befinden).

Beispiel: Im folgenden Beispiel werden die Werte in A% und B% addiert und das Ergebnis nach C% gespeichert:

Dies sind die zugehörigen BASIC-Anweisungen:

```
1ØØ A%=2: B%=3
2ØØ DEF SEG=&H27EØ
25Ø BLOAD "SUBRT.EXE",Ø
3ØØ SUBRT=Ø
4ØØ CALL SUBRT (A%,B%,C%)
5ØØ PRINT C%
```

Hinweis: Zeile 200 setzt das Segment auf den Platz Hex 27E00. SUBRT wird auf 0 (Null) gesetzt, so daß der Aufruf von SUBRT das Unterprogramm an der Adresse &H27E00 ausführt.

Die folgenden Anweisungen sind im IBM Personalcomputer Macro-Assembler-Quellencode geschrieben:

```
CSEG    SEGMENT
        ASSUME   CS:CSEG
SUBRT   PROC     FAR
        PUSH BP          ;SAVE BP
        MOV BP,SP        ;SET BASE PARM LIST
        MOV SI,[BP]+10   ;GET ADDR PARM A
        MOV AX,[SI]      ;GET VALUE OF A
        MOV SI,[BP]+8    ;GET ADDR PARM B
        ADD AX,[SI]      ;ADD VALUE B TO REG
        MOV DI,[BP]+6    ;GET ADDR PARM C
        MOV [DI],AX      ;PASS BACK SUM
        POP BP           ;RESTORE BP
        RET 6            ;FAR RETURN TO BASIC
SUBRT   ENDP
CSEG    ENDS
        END
```

Hinweis: Wird das Unterprogramm mit Hilfe der Anweisung CALL aufgerufen, muß das Unterprogramm mit RET *n* zurückverzweigen, wobei *n* zweimal die Anzahl der Argumente in der Variablenliste bedeutet. Dies ist nötig, um den Stapelbereich auf die Adresse zu justieren, auf der er zu Beginn der Aufruffolge stand.

Ein anderes Beispiel:

```
 1Ø DEFINT A-Z
1ØØ DEF SEG=&H18ØØ
11Ø BLOAD "SUBRT.EXE",Ø
12Ø SUBRT=Ø
13Ø CALL SUBRT (A,B$,C)
```

Der folgende Makro-Assembler-Code zeigt, wie die Argumente (einschließlich der Adresse einer Zeichenkettenbeschreibung) übergeben und angesprochen werden, und wie das Ergebnis in der Variablen C gespeichert wird:

```
PUSH   BP           ;SAVE BP
MOV    BP,SP        ;GET CURRENT STK POSITION INTO BP
MOV    BX,[BP]+8    ;GET ADDR OF B$ STRING DESCRIPTOR
MOV    CL,[BX]      ;GET LENGTH OF B$ INTO CL
MOV    DX,1[BX]     ;GET ADDR OF B$ TEXT INTO DX
  .
  .
  .
MOV    SI,[BP]+1Ø   ;GET ADDR OF A INTO SI
MOV    DI,[BP]+6    ;GET ADDR OF C INTO DI
MOVS   WORD         ;STORE VARIABLE A INTO C
POP    BP           ;RESTORE BP
RET    6            ;RESTORE STACK, RETURN
END
```

Warnung:
Es liegt vollständig in der Verantwortung des Programmierers, daß die Argumente in der Anweisung CALL in Anzahl, Typ und Länge mit den Argumenten, die von dem Unterprogramm erwartet werden, übereinstimmen.

Im vorherigen Beispiel kopiert die Instruktion **MOVS WORD** nur zwei Bytes, weil ihre Variablen A und C ganzzahlig sind. Sind A und C jedoch von einfacher Genauigkeit, müssen vier Bytes kopiert werden. Sind A und C von doppelter Genauigkeit, müssen acht Bytes kopiert werden.

USR-Funktionsaufrufe

Die andere Art, Maschinenspracheunterprogramme von BASIC aufzurufen, ist die mit der Funktion USR. Das Format der Funktion USR ist wie folgt:

USR[*n*](*arg*)

n muß eine einstellige Zahl im Bereich 0 bis 9 sein.

arg ist ein numerischer Ausdruck oder der Name einer Zeichenkettenvariablen.

n gibt an, welches USR-Unterprogramm aufgerufen wird und stimmt mit der Ziffer überein, die für dieses Unterprogramm in der Anweisung DEF USR angegeben wird. Wird *n* weggelassen, wird USR0 angenommen. Die in der Anweisung DEF USR angegebene Adresse bestimmt die Startadresse des Unterprogramms. Auch wenn das Unterprogramm kein Argument benötigt, muß immer ein Scheinargument angegeben werden.

Wird die Funktion USR aufgerufen, enthält Register AL einen Wert, der den Typ des übergebenen Arguments angibt. Der Wert von AL ist einer der folgenden:

Wert in AL	Typ des Arguments
2	Ganze Zahl aus zwei Bytes (Zweier-Komplement)
3	Zeichenkette
4	Zahl in einfacher Genauigkeit
8	Zahl in doppelter Genauigkeit

Ist das Argument eine Zeichenkette, zeigt das Register DX auf eine drei Bytes lange Zeichenkettenbeschreibung (siehe "Gemeinsamkeiten von CALL und USR" wie zuvor beschrieben).

Ist das Argument eine Zahl und keine Zeichenkette, wird der Wert des Arguments in dem Gleitkommaakkumulator (FAC) gespeichert, der aus einem acht Bytes langen Bereich im BASIC-Datenbereich besteht. In diesem Fall enthält das Register BX den Relativzeiger in dem BASIC-Datenbereich zum fünften Byte des acht Bytes langen FAC. Im folgenden Beispiel soll sich FAC in den Bytes Hex 49F bis Hex 4A6 befinden; das bedeutet, daß BX den Wert Hex 4A3 enthält:

Das Argument ist eine ganze Zahl:

- Hex 4A4 enthält die oberen acht Bits des Arguments.

- Hex 4A3 enthält die niedrigen acht Bits des Arguments.

Ist das Argument eine Zahl in einfacher Genauigkeit:

- enthält Hex 4A6 den Exponenten minus 128, und der Binärpunkt befindet sich links des signifikantesten Bits der Mantisse. Hex 4A5 enthält die höchsten sieben Bits der Mantisse, wobei die führende 1 unterdrückt wird (impliziert). Bit 7 ist das Vorzeichen der Zahl (0 = positiv; 1 = negativ);

- enthält Hex 4A4 die mittleren acht Bits der Mantisse;

- enthält Hex 4A3 die niedrigsten acht Bits der Mantisse.

Ist das Argument eine Zahl mit doppelter Genauigkeit:

- enthalten Hex 4A3 bis Hex 4A6 die gleichen Werte, wie unter einer Zahl mit einfacher Genauigkeit im vorherigen Abschnitt beschrieben;

- enthalten Hex 49F bis Hex 4A2 vier weitere Bytes der Mantisse (Hex 49F enthält die niedrigsten acht Bits).

Gewöhnlich sind die mit der Funktion USR übergebenen Werte vom gleichen Typ (ganzzahlig, Zeichenkette, einfache Genauigkeit oder doppelte Genauigkeit) wie das Argument, das an die Funktion übergeben wurde. Jedoch kann ein numerisches Argument der Funktion, unabhängig von seinem Typ, als ganzzahliger Wert behandelt werden, wenn das Unterprogramm FRCINT aufgerufen wird, um das ganzzahlige Äquivalent des Arguments zu erhalten, das im Register BX gespeichert wurde.

Soll der durch die Funktion übergebene Wert ganzzahlig sein, muß der sich ergebende Wert im Register BX gespeichert werden. Dann muß ein Aufruf zu MAKINT erfolgen, kurz bevor der Zwischensegmentrücksprung ausgeführt wird. Dadurch wird der Wert zurück an BASIC übergeben, indem er nach FAC (Gleitkommaakkumulator) gespeichert wird.

Das Ansprechen von FRCINT und MAKINT wird im folgenden Beispiel gezeigt:

```
100 DEF SEG=&H1800
120 BLOAD "SUBRT.EXE",0
130 DEF USR0=0
140 X = 5 'X hat einfache Genauigkeit
150 Y = USR0(X)
160 PRINT Y
```

An die Adresse 1800:0 (Segment:Relativzeiger) wurde das folgende Makro-Assembler-Unterprogramm geladen. Das Unterprogramm verdoppelt das übergebene Argument und gibt ein ganzzahliges Ergebnis zurück:

```
RSEG      SEGMENT AT 0F600H ;BASE OF BASIC ROM
          ORG   3       ;OFFSET TO FORCE INTEGER
  FRCINT  LABEL FAR
          ORG   7       ;OFFSET TO MAKE INTEFER
  MAKINT  LABEL FAR
RSEG      ENDS
  .
  .
  .
CSEG      SEGMENT
USRPRG    PROC  FAR     ;ENTRY POINT
          CALL  FRCINT  ;FORCE ARG IN FAC INTO [BX]
          ADD   BX,BX   ;[BX] = [BX]  * 2
          CALL  MAKINT  ;PUT INT RSLT IN BX INTO FAC
          RET           ;INTER-SEGMENT RETURN TO BASIC
USRPRG    ENDP
CSEG      ENDS
```

Hinweis: FRCINT und MAKINT führen Zwischensegmentrücksprünge aus. Man muß sicherstellen, daß die Aufrufe zu FRCINT und MAKINT durch eine FAR- Prozedur definiert sind.

Anhang D. Programmumwandlung in BASIC des IBM Personalcomputers

Da BASIC des IBM Personalcomputers sehr ähnlich zum BASIC vieler Mikrocomputer ist, unterstützt der IBM Personalcomputer Programme, die für eine große Anzahl von Mikrocomputern geschrieben sind. Stehen Programme zur Verfügung, die in einem anderen BASIC als dem für den IBM Personalcomputer geschrieben sind, so sind nur einige kleine Änderungen erforderlich, bevor sie mit dem BASIC des IBM Personalcomputers laufen können. In diesem Abschnitt werden einige spezifische Änderungen aufgezeigt, wenn BASIC-Programme umgewandelt werden.

Dateiein-/-ausgabe

Mit BASIC des IBM Personalcomputers werden Informationen in eine Datei auf Diskette oder Kassette geschrieben oder von Diskette und Kassette gelesen, indem man eine Datei eröffnet und eine Verbindung mit einer speziellen Dateinummer herstellt. Danach werden die zugehörigen Ein-/Ausgabeanweisungen mit dieser Dateinummer benutzt. Ein-/Ausgabe zu Disketten- und Kassettendateien wird in einigen anderen BASICs unterschiedlich implementiert. Im Abschnitt "Dateien" in Kapitel 3 und in "Anweisung OPEN" in Kapitel 4 sind weitere Informationen enthalten.

Auch werden im BASIC des IBM Personalcomputers Sätze für wahlfreie Dateien automatisch geblockt, so daß so viele Sätze wie möglich in jeden Sektor passen.

Grafiken

Das Zeichnen auf dem Bildschirm variiert sehr stark zwischen den verschiedenen BASICs. In Kapitel 3 wird genau beschrieben, wie Grafiken mit dem IBM Personalcomputer erstellt werden.

IF...THEN

Die Anweisung IF im BASIC des IBM Personalcomputers enthält wahlweise die Angabe ELSE, die ausgeführt wird, wenn der zu testende Ausdruck falsch ist. Einige andere BASICs enthalten diese Möglichkeit nicht. In einem anderen BASIC könnte z.B. der folgende Programmabschnitt so geschrieben sein:

```
10 IF A=B THEN 30
20 PRINT "NICHT GLEICH"  : GOTO 40
30 PRINT "GLEICH"
40 REM CONTINUE
```

Diese Programmfolge läuft auch im BASIC des IBM Personalcomputers richtig; könnte aber dort wie folgt codiert werden:

```
10 IF A=B THEN PRINT "GLEICH" ELSE PRINT "NICHT GLEICH"
20 REM CONTINUE
```

Das BASIC des IBM Personalcomputers erlaubt mehrere Anweisungen in den Angaben THEN und ELSE. Dies könnte ein Programm, das in einem anderen BASIC geschrieben ist, anders ablaufen lassen. Beispiel:

```
1Ø IF A=B THEN GOTO 1ØØ : PRINT "NICHT GLEICH"
2Ø REM CONTINUE
```

Falls in einigen anderen BASICs der Test A=B falsch ist, verzweigt die Steuerung zur nächsten Anweisung. Das bedeutet, falls A nicht gleich B ist, wird "NICHT GLEICH" ausgegeben. Im BASIC des IBM Personalcomputers werden sowohl GOTO 100 und PRINT "NICHT GLEICH" als Teil der Angabe THEN der Anweisung IF betrachtet. Falls der Test falsch ist, verzweigt die Steuerung zur nächsten Programmzeile. Das ist in diesem Beispiel Zeile 20. PRINT "NICHT GLEICH" kann niemals ausgeführt werden.

Dieses Beispiel kann für das BASIC des IBM Personalcomputers wie folgt neu codiert werden:

```
1Ø IF A=B THEN 1ØØ ELSE PRINT "NICHT GLEICH"
2Ø REM CONTINUE
```

Zeilenvorschub

Wird in anderen BASICs ein Zeilenvorschub eingegeben, wird dieses Zeilenvorschubzeichen in den Text eingefügt. Beim IBM Personalcomputer wird durch Eingabe des Zeilenvorschubs der Rest der Bildschirmzeile mit Leerstellen gefüllt. Es wird kein Zeilenvorschubzeichen eingefügt. Versucht man, ein Programm mit Zeilenvorschubzeichen zu laden, erhält man den Fehler "Direct Statement in File" (Direktanweisung in der Datei).

Logische Operationen

Im BASIC des IBM Personalcomputers werden logische Operationen (NOT, AND, OR, XOR, IMP und EQV) Bit für Bit mit ganzzahligen Operanden durchgeführt, um ein ganzzahliges Ergebnis zu erhalten. In einigen anderen BASICs werden die Operanden einfach als Wahr- (nicht Null) oder Falsch- (Null) Werte betrachtet, und das Ergebnis der Operation ist entweder wahr oder falsch. Als Beispiel dieses Unterschieds dient dieses kleine Programm:

```
10 A=9: B=2
20 IF A AND B THEN PRINT "A UND B SIND WAHR"
```

Dieses Beispiel in einem anderen BASIC wird wie folgt ausgeführt: A ist nicht Null, also ist es wahr; B ist auch nicht Null, also ist es auch wahr. Da A und B wahr sind, ist auch A AND B wahr, deshalb druckt das Programm **A UND B SIND WAHR.**

Das BASIC des IBM Personalcomputers rechnet jedoch anders; A ist 1001 in Binärform und B ist 0010 in Binärform, so ist A AND B (Bit für Bit berechnet) 0000 oder Null; Null zeigt falsch an, deshalb wird die Nachricht *nicht* ausgegeben und das Programm fährt mit der Ausführung der nächsten Zeile fort.

Diese kann nicht nur Tests in IF-Anweisungen, sondern auch Berechnungen beeinflussen. Um gleiche Ergebnisse zu erhalten, müssen logische Ausdrücke wie folgt neu codiert werden:

```
1Ø A=9: B=2
2Ø IF (A <> Ø) AND (B <> Ø)
      THEN PRINT "A UND B SIND WAHR"
```

MAT-Funktionen

Programme, die die Funktionen MAT benutzen, die in einigen BASICs verfügbar sind, müssen mit Hilfe von FOR...NEXT-Schleifen neu geschrieben werden, um ausgeführt werden zu können.

Mehrfachzuordnungen

Einige BASICs erlauben Anweisungen der folgenden Form:

```
1Ø LET B=C=Ø
```

um B und C auf 0 zu setzen. Das BASIC des IBM Personalcomputers würde das zweite Gleichheitszeichen (=) als logischen Operator interpretieren und B gleich −1 setzen, falls C 0 wäre. Diese Anweisung muß deshalb in zwei Zuordnungsanweisungen umgewandelt werden:

```
1Ø C=0:B=Ø
```

Mehrfachanweisungen

Einige BASICs benutzen einen umgekehrten Schrägstrich (\ ), um mehrere Anweisungen auf einer Zeile zu trennen. Beim BASIC des IBM Personalcomputers müssen alle Anweisungen einer Zeile durch einen Doppelpunkt (:) getrennt werden.

Anweisungen PEEK und POKE

Die Anweisungen PEEK und POKE hängen vom jeweils benutzten Computer ab. Der *Zweck* für PEEK und POKE in einem Programm eines anderen BASIC muß geprüft werden, und die Anweisungen müssen so umgewandelt werden, daß sie die gleiche Funktion auf dem IBM Personalcomputer ausführen.

Vergleichsausdrücke

Im BASIC des IBM Personalcomputers ist der Wert, der von dem Vergleichsausdruck übergeben wird, wie z.B. für A>B entweder −1, was anzeigt, daß der Vergleich wahr ist, oder 0, was anzeigt, daß der Vergleich falsch ist. Einige andere BASICs übergeben eine positive 1 für wahr. Wird der Wert eines Vergleichsausdrucks in einer arithmetischen Berechnung benutzt, sind die Ergebnisse unterschiedlich.

Bemerkungen

Einige BASICs erlauben es, ans Ende einer Zeile mit Hilfe eines Ausrufezeichens (!) Bemerkungen hinzuzufügen. Dies muß für das BASIC des IBM Personalcomputers in ein Apostroph (') geändert werden.

Runden von Zahlen

Das BASIC des IBM Personalcomputers rundet Zahlen einfacher und doppelter Genauigkeit, wenn ein ganzzahliger Wert benötigt wird. Viele anderen BASICs unterdrücken dagegen den Wert hinter dem Komma. Dies kann den Programmablauf verändern, da es nicht nur Zuordnungsanweisungen beeinflußt (z.B. I%=2.5 ergibt in I% eine 3), sondern auch Funktions- und Anweisungsberechnungen (z.B. TAB(4.5) geht zur fünften Position, A(1.5) ist das gleiche wie A(2) und X=11.5 MOD4 ergibt einen Wert von 0 für X). Dieses Runden könnte das BASIC des IBM Personalcomputers dazu veranlassen, ein anderes Element aus einem Bereich auszuwählen als ein anderes BASIC - möglicherweise eines, das sich nicht mehr im Bereich befindet!

Aufruf des Alarmzeichens

Einige BASICs benötigen PRINT CHR$(7), um das ASCII-Alarmzeichen zu senden. Im BASIC des IBM Personalcomputers kann man diese Anweisung durch BEEP ersetzen, auch wenn dies nicht erforderlich ist.

Zeichenkettenverarbeitung

Zeichenkettenlänge: Da die Zeichenketten im BASIC des IBM Personalcomputers eine variable Länge haben, müssen alle Anweisungen gelöscht werden, die die Länge von Zeichenketten definieren. Eine Anweisung, wie z.B. DIM A$(I,J), die einen Zeichenkettenbereich für J-Elemente der Länge I dimensioniert, muß für das BASIC des IBM Personalcomputers in DIM A$(J) umgeschrieben werden.

Verkettung: Einige BASICs benutzen ein Komma (,) oder ein Ampersand (&), um Zeichenketten zu verbinden. Beide Zeichen müssen in ein Pluszeichen (+) geändert werden, da dies der Operator für die Zeichenverkettung im BASIC des IBM Personalcomputers ist.

Teilzeichenketten: Im BASIC des IBM Personalcomputers werden mit Hilfe der Funktionen MID$, RIGHT$ und LEFT$ Teile von Zeichenketten aus einer Zeichenkette entnommen. Formen, wie A$(I), um das Ith-Zeichen in A$ anzusprechen, oder A$(I,J), um eine Teilzeichenkette aus A$ von der Position I bis zur Position J herauszunehmen, müssen wie folgt geändert werden:

Anderes BASIC	BASIC IBM Personalcomputer
`X$=A$(I)`	`X$=MID$(A$,I,1)`
`X$=A$(I,J)`	`X$=MID$(A$,I,J-I+1)`

Steht die Angabe der Teilzeichenkette links der Zuordnung und werden X$ Zeichen in A$ ersetzt, muß dies wie folgt umgewandelt werden:

Anderes BASIC	BASIC IBM Personalcomputer
`A$(I)=X$`	`MID$(A$,I,1)=X$`
`A$(I,J)=X$`	`MID$(A$,I,J-I+1)=X$`

Benutzung von Leerstellen

Einige BASICs erlauben die Anweisungen ohne Trennung der Schlüsselwörter:

```
20FORI=1TOX
```

Im BASIC des IBM Personalcomputers müssen alle Schlüsselwörter durch Leerstellen getrennt sein:

```
20 FOR I=1 TO X
```

Anderes

Die BASIC-Sprache anderer Computer kann sich noch in anderen Punkten als den hier aufgelisteten vom BASIC des IBM Personalcomputers unterscheiden. Man muß sich im BASIC des IBM Personalcomputers so gut wie möglich auskennen, um jede benötigte Funktion richtig umzucodieren.

Notizen:

Anhang E. Mathematische Funktionen

Funktionen, die nicht im BASIC des IBM Personalcomputers eingebaut sind, können wie folgt berechnet werden:

Funktion	Äquivalent
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) = (ESP(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))

Funktion	Äquivalent
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 für den Arcussinus Hyperbolikus jedesmal, wenn er benötigt wird, zu codieren, könnte man dies folgendermaßen tun:

```
DEF FNARCSINH(X) = LOG(X+SQR(X*X+1))
```

Danach könte man die Funktion wie folgt aufrufen:

```
FNARCSINH(Y)
```

Anhang F. Datenfernverarbeitung

In diesem Anhang werden die BASIC-Anweisungen beschrieben, die benötigt werden, um die Schnittstelle RS232 asynchrone Datenfernverarbeitung mit anderen Computern und Peripheriegeräten zu unterstützen.

Eröffnen einer Datenfernverarbeitungsdatei

Durch OPEN "COM.. wird ein Puffer für die Ein-/Ausgabe in der gleichen Art und Weise wie mit dem OPEN für Diskettendateien angelegt. Siehe Anweisung OPEN "COM... in Kapitel 4.

Datenfernverarbeitungsein-/-ausgabe

Da jeder Datenfernverarbeitungsanschluß als Datei eröffnet wird, sind alle Ein-/Ausgabeanweisungen, die für Diskettendateien gültig sind, auch für die Datenfernverarbeitung gültig.

Sequentielle Ein-/Ausgabeanweisungen für Datenfernverarbeitung sind die gleichen wie für Diskettendateien. Es sind die folgenden:

INPUT #
LINE INPUT #
INPUT$

Die sequentiellen Ausgabeanweisungen für Datenfernverarbeitung sind die gleichen wie für Diskettendateien. Es sind die folgenden:

PRINT #
PRINT # USING
WRITE #

In den Abschnitten INPUT und PRINT wird die Syntax und ihr Gebrauch erklärt.

GET und PUT für Datenfernverarbeitungsdateien

GET und PUT unterscheiden sich für Datenfernverarbeitungsdateien nur wenig von denen für Diskettendateien. Sie werden für Ein-/Ausgabe fester Länge 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 übertragen werden sollen.

Diese Zahl darf den Wert, der durch die Angabe LEN in der Anweisung OPEN "COM... gesetzt wurde, nicht überschreiten. Siehe unter GET und PUT in Kapitel 4.

Ein-/Ausgabefunktionen

Der schwierigste Aspekt für die asynchrone Datenfernverarbeitung ist es, die Zeichen so schnell zu verarbeiten, wie sie empfangen werden. Bei Datenraten von 1200 bps oder höher ist es nötig, die Zeichenübertragung 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 Übertragung wieder aufgenommen werden kann. XOFF sagt dem anderen Computer, daß er mit der Übertragung aufhören soll, XON sagt ihm, daß er wieder senden kann.

> **Hinweis:** Dies ist eine allgemein benutzte Konvention, die aber nicht universal ist. Sie hängt von dem implementierten Protokoll ab, das zwischen dem anderen Computer oder Peripheriegerät und dem eigenen Computer aufgestellt wurde.

Im Disketten- und erweiterten BASIC bestehen drei Funktionen, die helfen, eine Überlaufbedingung zu bestimmen. Es sind die folgenden:

LOC(f) Übergibt die Anzahl der Zeichen im Eingabepuffer, die darauf warten, gelesen zu werden. Ist die Anzahl größer als 255, übergibt LOC 255.

LOF(f) Übergibt die Anzahl des freien Platzes im Eingabepuffer. Dies ist das gleiche wie *n*-LOC(f), wobei *n* die Größe des Datenfernverarbeitungspuffers ist, wie er durch die Auswahl /**C:** im BASIC-Befehl gesetzt wurde. Die Standardannahme für *n* ist 256.

EOF(f) Übergibt Wahr (−1), wenn der Eingabepuffer leer ist; Falsch (0), wenn irgendwelche Zeichen darauf warten, gelesen zu werden.

Hinweis: Es kann ein "Communication Buffer Overflow" (Datenfernverarbeitungspufferüberlauf) auftreten, wenn ein Lesen versucht wird, nachdem der Eingabepuffer voll ist (wenn LOF(f) eine 0 übergibt).

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 können. INPUT # wird am wenigsten benutzt, da die Eingabe stoppt, sobald ein Komma oder ein Schreibkopfrücklaufzeichen gelesen wird. LINE INPUT # stoppt, wenn ein Schreibkopfrücklaufzeichen gelesen wird.

Mit INPUT$ können alle Zeichen gelesen und einer Zeichenkette zugeordnet werden. INPUT$(*n,f*) übergibt *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 A$ gebracht und verarbeitet, und zwar so lange, wie sich Zeichen im Eingabepuffer befinden. Befinden sich mehr als 255 Zeichen im Puffer, werden zu einer Zeit nur 255 übertragen, damit kein String Overflow (Zeichenkettenüberlauf) auftritt. Weiter wird in diesem Fall EOF(1) falsch, und die Eingabe geht weiter, bis der Eingabepuffer leer ist – einfach, kurz und schnell.

Um die Daten schnell zu verarbeiten, sollte man nach Möglichkeit vermeiden, jedes Zeichen zu prüfen, 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.

Beispielprogramm

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

```
 10 REM   dumb terminal example
 20 'set screen to black and white text mode
 30 '     and set width to 40
 40 SCREEN 0,0: WIDTH 40
 50 'turn off soft key display; clear screen;
 60 '   make sure all files are closed
 70 KEY OFF: CLS: CLOSE
 80 'define all numeric variables as integer
 90 DEFINT A-Z
100 'define true and false
110 FASLE=0: TRUE= NOT FALSE
120 'define the XON, XOFF characters
130 XOFF$=CHR$(19): XON$=CHR$(17)
140 'open communications to file number 1,
150 ' 300 bps, EVEN parity, 7 data bits
160 OPEN "COM1:300,E,7" A$ #1
170 'use screen as a file, just for fun
180 OPEN "SCRN:" FOR OUTPUT AS 2
190 'turn cursor on
200 LOCATE ,,1
400 PAUSE=FALSE: ON ERROR GOTO 9000
490 '
500 'send keyboard input to com line
510 B$=INKEY$: IF B$<>"" THEN PRINT #1 B$;
520 'if no chars in com buffer, check key in
530 IF EOF(1) THEN 510
540 'if buffer more than 1/2 full, then
550 '   set PAUSE flag to say input suspended,
560 '   send XOFF to host to stop transmission
570 IF LOC(1)>128 THEN PAUSE=TRUE: PRINT #1,XOFF$;
580 'read contents of com buffer
590 A$=INPUT$(LOC(1),#1)
600 'get rid of linefeeds to avoid double spaces
610 '   when input displayed on screen
620 LFT=0
```

```
630 LFP=INSTR(LFP+1,A$,CHR$(10)) 'look for LF
640 IF LFP>0 THEN MID$(A$,LFP,1)=" ": GOTO 630
650 'display com input, and check for more
660 PRINT #2,A$;: IF LOC(1)>0 THEN 570
670 'if transmission suspended by XOFF,
680 '   resume by sending XON
690 IF PAUSE THEN PAUSE=FALSE: PRINT #1,XON$;
700 'check for keyboard input again
710 GOTO 510
8999 'if error, display error number and retry
9000 PRINT "ERROR NO.";ERR: RESUME
```

Programmhinweise

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

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

- Man beachte, daß in Zeile 510 INKEY$ eine leere Zeichenkette übergeben wird, wenn kein Zeichen mehr verfügbar ist.

Operation der Steuersignale

Dieser Abschnitt enthält detailliertere technische Informationen, die benötigt werden, wenn eine Verbindung mit anderen Computern oder Peripheriegeräten über BASIC programmiert werden soll.

Die Ausgabe des asynchronen Datenfernverarbeitungsanschlusses entspricht dem EIA RS232-C Standard für Schnittstellen zwischen Datenendeinrichtungen und Datenfernverarbeitungseinrichtungen. Dieser Standard definiert eine Anzahl Steuersignale, die vom IBM Personalcomputer übertragen oder empfangen werden, um den Austausch von Daten zwischen einem anderen Computer oder einer peripheren Einheit zu steuern. Diese Signale sind Gerätesteuerspannungen, die entweder Ein (größer als +3 Volt) oder Aus (kleiner als −3 Volt) sind. Siehe *IBM Personalcomputer Technisches Handbuch.*

Steuerung der Ausgabesignale mit OPEN

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

Benutzung der Eingabesteuersignale

Ist normalerweise eine der Leitungen CTS (Clear To Send - Senderbereitschaft) oder DSR (Data Set Ready - Betriebsbereitschaft) aus, wird die Anweisung OPEN "COM... nicht ausgeführt. Nach einer Sekunde kehrt BASIC mit dem Fehler Device Timeout (Einheitenzeitsperre) (Fehlercode 24) zurück. Der CD (Carrier Detekt-Empfangssignalpegel) kann entweder Ein oder Aus sein. Er hat auf die Operation des Programms keinen Einfluß.

Man kann jedoch angeben, wie diese Leitungen mit Hilfe der Auswahlen **RS, CS, DS** und **CD** in der Anweisung OPEN "COM... zu testen sind. Siehe unter "Anweisung OPEN "COM..." in Kapitel 4.

Wird eines der zu testenden Signale ausgeschaltet während das Programm ausgeführt wird, arbeiten die Ein-/Ausgabeanweisungen für die Datenfernverarbeitungsdatei nicht. Wird z.B. die Anweisung PRINT # ausgeführt nachdem die Leitung CTS oder DSR ausgeschaltet wurde, erscheint einer der Fehler Device Fault (Fehlercode 25) (Einheitenfehler) oder Device Timeout (Einheitenzeitsperre) (Fehlercode 24). Die Leitungen für RTS und DTR bleiben an, auch wenn einer dieser Fehler auftritt.

Man kann eine Leitungsunterbrechung mit Hilfe der Funktion INP testen, um die Bits aus dem MODEM Statusregister der Anschlußkarte für asynchrone Übertragung zu lesen. Im folgenden Abschnitt "Testen der Modemsteuersignale" werden die Einzelheiten beschrieben.

Testen der Modemsteuersignale

Vier Eingabesteuersignale werden von der Anschlußkarte für asynchrone Übertragung aufgenommen. Es handelt sich um die Signale CTS und DSR, die vorher beschrieben wurden, den Empfangssignalpegel (Carrier Detect oder Received Line Signal Detect) (Stift 8) und den Ringanzeiger (Stift 22). Man kann in der Anweisung OPEN "COM... angeben, wie die CTS-, DSR- und CD-Leitungen getestet werden sollen. Der Ringanzeiger wird von den Datenfernverarbeitungsfunktionen in BASIC überhaupt nicht benutzt.

Soll eines dieser Signale in einem Programm getestet werden, kann man die den Signalen zugehörigen Bits im Modemstatusregister auf der Anschlußkarte für asynchrone Übertragung prüfen.

Mit der Funktion INP liest man die acht Bits dieses Registers. Um die Register eines unveränderten Datenfernverarbeitungsanschlusses zu lesen, benutzt man INP(&H3FE). Um die Register eines veränderten Datenfernverarbeitungsanschlusses zu lesen, benutzt man INP(&H2FE). Siehe unter "Adapter für asynchrone Übertragung" im *IBM Personalcomputer Technisches Handbuch*. Es enthält eine Beschreibung, welche Bits im Statusregister zu welchen Steuersignalen gehören. Man kann auch die Delta-Bits in diesem Register benutzen, um zu bestimmen, ob Zwischensignale auf einer der Steuerleitungen erschienen sind. Ein Steuersignal hat nur Bedeutung, wenn der Stift, der zu diesem Signal gehört, mit dem Kabel des Modems oder des anderen Computers verbunden ist.

Man kann auch die Bits in dem Zeilenstatusregister der Anschlußkarte für asynchrone Übertragung testen. Mit INP(&H3FD) erhält man den Inhalt des Registers für einen unveränderten Datenfernverarbeitungsanschluß. Mit INP(&H2FD) erhält man den Inhalt in einem veränderten Datenfernverarbeitungsanschluß. Auch diese Bits sind im *IBM Personalcomputer Technisches Handbuch* beschrieben. Mit diesen Bits kann man bestimmen, welche Fehlertypen beim Empfang der Zeichen über die Datenfernverarbeitungsleitung auftraten, oder ob ein Unterbrechungssignal erkannt wurde.

Direkte Steuerung der Ausgabesteuersignale

Man kann die RTS- und DTR-Steuersignale direkt über ein BASIC-Programm mit der Anweisung OUT steuern. Der Status (Ein oder Aus) dieser Signale wird über Bits im Modemsteuerregister der Anschlußkarte für asynchrone Übertragung gesteuert. Die Adresse dieses Registers ist &H3FC an einem nicht veränderten Datenfernverarbeitungsanschluß und &H2FC an einem veränderten Datenfernverarbeitungsanschluß. Im *IBM Personalcomputer Technisches Handbuch* wird beschrieben, welches dieser Bits zu welchem Signal gehört.

Die Bits im Leitungssteuerregister der Anschlußkarte für asynchrone Übertragung können auch verändert werden. Man muß jedoch bei der Veränderung vorsichtig sein, da die meisten dieser Bits im Register durch BASIC zur Zeit der Ausführung der Anweisung OPEN festgesetzt wurden und die Veränderung eines Bits einen Fehler in der Datenfernverarbeitung verursachen könnte. Das Leitungssteuerregister liegt auf Adresse &H3FB bei einem nicht veränderten Datenfernverarbeitungsanschluß und auf Adresse &H2FB bei einem veränderten Datenfernverarbeitungsanschluß.

Wenn Bits im Modemsteuerregister oder im Zeilensteuerregister verändert werden, muß zuerst das Register (mit der Funktion INP) gelesen werden und dann das Register nur mit dem oder den veränderten Bits zurückgeschrieben werden.

Ein Bit, das in dem Leitungssteuerregister gesteuert werden könnte ist Bit 6, das Bit zum Setzen einer Unterbrechung. Dieses Bit erlaubt es, ein Unterbrechungssignal auf der Datenfernverarbeitungs-Sendeleitung zu erzeugen. Eine Unterbrechung wird oft benutzt, dem remote Computer mitzuteilen, daß er die Übertragung beenden soll. Im Normalfall dauert eine Unterbrechung eine halbe Sekunde. Um dieses Signal zu erzeugen, muß das Bit "Setzen Unterbrechung" eingeschaltet, die gewünschte Zeit für das Unterbrechungssignal abgewartet werden und dann das Bit wieder ausgeschaltet werden. Mit den folgenden BASIC-Anweisungen wird ein Unterbrechungssignal von ungefähr einer halben Sekunde an einem nicht veränderten Datenfernverarbeitungsanschluß erzeugt.

```
100 IC%=INP(&H3FB) 'get contents of modem register
110 IZ%=IC% OR &H40 'turn ON the Set Break bit
120 OUT &H3FB,IZ%  'transmit to modem control register
130 FOR I=1 TO 500: NEXT I 'delay half a second
140 OUT &H3FB,IC% 'turn Set Break bit OFF in register
```

Datenfernverarbeitungsfehler

Fehler treten für Datenfernverarbeitungsdateien in der folgenden Reihenfolge auf:

1. Beim Eröffnen 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" (Datenfernverarbeitungspufferüberlauf): Wenn ein Überlauf eintritt.

   b) "Device I/O Error" (Einheitenein-/-ausgabefehler): Für Überlauf, Unterbrechung, Parität 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 verlorengeht, während BASIC etwas anderes ausführte.

   b) "Device Timeout" (Einheitenzeitsperre): Wenn CTS, DSR oder CD verlorengeht, während das Programm darauf wartet, Daten in den Ausgabepuffer stellen zu können.

Notizen:

Anhang G: ASCII-Zeichencodes

In der folgenden Tabelle sind alle ASCII-Codes (dezimal) und ihre zugehörigen Zeichen aufgelistet. Diese Zeichen können mit Hilfe von PRINT CHR$(*n*) angezeigt werden, wobei *n* der ASCII-Code ist. In der Spalte mit der Überschrift "Steuerzeichen" sind die Standardinterpretationen der ASCII-Codes 0 bis 31 aufgeführt (gewöhnlich für Steuerfunktionen oder Datenfernverarbeitung benutzt).

Jedes dieser Zeichen kann über die Tastatur eingegeben werden, indem man die Taste "Alt" betätigt und niederhält und dann die Ziffern für den ASCII-Code auf der Zehnertastatur eingibt. Dabei muß beachtet werden, daß einige dieser Codes eine besondere Bedeutung für das BASIC-Korrekturprogramm haben - das BASIC-Korrekturprogramm benutzt seine eigene Interpretation für diese Codes und kann die hier aufgelisteten Sonderzeichen nicht anzeigen.

ASCII-Wert	Zeichen	Steuerzeichen	ASCII-Wert	Zeichen
000	(Leer)	NUL	032	(Leerstelle)
001	☺	SOH	033	!
002	☻	STX	034	"
003	♥	ETX	035	#
004	♦	EOT	036	$
005	♣	ENQ	037	%
006	♠	ACK	038	&
007	(Alarm)	BEL	039	'
008	◘	BS	040	(
009	(Tabulator)	HT	041	)
010	(Zeilenvorschub)	LF	042	*
011	(Grundstellung)	VT	043	+
012	(Seitenvorschub)	FF	044	,
013	(Schreibkopfrücklauf)	CR	045	-
014	♫	SO	046	.
015	☼	SI	047	/
016	►	DLE	048	0
017	◄	DC1	049	1
018	↕	DC2	050	2
019	!!	DC3	051	3
020	¶	DC4	052	4
021	§	NAK	053	5
022	▬	SYN	054	6
023	↨	ETB	055	7
024	↑	CAN	056	8
025	↓	EM	057	9
026	→	SUB	058	:
027	←	ESC	059	;
028	(Positionsanzeiger nach rechts)	FS	060	<
029	(Positionsanzeiger nach links)	GS	061	=
030	(Positionsanzeiger nach oben)	RS	062	>
031	(Positionsanzeiger nach unten)	US	063	?

ASCII-Wert	Zeichen	ASCII-Wert	Zeichen
064	@	096	`
065	A	097	a
066	B	098	b
067	C	099	c
068	D	100	d
069	E	101	e
070	F	102	f
071	G	103	g
072	H	104	h
073	I	105	i
074	J	106	j
075	K	107	k
076	L	108	l
077	M	109	m
078	N	110	n
079	O	111	o
080	P	112	p
081	Q	113	q
082	R	114	r
083	S	115	s
084	T	116	t
085	U	117	u
086	V	118	v
087	W	119	w
088	X	120	x
089	Y	121	y
090	Z	122	z
091	[	123	{
092	\	124	|
093	]	125	}
094	^	126	~
095	_	127	⌂

ASCII-Wert	Zeichen	ASCII-Wert	Zeichen
128	Ç	160	á
129	ü	161	í
130	é	162	ó
131	â	163	ú
132	ä	164	ñ
133	à	165	Ñ
134	å	166	ª
135	ç	167	º
136	ê	168	¿
137	ë	169	⌐
138	è	170	¬
139	ï	171	½
140	î	172	¼
141	ì	173	¡
142	Ä	174	«
143	Å	175	»
144	É	176	▒
145	æ	177	▒
146	Æ	178	▒
147	ô	179	│
148	ö	180	┤
149	ò	181	╡
150	û	182	╢
151	ù	183	╖
152	ÿ	184	╕
153	Ö	185	╣
154	Ü	186	║
155	¢	187	╗
156	£	188	╝
157	¥	189	╜
158	Pt	190	╛
159	ƒ	191	┐

ASCII-Wert	Zeichen	ASCII-Wert	Zeichen
192		224	α
193		225	β
194		226	Γ
195		227	π
196		228	Σ
197		229	σ
198		230	μ
199		231	τ
200		232	Φ
201		233	Θ
202		234	Ω
203		235	δ
204		236	∞
205		237	Ø
206		238	ϵ
207		239	∩
208		240	≡
209		241	±
210		242	≥
211		243	≤
212		244	⌠
213		245	⌡
214		246	÷
215		247	≈
216		248	°
217		249	•
218		250	·
219		251	√
220		252	n
221		253	2
222		254	■
223		255	(Leerzeichen 'FF')

Erweiterte Codes

Für gewisse Tasten und Tastenkombinationen, die nicht im Standard ASCII-Code dargestellt werden können, wird ein erweiterter Code von der Systemvariablen INKEY$ übergeben. Eine Leerstelle (ASCII-Code 000) wird als erstes Zeichen einer Zwei-Zeichen-Zeichenkette übergeben. Wird von INKEY$ eine Zeichenkette aus zwei Zeichen empfangen, sollte man zurückverzweigen und das zweite Zeichen prüfen, um zu bestimmen, welche Taste betätigt wurde. Im allgemeinen, aber nicht immer, ist dieser zweite Code der Prüfcode der Primärtaste, die betätigt wurde. Die ASCII-Codes (dezimal) für dieses zweite Zeichen und die zugehörigen Tasten sind auf der folgenden Seite 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	Positionsanzeiger nach oben
73	Pg Up
75	Positionsanzeiger nach links
77	Positionsanzeiger nach rechts
79	End
80	Positionsanzeiger 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-Positionsanzeiger nach links (vorheriges Wort)
116	Ctrl-Positionsanzeiger nach rechts (nächstes 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

Notizen:

Anhang H: 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

Notizen:

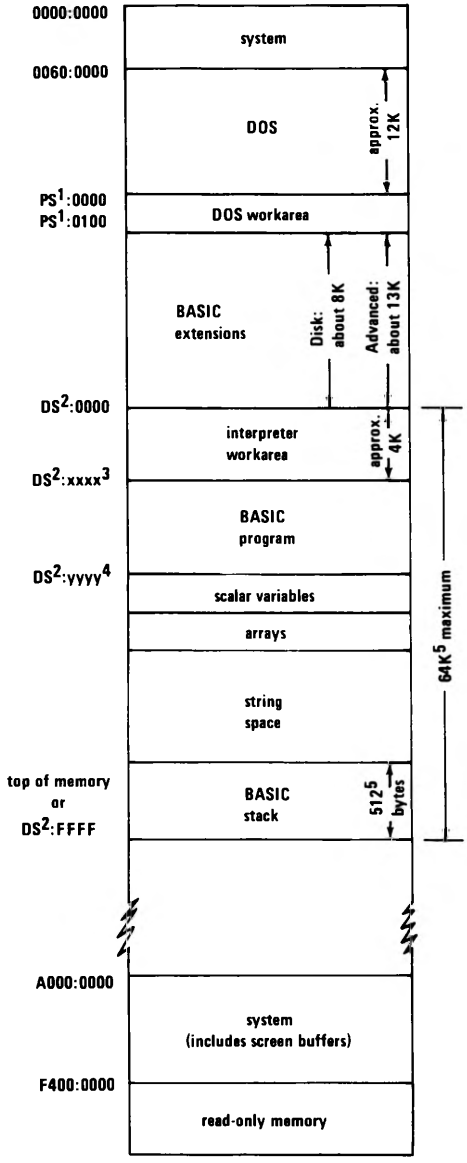
Anhang I: Technische Informationen und Tips

Dieser Anhang enthält weitere spezifische technische Informationen über BASIC. Es ist eine Beschreibung enthalten über Aufteilung des Hauptspeichers, Beschreibungen, wie BASIC Daten intern speichert und einige spezielle Techniken, wie man den Programmdurchsatz verbessern kann.

Weitere Informationen siehe im *IBM Personalcomputer Technisches Handbuch.*

Hauptspeicherbelegung

Die folgende Hauptspeicheraufteilung gilt für Disketten- und erweitertes BASIC. DOS und die BASIC-Erweiterungen gibt es nicht für Kassetten-BASIC. Die Adressen stehen in der folgenden Hexadezimalform *Segment:Relativzeiger.*

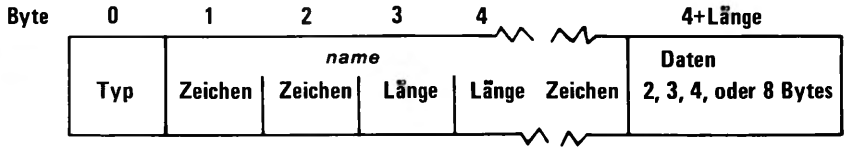


Hinweise:

1. PS bezieht sich auf das DOS-Programmsegment
2. DS bezieht sich auf das BASIC-Datensegment
3. Die Nummer xxxx steht in den Adressen DS:30 und DS:31 (niedriges Byte, hohes Byte)
4. Die Nummer yyyy steht in den Adressen DS:358, DS:359 (niedriges Byte, hohes Byte)
5. Oder wird vom Befehl CLEAR gesetzt

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 zusätzlichen Zeichen werden ab Byte 4 gespeichert.

Dies bedeutet, daß jeder Variablenname mindestens drei Bytes belegt. Ein Name aus ein oder zwei Zeichen benötigt exakt drei Bytes. Ein Name aus x Zeichen benötigt $x+1$ 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 übergeben wird, zeigt auf diese Daten.

Für Zeichenkettenvariablen bedeutet *Daten* die Zeichenkettenbeschreibung:

- Das erste Byte der Zeichenkettenbeschreibung enthält die Länge 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 dem höheren Byte danach gespeichert:

  — Das zweite Byte der Zeichenkettenbeschreibung enthält also das niedrige Byte des Relativzeigers.

  — Das dritte Byte der Zeichenkettenbeschreibung enthält das hohe Byte des Relativzeigers.

Für numerische Variablen enthält *Daten* den aktuellen Wert der Variablen:

- Ganzzahlige Werte werden in zwei Bytes gespeichert, das niedrigste Byte zuerst und das hohe Byte danach.

- Werte einfacher Genauigkeit werden in vier Bytes gespeichert, und zwar in der internen Gleitkommabinärdarstellung von BASIC.

- Werte doppelter Genauigkeit werden in acht Bytes gespeichert, und zwar in der internen Gleitkommabinärdarstellung von BASIC.

BASIC-Dateisteuerblock

Wird VARPTR mit einer Dateinummer als Argument aufgerufen, ist der übergebene Wert die Adresse des BASIC-Dateisteuerblocks. Diese Adresse ist der Relativzeiger in das BASIC-Datensegment. (Es muß beachtet werden, daß der BASIC-Dateisteuerblock nicht der gleiche ist wie der DOS-Dateisteuerblock).

Der Dateisteuerblock enthält die folgenden Informationen (Relativzeiger sind relativ zum durch VARPTR übergebenen Wert):

Relativ-zeiger	Länge	Beschreibung
0	1	Der Modus, in dem die Datei eröffnet wurde: 1 - Nur Eingabe 2 - Nur Ausgabe 4 - Direkter Zugriff 16 - Nur Hinzufügen 32 - Interne Benutzung 128 - Interne Benutzung
1	38	DOS-Dateisteuerblock
39	2	Für sequentielle Dateien, die Anzahl der Sektoren, die gelesen oder geschrieben werden. Für Dateien mit wahlfreiem Zugriff enthält 1 + die letzte Satznummer, die gelesen oder geschrieben wurde.

Relativ-zeiger	Länge	Beschreibung
41	1	Anzahl der Bytes im Sektor, wenn gelesen oder geschrieben wird.
42	1	Anzahl der Bytes, die noch im Eingabepuffer stehen.
43	3	(Reserviert)
46	1	Einheitennummer: 0,1 - Diskettenlaufwerke A: und B: 248 - LPT3: 249 - LPT2: 250 - COM2: 251 - COM1: 252 - CAS1: 253 - LPT1: 254 - SCRN: 255 - KYBD:
47	1	Einheitenbreite.
48	1	Position im Puffer für PRINT #
49	1	Interne Benutzung für LOAD und SAVE. Wird für Datendateien nicht benutzt.
50	1	Ausgabeposition während einer Tabulatorerweiterung.

Relativ-zeiger	Länge	Beschreibung
51	128	Physischer Datenpuffer für die Datenübertragung zwischen DOS und BASIC. Dieser Relativzeiger wird benutzt, um Daten im sequentiellen Ein-/Ausgabemodus zu prüfen.
179	2	Variabel lange Satzgröße. Standardannahme ist 128. Gesetzt durch den Längenparameter in der Anweisung OPEN.
181	2	Laufende physische Satznummer.
183	2	Laufende logische Satznummer.
185	1	(Reserviert)
186	2	Nur für Diskettendateien. Position für PRINT #, INPUT # und WRITE #.
188	n	Aktueller FIELD-Datenpuffer. Die Größe n wird durch die Auswahl /S: im BASIC-Befehl bestimmt. Dieser Relativzeiger muß benutzt werden, um Dateidaten im wahlfreien Zugriff zu prüfen.

Tastaturpuffer

Zeichen, die über die Tastatur eingegeben werden, werden im Tastaturpuffer gespeichert, bis sie verarbeitet werden. Bis zu 15 Zeichen können im Puffer gehalten 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$ können 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 gelöscht werden:

```
DEF SEG=Ø: POKE 1Ø5Ø, PEEK(1Ø52)
```

Diese Technik kann nützlich sein, wenn der Puffer gelöscht werden soll, bevor der Benutzer "irgendeine Taste" betätigen 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 gelöscht werden:

```
DEF SEG: POKE 1Ø6,Ø
```

Suchordnung für Anschlüsse

Die Drucker mit Zuordnung LPT1:, LPT2: und LPT3: werden zugeordnet, wenn der Computer eingeschaltet wird. Das System prüft die Druckeranschlüsse nach einer bestimmten Reihenfolge. Der erste gefundene Druckeranschluß erhält LPT1:, der zweite Anschluß (falls vorhanden) wird LPT2: und der dritte (falls vorhanden) wird LPT3:. Die Suchordnung ist wie folgt:

1. IBM Schwarz/Weiß-Bildschirm und Paralleldruckeranschluß
2. Paralleldruckeranschluß
3. Paralleldruckeranschluß, der verändert wurde, um die Basisadresse zu ändern.

Wenn ein Drucker mit Hilfe des Befehls MODE von DOS neu verbunden wird, ist diese Änderung auch in BASIC effektiv.

Die Datenfernverarbeitungseinheiten COM1: und COM2: erhalten ihre Zuordnung ähnlich wie die Drucker. Ihre Suchordnung ist:

1. Anschlußkarte für asynchrone Übertragung
2. Eine veränderte Anschlußkarte für asynchrone Übertragung.

Umschalten der Bildschirme

Besitzt der IBM Personalcomputer die Anschlußkarte für Farb-/Grafikbildschirm und IBM 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 Schwarz-weiß-Bildschirm
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-Bildschirm
20 DEF SEG = 0
30 POKE &H410, (PEEK(&H410) AND &HCF) OR &H10
40 SCREEN 1,0,0,0
50 SCREEN 0
60 WIDTH 40
70 LOCATE ,,1,6,7
```

Hinweis: Mit dieser Technik wird der Bildschirm, zu dem umgeschaltet wird, gelöscht. Vielleicht sollte auch die Stellung der Positionsanzeiger, unabhängig für jeden Bildschirm, verfolgt werden.

Einige Techniken für Farbe

16 Hintergrundfarben: Wird im Textmodus das Blinken nicht benutzt, kann man alle 16 Farben (0 bis 15) als Hintergrundfarbe erhalten, und zwar wie folgt:

Für 40-Spaltenbreite: `OUT &H3D8,8`

Für 80-Spaltenbreite: `OUT &H3D8,9`

Zeichenfarbe im grafischen Modus: Im grafischen Modus können normale Textzeichen angezeigt werden. Wird nicht die US- Tastatur verwendet, siehe Zeichensatz Farb-/Grafikadapter im Kapitel DOS des Bedienerhandbuchs. Für die mittlere Auflösung ist die Vordergrundfarbe für die Zeichen Farbe Nummer 3; für den Hintergrund ist die Farbe Nummer 0.

Durch die folgende Programmzeile kann die Vordergrundfarbe der Zeichen von 3 nach 2 oder 1 geändert werden.

```
DEF SEG: POKE &H4E, Farbe
```

wobei *Farbe* die gewünschte Vordergrundfarbe ist (1, 2 oder 3 - 0 ist *nicht* erlaubt). Spätere Anweisungen PRINT benutzen die angegebene Vordergrundfarbe.

Tips und Techniken

Oft gibt es verschiedene Wege, etwas in BASIC zu codieren, um das gleiche Ergebnis zu erhalten. Dieser Abschnitt enthält verschiedene Codiertips, die die Programmausführungszeit zu verbessern.

ALLGEMEIN

- **Anweisungen kombinieren,** wo es möglich ist, um den Vorteil der Anweisungslänge von 255 Zeichen auszunutzen. Beispiel:

  Besser

  ```
  1ØØ FOR I=1 TO 1Ø: READ A(I): NEXT I
  ```

  Als

  ```
  1ØØ FOR I=1 TO 1Ø
  11Ø READ A(I)
  12Ø NEXT I
  ```

- **Wiederholte Berechnungen für Ausdrücke vermeiden.** Werden identische Berechnungen in mehreren Anweisungen ausgeführt, kann man diese Berechnungen einmal ausführen und das Ergebnis in einer Variablen speichern, bevor diese in späteren Anweisungen benutzt wird. Beispiel:

  Besser

  ```
  3ØØ X=C*3+D
  31Ø A=X+Y
  32Ø B=X+Z
  ```

  Als

  ```
  31Ø A=C*3+D+Y
  32Ø B=C*3+D+Z
  ```

Jedoch geht die Zuordnung einer Konstanten zu einer Variablen schneller, als die Zuordnung des Wertes einer anderen Variablen zu einer Variablen.

- **Einfache Arithmetik benutzen.** Im allgemeinen geht eine Addition schneller als eine Multiplikation, und die Multiplikation ist schneller als die Division oder Potenzierung.

  Betrachten wir diese Beispiele:

Besser	Als
`25Ø B=A*.5`	`25Ø B=A/2`
`5ØØ B=A+A`	`5ØØ B=A*2`
`65Ø B=A*A*A`	`65Ø B=A^3`
`75Ø B%=A%\4`	`75Ø B%=INT(A%/4)`

- **Eingebaute Funktionen benutzen.** Eingebaute Funktionen sind immer schneller, als die gleiche Möglichkeit, in BASIC codiert.

- **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 anfügen, als eine eigene Anweisung benutzen. Dadurch wird der Durchsatz verbessert und es wird Hauptspeicher eingespart, weil keine Zeilennummer benötigt wird. Beispiel:

  Besser

```
1Ø FOR I=1 TO 1Ø
2Ø A(I)=3Ø ' initialisiere A
3Ø NEXT I
```

  Als

```
1Ø FOR I=1 TO 1Ø
25 ' initialisiere A
2Ø A(I)=3Ø
3Ø NEXT I
```

- Hinweis zum BASIC des IBM Personalcomputers. Wenn BASIC zu einer Zeilennummer verzweigt, weiß es nicht genau, wo sich die Zeile im Hauptspeicher befindet. Deshalb muß BASIC durch die Zeilennummern des Programms suchen, und zwar vom Beginn des Programms bis zum Auffinden der Zeilennummer.

  In einigen anderen BASICs muß dieses Suchen jedesmal durchgeführt werden, wenn eine Verzweigung im Programm stattfindet - im BASIC des IBM Personalcomputers wird das Suchen nur einmal durchgeführt. Danach ist die Verzweigung direkt. Das Programm läuft also nicht schneller, wenn häufig benutzte Unterprogramme an den Beginn des Programms gelegt werden.

LOGISCHE STEUERUNG

- **Die Fähigkeiten der Anweisung IF nutzen.** Durch die Benutzung von AND und OR und der Angabe ELSE können mehrere Anweisungen IF und zusätzlicher 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 häufigsten auftretende Bedingung zuerst getestet wird. Dadurch werden unnötige Tests vermieden, z.B. wenn eine Dateneingabedatei für 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
```

  Würden 100 Gruppen mit jeweils zehn Transaktionen pro Gruppe vorliegen, würden dadurch, daß der Test der Transaktionen an den Anfang der Liste gelegt wird, 1800 IF-Anweisungen weniger ausgeführt.

Im folgenden wird ein weiteres Beispiel gezeigt, in dem Anweisungen IF so angeordnet werden, daß so wenig Tests wie möglich 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 Zähler** in FOR...NEXT -Schleifen benutzen, wo dies möglich ist. Arithmetik mit ganzen Zahlen geht schneller als Arithmetik mit Zahlen einfacher und doppelter Genauigkeit.

- **Die Variable in der Anweisung NEXT weglassen,** wo dies möglich ist. Wird eine Variable eingefügt, benötigt BASIC ein wenig Zeit, um zu prüfen, ob sie in Ordnung ist. Es kann nötig sein, eine Variable in der Anweisung NEXT anzufügen, wenn aus geschachtelten Schleifen heraus verzweigt wird. Siehe unter Anweisungen FOR und NEXT in Kapitel 4.

- **FOR...NEXT Schleifen** statt Kombinationen aus Anweisungen 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
```

- **Unnötigen Code aus Schleifen entfernen.** Das sind Anweisungen, die die Schleife nicht beeinflussen, sowie nicht ausführbare 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 nötig, den Wert von A jedesmal zu berechnen, wenn die Schleife verarbeitet wird, da in der Schleife der Wert von A nie geändert wird.

Im nächsten Beispiel wird eine nicht ausführbare 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
```

Unter "Tips für den Durchsatz" in Anhang B befinden sich auch einige Tips, die sich auf Diskettendateien beziehen.

Anhang J. Glossar

Hier werden technische Ausdrücke erklärt, die in dieser BASIC-Broschüre vorkommen.

Abfangen: Eine Reihe von Bedingungen, die ein Ereignis beschreiben, das abgefangen werden soll, und die Aktion, die nach dem abfangen ausgeführt werden soll.

Abschneiden: Die endenden Elemente einer Zeichenkette abschneiden oder Abschneiden der Kommastellen hinter einer Zahl.

Absolute Koordinatenform: In der Grafik bedeutet dies die Position eines Punktes, abhängig vom Ursprung des Koordinatensystems.

Adresse: 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. Dies Punkte sind zusammen eine endliche Anzahl und bilden ein diskretes Gitter über dem Bildschirm.

Aktive Seite: Beim Farb-/Grafikbildschirmanschluß der Bildschirmpuffer, der Informationen zum Bilschirm 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 für die Lösung 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: Ein Mechanismus, um Teile anzuschließen.

Anweisung: Ein Ausdruck, der eine Operation beschreiben oder spezifizieren kann und im Sinne der BASIC-Programmiersprache komplett ist.

Anwendungsprogramm: Ein Programm, das von oder für 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 übergeben wird.

Arithmetischer Überlauf: Siehe Überlauf.

ASCII: American National Standard Code for Information Interchange. Der Standardcode, der benutzt wird, um Informationen zwischen Datenverarbeitungssystemen und den zugehörigen Einheiten auszutauschen. Eine ASCII-Datei ist eine Textdatei, die Zeichen enthält, die im ASCII-Code dargestellt sind.

Asynchron: Ohne reguläre Zeitbeziehung; nicht vorhersehbar in Bezug auf die Ausführung von Programminstruktionen.

Attribut: Eine Eigenheit oder Charakteristik für eine oder mehrere Angaben.

Auffüllen: Ein Block wird mit Scheindaten aufgefüllt, normalerweise mit Nullen oder Leerstellen.

Auflösung: In Computergrafiken das Maß der Schärfe eines Bildes, ausgedrückt durch die Anzahl der Zeilen pro Längeneinheiten in diesem Bereich.

Aufruf (Invoke): Eine Prozedur an einem ihrer Eingangspunkte aufrufen.

Ausführen: Eine Instruktion oder ein Computerprogramm ablaufen lassen.

Baud: Gibt die Anzahl von Signalen pro Sekunde an.

Bedingungen: Eine Bedingung, die eine Programmunterbrechung verursachen könnte. 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.

Bereinigen: Wenn BASIC Zeichenkettenbereiche verdichtet, indem es alle nützlichen Daten sammelt und unbenutzte Bereiche des Hauptspeichers freimacht, die vorher für Zeichenketten benutzt wurden.

Betriebssystem: Software, die die Ausführung von Programmen steuert. Bezieht sich auf DOS.

Binär: Bezieht sich auf eine Bedingung, die aus zwei Werten bestehen kann. Siehe auch Nummernsystem zur Basis 2.

Bit: Eine Binärziffer.

Blinken: Reguläres Ändern der Intensität eines Zeichens auf dem Bildschirm.

Bool'scher Wert: Ein numerischer Wert, der als "wahr" (falls er nicht Null ist) oder "falsch" (falls er Null ist) interpretiert wird.

bps: Bits pro Sekunde.

Byte: Enthält die binäre Darstellung eines Zeichens und besteht aus acht Bits.

Call: Um ein Programm oder ein Unterprogramm aufzurufen, gewöhnlich durch die Angabe der Eingangsbestimmungen und Verzweigung zu seinem 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 zusammenhängender Sätze, 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 Übertragung und das Empfangen von Daten.

Datenstation: Einheit, gewöhnlich ausgerüstet mit einer Tastatur und einem Bildschirm, der fähig ist, Informationen zu senden und zu empfangen.

Diagnostik: Hilft bei der Auffindung und Eingrenzung einer Funktion, die falsch ist, oder eines Fehlers.

DOS: Disk Operating System. In dieser Broschüre bezieht es sich nur auf das IBM Personal Computer Disk Operating System (Betriebssystem).

Duplex: In der Datenfernverarbeitung bezieht es sich auf eine unabhängige simultane Zweiwege-Übertragung in beide Richtungen, das gleiche wie Vollduplex.

Dynamisch: Passiert zur Zeit der Ausführung.

Echo: Reflektiert empfangene Daten zum Sender. Beispiel: Die Antwort auf das Betätigen einer Taste ist normalerweise die Anzeige eines Zeichens am Bildschirm.

Element: Ein Teil eines Satzes; im besonderen eine Angabe in einem Bereich.

Ereignis: Im erweiterten BASIC des IBM Personalcomputers bezieht es sich besonders auf Ereignisse, die durch COM(n), KEY(n), PEN und STRIG(n) getestet werden.

Erweiterung: Ein fortlaufender Platz auf einer Diskette, der für eine bestimmte Datei belegt oder reserviert 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.

Feste Länge: Die Länge verändert sich nicht. Dateien für wahlfreien Zugriff haben z.B. eine feste Satzlänge, d.h. jeder Satz der Datei hat die gleiche Länge.

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 nächsten Seite gebracht wird.

Fortschreiben: Normalerweise eine Stammdatei mit laufenden Informationen ändern.

Führend: Führende Nullen oder führende Leerstellen als erster Teil z.B. einer Zeichenkette.

Funktion: Eine Prozedur, die einen Wert übergibt, der vom Wert einer oder mehrerer unabhängigen Variablen in einem spezifizierten Weg abhängt.

Funktionstaste: Eine der zehn Tasten mit den Namen F1 bis F10 auf der linken Seite der Tastatur.

Ganze Zahl: Eine der Zahlen 0, ±1, ±2, ±3, ...

Gedruckte Maschinenausgabe: Eine gedruckte Kopie einer Maschinenausgabe in einer visuellen lesbaren Form.

Genauigkeit: Die Genauigkeit der Zahlendarstellung. Für eine Maschine kann dies genau angegeben werden und bezieht sich auf die Größe 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 unabhängige Übertragung in eine Richtung zu einer Zeit.

Hauptspeicher: Speicher den man an jeder gewünschten 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 Priorität, die arithmetischen oder logischen Operationen zugeordnet ist, die ausgeführt werden sollen.

Hintergrund: Der Bereich, der das Subjekt umgibt, in diesem Falle den Teil des Bildschirms, der ein Zeichen umgibt.

Implizite Definition: Das Festsetzen einer Dimension für einen Bereich, ohne ihn 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 Bereich identifiziert.

Inhaltsverzeichnis: Eine Tabelle von Angaben, die sich auf bestimmte Daten beziehen. Das Inhaltsverzeichnis der Diskette enthält z.B. die Namen der Dateien auf der Diskette sowie Informationen, die dem DOS sagen, wo die Datei auf der Diskette zu finden ist.

Initialisieren: Zum Setzen von Zählern, 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.

K: Auf die Hauptspeicherkapazität bezogen bedeutet dies 2^{10} oder 1024 in dezimaler Darstellung.

Kanal: Ein Pfad, auf dem Signale gesendet werden können, z.B. ein Datenkanal oder ein Ausgabekanal.

Kennzeichen: Eine der verschiedenen Arten von Indikatoren, die für eine Identifikation benutzt werden, z.B. ein Zeichen, das das Stattfinden 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 können.

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.

Konstante: Ein fester Wert oder eine Datenangabe.

Koordinaten: Zahlen, die den Platz auf dem Bildschirm identifizieren.

Kopfsatz: Ein Satz, der allgemeine oder konstante oder identifizierende Information über eine Anzahl von Sätzen enthält, die folgen.

Korrigieren: Zur Eingabe, zum Ändern und zum Löschen von Daten.

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 lädt.

Leerstellen: Ein Teil eines Datenmediums, in dem kein Zeichen gespeichert ist.

Lichtstift: Eine lichtsensitive Einheit, die benutzt wird, um einen Platz auf dem Bildschirm dadurch auszuwählen, daß man auf dem Bildschirm mit dem Lichtstift darauf deutet.

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}; 1.048.576 in dezimaler Schreibweise.

Mantisse: Für eine Zahl, die in Gleitkommadarstellung ausgedrückt wird, die Zahl, die nicht der Exponent ist.

Maschinenunendlichkeit: Die größte Zahl, die im internen Computerformat dargestellt werden kann.

Maske: Eine Zeichenschablone, die benutzt wird, um die Beibehaltung oder Eliminierung einer anderen Zeichenschablone zu steuern.

Matrix: Ein Bereich aus zwei oder mehr Dimensionen.

Matrixdrucker: Ein Drucker, bei dem jedes Zeichen durch eine Punktschablone dargestellt ist.

Menü: Eine Liste der verfügbaren Operationen. Man wählt die gewünschte 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 für ihre Benutzung in der Darstellung von Daten.

Oktal: Das Zahlensystem zur Basis 8.

Operand: Das was verarbeitet wird.

Parameter: Ein Name in einer Prozedur, der benutzt wird, um ein Argument anzusprechen, das zu dieser Prozedur übertragen wurde.

Paritätsprüfung: Eine Technik zum Testen übertragener Daten. Normalerweise wird eine Binärziffer angehängt, um die Summe aller Ziffern immer gerade (gerade Parität) oder immer ungerade (ungerade Parität) zu machen.

Platz: Eine Adresse, an der Daten gespeichert werden können.

Port: Ein Anschlußpunkt für Datenein- oder ausgabe.

Position: In einer Zeichenkette jeder Satz, der durch ein Zeichen besetzt werden und durch eine Nummer identifiziert werden kann.

Positionsanzeiger: Eine bewegliche Markierung, die benutzt wird, um einen Platz auf dem Bildschirm anzuzeigen.

Puffer: Ein Bereich im Hauptspeicher, der benutzt wird, um eine Differenz in der Übertragungsrate von Daten oder der Zeit für das Auftreten von Ereignissen zu kompensieren, wenn Daten von einer Einheit zu einer anderen übertragen werden. Normalerweise bezieht er sich auf einen Bereich, der für Ein-/Ausgabeoperationen reserviert ist, aus dem Daten gelesen oder in den Daten geschrieben werden.

Read-only: Ein Typ des Datenzugriffs, der nur Lesen aber kein Ändern erlaubt.

Relative Koordinaten: In der Grafik sind dies Werte, die den Platz eines Punktes identifizieren, indem die Entfernung zu einem anderen Punkt angegeben wird.

Relativzeiger: 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.

Rekursiv: Ein Prozeß, in dem jeder Schritt das Ergebnis früherer Schritte benutzt, wie z.B. eine Funktion, die sich selbst aufruft.

Reserviertes Wort: Ein Wort, das in BASIC für einen speziellen Zweck difiniert ist und das nicht als Variablennamen benutzt werden darf.

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 für Teil prüfen.

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 ausgeführt werden kann, solange eine bestimmte Bedingung wahr ist.

Schlüsselwort: Eines der vordefinierten Wörter einer Programmiersprache; ein reserviertes Wort.

Schnittstelle: Eine gemeinsame Grenze.

Schreiben: Daten in einer Speichereinheit oder auf einem Datenmedium aufzeichnen.

Schreibkopfrücklaufzeichen (CR): Ein Zeichen, durch das die Druck- oder Bildschirmanzeigeposition an die erste Position derselben Zeile zurückgesetzt wird.

Schrittweite: Ein Wert, der benutzt wird, um einen Zähler zu verändern.

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 unabhängig angezeigt und/oder geschrieben werden kann.

Sequentieller Zugriff: Eine Zugriffsmethode, bei der Sätze in der gleichen Reihenfolge abgerufen werden, in der sie geschrieben wurden. Jeder neue Zugriff auf die Datei bezieht sich auf den nächsten 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 Bereich ist.

Speicher: Eine Einheit oder ein Teil einer Einheit, der Daten enthalten kann. Hauptspeicher.

Spielpult: Ein Hebel, der in alle Richtungen bewegt und als Lokalisierungseinheit benutzt werden kann.

Standardannahme: Ein Wert oder eine Auswahl, die angenommen wird, wenn nichts angegeben wurde.

Stapeln (Stack): Eine Methode, Daten temporär 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, verändert oder stoppt. Eine Steueroperation ist eine Aktion, die das Aufnehmen, die Verarbeitung, Übertragung oder Interpretation von Daten beeinflußt, z.B. Schreibkopfrücklauf oder Übertragungsende.

Stopp-Bit: Ein Signal, das einem Zeichen oder Block folgt und die empfangende Einheit darauf vorbereitet, das nächste Zeichen oder den nächsten Block zu empfangen.

Syntax: Die Regeln für die Struktur einer Sprache.

Tabelle: Eine Anordnung von Daten in Zeilen und Spalten.

Taktgeber: Eine Einheit, die periodisch Signale erzeugt, die für die Synchronisation benutzt werden.

Testen: Fehler in einem Programm auffinden und entfernen.

Trennzeichen: Ein Zeichen, das Wörter oder Werte in einer Eingabezeile gruppiert oder trennt.

Überlagerung: Benutzung derselben Bereiche des Hauptspeichers für verschiedene Teile eines Programms zu verschiedenen Zeiten.

Überlauf: Das Ergebnis einer Operation überschreitet die Kapazität einer angegebenen Speichereinheit.

Überschreiben: Einen Bereich des Hauptspeichers beschreiben, wobei die Daten, die vorher gespeichert waren, zerstört werden.

Umwandlungsprogramm: Programm, das den Programmcode einer Programmiersprache in den Maschinencode des Computers umsetzt. Ein Compiler ist ein Umwandlungsprogramm für höhere Programmiersprachen wie BASIC, FORTRAN. Ein Assembler ist ein Umwandlungsprogramm für 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 für allgemeine oder häufige Benutzung bestimmt sind.

Variable: Eine Quantität, die irgendeinen Wert eines gegebenen Satzes von Werten annehmen kann.

Variabel langer Satz: Ein Satz, der eine Länge unabhängig von der Länge anderer Sätze 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 eindimensionaler Bereich.

Verdichtung: Eine Datenanordnung, die einen minimalen Platz benötigt.

Verkettung: Durch diese Operationen werden zwei Zeichenketten in der angegebenen Reihenfolge verbunden und bilden eine einzelne Zeichenkette mit einer Länge gleich der Summe der Länge der zwei Zeichenketten.

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.

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.

Zeichenumsetzung: Eine Technik, Daten in eine gewünschte Form umzuwandeln, wenn sie nicht in dieser Form eingegeben werden. Zum Beispiel können 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 zurückgegangen wird. Für Eingabe bedeutet sie eine Zeichenkette, die vom System als ein Eingabeblock angenommen wird; z.B. alle Zeichen, die eingegeben werden, bevor die Eingabetaste betätigt 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 übertragen.

Zeilenvorschub (LF): Ein Zeichen, durch das die Druck- oder Anzeigeposition auf die gleiche Position der nächsten Zeile gebracht wird.

Zentralsystem: Der primäre oder steuernde Computer in einer Installation mehrerer Computer.

Zugriffsmethode: Eine Technik, die benutzt wird, um einen speziellen logischen 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 binären Zahlensystem.

Zyklische Adreßfolge: Die Technik, Angaben anzuzeigen, deren Koordinaten außerhalb des Bildschirmbereichs liegen.

IBM United Kingdom International Products Limited,
PO Box 41, North Harbour, Portsmouth PO6 3AU, England

Printed in Great Britain by Collins, Glasgow.