FRANZIS

FSS

SOFTWARE SERVICE

FRANZIS SOFTWARE SERVICE

Hebas

CP/M-Basic für Z-80

Franzis Verlag, Karlstraße 37–41, 8000 München 2

Franzis'

H E B A S

Ein DISK-BASIC für CP/M-Z80-Systeme

(von NEIL COLVIN 1978)

Befehle und Erläuterungen der erweiterten Version

von Dr. Hehl Hans

Inhaltsverzeichnis: Seite:

IV. BASIC-BEFEHLE:

Alphabetische Liste der BASIC-Befehle

Die nach einigen Befehlen angegebene Klammer soll darauf hinweisen, daß dem BASIC-Befehl ein Ausdruck in Klammern folgen muß (sonst Syntaxfehler).

ABS(	INT(	SGN(
AND	INTERRUPT	SIN(
ASC(	KILL	SPACE$(
ATN(	LEFT$(	SPC(
AUTO	LEN(	SQR(
BYE	LET	STEP
BYTE	LIST	STOP
BYTE(	LOAD	STR$(
BYTE$(	LOADGO	STRING$(
BYTEPOLL(	LOC(	TAB(

```
CALL                    LOF(                    TAN(
CHR$(                   LOG(                    THEN
CLEAR                   LOG10(                  TO
CLOSE                   LOOKUP                  TRACE
CONT                    LVAR                    USING
COPY                    MAT
COS(                    MERGE                   VAL(
DATA                    MID$(                   VARPTR(
DATE$                   MOD                     WAIT
DEF                     NEW                     WRITE
DELETE                  NEXT                    XOR
DIM                     NOT                     !  PRINT USING
DIR                     ON                      ^  EXPONENT
EDIT                    OPEN                    \  GANZZAHLDIVISION
EE (2.7182818285)       OPTION                  *  MULTIPLIKATION
ELSE                    OR                      +  PLUS
END                     OUT                     -  MINUS
EOF                     OUTBYTE                 /  DIVISION
EOF(                    PEEK(                   ?  PRINT
EQV                     PI (3.1415926536)       <  KLEINER
ERASE                   POKE                    =  GLEICH
ERL                     POS(                    >  GRöSSER
ERR                     PRECISION               '  REM
ERR(                    PRINT
ERROR                   PRIVACY
EXCHANGE                RANDOMIZE
EXIT                    READ
EXP(                    READ(
FIND                    REM
FIX$(                   RENAME
FIX(                    RENUMBER
FN                      REPLACE
FNEND                   RESAVE
FNRETURN                RESET
FRE(                    RESTORE
FOR                     RESUME
GOSUB                   RETURN
GOTO                    RIGHT$(
HEX$(                   RINGBELL
IF                      RND
IMP                     RND(
INP(                    RUN
INPUT                   SAVE
INPUT LINE              SETLOC
INSTR(
```

Fehlermeldungen:

NEXT WITHOUT FOR	1	NEXT ohne FOR
SYNTAX ERROR	2	Syntaxfehler
RETURN WITHOUT GOSUB	3	RETURN ohne GOSUB
OUT OF DATA	4	keine Daten
ILLEGAL FUNCTION	5	nicht erlaubt
ARITHMETIC OVERFLOW	6	Ergebnis zu gross
OUT OF MEMORY	7	kein Speicherplatz
UNDEFINED STATEMENT	8	Sprungziel fehlt
SUBSCRIBT OUT OF RANGE	9	DIM-Befehl falsch oder fehlt
RE-DIMENSIONED ARRAY	10	DIM-Befehl doppelt
CAN'T DIVIDE BY ZERO	11	durch 0 teilen ?
ILLEGAL DIRECT	12	kein Direktbefehl
TYPE MIS-MATCH	13	Datentypen (Zahl/String) vermischt

NO STRING SPACE	14	kein Stringplatz
STRING TOO LONG	15	String zu lang
EXPRESSION TOO COMPLEX	16	Ausdruck zu schwierig
CAN'T CONTINUE	17	CONT geht nicht
UNDEFINED USER FN CALL	18	USER FN CALL fehlt
ILLEGAL EOF	19	Dateiende falsch
BAD STATEMENT NUMBER	20	Zeilen-Nr. falsch
RECOVERED	21	BASIC verfügbar
FNRETURN WITHOUT FN CALL	22	FNRETURN ohne FN CALL
MISSING STATEMENT NUMBER	23	Zeilen-Nr. fehlt
RECORD TOO LARGE	24	RECORD zu lang
UNDEFINED MATRIX	25	MATRIX nicht definiert
INVALID UNIT NUMBER	26	Kanal falsch
RESUME WITHOUT ERROR	27	RESUME ohne ERROR
DIRECTORY FULL	28	Inhaltsverzeichnis voll
EXTENSION ERROR	29	Extension falsch
NO DISK SPACE	30	DISK voll
INPUT FILE NOT FOUND	31	Datei nicht gefunden
DUPLICATE OUTPUT FILE	32	Datei existiert
OUTPUT CLOSE ERROR	33	Datei geschlossen
INVALID OPEN TYPE	34	OPEN-Modus falsch
INVALID FILE ID	35	Dateiname falsch
INVALID OPTION	36	OPTION falsch
NOT OPEN INPUT	37	INPUT nicht offen
NOT OPEN OUTPUT	38	OUTPUT nicht offen
NOT RANDOM DEVICE	39	Wahlzugriff geht nicht
FILE NOT OPEN	40	Datei nicht offen
NOT BINARY UNIT	41	keine Binäreinheit
NO FILE SPACE	42	kein Dateiplatz
INVALID RECORD NUMBER	43	RECORD-Nr. falsch
DISK PROTECTION VIOLATION	44	DISK geschützt
EXIT WITHOUT FOR	45	EXIT ohne FOR
NOT AVAILABLE	46	nicht verfügbar
NOT IMPLEMENTED	47	nicht vorhanden
UNKNOWN ERROR	0/48	Fehler unbekannt

Systemfehlermeldungen:

INVALID INPUT	* Falsche Eingabe*
EXTRA IGNORED	* Komma falsch *
BREAK	** HALT **
BREAK § Line ZE	** HALT ** Zeile ZE
FILE NOT FOUND	nicht gefunden
UNDEFINED STATEMENT	* Sprungziel fehlt *

IOBYTE-Einstellungen

Abkürzungen:

B = Batch C = Console L = List-Device P = Punch
R = Reader T = Terminal U = Eigenbau V = Video

Nach Start des Monitors sind die Einstellungen C=T, R=T, P=T und L=T vorgegeben.

Das IOBYTE wird beim mc-Monitor in der Speicherstelle F101 abgelegt. Die möglichen Zuordnungen besitzen folgende dezimale, sedezimale und binäre Werte.

C = T	0	0	0000 0000
C = V	1	1	0000 0001
C = B	2	2	0000 0010
C = U	3	3	0000 0011
R = T	0	0	0000 0000
R = P	4	4	0000 0100
R = C	8	8	0000 1000
R = U	12	C	0000 1100
P = T	0	0	0000 0000
P = P	16	10	0001 0000
P = C	32	20	0010 0000
P = U	48	30	0011 0000
L = T	0	0	0000 0000
L = V	64	40	0100 0000
L = L	128	80	1000 0000
L = V	192	C0	1100 0000

Zuweisungen für HEBAS (mit A-Kommando des Monitors):

A L = L Drucker auf Kanal Nr. 2

A P = P Drucker auf Kanal Nr. 4

A P = C Konsole auf Kanal Nr. 3

Vorwort:

Der BASIC-Interpreter HEBAS läuft auf einem Standard-CP/M-Computer mit der Z-80 CPU und 64 KByte Speicher, sowie deutschem Zeichensatz mit Umlauten . Als Monitor findet der mc-Monitor von R.-D. Klein Verwendung. Version 3.2 oder 3.4.
Der Interpreter belegt im Speicher den Platz von 100h bis 47FFh, wobei Basic-Programme den Interpreter ab 45D7h überschreiben und der Kaltstartteil des Interpreters somit nicht mehr zur Verfügung steht.

Im Handbuch werden Sedezimalzahlen ("Hexzahlen") durch ein nachgestelltes "h" gekennzeichnet, soweit in der Sedezimalzahl keine Buchstaben enthalten sind und eine Verwechslung mit Dezimalzahlen möglich ist. Basic-Befehle werden im Handbuch groß geschrieben (LOAD, SAVE), dürfen jedoch beliebig als Groß- oder Kleinbuchstaben eingegeben werden. Drücken der RETURN-Taste wird durch die Kurzfassung <CR> beschrieben, die Kontroll-Funktionen werden ähnlich abgekürzt, z.B. <CTRL-E>. Falsche Schreibweise der Befehle führt zur Meldung "Syntaxfehler".

I. Start des BASIC-Interpreters HEBAS.COM

Durch den mc-Monitor sind beim Start folgende IOBYTE-Belegungen vorgegeben: C=T, R=T, P=T, L=T. Ist ein Drucker am System angeschlossen, so muß vor dem CP/M-Start "AL=L" eingegeben werden. Nach dem Laden und Starten des CP/M-Betriebssystems (Version 2.2) und des Basic-Interpreters mit dem Programmnamen HEBAS.COM wird nach der Speichergrenze gefragt. Eingabe von <CR> reserviert 38144 Bytes Speicherplatz für Programme und zusätzlich 100 Bytes für Zeichenketten. Eingabe einer Dezimal- oder einer Sedezimalzahl (mit Kennzeichnung "&") begrenzt den Speicher, um z.B. ein Maschinenprogramm im oberen Speicherbereich zu schützen.

Beispiele:

Eingabe:	45056 <CR>	Anzeige:	26873 Bytes frei
	&B000 <CR>		26873 " "
	&DC07 <CR>		38144 " "

Bei der Eingabe von Sedezimalzahlen sind nur Großbuchstaben erlaubt (sonst Fehlermeldung "Syntaxfehler"). Bei Zahlen größer 56325 erfolgt die Fehlermeldung "kein Speicherplatz" und es kann eine neue Zahl eingegeben werden. Zahlen kleiner 18400 sollte man nicht eingeben.
Danach wird nach dem Datum gefragt. Es können bis zu 20 beliebige Zeichen eingegeben werden, die mit PRINT DATE$ abrufbar sind, z.B. 31. Dezember 1984 **. Die Eingabe von <CR> ergibt die Meldung "Fertig" ohne aktuelles Datum.

II.A) Basic-Warmstart

Wird während des Betriebes die RESET-Taste (auch BREAK-Taste genannt) betätigt, so wird aus dem Eprom der Monitor neu geladen, ohne daß der Speicher gelöscht wird. Arbeitet man nur mit dem Laufwerk A weiter, genügt die Eingabe "g100" und

der BASIC-Interpreter antwortet mit der Meldung "BASIC verfügbar". Wurde z.B. in einer Endlosschleife mit der RESET-Taste abgebrochen, erfolgt nach dem Warmstart die Meldung "BASIC verfügbar Zeile XYZ". Will man nach dem Warmstart mit Laufwerk B arbeiten, muß nach dem Monitor-Start zuerst CP/M geladen werden. Danach wird durch Aufruf eines Pseudoprogrammes CONT.COM auf Adresse 100h gesprungen und der Interpreter gestartet. Das Programm CONT.COM enthält keine Bytes und wird einfach mit dem CP/M-Befehl "SAVE 0 CONT.COM" gespeichert. Laden oder Abspeichern von Programmen mit dem Befehl LOAD oder SAVE ergibt die Fehlermeldung "kein Dateiplatz". Der Befehl CLEAR,0 (Komma eingeben!) reserviert für diese Vorgänge wieder Speicherplatz und Laden oder Abspeichern von BASIC-Programmen funktionieren wieder.

Tritt beim Betrieb des BASIC-Interpreters z.B ein Bdos Err On A: R/O auf, so meldet sich nach Eingabe von <CTRL-C> CP/M mit dem Laufwerkbuchstaben, z.B. "A>". Durch Eingabe von CONT (Meldung:BASIC verfügbar) und CLEAR,0 kann mit dem BASIC-Interpreter weiter gearbeitet werden, ohne daß ein Programmverlust eingetreten ist.

Das mit DATE$ abrufbare aktuelle Datum bleibt beim Warmstart erhalten.

Will man nachträglich das IOBYTE verändern, so kann dies auf zweierlei Art geschehen. Mittels RESET-Taste wird der Monitor gestartet. Nun kann eine andere Zuordnung getroffen und ein BASIC-Warmstart durchgeführt werden. Man kann aber auch mittels POKE-Befehl den Dezimalwert des IOBYTE in die Speicherstelle 61697 (F101h) schreiben (POKE&F101, Dezimalwert). Der Dezimalwert 128 gilt für die Einstellung AL=L, 0 für AL=T (siehe IOBYTE-Tabelle).

II.B) Kontrollzeichen:

<CTRL-E>:
Die Ausführung eines BASIC-Programmes oder eines Direktbefehles wird abgebrochen und die Meldungen "^E","** HALT ** Zeile XYZ" oder nur "** HALT **" und dann "Fertig" ausgegeben.

Die Abbruchmöglichkeit mit <CTRL-E> wird mit dem Befehl INTERRUPT Variable in einem BASIC-Programm abgeschaltet, wobei der Variablen der Wert 0 zugewiesen oder gleich INTERRUPT 0 vorgesehen wird. Der Ausstieg aus der Schleife in einer Programmzeile "15 goto 15" ist nur noch durch Neustart des Monitors (RESET-Taste) möglich. Allerdings wird die <CTRL-E>-Funktion durch Befehle wie INPUT im Programm wieder eingeschaltet, ebenso durch INTERRUPT 1.

<CTRL-S>:
Die Ausführung eines BASIC-Programmes oder eines Direktbefehles wird unterbrochen.

<CTRL-Q>:
Die Unterbrechung kann nur mit <CTRL-Q> aufgehoben werden.

<CTRL-U>:
Eine Eingabe im Programm oder als Direktbefehl wird gelöscht, solange nicht <CR> eingegeben worden war.

<CTRL-O>:
Die Ausgabe auf den Bildschirm wird im Programm oder bei einem Direktbefehl bis zur Meldung "Fertig" unterdrückt, aber nicht gestoppt.

<CTRL-R>:
Es werden alle durch die Taste <DELETE> (Rubout) eingefügten Trennzeichen ('\') und die als "Echo" angezeigten zu löschenden Zeichen entfernt und die bereinigte Zeile neu ausgegeben. Der Cursor steht am Ende dieser Zeile und es kann mit der Eingabe fortgefahren werden.

<CTRL-Z>:
Es wird die Meldung "Dateiende falsch" ausgegeben. Dieser Befehl beendet z.B. auch eine Eingabe von der Tastatur über den READER-Kanal mit LOAD#3.

III. ALLgemeine BASIC-Befehle (Direktbefehle)

Die in diesem Kapitel beschriebenen Befehle können bis auf PRIVACY als Direktbefehl verwendet werden. Außer bei RENUMBER und CONT ist auch der Einsatz in einer BASIC-Zeile möglich, aber nicht immer sinnvoll. Befehle wie LIST, COPY, DELETE, EDIT, ERROR, MERGE und NEW bewirken in einer Programmzeile nach ihrer Funktion einen Programmstop. Bei den Befehlsbeispielen sind alle angegebenen Zeichen einzutippen, auch die Kommas oder Anführungszeichen.

Beispielprogramm:
```
10 DIR"B:*.*"
20 TRACE 1
30 REM - 1000 BASIC-Programm
1010 TRACE 0
1020 RESAVE "B:TEST"
1030 LIST
```

Der Befehl LET wurde in die Beschreibung nicht aufgenommen, da er bei der Zuweisung eines Wertes zu einer Variablen nicht benötigt wird und auch keinen Geschwindigkeitsvorteil bei der Bearbeitung eines Programmes bietet.

1) AUTO:

Befehlsform: AUTO Zeilennummer + , Zeilenabstand

Dieser Befehl gibt automatisch Zeilennummern für Programmzeilen aus. Erste Zeilennummer und Differenz zwischen den Zeilennummern können zusätzlich angegeben werden.
In Adresse 7F3h und B00h sind als Standardwerte 0Ah für Zeilennummer und Differenz enthalten.

a) AUTO <CR>

Es wird mit Zeilennummer 10 begonnen. Nach jedem nächsten <CR> wird zur Zeilennummer die Zahl 10 addiert und diese ausgegeben.

b) AUTO 100 <CR>

Es wird mit Zeilennummer 100 begonnen und nach jedem <CR> 10 dazu addiert.

c) AUTO 10,100 <CR>

Es wird mit Zeilennummer 10 begonnen, aber nach jedem <CR> wird die Zahl 100 dazu addiert. Wird die Zeilennummer 65535 erreicht, so erfolgt die Fehlermeldung "Zeilen-Nr. falsch".

d) AUTO 10+,100 <CR>

Es wird der Zeilenabstand zum Anfangswert addiert und mit der Summe begonnen.

e) Existiert schon eine durch AUTO erhaltene Zeilennummer, wird diese mit einem vorangestelltem + Zeichen ausgegeben. <CR> läßt die dazugehörige Programmzeile unverändert. Sie kann durch eine neue überschrieben werden.

f) <CTRL-E> beendet die automatische Zeilennummerierung. Wird sie nachher mit AUTO. (Punkt eingeben) wieder begonnen, so werden die Zeilennummern beginnend mit der letzten Nummer weitergeführt, wobei die Differenz wieder angegeben werden kann, z.B. AUTO .,20 (Differenz 20).

2) BYE:

Nach einem RESET wird der Basic-Interpreter verlassen und CP/M steht zur Verfügung. Alle Kanäle, über die Daten ausgegegeben wurden, müssen geschlossen sein, sonst können Daten verloren gehen.

3) CALL:

Befehlsform: CALL Adresse, Wert 1, Wert 2, ..., Wert n

Mit diesem Befehl wird zu der angegebenen Adresse in Maschinenspracheroutinen gesprungen. Dabei können mit dem Befehl die angegebenen Werte übertragen werden. Sie werden auf dem Stapel abgelegt und sind mit dem Z-80-Befehl pop zugänglich. Das BC-Register enthält die Anzahl n der übertragenen Werte, diese ist nur durch den Speicher begrenzt. Im HL-Register ist die Rücksprungadresse angegeben. Wird diese auf den Stapel gebracht, kann mit ret zurückgesprungen werden.

a) CALL 2024 <CR>

Es wird in die Interpreterroutine gesprungen und der Text "Fertig" ausgegeben. Die Adresse kann auch sedezimal angegeben werden, z.B. CALL &7E8. CALL &337F gibt ein FFh an die Konsole aus, d.h. der Bildschirm wird gelöscht.

b) CALL &8000,45,3 <CR>
Ab Adresse 32768 (8000h) muß sich ein Maschinenprogramm befinden, dem die Werte 45 und 3 übergeben werden. Register BC enthält 00 02, auf dem Stapel befinden sich 03 00 2D 00.

c) CALL 0 <CR>
BASIC wird verlassen und es meldet sich das Betriebssystem mit dem Laufwerkbuchstaben, z.B. "A>". Ein Warmstart von BASIC ist mit dem Pseudoprogramm CONT.COM möglich.

4) CLEAR:

Befehlsform: Zeilennummer CLEAR Byteanzahl,Kanalanzahl

Es können Variablenwerte gelöscht, Platz für Zeichenketten reserviert und mehrere, zusätzliche Kanäle für Ein- oder Ausgabebetrieb gleichzeitig geöffnet werden (siehe Abschnitt VI). Mit dem Befehl FRE(X) bzw. FRE(X$) wird der noch freie Speicherplatz ermittelt. X ist eine Hilfsvariable und enthält keinen Wert <siehe den Basic-Befehl FRE(X)>. Noch offene Kanäle werden ohne ordnungsgemäßen Abschluß getrennt.

a) CLEAR <CR>

Dieser Befehl löscht alle Variablen im Speicher. In einem

Programm muß dieser Befehl am Anfang stehen, sonst werden nachher eingegebene Werte nicht mehr berücksichtigt. Es sind dabei 100 Bytes für Zeichenketten und 182 Bytes für einen Kanal für Ein- oder Ausgabebetrieb reserviert.

b) CLEAR 10000 <CR>

Es werden nun 10000 Bytes für Zeichenketten reserviert. PRINT FRE(X) <CR> ergibt die Anzeige "28246" (Der Wert ist um 102 Bytes größer, da die Standardreservierung für Zeichenketten entfällt); Print FRE(X$) <CR> ergibt den Wert 10000. Die größte mit CLEAR zu reservierende Byteanzahl beträgt 38100. Dies ist wenig sinnvoll, da für ein Programm kein Platz mehr vorhanden ist (FRE(X) ergibt dann 146 Bytes). Bei noch größeren Zahlen erscheint die Fehlermeldung "kein Speicherplatz", bei Zahlen über 65535 die Meldung "nicht erlaubt".
Wurde in einem Programm mit dem CLEAR-Befehl zu wenig Speicherplatz für Zeichenketten reserviert, kommt die Fehlermeldung "kein Stringplatz". Es kann der reservierte Platz auch wieder verkleinert werden. CLEAR 0 beseitigt die Standardreservierung von 100 Bytes für Zeichenketten.

c) CLEAR 1000,2 <CR>

Es werden 1000 Bytes für Zeichenketten und zusätzlich zum Standardkanal noch Speicherplatz für zwei weitere Kanäle für Ein- oder Ausgabebetrieb reserviert. Jeder Kanal braucht weitere 182 Bytes, wie mit FRE(X) beobachtet werden kann.

d) CLEAR,1 <CR>

Es werden alle Variablen gelöscht und für einen weiteren Kanal Speicherplatz reserviert (siehe II.A Warmstart). Bei ausreichendem Speicherplatz könnten theoretisch bis zu 255 Kanäle reserviert werden.

5) CLOSE:

Befehlsform: CLOSE # Kanal, # Kanal, usw.

Der Befehl beendet die Ein- oder Ausgabe über geöffnete Kanäle und schließt diese. Nähere Einzelheiten sind bei den Dateibefehlen beschrieben.

a) CLOSE #0 <CR>

Durch Ausgabe von 0Ch wird der Bildschirm gelöscht, der Kanal bleibt offen.

b) Wird ein Kanal geschlossen, der nicht geöffnet war, ergibt dies die Fehlermeldung "Datei nicht offen".

Sind noch Kanäle offen und wird ein BASIC-Programm mit <CTRL-E> abgebrochen, so können Programme nicht abgespeichert werden (Fehlermeldung: kein Speicherplatz), bzw. nicht geladen werden (Fehlermeldung: kein Dateiplatz). Mit CLOSE werden die Kanäle geschlossen und die Befehle SAVE bzw. RESAVE funktionieren einwandfrei.

6) CONTINUE oder CONT

Wird <CTRL-E> während eines Programmablaufes eingegeben oder kommt der Interpreter zu einem STOP-Befehl in einer Programmzeile, so hält das Programm mit der Meldung "** HALT ** Zeile XYZ" an, wobei XYZ die Zeilennummer angibt, in der abgebrochen wurde.
Nun kann man mit dem Befehl LVAR die Variablenwerte anschauen, mit LVAR#2 ausdrucken oder die Werte ändern, z.B a = 15 oder a$ = "Text".
Anschließend wird mit dem Befehl CONT das Programm unter Berücksichtigung der eventuell geänderten Werte fortgesetzt. So kann man zu Testzwecken die Variable des Schleifenzählers einer FOR-NEXT-Schleife verändern.
Nach einer Programmänderung oder nach Auftreten eines Fehlers ergibt der Befehl CONT die Meldung "CONT geht nicht".
Nach Abbruch mit <CTRL-E> bei einem INPUT-Befehl wird die Eingabeaufforderung wiederholt.

7) COPY:

Befehlsform:

COPY neue Zeilen-Nr.,Zeilenabstand = alte Zeilennummer
oder
COPY neue Zeilen-Nr.,Zeilenabstand = Zeilenbereich

Der Befehl verschiebt oder dupliziert einzelne Programmzeilen oder Programmteile. Dabei werden die im Zeilenbereich angegebenen Zeilen beginnend mit der neuen Zeilennummer mit dem angegebenen Zeilenabstand durchnummeriert und kopiert. Vorhandene Programmzeilen werden nicht überschrieben, ergeben die Fehlermeldung "nicht erlaubt" und der COPY-Befehl wird nicht ausgeführt.

a) COPY 200 = 150 <CR>

Die Programmzeile 150 wird kopiert und als Programmzeile 200 abgelegt. Auch COPY 25 = 150 ist möglich.

b) COPY 200 = 50 - 70 <CR>

Die Zeilen 50 bis 70 werden beginnend mit 200 im Zehnerschritt durchnummeriert und ab 200 abgelegt.
Wird aus einem höheren Zeilenbereich in einen niedrigeren Bereich kopiert, so ist darauf zu achten, daß genügend Platz für die einzufügenden Zeilen vorhanden ist, sonst macht die Fehlermeldung "Zeilen-Nr. falsch" darauf aufmerksam.

c) COPY 101,1 = 500 - 550 <CR>

Die Programmzeilen 500 bis 550 werden direkt hintereinander nach der Programmzeile 100 eingefügt (Zeilenabstand = 1).

---

8) DELETE:

Befehlsform: DELETE Zeilennummer oder Zeilenbereich

Eine Programmzeile oder ein Zeilenbereich werden gelöscht.

a) DELETE 100 <CR>

Die Zeile 100 wird gelöscht. Dies geht jedoch schneller durch Eingabe der Zeilennummer mit anschließendem <CR>.

b) DELETE 200-300 <CR>

Es werden die Zeilen 200 bis 300 einschließlich gelöscht.

9) DIR:

Befehlsform:
DIR #Kanal §Satznummer "Laufwerk : Dateiname . Dateizusatz"

Das Inhaltsverzeichnis der Diskette wird auf den angebenen Kanal ausgegeben. Wie bei dem CP/M-Befehl DIR können Dateiname oder Dateizusatz durch ein Sternchen "*" ersetzt werden. Auch Fragezeichen sind im Dateinamen zulässig. Das Anführungszeichen (") am Befehlsende kann entfallen; fehlt es am Anfang des Befehls, ergibt dies die Fehlermeldung "Datentypen (Zahl/String) vermischt". Die Angabe einer Satznummerzahl mit vorangestelltem Klammeraffen "§" ist nur bei Dateien mit wahlfreiem Zugriff notwendig, wie im Beispiel f) beschrieben. Ein BASIC-Programm im Speicher wird nicht gelöscht.

a) DIR <CR>

Es wird das Inhaltsverzeichnis aller Dateien der Diskette in dem Laufwerk, das unter CP/M angemeldet war, am Bildschirm angezeigt. DIR#2 gibt das Verzeichnis über den Kanal Nr. 2 auf den Drucker aus.

b) DIR"B:Dat1" <CR>

Ist diese Datei im Disketteninhaltsverzeichnis vorhanden, wird der Dateiname mit Dateizusatz ausgegeben, sonst erscheint die Fehlermeldung "nicht gefunden".

c) DIR "C:*.*" <CR>

Es wird eine Liste aller Dateien der Diskette des Laufwerkes Nr. C ausgegeben.

d) DIR "B:*" <CR>

Es werden nur die Dateien aufgezeigt, die den Dateizusatz ".BAS" besitzen. Werden Dateien nicht gefunden, erscheint die Fehlermeldung "nicht gefunden".

e) DIR#20 <CR>

Das Inhaltsverzeichnis der unter CP/M angemeldeten Diskette wird in eine zuvor mit dem Befehl OPEN#20,"O","A:Test" geöffnete Datei "Test" im Laufwerk A sequentiell abgelegt.
f) DIR#20§500 <CR>

Diesen Befehl benötigt man z.B., um bei automatisierten Diskettenverwaltungsprogrammen das Inhaltsverzeichnis der Dis-

kette in eine Datei mit wahlfreiem Zugriff zu schreiben. Dies ähnelt den Möglichkeiten, die sonst nur Dienstprogramme mit UNIX-Merkmalen für Großrechner bieten. Hier wird das Inhaltsverzeichnis über den Kanal Nr.20 in eine zuvor geöffnete Datei mit wahlfreiem Zugriff (Random-Datei) als Datensatz (Record) Nr.500 geschrieben (siehe Dateibefehle).

10) EDIT:

Befehlsform: EDIT Zeilennummer oder EDIT. (Punkt eingeben)

Um Programmzeilen zu korrigieren oder zu ergänzen, wird der EDIT-Befehl verwendet. Anfangs muß man öfters in der EDIT-Kommandoliste nachschauen. Man kann natürlich bei kurzen Programmzeilen die ganze Zeile neu schreiben, was schneller und einfacher geht. Der Editor bearbeitet eine Kopie der Programmzeile in einem Puffer, so daß die eigentliche Programmzeile erst zum Schluß durch die Kopie ersetzt wird. EDIT ohne Zeilennummer oder Punkt ergibt die Fehlermeldung "nicht erlaubt".

a) EDIT 120 <CR>

Es wird die Zeilennummer 120 ausgegeben und auf einzelne EDIT-Kommandos gewartet.

b) EDIT . <CR> (Punkt eingeben)

Nach einer Fehlermeldung, bei einem Programmabbruch oder erneutem EDIT-Kommando kann die Zeilennummer durch einen Punkt ersetzt werden. Erscheint z.B. nach Eingabe von RUN die Fehlermeldung "Syntaxfehler Zeile 115", so kann nach Eingabe von EDIT . die Zeile mit den Edit-Kommandos verbessert werden.

c) EDIT-Kommandos:

Wird für n keine Zahl angegeben, gilt n = 1.

A: Neustart des Editors nach einem Fehler

nD: Löscht n Zeichen. Die gelöschten Zeichen werden mit Hochkomma (') eingerahmt und angezeigt.

E: Beendet das Editieren. Die Programmzeile wird durch die Kopie ersetzt, aber nicht angezeigt.
Es erfolgt keine "Fertig"-Meldung des Interpreters.

nFX: Suchbefehl in der Programmzeile:
Zeigt die Zeile bis zum n-ten Zeichen X und setzt den Cursor davor.

H: Löscht alles rechts vom Cursor und wartet auf Eingabe.

I: Eingabe von Zeichen ab der Cursorstellung in den Editpuffer und gleichzeitige Anzeige. <ESCAPE> beendet die Eingabe, <CR> beendet den Editor.

nKX: Löschbefehl in der Programmzeile:
Lösche alle Zeichen rechts vom Cursor bis zum n-ten Zeichen X, ohne X zu löschen.

L: Zeigt die im Puffer enthaltene Programmzeile und setzt den Cursor nach der Zeilennummer an den Anfang der Zeile.

Q: Abbruch des Editors, die Programmzeile bleibt unverändert und "Fertig" wird ausgegeben.

nR: Ersetzt n Zeichen ab dem Cursor durch die dann eingebenen Zeichen.

X: Zeigt die Zeile, bewegt den Cursor an das Zeilenende und wartet auf Zeicheneingabe (I-Modus).

<SPACE>: Die Leertaste bewegt den Cursor nach rechts, die Zeile wird bis zum Cursor angezeigt.

<CR> : beendet den Eingabemodus (I) und den Editor.

<DEL> : bewegt den Pufferzeiger nach links. Die Zeichen werden jedoch als Echo nach rechts ausgegeben (Rubout-Taste).

<ESCAPE>: beendet den Eingabemodus (I) und bleibt im Editor.

<CTRL-E>: keine Funktion beim EDIT-Befehl.

11) ERASE:

Befehlsform: ERASE " Laufwerk : Dateiname . Dateizusatz "

Es wird die angegebene Datei gelöscht. ** Achtung ** : Soll nach dem ERASE-Befehl die Diskette aus dem Laufwerk genommen werden, muß man unbedingt vor dem Herausnehmen den BASIC-Befehl RESET eingeben, damit das Inhaltsverzeichnis auf der Diskette berichtigt wird.

a) ERASE "B:DAT1.BAS" <CR>

Es wird die Datei DAT1.BAS auf der Diskette im Laufwerk B gelöscht. Falls die Datei nicht vorhanden war, wird keine Fehlermeldung "Datei nicht gefunden" ausgegeben. Ein Austausch von Dateiname oder Zusatz durch "*" ist möglich. Es werden alle Dateien mit der Bezeichnung gelöscht.

b) ERASE "DAT1" <CR>

Der Dateizusatz ".BAS" wird zum Befehl hinzugefügt und die Datei "DAT1.BAS" auf dem Laufwerk, das unter CP/M angemeldet war, gelöscht.

c) ERASE "*.*" <CR>

Dieser Befehl bedeutet den Verlust aller Dateien auf der Diskette, denn es werden alle Dateien gelöscht.

Allerdings können sie unmittelbar danach mit dem Programm UNERASE.COM vom SOFTWARE-Service (Franzis-Verlag) gerettet werden.

d) Wird mit dem ERASE-Befehl auf eine Datei zugegriffen, die nicht vorhanden ist, erfolgt keine Fehlermeldung!

12) ERROR:

Befehlsform: ERROR Nummer

Mit diesem Befehl können einerseits die Fehlermeldungen des Interpreters gelesen, andererseits beliebige Meldungen definiert und ausgegeben werden. Weitergehende Informationen siehe VI.D) Fehlerbehandlung (ON ERROR GOTO, ERR und ERL).

ERROR 15 <CR>

Es wird die Fehlermeldung "String zu lang" ausgegeben. Bei einer Zahl größer 47 erscheint die Meldung "Fehler unbekannt". Möglich sind Zahlen von 0 bis 255.

13) FIND:

Befehlsform:
FIND Trennzeichen Zeichenkette Trennzeichen , Zeilenbereich

Beliebige Zeichenketten, z.B. Variablennamen können in einem Programm ermittelt werden. Es werden die Zeilen mit der gesuchten Zeichenkette ausgegeben. Als Trennzeichen können beliebige ASCII-Zeichen im Bereich von 21h bis 40h verwendet werden mit einigen Ausnahmen: keine Zahlen, kein Komma. Wird ein als Trennzeichen verwendbares Zeichen gesucht, darf dieses natürlich nicht als Trennzeichen verwendet werden.
FIND ... ergibt die Fehlermeldung "Syntaxfehler" und muß als FIND '.' eingegeben werden.

a) FIND "DATA",100-200 <CR>

Es wird im Programm von Zeile 100 bis Zeile 200 nach der Zeichenkette DATA gesucht. Dabei ist zu beachten, daß BASIC-Befehle nach der Eingabe in Großbuchstaben umgesetzt sind und in Kleinbuchstaben nicht gefunden werden (FIND 'gosub' findet nichts).

b) FIND .Eingabe. <CR>

Es wird das Wort Eingabe im ganzen Programm gesucht.

14) FRE(X)

Befehlsform: PRINT FRE(X) oder A = FRE(X)
Es wird der für BASIC-Programme und Variablen zur Verfügung stehende Speicherplatz angegeben. Der Buchstabe in Klammern kann beliebig sein (dummy variable).

PRINT FRE(X) <CR>

Ist beim Start des Interpreters die Frage nach der Speichergrenze mit <CR> beantwortet worden, so gibt dieser Befehl PRINT FRE(X) die Zahl 38146 aus, d.h. 38146 Bytes sind verwendbar. Nach dem Laden eines Basic-Programmes wird der noch zur Verfügung stehende Speicherplatz angegeben. Die Länge eines Programmes in Byte wird durch PRINT 38146 - FRE(X) ermittelt.

15) FRE(X$):

Befehlsform: PRINT FRE(X$) oder A = FRE(X$)

Der Befehl ermittelt den für Zeichenketten verfügbaren Speicherplatz. Nach dem Start des Interpreters sind für Zeichenketten immer 100 Bytes reserviert. Durch CLEAR kann diese Reservierung verändert werden (siehe CLEAR!). Der Buchstabe X ist beliebig wählbar.

PRINT FRE(X$) <CR>

Ohne Speicherplatzreservierung durch CLEAR wird die Zahl 100 angegeben. Bei vorheriger Eingabe von CLEAR 1000 ergibt PRINT FRE(X$) den Wert 1000.

16) GOTO:

Befehlsform: GOTO Zeilennummer

Der Befehl kann wie RUN zum Start eines Programmes verwendet werden, wenn die Zeilennummer der ersten BASIC-Zeile angegeben wird. Variableninhalte werden dann nicht gelöscht. Es kann auch zu einer beliebigen Zeilennummer gesprungen werden, um dort das Programm fortzusetzen. Existiert die angegebene Zeilennummer nicht, ist die Folge ein "Sprungziel fehlt".

Der Befehl wird im Kapitel IV näher beschrieben.

17) INP: bzw. OUT:

Von BASIC aus werden mit diesem Befehl bei 8080- und Z-80-Systemen die Eingangs-Ports direkt angesprochen, wobei ein Byte gelesen und dezimal angezeigt wird. Die Ausgabe eines Bytes erfolgt mit dem Befehl OUT. Man kann so z.B. von BASIC aus die Ports der Floppy-Karte FLO1 testen.

Befehlsform: INP (Portnummer)
OUT (Portnummer) , BYTE

a) PRINT INP(50) <CR>

Es wird der Wert des Ports Nummer 50 gelesen und dezimal ausgegeben. Der Wert kann auch einer Variablen zugewiesen

werden: A = INP(&32).

b) OUT &30,15 <CR>
An das Ausgabeport 30h wird der Wert 15 gegeben.
c) OUT 52,1 <CR> OUT 48,15 <CR>

Laufwerk A (=1) wird ausgewählt und der Kopf über Spur 0 positioniert (Restore-Befehl der 5,25-Zoll-Floppy).

d) OUT &32,&AA <CR> PRINT INP(&32) <CR>

Der Wert AAh wird auf Port 50 ausgegeben und wieder zurückgelesen, der Dezimalwert 170 wird angezeigt (Test für Lese-Schreibbefehle).

18) INTERRUPT: Siehe II.B) Kontrollzeichen

19) KILL:

Befehlsform: KILL erste Variable, zweite Variable, ...

Sind im Programm für größere Variablenmengen mit dem DIM-Befehl Speicherbereiche reserviert, so kann die Reservierung wieder aufgehoben werden. Das Dollar-Zeichen kennzeichnet Variablen für Zeichenketten.

a) KILL A$,B$,C$ <CR>

Es werden die Dimensionierungen von A$,B$ und C$ aufgehoben. Eine Löschung einer nicht existierenden Dimensionierung ergibt keinen Fehler.

b) KILL A,B <CR>
Die Dimensionierung für Zahlen wird gelöscht.

20) LIST:

Befehlsform:
LIST # Kanal § Satznummer , Zeilennummer oder Zeilenbereich

Es werden BASIC-Programmzeilen auf den angegebenen Kanal ausgegeben. Ohne Kanalangabe erfolgt die Ausgabe am Bildschirm, mit LIST#2 über den Kanal Nr.2 auf den Drucker.
Mit dem Befehl OPTION kann die vorgegebene Zeilenlänge von 80 Zeichen getrennt für Bildschirm und Drucker verändert werden (siehe OPTION).
Eine Zeilennummer kann durch einen Punkt ersetzt werden. Mit <CTRL-S> stoppt die Ausgabe, mit <CTRL-Q> wird sie fortgesetzt, mit <CTRL-E> abgebrochen (Meldung: ** HALT **). <CTRL-O> verhindert die Bildschirmanzeige, am Ende wird die Systemmeldung "Fertig" ausgegeben. Wie beim Befehl DIR ist die Angabe einer Satznummerzahl mit vorangestelltem Klammeraffen "§" nur für Dateien mit wahlfreiem Zugriff notwendig (siehe Beispiel h).

a) LIST <CR>
Das Programm wird von Anfang an am Bildschirm aufgezeigt.

b) LIST 100 <CR>

Die Programmzeile 100 wird angezeigt. Auch LIST 0 ist möglich, wenn die Zeile 0 vorhanden ist.

c) LIST 130-250 <CR>
Ausgabe ab Zeilennummer 130 bis 250 einschließlich.

d) LIST -200 <CR>
Ausgabe ab der ersten Zeile bis zur Zeile 200.

e) LIST 1000- <CR>
Ausgabe ab Zeile 1000 bis zum Programmende.

f) LIST . - 200 <CR>
Ausgabe ab der zuletzt bearbeiteten Zeile bis Zeile 200.

g) LIST#15 <CR>

Es wird das gesamte BASIC-Programm in eine zuvor mit dem Befehl OPEN#15,"O","A:Test" geöffnete Datei "Test.BAS" im Laufwerk A sequentiell in ASCII-Zeichen abgelegt (siehe OPEN-Befehl). In den Programmzeilen dürfen jedoch keine Zeichen 0Ah (Linefeed) zur Gliederung des Programmes vorkommen, da sonst die nachfolgenden Zeilen nicht abgespeichert werden. Mit einem Texteditor kann man dann die Datei bearbeiten. Allerdings erfüllt der Befehl SAVE "A:TEST",A (Satzzeichen miteingeben) einen ähnlichen Effekt, wobei aber das Zeichen 1Ah als Dateiende mitabgespeichert wird.

h) LIST #20§0,150 <CR>

Die Zeile 150 wird über den Kanal Nr. 20 in eine zuvor geöffnete Datei mit wahlfreiem Zugriff als Datensatz Nr. 0 geschrieben.

21) LOAD:

Befehlsform:
LOAD # Kanal , " Laufwerk : Dateiname . Dateizusatz "

BASIC-Programmdateien werden in den Speicher geladen, wobei die Dateien auch im ASCII-Format (siehe RESAVE-Befehl) vorliegen dürfen. Wird das Laufwerk nicht angegeben, sucht der Interpreter die Datei auf der Diskette in dem Laufwerk, das unter CP/M angemeldet war. Der Zusatz .BAS und das letzte Anführungszeichen können entfallen. Fehlt das erste Anführungszeichen, kommt die Fehlermeldung "Datentypen (Zahl/String) vermischt". Alle dem LOAD-Befehl folgenden Angaben können auch durch Textvariablen ersetzt oder durch ein Sternchen (*) abgekürzt werden.

a) LOAD "ALLDRU <CR>

An den Dateinamen ALLDRU wird der Zusatz .BAS angehängt, das BASIC-Programm von dem aktuellem Laufwerk geladen und die Fehlermeldung "Datei nicht gefunden" ausgegeben, falls die Datei nicht vorhanden ist.
b) LOAD "REC* <CR>

Von der Diskette im angemeldetem Laufwerk wird das erste Programm geladen, das mit den Buchstaben "REC" beginnt.

c) LOAD"* <CR>

Von der Diskette im angemeldetem Laufwerk wird das erste Programm geladen.

c) LOAD "B:ALLDRU.COM <CR>

Es wird versucht, von dem Laufwerk B das Programm ALLDRU.COM zu laden. Liegt kein BASIC-Programm vor, fehlen natürlich die Zeilennummern (Fehlermeldung "Zeilen-Nr. fehlt").

d) LOAD D1$ + ":" + DA$ <CR>

In D1$ ist die Laufwerkangabe (ein Buchstabe), in DA$ der Dateiname mit oder ohne Zusatz enthalten. Auch die Zusammenfassung aller Angaben in einer Variablen ist erlaubt.

e) LOAD#3 <CR>

Wurde beim Einschalten des Gerätes die IOBYTE-Einstellung P=T belassen, so wird mit diesem Befehl von der Tastatur gelesen. Es können BASIC-Programmzeilen mit Zeilennummer eingegeben werden, allerdings werden die eingegebenen Zeichen nicht auf dem Bildschirm angezeigt. <CTRL-Z> beendet die Eingabe und die Meldung "Fertig" erscheint. Mit LIST wird die gesamte Eingabe ausgegeben. Mit dem Befehl LOAD#3 kann auch ein Strichcodeleser (siehe mc 3/1981) oder ein Modem bedient werden. Die Kanalnummern 2 und 4 ergeben die Fehlermeldung "OPEN-Modus falsch", die anderen außer Nummer 3 die Meldung "Syntaxfehler" (siehe auch MERGE#3).

22) LOADGO:

Befehlsform:
LOADGO " Laufwerk : Dateiname . Dateizusatz " , Startzeile

Mit diesem Befehl wird ein Basic-Programm geladen und dann gestartet, also der RUN-Befehl ausgeführt. Die Ausführungen zum LOAD-Befehl gelten hier ebenso. Allerdings kann man keine Programme, die im ASCII-Format abgespeichert wurden (siehe RESAVE), mit dem LOADGO-Befehl laden (ergibt die Fehlermeldung: Syntaxfehler).
Programme lassen sich so über ein Menue bequem laden, ohne daß man sich um die Programmlängen wie bei anderen BASIC-Interpretern kümmern muß. Variablen werden allerdings gelöscht. Sie können vorher z.B. mit dem Befehl MAT WRITE auf Diskette geschrieben werden.

23) LVAR:

Befehlsform: LVAR #Kanal §Satznummer

Die in einem Programm verwendeten Variablen und ihre Werte können auf den Bildschirm (keine Kanalangabe oder #0) bzw. auf den Drucker (Kanal #2) ausgegeben werden, allerdings erst, nachdem das Programm mit LOADGO geladen und gestartet oder nur mit RUN gestartet wurde. Die Ausgabe kann wie beim LIST-Befehl mit <CTRL-Q> gestoppt, mit <CTRL-S> fortgesetzt und mit <CTRL-E> abgebrochen werden (siehe LIST-Befehl).

a) LVAR <CR>

Ausgabe der verwendeten Variablen am Bildschirm.

b) LVAR#2 <CR>

Ausgabe auf den Drucker

c) LVAR#20 <CR>

Die Variablen werden über den Kanal Nr.20 in eine zuvor geöffnete Datei geschrieben (siehe auch den LIST-Befehl).

d) LVAR#20§1 <CR>

Mit diesem Befehl werden die Variablen in eine Datei mit wahlfreiem Zugriff als Datensatz Nr. 1 geschrieben.

24) MERGE:

Befehlsform:
MERGE # Kanal "Laufwerk : Dateiname . Dateizusatz"

Mit diesem Befehl können nur BASIC-Programme im ASCII-Format (siehe RESAVE) von der Diskette geladen und miteinander verbunden werden. Die Befehlsform entspricht der des LOAD-Befehls (Bitte dort nachlesen). Dabei überschreiben nachgeladene Programmzeilen die im Speicher mit der gleichen Zeilennummer vorhandenen Zeilen. Vor dem MERGE-Befehl muß man also die nachzuladenden Programmteile eventuell mit dem Befehl RENUMBER neu durchnummerieren (siehe RENUMBER).

a) MERGE "B:TEST.BAS" <CR>

Zu den im Speicher vorhandenen Programmzeilen wird vom Laufwerk B das Programm TEST.BAS hinzugefügt.

b) MERGE #3 <CR>

Da normalerweise die IOBYTE-Einstellung P=T vorgegeben ist, können mit diesem Befehl an ein im Speicher vorhandenes Programm über die READER-Routine von der Tastatur weitere Programmzeilen angehängt werden, die mit <CR> abschließen. Es erfolgt wie beim Befehl LOAD#3 keine Bildschirmanzeige der eingegebenen Zeichen. Mit <CTRL-Z> wird die Eingabe beendet. Werden Zeichen ohne Zeilennummer eingegeben, so führt dies zur Fehlermeldung "Zeilen-Nr. fehlt". Das Ganze mag recht merkwürdig erscheinen. Für ein sinnvolles Funktionieren dieses Befehls muß jedoch ein eigenes Maschinenprogramm z.B. für einen Strichcodeleser vorhanden sein, das von der READER-Sprungstelle im BIOS angesprungen wird (siehe mc 3/1981).

25) NEW:

Dieser Befehl löscht scheinbar ein im Speicher vorhandenes BASIC-Programm, da nicht mehr darauf zugegriffen werden

kann. Es wird allerdings nur der Zeilenzeiger gelöscht, der zu Beginn der ersten BASIC-Zeile in den Speicherstellen 45D8 und 45D9 als LOW-Byte und HIGH-Byte enthalten ist. Startet man mit der BREAK-Taste den Monitor, so wird das noch im Speicher vorhandene Programm z.B. mit dem Monitorbefehl D45D0,46FF als "Hexdump" angezeigt. Die erste BASIC-Zeile beginnt mit der Zeilennummer ab 45DA und endet mit dem Byte 0 (als 00 angezeigt). Die folgenden zwei Bytes enthalten den Zeiger auf die dritte BASIC-Zeile. Mit dem S-Befehl des Monitors wird die Adresse des ersten Bytes des Zeigers in die Speicherstellen 45D8 und 45D9 geschrieben. Nach dem Warmstart des Interpreters steht das Programm wieder zur Verfügung. Allerdings sind Variableninhalte nicht mehr zugänglich, reservierter Speicherplatz bleibt erhalten.

26) OPTION:

Befehlsform: OPTION # Kanal , " Parameter " , Argumente

Dieser Befehl ermöglicht die Veränderung der vorgegebenen Werte der Parameter der Ein- und Ausgabekanäle.

a) Parameter W (Zeilenlänge):

Mit dem Parameter W wird als Argument die Anzahl der Zeichen einer Zeile angegeben, nach denen automatisch ein Carriage Return (CR) und ein Linefeed (LF) folgen. Möglich sind Zeichenanzahlen von 14-255 (sonst Fehlermeldung "nicht erlaubt"). So können verschiedene Bildschirme und Drucker angepaßt werden.

OPTION#0,"W",60 <CR> ergibt eine Bildschirmzeilenlänge von 60 Zeichen. Eine Anpassung der Zeilenlänge bei einem Drukker erfolgt mit OPTION#2,"W",60 <CR>.

Der im Argument angegebene Wert wird für den Kanal #0 in der Speicherstelle 4143 (102F) und für den Kanal #2 in 4155 (103B) abgelegt.

b) Parameter N (Zeichenanzahl, Art nach CR und LF):

Im Argument 1 kann die Anzahl der Zeichen, im Argument 2 die Zeichenart als Dezimalwert angegeben werden, die nach CR/LF zusätzlich vom Interpreter ausgegeben werden. Damit können Zeitprobleme bei Peripherieeinheiten umgangen werden. Möglich sind 0-255 Zeichen im Argument 1. Fehlt Argument 2, wird es gleich 0 gesetzt.

Beim Bildschirm oder Drucker kann mit diesem Befehl der linke Rand nach rechts verschoben werden.

Der Befehl OPTION#0,"N",7,32 <CR> gibt zu Beginn jeder Bildschirmzeile 7 Leerzeichen (Blanks) aus, was auch für den Druckerkanal Nr.2 möglich ist.

c) Parameter Q
(Anfangs- und Schlußzeichen für Zeichenketten)

Als Argument kann der Dezimalwert dieser Zeichen angegeben werden. Null bedeutet ohne Begrenzungszeichen, 34 ergibt ein

Anführungszeichen ("). Die Wahl des Begrenzungszeichen von Zeichenketten (strings) erleichtert die Bearbeitung von sequentiellen Dateien, die durch ein BASIC-Programm erstellt wurden, mittels Textverarbeitungsprogramm, ohne daß die Trennzeichen erscheinen. So kann z.B. eine sequentielle Datei leicht mittels BASIC-Routine alphabetisch sortiert werden, was oft auch mit kommerziellen Textprogrammen nicht geht.

Der Befehl OPTION#10,"Q",0 ergibt Dateien ohne Trennzeichen.

d) Parameter P (Zeichen nach dem Input-Befehl = Prompt)

Wartet der BASIC-Interpreter bei einem Input-Befehl auf eine Eingabe, so wird dies normalerweise durch ein Fragezeichen am Bildschirm angezeigt. Als Argument kann der Dezimalwert eines beliebigen Zeichens eingesetzt werden, mit 0 entfällt dieses. Der Wert 63 ergibt ein Fragezeichen.
Mit OPTION#0,"P",42 <CR> wird nach dem INPUT-Befehl ein Sternchen (*) ausgegeben (Speicherstelle für das Zeichen: 4149 (1035h).

Voreinstellungen:

Mit dem Start des Interpreters sind für die jeweiligen Kanäle folgende Voreinstellungen gegeben:

Bildschirm (Kanal#0): W = 80, N = 3,0 , Q = 0, P = 63

Drucker (Kanal#2): W = 80, N = 3,0 , Q = 0

PUNCH (Kanal#4): W = 253, N = 0,0 , Q = 34

DISK (Kanal#10 bis Kanal#255): W = 253, N = 0,0 , Q = 34

27) OUT: siehe 17) INP

28) PEEK(:

Befehlsform: PEEK (Speicheradresse)

Es können Speicheradressen von 0 bis 65535 gelesen werden. Der Inhalt wird dezimal ausgegeben. Andere Adressen ergeben die Fehlermeldung "nicht erlaubt". Der Inhalt einer Speicherstelle kann auch einer Variablen zugewiesen werden.

PRINT PEEK(4149) <CR> oder PRINT PEEK(&1035) <CR>

Ausgabe von "63", da in dieser Speicherstelle das Zeichen abgelegt ist, das nach einem INPUT-Befehl ausgegeben wird (siehe den BASIC-Befehl OPTION). B = PEEK(4149) <CR> ist auch möglich.

29) POKE:

Befehlsform: POKE Speicheradresse , Wert

In die Speicherstellen 0 bis 65535 können Dezimalwerte von 0 bis 255 geschrieben werden. Beide Parameter sind auch über

Variablen zuweisbar.

POKE &1035,&3E <CR>

Der Wert 62 wird in die Speicherstelle 4149 geschrieben. Die Zahlen wurden sedezimal angegeben (Kennzeichnung mit "&", Buchstaben müssen groß geschrieben werden!)

30) PRINT:

Befehlsform: PRINT # Kanal § Satznummer , Konstante oder Variable oder Zeichenkette

Mit dem PRINT-Befehl erfolgt die Ausgabe von Zeichen oder Zeichenketten über den angegebenen Kanal (siehe Dateibefehle).

a) PRINT 123; A; "Testsatz" <CR>

Am Bildschirm werden die Zahl 123, der Wert der Variablen A und das Wort "Testsatz" jeweils mit zwei Leerzeichen Abstand voneinander ausgegeben. Kommas zwischen den Zeichen oder Zeichenketten ergeben einen Ausdruck jede 14. Position, solange die Zeichenketten kürzer als 14 Zeichen sind.

b) PRINT#2:PRINT#2,"kurzer";"Testsatz" <CR>

Vom Drucker wird nach einer Leerzeile der Satz "kurzer Testsatz" ausgegeben. Je nach Drucker müssen nach diesem Direktbefehl noch weitere Befehle oder eine Leerzeile folgen, damit der Ausdruck stattfindet.

c) ? 123 <CR>

An Stelle des Befehlswortes PRINT kann auch die Kurzform, ein Fragezeichen eingegeben werden. In BASIC-Zeilen wird das Fragezeichen jedoch nicht durch PRINT ersetzt, auch nicht beim Speichern oder Laden von Programmen.

31) PRIVACY:

Befehlsform: PRIVACY "Zeichenkette"

Durch die Angabe eines Schlüsselwortes (password) an erster Stelle in irgendeiner Programmzeile sind alle Befehle geschützt, mit denen der Inhalt eines BASIC-Programmes zugänglich gemacht oder verändert werden kann (AUTO, COPY,

DELETE, EDIT, FIND, LIST, RENUMBER, REPLACE, RESAVE UND SAVE). In allen Fällen wird die Fehlermeldung "nicht verfügbar" ausgegeben. Auch das Löschen einer Zeile durch Eingeben der Zeilennummer mit nachfolgendem <CR> geht nicht. Das Schlüsselwort kann irgendeine Konstante, Variable oder beliebige Zeichenkette sein.

10020 Privacy "SeCrEt"

In der Zeile 10020 wird als Schlüsselwort die Zeichenkette "SeCrEt" definiert.

Allen oben aufgeführten Befehlen muß unmittelbar nach dem Befehl das Schlüsselwort in Anführungszeichen folgen, danach können weitere, durch ein Komma getrennte Befehle angegeben werden, z. B. LIST "SeCrEt",100-200.

32) RENAME:

Befehlsform:
RENAME "Laufwerk : alter Name" , "neuer Name"

Der Dateiname wird im Inhaltsverzeichnis der Diskette des angegebenen Laufwerks geändert. Ein nicht angegebener Dateizusatz wird durch ".BAS" ersetzt. Ohne Angabe des Laufwerkes wird auf das unter CP/M angemeldete Laufwerk zugegriffen. Besteht schon eine Datei unter dem neuen Namen, erfolgt die Fehlermeldung "Datei existiert". Bei einem anschließendem Diskettenwechsel unbedingt den Befehl RESET eingeben, damit auf der Diskette das Inhaltsverzeichnis geändert wird.

RENAME "B:TEST1.BAS","TEST.BAS" <CR>

Der Name TEST1.BAS wird auf Laufwerk B in TEST.BAS geändert.

33) RENUMBER:

Befehlsform: RENUMBER Zeile , Abstand , Startzeile

Sind bei einem BASIC-Programm die Zeilennummern mit zu kleinem Abstand nummeriert worden, so schafft dieser Befehl Abhilfe. Alle Zeilennummern (Sprünge wie GOTO, GOSUB usw.) in einer BASIC-Zeile werden entsprechend berichtigt. Es kann auch erst ab einer bestimmten Zeile mit der Neunummerierung begonnen werden. Fehlt einer der Parameter, so wird der Wert 10 dafür angenommen. In einer Programmzeile XY bewirkt der Befehl RENUMBER die Meldung "nicht erlaubt Zeile XY".

a) RENUMBER <CR>

Alle Programmzeilen werden mit 10 beginnend mit dem Abstand 10 durchnummeriert. Die Befehle RENUMBER 10 <CR>, RENUMBER,10 <CR> oder RENUMBER 10,10 <CR> bewirken dasselbe.

b) RENUMBER 200,50 <CR>

Alle Programmzeilen werden mit 200 beginnend mit dem Abstand

50 durchnummeriert.

c) RENUMBER 1000,20,500 <CR>

Ab der alten Programmzeile 500 wird beginnend mit 1000 mit dem Abstand 20 durchnummeriert. Wird als Startzeile eine nicht existierende Programmzeile angegeben, erfolgt die Fehlermeldung "Sprungziel fehlt".

34) REPLACE:

Befehlsform:
REPLACE "alte Zeichenkette" "neue Zeichenkette",Zeilennummer oder Zeilenbereich

Ein bestimmte Zeichengruppe kann durch eine andere ersetzt werden. Dies erspart viele EDIT-Aufrufe, wenn z.B. in allen Programmzeilen ein Variablenname durch einen anderen ersetzt werden soll. Die veränderten Zeilen werden angezeigt. Die Trennzeichen können verschiedenartig sein, sie sind bei dem Befehl FIND näher erläutert. Allerdings müssen Anfangs- und Endzeichen jeweils gleich sein. Da BASIC-Befehle nur in Großbuchstaben in Programmen vorhanden sind, werden klein geschriebene Zeichenketten natürlich nicht gefunden. Die maximale Zeichenanzahl von 254 Zeichen darf durch die Veränderung nicht überschritten werden, sonst ergibt dies die Fehlermeldung "String zu lang".

REPLACE 'A$'"B$",10-2300

Die Variable A$ wird, soweit vorhanden, in den Zeilen 10 - 2300 durch die Variable B$ ersetzt. Sind Hochkommas in einer Zeichenkette zu ersetzen, müssen andere Trennzeichen wie z.B. (') verwendet werden.

35) RESAVE:

Befehlsform:
RESAVE "Laufwerk : Dateiname . Dateizusatz" , A

Es wird ein BASIC-Programm ab der Speicherstelle 45D8 bis zu der Adresse, ab der zwei Null-Bytes folgen, unter dem angegebenen Namen auf der Diskette abgelegt. Ein mit diesem Namen auf der Diskette vorhandenes Programm wird vorher gelöscht. Ein nicht angegebener Dateizusatz wird mit ".BAS" ergänzt. Nur der Dateizusatz darf durch ein "*" abgekürzt werden, er wird dann durch ".???" ersetzt. Beide Dateikennzeichnungen dürfen beliebig in Groß- oder Kleinbuchstaben eingegeben werden. Es sind alle Zeichen außer den Anführungszeichen im Dateinamen erlaubt, also auch Komma, Punkt und sogar Linefeed. Die Länge des Dateinamens darf 8 Zeichen, die des Zusatzes 3 Zeichen betragen (sonst Fehlermeldung: Dateiname falsch).
** Achtung ** Vor anschließendem Diskettenwechsel unbedingt den RESET-Befehl eingeben.

Wurde am Ende des Befehls der Buchstabe "A" nach dem Komma angegeben, wird der Speicherinhalt nicht in der Kurzform mit 1-BYTE-Befehlen abgespeichert (internes Format), sondern jedes Zeichen einzeln (ASCII-Format), was jedoch länger

dauert und mehr Platz auf der Diskette beansprucht. Mit dem Befehl MERGE können nur solche Programme im ASCII-Format geladen werden.

Internes und ASCII-Format unterscheiden sich dabei in den auf der Diskette abgelegten Zeichen erheblich, wie folgendes Beispiel einer Programmzeile zeigt:

```
Programmzeile:      10    PRINT A
internes Format:    E0 45 0A 00 95 20 41 00
```

ASCII-Format: 31 30 20 50 52 49 4E 54 20 41 0D 0A

Sind mehrere Programmzeilen vorhanden, so endet die letzte Zeile im ASCII-Format mit dem Zeichen 1A (CTRL-Z). BASIC-Programme kann man somit mit Texteditoren (z.B. mit ED.COM oder komfortableren Programmen) bearbeiten oder neu schreiben und dann im kürzeren internen Format erneut abspeichern.

a) RESAVE "TEST <CR> (letztes Anführungszeichen unnötig)

Ein Programm mit dem gleichen Namen wird zuvor auf der Diskette im angemeldetem Laufwerk gelöscht und dann das Programm TEST.BAS im internen Format auf die Diskette abgespeichert. Dies führt bei Verwechslung der Dateinamen zum Verlust des ursprünglichen Programmes.

b) RESAVE "b:DaT.,#!%&" <CR>

Diese Schreibweise ist unzulässig. Nach dem Punkt folgen 4 Zeichen als Dateizusatz, erlaubt sind nur drei.

c) RESAVE "TEST." <CR> (Punkt miteingeben)

Der Dateiname wird ohne Dateizusatz unverändert mit Punkt abgespeichert.

d) RESAVE "TEST.*" <CR>

Das Sternchen (*) wird durch drei Fragezeichen ersetzt und der Dateiname als TEST.??? abgespeichert.

** Achtung **
Der Befehl RESAVE "*.*" löscht alle Dateien auf der angemeldeten Diskette und speichert ein vorhandenes Programm mit dem Namen ????????.??? ab.

Wird nach dem RESAVE-Befehl die Fehlermeldung "kein Dateiplatz" ausgegeben, so wurde entweder ein Warmstart ohne den Befehl CLEAR,0 durchgeführt oder vorher mit <CTRL-E> ein Programmablauf abgebrochen, ohne mit CLOSE offene Kanäle zu schließen.

36) RESET:

Damit ein Diskettenwechsel unter BASIC erkannt wird, muß der Befehl RESET <CR> eingegeben werden. Dies entspricht dem <CTRL-C> unter CP/M. Wurde auf eine Diskette ein Schreibzugriff durchgeführt, muß vor und nach dem Wechsel der Diskette der Befehl RESET erfolgen, außer es wurde dann auf ein

anderes Laufwerk zugegriffen. Der Befehl darf nicht mit der auch so bezeichneten RESET-Taste (BREAK-Taste) verwechselt werden. Alle Kanäle müssen vorher geschlossen sein.

37) RINGBELL:

Befehlsform: RINGBELL

Mit diesem Befehl wird eine Maschinenroutine angesprungen,

die einen kurzen Piepton über die Ausgänge der PIO-A ausgibt. An die PIO muß über einen Treiber ein Lautsprecher angeschlossen sein. Die Tonhöhe ist in Speicherstelle 308, die Tondauer in 280 bzw. 281 enthalten.Beide können mit POKE verändert werden. Damit ist die Simulation der BELL-Funktion bei Eigenbauterminals möglich, zusätzlich lassen sich verschiedene Töne ausgeben.

38) RUN:

Befehlsform: RUN Zeilennummer der Startzeile

Es erfolgt die Löschung aller eventuell vorhandener Variablen und das Programm wird ab der angegebenen Zeilennummer gestartet. Entfällt die Zeilennummer, wird mit der ersten Programmzeile begonnen. RUN XY ergibt bei fehlender Zeile XY die Fehlermeldung "Sprungziel fehlt" (siehe auch GOTO).

39) SAVE:

Befehlsform: SAVE "Laufwerk : Dateiname . Dateizusatz ",A

Es gelten für diesen Befehl die selben Anweisungen wie für den RESAVE-Befehl mit einer Ausnahme: Befindet sich ein Programm mit dem angegebenen Namen auf der Diskette, wird die Fehlermeldung "Datei existiert" ausgegeben und dann die "Fertig"-Meldung, die Abspeicherung also nicht durchgeführt.

40) STOP:

Befehlsform: STOP

Mit diesem Befehl kann ein Programm, das nicht so funktioniert wie erwartet (soll häufig vorkommen!), an einer beliebigen Stelle im Programm angehalten werden. Alle Variablen bleiben erhalten und können z.B. mit LVAR oder mittels PRINT-Befehl untersucht werden.

145 STOP <CR>

Das Programm hält in Zeile 145 an und gibt die Meldung "** HALT ** Zeile 145" und "Fertig" aus.

41) TRACE:

Befehlsform: TRACE # Kanal § Satznummer , Wert

Die Zeilennummern der vom BASIC-Interpreter bearbeiteten Programmzeilen werden nacheinander auf den angegebenen Kanal ausgegeben, wenn für den Parameter WERT die Zahl 1 eingesetzt wurde. Wie bei dem Befehl LVAR kann die Ausgabe auch auf Diskette sequentiell oder mit wahlfreiem Zugriff erfolgen. TRACE wird durch den WERT 0 wieder ausgeschaltet.

a) TRACE 1 <CR>

Ausgabe der Zeilennummern der bearbeiteten Programmzeilen in Spitzklammern am Bildschirm (<10><20><30><145> usw.). Trace 0 schaltet den Befehl ab.

b) TRACE#2,1 <CR>

Ausgabe der Zeilennummern auf den Drucker.

c) TRACE#11,1 <CR>

Die Zeilennummern werden z.B. in eine mit OPEN#11,"O","TRA-TEST" eröffnete Datei geschrieben. Im Programm darf jedoch kein CLEAR-Befehl enthalten sein, da sonst der Kanal #11 wieder geschlossen wird (Fehlermeldung "Datei nicht offen"). Der CLEAR-Befehl kann ersatzweise direkt vor dem RUN-Befehl eingegeben werden.

42) VARPTR:

Befehlsform: VARPTR (Variable)

Die dezimale Adresse der Inhalte von Variablen und Matritzenelementen im Speicher wird mit diesem Befehl ermittelt und kann direkt bei den Befehlen PEEK und POKE verwendet werden. Bei Variablen für Zeichenketten (z.B. A$) wird die Adresse des Stringzeigers angegeben. Das erste Byte an dieser Adresse enthält die Länge der Zeichenkette, das nächste Byte ist immer 0. Die folgenden beiden Bytes enthalten die Adresse der Zeichenkette, das LOW-Byte zuerst. Die letzten beiden Bytes sind nicht benützt.

Für die Befehlsbeispiele ist folgendes Programm einzugeben und mit RUN zu starten:

```
10 A = 2
20 A$ = "TEST"
```

a) PRINT VARPTR(A) <CR>

Ausgabe der Zahl 17911 (45F7)

b) PRINT VARPTR(A$) <CR>

Ausgabe der Zahl 17919 (45FF). PRINT PEEK(17919) ergibt 4, die Länge der Zeichenkette in A$. In der Speicherstelle 17921 (4601h) ist der Wert 237 (ED), in 17922 (4602h) der Wert 69 (45h) enthalten. Die Adresse des Wortes TEST berechnet sich nach HIGH-Byte-Wert * 256 + LOW-Byte-Wert, also 69 * 256 + 237 = 17901 (45ED). Eine Kontrolle ist anhand des Speicherauszuges möglich.

```
45D0 00 00 00 00 00 00 00 00 E3 45 0A 00 41 20 CD 20
45E0 11 82 00 F3 45 14 00 41 24 20 CD 20 22 54 45 53
45F0 54 22 00 00 00 00 41 00 00 00 00 00 82 80 41 04
4600 00 ED 45 00 00 00 00 00 00 00 00 00 00 00 00 00
4610 00 .
```

43) WAIT:

Befehlsform: WAIT (Adresse A, Maske M, Zustand Z)

Der BASIC-Interpreter wartet in einer Schleife, bis ein Eingangsport mit der dezimalen Adresse A in den mit der Maske M angegebenen Bits den Zustand Z erreicht hat. Erst dann wird die nächste Programmzeile bearbeitet. So können z.B. direkt vom Port A der PIO einzelne Bits untersucht werden, ob sie den Zustand Ø oder 1 besitzen.

Der von A eingelesene Wert wird mit dem Wert Z Exklusiv-Oder verknüpft und das Ergebnis mit dem in M angegeben Wert Und verknüpft. Solange das Ergebnis gleich Ø ist, wird das Ganze wiederholt. Erst dann geht es zum nächsten Befehlsabschnitt.

Wird der Parameter Z nicht angegeben, so ist Z=Ø. Die Werte A, M und Z müssen dezimal angegeben werden (einfach sedezimal mit "&" angeben).

Beispiel: Es soll solange gewartet werden, bis am Bit 1 der PIO-B der Pegel 5 Volt angelegt wird.

Für die PIO ergeben sich folgende Adressen (mc-Computer):

Port A: Daten: F4 Steueradresse: F5
Port B: Daten: F6 Steueradresse: F7

Für die Ein- und Ausgabe von Bitkombinationen muß das Steuerwort 11XX1111 lauten (X beliebig).

a) Einschalten der PIO-B auf Bit-Modus:

Out &F7,&CF <CR> (CF oder 207, binär 11001111)

b) Abfragen des Bit Ø der PIO-B:

WAIT &F6,1 <CR>

Werden 5 Volt an den Anschluß 27 der PIO gelegt, erscheint die Meldung "Fertig".

IV. BASIC-Befehle

Die nun folgenden Befehle lassen sich nicht allein als Direktbefehl verwenden, jedoch oft nach einem solchen. Die Befehle sind wieder alphabetisch geordnet (außer zusammengehörende Befehle).

1) ASC(:

Befehlsform: ASC(Zeichen oder Variable oder Zeichenkette)

Es wird der dezimale Wert (nach ASCII) des Zeichens, des Variablenwertes oder des ersten Zeichens der Zeichenkette ermittelt.

Beispiele:

```
A = ASC("H") <CR>    PRINT A <CR>          Anzeige: 72
A$ = "Meier" <CR>    PRINT ASC(A$) <CR>    Anzeige: 77

PRINT ASC("") <CR>  Anzeige: 0
```

2) BYTEPOLL(:

Befehlsform: BYTEPOLL (# Kanal)

In Kombination mit dem INTERRUPT-Befehl kann abgefragt werden, ob ein Byte an einer Eingabeeinheit zum Lesen vorliegt, z.B. ob eine Taste der Tastatur gedrückt wurde. BASIC führt eine Tastaturabfrage nach Kontrollzeichen normalerweise ständig durch. Der Befehl BYTEPOLL kann so nie den Wert -1 (ein Byte liegt vor und kann gelesen werden) erreichen. Mit dem Befehl INTERRUPT 0 muß daher die automatische Tastaturabfrage ausgeschaltet werden.

3) DATE$: siehe I. Start des Interpreters

4) CHR$(:

Befehlsform: CHR$(dezimaler Wert 0 - 255 oder nummerische Variable)

Das der angegebenen Zahl entsprechende Zeichen nach ASCII kann ausgegeben oder einer Variablen zugewiesen werden. So lassen sich auch Steuerzeichen an Drucker und Videoeinheiten übermitteln.

Beispiele:

a) CR$ = "TEST" + CHR$(13) + CHR$(10) + "ENDE" <CR>
   PRINT CR$ <CR>
   Anzeige: TEST
            ENDE

b) A$ = CHR$(34) <CR> PRINT A$ <CR> Anzeige: "
   dagegen: A$ =""" <CR> Anzeige: Syntaxfehler

c) PRINT CHR$(12) <CR> Bildschirmlöschung (Form-Feed)
d) Ausgabe einer Escape-Sequenz an den Drucker:
   PRINT#2,CHR$(27);CHR$(17) <CR>
   (je nach Drucker unterschiedliche Wirkung)

e) PRINT CHR$(256) <CR> Anzeige: nicht erlaubt
   PRINT CHR$(-5) <CR> " : " "

f) 10 INPUT"Kleinbuchstabe eingeben";a$ Eingabe: w
   20 PRINT CHR$(ASC(A$)-32):GOTO 10 Anzeige: W

5) DATA ..... READ ..... RESTORE XYZ

Diese Befehle werden zusammen benötigt, um in einem BASIC-Programm beliebigen Variablen bestimmte Werte oder Zeichen

zuweisen zu können.

a) DATA

Befehlsform: DATA Zahl oder Zeichenkette, durch Komma getrennt (insgesamt 255 Zeichen in einer BASIC-Zeile)

Mögliche Zuweisungen:

```
10 DATA 1,2," HUBER",MEIER,,,4,ZZ
```

Sollen Leerzeichen am Anfang einer Zeichenkette mitgelesen werden, so müssen die Zeichenketten in Anführungszeichen gesetzt werden (z.B. " HUBER").

b) READ

Befehlsform: READ nummerische Variable, Stringvariable, durch Komma getrennt

```
10 DATA 1,2," HUBER",MEIER,,,4,ZZ
20 READ A,B,A$,B$,C,C$,D(1,1,1),D$(1,3)
30 PRINT A;B;A$;B$;C;C$;D(1,1,1);D4(1,3)
```

Anzeige: 1 2 HUBERMEIER 0 4 ZZ

Die Variable C enthält den Wert 0, C$ ist leer.

c) RESTORE XYZ

Befehlsform: RESTORE Zeilennummer

Mit jedem Lesevorgang wird ein Zeiger auf die nächste Konstante innerhalb der DATA-Zeilen gesetzt. Am Ende der letzten DATA-Zeile eines Programmes zeigt der DATA-Zeiger ins "Leere", ein weiterer Lesezugriff ergibt die Fehlermeldung "keine Daten Zeile XYZ". Ein RESTORE-Befehl setzt den DATA-Zeiger auf die erste Konstante zurück.

Die Angabe der Zeilennummer einer DATA-Zeile setzt den Zeiger auf die erste Konstante in dieser Zeile zurück. Die Fehlermeldung "Sprungziel fehlt Zeile XYZ" wird durch Angabe einer nicht vorhandenen BASIC-Zeile nach dem Befehl RESTORE verursacht.

6) DIM:

Befehlsform:
DIM Variable 1 (Elementanzahl), Variable 2 (Elementanzahl), usw.

Ohne diesen Befehl am Programmanfang werden pro Matrixvariable 10 Elemente zugelassen (0 - 9). Matrixvariablen (indizierte Variablen) sind Variablen vom Typ A(I), A$(I), A(2,3) oder A$(20,8,2). Zur Reservierung des benötigten Speicherplatzes bei mehr als 10 Elementen dient der DIM-Befehl. Doppelte Reservierung für die gleiche Variable ergibt die Fehlermeldung "DIM-Befehl doppelt".

Der Interpreter HEBAS besitzt zur Löschung der Dimensionie-

rungen von Variablen den Befehl KILL (siehe Abschnitt III. 20), falls Speicherplatz knapp werden sollte.

Beispiel:

```
DIM A$(100),B(20,20),C$(30,30,30),F$(A),D$(X,Y,Z)
```

Die Elementanzahl einer indizierten Variablen kann selber durch eine Variable angegeben werden.

7) ELSE:

Dieser Befehl wird in der Befehlsgruppe IF .... THEN .... ELSE verwendet und dort beschrieben.

8) END:

Nach einer solchen Anweisung in einer BASIC-Programmzeile wird das Programm verlassen und die Meldung "Fertig" angezeigt. Direkt-Befehle wie LIST, EDIT usw. können eingegeben werden. Der Befehl ist nicht unbedingt als Programmende notwendig; er trennt jedoch Unterprogramme, die mit GOSUB aufgerufen werden, vom Hauptprogramm ab.

9) EOF:

a) Befehlsform: ON EOF GOTO Zeilennummer

Der Befehl ermittelt in den gelesenen Daten das Zeichen (Dateiende) CTRL-Z (1A), falls Daten im "O"-Modus auf Diskette geschrieben wurden und verzweigt zur angegebenen Zeilennummer (siehe Abschnitt VI. B.2). Der Befehl kann vor dem OPEN-Befehl angeordnet werden.

Beispiel:

```
10 INPUT"Dateiname";D1$
20 ON EOF GOTO 70
30 OPEN#11,"I",D1$
40 INPUT#11,A$
50 PRINT A$
60 GOTO 40
70 CLOSE #11
```

Es werden solange Daten gelesen, bis das Dateiende-Zeichen erkannt wurde.

10) EXCHANGE:

Befehlsform: EXCHANGE Variable 1, Variable 2

In Sortierroutinen müssen oft Variablen vertauscht werden. Dies geschieht im einfachsten Fall mit einer Hilfsvariablen wie beim sog. Dreieckstausch:

```
W$ = A$(I): A$(I) = A$(I+1): A$(I+1) = W$
```

Dies kostet Zeit und Speicherplatz. Wesentlich schneller geht es mit dem Befehl EXCHANGE A$(J),A$(I), wobei nun nur Zeiger auf die Variablen vertauscht werden.

11) EXIT:

siehe den Befehl FOR.

12) FIX$(:

Befehlsform: FIX$(Variable , Zeichenanzahl)

Für Stringoperationen benötigt man oft eine ganz bestimmte Länge einer Zeichenkette. Dieser Befehl füllt eine vorhandene Zeichenkette mit Leerzeichen (space) bis zur angegebenen Länge (bis zu 255 Zeichen) auf oder schneidet sie ab.

Beispiel:

```
10 INPUT"Wort, Länge";A$,L
20 IF A$ = "E" THEN END
30 A$ = FIX$(A$,L)
40 PRINT A$;"Länge";LEN(A$)
50 GOTO 10
```

13) FOR ... TO ... STEP ... EXIT ... NEXT

Diese Befehle dienen zum Aufbau von Befehlsschleifen.

Befehlsform:
FOR Laufvariable = Anfangswert TO Endwert STEP Schrittweite
beliebige Befehle
NEXT Laufvariable

Beispiel:

```
10 FOR I = -1 TO 1 STEP 0.1
20 PRINT I,"TEST"
30 NEXT I
```

Die Angabe der Laufvariable nach dem Befehl NEXT kann bei nur einer Schleife entfallen (siehe auch Abschnitt V.4: Fallentest).
Bei verschachtelten Schleifen muß die jeweilige Laufvariable angegeben werden. Solche Schleifen müssen nach dem Schema FOR A..FOR B..FOR C..usw - NEXT C..NEXT B..NEXT A aufgebaut sein. Da die Rücksprungadressen der Schleifen in einem bestimmten Speicherbereich (stack) zwischengespeichert werden, ist auch die Anzahl der verschachtelten Schleifen begrenzt.

Beispiel:

```
 5 DIM A(15,15,15)
10 FOR I = 1 TO 15
20 FOR K = 1 TO 15
30 FOR L = 1 TO 15
40 A(I,K,L) = 1
50 NEXT L,K,I
```

Eine dreidimensionale Matrix wird mit dem Wert 1 aufgefüllt. Mehrere Laufvariablen können, durch ein Komma getrennt, in eine Zeile geschrieben werden.

Mit dem Befehl EXIT wird die Schleife verlassen, bevor die jeweilige Laufvariable ihren Endwert erreicht hat. Dies kann zwar auch mit dem Befehl GOTO durchgeführt werden, aber dann kommt es bei nachfolgenden Schleifen zu der Fehlermeldung "NEXT ohne FOR Zeile XYZ".

Befehlsform: EXIT Zeilennummer, Laufvariable

Beispiel:

```
10 FOR I = 1 TO 10
15 INPUT"Eingabe";X$
20 IF X$ = "e" THEN 100
25 A$(I) = X$
30 NEXT
40 END
50 :
100 FOR J = 1 TO 10
110 FOR I = 1 TO 10
120 PRINT I;J
130 NEXT I
140 NEXT J
```

Nach dem Sprung zur Zeile 100 tritt beim ersten NEXT J die Fehlermeldung "NEXT ohne FOR Zeile 140" auf. Der EXIT-Befehl in einer neuen Zeile 20 beseitigt diesen Fehler: 20 IF X$ = "e" THEN EXIT 100,I

14) GOSUB ... RETURN

a) GOSUB:

Befehlsform: GOSUB Zeilennummer

Der Interpreter springt zur angegebenen Zeilenummer und setzt dort seine Arbeit fort, bis der Befehl RETURN auftaucht. Dabei wird die Rücksprungadresse gespeichert. Mehrere GOSUB-Befehle können nacheinander folgen.

Befehlsform:

ON Z GOSUB Zeilennummer A, Zeilennummer B, usw.

Entsprechend der Zahl Z (0 - 255) wird zu den angegebenen Zeilen verzweigt, bei Z = 2 also zur Zeilennummer B. Ist Z = 0 oder größer als die Anzahl der angegebenen Zeilennummern, so wird der nächste Befehl nach dem vollständigen GOSUB-Befehl bearbeitet.

b) RETURN:

Befehlsform: RETURN Zeilennummer

Nach der Bearbeitung eines mit dem Befehl GOSUB erreichten Unterprogrammes wird nicht hinter diesen Befehl gesprungen, sondern zu der angegebenen Zeilennummer. So kann über GOSUB-Ebenen hinweg gesprungen werden.

Befehlsform:ON Z RETURN Zeilennummer A, Zeilennummer B, usw.

Wie beim GOSUB-Befehl kann auch unter bestimmten Bedingungen der RETURN-Befehl ausgeführt werden.

15) GOTO:

Befehlsform: GOTO Zeilennummer

Der Interpreter verzweigt zur angebenen Zeilennummer, falls sie vorhanden ist, sonst erfolgt Fehlermeldung "Sprungziel fehlt". Ein Rücksprung ist nicht möglich.

Befehlsform: ON Z GOTO Zeilennummer A, Zeilennummer B, usw.

Die Wirkungsweise ist dieselbe wie beim GOSUB-Befehl, ein Rücksprung kann jedoch nicht erfolgen (siehe auch III.16).

16) IF...THEN (GOTO)...ELSE

Befehlsform:
IF Bedingung THEN (oder GOTO) Befehlsgruppe 1 ELSE Befehlsgruppe 2

Programmverzweigungen unter bestimmten Bedingungen sind mit dieser Befehlsgruppe möglich, ebenso mehrere aufeinanderfolgende IF-Befehle, die durch logische Operatoren (siehe Abschnitt V) verknüpft sind. Klammern sind bei der folgenden Form erforderlich: IF X$ = (A$ + B$) THEN .. , sonst erfolgt die Fehlermeldung "Datentypen (Zahl/String) vermischt".

Beispiel:

```
10 INPUT A,B$
10 IF A = 2 AND B$ = "E" THEN 30 ELSE 40
20 STOP
30 PRINT"EINGABE RICHTIG":END
40 GOTO 10
```

17) INPUT:

a) Befehlsform:

INPUT "beliebiger Text" ; Variable 1, Variable 2, usw.

Mit diesem Befehl werden Zahlen in nummerische (A0 - Z9) und Zeichenketten in Stringvariablen (A0$ - Z9$) eingelesen (siehe VARPTR). Variablenkennzeichnungen mit mehr als zwei Buchstaben sind erlaubt, werden aber nicht unterschieden. <CR> als Eingabe führt nicht zum Programmabbruch, sondern übernimmt den Wert 0 bei nummerischen bzw. einen Leerstring bei Stringvariablen (siehe auch VI.B.b: INPUT#).

Beispiel: 10 INPUT "EINGABE";A

Auf dem Bildschirm erscheint nach dem Wort "Eingabe" ein Fragezeichen (siehe OPTION) und der Interpreter wartet solange, bis eine Zahl über die Tastatur eingegeben wurde. Der Versuch, Buchstaben einzugeben, ergibt die Fehlermeldung "* Falsche Eingabe *". Sedezimale Zahlen werden mit vorangestelltem "&"-Zeichen und mit Großbuchstaben eingegeben.

10 INPUT "Eingabe";A1,BA$,A2

Nach der Ausgabe des Fragezeichens können eine Zahl, eine Zeichenkette und wieder eine Zahl, durch Komma getrennt, eingegeben werden. Wird die erste Eingabe mit <CR> abgeschlossen, so erscheinen zwei Fragezeichen ohne daß der Text in Anführungszeichen nochmal ausgegeben wird. Bei der zweiten Eingabe (Variable BA$) ergibt ein Komma die Fehlermeldung "* Komma falsch *" und die Zeichen nach dem Komma werden nicht übernommen.

b) Befehlsform: LINE INPUT "Text"; Stringvariable

Nun können auch Kommas in der Eingabe enthalten sein, ohne daß es zu Fehlermeldungen kommt. Es sind nur Stringvariablen erlaubt.

18) INSTR(:

Befehlsform:
INSTR ("Zeichenkette 1", "Zeichenkette 2", Startzeichen, Zeichenanzahl)

Dieser Befehl durchsucht eine Zeichenkette (Stringvariable oder Zeichenkette in Anführungszeichen) nach einer zweiten Zeichenkette. Die Suche beginnt ab dem ersten Zeichen oder falls angegeben, ab dem Startzeichen und erstreckt sich über soviele Zeichen wie mit der Zeichenanzahl angegeben sind.

Beispiele:

```
PRINT INSTR("aabbccddeeffgg","bb")    <CR>      Anzeige: 3
PRINT INSTR("aabbccddeeffgg","hh")    <CR>          "  : 0
PRINT INSTR("abcdefabcdefgh","cd",4) <CR>          "  : 9
```

19) LEFT$(... RIGHT$(:

a) Befehlsform:

LEFT$("Zeichenkette" oder Stringvariable, Zeichenanzahl 0 -

255)

Vom linken Teil der Zeichenkette wird die angegebene Zeichenanzahl übernommen. Eine negative Zeichenanzahl ist nicht erlaubt.

```
Beispiel:
10 A$ = "1234567890"          Eingabe: 3   Anzeige: 123
20 INPUT"Zahl";Z                  "    : 0   keine Anzeige
25 IF Z = -1 THEN END             "    : 11  Anzeige: 1234567890
30 LE$ = LEFT$(A$,Z)              "    : -2      "    : nicht er-
40 PRINT LE$                                 laubt Zeile 30
50 GOTO 20
```

b) Befehlsform:

RIGHT$("Zeichenkette" oder Stringvariable, Zeichenanzahl 0 - 255)
Für diesen Befehl gelten die gleichen Bedingungen wie für LEFT$(, nur wird jetzt der rechte Teil der Zeichenkette übernommen.

20) LEN(:

Befehlsform: LEN("Zeichenkette" oder Stringvariable)
Die Byteanzahl der Zeichenkette wird dezimal ausgegeben.

```
PRINT LEN("EEEEEEEE") <CR>          Anzeige: 8
```

21) MID$(:

a) Befehlsform:

MID$("Zeichenkette" oder Stringvariable, Startzeichen, Zeichenanzahl)

Eine durch das Startzeichen und die Zeichenanzahl bestimmte Folge von Zeichen wird von links beginnend in der Zeichenkette gesucht und übernommen.

```
Beispiel:
10 INPUT"Wort";A$
20 IF A$ = "E" THEN END
30 INPUT"Startzeichen";SZ
40 INPUT"Zeichenanzahl";ZA
50 B$ = MID$(A$,SZ,ZA)
60 PRINT B$:GOTO 10
```

Eingabe: Wort:	Startzeichen:	Zeichenanzahl:	Anzeige:
Testwort	1	3	Tes
"	5	1	w
"	0	3	nicht erlaubt
"	1	10	Testwort
"	<CR>	<CR>	nicht erlaubt

b) Befehlsform:
MID$(Stringvariable,Startzeichen,Zeichenanzahl)=Zeichenkette

In einem vorhandenen String kann ab einem Startzeichen eine bestimmte Zeichenanzahl mit Zeichen aus einer Zeichenkette überschrieben werden. Dies ist wesentlich schneller als eine Teilung und Ergänzung einer Zeichenkette.

Beispiel:

```
10 INPUT"Satz";A$
20 INPUT"Startzeichen";SZ
30 INPUT"Zeichenanzahl";ZA
40 INPUT"Zeichenfolge";B$
50 MID$(A$,SZ,ZA) = B$
60 PRINT A$:GOTO 20
```

Eingabe:

```
Satz          : Drei Chinesen mit dem Kontrabaß
Startzeichen  : 8
Zeichenanzahl: 6
Zeichenfolge  : anasen
```

Anzeige: Drei Chanasen mit dem Kontrabaß

Fehlt die Zeichenanzahl, wird die gesamte Zeichenfolge ab dem Startzeichen eingesetzt. Ist die vorhandene Zeichenkette kürzer, wird der einzusetzende Textteil auf die angegebene Zeichenanzahl mit Leerzeichen ergänzt.

22) POS(X):

Die momentane Druck- oder Cursorposition wird ausgegeben oder einer Variablen zugewiesen, wobei die erste Postion den Wert 0 besitzt. X ist eine beliebige Hilfsvariable.

Beispiel:

```
10 CLEAR 250
20 FOR I = 1 TO 70
30 A$ = A$ + "*":PRINT A$;:PRINT POS(X):NEXT
```

23) REM:

Zur Dokumentation von Programmen empfehlen sich Hinweise, die aber nicht vom Interpreter bearbeitet werden dürfen. Der Befehl REM oder die Kurzfassung (') kennzeichnen solche Hinweise in BASIC-Zeilen, die durch einen Doppelpunkt oder durch das Ende der Programmzeile beendet werden. Es können also weitere BASIC-Befehle in einer Programmzeile nach dem REM-Befehl angeordnet werden.

Beispiel:

```
10 A1 = 0: ' Anzahl Daten : B1 = 0: ' Anzahl Zeilen
```

24) SPACE$(:

Befehlsform: SPACE$(Anzahl der Leerzeichen)

Dieser Befehl ergibt eine Leerzeichenkette mit der angegebenen Länge, wobei maximal 255 Zeichen erlaubt sind und diese einer Variablen zugewiesen werden können.

Beispiel:

```
10 A$ = SPACE$(20)
20 PRINT "A" + A$ + "Z"
```

Anzeige: A (20 Leerzeichen) Z

25) SPC(:

Befehlsform: SPC(Zahl der Leerzeichen)

Zur Ausgabe von Zwischenräumen dient der Befehl SPC(. Eine Zuweisung zu einer Variablen ergibt die Fehlermeldung "Syntaxfehler Zeile XYZ".

Beispiel:

```
PRINT "TEST";SPC(10);"TEST" <CR>
```

Anzeige: TEST TEST

Es werden zehn Leerzeichen eingefügt.

26) STRING$(:

Befehlsform:
STRING$("Zeichenkette" oder Stringvariable , Anzahl X)

Eine bestimmte Zeichenkette wird mit der angegebenen Anzahl X "multipliziert", d.h. die neue Zeichenkette enthält X-mal die alte Zeichenkette. Die Summe aller Zeichen darf die Zahl 255 nicht überschreiten (Fehlermeldung: nicht erlaubt Zeile 20).

Beispiel:

```
10 A$ = "*-"
20 B$ = STRING$(A$,10)
30 PRINT B$
```

Anzeige:

--*-*-*-*-*-*-*-*-

27) TAB(:

Befehlsform: TAB(Position X ab linkem Rand);

Bei der Ausgabe eines Zeichens am Bildschirm oder auf den Drucker wird der Cursor oder Druckkopf immer vom linken Rand aus X Zeichen nach rechts versetzt. Im Beispiel kann daher der zweite Befehl TAB(5) nicht ausgeführt werden.

```
PRINT TAB(10);"*";TAB(5);"*" <CR>
Anzeige:          **
```

28) USING

Bei der Ausgabe von Zahlen und Zeichenketten ergibt sich die Anordnung aus der Zeichenanzahl. Ein Tabellenaufbau ist daher nur mühsam mit entsprechenden Stringoperationen möglich. Der Befehl USING bietet einfache Tabellierungsanweisungen, die sogar im Programm gewechselt werden können.

a) Befehlsform:

PRINT # Kanal, USING Zeilennummer XYZ; Variable 1, Variable 2, usw.

XYZ ! Formatierungsangabe (#,.,+,-,**,$$,**$,',^^^^,L,R,C,E)

b) Befehlsform:

PRINT # Kanal, USING "Formatierungsanweisung" oder Variable; Variable 1, usw.

A) Zahlenformatierung:

Die Beispiele enthalten zuerst das Format, dann die Zahl und die Anzeige (bzw. Ausdruck).

```
10 INPUT LINE "Format";A$
20 INPUT"Zahl";A
30 PRINT USING A$;A
```

a) #

Jedes Nummernzeichen (#) kennzeichnet eine Zeichenposition in einem Zahlenfeld. Die Zahl wird rechtsbündig ausgegeben und notfalls gerundet.

```
######        234         234
          122345.6    122346
```

b) . (Punkt!)

Ein Dezimalpunkt innerhalb der Nummernzeichen bestimmt die Anordnung der Dezimalzahl, die notfalls gerundet wird. Bei Zahlen kleiner 1 wird vor den Punkt eine Null gesetzt.

```
###.##         23.456       23.46
                .12345        0.12
###.###       -24.5        -24.500
```

c) +

Das Plus-Zeichen kann am Anfang oder am Ende eines Nummernfeldes benutzt werden, entsprechend wird das Vorzeichen (+ oder -) vor oder hinter die Zahl gedruckt.

```
####.###+     234          234.000+
             -234.5        234.500-
+####.###     234         +234.000
             -234.5       -234.500
```

d) -

Ein Minus-Zeichen am Ende des Nummernfeldes ergibt bei negativen Zahlen ein Minus-Zeichen am Ende, bei positiven ein Leerzeichen.

```
####.###-    123.567       123.567
            -123.567       123.567-
```

e) **

Zwei Sternchen am Anfang eines Nummernfeldes bedeuten, daß an jede nicht besetzte Position des Nummernfeldes ein Sternchen gesetzt wird (z.B. für Zahlkarten üblich).

```
*****.###   1212.12     *1212.12
              12.12     ***12.12
               2.30     ****2.30
```

f) $$

Zwei Dollarzeichen am Anfang eines Nummernfeldes setzen das Dollarzeichen unmittelbar vor das erste Zeichen der Zahl, das nicht Null ist.

```
$$####.##      555         $555.00
                 4.45        $4.45
```

g) **$

Es werden die Anweisungen Nr.5 und 6 gemeinsam benützt.

```
**$##.##       333        *$333.00
                 3.4      ***$3.40
```

h) , (Komma!)

Ein Komma links vom Dezimalpunkt im Nummernfeld bedingt, daß alle drei Zeichen ein Komma gesetzt wird.

```
########,.##        12345     12,345.00
                1234567.1  1,234,567.10
```

i) ^^^^

Vier Pfeile am Ende der Kennzeichnung des Nummernfeldes deuten an, daß die Zahl immer in exponentialer Schreibweise angegeben wird (E+nn oder E-nn). Ein Dezimalpunkt ist erlaubt.

```
###.##^^^^      1234567     12.35E+05
               0.123       12.30E-02
```

Paßt eine Zahl nicht in das Nummernfeld, wird sie doch ganz

ausgegeben und durch ein vor die Zahl gestelltes Prozentzeichen (%) gekennzeichnet.

B) Textformatierung

Textformatierungsanweisungen sind durch ein Apostroph (') gekennzeichnet. Jedes Apostroph bedeutet ein einzelnes Zeichen. Zur Vereinfachung gibt es die Buchstaben L, R, C und E (nur Großbuchstaben erlaubt) zur Kennzeichnung von längeren Zeichenketten.

a) L (linksbündig)

Befehlsform: 'LLLL
Die Anzahl der L-Buchstaben ergibt die max. Länge der Zeichenkette, eine längere wird rechts abgeschnitten.

b) R (rechtsbündig)

Befehlsform: 'RRRRRRRR
Die Anzahl der R-Buchstaben ergibt die max. Länge der Zeichenkette, eine längere wird links abgeschnitten.

c) C (zentrieren)

Befehlsform: 'CCCCCCCCCCCCCCCCCCCC
Die Zeichenkette wird im C-Feld zentriert, eine zulange Zeichenkette wird rechts abgeschnitten.'

d) E (linksbündung und ergänzen)

Befehlsform: 'E
Es braucht nur ein E angegeben werden, da die gesamte Zeichenkette ausgegeben wird.

Beispiel:

```
 5 CLEAR 250
10 INPUT"Zeichenkette";A$
20 INPUT LINE "Format";X$
30 PRINT X$
40 PRINT USING X$; A$
50 GOTO 20
```

Anzeige: (Format und darunter Ausdruck)

```
'E
Wir gehen nach Hause und schlafen

'LLLLLLLLLLLLLLLLLLLLLLLL
Wir gehen nach Hause und s

'RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
       Wir gehen nach Hause und schlafen

'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
       Wir gehen nach Hause und schlafen
```

Im ersten Teil sind die mathematischen Operatoren, Vergleichsoperatoren und logischen Operatoren in abnehmender Priorität aufgeführt. Im zweiten Teil folgen mathematische Funktionen, dann Umwandlungsbefehle wie Zahl zu String usw., im vierten Teil sind einige Beispiele zur Rechengenauigkeit aufgeführt. Alle Befehle können als Direktbefehl und in BASIC-Zeilen verwendet werden. Abschließend wird die Definition eigener BASIC-Funktionen mit dem DEF-Befehl beschrieben.

PRECISION:

Befehlsform: PRECISION (Zeichenanzahl)

Ergebnisse von Berechnungen (z.B. Dezimalbrüche) werden mit maximal 11 Zeichen (ohne Punkt) ausgegeben. Diese Anzahl kann verringert werden, wobei die letzte ausgegebene Stelle gerundet ist. Intern wird mit 11 Zahlen weiter gerechnet. Die Zahl 0 oder keine Angabe nach dem Befehl ergeben wieder 11 Zeichen.

PRECISION 5 <CR>

Es werden bei Rechenergebnissen Zahlen mit maximal 5 Zeichen ohne Punkt ausgegeben. PRINT 10/7 <CR> ergibt die Zahl 1.4286, Print 100/7 <CR> ergibt 14.286. Mit PRECISION 0 <CR> wird nach PRINT 10/7 die Zahl 1.4285714286 ausgegeben.

1) Operatoren:

a) () Ausdruck in Klammern

b) ^ Exponentialoperator

c) - Negation (Zahl oder Ausdruck)

d) * Multiplikation / Division

e) \ Ganzzahldivision

f) MOD Modulus

g) + Addition - Subtraktion

h) = Gleichheitszeichen

<> ungleich

< kleiner als

> größer als

<= kleiner oder gleich =< gleich oder kleiner

>= größer oder gleich => gleich oder größer

i) NOT binäre Negation

j) AND logische UND-Verknüpfung

k) OR logische ODER-Verknüpfung

l) XOR logische Exklusiv-Oder-Verknüpfung

m) EQV logische Gleichwertigkeit

n) IMP logische Ungleichheit (Implizierung)

Bei Operatoren in einer Zeile nimmt die Priorität von links nach rechts ab. Alle logische Operationen i) - n) setzen ihre Operatoren in 16-Bit-Ganzzahlwerte um. Dabei ist der Bereich von 0 - 65535 (FFFF) oder von -32768 bis 32767 erlaubt (sonst Fehlermeldung: nicht erlaubt).

2) Mathematische Funktionen

Die Klammer nach dem BASIC-Befehl dient als Hinweis, daß immer ein Ausdruck in Klammern folgen muß, der auch durch eine nummerische Variable oder durch einen arithmetischen Ausdruck ersetzt sein kann. Der berechnete Wert kann einer nummerischen Variablen zugewiesen werden.

Bei Winkelfunktionen muß der Winkel in Bogenmaß (rad) angegeben werden. Die Umrechnung erfolgt durch Division des Winkelwertes (Grad = Altgrad) mit 180 und Multiplikation mit der Zahl Pi (1 Grad entspricht 0,017453.. rad).
Die Variablen EE und PI enthalten fest gespeichert die Zahlenwerte für die Eulersche Zahl e und die Kreiszahl Pi.

a) ABS(:

Befehlsform: ABS(Zahl oder numerische Variable)

Es findet eine Absolutwertbildung statt, d.h. negative Zahlen werden positiv. Exponent und Mantisse werden nicht verändert.

Beispiel:
PRINT ABS(-523) <CR> Anzeige: 523
PRINT ABS(-5.1234 * 10^4) <CR> " : 51233.999999

Hier zeigt sich ein grundlegender Fehler der Binär-Interpreter (siehe 4) Rechengenauigkeit).

b) ATN(:

Es liegt die Umkehrfunktion des Tangens, der Arcustangens vor. Das Ergebnis wird im Bogenmaß angegeben.

Beispiel: PRINT ATN(2) <CR> Anzeige: 1.1071487178

c) COS(:

Es kann eine beliebige positive oder negative Zahl angegeben werden.
Beispiele: PRINT COS(0) <CR> Anzeige: 1
PRINT COS(60*PI/180) <CR> " : .5

d) EE:

EE ohne Argument enthält die Eulersche Zahl e, die Basis der natürlichen Logarithmen.

```
Beispiele: PRINT EE <CR>          Anzeige: 2.7182818285
```

e) EXP(:

Es wird die Exponentialfunktion Y = e^X berechnet, wobei e die Eulersche Zahl 2.718..... darstellt.

```
PRINT EXP(88) <CR>                Anzeige: 1.651636255E+38
```

Größere Zahlen X bewirken die Fehlermeldung "Ergebnis zu groß". Allerdings ist der Befehl PRINT EE^88 <CR> auch anwendbar. Bei verschiedenen Zahlen X zeigt sich dann, daß die mit beiden Befehlen für gleiche X-Werte erhaltenen Zahlen sich in der letzten Stelle unterscheiden.

```
              PRINT EE^88  <CR>   Anzeige: 1.6516362552E+38
A = 2 <CR>    PRINT EXP(A) <CR>       "   : 7.3890560989
              PRINT EE^A   <CR>       "   : 7.389056099
```

f) FIX(:

Dieser Befehl schneidet Zahlen hinter dem Dezimalpunkt ab und unterscheidet sich daher vom Befehl INT(, der die nächste größte Ganzzahl ermittelt, die kleiner oder gleich der als Argument verwendeten Zahl ist.

Beispiele:

```
A = 123.567 <CR> PRINT FIX(A) <CR>   Anzeige: 123
                   "   INT(A)  "         "     123
A = - 0.5   <CR>   "   FIX(A)  "         "   :   0
dagegen:         PRINT INT(A) <CR>       "   :  -1
```

g) INT(:

Mit dem Befehl INT(kann eine Zahl Z auf X Stellen gerundet werden.

```
E = INT(Z*10^X + 0.5)/10^X
```

h) LOG(:

Es wird der natürliche Logarithmus (Logarithmus zur Basis e) berechnet.

```
Beispiele: PRINT LOG(1000) <CR>     Anzeige: 6.907755279
           PRINT LOG(EE) <CR>       Anzeige: 1
```

i) LOG10(:

Es erfolgt die Berechnung des dekadischen Logarithmus (zur Basis 10).

```
Beispiele: PRINT LOG10(1000) <CR>  Anzeige: 3
           PRINT LOG10(EE)   <CR>      "   : .4342944819
```

j) MOD:

Befehlsform: Dividend MOD Divisor

Es wird der Rest ermittelt, der bei der Division des Dividenden durch den Divisor erhalten wird.

```
Beispiel: PRINT 9 MOD 2 <CR>        Anzeige: 1
          "     8 MOD 2 <CR>              "   : 0
```

k) PI:

Die Variable PI enthält den Wert der Kreiszahl.

```
Beispiele:  PRINT PI <CR>          Anzeige: 3.1415926536
            PRINT PI/180             "    : .01745329252
```

l) RND:

Ohne Argument liefert der RND-Befehl Pseudo-Zufallszahlen zwischen 0 und 1, die einem festgelegten Algorithmus entstammen. Dadurch ergeben sich nach dem Start des BASIC-Interpreters immer dieselben Zahlen.

```
PRINT RND <CR>                      Anzeige: .26943583396
  "    "   "                              "   : .72796806886
  "    "   "                              "   : .089675555242
```

Der Befehl RANDOMIZE ergibt einen zufälligen Startpunkt der Pseudo-Zufallszahlenfolge (Zahl aus dem Z-80-Refresh-Register R).

RND(:

Wird als Argument der Wert Null angegeben, wird erstmalig eine beliebige Zahl angezeigt und dann wiederholt.

```
PRINT RND(0) <CR>       Anzeige z.B.: .5019531548
PRINT RND(0) <CR>                     .5019531548
```

Werte größer Null entsprechen dem RND-Befehl ohne Argument. PRINT RND(1) ergibt dieselben Zahlen wie PRINT RND.

Negative Argumente bestimmen den Startpunkt der Zufallszahlenfolge. Die folgenden Zufallszahlensequenzen sind nach jedem BASIC-Start wiederum identisch.

```
A = RND(-280) <CR> PRINT RND <CR>  Anzeige:1.0207557995E-03
                   PRINT RND <CR>      "   : .94715060253
                   PRINT RND <CR>      "   : .826904723
```

Der Bereich ganzzahliger Zufallszahlen kann durch Angabe eines Anfangswertes AW und eines größeren Endwertes EW begrenzt werden.

B = INT((EW-AW)*RND+AW)

m) SGN(:

Ist der im Argument enthaltene Zahlenwert positiv, Null oder negativ, so kann dies mit dem Befehl SGN ermittelt werden. Positive Werte ergeben als Ergebnis 1, negative -1. die Zahl 0 ergibt den Wert 0. Mit diesem Befehl kann eine Dezimalzahl in eine Binärzahl umgewandelt werden.

```
Beispiel: 10 INPUT D
          20 FOR I = 7 TO 0 STEP -1
          30   PRINT SGN(X AND 2^I);
          40 NEXT
```

n) SIN(:

Das als Argument angegebene Bogenmaß wird in den Sinus umgerechnet.

Beispiel: SIN(45*PI/180) <CR> Anzeige: .70710678119

o) SQR(:

Es wird die Quadratwurzel der als Argument angegebenen Zahl berechnet. Negative Zahlen ergeben die Fehlermeldung "nicht erlaubt". Höhere Wurzeln lassen sich durch den inversen Exponenten der Zahl X berechnen.

```
Beispiele: PRINT SQR(2) <CR>          Anzeige: 1.4142135624
           PRINT SQR(5)*SQR(5) <CR>      "   : 5
           PRINT    10000^(1/4)          "   : 10 (4.WURZEL)
```

p) TAN(:

Für den im Bogenmaß angegebenen Wert wird der Tangens berechnet.

Beispiele: PRINT TAN(45*PI/180) <CR> Anzeige: 1

3) Umwandlungsfunktionen Zahl-Zeichenkette und sonstige mathematische Befehle:

a) STR$(:

Befehlsform:
STR$ (Zahl oder nummerische Variable oder Ausdruck)

Die im Argument enthaltenen Zahlen werden mit diesem Befehl als Zeichenkette gekennzeichnet.

Beispiele:

```
A$ = STR$(123456) <CR> PRINT A$ <CR>    Anzeige: 123456
PRINT ASC(LEFT$(A$,1) <CR>                  "   : 32
```

Das erste Zeichen der Zeichenkette A$ ist also ein Leerzeichen, was oft zu einem Fehlverhalten eines BASIC-Programmes führt.

b) VAL(:

Befehlsform:
VAL (Stringvariable oder Zeichenkette)

Die Fehlermeldung "Datentypen (Zahl/String) vermischt" erscheint, wenn man einer nummerischen Variablen den Inhalt einer Zeichenkette zuordnen will. Mit dem Befehl VAL ist dies möglich. Es werden nur Zahlen am Anfang einer Zeichenkette ausgewertet.

Beispiele:
A$ = "45.7" <CR> PRINT VAL(A$) <CR> Anzeige: 45.7
PRINT VAL("1234ABC") <CR> " : 1234
PRINT VAL("ABC1234") <CR> " : 0
PRINT VAL("123.45A") <CR> " : 123.45

c) HEX$(:

Diese Stringfunktion wandelt eine als Argument angegebene Dezimalzahl in einen String um, der die Sedezimalzahl enthält.

Beispiele:
PRINT HEX$(1000) <CR> Anzeige: 03E8
A = 2020 <CR> PRINT HEX$(A) <CR> " : 07E4

d) &:

Dieses Zeichen muß sedezimalen Zahlen vorangestellt werden, sonst kommt z.B. bei dem Befehl INPUT A die Fehlermeldung "* Falsche Eingabe *" (siehe auch Kapitel I). Dagegen ergibt A = 3E8 bei PRINT A den Wert 300000000, also 3*10^8.

PRINT &3E8 <CR> Anzeige: 1000

e) \ : (Ganzzahldivision: ö = Schrägstrich!)

Wird eine Division mit diesem BASIC-Befehl durchgeführt, so entfällt die Verwendung des INT(-Befehles, wenn ein ganzzahliges Ergebnis benötigt wird. Bei der deutschen Tastatur entspricht der Großbuchstabe "ö" dem Schrägstrich.

PRINT 5\3 <CR> Anzeige: 1

4) Rechengenauigkeit des Interpreters

Die folgenden Beispiele zeigen, daß dieser BASIC-Interpreter weniger Probleme mit Rundungsfehlern hat als andere (siehe mc 4/82 Grundlagen).

X = 66/20 <CR> PRINT INT(10*X)/10 <CR> Anzeige: 3.3

X = SQR(2) <CR> PRINT X^2 <CR> Anzeige: 2

PRINT LOG(0.99995) <CR> Anzeige -5.0001249077E-05

Allerdings kann aufgrund der binären Zahlendarstellung bei einer Schleife von -1 nach 1 in Schritten 0.1 der Wert Null nicht erreicht werden (Fallentest Funkschau 2/1980).

```
10 FOR N = -1 TO 1 STEP .1
20 PRINT N
30 IF N=0 THEN PRINT "N = 0"
40 NEXT
```

Nach der Anzeige der Werte -1, -.9 bis -.2 kommen -.099999999999 und 1.1368683772E-12

```
PRINT 1^100000  <CR>
```

Anzeige:1.0000063042

5) Definition eigener BASIC-Funktionen

Mit diesen Befehlen können mathematische Funktionen definiert werden, die in einem Programm mehrmals für unterschiedliche nummerische Variablen anwendbar sind. HEBAS erlaubt sogar die Verwendung von Variablen für Zeichenketten bei entsprechenden Funktionen.

DEF FN:

a) Befehlsform: DEF FN Name (Dummy-Variable) = math. Funktion

Funktionen müssen mit einem Namen gekennzeichnet sein, der mit FN beginnt und bis zu zwei weitere Zeichen (nach den Regeln für Variablen ausgewählt) enthalten kann.

Beispiele für Funktionsnamen: FN Q, FNA9, FNZZ usw.

Mehr als zwei weitere Zeichen nach FN sind möglich, werden aber nicht ausgewertet. Die Dummy-Variable dient als Platzhalter.

Beispiele für Funktionen:

```
DEF FN Q(X) = X*A
DEF FN A(X) = 1/COS(X)
DEF FN SR(V) = V/AC*20
DEF FN(X) = I+(K-1)*25+(L-1)*1000
```

Im Programm wird die am Programmanfang definierte Funktion mit dem Befehl FN Name (Variable) aufgerufen.

```
10 DEF FNQ(X) = X*A
20 INPUT A,B
30 PRINT FNQ(B)+5
40 C = FNQ(A):PRINT C
```

auch möglich:

```
S1 = T(FNA(X)) oder PRINT W$(T(FNA(X)))
```

HEBAS erlaubt auch die Verwendung von mehreren Dummy-Variablen sowie von Funktionen für die Verarbeitung von Zeichenketten. Funktionen dürfen sich auch über mehrere BASIC-Zeilen erstrecken ("multi-line"-Funktion), auch mit GOSUB-Befehlen. Dabei können andere Funktionen ebenso wie die gerade bearbeitete aufgerufen werden.

b) Befehlsform:

DEF FN Name (Dummy 1, Dummy 2, ..., Dummy n)
Funktion in mehreren BASIC-Zeilen
FNEND (Variable für Funktionswert)

Die Funktion kann jedoch nur einfache Funktionen mit der ersten Befehlsform enthalten, keine "multi-line"-Funktion. FNEND beendet die Funktion und übergibt den berechneten Ausdruck der angegebenen Variablen, die vom selben Typ (Zahl oder Zeichenkette) wie die Funktionsvariablen sein muß.

Beispiel 1)

```
10 DEF FNFAC(I)
20 IF I = 0 THEN FNRETURN 1   REM FAC(0) =1
30 FNEND FNFAC(I-1)*I         REM FAC(I) = FAC(I-1)*I
```

Beispiel 2)

```
10 DEF FN REP$(I$,I)
20 J$ = ""
30 IF I<=0 THEN FNRETURN J$
40 FOR J = 1 TO I
50 J$ = J$ + I$
60 NEXT
70 FNEND J$
80 :
90 INPUT"ZEICHEN, ANZAHL";I$,I
100 PRINT FNREP$(I$,I)
```

Der Befehl FNRETURN (Variable für Funktionswert) beendet die Funktion und übergibt einen bestimmten Wert an das Programm. Mehrere FNRETURN-Befehle sind erlaubt.

** Achtung **
Ein Syntaxfehler in der Definition der Funktion wird erst beim Aufruf der Funktion erkannt, wobei dann aber die Zeilennummer der BASIC-Zeile ausgegeben wird, in der der Funktionsaufruf steht und nicht die Zeile mit der Definition.

Wird die Funktion aufgerufen, ohne daß eine Definition der Funktion stattgefunden hat, erfolgt die Fehlermeldung "USER FN CALL fehlt Zeile XY".

VI. Dateistrukturen und sonstige Ein- und Ausgabefunktionen

Moderne Textverarbeitungssysteme erübrigen oft umständliche Eingabe- und Ausgaberoutinen in BASIC. Eine Adressendatei z.B. läßt sich mit einem Textprogramm leicht anlegen, verändern und drucken. Aber schon ein Sortiervorgang oder eine spezielle Dateiart erfordern eine Bearbeitung mit einem eigens dafür entworfenen Programm.

Der BASIC-Interpreter HEBAS bietet hierfür umfangreiche Befehle.

A) OPEN:

Befehlsform:
OPEN # Kanal,"Modus","Laufwerk:Dateiname.Dateizusatz",Satzlänge

1) # Kanal:

Für die Zusammenarbeit des Computers mit Peripheriegeräten werden Verbindungskanäle benötigt. Diese werden mit Nummern von 0 bis 255 gekennzeichnet, wobei die für CP/M übliche Kennzeichnung verwendet wurde.

Tabelle der Kanalnummern:

Konsole	0
Laden und Speichern von Programmen	1
Drucker (List-Device)	2
Leser (Reader)	3
Stanzer (Punch)	4
für spätere Erweiterungen	5-9
Floppy	10-255

Prinzipiell können alle Ein- bzw. Ausgabefunktionen über irgendeinen sinnvollen Kanal ablaufen. Eine Ausgabe von Zeichen auf die Konsole ist natürlich ebenso sinnlos wie die Ausgabe an die Leser-Schnittstelle. Laden und Speichern von Programmen kann nur über Kanal 1 ablaufen. Einem nicht angegebenen Kanal wird die Nummer 0 zugewiesen. Folgt einer Kanalzuweisung ein weiteres Argument, muß dies durch ein Komma von der Kanalnummer abgetrennt sein. Schließen eines nicht offenen Kanales bewirkt die Fehlermeldung "Datei nicht offen".

Kanalmerkmale:

Die Voreinstellungen der Kanalparameter sind beim Befehl OPTION erläutert. Die folgenden Erläuterungen gelten für die IOBYTE-Einstellung L = L nach dem Start des Monitors.

a) Kanal #0

Dieser Kanal ist immer offen. Der Befehl OPEN#0,"O" <CR> gibt ein Formfeed (0C) aus, d.h. der Bildschirm wird gelöscht. Außerdem werden die mit dem Befehl OPTION veränderten Voreinstellungen rückgängig gemacht.
Der Befehl CLOSE#0 <CR> gibt ebenfalls das Zeichen 0C aus und der Kanal bleibt offen.

b) Kanal #1

Laden oder Speichern von Programmen läuft über diesen fest eingestellten Kanal ab, was nicht verändert werden kann. Eine Kanalangabe wird beim LOAD- und SAVE-Befehl nicht berücksichtigt, bei RESAVE wird "Syntaxfehler" ausgegeben.

c) Kanal #2

Der Kanal zum Drucker ist immer offen. Der Befehl OPEN#2,"O" <CR> gibt ein Formfeed aus; die Kanalparameter (falls mit dem Befehl OPTION verändert) werden zurückgestellt. CLOSE#2 gibt ein Formfeed aus, der Kanal bleibt offen.

d) Kanal #3

Der Reader-Kanal ist immer offen. Der Befehl OPEN#3,"I" <CR> stellt die Kanalparameter zurück. CLOSE#3 hat keine Auswirkung. Mit dem Befehl INPUT#3,A$ kann z.B. über die Tastatur ein Wort eingegeben werden, ohne daß die Zeichen auf dem Bildschirm erscheinen.

e) Kanal #4

Der Punch-Kanal ist immmer offen. Der Befehl OPEN#4,"O" gibt 16 Bytes mit dem Wert 255 (FF) aus und stellt die Kanalparameter zurück. CLOSE#4 gibt ein CTRL-Z (1A) aus, gefolgt von 16 Bytes FF; der Kanal bleibt offen.

f) Die Kanäle 5 - 9 sollten nicht verwendet werden.

g) Kanäle 10 - 255

Für Diskettenlaufwerke sind die Kanäle 10 bis 255 verwendbar. Jeder Ausgabekanal wird mit einer eigenen Datei verbunden, deren Name im OPEN-Befehl angegeben wird. Die maximale Anzahl der durch ein Programm gleichzeitig geöffneten Kanäle wird durch den CLEAR-Befehl bestimmt (siehe dort).

2) Modus:

Für die Verbindungskanäle zwischen Computer und Diskettenlaufwerk gibt es verschiedene Ausführungsarten, die durch die Großbuchstaben I, O, U und R abgekürzt, in Anführungszeichen gesetzt und mit Kommas von anderen Argumenten abgegrenzt werden müssen. Der Modus kann auch durch eine Stringvariable angegeben werden (ansonsten Fehlermeldung "OPEN-Modus falsch").

a) I = Eingabe-Modus (Lese-Modus)

Diese Datei-Eröffnungsart wird für sequentielle Lesevorgänge von der Diskette benötigt. Die mit dem Dateinamen angegebene Datei wird auf der Diskette im aktuellen (unter CP/M angemeldetem) oder eigens bezeichnetem Laufwerk gesucht und die Kanalverbindung hergestellt. Fehlt die Datei, erfolgt Fehlermeldung "Datei nicht gefunden". Der CLOSE-Befehl für einen solchen Kanal hat keine Funktion.

b) O = Ausgabe-Modus (Schreib-Modus)

Sequentielle Schreibvorgänge erfordern diesen Modus. Die angegebene Datei wird wie beim I-Modus gesucht. Ist sie vorhanden, erfolgt die Fehlermeldung "Datei existiert Zeile XY", wenn nicht vorhanden, wird die Kanalverbindung hergestellt. Ein CLOSE nach einem solchen OPEN-Modus bewirkt die Ausgabe von CTRL-Z (1A), Leerung aller Zwischenspeicher und die Aktualisierung des Diskettenin haltsverzeichnisses.

c) U = Ergänzungsmodus (Lese- und Schreibmodus)

Lesen und Schreiben sind über einen Kanal möglich. Bei diesem Modus wird bei nicht gefundener Datei die Fehlermeldung

"Datei nicht gefunden Zeile XY" angezeigt. ansonsten die Verbindung hergestellt. Jeder Schreibvorgang beginnt an der gleichen Stelle wie der vorhergehende Lesevorgang, jeder folgende Lesevorgang beginnt mit dem ersten, noch nicht bearbeitetem Byte. CLOSE ergibt eine Leerung der Zwischenspeicher und die Aktualisierung des Inhaltsverzeichnisses.

d) R = Random-Modus (Lese- und Schreibmodus, wahlfrei)

Hier kann sowohl sequentiell als auch mit wahlfreiem Zugriff geschrieben oder gelesen werden. Fehlt die angegebene Datei, wird sie neu angelegt. Ist sie vorhanden, wird sie ergänzt oder an den bezeichneten Stellen überschrieben (wahlfreier Zugriff), ohne daß Fehlermeldungen wie bei den anderen Zugriffsarten erfolgen. CLOSE leert die Zwischenspeicher und aktualisiert das Verzeichnis.

Allgemein:

Ein OPEN-Befehl setzt den Byte-Zeiger der Datei auf das erste Byte des Dateiinhaltes. Dies kann ohne Fehlermeldung bei der gleichen Datei beliebig oft wiederholt werden.

Ein CLOSE-Befehl schließt die Bearbeitung der Datei ab und trennt die Verbindung zur Datei. Mit dem CLEAR-Befehl reservierter Speicherplatz bleibt reserviert.

3) Laufwerk:Dateiname.Dateizusatz

Fehlt die Laufwerksangabe, wird die Datei auf der Diskette in dem unter CP/M angemeldetem Laufwerk gesucht. Sternchen im Dateinamen können bei Schreibvorgängen nicht als Abkürzung verwendet werden (Auffüllung mit Fragezeichen). Fehlt der Dateizusatz, wird er durch .BAS ergänzt. Die drei Parameter dürfen auch durch Variableninhalte bestimmt werden.

4) Satzlänge:

Diese Zahl bestimmt die Länge eines einzelnen Datensatzes und muß als Ganzzahl im Bereich von 1 bis 32767 eingegeben werden. Beim R-Modus wird sie sonst gleich 1 gesetzt, bei den anderen Modus-Arten bleibt sie unberücksichtigt.
Um auf beliebige Teile einer Datei zugreifen zu können, ohne daß die ganze Datei in den Kernspeicher geladen wird, benötigt man die Adresse des Anfangsbytes des gesuchten Dateiteiles. Diese wird intern durch Multiplikation einer Satznummer (1 - 4194303) mit der Satzlänge errechnet. Da jedoch die Satzlänge auch die Trennzeichen (es sind dies die Bytes 22h, 0A und 0D) der Dateisätze beinhaltet (siehe OPTION), muß zur eigentlichen Datensatzlänge noch der Wert 4 dazu addiert werden, sonst wird nicht auf den Anfang der Datensätze zugegriffen. Die gesamte Dateilänge (Summe der Satzlängen) kann unberücksichtigt bleiben oder man beschreibt beim Anlegen der Datei alle Datensätze mit einem Leerstring.

B) Für Schreib- bzw. Lesevorgänge werden zunächst nur noch folgende Befehle benötigt.

a) PRINT:

Befehlsform:
PRINT # Kanal § Satznummer, Variable 1, Variable 2, usw.

Die Satznummer wird durch den vorangestellten Klammeraffen "§" gekennzeichnet, beide werden nur für Schreibvorgänge bei wahlfreiem Zugriff benötigt. Nummerische Variablen (Zahlen), Textvariablen und Ausdrücke sind erlaubt.

b) Input:

Befehlsform:
INPUT # Kanal § Satznummer, Variable 1, Variable 2, usw.

Bis zu 250 Zeichen können über den angegebenen Kanal eingelesen werden (sonst Fehlermeldung "Record zu lang Zeile XY"). Um auch Kommas einlesen zu können, bedarf es des Befehles INPUT LINE # Kanal ... . Die mit dem Befehl OPTION angegebenen Trennzeichen werden mitgelesen, ausgenommen die ASCII-Zeichen 32h, 34h und 27h.

Es folgen nun Beispiele zur sequentiellen sowie zur wahlfreien Dateibearbeitung.

1) sequentiell schreiben

```
 5 CLEAR 250
10 OPEN#15,"O","A:TEST.DAT"
20 INPUT A$:PRINT#15,A$
30 CLOSE#15
```

Es wird die Datei TEST.DAT über den Kanal Nr. 15 erstmalig im Ausgabe-Modus eröffnet (Fehlermeldung, wenn die Datei TEST.DAT schon im Inhaltsverzeichnis der Diskette steht, siehe auch den Befehl LOOKUP im Abschnitt C). Nach der Eingabe bis zu 250 Zeichen ohne Kommas in die Variable A$ wird diese über den Kanal Nr. 15 ausgegeben, auf die Diskette im Laufwerk A geschrieben und mit CLOSE#15 die Bearbeitung abgeschlossen.
Wurde z.B. die Zeichenkette AAA* eingegeben, so werden beim Schreibvorgang folgende Zeichen (sedezimal) auf Diskette geschrieben:

22 41 41 41 2A 22 0D 0A 00 00 00 00 00 usw., auf 128 Bytes mit Nullen aufgefüllt.

Mehrere Eingaben können auch über eine FOR-NEXT-Schleife mit indizierten Variablen, z.B. A$(I), durchgeführt werden.

2) sequentiell lesen

```
 5 CLEAR 250
10 OPEN#15,"I","TEST.DAT"
15 ON EOF GOTO 30
20 INPUT#15,A$:PRINT A$
30 CLOSE#15
40 END
```

Die in der Datei TEST.DAT enthaltene Zeichenkette wird in die Variable A$ über den Kanal Nr. 15 eingelesen und auf den Bildschirm ausgegeben.

In Zeile 20 beider Beispiele können, durch ein Komma getrennt, auch mehrere Variablen stehen:

a) Schreiben:
```
20 INPUT A$,B$,C$,D$:PRINT#15,A$,B$,C$,D$
```

b) Lesen:
```
20 INPUT#15, A$,B$,C$,D$:PRINT A$,B$,C$,D$
```

3) sequentiell lesen und schreiben

```
 5 CLEAR 250
10 OPEN#25,"U","TEST.DAT"
20 INPUT#25,A$:PRINT A$
30 INPUT"RICHTIG";X$
40 :  IF X$ ="NEIN" THEN INPUT A$
50 PRINT#25,A$:GOTO 20
60 END
```

Aus einer bestehenden Datei wird jeweils eine Zeichenkette gelesen und falls für richtig befunden, wieder auf Diskette geschrieben. Nach dem Lesen der letzten Zeichenkette der Datei ergibt ein weiterer Lesezugriff die Fehlermeldung "Dateiende falsch Zeile 20". Dies kann durch eine Programmzeile 25 ON EOF GOTO 60 abgefangen werden.

4) wahlfrei schreiben

```
 5 CLEAR 250
10 OPEN#10,"R","TEST.REC",104
20 FOR I = 1 TO 5
30 INPUT A$(I):PRINT#10%I,A$(I)
40 NEXT:CLOSE#10
```

Es werden 5 Datensätze mit einer Länge von 100 Zeichen in der Datei TEST.REC abgelegt.

5) wahlfrei lesen

```
 5 CLEAR 250
10 OPEN#250,"R","TEST.REC",104
20 INPUT"Satznummer";SN: IF SN = 0 THEN END
30 INPUT#250%SN,A$:PRINT A$: GOTO 20
40 CLOSE#250
```

Jeder der mit Beispiel Nr.4 eingebenen fünf Datensätze kann beliebig gelesen werden. Hier zeigt sich die Grenze solcher einfacher Beispiele. Eine falsche Datennummer SN ergibt falsche Zeichenketten oder die Fehlermeldung "Dateiende falsch Zeile XY". Dies muß durch das Programm abgefangen werden.

C) Weitere Befehle zur Dateiverarbeitung:

1) BYTE(:

Befehlsform: BYTE (# Kanal)

Damit können über den angegebenen Kanal sequentiell die dezimalen Werte der Bytes einzeln gelesen werden. Das Nummernzeichen "#" kann entfallen.

Beispiel:

```
 5 CLEAR 250
10 OPEN#15,"I","A:TEST.DAT"
20 B = BYTE(#15):PRINT B;
30 ON EOF GOTO 100
40 GOTO 20
50 :
100 CLOSE#15:PRINT"ENDE"
```

Es erfolgt die Ausgabe der Datei TEST.DAT zeichenweise in dezimalen Werten, z.B. der Datensatz "AAA" ergibt die Zeichenfolge 34 65 65 65 34 13 10 0 0 0 0 0 0 0 usw. bis 128 Bytes gelesen wurden.

Mit einer Programmzeile B = BYTE(#0) ist eine Tastaturabfrage möglich. Der Interpreter hält mit der Programmausführung an, bis eine Taste gedrückt wurde. Die Variable B enthält den dezimalen Wert der Taste, das Zeichen erscheint nicht auf dem Bildschirm.

```
10 PRINT"TASTE DRÜCKEN:";
20 B = BYTE(#0)
30 PRINT"ZEICHEN DEZ.:";B
40 :  IF B = 5 THEN STOP ELSE GOTO 20
```

Da auch Steuerzeichen wie <CTRL-E> als dezimaler Wert (5) ausgegeben werden, muß zum Programmabbruch die Zeile 40 vorhanden sein.

Weiterhin ist die Verwendung des Befehles auch in Verbindung mit dem Zusatz ON möglich:

```
 10 ON BYTE(#0) GOTO 100,200
 20 STOP
100 PRINT"1. Sprung"
110 STOP
200 PRINT"2. Sprung"
```

Bei Eingabe von <CTRL-A> erfolgt ein Sprung zur Zeile 100, bei <CTRL-B> ein Sprung nach 200, da die beiden Eingaben die dezimalen Werte 1 und 2 besitzen.

2) BYTE$(

Befehlsform: BYTE$(# Kanal , Zeichenanzahl)

Damit kann jetzt ein ganze Zeichenkette auf einmal eingelesen werden. Das Nummernzeichen "#" kann entfallen. Die Trennzeichen (" oder mit OPTION verändert) der Zeichenkette

werden mitgelesen und angezeigt. Eine fehlende Zeichenanzahl wird durch die Zahl 1 ersetzt.

```
 5 CLEAR 250
10 OPEN#10,"I","A:TEST.DAT"
20 B$ = BYTE$(#10):PRINT B$;
30 ON EOF GOTO 100
40 GOTO 20
50 :
100 CLOSE#10:PRINT"ENDE"
```

Die mit dem Beispiel Nr.1, Abschnitt B) geschriebene Datei wird gelesen und beim Zeichen CTRL-Z (1A) zur Zeile 100 gesprungen.

Ausgabebeispiel: "aaaaaaaaaa"
"cccccccccccccc"

Der Befehl BYTE$(#0,1) vereinfacht Eingaberoutinen, die nur bestimmte Zeichen als Eingabe zulassen.

```
10 PRINT"Bitte die Taste <F> drücken"
20 B$ = BYTE$(#0,1): IF B$ <> "f" THEN 20
30 PRINT"Fertig"
```

3) OUTBYTE:

Befehlsform:
OUTBYTE # Kanal § Satznummer, Zahl oder Variable oder Zeichenkette

Es sind Zahlen von 0 bis 255 erlaubt. Von einer Zeichenkette wird jedes Zeichen als einzelnes Byte ausgegeben. Die Länge der Zeichenkette wird nicht wie beim WRITE-Befehl angegeben. Steuerzeichen wie CTRL-I und andere lassen sich so an Daten anhängen.

Beispiel:

```
PRINT#12,A$;:OUTBYTE#12,9:PRINT#21,B$;:OUTBYTE#12,9
```

Diese Befehle fügen beim Schreibvorgang in eine Datei an A$ und B$ das Zeichen CTRL-I (9) an.

Weiterhin lassen sich mit OUTBYTE(#0) Zahlenkolonnen ohne Zwischenraum ausgeben wie es bei dem Befehl PRINT nicht der Fall ist.

```
 5 A = 1
10 FOR I = 1 TO 10
20 PRINT A;                    Ausgabe: 1 1 1 1 1 1 1 1 1 1
30 NEXT
```

dagegen:

```
 5 A = 49
10 FOR I = 1 TO 10
20 OUTBYTE#0,A                 Ausgabe: 1111111111
30 NEXT
```

4) LOC(:

Befehlsform: LOC(# Kanal)

Der Befehl gibt nach einem Schreib- oder Lesebefehl die Stellung des Byte-Zeigers an, d.h. die Zahl der insgesamt übertragenen Bytes einer Zeichenkette einschließlich Trennzeichen. Wurde z. B. mit dem Programm Nr.1 im Abschnitt B) eine Zeichenkette mit 10 Zeichen auf Diskette geschrieben, so gibt der Befehl PRINT LOC(#15) als Zeile Nr. 25 die Zahl 14 aus. Diese Zahl ist um 4 größer als die Länge der Zeichenkette (siehe auch Abschnitt A), Satzlänge). Das Nummernzeichen "#" kann entfallen. Wurde z.B. mit dem Befehl OPTION #10,"Q",0 das Trennzeichen zwischen einzelnen Datensätzen abgeschafft, so ist diese Zahl nur um 2 größer.

5) SETLOC:

Befehlsform:
SETLOC # Kanal § Bytenummer, # Kanal § Bytenummer, usw.

Dies ist das Gegenstück zum LOC(-Befehl. Es wird die Stelle in einem Datensatz angegeben, ab der ein Zugriff erfolgen soll. So kann bei sonst sequentieller Dateiverarbeitung in einem Datensatz wahlfrei gelesen oder geschrieben werden.

SETLOC#15§5 bedingt einen Zugriff ab dem fünften Byte des Datensatzes, setzt also den Byte-Zeiger auf das 5.Byte des Datensatzes der mit Kanal Nr. 15 eröffneten Datei. Man erspart sich so umständliche Stringoperationen bei sequentiellen Dateien.

Beispiel:

```
10 OPEN#10,"I","A:TEST.DAT"
20 ON EOF GOTO 70
30 INPUT"Byte-Nummer";BN
40 SETLOC#10§BN
50 PRINT BYTE(#10,2)
60 GOTO 30
70 END
```

Ab dem angegebenen Byte werden zwei Zeichen gelesen und angezeigt (siehe BYTE-Befehl).

6) LOF(:

Befehlsform: LOF(# Kanal)

Es wird die von einer Datei auf einer Diskette belegte Byteanzahl (immer Vielfache von 128) ausgegeben, wobei zuvor diese Datei über einen Kanal eröffnet werden muß.

Beispiel:

```
10 INPUT "Dateiname";A$
15 IF A$="e" THEN END
20 OPEN#10,"I",A$
30 PRINT LOF(#10)
```

```
40 GOTO 10
```

7) LOOKUP(:

Befehlsform: LOOKUP("Laufwerk: Dateiname. Dateizusatz")
oder LOOKUP(Stringvariable)

Wird mit dem O-Modus eine Datei eröffnet, so wird die Fehlermeldung "Datei existiert Zeile XY" ausgegebenen, falls eine Datei mit gleichem Namen schon vorhanden ist. Mit LOOKUP(kann dies abgefragt und mit dem ERASE-Befehl die Datei gelöscht werden. Der Befehl liefert den Wert -1, wenn die Datei vorhanden bzw. 0, wenn sie nicht vorhanden ist. Der Ausdruck in Klammern kann auch durch Variablen angegeben werden.

So kann beim Beispiel Nr.1, Abschnitt B) die Zeile Nr. 7 eingefügt werden.

```
7 IF LOOKUP("A:TEST.DAT") THEN ERASE "A:TEST.DAT"

10 IF NOT LOOKUP(LW$+":"+D1$) THEN .........
```

8) WRITE:

Befehlsform:
WRITE # Kanal § Satznummer , Variable oder "Zeichenkette"

Dieser Befehl kann nur für binäre Ausgabeeinheiten verwendet werden (Kanal Nr.0 und Nr.2 ergeben die Fehlermeldung "keine Binäreinheit). Die Länge der in der Variablen enthaltenen Zeichenkette wird als erstes Byte vor den Zeichen der Zeichenkette auf Diskette geschrieben. Bei den im Abschnitt B) angegebenen Beispielen Nr.1 und 4 kann der Befehl PRINT # Kanal durch WRITE # Kanal ersetzt werden.

Der Befehl WRITE #4 ohne weitere Argumente dient zur Ausgabe einer aus acht Bytes bestehenden Zeichenkette (FF FF FF FF FF FF FF 00), die als "data header" beim Kanal Nr. 4 verwendet wird, der vorher mit OPEN#4,"O" eröffnet werden muß (sonst Fehlermeldung "OUTPUT nicht offen).

9) READ:

Befehlsform: READ # Kanal § Satznummer , Variable

Nur von binären Eingabeeinheiten (nicht die Tastatur) kann mit diesem Befehl gelesen werden (sonst Fehlermeldung "keine Binäreinheit). Es werden soviele Zeichen eingelesen wie im ersten Byte der Zeichenkette angegeben sind.

READ #3 ohne Argumente liest solange Zeichen, bis ein "data header" gefunden wurde.

10) MAT WRITE:

Befehlsform:
MAT WRITE # Kanal § Satznummer , Matrix 1, Matrix 2, usw.

Mit dem Befehl können ohne großen Programmieraufwand ein- oder zweidimensionale nummerische Matritzen über den angegebenen Kanal ausgegeben werden (nur Binäreinheiten).

Beispiel:

```
 5 DIM A(3,5)
10 FOR I = 1 TO 3
20 FOR J = 1 TO 5
30 INPUT A(I,J)
40 NEXT J,I
50 :
60 OPEN#10,"O","A:TEST.MAT"
70 MAT WRITE#10,A
80 CLOSE#10
```

Nach der Eingabe beliebiger Zahlen wird die Matrix ohne FOR-NEXT-Befehle direkt über Kanal Nr. 10 in die Datei TEST.MAT geschrieben.

11) MAT READ:

Befehlsform:
MAT READ # Kanal § Satznummer , Matrix 1, Matrix 2 usw.

Die mit obigem Beispiel erstellte Matrix wird mit diesem Befehl wieder gelesen.

```
10 DIM A(3,5)
20 OPEN#10,"I","A:TEST.MAT"
30 MAT READ#10,A
40 CLOSE#10
50 REM Anzeige der Matrix
60 FOR I = 1 TO 3
70 FOR J = 1 TO 5
80 PRINT A(I,J);
80 NEXT J :PRINT: NEXT I
```

Zum Einlesen der Matrix muß diese vorher dimensioniert werden. Eine Dimensionierung, die von der beim Schreibvorgang vorhandenen abweicht, ergibt Lesefehler.

D) Fehlerbehandlung:

1) ERROR:

Befehlsform: ERROR (Fehlernummer)

Wie schon im Abschnitt III bei den Direktbefehlen erläutert, kann mit dem Befehl ERROR XY (0-48) die Fehlermeldung XY ausgegeben werden.

Fehlermeldungen sind aber auch beliebig definierbar, wobei die Texte den Nummern 0 - 255 zugeordnet werden können. Entsprechend der gewünschten Abbruchbedingung des Programmes wird die Fehlernummer angegeben.

Beispiel: 30 IF A > 5 THEN ERROR 50

Wird während des Programmlaufes die Zeile 30 erreicht und der Wert der Variablen A ist größer als 5, dann wird die Zahl 50 als Fehlernummer gespeichert. Zu Beginn des Programmes setzt man einen Sprungbefehl:

10 ON ERROR GOTO 1000

Im Unterprogramm ab Zeile 1000 kann dann der zugehörige Text ausgegeben werden.

1000 IF ERR = 50 THEN PRINT "A zu groß": GOTO ....

2) ERR

a) Befehlsform: ERR = Fehlernummer

Die Variable ERR enthält die Fehlernummer, die mit der IF-Bedingung abgefragt werden kann.

Beispiel: 1000 IF ERR = 32 THEN ERASE "A:TEST.BAS"

Falls das Programm TEST.BAS existiert, wird es gelöscht.

b) Befehlsform: ERR(# Kanal)

Tritt ein Fehler bei der Benützung des angegebenen Kanals auf, so enthält die Funktion ERR(#Kanal) den Wert -1, sonst 0.

3) ERL :

Befehlsform: ERL = Zeilennummer

Die Variable ERL enthält die Zeilennummer der Programmzeile. in der der Fehler auftrat. Ein Fehler bei der Ausführung eines Direktbefehles ergibt den Wert 65535 für ERL.

4) RESUME:

Soll nach der Fehlerbehandlung in einem Unterprogramm mit der normalen Programmausführung fortgefahren werden, muß der Befehl RESUME das Unterprogramm abschließen. Dieser Befehl entspricht dem RETURN nach einem GOSUB.

a) Befehlsform: RESUME

Es wird der Befehl wiederholt, der den Fehler verursachte.

b) Befehlsform: RESUME Zeilennummer

Das Programm wird mit der angegebenen BASIC-Zeile fortgesetzt.

c) Befehlsform: RESUME NEXT

Das Programm wird nach dem Befehl fortgesetzt, der den Fehler verursachte.

Am Anfang eines Unterprogrammes zur Fehlerbehandlung kann auch ein weiterer ON ERROR GOTO-Befehl stehen.

Franzis'