Video Genie System

BASIC MANUAL

INHALTSVERZEICHNIS

EINFUEHRUNG

Es gibt im VIDEO GENIE System vier Funktionseben:

(1) Die aktive Befehlsebene: Auf dieser Ebene antwortet der Computer auf Befehle, direkt wenn sie eingegeben und durch die Taste NEW LINE abgeschlossen worden sind. Immer wenn > – Zeichen auf dem Schirm zu finden sind, ist der Benutzer auf der aktiven Befehlsebene. (fuer naehere Beschreibung siehe Kapitel 1)

(2) Die Programmausfuehrungsebene: Diese Ebene wird durch Eingabe des RUN-Befehls betreten, und damit wird das BASIC Programm, das sich im Speicher befindet ausgefuehrt. Alle Numerischen Variablen werden auf Null gesetzt und allen Stringvariablen wird der leere String (Null String) zugewiesen. Dann beginnt die Programmausfuehrung, die mit dem RUN Befehl gestartet wurde. (fuer naehere Beschreibung des RUN-Befehls siehe Kapitel 1)

(3) Die Text Editierungsebene: Diese Ebene erlaubt es dem Benutzer, gespeicherte Programme zu aendern, zu loeschen oder Text zu ihnen hinzuzufuegen. Der Benutzer muss fehlerhafte Zeilen nicht mehr ganz neu schreiben, sondern kann die falschen Stellen korrigieren. (siehe Kapitel 2)

(4) Die Monitorebene: Diese Ebene erlaubt es dem Benutzer, maschinensprachliche "Objekt Dateien" in den Speicher zu laden. Auf diese Objekt Dateien koennen andere BASIC Programme zugreifen, oder sie koennen auch voellig unabhaegig von BASIC Programmen ausgefuehrt werden. Sie koennen Daten oder Maschinenprogramme enthalten. (siehe SYSTEM Befehl in Kapitel 1)

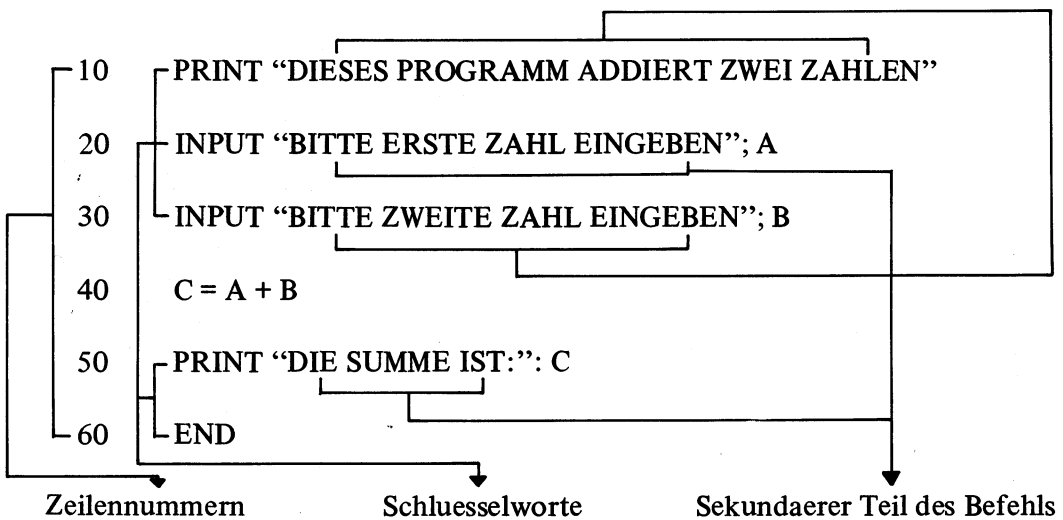
Bevor wir beginnen, Programme zu schreiben, sollten wir uns mit einigen Grundlagen der Programmierung bekannt machen.

1. Schluesselworte: Es gibt im erweiterten BASIC eine Menge von Schluesselworten, die das Skelett eines Programms bilden. Einige von ihnen sind:

   PRINT
   INPUT
   IF
   THEN
   GOTO
   END

Eine vollstaendige Liste der Schluesselworte finden Sie in Anhang A. Diese Schluesselworte bilden die Struktur-elemente eines Programms.

Beispiel:



Dieses Programm liest zwei Zahlen, addiert sie und gibt das Ergebniss aus.

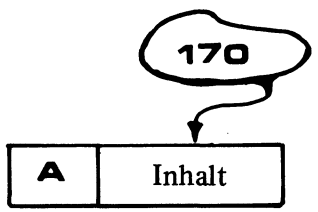
```
READY
>RUN

DIESES PROGRAMM ADDIERT ZWEI ZAHLEN
BITTE ERSTE ZAHL EINGEBEN ? 42
BITTE ZWEITE ZAHL EINGEBEN ? 45
DIE SUMME !ST: 87
```

2. Variablen: Sie kennen sicher Ihre Hausnummer. Sie dient als Marke, um Ihr Haus von den anderen in Ihrer Strasse zu unterscheiden. Mit dieser Angabe kann der Brieftraeger Ihre Post in den richtigen Briefkasten befoerdern.

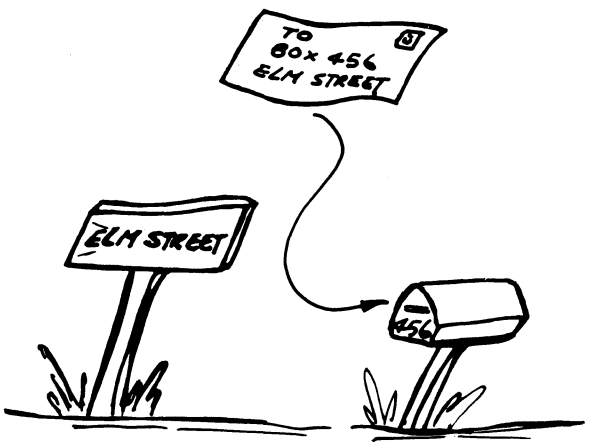
Die Variablennamen haben die gleiche Funktion wie die Hausnummern. Nur statt mit Zahlen arbeiten sie mit Buchstaben.

Sehen Sie sich folgenden Befehl an:
A = 170



Variablen-
name

(a)



(b)

Bei dem Ereignis (a), bei dem der Computer den Befehl A = 170 ausfuehrt, sucht er die Stelle im Speicher, wo die Variable A steht und schreibt 170 hinein. Genauso sieht es bei dem Ereignis (b) aus. Der Brieftraeger sieht die Addresse auf dem Brief und bringt ihn zum Briefkasten Elmstr. 470. Ein einfacher Vorgang.

Sehen Sie sich bitte unter dem gleichen Aspekt folgendes Programm an:

```
10 A = 1          : REM  SCHREIBE EINE 1 NACH A
20 A = A + 10     : REM  ADDIERE 10 AUF A, SCHREIBE ERGEBNIS NACH A
30 B = A * 2      : REM  A MAL 2, ERGEBNIS NACH B
40 PRINT"DIE ERGEBNISSE SIND:"; A, B
50 END            : REM  PROGRAMMENDE
```

Zeile 10	A	1
Zeile 20	A	11
Zeile 30	A	11
	B	22

```
READY
>RUN
DIE ERGEBNISSE SIND: 11          22
```

Bis jetzt haben wir nur Variablen behandelt, die Zahlen enthalten. Aber Variablen koennen auch Buchstaben (engl. Characters) oder ganze Zeilen von Buchstaben (engl. Strings) enthalten.

Diese Variablen sehen aber etwas anders aus, als die Numerischen Variablen.

```
10  A$ = "HERR JUPP ADAM"
20  B$ = "MUSEUMSTR. 12"
30  C$ = "5300 BONN"
40  PRINT A$
50  PRINT B$
60  PRINT C$
70  END

READY
>RUN

HERR JUPP ADAM
MUSEUMSTR. 12
5300 BONN
```

Dieses Programm weist den Variablen A$, B$, C$ die Werte "HERR JUPP ADAM", "MUSEUMSTR. 12", 5300 BONN" zu. Dann werden die Inhalte dieser Variablen auf den Schirm ausgedruckt. Beachten Sie bitte, dass diese Zeichenketten (engl. String) Variablen durch ein $ hinter ihrem Namen gekennzeichnet werden. Durch dieses $ weiss der Computer, dass die entsprechende Variable nicht nur Zahlen, sondern zu den Zahlen noch Buchstaben enthalten darf. Die Zeichenketten, die diesen Variablen zugewiesen werden, muessen in Hockhommata eingeschlossen werden. Beispiel: D$ = "ABCDE 12345 *=?+"

Beachten Sie bitte: Wenn sie einem Variablentyp Werte von einem falschen Typ zuweisen, wird der Computer das nicht verstehen und eine Fehlermeldung ausgeben.

Beispiel:
A = "FALSCHER DATENTYP" (A ist numerische Variable)
B$ = 100 (B$ ist Zeichenketten Variable)

An Rande bemerkt, Variablennamen muessen einheitlich sein. Zwei Haeuser koennen nicht die gleiche Hausnummer in der selben Strasse haben.

Das Video Genie System akzeptiert Variablen, die laenger als zwei Zeichen sind. Es benutzt aber nur die ersten zwei Zeichen, um Variablen zu unterscheiden (PR und PREIS werden als gleich aufgefasst).
Ein Variablenname muss mit einem Buchstaben beginnen (A bis Z), worauf ein anderer Buchstabe oder eine Zahl (0 bis 9) folgen kann.
Folgende Variablennamen sind fuer den Computer richtig und auch unterscheidbar:
A, AA, AB, AC, A0, A1, BN, BZ, B7, ZZ Z1 usw.

Beachte: Variablennamen sollten k e i n e Schluesselworte der BASIC Sprache enthalten. Z.B. ist in dem Namen CIF das Schluesselwort IF enthalten. Eine Liste aller Schluesselworte finden Sie im Anhang A.

Typen von Variablen

Das Video Genie System arbeitet mit vier verschiedenen Typen von Variablen. Die ersten drei operieren mit numerischen Werten, die letzte speichert Zeichenketten.

1. % : Ganzzahlig (engl. integer), Zahlbereich von -32768 bis + 32767

   Beispiel: A% = -30
   BB% = 8000
   N i c h t : X% = 3.141

2. ! : einfache Genauigkeit (6 Stellen)

   Beispiel: A ! = -50.33
   D4 ! = .1241

3. # : Doppelte Genauigkeit (16 Stellen)

   Beispiel: A# = 3.141592653589
   A2# = -4507.8976554

4. $. Zeichenketten (strings) (maximale Laenge: 255 Zeichen)

   Beispiel: A$ = "VIDEO GENIE SYSTEM"
   M2$ = "DAS ERGEBNISS VON (A*B+15)/2.5 IST:"

Obwohl A%, A!, A# und A$ alle das "A" im Variablennamen haben, sind es trotzdem fuer den Computer verschiedene Variablen mit verschiedenen Werten.

Arithmetische Operatoren

Immer wenn ein Programm einen Wert berechnen muss, werden arithmetische Operatoren benutzt.

Beispiel:

```
5   R = 6
10  A = R * 3.1416 * 2 : REM BERECHNE KREISUMFANG
20  PRINT "DER KREISUMFANG IST:"; A
30  B = 3.1416 * R [ 2 : REM BERECHNE KREISINHALT
40  PRINT "DIE KREISFLAECHE IST:";B
50  END

READY
>RUN

DER KREISUMFANG IST : 37.6992
DIE KREISFLAECHE IST : 113.098
```

Das Video Genie System benutzt gaengige arithmetische Symbole:

+ fuer Addition, – fuer Subtraktion, * fuer Multiplikation / fuer Division und [(ESC Taste) fuer Expotentiation. Zum Beispiel ist das Ergebniss von 5 mal 12 hoch 1/3 im Video Genie System equivalent zu 5 * 12 [(1/3).
Beachte: Sie werden keine [-Taste auf der Tastatur finden, sie wird durch die ESC-Taste dargestellt.

Relationsoperatoren

Immer wenn ein Programm eine Entscheidung treffen soll, (bedingter Sprung) werden Relationsoperatoren gebraucht. Es stehen folgende zur Verfuegung:

<	kleiner als	< =	kleiner oder gleich
>	groesser als	> =	groesser oder gleich
<>	ungleich	=	gleich

Beispiel:

```
120 IF A<B THEN PRINT "B IST GROESSER ALS A"
```

Wenn der Computer diesen Befehl ausfuehrt und an diesem Punkt der Inhalt von B groesser ist als der von A (d.h die Bedingung A<B ist wahr), so druckt der Computer den Satz : B IST GROESSER ALS A. Anderfalls geht der Computer direkt zum naechsten Befehl.

Logische Operatoren

Das Video Genie System kennt die logischen Operatoren AND, OR und NOT.

Beispiel:

```
10 IF A = 1 AND B = 5 GOTO 50
```

Der Computer verzweigt zur Zeile 50, wenn A gleich 1 und B gleich 5 ist. Ist das nicht der Fall, so fuehrt er den Befehl in der naechsten Zeile aus.

```
20 A = (B = 2) AND (C > 10)
```

A bekommt den Wert -1, wenn beide Aussagen, B = 2 und C > 10 wahr sind. Wenn eine von beiden falsch ist, so wird A Null.

```
40 A = (D < 2) OR (E < 20)
```

A bekommt den Wert -1, wenn entweder D < 2 oder E < 20 wahr ist. Wenn beide Aussagen falsch sind, so wird A Null.

```
70 A = NOT (F > 5)
```

A wird -1, wenn F < = 5, sonst Null.

-1 entspricht also dem "wahr" und 0 dem "falsch".

Zeichenkettenoperatoren

Mit Zeichenkettenoperatoren koennen Zeichenketten verglichen und aneinander gehaengt werden.

Folgene Aussagen sind wahr:

"B" < "C"	Der Code von B ist kleiner als der von C (bezieht sich auf ASCII-Code)
"JOHN" > "JACK"	wie oben
"ZEICHEN" = "ZEICHEN"	
"KARL" < > " KARL"	Leerzeichen zaehlen auch
A$ = "OT" + "TO"	A$ bekommt den Wert OTTO

Reihenfolge der Operationen

Die Operationen auf der innersten Ebene der Klammern werden zuerst ausgefuert, dann folgen die der naechsten Ebene usw. Operationen auf der gleichen Klammerebene folgen den Prioritaeten:

1.	Exponentation	A [B
2.	Negation	–C
3.	Multiplikation und Division	A*B, C/D
4.	Addition und Subtraktion	C+D, E–F
5.	Relationsoperatoren	A < B, X$ = Y$, 15 < > 16
6.	Logische Operatoren	NOT, AND, OR

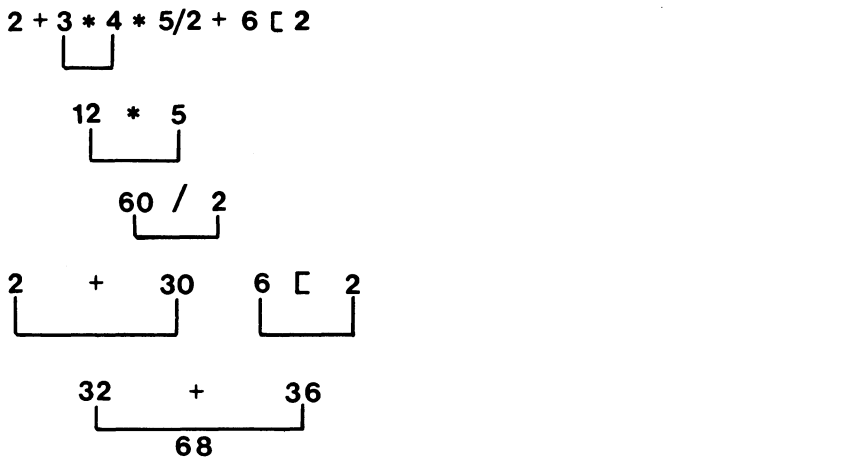
Sehen wir uns z.B. folgende Formel an:

```
10 ANS = A + B * C * C / 2 + E [ 2
```

Setzen wir:

```
A = 2
B = 3
C = 4
D = 5
E = 6
```

Dann wenden wir obige Formel an:



Das Ergenbniss ist also 68.

KAPITEL 1

AKTIVE BEFEHLE

Wenn das System zusammengestellt und der Computer eingeschaltet wird, so muesste sich der Benutzer auf der aktiven Befehlsebene befinden. Normalerweise erkennt man das daran, dass das Wort "READY", gefolgt von einem ">" – Zeichen in der naechsten Zeile in der oberen, linken Ecke des Schirms (Monitor oder Fernsehgeraet) erscheint. Diese Anzeige nennen wir von nun an Fertigmeldung.

Nun sollte der Benutzer die NEW LINE-Taste druecken und kann danach eines der folgenden Kommandos eingeben:

1. AUTO
2. CLEAR
3. CLOAD
4. CLOAD?
5. CONT
6. CSAVE
7. DELETE
8. EDIT
9. LIST
10. NEW
11. RUN
12. SYSTEM
13. TROFF
14. TRON
15. LLIST

Wir werden diese Kommandos im einzelnen diskutieren. Bitte beachten Sie, dass alle Angaben in Klammern optional sind; angegeben werden können, aber nicht müssen. Beispiel: AUTO (Zeilennummer, Inkrement) Das konkrete Kommando sieht so aus:

AUTO 10, 5 oder AUTO oder AUTO 10

Wollen Sie Z.B. Ihr gesamtes Programme auflisten, so geben Sie LIST ein.

Ein Kommando wird mit der NEW LINE – Taste an den Computer abgeschickt, der es dann ausfuehrt.

1.1 AUTO (Zeilennummer, Inkrement)

Dieses Kommando erzeugt automatisch Zeilennummern vor jeder Programmzeile, die Sie eingeben wollen. Die Optionen erlauben Ihnen, die Anfangszeilennummer und die Schrittweite zwischen den Zeilen, anzugeben. Wird nur AUTO, wie bei allen Kommandos gefolgt von der NEW LINE Taste, eingegeben, so nimmt der Computer als erste Zeilennummer die Zahl 10 und als Schrittweite auch die 10 an. Sie koennen den Programmbefehl direkt nach der Zeilennummer eingeben.

Beispiel:

```
30 PRINT "HIER 1ST ZEILE 30"
```

Nach jeder Eingabe einer Programmzeile erhoeht der Computer automatisch die Zeilennummer um den im Inkrement angegebenen Betrag. Das AUTO Kommando bleibt so lange aktiv, bis die BREAK Taste gedrueckt wird. Stoesst der Computer im AUTO Modus auf eine schon benutzte Zeile, so erscheint ein Stern "*"rechts neben der Zeilennummer. Wollen Sie diese Zeile nicht neu schreiben, so betaetigen Sie die BREAK Taste und sind damit wieder auf der Kommando Ebene.

Beispiel:

```
READY
>AUTO 1, 2
1 ZEILE 1
3 ZEILE 3                    NEW LINE
5 ZEILE 5                       ↓
7 ZEILE 7
9 BREAK Taste

READY
>AUTO 2, 2
2 ZWEITE ZEILE
4 VIERTE ZEILE               NEW LINE
6 SECHSTE ZEILE                 ↓
8 BREAK Taste

READY
>AUTO
10 ZEILE 10
20 ZEILE 20                  NEW LINE
30 ZEILE 30                     ↓
40 BREAK Taste

READY
>AUTO 1, 1
1*
2*                           NEW LINE
3*                              ↓
4*
5*
```

1.2 CLEAR (Anzahl von Bytes)

Das CLEAR Kommando loescht eine gewisse Zahl von Bytes und stellt sie zum Abspeichern von Zeichenketten zur Verfuegung. Ausserdem werden alle numerischen Variablen auf Null gesetzt und leere Zeichenketten in die Zeichenket – tenvariablen geschrieben. CLEAR ohne Angabe einer Anzahl von Bytes ordnet den Zeichenkettenvariablen automatisch 50 Bytes Speicherplatz zu, CLEAR 100 ordnet ihnen 100 Bytes zu. Wenn die Fehlermeldung "OS" (out of stringspace) auftritt, so kann mit dem CLEAR Kommando mehr Platz fuer die Zeichenkettenvariablen freigegeben werden.

1.3 CLOAD (#–Kassetteneinheit, "Dateiname")

Dieses Kommando laedt das durch "Dateiname" spezifizierte Programm von dem durch Kassetteneinheit bezeichneten Recorder. Bevor Sie dieses Kommando benutzen, sollten Sie das Band zurueckspulen, nachpruefen, ob alle Kabel in den richtigen Steckern sitzen und der PLAY Knopf am Recorder gedrueckt ist. Wenn dies alles stimmt, koennen Sie z.B. eintippen: CLOAD #–1, "A". NEW LINE startet dieses Kommando wie alle anderen auch. Der Recorder beginnt dannach der Datei zu suchen, bis er den Dateinamen "A" findet. Wenn die Datei gefunden wird, sehen Sie ein stabiles und ein blinkendes Sternchen (*) in der rechten, oberen Ecke des Schirms. Daran sehen Sie, dass das Programm geladen wird. Ist das Programm vollstaendig geladen, erscheint die Fertigmeldung READY auf dem Schirm.

Beispiel: CLOAD #–1, "3"
Von dem Recorder Nr. 1 wird die Datei namens "3" geladen

Beachten Sie, dass nur das erste Zeichen des Dateinamens benutzt wird, um verschiedene Dateien zu unterscheiden. Die Namen "P" und "PREIS" werden als gleich angesehen.

CLOAD? ("Dateiname")

Dieses Kommando vergleicht die Datei auf der Kassette mit dem Programm im Speicher. Normalerweise dient dieses Kommando dazu, nachzupruefen, ob die Daten nach einem CSAVE Kommando korrekt auf der Kassette stehen. Es ist emfehlenswert, den Dateinamen im Zusammenhang mit diesem Kommando immer anzugeben, weil der Computer dann nach der konkreten Datei sucht und seinen Vergleich nicht bei der ersten Datei, die auf dem Band steht, beginnt.
Erscheint auf dem Schirm die Meldung "BAD", so sollte die Kassette zurueckgespult werden und das Programm mit CSAVE noch einmal aufgezeichnet werden.

1.5 CONT

Wird ein Programm durch die BREAK Taste unterbrochen oder hat das Programm einen STOP-Befehl ausgefuehrt, so kann die Ausfuehrung des Programms mit CONT an der Stelle, an der es vorher unterbrochen wurde, wieder aufgenommen werden.

1.6 CSAVE #–Kassetteneinheit, "Dateiname"

Dieses Kommando schreibt das Programm im Speicher des Computers auf die Kassette. Kassetteneinheit und Dateiname m u e s s e n angegeben werden. Jedes alphanumerische Zeichen, ausser den Anfuehrungszeichen, wird als Dateiname erkannt. Wie beim CLOAD muss die Kassette an die richtige Stelle gespult werden, damit keine Programme, die evtl. noch auf dem Band stehen, versehentlich geloescht werden. Alle Stecker muessen richtig sitzen und die Tasten PLAY und RECORD an Recorder musessen gleichzeitig gedrueckt worden sein.
Beispiel: CSAVE #–2, "C"
Schreibt das Programm im Hauptspeicher unter dem Namen "C" auf die Kassetteneinheit Nr. 2.

1.7 DELETE Zeilennummer (– Zeilennummer)

Dieses Kommando loescht die angegebene(n) Zeile(n).

Beispiele:

DELETE 5	Loescht Zeile 5
DELETE 7 – 10	Loescht Zeile 7, 10 und alle dazwischen
DELETE – 12	Loescht alle Zeilen vom Programmanfang bis einschliesslich Zeile 12
DELETE .	Loescht die gerade eingegebene oder Editierte Zeile

1.8 EDIT Zeilennummer

Dieses Kommando veranlasst den Computer von der aktiven Kommandoebene in den Editierungsmodus ueberzugehen. Die Editierungsebene erlaubt es, Zeilen im Speicher zu veraendern ohne sie neu eintippen zu muessen. Das EDIT Kommando hat eine Reihe vom Subkommandos, die auf der Editierrungsebene ausgefuehrt werden koennen.
(siehe Kapitel 2) Dem EDIT Kommando muss eine gueltige Zeilennummer folgen.

Beispiel: EDIT 20

Schaltet um auf die Editierungsebene, um Zeile 20 zu untersuchen.

1.9 LIST (Zeilennummer – Zeilennummer)

Mit diesem Kommando veranlassen Sie den Computer, eine oder mehrere Programmzeilen auf dem Schirm darzustellen. LIST ohne die Optionen in Klammern schreibt das gesamte Programm, dass sich momentan im Speicher befindet, heraus. Um den schnellen Durchlauf der Programmzeilen auf dem Schirm anzuhalten, betaetigen Sie die SHIFT und die @ Taste gleichzeitig. Das Durchlaufen der Textzeilen (engl. up scrolling) wird durch Druecken einer beliebigen Taste wieder gestartet.

Beispiele:

List 3	Zeile 3 wird ausgegeben
LIST 10 – 20	Zeile 10, 20 und alle dazwischen
LIST – 50	von der ersten Zeile bis Zeile 50
LIST 20 –	von Zeile 20 bis zum Programmende
LIST	die Zeile, die gerade gerade editiert wurde
LIST	alle Zeilen im Speicher

1.10 NEW

Dieses Kommando loescht alle Programmzeilen, setzt alle numerischen Variablen auf Null und schreibt in alle Zeichenkettenvariablen die leere Zeichenkette. Der Speicherplatz fuer die Zeichenkettenvariablen, der mit CLEAR gesetzt wurde, wird nicht veraendert.

1.11 RUN (Zeilennummer)

Dieses Kommando startet ein Benutzerprogramm im Speicher. Wird die Zeilennummer nicht angegeben, so wird das Programm in der ersten Zeile gestartet. Wird jedoch eine Zeilennummer angegeben, so wird das Programm in dieser Zeile gestartet. Eingabe einer ungueltigen Zeilennummer fuert zu einer Fehlermeldung.
Mit jedem RUN Kommando wird automatisch auch ein CLEAR Kommando ausgefuehrt.

Beispiele:

RUN 50	Starte das Programm in Zeile 50
RUN	Starte das Programm in der ersten Zeile

1.12 SYSTEM

Dieses Kommando bringt den Computer in den Monitor-Modus In diesem Modus koennen Maschinenprogramme von der Kassette geladen und ausgefuehrt werden. Nach Eingabe des SYSTEM-Kommandos erscheint ein "*?" auf dem Schirm. Der Computer erwartet dann die Eingabe des Dateinamens der Maschinencodedatei (Ojekt code), die er von der Kassette laden soll. Ist das Maschinenprogramm vollstaendig geladen, so erscheint ein weiteres "*?". Um das Programm nun zu starten, tippen Sie einen / , gefolgt von der Startadresse des Maschinenprogramms, (in dezimaler Schreibweise) ein. Wird nur ein / eingegeben, so startet das Programm an der Adresse, die in der Datei spezifiziert wurde.

1.13 TROFF

Dieses Kommando schaltet die TRON Funktion aus. Normalerweise folgt es hinter einem TRON.

1.14 TRON

Dieses Kommando schaltet die Trace-Funktion ein. Sie erlaubt es, den Ablauf eines Programms zur Fehlersuche zu verfolgen. Sobald der Computer eine neue Programmzeile ausfuehrt, wird ihre Zeilennummer in Klammern ausgegeben.

Beispiel:

```
10 PRINT " ** PROGRAMM 1 ** "
20 A = 1
30 IF A = 3 THEN 70
40 PRINT A
50 A = A + 1
60 GOTO 30
70 PRINT " ** ENDE VON PROGRAMM 1 ** "
80 END
```

Geben Sie ein:

```
> TRON
> RUN
(10) ** PROGRAMM 1 **
(20) (30) (40) 1
(50) (60) (30) (70) ** ENDE VON PROGRAMM 1 **
(80)
```

Soll die Programmausfuehrung vor dem Programmende kurz unterbrochen werden, so kann das durch gleichzeitiges Betaetigen der SHIFT und der @ Taste erreicht werden. Um fortzufahren, bestaetigen Sie irgendeine Taste. TRON und TROFF koennen auch in einem Programm verwandt werden.
Beispiel:

```
.
.
.
90 IF A = B THEN 160
100 TRON
110 A = B + C
120 TROFF
.
.
.
```

Ist bei der Ausfuehrung dieses Programmteil A nicht gleich B, werden Zeile 100 bis 120 ausgefuehrt. Die Trace Option wird eingeschaltet und das Durchlaufen des Befehls in Zeile 110 wird angezeigt. Danach wird die Trace-Option durch TROFF wieder abgeschaltet. TRON und TROFF koennen nach der Fehlersuche wieder aus dem Programm entfernt werden.

1.15 LLIST

Listet ein Programm auf dem Drucker. Dieses Kommando arbeitet genau so, wie das LIST. Wird das LLIST ausgefuehrt, wenn kein Drucker an das System angeschlossen ist, so geraet der Computer in eine endlose Schleife, aus der er nur mit der RESET-Taste befreit werden kann.

KAPITEL 2

Text Editierung

Der Text Editor des Video Genie System erlaubt seinem Benutzer, die Programmzeilen, die er geschrieben hat, zu korrgieren, ohne sie neu eintippen zu muessen. Die Notwendigkeit eines Editors wird besonders bei langen, komplexen Programmzeilen deutlich. In diesem Kapitel werden wir alle Editierungsfunktionen des Video Genie Systems mit ihren Subkommandos, besprechen. Wir werden unsere Erklaerungen mit vielen Beispielen verstaendlicher machen. Dem Benutzer wird empfohlen, jedes Editierungskommando zu probieren, bevor er sich an das Schreiben von Programmen begibt.

2.1 EDIT Zeilennummer

Dieses Kommando bringt den Computer von der aktiven Kommandoebene in die Editierungsebene. Der Benutzer muss angeben, welche Zeile er editieren will. Wird keine Zeilennummer angegeben, reagiert der Rechner mit einer FC Fehlermeldung.

Beispiel:

EDIT 100	eroeffnet Editierung von Zeile 100
EDIT .	eroeffnet Editierung der gerade eingetippten Zeile.

Auf der Editierungsebene kann der Computer folgende Subkommandos ausfuhren:

2.2. NEW LINE Taste

Wird die NEW LINE Taste betaetigt, wenn sich der Rechner im Editierungsmodus befindet, so speichert er alle vorher gemachten Aenderungen ab und kehrt auf die aktive Kommandoebene zurueck.

2.3 Leertaste

Im Editierungsmodus wird bei jeder Betaetigung der Leertaste ein weiteres Zeichen in der editierten Zeile angezeigt. Wird eine Zahl n vor der Betaetigung der Leertaste eingegeben, so werden n Zeichen der gerade editierten Zeile ausgegeben.

Nehmen wir an, wir haben folgende Zeile eingegeben:

```
AUTO 100
100 IF A = B THEN 150 : A = A + 1 : GOTO 100
```

Soll nun diese Zeile 100 editiert werden, so muss der Benutzer EDIT 100 eingeben.
> EDIT 100

Der Computer antwortet:
100 _
Wird die Leertaste 12 mal betaetigt, so bewegt sich der Cursor 12 Zeichen nach rechts (Cursor = der Unterstrich, den der Computer hinter jedes eingegebene Zeichen setzt). Auf dem Schirm steht:

100 IF A = B THE _

Man muss nicht fuer jedes Zeichen einzeln die Leertaste betaetigen. Geben Sie 8, gefolgt von der Leertaste ein, erscheint folgende Zeile:

100 IF A = B THEN 150 : _

Geben Sie die 20, gefolgt von der Leertaste, ein, so erscheint schliesslich die gesamte Zeile:

100 IF A = D THEN 150 : A = A + 1 : GOTO 100 _

2.4 BACKSPACE Taste

Bewegt den Cursor ein Zeichen zurueck (entgegengesetzt zur Leertaste). Wird eine Zahl vor dem Betaetigen der BACKSPACE Taste eingegeben, so wird der Cursor um den Betrag dieser Zahl nach links bewegt. Entgegen der normalen Backspace Funktion bleiben die so auf dem Schirm geloeschten Zeichen im Speicher erhalten.

Beispiel:
100 IF A = B THEN 150 A = A + 1 : GOTO 100 _

Nach 5 maligem Betaeigen der BACKSPACE Taste:

100 IF A = B THEN 150 : A = A + 1 : GOT _

Geben Sie nun eine 10, gefolgt von der BACKSPACE Taste ein, erscheint folgende Zeile:

100 IF A = B THEN 150 : A = A _

Nochmals betont, um den Editierungsmodus zu verlassen, betaetigen Sie die NEW LINE Taste. Der Computer geht zurueck auf die aktive Kommandoebene und auf dem Schirm steht:
> _

Wenn nun noch weitere Aenderungen an Zeile 100 vorgenommen werden sollen, muessen sie mit EDIT 100 in den Editierungsmodus zurueck.

2.5 SHIFT ESC Taste

Wird die SHIFT und die ESC (escape) Taste gleichzeitig betaetigt, so stellt der Computer die Ausfuehrung eines vorher gestarteten Subkommandos (zum Einsetzen von Zeichen H, I, X) ein. Der Computer befindet sich danach weiter auf der Editierungsebene und die Position des Cursors bleibt unveraendert. Ein anderer Weg, um die Ausfuehrung eines der Subkommandos zu beenden, ist die NEW LINE Taste. Wird Sie betaetigt, so kehrt der Computer auf die aktive Kommandoebene zurueck.

2.6 H Taste

"H" meint trenne ab (engl. hack) und setze ein. D.h. der Rest der Zeile wird geloescht und an der momentanen Position des Cursors koennen Zeichen eingesetzt werden.

Beispiel:

100 IF A = B THEN 150 : A = A + 1 : GOTO 100

Nehmen Sie an, Sie wollten A = A + 1 durch A = A + B ersetzen und das GOTO 100 loeschen.
Zunaechst gehen Sie in den Editierungsmodus mit
EDIT 100
Die Zahl 25, gefolgt von der Leertaste bringt den Cursor auf die 25. Position der Zeile:

100 IF A = B THEN 150 : A = A _

Nun betaetigen Sie die H Taste und tippen + B ein. Dann betaetigen Sie die NEW LINE Taste und sind auf der aktiven Kommandoebene. Eine andere Moeglichkeit, die Sie im Editierungmodus bleiben laesst, ist das gleichzeitige Betaetigen der SHIFT und der ESC Taste. Geben sie nun L fuer List ein. Der Computer gibt einmal die gesamte Zeile aus und der Cursor geht an den Anfang der Zeile:

IF A = B THEN 150 : A = A + B

100 _
Alles, was nicht ausgegeben wurde ist effektiv geloescht.

2.7 I Taste

"I" meint Einsetzen (insert). Mit diesem Subkommando koennen Sie Zeichen an einer beliebigen Position einsetzen, ohne andere Teile der Zeile aendern zu muessen.

Beispiel:
Nehmen Sie an, Sie wollen den Befehl "PRINT A" zwischen "A = A + 1" und "GOTO 100" in der Zeile 100, einsetzen. Die Zeile 100 soll vorher so aussehen:

```
100 IF A = B THEN 150 : A = A + 1 : GOTO 100
```

Mit der Leertaste bringen Sie den Cursor zur Position:

```
100 IF A = B THEN 150 : A = A + 1 : _
```

Nun betaetigen Sie die "I" Taste und geben "PRINT A :" ein. Mit der SHIFT und ESC Taste verlassen Sie den Einsetzmodus (insert mode). Nun koennen Sie mit L die gesamte Zeile listen:

```
100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

100 _
```

Mit NEW LINE koennen Sie dann auf die aktive Kommandoebene zurueck.

2.8 X Taste

"X" meint setze am Ende der Zeile ein.Der Cursor wird an das Ende der Zeile bewegt und der Computer geht in den Einsetzmodus. Man kann nun am Ende der Zeile Zeichen hinzufuegen oder mit der BACKSPACE Taste am Ende der Zeile Zeichen loeschen.

Beispiel:

In den Editierungsmodus:

```
> EDIT 100
100 _
```

Geben Sie nun "X" (ohne NEW LINE Taste) ein:

100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

100 _

An dieser Stelle koennen Sie neue Zeichen eingeben, schon existierende loeschen und dieses Subkommando mit SHIFT und ESC verlasen.

2.9 L Taste

"L" meint liste Zeile. Ist der Computer im Editierungsmodus, fuehrt aber im Moment keines der Subkommandos H, I und X aus, so kann man mit dem L Subkommando den verbliebenen Teil der Zeile auf dem Schirm ausgeben.

Beispiel:

> EDIT 100
100 _

Betaetigen Sie die L Taste:

100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

100 _

In der zweiten Zeile koennen Sie editieren und dabei den Ursprungstext im Auge behalten.

2.10 A Taste

"A" meint beginne neu (cancel and restart). Der Cursor wird wieder zum Zeilenangang gebracht und alle vorher gemachten Aenderungen werden geloescht. Die Zeile bleibt in ihrem urspruenglichen Zustand.

2.11 E Taste

Dieses Kommando bringt den Computer zurueck auf die aktive Kommandoebene und traegt alle vorher gemachten Aenderungen in der editierten Zeile ein. Der Computer darf kein Subkommando ausfuehren (H, I und X), wenn E eingegeben wird.

2.12 Q Taste

Dieses Kommando bringt der Computer von der Editierungsebene zurueck auf die aktive Kommandoebene, aber loescht alle voher gemachten Aenderungen. Die Zeile bleibt in ihrem urspruenglichen Zustand.

2.13 D Taste

"D" meint locsche (delete). Wird vor dem D eine Zahl n eingegeben, so werden n Zeichen rechts von der momentanen Cursorposition geloescht. Die geloeschten Zeichen werden in Ausrufezeichen eingeschlossen, um zu zeigen, dass sie von der Operation betroffen sind.

Beispiel:
100 IF A = B THEN 150 : A = A + 1 : PRINT A : GOTO 100

Schalten Sie auf die Editierungsebene und bewegen Sie den Cursor mit der Leertaste in folgende Position:

100 IF A = B THEN 150 : A = A + 1 _

Nun geben 15 D ein (zerstoere 15 Zeichen–:

100 IF A = B THEN 150 : A = A + 1! : PRINT A : GO!

Benutzen Sie L, um die gesamte Zeile zu listen:

100 IF A = B THEN 150 : A = A + 1! : PRINT A : GO! TO 100

100 _

Ein weiters List bringt folgendes auf den Schirm:

100 IF A = B THEN 150 : A = A + 1TO 100

100 _

Um das "TO 100" zu loeschen, benutzen Sie die X Taste und die BACKSPACE Taste. Das Endergebniss sollte sein:

100 IF A = B THEN 150 : A = A + 1

2.14 C Taste

"C" meint veraendern (change). Wird eine Zahl n vor dem C eingegeben, so werden n Zeichen rechts von der momentanen Cursorposition veraendert. Wird die Zahl nicht angegeben, so wird nur ein Zeichen veraendert.

Beispiel:
100 IF A = B THEN 150 : A = A + 1

Wollen Sie die 150 in eine 230 aendern, so schalten Sie den Editierungsmodus (EDIT 100) ein und bewegen den Cursor mit der Leertaste in folgende Position:

100 IF A = B THEN

Geben Sie nun ein 2C (aendere 2 Zeichen), gefolgt von 23 (neue Daten). Dann verlassen Sie das C Subkommando mit SHIFT und ESC. Listen Sie die Zeile mit L:

100 IF A = B THEN 230 : A = A + 1

100 _

2.15 nSc

Dises Commando sucht (search) nach dem n'ten Vorkommen des Zeichens c in der gerade editierten Zeile and setzt den Cursor an dessen Position. Wird n nicht angegeben, so wird nach dem ersten Auftreten des Zeichens c gesucht. Wird das Zeichen c nicht gefunden, so steht der Cursor am Ende der Zeile. Wie bei den anderen Subkommandos, beginnt der Rechner an der momentanen Cursorposition zu suchen.

Beispiel:

100 IF A = B THEN 230 : A = A + 1

Schalten Sie auf den Editierungsmodus (> EDIT 100):

100 _

Nun geben Sie 2S = ein, um dem Computer mitzuteilen, er soll nach dem zweiten Auftreten des Zeichens "=" suchen:

100 IF A = B THEN 230 : A _

Nun kann ein anderes Subkommando aktiviert werden, um die so aufgefundenen Daten zu veraendern. Etwa: Geben Sie H und danach = A + 2 (neue Daten) ein:

100 IF A = B THEN 230 : A = A + 2 _

2.16 nKc

Dieses Kommando locscht alle Zeichen bis zum n-ten Vorkommen des Zeichens c und setzt den Cursor an diese Stelle. Folgendes Beispiel:

100 IF A = B THEN 230: : A = A + 2

Schalten Sie auf den Editierungsmodus (> EDITO 100):

100 _

Nun geben Sie ein 1K:, um dem Computer mitzuteilen, er solle nach dem ersten Auftreten des Zeichens Doppelpunkt ":" suchen und solle alles loeschen, was vor ihm in der Zeile steht:

100! IF A = B THEN 230!

Soll der Doppelpunkt noch dazu geloescht werden, geben Sie D ein:

100! IF ! = B THEN 230!! : !

Um das Ergebniss zu betrachten, geben Sie L ein:

100 A = A + 2

100 _

KAPITEL 3

BASIC Programm Statements

Wir wollen in diesem Kapitel die Programmbefehle (= Statements) unserer BASIC Programmiersprache erlaeutern. Im ersten Teil setzen wir uns mit den Ein– Ausgebebefehlen, die dem Computer zur Verfuegung stehen, um mit der Aussenwelt in Verbindung zu treten, auseinander. Besonders behandelt werden solche, die mit dem Bildschirm und der Tastatur arbeiten und die Datenspeicherung auf der Kassette ermoeglichen.

Der zweite Teil dieses Kapiltels handelt von verschiedenen Funktionen der Programmbefehle (= Statements), die vom Video Genie System interpretiert werden. Da dies eine recht grosse Anzahl von Statements ist und jedes seine eigenen, charakteristischen Eigenschaften hat, wird dem Benutzer nahe gelegt, jedes Statement mit Hilfe der angegebenen Beispiele genau zu studieren.

Ein– Ausgabestatements

3.1 PRINT Item Liste

Schreibt ein Item oder eine Itemliste auf den Schirm. Als Item wird folgendes aufgefasst:

a)	Numerische Konstante	(Zahlen wie 0, 36872, 0.2, –34)
b)	Zeichenkonstanten	(Zeichen in Anfuehrungszeichen, wie etwa "VIDEO GENIE SYSTEM", "123 #$%" usw.)
c)	Zeichenvariablen	(Namen, die eine Zeichenkonstante repraesentieren, wie A$, X$, PR$ usw.)
d)	Ausdruecke	(Verknuepfung der obrigen Items, wie (X + 10)/Y oder "COMP" + "UTER" usw.)

Die Items in der Itemliste koennen durch Kommas oder Semikolons getrennt werden. Werden Kommas benutzt, so geht der Cursor automatisch zur naechten Schreibzone, bevor er das Item ausgibt. Werden Semikolons benutzt, so wird kein Platz zwischen den Items gelassen; eines wird an das andere gehaengt. Bei numerischen Items wird allerdings eine Leerstelle gelassen.

Beispiel:

```
10 N = 25 + 7
20 PRINT "25 + 7 IST GLEICH"; N
30 END
```

```
READY
> RUN

25 + 7 IST GLEICH 32
```

Beispiel:

```
10 H$ = "HEIM"
20 C$ = "COMPUTER"
30 PRINT "PROBIEREN SIE UNSEREN"; H$; C$
40 END

READY
> RUN

PROBIEREN SIE UNSEREN HEIMCOMPUTER
```

Werden Kommas benutzt, um die Items zu trennen, so erzeugt der Computer 4 Spalten pro Zeile. Jede Spalte kann maximal 16 Zeichen enthalten. Zeichen, die darueber hinaus gehen, werden in die naechste Zeile geschrieben.

Beispiel:

```
10 PRINT "SPALTE 1", "SPALTE 2", "SPALTE 3", "SPALTE 4", "SPALTE 5"
20 END

READY
> RUN

SPALTE 1        SPALTE 2        SPALTE 3        SPALTE 4
SPALTE 5
```

Werden zwei oder mehr Kommas angegeben, so erzeugt jedes 16 Leerstellen (blanks).

Beispiel:

```
10 PRINT "SPALTE 1", "SPALTE 2"
20 END

READY
> RUN

SPALTE 1                        SPALTE 2
```

Beachten Sie folgendes Beispiel:

```
10 PRINT "ZEILE 1"
20 PRINT "ZEILE 2"
30 END

READY
> RUN

ZEILE 1
ZEILE 2

10 PRINT "ZEILE 1",
20 PRINT "ZEILE 2"
30 END

READY
> RUN

ZEILE 1         ZEILE 2
```

3.2 PRINT @ Stelle, Item Liste

Dieses Statement gibt die items der Itemliste an der angegebenen Stelle auf dem Schirm aus. Das "@" Zeichen muss unmittelbar auf das PRINT folgen und Stelle darf Werte zwischen 0 und 1023 annehmen. Naehere Informationen folgen in Anhang E.

Beispiel:

```
20 PRINT @ 100, "STELLE 100"
```

Wenn der Benutzer ein PRINT Statement eingibt, das auf die letzte Zeile des Schirms schreibt, wird automatisch ein Zeilenvorschub erzeugt, der alle Zeilen um eine Position nach oben wandern laesst. Um dies zu unterbinden, setzen Sie ein Semikolon an das Ende des Statements.

Beispiel:

```
10 PRINT @ 999, "LETZTE ZEILE";
```

3.3 PRINT TAB (Ausdruck)

Erlaubt es, den Cursor an jede, beliebige Stelle in der Zeile zu setzen. Wenn der Wert von Ausdruck groesser als 63 wird, wird die naechste Zeile benutzt. Man kann mehr als ein TAB in einem PRINT Statement benutzen. Doch der Wert von Ausdruck muss zwischen 0 und einschl. 255 liegen.

Beispiel:

```
10 PRINT TAB (10) "POSITION 10" TAB (30) "POSITION 30"

READY
> RUN

         POSITION 10              POSITION 30

10 N = 4
20 PRINT TAB (N) "POS"; N TAB (N + 10) "POS"; N + 10 TAB (N + 20) "POS"; N + 20
30 END

READY
> RUN

POS 4        POS 14        POS 24
```

3.4 PRINT USING Format, Item Liste

Dieses Statement erlaubt es, Daten in einem vorher festgelegtem Format auszugeben. Daten koennen numerische oder Zeichenketten werte sein.
Die Format – und Item Liste im PRINT USING Statement kann Variablen oder Konstanten enthalten. Das Statement schreibt die Items in der Form, wie es in der Formatangabe vorgegeben wird.

Die folgenden Deskriptoren koennen in dem Formatfeld benutzt werden:

\# Dieses Zeichen repraesentiert die richtige Stellung jeder Dezimalstelle in einer Zahl der Itemliste. Die Anzahl der # Zeichen bildet das erwuenschte Format. Ist das Formatfeld groesser als die Anzahl der Stellen in der Zahl, so werden die unbenutzten Feldpositionen links von der Zahl als Leerzeichen (blanks) und die rechts vom Dezimalpunkt als Nullen ausgegeben. Der Dezimalpunkt kann irgentwo in das, durch die # Zeichen gebildete Formatfeld gesetzt werden. Werden Nachkommastellen unterdrueckt, so wird gerundet. Wird ein Komma in eine Position zwischen der ersten Ziffer und dem Dezimalpunkt gesetzt, so erscheint in der Ausgabe nach je drei Vorkommaziffern ein Komma (beachte: der Computer benutzt die angloamerikanische Dezimalschreibweise, das deutsche Dezimalkomma entspricht hier dem Punkt)

Beispiel:

```
10 INPUT "FORMAT EINGEBEN"; F$
20 IF F$ = "STOP" END
30 INPUT "ZAHL EINGEBEN"; N
40 PRINT USING F$; N
GOTO 10
```

Dieses Programm fragt nach einer Formatliste und nach einem Item (in diesem Fall ein numerischer Wert). Es stoppt erst, wenn "STOP" eingegeben wird

Nun probieren Sie das Programm aus:

```
READY
> RUN

FORMAT EINGEBEN ? ## . ##
ZAHL EINGEBEN ? 12.34
12.34
FORMAT EINGEBEN ? ### . ##
ZAHL EINGEBEN ? 12.34
12.34
FORMAT EINGEBEN ? ## . ##
ZAHL EINGEBEN ? 123.45
%123.45
FORMAT EINGEBEN ? STOP
```

Das % Zeichen wird ausgegeben, wenn das angegebene Feld fuer die Zahl zu klein ist. Die letzte Zahl links vom Dezimalpunkt wird nach dem % Zeichen ausgegeben.

Nun starten wir das obige Programm wieder:

```
READY
> RUN

FORMAT EINGEBEN ? ## . ##
ZAHL EINGEBEN ? 12.345
12.35
FORMAT EINGEBEN ? STOP
```

Weil im Format nur zwei Nachkomma (bzw. Punkt) – Stellen angegeben sind, wird aufgerundet.

** Zwei Sternchen am Anfang des Formatfeldes bewirken, dass die unbenutzten Positionen links vom Dezimalpunkt mit Sternchen aufgefuellt werden. Die zwei Sternchen erzeugen zwei Feldpositionen mehr.

$$ Zwei Dollarzeichen am Anfang des Formatfeldes erzeugen ein gleitendes Dollarzeichen. D.h. ein Dollarzeichen wird vor der hoechsten Stelle der Zahl ausgegeben.

**$ Kombiniert den Effekt von ** und $$. Jede leere Position links von der Zahl wird mit Sternchen aufgefuellt und das Dollarzeichen wird vor der hoechsten Stelle der Zahl ausgegeben.
Betrachten Sie das Beispielprogramm von vorhin:

```
READY
> RUN

FORMAT EINGEBEN ? ** ## . ##
ZAHL EINGEBEN ? 12.3
** 12.3
FORMAT EINGEBEN ? $$ ## . ##
ZAHL EINGEBEN ? 12.34
$12.34
FORMAT EINGEBEN ? **$ ### . ##
ZAHL EINGEBEN ? 12.34
***$12.34
FORMAT EINGEBEN ? STOP
```

+ Wird ein + Zeichen am Anfang oder am Ende des Formatfelds angegeben, so schreibt der Computer ein + Zeichen fuer positive Zahlen und ein – Zeichen fuer negative Zahlen an Anfang bzw. Ende der Zahl.

– Wird ein – Zeichen an das Ende des Formatfeldes gesetzt, so wird ein – Zeichen nach jeder negativen Zahl ausgegeben und ein Leerzeichen fuer positive Zahlen.

Beispiel:

```
READY
> RUN

FORMAT EINGEBEN? #### , #
ZAHL EINGEBEN ? 12345.6
12,346
```

```
FORMAT EINGEBEN ? + ## . ##
ZAHL EINGABE? 12.34
+ 12.34
FORMAT EINGEBEN ? + ## . ##
ZAHL EINGEBEN ? – 12.34
– 12.34
FORMAT EINGEBEN ? ## . ## +
ZAHL EINGEBEN ? –12.34
12.34 –
FORMAT EINGEBEN ? ## . ##
ZAHL EINGEBEN ? 12.34
12.34
FORMAT EINGEBEN ? ## . ###
ZAHL EINGEBEN ? 123456
%123456.000
FORMAT EINGEBEN ? STOP
```

% LEERZEICHEN%
Dient zum Definieren eines Zeichenfeldes, das mehr als ein Zeichen enthaelt. Die Laenge des so formatierten Feldes ist die Zahl der Leerzeichen zwischen den Prozentzeichen plus 2. Ein Ausrufungszeichen bringt den Computer dazu, nur das erste Zeichen einer aktuellen Zeichenkette zu benutzen.

Beispielprogramm:

```
10 INPUT "FORMAT EINGEBEN"; F$
20 IF F$ = "STOP" END
30 INPUT "ZEICHENKETTE EINGEBEN"; C$
40 PRINT USING F$; C$
50 GOTO 10
```

Dieses Programm arbeitet aehnlich wie das vorherige, nur wird jetzt keine numerische Variable mehr benuzt, sondern eine Zeichenkettenvariable.

Nun starten wir dieses Programm:

```
READY
> RUN

FORMAT EINGEBEN ?
ZEICHENKETTE EINGEBEN ? ABCDE
A
FORMAT EINGEBEN ? % %
ZEICHENKETTE EINGEBEN ? ABCDE
ABC
FORMAT EINGEBEN ?   %
ZEICHENKETTE EINGEBEN ? ABCDEF
ABCDE
Format eingeben ? STOP
```

! Mit dem Ausrufezeichen koennen Sie Zeichenketten hintereinander haengen.

Beispiel:

```
10 INPUT "GIB DREI ZEICHENKETTEN EIN"; A$, B$, C$
20 PRINT "DAS ERGEBNISS IST:"; : PRINT USING "!"; A$; B$; C$
30 END

READY
> RUN

GIB DREI ZEICHENKETTEN EIN ? ABC, XYZ, IJK
DAS ERGEBNISS IST: AXI

GIB DREI ZEICHENKETTEN EIN ? A, COMPUTER, PROGRAMM
DAS ERGEBNISS IST: ACP
```

Benutzt man mehr als das eine Ausrufezeichen, wird der erste Buchstabe der Zeichenkette mit so vielen Leerzeichen dahinter ausgedruckt, wie Leerzeichen zwischen den Ausrufezeichen eingesetzt wurden.

Folgendes Beispiel:

```
10 INPUT "GIB DREI ZEICHENKETTEN EIN"; A$, B$, C$
20 PRINT "DAS ERGEBNISS IST:";: PRINT USING "! ! !"; A$; B$; C$
30 END

READY
> RUN

GIB DREI ZEICHENKETTEN EIN ? XYZ, FGH, ABC
DAS ERGEBNISS IST: X F A

GIB DREI ZEICHENKETTEN EIN ? A, COMPUTER, PROGRAMM
DAS ERGEBNISS IST: A C P
```

3.5 INPUT Itemliste

Dieses Statement veranlasst den Computer, die Programmausfuehrung zu unterbrechen und zu warten, bis der Benutzer Daten eines spezifizierten Typs und einer spezifizierten Anzahl auf der Tastatur eingegeben hat. Einzugebende Daten koennen numerisch oder Zeichenketten, abhaengig vom Typ der Variablen, sein. Jedes Item (wenn mehr als eins angegeben wird) muss durch ein Komma vom anderen getrennt werden.

Beispiel:

```
10 INPUT A$, B$, A, B
```

Bei Ausfuehrung dieses Statements muss der Benutzer zwei Zeichenketten und zwei numerische Werte eingeben. Die Reihenfolge der Eingaben muss konsistent sein. Fuehrt der Computer ein INPUT Statement aus, so schickt er ein Signal zum Bildschirm:

```
?
```

Und wartet auf die Eingabe (n). Man kann alle vier Werte in einer Zeile eingeben (durch Kommas getrennt).
In diesem Fall koennte die Eingabe wie folgt aussehen:

```
BANANE, APFEL, 59, 3.14        NEW LINE Taste
```

Der Computer ordnet die Werte dann wie folgt zu:

A$ = "BANANE"
B$ = "APFEL"
A = 59
B = 3.14

Eine andere Methode, diese Daten einzugeben waere es, sie in separate Zeilen aufzuteilen. Nach jeder Betaetigung der NEW LINE Taste erinnert Sie der Computer dann daran, das er noch Daten erwartet, indem er ausgibt:

??

bis er Werte fuer alle Variablen erhalten hat. Dann geht er zum naechsten Statement ueber. Die Eingaben muessen mit den Variablentypen kompatibel sein. Man darf also fuer eine numerische Variable keine Zeichenkette eingeben. Wenn solches versucht wird, reagiert der Computer mit:

? REDO

?

Und erwartet neue Daten fuer die Variable, deren Datentyp mit der vorherigen Eingabe nicht uebereingestimmt hat.

Beispiel:

```
10 INPUT A$, A
20 PRINT A$, A
30 END

READY
> RUN

? STRING, 10
STRING        10

READY
> RUN
```

```
? DIES IST EINE ZEICHENKETTE, 13.5
DIES IST EINE ZEICHENKETTE        13.5

READY
> RUN

? ABCD, IJK
? REDO
? ABCDE
?? 25
ABCDE 25
```

Besteht eine Eingabezeichenkette nur aus Leerzeichen, so muss sie in Anfuehrungszeichen eingeschlossen werden.

Um klarer zu machen, welche Eingabe nun erwartet wird, kann man jedem INPUT Statement eine Nachricht mitgeben, die ausgedruckt wird, bevor der Computer mit dem ? Zeichen Daten erwartet. Diese Nachricht muss dem INPUT Statement unmittelbar folgen, muss in Anfuehrungszeichen eingeschlossen sein und von einem Semikolon gefolgt werden.

Beispiel:

```
10 INPUT "ART DER WARE UND STUECKZAHL"; N$, S

READY
> RUN

ART DER WARE UND STUECKZAHL ? . . . . . .
```

3.6 DATA Itemliste

Dieses Statement erlaubt es dem Benutzer, Daten im Programm anzugeben und mit dem READ Statement auf sie zuzugreifen. Auf die Daten greift der Computer sequentiell, beginnend mit dem ersten Item und endend mit dem letz ten, zu, Jedes Item darf eine Zeichenkette oder auch eine numerische Variable sein. Genau wie bei der Eingabe ueber die Tastatur, muss jede Zeichenkette, die entweder Leerzeichen oder Semikolons oder Kommas enthaelt, in Anfuehrungszeichen eingeschlossen werden.

Die Anordung der Items im DATA Statement muss mit dem Typ der Variablen im dazu gehoerigen READ Statement uebereinstimmen (es darf nicht versucht werden, Zeichenketten in numerische Variablen zu lesen).Das DATA Statement darf an beliebiger Stelle im Programm stehen.

Beisipiel:

```
10 READ A$, B$, C, D
20 PRINT A$, B$, C, D
30 DATA "ZEICHEN", "EIN LANGER SATZ"
40 DATA 20,137.54
50 END

READY
>RUN
ZEICHEN        EIN LANGER SATZ    20              137.54
```

3.7 READ Itemliste

Dieses Statement erlaubt es, Daten vom DATA Statement zu lesen und Variablen zuzuordnen. Die Werte im DATA Statement werden sequentiell vom READ Statement gelesen. Sind alle Daten im ersten DATA Statement gelesen, so liest das naechste vorkommende READ Statement aus dem naechsten DATA Statement. Sind dann alle Werte in allen DATA Statements einmal gelesen und versucht danach ein READ Statement weitere Werte zu bekommen, so tritt ein "keine Daten mehr" Fehler auf (out of data error) mit dem Code OD.

Beispiel:

```
10 READ C$
20 IF C$ = "EOF" GOTO 60
30 READ Q
40 PRINT C$, Q
50 GOTO 10
60 PRINT : PRINT "ENDE DER LISTE" : END
70 DATA BUECHER, 4, BLEISTIFTE, 5, KUGELSCHREIBER, 6
80 DATA FUELLER, 6, TIPPEX, 7, EOF
READY
>RUN

BUECHER              4
BLEISTIFTE           5
KUGELSCHREIBER       6
FUELLER              6
TIPPEX               7

ENDE DER LISTE
```

3.8 RESTORE

Dieses Statement erlaubt es dem naechsten READ Statement, das erste Item der ersten DATA Liste zu lesen, auch wenn zuvor schon andere READS ausgefuehrt wurden.

Beispiel:

```
10 READ A$, A
20 PRINT A$, A
30 RESTORE
40 READ B$, B
50 PRINT A$, A, B$, B
60 DATE 'JUPP SCHMITT", 25, "FRANZ SCHULZ", 32, "DRACULA", 234
70 END
```

```
READY
>RUN

JUPP SCHMITT          25
JUPP SCHMITT          25                JUPP SCHMITT          25
```

Dieses Programm zeigt, dass das RESTORE Statement es nicht nur erlaubt, auf das erste Item im ersten DATA Statement zuzugreifen, sondern dass es ausserdem keinen Einfluss auf die vorher gemachten Zuordnungen hat.

3.9 PRINT #-Kassettennummer, Itemliste

Dieses Statement gibt die Werte der angegeben Variablen in Itemliste auf dem Kassettenrecorder aus. Der Recorder muss geeignet vorbereitet sein, bevor dieses Statement ausgefuehrt wird. Naehere Informationen darueber finden Sie im Benutzerhandbuch. Das Video Genie System kann bis zu zwei Recorder kontrollieren, der Benutzer kann mit Kassettennummern angeben, welcher angesprochen werden soll.

Beispiel:

```
10 A$ = "ANFANG BAND"
20 B = 3.1416
30 C = 50
40 D$ = "DATEN"
50 PRINT #-1, A$, B, C, D$, "ENDE DER DATEI"
60 END
```

Dieses Programm weist zunaechst den Variablen A$, B, C und D$ Werte zu und schreibt sie dann auf Recorder Nr. 1. Beachten Sie, dass die Zeichenkettenkonstante "ENDE DER DATEI" genauso auf Band geschrieben wird, wie die Variablen. Sind die Daten einmal auf Band gespeichert, so koennen sie zurueck gelesen werden. Das ist im Prinzip der gleiche Vorgang, wie beim Abspielen einer Musikkassette. Bitte beachten Sie, dass bei dem INPUT Statement, das die Daten wieder einlesen soll, die Variablentypen mit den Datentypen auf Blank uebereinstimmen muessen (analog zur Eingabe ueber Tastatur mit INPUT). Die Namen der Variablen duerfen natuerlich verschieden sein.

WICHTIG:
Die gesamte Anzahl der einzelnen Zeichen in allen Variablen und Konstanten der Itemliste eines PRINT # Statements darf 255 nicht uebersteigen. Werden mehr als 255 Zeichen angegeben, so werden die Restlichen ignoriert.

Beispiel:

```
10 PRINT #-1, A$, B$, C$, D$, E$
```

Nehmen wir an, die gesamte Anzahl der Zeichen in A$, B$, C$ und D$ betraegt 250 und E$ habe die Laenge 35. E$ wird nicht auf Band gespeichert, der Versuch, E$ dann spaeter wieder einzulesen, erzeugt einen "keine Daten mehr" Fehler. (out of data error: Code: OD)

3.10 INPUT #-Kassettennummer, Itemliste

Dies Statement veranlasst den Computer, Werte von dem angegebenen Rekorder zu lesen und sie in die Variablen der Itemliste zu schreiben.

Beispiel:

```
10 INPUT #-1, A$, B, C, D$
```

Dieses Statement liest Daten von Rekorder Nr. 1. Der erste Wert wird A$ zugewiesen, der zweite B usw . Die PLAY Taste beim Rekorder muss gedrueckt sein. Sobald der Computer dieses Statement ausfuehrt, schaltet er den Recorder ein und wenn er fertig ist, schaltet er ihn wieder aus und geht zum naechsten Statement.

Wird mit INPUT versucht, eine Zeichenkette vom Band in eine numerische Variable zu lesen, wird eine "schlechte Datei" (bad file, Code: FD) Fehlermeldung ausgegeben. Eine "keine Daten mehr" (out of Data, Code: OD) Fehlermeldung tritt auf, wenn nicht genug Daten fuer alle Variablen in der Itemliste auf Band stehen.

3.11 DEFINT Buchstabenbereich

Variablen, die mit einem Buchstaben aus Buchstabenbereich beginnen, werden als ganze Zahlen (integers) behandelt und abgespeichert. Eine Typenangabe (mit %, $ usw.) aber ueberschreibt diese Typendefinition. Erklaert man eine Variable zur ganzzahligen Variable, so spart man damit nicht nur Speicherplatz sondern auch Computerzeit. Berechnungen mit ganzzahligen Variablen sind schneller als solche mit Variablen einfacher und doppelter Genauigkeit. Beachten Sie, dass eine ganzzahlige Variable nur Werte zwischen –32768 und +32767 annehmen kann.

Beispiel:

```
10 DEFINT X, Y, Z
```

Hat der Computer Zeile 10 ausgefuehrt, so werden alle Variablen, die mit X, Y oder Z anfangen, als ganzzahlig behandelt. Daher sind von da an X2, X3, YA, YB, ZI, ZJ ganzzahlige Variablen. Jedoch X1#, YA#, YB# ZI#, ZJ# bleiben Variablen doppelter Genauigkeit weil die Typenfestlegung mit "#" die mit DEFINT ueberschreibt.

Beispiel:

```
10 DEFINT A – D
```

Erklaert alle Variablen, die mit A, B, C, oder D beginnen zu ganzzahligen Variablen.
Beachten Sie, das DEFINT zwar an beliebiger Stelle im Programm eingesetzt werden kann, aber es die Bedeutung von Variablen ohne expliziter Typendeklaration ($, #, %) recht unkontrolliert aendern (Sprungbefehle) kann. Deshalb sollte es normalerweise an den Anfang eines Programmes gestellt werden.

3.12 DEFSNG Buchstabenbereich

Variablen, die mit einem Buchstaben aus 'Buchstabenbereich' beginnen, werden als Variablen einfacher Genauigkeit behandelt. Explizite Typfestlegung (mit #, $, %) ueberschreibt diese Definition.
Variablen und Konstanten einfacher Genauigkeit werden mit 7 Stellen gespeichert und mit 6 Stellen ausgegeben. Alle numerischen Variablen haben einfache Genauigkeit, wenn nichts anderes angegeben wurde. Das DEFSNG Statement dient vor allem dazu, Variablen neu zu definieren, die vorher als Variablen doppelter Genauigkeit oder ganzzahlige Variablen festgelegt wurden.

Beispiel:

```
10 DEFSNG A–D, Y
```

Macht alle Variablen, die mit A bis D oder Y beginnen, zu Variablen einfacher Genauigkeit. Aber A# bleibt weiterhin eine Varible doppelter Genauigkeit und Y% bleibt eine ganzzahlige Variable.

3.13 DEFDBL Buchstabenbereich

Variablen, die mit einem Buchstaben aus 'Buchstabenbereich' beginnen, werden als Variablen doppelter Genauigkeit behandelt und gespeichert. Eine explizite Typfestlegung (mit!, $, %) kann diese Definition ueberschreiben. Variablen mit doppelter Genauigkeit ermoeglichen das Rechnen mit 17 Stellen, wovon 16 ausgegeben werden.

Beispiel:

```
10 DEFDBL M–P, G
```

Macht alle Variablen, die mit M bis P oder G beginnen zu Variablen doppelter Genauigkeit. (M% oder G# bleiben unbeeinflusst).

3.14 DEFSTR Buchstabenbereich

Variablen, die mit einem Buchstaben aus 'Buchstabenbereich' beginnen, werden als Zeichenkettenvariablen behandelt und abgespeichert.
Explizite Typenfestlegung (mit #, !, %) ueberschreibt diese Definition. Wenn genug Platz fuer Zeichenketten zur Verfuegung steht (siehe CLEAR), kann eine Zeichenkettenvariable bis zu 255 Zeichen aufnehmen.

Beispiel:

```
10 DEFSTR A–D
```

Macht alle Variablen, die mit einem Buchstaben von A bis D beginnen zu Zeichenkettenvariablen, ausser es wurde eine explizite Typfestlegung angegeben (#, !, %). Nach der Ausfuehrung von Zeile 10, wird folgender Ausdruck richtig: B3 = "EINE ZEICHENKETTE".

3.15 CLEAR n

Dieses Statement setzt alle Variablen auf Null. Wird eine Zahl n angegeben, so stellt der Computer n Bytes zur Speicherung von Zeichenketten ab.
Wird das Video Genie System eingeschaltet, so werden jedes Mal automatisch 50 Bytes zur Speicherung von Zeichenketten freigesetzt.
Das CLEAR Statement wird dann kritisch, wenn der Computer waehrend der Programmausfuehrung auf einen "keine Platz fuer Zeichenketten" (out of stringspace, Code OS) Fehler stoesst. Dieser Fehler tritt auf, wenn die Zeichenketten des Programms mehr Platz beanspruchen, als freigegeben wurde.

Beispiel:

```
10 CLEAR 1000
```

Setzt 1000 Bytes fuer Zeichenketten frei und loescht alle Variablen.

3.16 DIM Name (diml, ..., dim n)

Diese Statement legt eine Variable oder eine Liste von Variablen als Feld (array) fest. Die Zahl der Elemente jeder Dimension koennen mit dim1, dim2 usw. angegeben werden. Wird dim n nicht angegeben, so wird angenommen, das jede Dimension 11 Elemente hat. Die Anzahl der moeglichen Dimensionen ist nur durch den Speicherplatz begrenzt.

Beispiel:

```
10 DIM A(5), B(3, 4), C( 2, 3, 3)
```

Dieses Statement definiert A als eindimensionales Feld (Vektor) mit 6 Elementen (0 bis 5), B als zweidimensionales Feld mit 20 Elementen (4 mal 5) und C als dreidimensionales Feld mit 48 Elementen (3 mal 4 mal 4).

DIM darf an beliebiger Stelle in das Programm gesetzt werden. Die Dimensionsangabe darf ganzzahlig oder ein Ausdruck sein.

Beispiel:

```
10 INPUT "ANZAHL DER PUNKTE"; N
20 DIM A(N+2, 4)
```

Die Anzahl der Elemente von A kann, abhaengig von N, veraendert werden.
Um ein Feld neu zu dimensionieren, muss vorher ein CLEAR Statement ohne Argument n eingegeben werden, sonst wird eine Fehlermeldung ausgegeben.

Beispiel:

```
10 X(2) = 13.6
20 PRINT "DAS ZWEITE ELEMENT IST:"; X(2)
30 DIM X(15)
40 PRINT X(2)
50 END

READY
>RUN

? DD ERROR IN 30
```

Durch den Gebrauch von X(2) in 10, vor dem DIM in 30, wurde es implizit mit DIM X(10) dimensioniert.

3.17 LET Variable = Ausdruck

Dieses Statement ordnet einer Variablen einen Wert zu. Das Wort LET ist in einem solchen Zuordnungsstatement beim Video Genie System BASIC nicht unbedingt erforderlich, wurde aber in seinen Sprachumfang aufgenommen, um Kompatibilitaet mit anderen Systemen zu gewaehrleisten.

Beispiel:

```
10 LET A = 5.67
20 B% = 20
30 S$ = "ZEICHEN"
40 LET D% = D% + 1
50 PRINT A, B%, S$, D%
60 END

READY
>RUN

5.67            20          ZEICHEN          1
```

In den obigen Zuordnungen wird in die Variable rechts vom Gleichheitszeichen der Wert des Ausdrucks links vom Gleichheitsszeichen, geschrieben. All diese Statements sind korrekt.

3.18 END

Diese Statement bewirkt die Beendigung der Programmausfuehrung. Das End Statement wird primaer zur Beendigung von Programmen an einer anderen Stelle als am Ende ihres logischen Textes eingesetzt.

Beispiel:

```
5  B = 3 : C = 14
10 A = C + B
20 GOSUB 70
30 D = X + Y
40 PRINT "DAS ERGEBNISS IST:";
50 PRINT A, D
60 END
```

```
70 X = 50
80 Y = A * X
90 RETURN

DAS ERGEBNISS IST : 17                    900
```

Das END Statement in Zeile 60 haelt den Computer davon ab, Zeile 70 bis 90 auszufuehren. Damit kann das Unterprogramm, das in Zeile 70 beginnt nur von der Zeile 20 ausgefuehrt werden.

3.19 STOP

Dieses Statement ist eine grosse Hilfe bei der Fehlersuche in Programmen. Es setzt einen Unterbrechungspunkt (break point) in der Programmausfuehrung und erlaubt dann, Werte zu veraendern oder auszugeben. Fuehrt der Computer einen STOP Befehl aus, so gibt er die Nachricht BREAK IN LINE Zeilennummer (Unterbrechung in Zeile . . .) aus. Mit dem aktiven Kommando CONT koennen Sie die Programmausfuehrung an dem Punkt, an dem vorher unterbrochen wurde, wieder aufnehmen.

Beispiel:

```
5 INPUT B, C
10 A = B + C
20 STOP
30 X = (A + D) / 0.74
40 IF X < 0 GOTO 70
50 PRINT A, B, C
60 PRINT X
70 END

READY
>RUN

? 2, 4
BREAK IN 20
READY
>PRINT A
6
READY
>CONT

6             2             4
8.10811
```


Das Stop Statement gab uns die Moeglichkeit, den Wert von A zu betrachten, bevor Zeile 30 ausgefuehrt wurde.

3.20 GOTO Zeilennummer

Dieses Statement uebergibt die Kontrolle ueber das Programm an die angegebene Zeile (Sprungbefehl). Wird es unabhaengig benutzt, so wird ein unbedingter Sprung ausgefuehrt. Ein Bedingungsstatement kann vor das GOTO gesetzt werden, um einen bedingten Sprung zu erzeugen.

Beispiel:

```
10 A = 10
20 B = 45
30 C = A + B
40 C = C * 3.4
50 GOTO 100
60 .
70 .
80 .
90 .
100 PRINT "A ="; A, "B ="; B, "C ="; C
110 END

READY
>RUN

A = 10                B = 45                C = 187
```

Wird Zeile 50 ausgefuehrt, so wird die Kontrolle an Zeile 100 uebergeben.

Beispiel:

```
10 IF A = 2 GOTO 120
```

Wird Zeile 10 ausgefuehrt und A ist gleich 2, dann springt der Computer zur Zeile 120. Ist A nicht gleich 2, geht er zum naechsten Statement.

Man kann das GOTO auch auf der aktiven Kommandoebene als Alternative zum RUN benutzen. Mit diesem Vorgehen vermeidet man, dass die Variablen geloescht werden.

3.21 GOSUB Zeilennummer

Übergibt die Kontrolle an die Zeile, in der das angegebene Unterprogramm beginnt. Führt der Computer dann in dem Unterprogramm ein RETURN Statement aus, so kehrt er an die dem GOSUB folgende Zeile im (Haupt) Programm zurück. Wie beim GOTO darf auch beim GOSUB ein Bedingungsstatement vorausgehen, wie etwa:

IF A = B THEN GOSUB 100

Beispiel:

```
10 PRINT "HAUPTPROGRAMM"
20 GOSUB 50
30 PRINT "HAUPTPROGRAMM ENDE"
40 END
50 PRINT "UNTERPROGRAMM"
60 RETURN

READY
>RUN

HAUPTPROGRAMM
UNTERPROGRAMM
HAUPTPROGRAMM ENDE
```

3.22 RETURN

Dieses Statement beendet ein Unterprogramm und uebergibt die Kontrolle an das Statement, das dem GOSUB folgt. Ein Fehler tritt auf, wenn fuer ein RETURN kein entsprechendes GOSUB vorhanden war.

3.23 ON n GOTO Zeilennummern Liste

Dieses Statement erlaubt es, gleich mehrere sprungziele anzugeben. Gesprungen wird in Abhaengigkeit von n. Das generelle Format des ON n GOTO ist:

ON Ausdruck GOTO 1. Zeilennummer, 2. Zeilennummer . . .

Der Wert von Ausdruck muss zwischen 0 und 255 einschl. sein.

Wird ein ON – GOTO Statement ausgefuhert, so wird zunaechst der ganzzahlige Anteil von Ausdruck berechnet (entspricht INT (Ausdruck)). Dieses Ergebniss sei n. Dann sucht der Computer das n te Element der Zeilennummern Liste und springt zu dieser Zeilennummer. Ist n nun groesser als die Anzahl der angegebenen Zeilennummern, so wird das, auf das ON – GOTO Statement folgende Statement ausgefuehrt. Ist n kleiner als Null, so tritt ein Fehler auf. Die Zeilennummern Liste kann eine beliebige Anzahl von Zeilennummern enthalten.
Beispiel:

```
10 INPUT "KOMMANDO EINGEBEN"; C
20 ON C GOTO 100, 120, 130, 150, 130
30 PRINT "ENDE DES PROGRAMMS" : END
100 PRINT "HIER ZEILE 100" : GOTO 10
120 PRINT "HIER ZEILE 120" : GOTO 10
130 PRINT "HIER ZEILE 130" : GOTO 10
150 PRINT "HIER ZEILE 150" : GOTO 10
READY
>RUN
KOMMANDO EINGEBEN ? 5
HIER ZEILE 130
KOMMANDO EINGEBEN ? 4
HIER ZEILE 150
KOMMANDO EINGEBEN ? 1
HIER ZEILE 100
KOMMANDO EINGEBEN ? 2
HIER ZIELE 120
KOMMANDO EINGEBEN ? 3
HIER ZEILE 130
KOMMANDO EINGEBEN ? 0
ENDE DES PROGRAMMS
READY
>RUN
KOMMANDO EINGEBEN ? 4
HIER 1ST ZEILE 150
KOMMANDO EINGEBEN ? 6
ENDE DES PROGRAMMS
```

Das ON – GOTO Statement ist eine elegante Methode, um dass gleiche zu erreichen, was folgende IF-GOTO Statements bewirken:

```
10 IF C = 1 GOTO 100
20 IF C = 2 GOTO 120
30 IF C = 3 GOTO 130
40 IF C = 4 GOTO 150
50 IF C = 5 GOTO 130
60 IF C < 1  OR  C > 5 GOTO 70 : REM GEHE ZUM NAECHSTEN STATEMENT
```

3.24 ON n GOUSUB Zeilennummern Liste

Arbeitet wie ON n GOTO, nur statt der Spruenge werden Unterprogramme aufgerufen.
Beispiel:

```
10 PRINT "** FUNKTION DER UNTERPROGRAMME **"
20 PRINT "    1. FUNKTION A"
30 PRINT "    2. FUNKTION B"
40 PRINT "    3. FUNKTION C"
50 INPUT "GIB 1, 2 ODER 3 EIN"; N
60 ON N GOSUB 150,100,250
70 END
100 PRINT "HIER FUNKTION B" : RETURN
150 PRINT "HIER FUNKTION A": RETURN
250 PRINT "HIER FUNKTION C" : RETURN
READY
>RUN
** FUNKTION DER UNTERPROGRAMME **
   1. FUNKTION A
   2. FUNKTION B
   3. FUNKTION C
GIBT 1, 2 ODER 3 EIN ? 2
HIER 1ST FUNKTION B
READY
>RUN
** FUNKTION DER UNTERPROGRAMME **
   1. FUNKTION A
   2. FUNKTION B
   3. FUNKTION C
GIBT 1, 2 ODER 3 EIN ? 1
HIER IST FUNKTION A
```

3.25 FOR Name = Ausdruck TO Ausdruck STEP Ausdruck
NEXT Name

Diese Statements bilden eine iterative Schleife. Alle Statements, die zwischen FOR und NEXT stehen, werden einige Male ausgefuehrt.
Das allgemeine Format ist:

```
FOR Zaehler = Anfangswert TO Endwert STEP Inkrement
*
*
*
Statements
*
*
*
NEXT Zaehler
```

Beim For Statement koennen Anfangswert, Endwert und Inkrement (Wert, um den hochgezaehlt wird) Konstanten, Variablen oder Ausdruecke sein. Wird das FOR Statement das erste Mal ausgefuehrt, so werden sie berechnet und gespeichert. Veraendern sich diese Werte in der Schleife, so hat das keine Auswirkungen auf die Arbeitsweise der Schleife selbst. Aber der Wert des Zaehlers darf nicht veraendert werden, sonst arbeitet die Schleife nicht korrekt.
Die FOR – NEXT Schleife funktioniert wie folgt:
Wird das FOR Statement zum ersten Mal ausgefuehrt, so wird der Zaehler auf den Anfangswert gesetzt. Das Programm wird weiter ausgefuehrt, bis ein NEXT Statement gefunden wird. Jetzt wird der Zaehler um den Wert von Inkrement, der nach dem STEP angegeben wurde, erhoeht. Wird STEP und Inkremet nicht angegeben, so wird automatisch fuer Inkrement eine 1 angenommen. Ist Inkrement negativ, so wird der Zaehler herunter gezaehlt. Der Zaehler wird dann mit dem Endwert des FOR Statements verglichen. Ist der Zaehler groesser als der Endwert, so wird die Ausfuehrung der Schleife eingestellt und die Programmausfuehrung geht nach dem NEXT Statement weiter. (ist Inkrement negativ, so endet die Ausfuehrung, wenn der Zaehler kleiner als der Endwert ist) Hat der Zaehler den Endwert noch nicht ueberschritten, so geht der Computer zurueck zum ersten Statement nach dem FOR Statement.

Beispiel:

```
10 FOR K = 0 TO 1 STEP 0.3
20 PRINT "WERT VON K:"; K
30 NEXT K
40 END

READY
>RUN

DER WERT VON K : 0
DER WERT VON K : .3
DER WERT VON K : .6
DER WERT VON K : .9
```

Dann ist K = 1.2 und damit groesser als der Endwert 1. Deshalb endet die Schleife dort und druckt 1.2 nicht mehr.

Beispiel:

```
10 FOR N = 5 TO 0
20 PRINT "DER WERT VON N:"; N
30 NEXT N
40 END
READY
>RUN
DER WERT VON N : 5
10 FOR N = 5 TO 0 STEP -1
20 PRINT "DER WERT VON N:"; N
30 NEXT N
40 END
READY
>RUN
DER WERT VON N : 5
DER WERT VON N : 4
DER WERT VON N : 3
DER WERT VON N : 2
DER WERT VON N : 1
DER WERT VON N : 0
```

Wird kein STEP angegeben, so wird automatisch STEP 1 angenommen. N wird das erste Mal hochgezaehlt, wird zu N=6 und ist damit groesser als der Endwert 0; die Schleife endet. Der Vorgang verlaeuft anders, wenn STEP -1 angegeben wurde (s.o.)

Beispiel:

```
10 FOR A = 0 TO 3
20 PRINT "DER WERT VON A :"; A
30 NEXT
40 END
READY
>RUN
DER WERT VON A : 0
DER WERT VON A : 1
DER WERT VON A : 2
DER WERT VON A : 3
```

Statt NEXT A koennen wir, wie in Zeile 30, auch einfach NEXT schreiben. Bei der Programmierung von verschachtelten FOR NEXT Schleifen ist es jedoch guenstig, anzugeben, welche Schleife man nun mit dem konkreten NEXT abgeschlossen hat.

Hier ist ein Beispiel fuer geschachtelte Schleifen, dass zeigen soll, wie man den Zaehler in jedem NEXT Statement identifiziert:

```
10 I = 1
20 J = 2
30 K = 3
40 FOR N = I+1 TO J+1
50 PRINT "ERSTE SCHLEIFE"
60 FOR M = I TO K
70 PRINT "      ZWEITE SCHLEIFE"
80 NEXT M
90 NEXT N
100 END

READY
>RUN

ERSTE SCHLEIFE
   ZWEITE SCHLEIFE
   ZWEITE SCHLEIFE
   ZWEITE SCHLEIFE
ERSTE SCHLEIFE
   ZWEITE SCHLEIFE
   ZWEITE SCHLEIFE
   ZWEITE SCHLEIFE
```

3.26 ERROR Code

Dieses Statement wird benutzt, um in eine ON ERROR GOTO Routine zu verzweigen. Fuehrt der Computer ein ERROR Code Statement aus, so verhaelt er sich genau so, als waere dieser Fehler aufgetreten. Er simuliert die Fehlersituation.

Beispiel:

```
30 ERROR 1

READY
>RUN

? NF ERROR IN 30
```

Die Bedeutung der Error (=Fehler) Codes finden Sie in Anhang B.

3.27 ON ERROR GOTO Zeilennummer

Dieses Statement erlaubt es, ein Abfangprogram fuer Fehler (engl. error trapping routine) aufzurufen. Damit kann man, kommt es bei der Programmausfuehrung zu einem Fehler, erreichen, dass das Programm nicht mit einer Fehlermeldung anhaelt, sondern in ein Programm verzweigt, das die Ursache des Fehlers kennt und beseitigt. Meist hat der Benutzer einen bestimmten Fehlertyp im Auge, wenn er das ON ERROR GOTO Statement verwendet. Fuehrt das Programm z.B. eine Division durch und der Fall der Division durch Null ist nicht von vorn herein ausgeschlossen worden, so kann die verbotene Division durch Null mit einem Fehlerbehandlungs Programm abgehandelt werden.

Beispiel:

```
5  B = 15 : C = 0
10  ON ERROR GOTO 120
20  A = B/C
30  PRINT A, B, C
40  END
120  PRINT "MAN DARF NICHT DURCH NULL TEILEN!!"
130  END

READY
>RUN

MAN DARF NICHT DURCH NULL TEILEN!!
```

In diesem Beispiel hat C den Wert 0 und in Zeile 20 kommt es zu einer Division durch Null, was normalerweise eine Fehlermeldung und Beendigung des Programms zur Folge haette. Wegen Zeile 10 ignoriert der Computer aber diese Situation einfach und geht in die Zeile 120, zur Fehlerbehandlungroutine. Beachten Sie, dass das ON ERROR GOTO Statement vor dem moeglichen Auftreten des Fehler stehen muss, sonst hat es keine Wirkung. Ausserdem muss die Fehlerbehandlungsroutine mir RESUME abgeschlossen werden.

3.28 RESUME Zeilennummer

Dieses Statement beendet eine Fehlerbehandlungsroutine und gibt an, wo die normale Programmausfuherung weiter gehen soll.
RESUME 0 oder RESUME ohne Zeilennummer bewirkt, dass der Computer zu dem Statement zurueckkehrt, in dem es zu einem Fehler gekommen war. Ist eine Zeilennummer angegeben, so geht er zu dieser Zeile Zurueck.
RESUME NEXT Veranlasst den Computer, zu der Zeile zurueck zu gehen, die hinter der Zeile steht, in der der Fehler erkannt wurde.

Beispiel:

```
10 ON ERROR GOTO 80
20 PRINT "EINFACHE DIVISION."
30 INPUT "GIB ZWEI ZAHLEN EIN"; A, B
40 IF A = 0 END
50 C = A/B
60 PRINT "DER QUOTIENT IST :"; C
70 GOTO 20
80 PRINT "VERSUCH, DURCH NULL ZU TEILEN"
90 PRINT "VERSUCHEN SIE ES NOCH EINMAL . . ."
100 RESUME 20

READY
>RUN

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN ? 6, 2
DER QUOTIENT IST : 3

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN ? 7, 3
DER QUOTIENT IST : 2.33333

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN ? 5, 0
VERSUCH, DURCH NULL ZU TEILEN
VERSUCHEN SIE ES NOCH EINMAL . . .
EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN ? 9, 4
DER QUOTIENT IST : 2.25

EINFACHE DIVISION
GIB ZWEI ZAHLEN EIN ? 0, 0
```

3.29 REM

REM gibt Fussnoten an. Dieses Statement informiert den Computer, dass in dieser Zeile nur noch Kommentare folgen. Diese werden vom Programmablauf ignoriert. Es ermoeglicht dem Benutzer, durch die Kommentare uebersichtlichere Programme zu schreiben.

Beispiel:

```
10 REM *   BEDEUTUNG DER VARIABLEN    *
20 REM *     A = AUFWAND               *
20 REM *     B = TEILEZAHL             *
40 REM *     C = KOSTEN DER EINHEIT    *
50 REM - - - - - - - - - - - - - - - - - - - -
60 A = B * C : REM AUFWAND = ZAHL DER TEILE MAL KOSTEN
```

3.30 IF Ausdruck Aktion

Dieses Statement bringt den Computer dazu, einen logischen oder relationalen Ausdruck zu entscheiden. Ist der Ausdruck "wahr", so wird Aktion ausgefuehrt, ist er falsch, so wird Aktion ignoriert und mit dem naechsten Statement fortgefahren. Numerisch bedeutet "wahr", dass der Wert von Ausdruck nicht Null ist.

Beispiel:

```
10 INPUT "GIB ZAHL EIN (MAX. 20)"; A
20 IF A > 20 GOTO 60
30 A = A * 3.1416 * 2
40 PRINT "DER KREISUMFANG IST :"; A
END
60 PRINT "ZAHL ZU GROSS, MAX. 20" : GOTO 10

READY
>RUN

GIB ZAHL EIN (MAX. 20) ? 24
ZAHL ZU GROSS, MAX 20
GIB ZAHL EIN ? 18
DER KREISUMFANG IST : 113.098
```

In diesem Beispiel wird jedes Mal, wenn eine Zahl groesser als 20 eingegeben wird, eine Warnung ausgegeben und eine neue Eingabe erwartet. Ist A jedoch kleiner oder gleich 20, ignoriert der Computer das GOTO 60 in Zeile 20 und beendet seine Berechnung ohne eine Warnung auszugeben.

Beispiel:

```
120 INPUT A : IF A = 10 AND A < B THEN 160
120 INPUT A : IF A = 10 AND A < B GOTO 160
```

Beide Statements haben die gleiche Wirkung.

3.31 THEN Statement oder Zeilennummer

Bezeichnet den Beginn der "Aktion" in einem IF-THEN Statement. THEN kann entfallen, wenn es benutzt wird, um eine Zeilennummer als Sprungziel anzugeben, wie in: IF A=B THEN 100. THEN sollte in IF-THEN-ELSE Statement angegeben werden.

3.32 ELSE Statement oder Zeilennummer

Dieses Statement kann nur hinter einem IF Statement stehen und gibt eine Alternative an, wenn der Ausdruck im IF Statement "falsch" wird.

Beispiel:

```
10 IF A = 1 THEN 60 ELSE 40
```

In diesem Beispiel wird nach 60 verzweigt, wenn A gleich 1 ist, ist dies nicht der Fall, wird nach 40 verzweigt. IF-THEN-ELSE Statements koennen verschachtelt werden, aber die Anzahl der IFs und ELSEs mussen uebereinstimmen.

Beispiel:

```
10 INPUT "GIB DREI ZAHLEN EIN"; X, Y, Z
20 PRINT "DIE GROESSTE ZAHL IST :";
30 IF X < Y  OR  X < Z  THEN IF Y < Z  THEN PRINT Z ELSE PRINT Y ELSE PRINTX

40 END

READY
>RUN

GIB DREI ZAHLEN EIN ? 30, 75, 73
DIE GROESSTE ZAHL IST : 75
```

Das Programm erwartet drei Zahlen als Eingabe und gibt die groesste der drei aus.

3.33 LPRINT

Gibt Daten auf dem Drucker aus. Dieses Statement (auch Kommando) arbeitet analog zum PRINT Statement. Ist kein Drucker angeschlossen, so geraet der Computer bei Ausfuehrung dieses Statements in eine endlose Schleife, aus der er nur mit dem RESET Knopf befreit werden kann.

Beispiel:

```
10 FOR X = 1 TO 0 STEP –0.25
20 LPRINT "X BETRAEGT :"; X
30 NEXT X
40 END

READY
>RUN

X  BETRAEGT: 1
X  BETRAEGT: .075
X  BETRAEGT: .5
X  BETRAEGT: .25
X BETRAEGT: .0
```

KAPITEL 4

VERARBEITUNG VON FELDERN

Ein Feld (array) ist einfach eine geordnete Liste von Daten. Das Video Genie System verarbeitet numerische und auch Zeichenketten Felder. Beide Datentypen koennen aber nicht im gleichen Feld gemischt auftreten. Die Konzepte der Programmierung von Feldern sind sehr wichtig fuer die Computeranwendung und deshalb sollte der Benutzer versuchen, die Beispiele dieses Kapitels zu verstehen.

Nehmen wir an, Fritz Walter studiert an einer Universitaet. Das Gebaeude hat drei Stockwerke, von denen jedes vier Klassenraeume beherbergt. Jeder Raum hat 45 Plaetze. Fritz hat eine Vorlesung in Geschichte belegt und es sind nur 36 Studenten mit ihm in der Klasse. Sehen wir uns die Namesliste von Fritzs Klasse an:

NAMELISTE

1. Maria Adam
2. Jupp Braun
3. Heinrich Cox

*
*
*

36. Fritz Walter

Um eine Person in der Liste zu finden, muessen wir die Liste lediglich von oben nach untendurchlesen. Die Methode, mit der der Name gesucht wird, ist nicht sonderlich wichtig.

Wichtig ist, wie man eine Person in der Liste finden kann, wenn man nur ihre Nummer kennt. In der obigen Liste ist Maria Adam die erste, Jupp Braun die zweite Person usw . . Diese Zahlen geben also einen systematischen Weg an, um eine Person zu finden.

Benutzen wir den Computer, um die Namensliste aufzuzeichnen, koennen wir jedem Namen eine einheitliche Variable zuweisen:

```
10  N0$ = "MARIA ADAM"
20  N1$ = "JUPP BRAUN"
30  N2$ = "HEINRICH COX"
.
.
.
100  NS$ = "TOMAS HAGEN"
.
.
.
140  NZ$ = "FRITZ WALTER"
```

Bedenkt man, dass 37 Studenten in der Klasse sind, so sieht man, dass dies eine uneffektive und zeitraubende Methode ist.
Sehen wir das Programm genauer an, so stellen wir fest, dass alle Variablen laufende Nummern angehaengt bekommen haben. Das haben wir einfach intuitiv und von Hand gemacht. Erzeugt der Computer nun diese laufenden Nummern selbst, so ist das ein Feld (array). Wir definieren ein Feld AR$ mit 45 Elementen (45 Plaetze) und ordnen jedem Element einen Namen zu.

Beispiel:

```
5  CLEAR 1000 :              REM SETZE 1000 BYTES FUER ZEICHENKETTEN FREI
10  DIM AR$ (44) :           REM FELD AR$ HAT 45 ELEMENTE
20  FOR N = 0 TO 44 :        REM 45 SCHLEIFENDURCHLAEUFE
30  INPUT "GIB DEN NAMEN DES STUDENTEN EIN"; AR$ (N)
40          REM ORDNE DEN FELDELEMENTEN DIE NAMEN ZU
50  NEXT N
60  END
```

Dieses Programm liest 45 Namen ein und speichert sie in dem Feld AR$. Nach der Programmausfuehrung haben wir folgende Variablenbelegung:

Das Element AR$ (0) hat den Wert "Maria Adam"
Das Element AR$ (1) hat den Wert "Jupp Braun"
Das Element AR$ (2) hat den Wert "Heinrich Cox"
.
.
.
Das Element AR$ (36) hat den Wert "Fritz Walter"

Wollen wir die Liste nun ausdrucken, koennen wir folgendes Programm einsetzen:

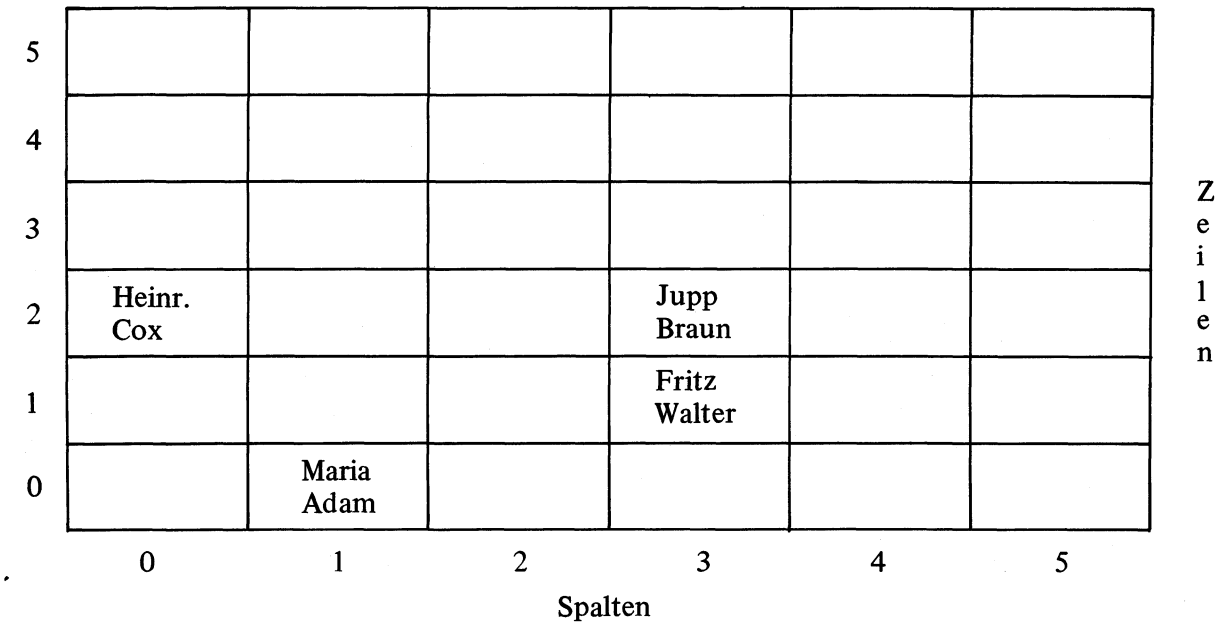
```
5  CLEAR 1000 :            REM SETZE 1000 BYTES FUER ZEICHENKETTEN FREI
10  DIM AR$ (44) :         REM FELD AR$ SOLL 45 ELEMENTE BEKOMMEN
20  REM  **  EINGABE DER FELDELEMENTE  **
20  FOR N = 0   TO  44 : REM  45 SCHLEIFENDURCHLAEUFE
30  INPUT "GIB DEN NAMEN DES STUDENTEN EIN"; AR$ (N)
40  REM ORDNE DEN FELDELEMENTEN DIE NAMEN ZU
50  NEXT N
60  REM  **  AUSDRUCKEN DER FELDELEMENTE  **
70  FOR N = 0  TO 44 :      REM WIEDER DIE 45 MAL SCHLEIFE
80  PRINT AR$ (N) :         REM DRUCKE DIE FELDELEMENTE (NAMEN)
90  NEXT N
100  END
```

Der Ausgabeteil des Programms ersetzt folgende 45 PRINT Statements:

```
10  PRINT N0$
20  PRINT N1$
30  PRINT N2$
40  .
50  .
60  .
70  PRINT NS$
80  .
90  .
100  .
110  PRINT NZ$
```

Nun sollten Sie ein Gefuehl fuer die Nuetzlichkeit der Felder bekommen haben.

Angenommen, ein Lehrer dieser Klasse moechte nun einen Sitzplan aufstellen. Es sind 6 Zeilen und 6 Spalten von Sitzplaetzen vorhanden:



Die Sitzpositionen unserer vier Beispiel-studenten sind im obligen Plan eingezeichnet. Da sie sich aber nicht der Namensliste folgend hingesetzt haben, muessen wir uns eine andere Methode ausdenken, um auf den Sitzplan zuzugreifen. Will der Lehrer z.B. festellen, ob Fritz Walter abwesend ist, so muss er nachsehen, ob der Stuhl in Zeile 1, Spalte 3 leer ist oder nicht. Er kann auch in Zeile 2 Spalte 0 nach Heinrich Cox suchen. Der Computer tut nun genau das gleiche, wie dieser Lehrer. Wir koennen diesen Sitzplan in ein 2 dimensionales Feld SP$ (5, 5) abbilden. Die erste 5 steht fuer die Zeilen, die zweite fuer die Spalten. Wollen wir nun Jppu Braun aufrufen, so muessen wir in SP$ (2, 3) nachschauen.

Nun wollen wir unseren Sitzplan ausdrucken. Das leistet folgendes Programm:

```
10 CLEAR 1000 : DIM SP$ (5, 5) : REM SP$ IST 6X6 FELD
20 FOR  R = 5 TO 0 STEP –1
        REM ZEILE 5 BIS ZEILE 0 AUSDRUCKEN
        FOR C = 0 TO 5
            REM SCHLEIFE, UM DIE NAMEN ALLER SPALTEN AUSZUGEBEN
            PRINT SP$ (R, C) : REM DRUCKE NAMEN IN ZEILE R
                                                UND SPAITE C
70      NEXT C
80      PRINT : REM PRINT ALLEIN = ZEILENVORSCHUB
90 NEXT R
100 END
```

Dieses Programm druckt den Sitzplan in Tabellenform. Es beginnt bei der letzten Sitzreihe der Klasse und endet mit der ersten. Das Programm initialisiert zuerst R = 5 und C = 0. Dann druckt es die Werte der Elemente.

SP$ (5, 0); SP$ (5, 1); SP$ (5, 2); SP$ (5, 3) . . . . . SP$ (5, 5)

An diesem Punkt wird der Wert von C=5. Der Computer springt von der inneren "C" Schleife zurueck in die aeussere "R" Schleife. Subtrahiert 1 von R und R wird damit zu 4. C wird wieder zu Null zurueckgesetzt und der Vorgang beginnt mit R=4 neu:

SP$ (4, 0); SP$ (4, 1); SP$ (4, 2); SP$ (4, 3) . . . . . SP$ (4, 5)

Der gesamte Prozess wird wiederholt, bis R = –1. Dann stoppt das Programm. Die endgueltige Liste sieht so aus:

```
SP$ (5, 0); SP$ (5, 1); SP$ (5, 2) . . . .    . . SP$ (5, 5)
SP$ (4, 0); SP$ (4, 1); SP$ (4, 2) . . . .    . . SP$ (4, 5)
SP$ (3, 0); SP$ (3, 1); SP$ (3, 2) . . . .    . . SP$ (3, 5)
SP$ (2, 0); SP$ (2, 1); SP$ (2, 2) . . . .    . . SP$ (2, 5)
SP$ (1, 0); SP$ (1, 1); SP$ (1, 2) . . . .    . . SP$ (1, 5)
SP$ (0, 0); SP$ (0, 1); SP$ (0, 2) . . . .    . . SP$ (0, 5)
```

Mit einem solchen, zweidimensionalen Feld koennen wir jeden Studenten in der Klasse lokalisieren. Aber wie koennte man einen Studenten lokalisieren, der auf dem gleichen Platz wie Fritz Walter, aber in der naechsten Klasse, sitzt? Natuerlich muessen wir dann neben Reihe und Spalte der Sitze noch die Nummer der Klasse angeben. Damit bekommt unser Feld noch eine weitere Dimension. Es gibt naemlich 12 Klassenraeume im Schulgebaeude. Es stehen verschiedene Wege zur Verfuegung, um dieses Problem zu loesen. Der zweite ist, die Raeume nach Stockwerken anzuordnen. 1. Raum im 1. Stock, 1. Raum im 2. Stock usw . . Hier brauchen wir dann zwei weitere Dimensionen (insgesamt: Reihe, Spalte, Raum, Stock).

Angenommen, Fritz's Klassenraum waere der 3. Raum im 2. Stock. Nach der ersten Methode koennten wir Fritz's Platz dann mit SP$ (N, R, C) angeben. Dabei ist: N die Zahl der Raeume, R die Reihe und C die Spalte. Jupp sitzt dann z.B. in SP$ (7, 1, 3) d.h. in Raum 7, Reihe 1, Spalte 3.
Benutzen wir die zweite Methode, bekommen wir: SP$ (F, N, R, C). Dabei ist: F die Nummer des Stockwerks, N die Nummer des Raumes in diesem Stock, R die Reihe und C die Spalte. Um Fritz nun anzusprechen, muessten wir sagen: SP$ (2, 3, 1, 3) d.h.: 2. Stock, 3. Raum, 1. Reihe, 3. Spalte.

Die Anzahl der Dimensioen steigt an, wenn wir noch mehr Studenten in unsere Systematik aufnehmen wollen. Wir koennen noch weitere Dimensionen einfuehren fuer die Gebaeude, die Universitaeten, Staedte, Laender usw . .

In jedem Video Genie System wird die Anzahl der moeglichen Dimensionen nur durch den vorhanden Speicherplatz begrenzt.

KAPITEL 5

Behandlung von Zeichenketten

Zeichenketten haben eine grosse Bedeutung in der Datenverarbeitung. Ein Computer, der keine Zeichenketten verarbeiten kann, ist nichts weiter als ein sehr leistungsfaehiger Taschenrechner. Dieser Erkenntniss folgend, stehen Ihnen im Video Genie System ausser den arithmetischen Funktionen viele, leistungsfaehige ZeichenkettenmanipulationsFunktionen zur Verfuegung.

In diesem Kapitel werden wir diese Ziechenkettenmanipulationsstatements, die in unserer erweiterten BASIC Programmiersprache benutzbar sind, erlaeutern. Wir werden von nun an Zeichenketten mit ihrem englischen Namen bezeichnen, naemlich als **Strings**.

5.1 Vergleichen von Strings (Zeichenketten)

Mit den relationalen Operatoren (=, <, >, usw.) koennen Strings auf ihre Gleichheit ueberprueft und nach ihrer Alfabetischen Groeese verglichen werden. Wird auf Gleichheit geprueft, so muessen alle Zeichen, fuehrende und abschliessende Leerzeichen (blanks) eingeschlossen, uebereinstimmen, sonst wird auf ungleich entschieden.

Beispiel:

```
100 IF A$ = "JA" THEN 250
```

Strings werden Zeichen fuer Zeichen, von links nach rechts, verglichen. Um genau zu sein: die ASCII-Darstellung der Zeichen wird verglichen. Ein Zeichen mit einer hoeheren Codenummer ist groesser als eines mit einer niederigeren. Mit anderen Worten "AC" groesser "AB". Fuer Buchstaben gilt de Alfabetische anfolge (A = Min, Z = Max). Werden Strings verschiedener Laenge verglichen, so wird der kuerzere String als kleiner angenommen, auch wenn seine Zeichen identisch mit denen des laengeren sind. Daher ist "B" Kleiner "B ". Die folgenden relationalen Operatoren koennen zum Vergleichen von Strings eingesetzt werden:

>, <, <=, =>, =, <>

5.2 String Operationen

Es gibt nur eine String Operation. Das ist das Aneinander haengen von Strings. Der Operator ist das "+" Zeichen.

Beispiel:

```
10 S1$ = "DIE SONNE"
20 S2$ = "SCHEINT"
30 S3$ = " , "
40 C$ = S1$ + S2$ + S3$ + S2$ + S3$ + S2$ + ". "
50 PRINT C$
60 END

READY
>RUN

DIE SONNE SCHEINT, SCHEINT, SCHEINT.
```

5.3 ASC (String)

Dieses Statement berechnet den ASCII-Code des ersten Zeichens des angegebenen Strings. Der String muss in Anfuehrungszeichen eingeschlossen sein. Ein leere String als Argument erzeugt eine Fehlermeldung.

Beispiel:

```
100 PRINT "DER ASCII CODE VON 'H' IST:"; ASC ("H")
105 S$ = "HEIM" : PRINT "DER STRING HEISST:"; S$
110 PRINT "DER ASCII CODE DES 1. BUCHSTABENS IST:"; ASC (S$)
120 END

READY
>RUN

DER ASCII CODE VON 'H' IST: 72
DER STRING HEISST: HEIM
DER ASCII CODE DES 1. BUCHSTABEN IST: 72
```

Beide Zeilen drucken die gleiche Zahl.

Ein Liste der ASCII Codes finden Sie in Anhang C

5.4 CHR$ (Ausdruck)

Dieses Statement macht das genaue Gegenteil der ASC Funktion. Es erzeugt das zum Wert von Ausdruck gehoerige ASCII Zeichen. Als Argument darf eine Zahl von 0 bis 255, oder ein Ausdruck mit diesem Wert, auftreten. Das Argument muss in Klammern gesetzt werden.

```
100 PRINT CHR$(33) : REM SCHREIBE EIN AUSRUFEZEICHEN
```

5.5 LEFT$ (String, n)

Das Statement erzeugt die ersten n Zeichen des angegebene Strings. String und n muessen in Klammern stehen. String kann eine Zeichenkettenvariable oder Konstante sein, n kann ein numerischer Ausdruck oder Konstante sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = LEFT$ (A$, 4)
30 PRINT B$
40 END

READY
>RUN

ABCD
```

5.6 RIGHT$ (String, n)

Erzeugt die n letzten Zeichen von String. Die Argumente muessen in Klammern stehen. String kann eine Zeichenkettenvariable oder Konstante sein, n kann eine numerische Variable oder Konstante sein. Ist die Laenge von String kleiner oder gleich n, wird der gesamte String erzeugt.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = RIGHT$ (A$, 3)
30 PRINT B$
40 END

READY
>RUN

EFG
```

5.7 LEN (String)

Gibt die Laenge von String an. String kann Variable, Konstante oder Zeichenkettenausdruck sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 PRINT "DIE LAENGE DER ZEICHENKETTE :"; LEN (A$)
30 END

READY
>RUN

DIE LAENGE DER ZEICHENKETTE: 7
```

5.8 MID$ (String, p, n)

Erzeugt einen Teilstring von String. Er beginnt an Position p und ist n Zeichen lang. Die Argumente muessen in Klammern stehen. String kann Konstante, Variable oder Ausdruck sein, p und n koennen numerische Konstante, Variable oder Ausdruecke sein.

Beispiel:

```
10 A$ = "ABCDEFG"
20 B$ = MID$ (A$, 3, 4)
30 PRINT "DER NEUE STRING HEISST :"; B$
40 END

READY
>RUN

DER NEUE STRING HEISST: CDEF
```

5.9 STR$ (Ausdruck)

Verwandelt eine numerische Konstante oder Ausdruck in ein Zeichen. Das Argument muss in Klammern stehen.

Beispiel:

```
10 A = 34.56
20 B$ = STR$ (A)
30 B$ = B$ + "%"
40 PRINT "DAS ERGEBNISS IST: "; B$

READY
>RUN

DAS ERGEBNISS IST: 34.56%
```

5.10 STRING$ (n, Zeichen oder Zahl)

Erzeugt einen String, der aus n mal dem Zeichen besteht.

Beispiel:

```
10 PRINT STRING$ (10, "*")
20 END

READY
>RUN
**********
```

An Stelle von Zeichen kann man auch eine Zahl von 0 bis 255 angeben. Sie wird als ASCII Code behandelt und das entsprechende Zeichen oder der Graphische Code werden erzeugt.

```
10 PRING STRING$ (10, 35)
20 END

READY
>RUN

##########
```

5.11 VAL (String)

Macht das Gegenteil der STR$ Funktion. Erzeugt einen numerischen Wert aus String.

Beispiel:

```
10 A$ = "56"
20 B$ = "23"
30 C = VAL (A$ + "." + B$)
40 PRINT "DAS ERGEBNISS IST: "; C; " , "; C + 100
50 END

READY
>RUN

DAS ERGEBNISS IST: 56.23, 156.23
```

KAPITEL 6

ARITHMETISCHE FUNKTIONEN

In diesem Kapitel werden wir die fest programmierten, arithmetischen Funktionen des Video Genie Systems besprechen. In den meisten Faellen muss ein Argument an die Funktion weitergegeben werden, bevor der Funktionswert berechnet werden kann. Dieses Argument kann eine numerische Konstante, eine numerische Variable oder ein numerischer Ausdruck sein. Das allgemeine Format ist:

Ergebnis = Funktion (Argument)

Beispiel:

```
10 A = RND (3)
20 B = INT (C)
30 E = SQR (F*G–4)
```

Wir behandeln folgende Funktionen:

1. ABS (X)
2. ATN (X)
3. CDBL (X)
4. CINT (X)
5. COS (X)
6. CSNG (X)
7. EXP (X)
8. FIX (X)
9. INT (X)
10. LOG (X)
11. RANDOM
12. RND (X)
13. SGN (X)
14. SIN (X)
15. SQR (X)
16. TAN (X)

6.1 ABS (X)

Berechnet den Absolutwert von X.

6.2 ATN (X)

Berechnet den Arcustangnes von X (im Bogenmass). Um das Ergebniss in Grad zu erhalten, multipliziere ATN (X) mit 57.29578

6.3 CDBL (X)

Erzeugt die doppelt genaue Darstellung von X. Das Ergebniss enthaelt 17 Stellen, wovon die Stellen, die das Argument enthalten signifikant sind.

6.4 CINT (X)

Berechnet die naechste, ganze Zahl, die kleiner als das Argument ist. Das Argument muss zwischen −32768 und +32767 liegen.

Beipiel: CINT (2.6) = 2; CINT (−2.6) = 3

6.5 COS (X)

Berechnet den Cosinus des Arguments (im Bogenmass). Soll er in Grad berechnet werden: COS (X * .0174533)

6.6 CSNG (X)

Erzeugt die einfach genaue Darstellung von X. Berechnet eine 6 stellige Zahl mit 4/5 Rundung bei doppelt genauem X.

6.7 EXP (X)

Berechnet die Exponential Funktion von X, d.h. e boch X. Das ist die Umkehrfunktion zu LOG (X).

6.8 FIX (X)

Trennt die Nachkommastellen von X ab. FIX (1.5) = 1; FIX (−1.5) = −1

6.9 INT (X)

Erzeugt die ganzzahlige Darstellung von X. Berechnet die groesste, ganze Zahl, die nicht groesser als X ist. X muss zwischen −32768 und 32767 liegen.

Beispiel: INT (3.5) =3; INT (−3.5) ist gleich −4

6.10 LOG (X)

Berechnet den natuerlichen Logarithmus von X. d.h. $\log_e(X)$. Um einen Logarithmus zu einer anderen Basis zu berechnen, benutzen Sie folgende Formel:

$\log_b(X) = \log(X)_e / \log_e(b)$ (b ist die Basis)

6.11 RANDOM

Fuehrt der Computer diese Funktion aus, so erzeugt er einen neuen Vorrat an Zufallszahlen. Diese Funktion benoetigt keine Argumente.

6.12 RND (X)

Erzeugt eine Pseudozufallszahl aus der aktuellen Menge von Zufallszahlen. (Werden intern generiert, kein Zugriff vom Benutzer moeglich)
RND (0) erzeugt eine einfach genaue Zufallszahl zwischen 0 und 1
RND (X) erzeugt eine ganze Zahl zwischen 1 und X
X muss kleiner als 32768 und positiv sein.

6.13 SGN (X)

Die Vorzeichenfunktion. Sie erzeugt eine −1, wenn X negativ ist, Null, wenn X Null ist und +1, wenn X positiv ist.

6.14 SIN (X)

Berechnet die Sinusfunktion von X (im Bogenmass)
Der Sinus in Grad: SIN (X * .0174533)

6.15 SQR (X)

Berechnet die Quadratwurzel von X.

6.16 TAN (X)

Berechnet die Tangensfunktion von X. (im Bogenmass)
TAN (X) in Grad: TAN (X * .0174533).

KAPITEL 7

GRAFISCHE FUNKTIONEN

Das Viedo Genie System verfuegt ueber vier grafische Funktionen. Sie sind aeusserst leistungsfaehig und erlauben es dem Benutzer mit Hilfe der erweiterten BASIC Programmiersprache jedes grafische Muster auf dem Schirm zu erzeugen.
Die einzelnen Grafikzeichen finden Sie in Anhang E.

7.1 SET (x, y)

Diese Funktion schaltet den durch die Koordinaten x und y angegebenen Grafikblock ein. Der Bildschirm ist in ein 128 (horizontal) mal 48 (vertikal) Gitter aufgeteilt. Die x – Koordinate geht von 0 bis 127, von links nach recht. Der Bereich der y – Koordinate liegt zwischen 0 und 47; von oben nach unten. Daher liegt der Punkt (0, 0) in der aeussersten, linken oberen Ecke des Schirms. Der Punkt (127, 47) liegt damit in der aeussersten, rechten unteren Ecke des Schirms. Der Punkt (127, 47) liegt damit in der aeussersten, rechten unteren Ecke des Schirms. Die Argumente x und y koennen numerische Konstanten. Variablen oder Ausdruecke sein. SET (x, y) benutzt nur den ganzzahligen Anteil der Argumente, deshalb ist es nicht noetig, dass die Argumente ganzzahlig sind.

7.2 RESET (x, y)

Diese Funktion schaltet den, durch die Koordinaten x und y angegebenen Grafikblock aus. Fuer ihre Argumente gilt das gleiche, wie fuer die SET (x, y) Funktion.

7.3 CLS

Diese Funktion loescht den Bildschirm und schaltet alle Grafikbloecke aus. Sie bringt ausserdem den Cursor in die obere, linke Ecke des Schirms. Diese Funktion erlaubt es dem Benutzer, den Schirm ausschliesslich fuer eine Ausgabe frei zu machen, ohne dass der vorige Inhalt des Schirms noch stoeren koennte.

7.4 POINT (x, y)

Diese Funktion sieht nach, ob ein, durch die Koordinaten x und y bestimmter, Grafikblock an- oder abgeschaltet ist. Ist der Block an (SET auf ihm ausgefuert), dann erzeugt POINT eine –1 (logisch "wahr"), ist der Block nicht gesetzet, so erzeugt sie eine 0 (logisch "falsch")

Beispiel:

```
A = POINT (3, 40)
```

Ist Block 3.40 gesetzt, so erhaelt A den Wert – 1. Ansonsten wird A = 0.

KAPITEL 8

8.1 INP (Portnummer)

Liest einen 8-bit Wert aus dem angegebenen Port. Das Video Genie System kann 256 Ports adressieren. Sie werden von 0 bis 255 numeriert. Normalerweise wird dieser Befehl nur benutzt, wenn die "Expansion Box" angeschlossen ist.

Beispiel:

```
10 A = INP (124)
```

Liest 8-bit Wert aus Port Nr. 124 nach A.

8.2 OUT Portnummer, Wert

Gibt einen 8-bit Wert auf dem angegebenen Port aus. Dieses Statement verlangt zwei Argumente: die Portnummer und den Wert, der an diesem Port ausgegeben werden soll. Das Video Genie System kann 256 Ports adressieren, sie sind von 0 bis 255 durchnumeriert.

Beispiel:

```
30 OUT 14,240
```

Gibt den Wert 240 an Port 14 aus. Beide Argumente muessen zwischen 0 und 255 liegen.

8.3 PEEK (adresse)

Diese Funktion holt einen 8-bit Wert aus der Speicherzelle, die mit Adresse dezimal angegeben wurde. Der Wert von PEEK ist ebenso dezimal und liegt zwischen 0 und 255.

Beispiel:

```
20 B = PEEK (3000)
```

Schreibt den Inhalt von Speicherzelle 3000 nach B.

8.4 POKE Adresse, Wert

Dieses Statement schreibt einen 8-bit Wert in die, durch Adresse dezimal angegebene Speicherzelle. Es braucht zwei Argumente: Adresse und Wert. Wert muss zwischen 0 und 255 liegen.

Beispiel:

```
10 A = 250
20 POKE 19000, A :    REM SCHREIBE A IN ZELLE 19000
30 B = PEEK (19000) : REM HOLE DEN WERT VON ZELLE 19000
40 PRINT "DAS ERGEBNISS IST : "; B
50 END

READY
>RUN

DAS ERGEBNISS IST: 250
```

8.5 MEM

Gibt die Zahl der ungeschuetzten Bytes im Speicher an.

Beispiel:

```
200 IF MEM < 180  THEN  700
```

Wenn MEM als Kommando eingesetzt werden soll, muss es zusammen mit PRINT ausgefuehrt werden. PRINT MEM gibt die Zahl der Bytes im Speicher aus, in denen kein Programm, Variablen, Zeichenketten usw. stehen.

ANHANG A

Die Schluesselworte des Video Genie Systems

ABS
AND
ASC
ATN
CDBL
CHR$
CINT
CLEAR
CLOSE
CLS
CONT
COS
DATA
DEFDBL
DEFFN
DEFINT
DEFSNG
DEFUSR
DEFSTR
DELETE
DIM
EDIT
ELSE
END
ERL
ERR
ERROR
EXP
FIX
FOR
FRE
GET
GOSUB
GOTO
IF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LET
LSET
LEN
LINE
LIST
LOAD
MEM
MID$
NAME
NEW
NEXT
NOT
ON
OUT
PEEK
POINT
POKE
POS
PRINT
PUT
RANDOM
READ
REM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
SET
SGN
SIN
SQR
STEP
STOP
STRING$
STR$
TAB
TAN
THEN
TROFF
TRON
USING
USR
VAL
VARPTR

* Keines dieser Worte darf in einem Variablennamen benutzt werden.

ANHANG B

FEHLERCODES

CODE	ABKUERZUNG	FEHLER
1	NF	NEXT without FOR
2	SN	Syntax error
3	RG	Return without GOSUB
4	OD	Out of Data
5	FC	Illegal function call
6	OV	Overflow
7	OM	Out of memory
8	UL	Undefined line
9	BS	Subscript out of range
10	DD	Redimensioned array
11	/0	Division by zero
12	ID	Illegal direct
13	TM	Type mismatch
14	OS	Out of string space
15	LS	String too long
16	ST	String formula too complex
17	CN	Cant's continue
18	NR	NO RESUME
19	RW	RESUME without error
20	UE	Unprintable error
21	MO	Missing operand
22	FD	Bad file data

Beschreibung der Fehlermeldungen

NF Next ohne FOR: Ein NEXT ohne entsprechendes FOR wurde benutzt. Dieser Fehler kann auch auftreten, wenn die NEXT Variablen in verschachtelten Schleifen vertauscht wurden.

SN Syntax Fehler: Ist normaler Weise das Ergebniss vòn falscher Interpunktion, offenenen Klammern ungueltigen Zeichen oder falsch geschriebenen Statements.

RG RETURN ohne GOSUB: Ein RETURN Statement wurde gefunden, ohne das ein entsprechendes GOSUB existiert.

OD Keine Daten mehr: Einem READ oder INPUT# Statement stehen nicht mehr genug Daten zur Verfuegung. Ein DATA Statement kann vergessen worden sein oder auf dem Band sind schon alle Daten gelesen.

FC Illegaler Funktionsaufruf: Es wurde der Versuch gemacht, eine Operation mit ungueltigen Parametern auszufuehren. Beispiel: Quadratwurzel mit negativem Argument, negative Matrixdimension, negativ oder Null Argumente fuer LOG, usw . . USR Aufruf, ohne die Startadresse zu POKEN.

OV Ueberlauf: Ein berechneter Wert oder eine Eingabe ist zu gross oder zu klein. Der Computer kann ihn nicht mehr handhaben.

OM Kein Speicherplatz mehr: Der gesamte Speicherplatz ist entweder benutzt oder reserviert worden. Die Ursache koennen zu grosse Felddimensionen oder verschachtelte Sprungbefehle wie GOTO, GOSUB und FOR-NEXT sein. Es kann aber auch schlicht das Programm zu lang sein.

UL Undefinierte Zeile: Es wurde versucht, zu einer nicht existierenden Zeile zu springen.

BS Dimension aus ihrem Bereich: Es wurde versucht, ein Feldelement anzusprechen, dessen Dimension ueber der, im DIM Statement angegebenen liegt.

DD Redimensionierung: Es wurde versucht, ein Feld zu dimensionieren, das schon dimensioniert ist (mit DIM oder implizit). Guter Programmierstil ist es, DIM Statements an den Programmanfang zu stellen.

/0 Division durch Null: Es wurde versucht, Null als Divisor zu benutzen.

ID Illegaler, direkter Einsatz: Es wurde versucht, das INPUT Statement auf der Kommandoebene auszufuehren.

TM Typ stimmt nicht ueberein: Es wurde versucht, in eine nicht-Zeichenkettenvariable eine Zeichenkette zu schreiben oder umgekehrt.

OS kein Platz mehr fuer Zeichenketten: Siehe CLEAR

LS Zeichenketten zu lang: Eine Zeichenkette darf nur 255 Zeichen lang sein.

ST Stringformel zu komplex: Ein Zeichenkettenformel ist zu komplex, um sie zu handhaben.

CN Kann CONT nicht ausfuehren: Ein CONT wurde eingegeben, obwohl kein Programm zum Weitermachen mehr vorhanden ist. Z.B. wenn das Programm mit END abgeschlossen wurde oder nach EDIT.

NR NO RESUME: Das Programmende wurde im Fehlererkennungsmodus erreicht.

RW RESUME ohne ERROR: Ein RESUME wurde gefunden, bevor ein ON ERROR GOTO ausgefuehert wurde.

UE Nicht ausgebbarer Fehler: Es wurde versucht, einen Felher mit nicht existierendem ERROR Code mit ERROR zu simulieren.

MO fehlender Operand: Es wurde versucht, eine Operation auszufuehren, bei der einer der Operanden fehlte.

FD Defekte Datei: Die Dateneingabe von einer externen Quelle (Kassettenrecorder usw.) war falsch, in der falschen Reihenfolge usw.

APPENDIX C

Control Codes: 1-31

Code	Function
8	Backspaces and erases current character
9	None
10-13	Carriage returns
14	Turns on cursor
15	Turns off cursor
16-22	None
23	Converts to 32 character mode
24	Backspace ←Cursor
25	Advance→ Cursor
26	Downward ↓ linefeed
27	Upward ↑ linefeed
28	Home, return cursor to display position (0,0)
29	Move cursor to beginning of line
30	Erases to the end of the line
31	Clear to the end of the frame

ASCII Character Codes 32-128

Code	Character
32	space
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	–
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
64	@

Code	Character
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	L
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	–
96-127	Lower case for codes 64-95
128	Space

ANHANG D

Speicherbereiche und Grenzen der Programmierung

Bereiche

ganze Zahlen	–32768 bis +32767 einschl.
einfache Genauigkeit	–1.701411E + 38 bis +1.701411E + 38 einschl.
doppelte Genauigkeit	–1.70141 1834244556E + 38 bis +1.70141 1834244556E + 38 einschl.
String Bereich	bis 255 Zeichen
Zeilennummern	0 bis 65529 einschl.
Laenge der Programmzeilen	bis zu 255 Zeichen

Speicherplatzbedarf
Eine Programmzeile braucht minimal 5 Bytes:
- Zeilennr.: 2 Bytes
- Zeilenpointer: 2 Bytes
- Carriage Return (Zeilenvorschub): 1 Byte

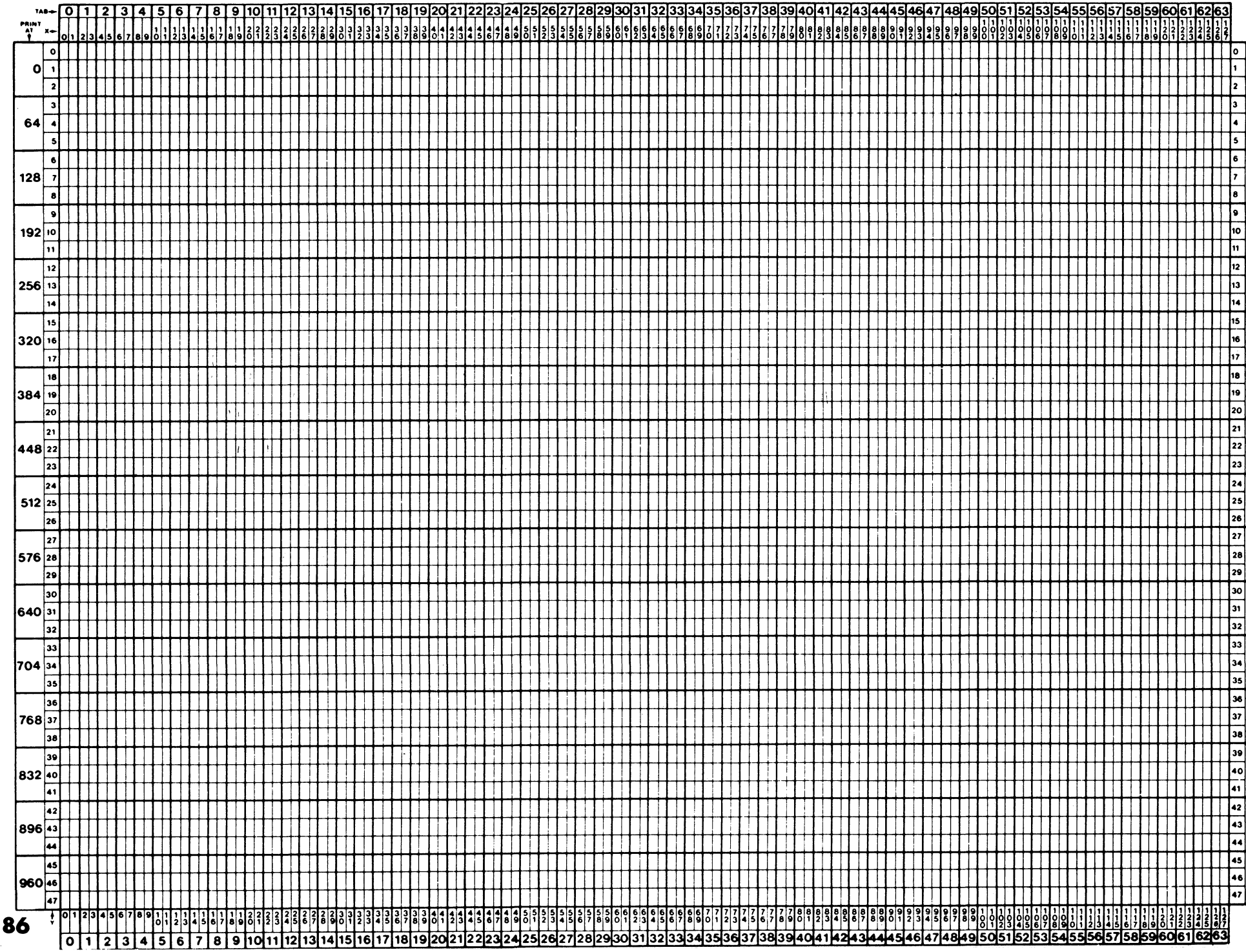
Dazu braucht jeder Befehl, Operator, Variablenname, Spezialzeichen und Konstantenziffer ein Byte.

Dynamische Speicher platzzuteilung (bei Programmausfuehrung)

ganzzahlige Variablen	5 Bytes jede
Variablen einfacher Geanuigkeit	7 Bytes jede
Variablen doppelter Genauigkeit	11 Bytes jede
String Variablen	6 Bytes minimal (3 fuer Variablenname, 3 fuer Kellerspeicher und Pointer, 1 fuer jedes Zeichen)
Feldvariablen	12 Bytes minimal (3 fuer Variablenname, 2 fuer Groesse, 1 fuer Anzahl der Dimensionen, und 2, 3, 4, oder 8 fuer jedes Element abhaengig vom Typ)
Jede aktive FOR-NEXT Schleife	16 Bytes
Jedes aktive GOSUB (noch kein Return)	6 Bytes

Jede Klammerebene 4 Bytes und 12 Bytes fuer Zwischenergebnisse

APPENDIX E VIDEO DISPLAY MAP



Four more BASIC commands should be included in the instruction set. They are (1) INKEY $, (2) POS, (3) USR and (4) VARPTR.

ADDENDUM

(1) INKEY$

Liest ein Zeichen von der Tastatur. Wurde zum Zeitpunkt der Ausfuehrung der INKEY Funktion keine Taste betaetigt, so wird ein Leerstring erzeugt (=" "). Die von der INKEY$ Funktion eingelesenen Daten erscheinen nicht auf dem Schirm.

Beispiel:

```
10  REM EINGABE EINES PASSWORDS
20  REM OHNE ES AUF DEM SCHIRM SICHTBAR
    ZU MACHEN
30  CLS
40  PRINT "GIB PASSWORD EIN"
50  A$ = INKEY$: IF AS = "O" THEN 60 ELSE 50
60  B$ = INKEY$: IF B$ = "K" THEN 70 ELSE 60
70  PRINT "WILLKOMMEN"
```

Beispiel:

```
10  A$ = INKEY$: IF A$ = " " THEN 10
```

Dieses Programm wartet, bis eine Taste betaetigt wurde.

(2) POS (dummy argument)

Es wird eine Zahl von 0 bis 63, die die momentane Position des Cursors angibt. "dummy argument" kann irgent eine Zahl sein, meist wird die "0" benutzt.

Beispiel:

```
100  A = POS (0)
```

(3) USR (argument)

Ruft ein Unterprogramm in Maschinensprache auf. Das Unterprogramm kann mit POKE erzeugt oder vom Band eingelesen werden. Wenn der Benutzer in der Programmierung in Maschinensprache noch nicht sicher ist, ist vom Einsatz dieser Funktion abzuraten.

Die Startadresse des Unterprogramms wird mit POKE in die Adressen 16526 und 16527 mit dem, am wenigsten signifikanten Byte in 16526, geschrieben.

Um ein Argument in das Unterprogramm zu uebergeben, sollte das Unterprogramm an seinem Anfang ein CALL 0A7F'H (2687 in dezimal), ausfuehren. Das Argument wird dann in das HL-Register geschrieben.

Um ins BASIC zurueck zu kommen, ohne einen Wert zurueck zu geben, wird das Unterprogramm mit RET abgeschlossen. Soll ein Wert zurueckgegeben werden, so wird er in das HL-Registerpaar geladen und am Ende des Unterprogramms ein JP 0A9A'H (= 2714 in dezimal) ausgefuehrt. Die Werte werden als 2 Byte, ganze Vorzeichenzahlen behandelt. Die USR Funktion reserviert 8 Stack Ebenen fuer das Unterprogramm.

Beispiel:

```
10  INPUT 1%: REM ARGUMENT EIGEBEN
15  REM * STARTADRESSE EINSCHREIBEN *
20  POKE 16526, 0: POKE 16527, 120
30  A = USR (I%): REM * A WIRD ZURUECKGEGEBEN*
```

Das Unterprogramm sollte in den ob

Das Unterprogramm sollte in den obersten Speicherbereich geschrieben werden. Um es davor zu schuetzen, dass es vom Basic als Programmspeicher betrachtet und damit sein Inhalt zerstoert wird, sollte der Benutzer, wenn das System nach dem Einschalten READY? ausgibt, die hoechste Speicheradresse, die das BASIC noch benutzen darf, ohne sein Unterprogramm zu ueberschreiben, angeben (in dezimal).

(4) VARPTR (variablen name)

Die Adresse, an der der Wert der Variblen im Speicher steht, wird berechnet.

```
10  K = VARPTR (A)
```

Fuer die verschiedenen Vraiablentypen bedeutet K dann folgendes:

(i) 2-Byte, ganzahlige Variablen (A%)
K = LSB
K + 1 = MSB

(ii) Variablen einfacher Genauigkeit (A)
K = LSB
K + 1 = naechstes MSB
K + 2 = MSB
K + 3 = Exponent

(iii) Variablen doppelter Genauigkeit (A#)
K = LSB
K + 1 = naechstes MSB
.
.
.
K + 6 = MSB
K + 7 = Exponent

(iv) Zeichenketten Variablen (string) (A$)
K = Laenge des Strings
K + 1 = LSB der Anfangsadresse des Strings
K + 2 = MSB der Anfangsadresse des Strings

LSB bezeichnet das, am wenigsten signifikante Byte.
MSB bezeichnet das, am hoechsten signifikante Byte.

Beispiel:

03F1 sind zwei Byte in Hexadezimal. 03 ist das MSB und F1 ist das LSB.

AKTIVE KOMMANDOS

PROGRAMMBEFEHLE

EDITIERUNGSBEFEHLE

NEWLINE	–	speichere alle Aenderungen
Leertaste	–	Cursor eine Position nach rechts
BACKSPACE	–	Cursor eine Position nach links
SHIFT-ESC	–	beendet Insert (I) Kommando
H	–	abschneiden und einsetzen
I	–	einsetzen (insert)
X	–	einsetzen an Zeilenende
L	–	Zeile listen
A	–	loescht alle Aenderungen
E	–	speichert die Aenderungen
Q	–	zurueck zur aktiven Kommandoebene ohne Aenderungen zu speichern
D	–	loesche Zeichen (delete)
C	–	aendere Zeichen (change)
S	–	suche Zeichen (search)
K	–	loesche angegebene Zeichen

STRING FUNKTIONEN

ARITHMETISCHE FUNKTIONEN

GRAFIK FUNKTIONEN

SPEZIAL FUNKTIONEN

*explained in ADDENDUM

EACA International Ltd.
11/F Eaca Industrial Bldg.,
13, Chong Yip St., Kwun Tong
Kowloon, Hong Kong.
Tel.: 3-896323 (8 lines)
3-424151 (6 lines)
Telex: 84035 ECHK HX