BASIC

Programmier-Handbuch

Speziell für die
BASIC-Versionen
der modernen
Personal-Computer

Einführung
und
Nachschlagewerk

C. Lorenz

ISBN 3–921682–48–7

6. völlig neu überarbeitete Auflage 1984

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

BASIC

Programmier-Handbuch

C. Lorenz

Einführung und Nachschlagewerk

Speziell für die BASIC-Versionen der modernen Personal-Computer

Vorwort

Die grundlegende Verbindung zwischen dem Programmierer und dem Computer ist die Programmiersprache. Eine der erfachsten und sehr leicht erlernbaren Computersprachen ist BASIC. Dieses Buch will Ihnen die Grundelemente von BASIC mitteilen und Sie in die Lage versetzen, selbst Programme schreiben zu können.

Die Beispielprogramme wurden auf SINCLAIR, APPLE, ATARI und COMMODORE Rechnern getestet. Auf Unterschiede zwischen den einzelnen BASIC Dialekten wird hingewiesen. Das Buch eignet sich darüber hinaus jedoch für alle heute am Markt befindlichen Personalcomputer.

Neben den ausführlichen Befehlsbeschreibungen findet der Leser einen BASIC-Einsteiger Kurs für Anfänger, der besonders den Erstanwender ansprechen soll.

Als Nachschlagewerk wird es dem erfahrenen Programmierer immer zur Seite stehen.

Wir wünschen Ihnen beim Programmieren viel Spaß und hoffen, daß auch Sie später einen praktischen Nutzen daraus ziehen können.

Holzkirchen, Frühjahr 1984 C. Lorenz

Inhaltsverzeichnis

Einführung

Einführung

Die Programmiersprache BASIC gehört zu den einfachsten höheren Programmiersprachen und wurde zu Anfang der 60er Jahre im amerikanischen Darthmouth College entwickelt.

Sie läßt sich sehr leicht erlernen und bietet doch ein sehr leistungsfähiges Werkzeug für den Programmierer. Der Erfolg dieser Programmiersprache rührt auch daher, daß BASIC in 90% aller Time-Sharing-Systeme verwendet wurde, und daß die Minicomputer ein erweitertes BASIC enthielten. Bei den modernen Microcomputern ist es heute selbstverständlich, daß ein BASIC-Interpreter zur Verfügung steht. Der große Einsatzbereich hat natürlich dazu geführt, daß sich unzählige BASIC-Versionen bis heute herausgebildet haben.

Dieses Buch will Ihnen die Version näher bringen, die heute auf den modernen Microcomputern implementiert ist. Wenn Sie es gelernt haben, diese eine Version anzuwenden, ist es sehr leicht, auch mit einer anderen Version zu arbeiten. Sie werden bald sehen, wie einfach es ist, sich selbst seine eigenen Programme zu erstellen (sei es für die Buchhaltung, Textverarbeitung, math. wissenschaftliche Probleme oder auch nur zur Unterhaltung).

Allgemeines

Die Popularität von BASIC läßt sich ganz kurz auf die wichtigsten Punkte zurückführen:

1. Einfache Ausdrücke, die sich auf eine begrenzte Zahl von Befehlen beschränken.
2. Die Möglichkeit, mit Strings (Zeichenketten) und Matrizen zu arbeiten.

3. Eingabe und Editiervorgänge sind einfach zu handhaben. Leichte Fehlersuche und Anwendungsmöglichkeit.
4. Leichte Übersetzung durch einen Interpreter, der zwar relativ langsam ist, aber eine interaktive Arbeit zwischen Programmierer und Computer ermöglicht (z. B. beim Fehlersuchen und Fehlerbeheben).
5. Relativ einfache Programmerstellung.

Erklärungen

Strings
Die Behandlung von Zeichen im Computer (anstelle von Zahlen). Sie befinden sich meist bereits in einer angeordneten Reihenfolge.

Implementieren
Auf ein System zuschneiden. In ein Computersystem anpassen.

Interpreter
Ein Interpreter ist jede Programmiersprache, die die Befehle des Quellenprogrammes direkt ausführt. Jeder Befehl wird für sich interpretiert und ausgeführt. Es wird nicht erst das gesamte Programm in Maschinensprache übersetzt und dann die Ausführung begonnen (dies ist der Unterschied zum Compiler).

Sie werden sicher auch oft vergeblich nach einfachen Erklärungen von schwierigen Zusammenhängen gesucht haben, wobei man Ihnen jedoch meist mit komplizierten Antworten auf einfache Fragen geantwortet hat.

Wir wollen jedoch im nachfolgenden Buch versuchen, Ihnen den Start in die BASIC-Programmierung so einfach wie möglich zu machen. Sie sollten nach Durcharbeitung des Buches in der Lage sein, selbstständig Programme zu entwickeln oder vorhandene Programme auf Ihren Bedarf abzuändern.

Was ist ein Programm ?

Ein Programm ist ein Paket von Befehlen, oder auch ein Rezept, Welches verwendet wird, dem Computer eine Information zu geben, damit er uns ein gewünschtes (gesuchtes) Ergebnis liefert.

Man gibt dem Computer eine Reihe von Daten (Zutaten beim Kuchenbacken) ein, die nach bestimmten Befehlen verarbeitet werden müssen (z. B. beim Backen: 1/4 Stunde rühren, etc.). Alles muß in einer bestimmten Reihenfolge eingegeben und verarbeitet werden. Am Ende erhalten wir dann ein Ergebnis (beim Backen ist es eben der Kuchen).

Wenn Sie Fehler machen, geht es Ihnen genau so wie beim Kuchenbacken. Es wird dann halt kein Kuchen, sondern eine Pizza.

Ein Programm (dies gilt für alle Programmiersprachen) muß folgende Bedingungen erfüllen, um einwandfrei laufen zu können.

1. Der Computer muß die Sprache des Programmierers verstehen.

2. Alle Anweisungen müssen so gegeben werden, daß sie der Computer auch versteht. Der Computer tut immer das, was Sie ihm eingeben. Sorgen Sie aber dafür, daß Sie ihm das eingeben, was Sie ihm wirklich mitteilen wollen.

Wir wollen nun anfangen zu programmieren

Das Programmschreiben in BASIC ist sehr einfach. Deshalb wollen wir jetzt gleich mit unseren ersten Versuchen beginnen. Wichtig ist es, daß Sie ein Computersystem zur Verfügung haben. Das Lernen auf dem "Trockenen" kann wie das Wort im doppelten Sinne schon sagt, sehr trocken sein. Aber es gibt heute bereits so preiswerte BASIC-Computer, so daß es nicht so schwer sein dürfte, sich einen zu beschaffen oder auszuleihen.

Schreiben eines BASIC-Programmes

Um ein BASIC-Programm zu schreiben, müssen Sie zuerst einmal eine Zeilenzahl eingeben. Der Zeilenzahl muß ein Statement (Befehl) folgen. Die Zeilenzahlen werden vom erfahrenen Programmierer so gewählt, daß man später noch etwas einfügen kann (z. B. fängt man mit Zeile 10 an und numeriert die Zeilen dann im Abstand von 10 nach oben).

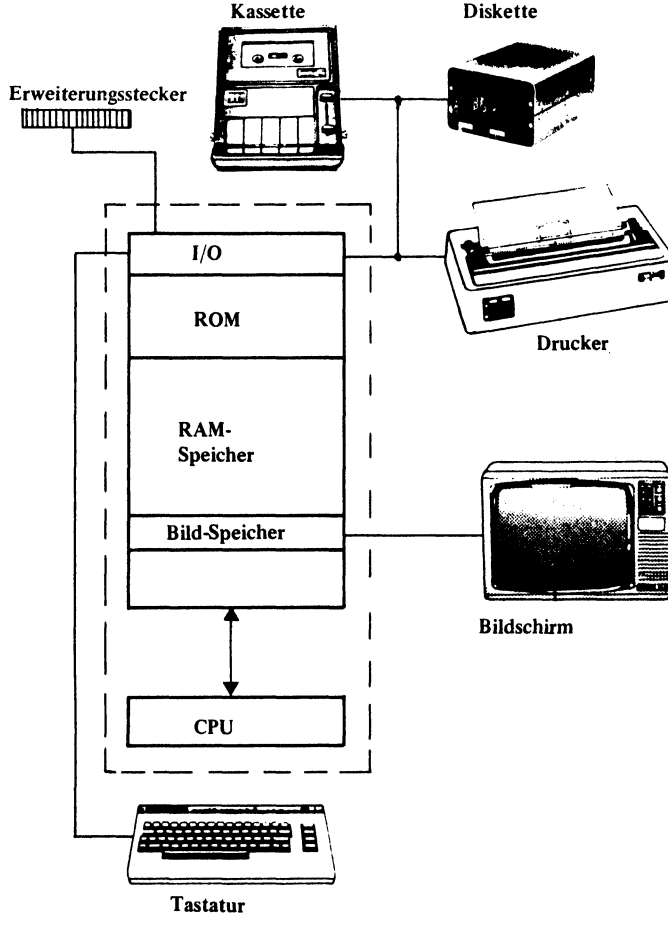
Der Computer führt das Programm dann der Reihenfolge nach aus. Er beginnt mit der niedrigsten Zeilenzahl und arbeitet sich dann aufwärts

weiter. Geben Sie z. B. zwei Zeilen mit der gleichen Zeilenzahl ein, wird die erste Zeilenzahl und der in der gleichen Zeile stehende Text überschrieben.

Der prinzipielle Aufbau eines Rechners

Der prinzipielle Aufbau eines Rechners

Die folgende Abbildung zeigt den prinzipiellen Aufbau eines Rechners.



Innerhalb des gestrichelten Teils befindet sich der eigentliche Rechner. Er besteht aus einem Speicher, der Zentraleinheit und dem Ein-/Ausgabeteil.

Der Speicher ist aus zwei verschiedenartigen Elementen zusammengesetzt. Dies ist der RAM- und ROM-Speicher.

RAM bedeutet Random-Access-Memory, d. h. Speicher mit wahlfreien Zugriff. Jede einzelne Zelle dieses Bereiches kann beschrieben und gelesen werden.

Der ROM-Speicher ist ein "Nur Lese Speicher" (Read Only Memory). Aus diesem Bereich können nur Daten gelesen, aber nicht eingeschrieben werden. In diesem Speicher ist das Betriebssystem des Rechners gespeichert, das nach dem Einschalten des Rechners aktiv wird und auch der BASIC-Interpreter gespeichert.

Die CPU (Central Prozessing Unit) überwacht diesen Speicher und führt die dort gespeicherten Befehle aus. Diese Befehle sind nicht Worte der BASIC-Sprache, sondern Bitmusterfolgen, die durch diese Worte aufgerufen werden.

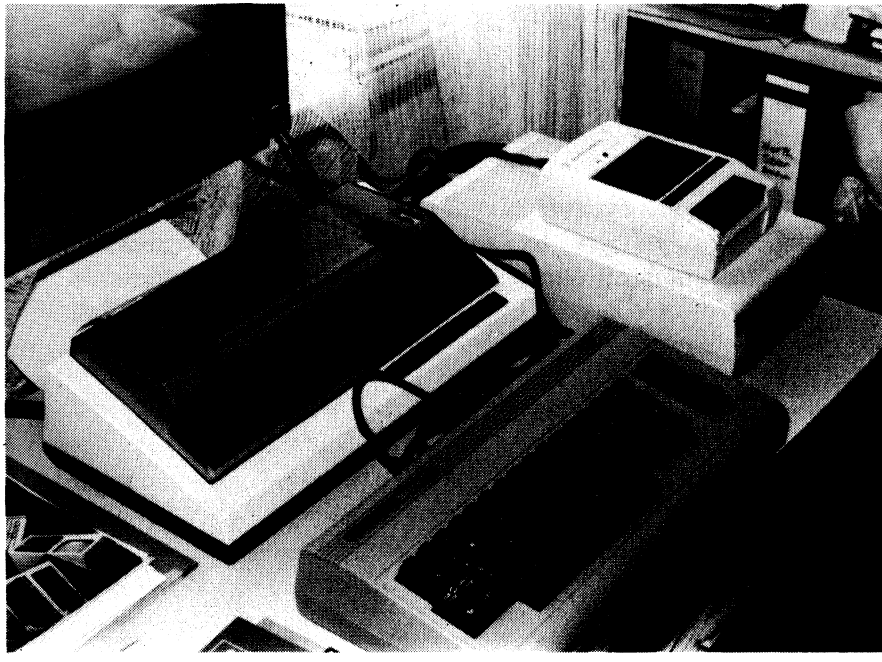
Außerhalb der gestrichelten Linie befinden sich die externen (periphere) Geräte. Für die Eingabe wird eine Tastatur verwendet. Von ihr werden Bitmuster über den Ein-/Ausgabeteil (I/O) in den Speicher gebracht. Nach betätigen der RETURN-Taste wird die Eingabe entschlüsselt und das Betriebssystem oder der BASIC-Interpreter entsprechend verzweigt.

Die Ausgabe erfolgt in den meisten Fällen auf einem Bildschirm. Dieser benötigt einen kleinen Teil des RAM-Speichers, um die Zeichen, die auf dem Bildschirm angezeigt werden, zu speichern. Über die Ein-/Ausgabe-Schnittstelle sind auch der Massenspeicher, Kassettenrekorder oder das Diskettenlaufwerk angeschlossen. Mit diesem werden auf Magnetbändern oder Disketten Daten aus dem Speicher aufgezeichnet.

Für die Druckausgabe kann über die gleiche Schnittstelle ein Drucker angeschlossen werden. Dieser Prinzipielle Aufbau gilt für alle Rechner. Für den normalen Benutzer spielt es keine Rolle, welche Zentraleinheit

(CPU) verwendet wird. Die bekanntesten sind:

6502	(APPLE, ATARI, COMMODORE)
Z80	(ZX81, SPECTRUM, TRS-80, GENIE, SHARP)
8080	(in älteren wie ATARI, Dai, usw.)
6909	(Dragon Color Computer)
8085	(TRS-80 Model 100)
8088	(IBM-PC-Junior, IBM PC)



Im Bild sehen Sie den Commodore-64

NOTIZEN

Systembefehle

Dies sind Befehle, die das Betriebssystem (Monitor) veranlassen, ein BASIC-Programm zu starten, auf einem externen Speicher abzulegen oder einen Programmausdruck auf den Bildschirm auszugeben.

RUN
RUN N

RUN
RUN N

RUN startet ein Programm. Wird RUN ohne Zahl eingegeben, so wird bei der ersten Zeilennummer begonnen. Wird zum Beispiel RUN 1000 eingegeben, so beginnt das Programm bei Zeilennummer 1000. RUN löscht die Werte aller Variablen und Felder.

STOP

STOP

STOP hält ein Programm an. Die Zeilennummer der STOP-Anweisung wird ausgegeben. Werden keine Veränderungen im Programm vorgenommen, so kann es mit CONT fortgesetzt werden. Durch die GOTO-Anweisung kann ein Programm an einer anderen Zeilennummer fortgesetzt werden, ohne daß die augenblicklichen Werte der Variablen geändert werden.

END

END

Die Anweisung END beendet ein Programm.

NOTIZEN

LIST N,M

LIST
LIST N,M

LIST gibt ein Programm auf den Bildschirm aus. Wird hinter LIST nur eine Zeilennummer angegeben, so wird diese Zeile ausgegeben. Werden zwei Zeilennummern angegeben, so wird von Zeile N bis Zeile M ausgedruckt. M muß größer N sein.

Für die Ausgabe auf einen Drucker müssen unter Umständen weitere Angaben gemacht werden. Bei einigen Rechnern (TRS-80, ZX81, SPECTRUM) wird durch LLIST N,M das Programm auf den Drucker ausgegeben.

Beim APPLE wird nur LIST eingegeben. Vorher muß aber durch PR# ⟨NR⟩ der Drucker eingeschaltet werden. ⟨NR⟩ ist die Nummer des Steckplatzes, in welchen die Interfacekarte für den Drucker steckt.

Beim ATARI wird die Ausgabe auf den Drucker durch LIST "P:" eingeleitet.

Beim COMMODORE-64: LIST N–M

NEW

NEW

NEW löscht das Programm im Speicher.

NOTIZEN

SAVE

SAVE

SAVE wird zur Speicherung eines Programmes auf Kassette oder Diskette verwendet. Dieser Befehl ist bei allen Rechnern verschieden.

Einige Beispiele:

ZX81, SPECTRUM
Speichern auf Kassette mit SAVE "NAME"

APPLE II
Speichern auf Kassette mit SAVE
Speichern auf Diskette mit SAVE NAME

ATARI
Speichern auf Kassette mit SAVE"C:NAME"
oder CSAVE"NAME"
Speichern auf Diskette mit SAVE"D:NAME"

COMMODORE-64
Speichern auf Kassette mit SAVE "NAME"
Speichern auf Diskette, Device # 8 mit SAVE "NAME",8

NOTIZEN

LOAD

LOAD

LOAD wird zum Lesen eines Programmes von Kassette oder Diskette verwendet. Dieser Befehl ist bei allen Rechnern verschieden.

Einige Beispiele:

ZX81 und SPECTRUM
Laden von Kassette mit LOAD"NAME"
Ist der Name nicht bekannt, so wird durch LOAD"" (kein Zwischenraum zwischen " ") das nächste Programm geladen, das auf der Kassette gefunden wird.

APPLE II
Laden von Kassette: LOAD
Laden von Diskette: LOAD NAME

ATARI
Laden von Kassette: LOAD "C:NAME"
oder CLOAD"NAME"
Laden von Diskette: LOAD "D:NAME"

COMMODORE 64
Laden von Kassette: LOAD "NAME"
Laden von Diskette, Device # 8: LOAD "NAME",8

FRE(X)

FRE(X)

PRINT FRE(X) gibt den für ein BASIC-Programm noch freien Speicherplatz aus. X ist eine Variable, die keine Bedeutung hat (Dummy).

NOTIZEN

Programm-Anweisungen

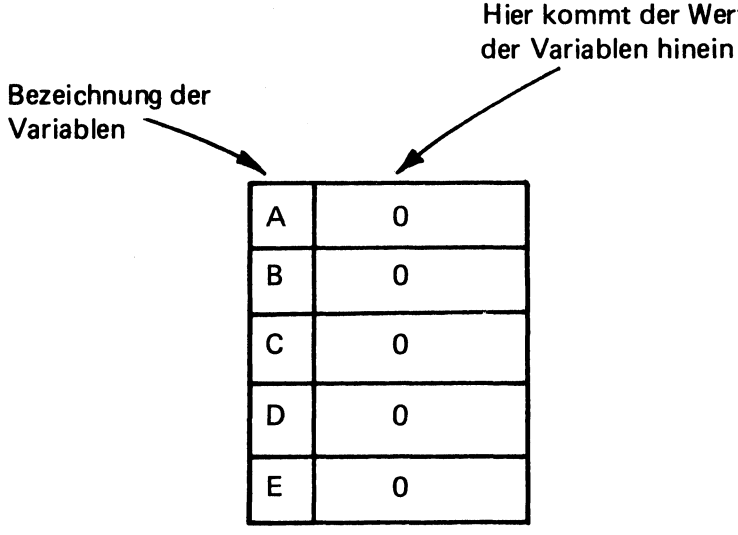
Diese Befehle dienen zur Festlegung des Programmablaufs. Damit werden Variablen Werte zugewiesen, endliche Schleifen durchlaufen oder Programme verzweigt.

LET

LET

Zuweisungs- oder Anweisungsbefehl

Variablen können wir uns als kleine Kästchen im Computer vorstellen, denen durch LET Werte zugewiesen werden.



Bei vielen BASIC-Versionen kann man das Wort LET aus Rationalisierungsgründen weglassen.
10 LET X = 1 wird dann einfach zu 10 X = 1.

Nur beim ZX81 und beim SPECTRUM muß LET eingegeben werden. Beide Rechner überwachen auch die Zuweisung von Werten an Variablen. Ist in der Zeile

```
20 LET X = A + 1
```

der Variablen A noch kein Wert zugewiesen worden, so erscheint eine Fehlermeldung.

Wollen wir jetzt eine neue Variable E einführen, und dieser den Wert 8 zuweisen, so können wir dies dem Computer mit der Zuordnungsanweisung:

```
100 LET E=8
```

mitteilen. Die Variablen können wir mit den Buchstaben des Alphabetes festlegen. Bei den meisten BASIC-Versionen kann man Variablennamen auch aus mehreren Buchstaben zusammensetzen. Die Werte, die der Variablen zugeordnet werden können, sind Zahlen beliebiger Größe, mathematische Ausdrücke oder auch Strings (Zeichenketten).

```
110 LET F=-259
120 LET G=5*9
130 LET H=F*E
140 LET I=J*K
150 PRINT G
160 PRINT H
170 PRINT I
```

Es können auch Variable einer anderen Variablen zugewiesen werden. Hier ist es jedoch wichtig, daß die als Wert verwendete Variable vorher einen Wert zugewiesen bekam.

Wenn nicht, wird ihr Wert einfach mit Null angenommen. Sie können dies leicht nachprüfen, wenn Sie jetzt die obigen Programmzeilen eingeben und mit RUN 100 starten, können wir uns die Ergebnisse ansehen. Das geschieht mit den PRINT-Befehlen. Sie geben uns die Werte für G, H und I auf dem Bildschirm aus.

Wie schon erwähnt, können wir einer Variablen auch eine Zeichenkette zuordnen.

```
180 LET H$="HOFACKER"
190 LET G$="VERLAG"
200 PRINT H$,G$
```

```
210 PRINT G
220 PRINT H
```

Geben wir dies nun ein und starten ab Zeile 180 mit RUN 180 (oder GOTO 180) so sehen wir, daß der Stringvariablen H$ der Wert "HOFACKER" und der Stringvariablen G$ der Wert "VERLAG" als Zeichenkette zugewiesen wurde. Wir drucken dies aus, um das Ergebnis zu sehen.

Wir drucken auch noch einmal die Vairablen G und H aus. Wir sehen, sie sind nicht durch G$ oder H$ verändert worden.

Eine Veränderung der Werte kann bei Variablen und Stringvariablen nur durch eine neue Anweisung erfolgen.

```
230 LET G$="SOFTWARE"
240 LET G=6*8
250 PRINT H$,G$
260 PRINT G
```

Wir geben RUN 100 und sehen, wie sich die Sache verändert. Probieren Sie auch einmal mit einem Semicolon zwischen den Strings H$ und G$ (H$; G$) und sehen Sie sich das Ergebnis an.

```
7 LET X$=" HANS"
10 PRINT "BASIC","HANDBUCH"
20 PRINT 19*4/3
30 PRINT A;B;C;D
40 PRINT "SERVUS";X$
50 PRINT "**********"
100 LET E=8
110 LET F=-259
120 LET G=5*9
130 LET H=F*E
140 LET I=J*K
150 PRINT G
160 PRINT H
```

```
170 PRINT I
180 LET H$="HOFACKER"
190 LET G$="VERLAG"
200 PRINT H$,G$
210 PRINT G
220 PRINT H
230 LET G$="SOFTWARE"
240 LET G=6*8
250 PRINT H$,G$
260 PRINT G
```

PRINT

PRINT

```
10 PRINT "BASIC","HANDBUCH"
20 PRINT 19*4/3
30 PRINT A;B;C;D
40 PRINT "SERVUS";X$
50 PRINT "**********"
```

Der PRINT-Befehl dient dazu, ein Ergebnis, die Werte von Daten, Variablen oder Strings (Zeichenketten) auf eine Ausgabeeinheit zu bringen. Die Ausgabeeinheit kann ein Drucker, ein Bildschirm, eine Siebensegmentanzeige etc. sein. Wie Sie aus dem Beispielprogramm ersehen, können auch mehrere Variablen oder Zeichen nach einem PRINT-Befehl folgen.

Für die Ausgabe von Daten stehen einfache Formatierungsbefehle zur Verfügung. Ein Komma (,) fügt zwischen die Ausgabe von zwei Variablen oder Strings 10 Leerzeichen ein.

Beispiel:

```
10 A=1:B=2
20 PRINT A,B

RUN
1          2
```

Wird in Zeile 20 das Komma durch einen Strichpunkt ersetzt, so werden die beiden Variablen ohne Zwischenraum nebeneinander ausgegeben.

```
10 A=1:B=2
20 PRINT A;B

RUN
12
```

Soll Text ausgegeben werden, so muß dieser in Anführungszeichen

gesetzt werden. Soll zum Beispiel zwischen den beiden Zahlen ein Zwischenraum eingefügt werden so wird Zeile 20 wie folgt geändert.

```
20 PRINT A;" ";B
```

Wird eine PRINT-Anweisung mit einem Strichpunkt abgeschlossen, so wird mit der nächsten PRINT-Anweisung keine neue Zeile angefangen, sondern die Ausgabe in der gleichen Zeile fortgeführt.

```
10 A=1:B=2
20 PRINT A;" ";
25 PRINT B

RUN
1 2
```

TAB

TAB

Für APPLE: **HTAB**

Die TAB(X)-Funktion sorgt für eine genaue und definierte Positionierrung von Zeichen oder Leerzeichen. Sie muß immer in Verbindung mit einem PRINT-Befehl verwendet werden.

Die TAB-Funktion repräsentiert immer einen absoluten Wert, der den Abstand vom linken Bildschirmrand angibt.

Der Wert in Klammern sollte immer eine ganze Zahl sein. Wenn nicht, wird von den meisten BASIC-Versionen automatisch auf- oder abgerundet.

Beispiel:

```
900  REM  TAB DEMO
905  REM  FOR APPLE TAB=HTAB
907  REM  NO SUCH WORD ON THE ATARI
910 X = 3:Y = 4
920 A = 1:B = 2:C = 3:D = 4
930  PRINT A;: HTAB X: PRINT B;: HTAB Y
940  PRINT C;: HTAB X * Y: PRINT D
```

Ob ein Semikolon oder ein Komma zwischen den einzelnen Werten gesetzt wird, hängt von Ihrer BASIC-Version ab. Bitte sehen Sie in Ihrem Rechner-Manual nach und machen sich entsprechende Notizen.

```
10  FOR X = 1 TO 10
20  HTAB X: PRINT "HOFACKER VERLAG"
39  NEXT X


]RUN
HOFACKER VERLAG
 HOFACKER VERLAG
  HOFACKER VERLAG
   HOFACKER VERLAG
    HOFACKER VERLAG
     HOFACKER VERLAG
      HOFACKER VERLAG
       HOFACKER VERLAG
        HOFACKER VERLAG
         HOFACKER VERLAG
```

PRINT AT

PRINT AT

Der PRINT AT-Befehl ist in einigen BASIC-Versionen wie z. B. beim TRS-80 Model I Level II und Modell II und beim Video Genie implementiert. Ebenso beim ZX81 und SPECTRUM.

Er ermöglicht es, daß ein PRINT-Befehl ab einer bestimmten Stelle auf dem Bildschirm ausgeführt wird. Die AT-Funktion kann eine Zahl, eine Variable oder ein mathematischer Ausdruck sein. Zwischen dem String und der Funktion muß ein Komma eingefügt werden.

```
5 REM NOT ON APPLE, ATARI, COMMODORE
10 REM PRINT AT TEST
20 PRINT AT(22),"ANFANG?"
30 PRINT AT(0),"ENDE?"
40 GOTO 40
90 END
```

Beim ZX81 und beim Spectrum folgen nach AT zwei Angaben Z und S.

```
PRINT AT Z,S;
```

Dabei ist Z die Zeile und S die Spalte.

```
PRINT AT 10,15; "A"
```

schreibt ein A in die 10te Zeile und die 15te Spalte. Nach Z, S muß ein Strichpunkt folgen.

```
100 LET A$="▄▀▄ "
110 LET V=0
120 FOR H=0 TO 26
130 PRINT AT V,H;A$
140 NEXT H
150 FOR H=26 TO 0 STEP -1
160 PRINT AT V,H;A$
170 NEXT H
180 GOTO 120
```

Leerzeichen

Beispiel für ZX81

Für ATARI:

```
100 POSITION 8,20:PRINT"A"
```

INPUT

INPUT

Jetzt wollen wir einmal einen Befehl kennenlernen, der dem Computerprogramm direkt einen Wert zuführt, d. h., wenn der Computer im Laufe seines Programmlaufes feststellt, daß er einen Wert oder eine Information vom Programmierer benötigt, muß er sich ja bemerkbar machen können. Er tut dies, indem er mit dem Programm anhält und einen Wert erfragt.

Für diesen Zweck verwenden wir den INPUT-Befehl. Dieser Befehl bringt den Wert für eine Variable auf Anforderung des Computers in die besagten Kästchen. Es gilt hier vieles bzüglich Strings etc. wie bei der LET-Anweisung.

Beispiel:

```
300 REM INPUT STATEMENT
310 LET PI=3.14159
320 INPUT Y
330 LET R=SQR(Y)/PI
340 PRINT "RADIUS=";R
350 INPUT A,B
355 LET F=A*B
360 PRINT "FLAECHE=",F
370 INPUT X,Y,Z,O,P
380 LET L=X*Y*Z*O*P
390 PRINT "ERGEBNIS=";L
395 END
```

Merke:

Sollten mehrere Eingaben in einer Zeile erfolgen, müssen die einzelnen Variablen durch Komma getrennt werden. Siehe Zeile 370. Wie man hier mit Zahlen umgehen kann, so kann man es auch mit Zeichenketten tun.

```
100  REM  INPUT MIT ZEICHENKETTEN
110  INPUT "WIE HEISST DU ? ";A$
120  PRINT : PRINT "DANKE": PRINT
130  PRINT " JETZT WEISS ICH, DASS DU ";A$;" HEISST"
140  PRINT " DIES BEHALTE ICH SOLANGE, BIS"
150  PRINT " DU DEN WERT A$ AENDERST"
160  PRINT " DURCH PRINT A$ WIRD DEIN NAME"
170  PRINT " IMMER WIEDER GESCHRIEBEN"
180  PRINT : PRINT A$: PRINT
190  PRINT " WILLST DU, DASS ICH DEINEN"
200  PRINT " NAMEN IN DIE GLEICHE ZEILE MIT"
210  PRINT " ABSTAND DRUCKE, SO GILT FOLGENDES:"
220  PRINT " DEIN NAME IST",A$
230  PRINT " DEIN NAME IST ";A$
```

Bei INPUT-Funktion können auch Strings und Werte gemischt werden.

```
300  REM  INPUT MIT ZEICHENKETTEN
310  REM  UND ZAHLEN
320  PRINT : PRINT " WIE ALT BIST DU ? ";A$
330  INPUT " ";A
340  PRINT " JETZT WEISS ICH, DASS DU ";A;" JAHRE ALT":
     PRINT " BIST"
350  PRINT A;" JAHRE IST EIN SCHOENES ALTER"
```

Bitte beachten Sie bei diesem Beispiel den Unterschied zwischen dem Komma und Semikolon (unterschiedliche Auswirkung).

INKEY$

INKEY$

Die INKEY$-Funktion finden wir bei den Radio Shack Modellen (TRS-80 Model I und II) sowie beim ZX81 und Spectrum.

INKEY$ wird dazu verwendet ein Zeichen von der Tastatur her einzugeben, ohne die RETURN-Taste drücken zu müssen, wie wir das vom INPUT-Befehl her bereits kennen.

Viele Computer kennen diesen Befehl leider nicht. Es lässt sich aber dann eine kleine Routine schreiben, die eine ähnliche Funktion bewirkt. Man muß wissen wo das Programm liegt, welches die Tastatur abfragt.

Wenn Sie eine bestimmte Taste prüfen wollen, müssen Sie die Variable P$ durch eine IF THEN Funktion vorher abfragen, bevor Sie ein Programm weiterfahren und solange in die INKEY$ Routine springen, bis der gewünschte Buchstabe auf der Tastatur gedrückt wird.

Beispiel für den ZX-SPECTRUM:

Bewegen des Buchstaben A mit den Cursortasten über den Bildschirm.

```
300  LET v=1: LET u=0: PRINT AT v,u;"A"
310  IF  INKEY$ ="5" THEN  LET u=u-1
315  IF u<0 THEN  LET u=0
320  IF  INKEY$ ="6" THEN  LET v=v+1
325  IF v>17 THEN  LET v=17
330  IF  INKEY$ ="7" THEN  LET v=v-1
335  IF v<1 THEN  LET v=1
340  IF  INKEY$ ="8" THEN  LET u=u+1
345  IF u>26 THEN  LET u=26
```

```
350  GOSUB 208
355  IF  INKEY$ ="" THEN  GOTO 355
360  GOTO 310
```

GET

GET

Der GET-Befehl wird beim PET/CBM und APPLE meist zu Abfragen ohne Carriage Return (ähnlich INKEY$) verwendet.

```
10  PRINT "EINGABE ";: GET A$
20  IF A$ = "A" THEN  PRINT "AHA AHA"
30  IF A$ <  > "A" THEN  GOTO 10
40  END
```

NOTIZEN

READ DATA

READ DATA

Wir haben jetzt gesehen, wie man vom Programm her nach einer Reihe von Zahlen oder Zeichenketten fragen kann und eine Eingabe von außen erwartet.

Jetzt wollen wir uns mit einem Befehl beschäftigen, der es ermöglicht, große Mengen von Daten mit einer Anweisung einzugeben. Man kann für diesen Zweck auch den INPUT-Befehl genau so gut benutzen, aber dies würde sehr viel Rechnerzeit in Anspruch nehmen. Aus diesem Grunde verwendet man in diesen Fällen die READ DATA-Anweisung.

Diese Anweisung verhält sich bei Zahlen und Strings ähnlich wie die LET- und INPUT-Anweisung. Die READ DATA-Anweisung wird vom erfahrenen Programmierer so gehandhabt, daß die DATA-Anweisung meist an den Schluß des Programmes gelegt wird. Im Programm kann dann von den entsprechenden Stellen aus mit dem READ-Befehl aus den DATA-Statements gelesen werden.

Beispiel:

```
700 REM READ DATA BEFEHL
710 REM BEISPIEL
720 READ N
730 PRINT N
740 DATA 10
750 READ A$
760 PRINT A$
770 DATA "HOFACKER VERLAG"
780 READ N,M,O,P
790 PRINT N,M,O,P
800 DATA 1,2,3,4
```

Die Variablen und String-Variablen müssen immer durch Kommas getrennt sein. Dem letzten Wert im DATA-Statement darf kein Komma

folgen.

Ein schönes Beispiel für die Mischung von Strings und Variablen zeigt das folgende kleine Programm:

```
810 READ B$,C$,A,B,C
820 PRINT A;" ";B$;
825 PRINT B;" ";B$;C;C$;A+B+C
830 DATA PLUS , ERGIBT ,9,2,11
835 END
```

Die erste Variable oder Stringvariable in der READ-Anweisung gehört immer zu dem ersten Wert in der unmittelbar folgenden DATA-Anweisung.

Beispiel:

```
840 READ R,S
850 READ T
860 READ U,V
870 LET X=R+S-T+U+V
880 PRINT X
890 PRINT R,S,T,U,V
900 DATA 2,3,4
910 DATA 5
920 DATA 6
```

Sie müssen also beim Programmieren darauf achten, daß Sie immer die richtige Zuordnungsreihenfolge einhalten. Auch die Anzahl der Daten in den DATA-Statements muß genau mit der Anzahl der READ-Anweisungen übereinstimmen.

Der Unterschied zum INPUT-Befehl besteht jedoch darin, daß die Daten in den DATA-Statements als Bestandteil des Programmes zur Verfügung stehen müssen. Die Daten verändern sich nicht.

In DATA-Statements können nur Werte und Strings gegeben werden, keine mathematischen Ausdrücke oder Formeln !

Leerzeichen müssen wie andere Zeichen auch, zwischen Doppelapostrophe gesetzt werden.

RESTORE

RESTORE

Der Zusammenhang zwischen den Variablen im READ-Befehl und den Elementen in den DATA-Statements wird durch einen speziellen, internen Pointer (Zeiger) gesteuert.

Dieser Zeiger zeigt immer auf das DATA-Element, welches als nächstes gelesen werden muß.

Der Befehl RESTORE bringt diesen Pointer an den Anfang, so daß alle Werte in den DATA-Statements noch einmal gelesen werden.

```
1000 READ A,B,C,D
1010 RESTORE
1020 READ E,F,G
1030 RESTORE
1040 READ H,I
1050 DATA 1,2,3,4,5
1060 PRINT A,B,C,D
1070 PRINT E,F,G
1080 PRINT H,I
```

In unserem Beispiel werden zuerst den Variablen A, B, C, D die Werte 1, 2, 3, 4, 5 zugeordnet. Dann werden den Variablen E, F und G die Werte 1, 2, 3 zugeordnet. In Zeile 1040 werden dann den Variablen H, I die Werte 1 und 2 zugeordnet.

NOTIZEN

REM

REM

Wir haben diese Anweisung schon benutzt, ohne sie zu erklären. Die REM-Anweisung dient zur Dokumentation (Beschreibung) Ihres Programmes, damit Sie später noch wissen, wie Sie das Programm entwickelt haben und welche Bedeutung die nachfolgenden Programmzeilen haben.

```
300 REM INPUT STATEMENT
310 LET PI=3.14159
320 INPUT Y
330 LET R=SQR(Y)/PI
340 PRINT "RADIUS=";R
350 INPUT A,B
355 LET F=A*B
360 PRINT "FLAECHE=",F
370 INPUT X,Y,Z,O,P
380 LET L=X*Y*Z*O*P
390 PRINT "ERGEBNIS=";L
395 END
```

Eine weitere Anwendung der REM-Anweisung ist das Löschen einer Zeile, ohne sie aus dem Programmausdruck zu entfernen.

Beispiel:

```
10 A=20
20 B=30
30 REM C=40
40 D=A*B+C
```

In diesem Programm wurde für Testzwecke die Zeile 30 logisch ge-

löscht. Nach Entfernen der REM-Anweisung ist sie wieder in den Programmablauf eingefügt. Diese Technik wird dann angewendet, wenn die Zeilen sehr lang sind und zum Programmtesten aus dem Listing entfernt werden müssen.

GOTO

GOTO

Bis jetzt haben wir nur Befehle besprochen, die vom Computer nacheinander Schritt für Schritt ausgeführt werden. Alles ist der Reihe nach, mit der niedrigsten Zeilenzahl angefangen, bis zu höchsten Zeilenzahl abgelaufen.

Mit GOTO bekommen wir erstmals einen Befehl in die Hand, mit dem wir diesen Fluß umsteuern können. Der Befehl GOTO mit nachfolgender Zeilenzahl führt uns als unbedingter Sprung auf eine beliebig gewünschte Zeilenzahl. Dort läuft dann das Programm weiter ab (von der niederen Zeilenzahl zur höheren Zeilenzahl). Es sei denn, es stößt wieder auf einen anderen GOTO-Befehl.

```
1050 GOTO 300
1060 GOTO 200
```

Man kann nach vorne oder nach hinten im Programm springen. Der GOTO-Befehl läßt sich beim Zurückspringen auch für eine Schleife (Loop) verwenden.

Beispiel:

```
1070 PRINT "ICH BEFINDE MICH IN EINER"
1080 PRINT "GOTO-SCHLEIFE"
1090 GOTO 1070
```

Dieses kleine, simple Programm versetzt den Computer in eine Schleife und beläßt ihn dort, bis man durch die Break-Taste anhält.

Diese Wiederholfunktion läßt sich praktisch bei allen Programmen anwenden, die ständig durchlaufen werden müssen. Sollen andere Werte eingegeben werden, kann z. B. mit INPUT unterbrochen werden.

Beispiel:

```
1100 REM GOTO DEMOPROGRAMM
1110 PRINT "EINGABE A,B,C"
1120 INPUT A,B,C
1130 LET X=A*B*C
1140 PRINT "PRODUKT IST ";X
1150 GOTO 1110
```

Dieses Programm läuft solange, bis Sie es anhalten. Bitte sehen Sie im Manual Ihres speziellen Computers nach, welche Funktionstaste (CRL, Control C, RUN/STOP, etc.) hierfür in Frage kommt.

Das Programm läßt sich mit READ DATA auch von selbst anhalten, und zwar dann, wenn die Daten aufgebraucht sind.

```
1200 READ A,B,C
1210 LET D=A+B+C
1220 PRINT "DIE SUMME IST ";D
1230 DATA 10,20,30,40,50,60
1240 GOTO 1200
1250 END
```

Es endet mit einem OUT OF DATA-Error in Zeile 1200, da dem Programm nach dem zweiten Durchlauf die Daten ausgehen. Im ersten Durchlauf werden 10, 20 und 30 addiert. Im zweiten Durchlauf 40, 50 und 60. Dann sind die Daten aufgebraucht.

Wenn Sie jetzt noch mehrere Daten in Zeile 1230 eingeben, werden entsprechend mehrere Zyklen durchlaufen. Diesen OUT OF DATA-Error könnte man dadurch vermeiden, daß man dem Computer sagt, wie oft er durch die Schleife gehen soll. Hierzu sehen wir uns den nächsten Befehl an.

IF...THEN

IF . . . THEN

Der IF . . . THEN-Befehl dient zur Programmierung von Vergleichsoperationen und Entscheidungen. Es kann nach zwei Durchläufen in unserem vorhergehenden Programm z. B. eine Entscheidung getroffen werden, daß die Schleife verlassen werden soll.

```
1200 READ A,B,C
1210 LET D=A+B+C
1215 LET X=X+1
1220 PRINT "DIE SUMME IST ";D
1230 DATA 10,20,30,40,50,60
1240 IF X=2 THEN 1260
1250 GOTO 1200
1260 END
```

Bei diesem Programm zählen wir die Durchläufe mit dem Statement 1210 LET X = X + 1 und prüfen mit Zeile 1240 auf den Wert X. Wenn X = 2 ist, verlassen wir die Schleife und gehen zum Ende in Zele 1260.

Der IF . . . THEN-Befehl kann mit verschiedenen Bedingungen verknüpft werden.

IF (Bedingung) THEN (Was geschehen soll)

Die Bedingung kann z. B. sein:

X = 2
X ⟨ 3
X ⟩ 4
X = 2 + 3/4

Was geschehen soll, kann z. B. sein:

Zeilenzahl
GOTO-Zeilenzahl
PRINT "MITTEILUNG"

NOTIZEN

FOR...NEXT

FOR . . . NEXT

Der FOR . . . NEXT-Befehl erlaubt die einfache Programmierung von Schleifen. Oft haben wir beim Programmieren den Wunsch, daß ein Vorgang automatisch immer wieder wiederholt wird. Ein Beispiel wäre z. B. die Addition von mehreren Zahlen hintereinander.

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 etc.

Ein solches Problem könnte man auch ohne Schleife lösen, aber es würde erheblich mehr Aufwand erfordern und die Wahrscheinlichkeit, einen Fehler beim Programmieren zu machen, wäre auch relativ hoch.

Wir wollen nun an einem einfachen Programmbeispiel den Befehl FOR . . . NEXT demonstrieren.

```
400 FOR I=1 TO 100
410 PRINT I
420 NEXT I
```

Alles, was sich innerhalb der FOR-NEXT-Schleife befindet, in unserem Falle hier nur der Befehl PRINT I wird so lange wiederholt, wie in Zeile 400 angegeben wurde.

Insgesamt also 100 Schleifendurchläufe. Wenn hinter dem Befehl FOR I = 1 TO 100 nichts mehr folgt, so nimmt das Programm an, daß der Schritt 1 sein soll. Wir können dies z. B. ändern, indem wir folgendes programmieren:

```
430 FOR I=1 TO 100 STEP 2
440 PRINT I
450 NEXT I
```

Jetzt werden nur 50 Zahlen ausgedruckt, da der Schritt um zwei nach vorne erfolgt. Es können auch negative Schrittwerte verwendet werden.

```
460 FOR I=100 TO 1 STEP -2
470 PRINT I
480 NEXT I
```

Merke:

1. Anfangswert und Endwert müssen verschieden sein.
2. Die Laufvariable kann auch innerhalb der Schleife verwendet werden. Siehe vorheriges Beispiel. Sie wird durch FOR-TO-STEP festgelegt und läßt sich nicht mehr ändern.
3. Bei gleichem Anfangs- und Endwert wird die Schleife nur einmal durchlaufen.
4. Wenn die Schrittzahl positiv ist, muß der Anfangswert immer kleiner sein, als der Endwert.
5. Wenn die Schrittzahl negativ ist, muß der Anfangswert größer als der Endwert sein.

Verschachtelte FOR NEXT-Schleifen

Verschachtelte Schleifen bestehen aus zwei ineinanderliegenden Schleifen. Wichtig ist nur, daß die innere Schleife kleiner ist, als die äußere Schleife.

Dies bedeutet, daß immer zuerst die letzte Schleife durch NEXT abgeschlossen werden kann.

```
10 FOR X=1 TO 100
20 FOR Y=1 TO 10
25 PRINT Y;" ";
30 NEXT Y
35 PRINT X
40 NEXT X
```

Ein praktisches Beispiel:

Wir wollen nun einmal einen Casinogang simulieren. Es soll simuliert

werden, wie oft rot oder schwarz auf dem Rouletteller erscheint.

```
500 REM CASINOSIMULATION
510 REM WIE OFT KOMMT ROT UND SCHWARZ
520 PRINT "WUERFE","ROT","SCHWARZ"
530 FOR I=100 TO 1000 STEP 100
540 LET R=0
550 LET S=0
560 FOR J=1 TO I
570 LET A=INT(10*RND(1))
580 LET X=5*A
590 LET Y=X/2
600 LET W=INT(Y)
610 IF W=Y THEN 640
620 LET R=R+1
630 GOTO 650
640 LET S=S+1
650 NEXT J
660 PRINT I,R,S
670 NEXT I
680 END
```

NOTIZEN

DIM

Eindimensionale und mehrdimensionale Felder, indizierte Variablen

Wenn wir mehrere Variable addieren wollen, ist es oft von Vorteil, wenn man mit indizierten Variablen arbeitet. Nehmen wir einmal an, wir wollen 5 Variable addieren.

$$5 = b_1 + b_2 + b_3 + b_4 + b_5$$

In der Mathematik sieht dies wie folgt aus:

$$S = \sum_{K=1}^{5} b_K$$

Das Zeichen Σ gibt an, daß die Variablen b von K = 1 bis K = 5 addiert werden sollen.

Eine Variable wird in BASIC durch eine Variable mit nachfolgendem Klammerausdruck wie folgt dargestellt.

```
A(X)
B(C1)
C(Y + 1)
```

Das Argument in den Klammern darf nicht negativ sein. Bei den einzelnen BASIC-Versionen ist es auch teilweise begrenzt, z. B. bis 256 oder ähnlich.

Eine Ansammlung von Variablen mit dem selben Namen nennt man ein Feld oder auch eindimensionale Arrays.

Nachfolgend wollen wir ein Beispiel zur Anwendung eines Feldes

geben:

```
10  REM  DEMO FOR DIM
20  PRINT "WIEVIELE DATEN SOLLEN EINGEGBEN"
25  INPUT "WERDEN ? N=";N
30  DIM A(N)
40  REM  DATENEINGABE
50  FOR I = 1 TO N
60  PRINT "A(";I;: INPUT ")=";A(I)
70 Y = Y + A(I)
80  NEXT I
90 W = Y / N
100  PRINT "DER MITTELWERT IST ";W
```

Das vorherige Programm ermittelt uns den Durchschnittswert einer Reihe von Zahlen. Die Zahlen können vom Programmierer eingegeben werden. Ein Array wird aufgefüllt und die Werte addiert. Anschließend wird die Summe durch die Anzahl der Elemente geteilt.

Programme, wie oben beschrieben, können meist nur bis zu einer Größe von 10 Elementen verwendet werden. Bei mehr als 10 Elementen muß man die Anweisung DIM verwenden. Z. B. DIM A(N)

Der Befehl reserviert im Speicher einen bestimmten Platz für Variable. Dem Wert DIM folgen die verschiedenen Bezeichnungen für die Felder, z. B. DIM A(10), B(20), C(30)

Die einzelnen Feldbezeichnungen müssen durch Kommas getrennt werden.
Je nachdem, ob es sich um eindimensionale, zweidimensionale oder dreidimensionale Felder handelt, muß die Größe angegeben werden.

Zweidimensional: DIM A(10,15), B(22,30)
Eindimensional: DIM A(10), B(15), C(20)

Es können auch Arrays mit Strings aufgebaut werden. Dann werden eben keine Werte, sondern Zeichenketten abgespeichert.

```
DIM F$ (14,15)
```

Jedes Element kann im Array durch Angabe seines Platzes beschrieben werden. A (5,5) ist z. B. das Element in der 5. Reihe und 5. Spalte.

Bei der Dimensionierung von Arrays mit Zeichenketten treten Unterschiede auf. In Microsoft BASIC bedeutet die Angabe

```
DIM A$ (10)
```

ein Feld von 10 Zeichenketten mit je 256 Zeichen.

Beim ATARI, ZX81, SPECTRUM dagegen eine Zeichenkette mit 10 Zeichen.

Hier muß auch beachtet werden, daß bei einen Vergleich von Zeichenketten beide gleich dimensioniert sein müssen.

Beispiel (ATARI, ZX81, SPECTRUM):

```
DIM A$(5)
DIM B$(10)
LET A$ = "HANS"
LET B$ = "HANS"
```

Die Bedingung:

```
IF A$ = B$ THEN . . .
```

ist nicht erfüllt.

NOTIZEN

PEEK

PEEK

Der PEEK-Befehl erlaubt dem Programmierer, direkt in eine Speicherzelle zu sehen.

10 PEEK (Speicherzellennummer)

Die Speicherzellennummer muß dabei in Dezimal angegeben werden.

Die Sichtbarmachung des Speicherinhaltes geschieht durch

```
10 PRINT PEEK(59459)
20 PRINT PEEK(X)
```

Auslesen eines Speicherbereiches:

```
10 FOR X=1 TO 500
20 PRINT PEEK(X)
30 NEXT X
40 END
```

NOTIZEN

POKE

POKE

Der POKE-Befehl ermöglicht dem Programmierer das direkte Eingeben von Daten in eine Speicherzelle.

POKE A, B

wobei B = Daten oder eine Variable sein kann
und A = die Adresse in Dezimal.

10 POKE 59459,255

Da der Programmierer aber meist an Hexadezimalzahlen gewöhnt ist, muß zuerst in Dezimal umgerechnet werden.

Der POKE-Befehl erlaubt es, vom BASIC aus, Programme in Maschinensprache in einen gewünschten Speicherbereich zu legen. Über USR, CALL oder SYS-Befehle kann dann vom BASIC aus in das Maschinenunterprogramm gesprungen werden.

Beispiel:

```
10 REM POKE DEMO
20 FOR X=826 TO 836
30 READ Y
40 POKE X,Y
50 NEXT X
60 DATA 1,2,3,4,5,6,7,8,9,10,11
70 END
```

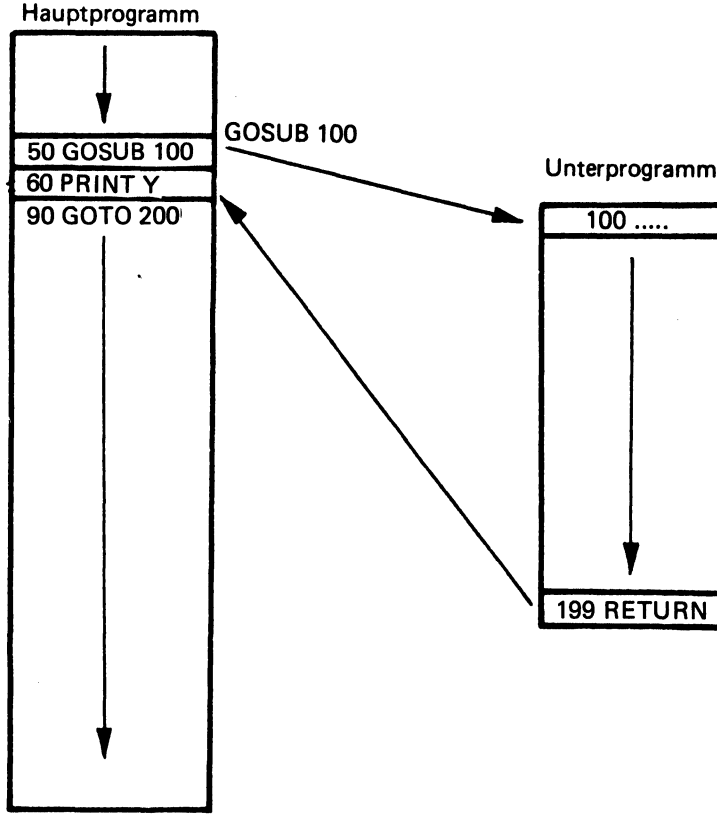
Das Programm liest die Werte 1 – 11 in die Speicherzellen 826 bis 836.

NOTIZEN

GOSUB...RETURN

GOSUB . . . RETURN

Der GOSUB-Befehl ist ein unbedingter Sprung in ein Unterprogramm. Dem GOSUB-Befehl folgt unmittelbar die Zeilennummer, in der das Unterprogramm beginnt. Am Ende des Unterprogrammes wird der Befehl RETURN gegeben. Dieser sorgt dafür, daß das Programm wieder zurück springt und im Hauptprogramm weiterfährt.



Programmbeispiel:

Das nachfolgende Programm verwendet eine sehr kleine Unterroutine, die eine bestimmte Berechnung durchführt. Eine solche Unterroutine kann jedoch auch aus wesentlich mehr Befehlszeilen bestehen. Sie können auch vom Unterprogramm aus noch in ein weiteres oder in mehrere Unterprogramme springen. Der Computer merkt sich automatisch die Zeilennummer, bei der er das Hauptprogramm verlassen hat und kehrt dann in die darauffolgende Zeile zurück.

```
10 REM DEMPROGRAMM FUER UNTERPROGRAMM
20 REM MIT DEM GOSUB BEFEHL
30 PRINT "GEBE EINE ZAHLE EIN ";
35 INPUT X
40 IF X=0 THEN 150
50 GOSUB 100
60 PRINT Y
70 GOTO 30
100 LET Y=X*15-3+4
110 RETURN
150 END
```

ON...GOSUB

ON . . . GOSUB

Der Befehl ON X GOSUB wird dann verwendet, wenn eine Reihe von Unterprogrammen aufgerufen werden sollen. Meist wird dazu eine Variable gesetzt (in unserem Beispiel X). Wird dieser Variablen der Wert 1 zugewiesen, so wird an die erste Adresse nach dem GOSUB-Befehl gesprungen. Wird der Variablen der Wert 2 zugewiesen, geht der Sprung an die darauffolgende Adresse.

Beispiel:

```
200 REM ON GOSUB DEMO
210 PRINT " EINGABE EINER ZAHL "
215 PRINT " ZWISCHEN 1 UND 5 ";
218 INPUT X
220 IF X>5 THEN 300
230 ON X GOSUB 260,280,290,295,285
235 PRINT Y
240 GOTO 210
260 Y=X*20
261 RETURN
280 Y=X/10
281 RETURN
285 Y=X+20
286 RETURN
290 Y=X+15
292 RETURN
295 Y=X^2
296 RETURN
300 END
```

Die Variable X kann auch ein mathematischer Ausdruck oder eine Formel sein. Z. B. X * 5 oder ähnlich.

Zweites Beispiel:

```
10  REM  ON GOSUB TEST
20  PRINT "MENUE-UEBERSICHT"
30  PRINT "(1) ADRESSEN EINGEBEN"
40  PRINT "(2) ADRESSEN AENDERN"
50  PRINT "(3) AUSDRUCKEN"
60  PRINT "(4) ENDE"
65  PRINT
70  INPUT "BITTE GEWUENSCHTE FUNKTION EINGEBEN:";X
80  ON X GOSUB 100,200,300,400
90  PRINT : GOTO 20
100  PRINT "GEBEN SIE RUHIG EIN!"
110  PRINT "TEST": RETURN
200  PRINT "AENDERN SIE RUHIG"
210  RETURN
300  PRINT "AHA JETZT WIRD GEDRUCKT"
310  RETURN
400  PRINT "AUF WIEDERSEHEN"
410  END
```

ON...GOTO

ON . . . GOTO

ON GOTO arbeitet ähnlich wie der ON GOSUB Befehl. Auch hier sind IF THEN Abfragen eingebaut. ON X GOTO 100, 200, 300 sagt dem Computer, daß je nach dem welcher Wert die Variable X (1, 2 oder 3) hat, das Programm nach Zeile 100, 200 oder 300 springen soll.

Beispiel:

```
10  REM  ON-GOTO TEST
30  INPUT "GEBEN SIE BITTE 1,2,3 ODER 4  EIN:";X
40  ON X GOTO 100,200,300,999
100  PRINT "SIE HABEN (1) EINGEGEBEN-STIMMTS ?"
110  GOTO 10
200  PRINT "SIE HABEN (2) EINGEGEBEN-STIMMTS ?"
210  GOTO 10
300  PRINT "SIE HABEN (3) EINGEGEBEN-STIMMTS ?"
310  GOTO 10
999  END
```

NOTIZEN

ON ERROR GOTO

ON ERROR GOTO

Das folgende Programmstück ist Teil eines Programmes, das Dateien auf einer Diskette verwaltet. Wird nun nach einer bestimmten Datei gesucht und das DOS findet diese nicht, wird das Programm mit einer Fehlermeldung abgebrochen. Um dies zu verhindern wird der ON ERROR GOTO Befehl verwendet.

```
5600 REM *****************
5605 REM DATEI VORHANDEN ?
5610 REM *****************
5615 ON ERR GOTO 5630
5620 PRINT D$;"VERIFY";F$;"CODE"
5625 GOTO 5650
5635 PRINT "*** DOS FEHLER ***":GOTO 400
5645 PRINT "*** DATEI NICHT VORHANDEN ***":GOTO 400
5650 RETURN
6530 I=PEEK(222):IF I=6 THEN GOTO 5645
```

In Zeile 5615 wird der Sprung in die Zeile 5630 bei Auftreten eines Fehlers in der folgenden Zeile 5620 programmiert. Dort wird in Zelle 222 nachgeschaut welcher DOS-Fehler aufgetreten ist. Bei nicht vorhandener Datei wird eine entsprechende Fehlermeldung ausgegeben. Tritt ein anderer Fehler auf, erscheint die Meldung DOS-FEHLER.

Nach einem tatsächlich aufgetretenen Fehler sind alle Rücksprungadressen verloren, so daß mit GOTO das Programm weitergeführt werden muß (gilt nur für APPLE II).

NOTIZEN

USR (X)

USR(X)

Die USR(X)-Funktion wird in einigen Microsoft-BASIC-Versionen dazu benutzt, einen Wert X an ein Maschinenunterprogramm zu übergeben. Die Anfangsadresse dieser Maschinenroutine muß dann in bestimmten Zellen (beim APPLE z. B. 0A, 0B) gespeichert sein. Der Wert X wird in den Floating-Point Accumulator geschrieben.

Eine andere Bedeutung hat der USR-Befehl beim ZX81 oder Spectrum. Hier ist X die Anfangsadresse eines Maschinenprogrammes. Dieser Befehl muß in einer Dummy-Zuweisung verwendet werden. Z. B.

```
PRINT USR X
```

Dann wird beim Rücksprung der Inhalt des BC-Registers der Z80 CPU auf den Bildschirm ausgegeben.

NOTIZEN

Funktionen

Bei einer Funktion wird der Wert einer Variablen durch die Funktion und durch das an die Funktion übergebene Argument bestimmt.

```
10 PI = 3.14159
20 Y = SIN (PI/2)
```

Der Wert von Y (= 1) wird durch die Funktion sin (sinus) und durch das Argument Pi/2 bestimmt. Das Argument muß bei allen Rechnern in Klammern gesetzt werden.
Ausnahme: ZX81, SPECTRUM.

Hier kann

```
20 Y = SIN PI
```

eingegeben werden, wenn das Argument aus einer Zahl oder Variablen besteht. Zusammengesetzte Ausdrücke müssen auch hier in Klammern gesetzt werden.

```
20 Y = SIN (PI/2)
```

RND

RND

Die Random-Funktion erzeugt Zufallszahlen. Diese Funktion hat auf den verschiedensten Rechnern unterschiedliches Verhalten.

ZX81, SPECTRUM

LET R = RND * 26 + 1

erzeugt Zufallszahlen zwischen 1 und 26.
Zusatzbefehl: RAND
Nach RAND 1 wird immer die gleiche Zufallszahl erzeugt. Nach RAND 0 oder nur RAND wird wieder eine Folge von Zufallszahlen erzeugt.

APPLE

R = RND(X)

X > 0 Erzeugt Zufallszahlen größer, gleich Null und kleiner 1 (0 = R < 1).

X = 0 Es wird immer die gleiche Zufallszahl erzeugt.

X < 0 Erzeugt für jedes Argument X die gleiche Zufallszahl.
RND (–1) ist ungleich RND (–4)

ATARI

R = RND(X) Erzeugt Zufallszahl größer, gleich 0 und kleiner 1
(0 = < R < 1).

X ist eine Dummy-Variable. Sie beeinflusst nicht die Bildung der Zufallszahlen.

COMMODORE-64

R = RND(X)

X > 0 Erzeugt Zufallszahl größer, gleich Null und kleiner 1.

Es wird immer die gleiche Folge von Zufallszahlen gebildet.

X = 0 Die Zufallszahl wird durch einen Zähler im Rechner gebildet.

X > 0 Bei jeden Programmlauf wird eine andere Folge von Zufallszahlen erzeugt.

INT

INT

Die INT-Funktion (Integer = ganze Zahl) liefert die ganze Zahl aus einer gebrochenen Zahl.

```
10 PRINT INT(X)
20 LET Y=INT(Z+I)
30 LET X=INT(100*RND(1))
```

Sie wird dort angewendet, wo von gebrochenen Zahlen nur der ganzzahlige, positive Wert benötigt wird, z. B. in unserer Casionosimulation. Dort werden Zufallszahlen zwischen 0 und 1 erzeugt (0,000001 ..... 0,999999).

Indem man die ganzen Zahlen herauslöst und mit 100 multipliziert, erhält man Zufallszahlen zwischen 0 und 100.

```
10 REM DEMOPROGRAMM
20 REM FUER DIE INT(X) FUNKTION
30 FOR X=1 TO 10
40 LET I=X/3
50 PRINT INT(I);" ";
60 PRINT X
70 NEXT X
```

Achtung: Beim ZX81 und SPECTRUM
INT (0.5 x 6) liefert 2 und nicht 3, wie bei anderen Rechnern.

NOTIZEN

SQR

SQR

Die SQR(X)-Funktion (Quadratwurzelfunktion) liefert die Quadratwurzel des Wertes, der in der nachfolgenden Klammer steht.

Beispiel:

```
10 PRINT SQR(100)
```

Der Ausdruck kann auch in einer Rechenoperation verwendet werden.

```
20 PRINT SQR(10*15+SQR(5))
30 LET X=2
40 Y=SQR(X)
50 PRINT Y
60 FOR X=1 TO 100
70 LET Y=SQR(X)
80 PRINT Y
90 NEXT X
100 PRINT "EINGABE EINER ZAHL ";
105 INPUT X
110 PRINT "DIE WURZEL IST ";SQR(X)
120 END
```

NOTIZEN

ABS

ABS

Mit der ABS-Funktion können Sie den Absolutwert einer Zahl oder einer Variablen erhalten. Unter dem Absolutwert einer Zahl versteht man die eigentliche Zahl ohne jedes Vorzeichen.

Beispiele:

a) PRINT ABS (−12)
druckt die Zahl 12 aus (ohne den Minusvorzeichen).
PRINT ABS (+14)
druckt die Zahl 14 ohne das Pluszeichen aus.

b) 5 LET X = −2.5 : LET Y = +4.75
10 PRINT ABS(X)
20 PRINT ABS(Y)

NOTIZEN

SIN

SIN

Diese Funktion liefert den Sinuswert für ein gegebenes Argument.

```
 5 PI = 3.14159
10 PRINT SIN (PI/2)
```

Das Argument wird immer im Bogemaß angegeben. Hier entspricht einen Winkel mit den Gradmaß 0^{o} der Wert 0 im Bogenmaß und einen Winkel von 360^{o} der Wert 2*PI.
Die Umrechnung von Grad ins Bogenmaß kann durch

$$Y = \frac{X}{180} * PI$$

mit X im Gradmaß erfolgen.

Beim ATARI kann durch die Anweisung DEGREE der Rechner zur Rechnung ins Gradmaß umgeschaltet werden.

COS

COS

Berechnung der Cosinusfunktion.

NOTIZEN

TAN

TAN

Berechnung der Tangensfunktion.

ATN

ATN

ATN ist die Umkehrfunktion zur Tangensfunktion. Sie liefert den Arcus-Tangens zu einem gegebenen Argument. Der Wert der Funktion liegt zwischen –PI/2 und +PI/2.

LOG

LOG

Diese Funktion liefert den natürlichen Logarithmus (ln) zu einem gegebenen Argument.

$$Y = \frac{LOG(X)}{LOG(10)}$$

rechnet den natürlichen Logarithmus (Basis 2) in den decadischen Logarithmus (Basis 10) um.

NOTIZEN

EXP

EXP

Durch EXP wird der Wert der e-Funktion für ein gegebenes Argument berechnet.

SGN

SGN

Die SGN (Signum) Funktion liefert den Wert des Vorzeichens für ein gegebenes Argument.

```
Y = SGN(X)
Y = 1  wenn  X > = 0
Y = −1  wenn  X < 0
```

Formeln für Funktionen, die nicht in BASIC enthalten sind:

Sinus-Umkehrfunktion ARCSIN

```
Y = ATN (X/SQR(−X*X+1))
```

Cosinus-Umkehrfunktion ARCCOS

```
Y = −ATN (X/SQR(−X*X+1))+PI/2
```

Sinus-Hyperbolicus SINH

```
Y = (EXP(X)−EXP(−X))/2
```

Cosinus-Hyperbolicus COSH

```
Y = (EXP(X)+EXP(−X))/2
```

Tangens-Hyperbolicus TANH

```
Y = -EXP(-X)/(EXP(X) + EXP(-X))*2 + 1
```

Modulo-Funktion

Die Modulo-Funktion liefert den Rest Y, der bei einer Division A/B auftritt.

```
Y = INT (A/B -INT(A/B))*B + 0.05
```

Diese Funktionen können mit DEF FN in eine BASIC-Programm eingebaut werden.

DEF FN

DEF FN

Wenn ein Teil eines Programmes als Funktion verwendet wird, und sehr häufig beim Programmieren eingesetzt werden muß, empfiehlt es sich, es über DEF in eine Funktion zu verwandeln.

DEF ermöglicht es dem Programmierer, z. B. einen mathematischen Ausdruck wie B = (X –5)/Y als Funktion festzulegen und über DEF aufzurufen.

Der DEF-Befehl besteht aus dem Wort DEF, gefolgt von einer Definition. Danach folgt ein Gleichheitszeichen mit einer Formel, Konstanten oder Variablen. Die Definition ist auch der Name der Funktion.

```
100 DEF FNI (X,Y) = (X + 1) / (Y – 3)
```

Es können auch Strings eingesetzt werden.

```
200 DEF FNI$ = "ELCOMP"
```

Programmbeispiel:

Die nachfolgede Programmzeile soll gebrochene Zahlen aufrunden:

```
10 DEF FNR(A) = INT (X * 100 + 0.5)/100
```

In Zeile 10 wird der mathematische Ausdruck rechts neben dem Gleichheitszeichen als Funktion FNR (A) festgelegt. Der Wert X in DEF FNR(X) ist nur eine Ersatzvariable, die später in die FNR(X)-Funktion eingesetzt wird. Jedesmal, wenn die Funktion benötigt wird, kann ich X durch die gewünschte Variable ersetzen.

```
F = FNR (X)
```

liefert dann das Ergebnis wie

```
Y = INT (X * 100 + 0.5)/100
```

Man spart sich dadurch sehr viel Schreibarbeit und vermindert die Fehlerwahrscheinlichkeit.

AND

AND

Der AND-Befehl wird in einzelnen Computern zur logischen Verknüpfung in FOR NEXT Befehlen verwendet. Weiterhin wird der AND-Befehl zur logischen Verknüpfung zweier Binärzahlen verwendet.

```
10 REM AND DEMO
20 X=12
30 Y=22
40 IF X=12 AND Y=22 THEN 100
100 PRINT "AHA! - ES GEHT"
110 REM AENDERN SIE JETZT EINMAL
120 REM DIE ZEILE 20 IN X=13
```

Sie können den AND-Befehl immer dann verwenden, wenn Sie mehrere Bedingungen abfragen müssen. Z. B. in einer Adressliste. Alle Kunden mit Anfangsbuchstaben B aus dem Postleitzahlgebiet 8150.

Einige Computer erlauben es mit der AND-Funktion auf zwei bestimmte Kriterien prüfen zu lassen. Wenn die beiden Kriterien erfüllt sind, liefert die AND-Funktion den Wert 1. Wenn einer der beiden Bedingungen nicht erfüllt ist, wird eine Null geliefert.

Beispiel:

```
10 REM AND TEST FUER LOGISCHE OPERATIONEN
20 PRINT "GEBEN SIE EINE ZAHL ZWISCHEN 2 UND 9 EIN "
30 INPUT X
35 IF X<0 THEN END
```

```
40 IF X>3 AND X<10 THEN GOTO 100
55 PRINT X;" IST NICHT GROESSER ALS 3"
56 PRINT " UND KLEINER ALS 10"
57 GOTO 10
100 PRINT X;" IST GROESSER ALS 3"
110 PRINT " UND KLEINER ALS 10"
120 GOTO 10
```

Sie können jedoch auch mit der AND-Funktion logische Operationen simulieren, z. B. die UND-Funktion.

UND-Funktion Tabelle:

Mit der AND-Funktion können Sie leicht die Wertetabelle eines UND-Gatters errechnen und ausdrucken (siehe Beispiel).

```
10 REM UND FUNKTION
20 X=0 AND 0
30 Y=1 AND 0
40 Z=0 AND 1
50 U=1 AND 1
60 PRINT "LOGIKTABELLE FUER UND"
70 PRINT
80 PRINT "0 UND 0 = ";X
90 PRINT "1 UND 0 = ";Y
100 PRINT "0 UND 1 = ";Z
110 PRINT "1 UND 1 = ";U
```

Bei einigen Rechnern verknüpft die AND-Funktion jede binäre Stelle des Wertes mit der binären Stelle des anderen Wertes.

Diese Technik wird oft in der Ein-, Ausgabeprogrammierung und bei der Programmierung von Joysticks etc. verwendet. D. h. überall dort wo einzlne Bits manipuliert werden müssen. Zusammen mit der OR-Funktion und einer Negierung (Verneinung NOT) können Sie alle beliebigen logischen Fuktionen simulieren und Ihr Programm einbauen.

OR

OR

Die OR-Funktion wird zu logischen Verknüpfung von zwei Zahlen oder Ausdrücken verwendet. Wenn einer der beiden Ausdrücke "wahr" ist, ist die Bedingung erfüllt. Genau wie der der AND-Funktion werden bei Dezimalzahlen die binären Aquivalente "geodert".

```
10 REM ODER FUNKTION DEMO
20 X=0 OR 0
30 Y=1 OR 0
40 Z=0 OR 1
50 U=1 OR 1
60 PRINT
70 PRINT "0 ODER 0 = ";X
80 PRINT "1 ODER 0 = ";Y
90 PRINT "0 ODER 1 = ";Z
100 PRINT "1 ODER 1 = ";U
```

```
10 IF B > A OR A > C THEN 7
   Ausdruck 1  Ausdruck 2
```

Wenn eine der beiden Ausdrücke erfüllt ist, B größer A oder A größer C, springt das Programm nach Zeile 7. Dieser Befehl wird oft zur Abfrage von Tasten bei der Eingabe verwendet.

NOTIZEN

NOT

NOT

Die NOT-Funktion in Microsoft-BASIC entspricht der logischen NICHT-Verknüpfung (Invertierung). Sie invertiert die binären Repräsentationen der einzelnen Werte. Ähnlich wie AND und OR.

```
10 REM NOT-FUNKTION
20 X=0
30 Y= NOT X
40 PRINT Y
```

Mit NOT wird die negative Zahl im Zweier-Complement gebildet.

Complementieren einer Binärzahl bedeutet vertauschen von Null und Eins. Bei der Bildung des 2-er Complements wird zu dieser Vertauschung noch eine Eins dazuaddiert.
Beispiel: Bestimmung der 2-er Complemente von +1:

	+1	=	% 0 0 0 0 0 0 0 1	
Complement		=	% 1 1 1 1 1 1 1 0	
Eins dazuaddiert			+% 0 0 0 0 0 0 0 1	
ergibt	−1	=	% 1 1 1 1 1 1 1 1	= $FF

Die Zahl −1 wird also durch % 1 1 1 1 1 1 1 1 oder $FF gekennzeichnet.

NOTIZEN

Funktionen für Zeichenketten

Die Sprache BASIC enthält Funktionen, die auf Zeichenketten angewendet werden können.

Diese Zeichenketten können "addiert", aufgeteilt und miteinander verglichen werden.

ASC(X$)

ASC(X$)

Die ASC(X$)-Funktion verwandelt ein Zeichen oder eine String-Variable in den ASCII-Integercode.

Diese Werte finden Sie in einer Dezimal-ASCII-Tabelle.
PRINT ASC("B") druckt den Dezimal-ASCII-Wert = 66 des Buchstabens B auf dem Bildschirm aus.

```
10  REM  ASC(X$) TEST
15  GOTO 100
20  PRINT "DER ASCII WERT FUER DEN"
30  INPUT "BUCHSTABEN:";X$
40  PRINT "IST ";
50  PRINT  ASC (X$)
55  PRINT
60  RETURN
99  END
100  GOSUB 20
110  IF  ASC (X$) = 65 THEN  END
120  GOTO 100
```

ZX81 verwendet CODE.
CODA A$ gibt den intern vereinbarten Code für A$. Dieser entspricht nicht dem ASCII-Code. Das gleiche gilt für SPECTRUM.

ASCII CHARACTER CODES

Code		Char	Code		Char	Code		Char	Code		Char
Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
Ø	ØØ	NUL	32	2Ø	SP	64	4Ø	@	96	6Ø	
1	Ø1	SOH	33	21	!	65	41	A	97	61	a
2	Ø2	STX	34	22	"	66	42	B	98	62	b
3	Ø3	ETX	35	23	#	67	43	C	99	63	c
4	Ø4	EOT	36	24	$	68	44	D	1ØØ	64	d
5	Ø5	ENQ	37	25	%	69	45	E	1Ø1	65	e
6	Ø6	ACK	38	26	&	7Ø	46	F	1Ø2	66	f
7	Ø7	BEL	39	27	'	71	47	G	1Ø3	67	g
8	Ø8	BS	4Ø	28	(	72	48	H	1Ø4	68	h
9	Ø9	HT	41	29	)	73	49	I	1Ø5	69	i
1Ø	ØA	LF	42	2A	*	74	4A	J	1Ø6	6A	j
11	ØB	VT	43	2B	+	75	4B	K	1Ø7	6B	k
12	ØC	FF	44	2C	,	76	4C	L	1Ø8	6C	l
13	ØD	CR	45	2D	-	77	4D	M	1Ø9	6D	m
14	ØE	SO	46	2E	.	78	4E	N	11Ø	6E	n
15	ØF	SI	47	2F	/	89	4F	O	111	6F	o
16	1Ø	DLE	48	3Ø	Ø	8Ø	5Ø	P	112	7Ø	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	5Ø	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
2Ø	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	12Ø	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	9Ø	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	6Ø	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
3Ø	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

CHR$(X)

CHR$(X)

Dieser Befehl bewirkt genau das Gegenteil, was ASC (X$) bewirkt. CHR$(X) besorgt Ihnen aus dem Dezimal-ASCII-Wert X in der Klammer den zugehörigen Buchstaben oder das zugehörige Zeichen.

PRINT CHR$(66) durckt Ihnen z. B. den Buchstaben B auf den Bildschrim. Der gesamte ASCII-Code kann mit den Zahlen 0 — 127 dargestellt werden. Einige Computer haben einen erweiterten Vorrat mit 255 ASCII-Zeichen, Symbole, Spielesymbole etc. (Grafik).

```
10  REM  CHR$(X) DEMO
20  FOR X = 1 TO 255
30 A$ =  CHR$ (X)
40  PRINT A$;" ";
50  NEXT X
```

Dieses Programm druckt Ihnen alle ASCII-Zeichen mit den Werte 1 — 255 auf dem Bildschirm aus.

Beim ZX81 wird das intern verwendete Zeichen gebracht. Beim SPECTRUM der ASCII-Code.

NOTIZEN

LEN (X$)

LEN(X$)

Der LEN(X$) Befehl liefert Ihnen die Anzahl der BYTE (Zeichen), eines Strings X$. Diesen Befehl brauchen Sie immer dort, wo Sie die Länge eines Strings oder einer Zahl wissen müssen. Handelt es sich um eine Zahl, müssen Sie diese zuerst in einen String umwandeln.

```
10 A$ = "ELCOMP":B$ = "SOFTWARE"
20 X$ = A$: GOSUB 100
30 X$ = B$: GOSUB 100
40  END
100  HTAB (40 -  LEN (X$)): PRINT X$
110  RETURN
```

Dieses kleine Programm druckt die Zeichenketten A$ und B$ rechtsbündig am rechten Rand aus.

Die Zahl 40 wurde so gewählt, weil die meisten Rechner 40 Zeichen/Zeile ausgeben.

Beispiel:

```
10  REM  LEN(X$) DEMO
20  INPUT "GEBEN SIE EINIGE ZEICHEN EIN ";A$
30  PRINT "SIE HABEN "; LEN (A$);" ZEICHEN EINGEGEBEN"
```

NOTIZEN

LEFT$(X$,Y)

LEFT$(X$,Y)

Dieser Befehl gibt Ihnen die ersten Y-Zeichen von links in einen String. Sie brauchen diesen Befehl unbedingt, wenn Sie mit Files und ernsthaften Geschäftsprogrammen arbeiten wollen.

Der Befehl:

PRINT LEFT$("HOFACKER VERLAG",8)

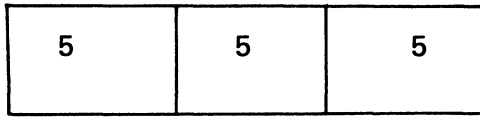
druckt Ihnen auf dem Bildschirm HOFACKER VERLAG aus.

Y = 8 (von links die ersten 8 Zeichen)

"HOFACKER VERLAG"

Der Befehl wird u. a. nach einer Zusammenfassung von mehreren Strings dazu verwendet umd die beiden wieder zu trennen.

Auch bei der Festlegung von Feldern in Files können Sie den LEFT$ (X$,Y) Befehl verwenden.



3 Felder a' 5 Stellen

1. Feld	2. Feld	3. Feld
B$	H$	I$

Sie wollen Zahlen in diese Felder einschreiben und immer auf fünf Stellen auffüllen (mit Leerzeichen oder Sternchen). Zuerst legen Sie ein Dummy-String mit fünf Leerfeldern fest. Z. B. N$ = " – – – – –" – = Leerzeichen (Blanks).

Dann verwandeln Sie Ihre Zahl in einen String und addieren den String, bestehend aus 5 Leerzeichen zu der in A ein String umgewandelten Zahl.

```
10  REM  LEFT$(A$,X) DEMO
20 G$ = "2":H$ = "15":I$ = "105"
30 N$ = "*****"
50 X$ = G$: GOSUB 100
60 X$ = H$: GOSUB 100
70 X$ = I$: GOSUB 100
80  END
100 X$ =  LEFT$ (X$ + N$,5)
105  PRINT X$;
110  RETURN

]RUN
2****15***105**
```

Im Beispiel haben wir Sternchen anstelle von Leerzeichen verwendet. Die Stringwerte 2, 15 und 105 sind jetzt in Felder gepackt, welche jeweils 5 Zeichen lang sind.

Wir sehen später, bei der Erläuterung des MID$(X$, Y, Z) Befehles, warum man den LEN(X$) Befehl in diesem Zusammenhang anwendet.

ZX81 und SPECTRUM verwenden die Schreibweise

A$ (n TO n')

Die Zeichenkette wird von Zeichen n bis Zeichen n' ausgedruckt. Mit A$ = "ELCOMP" entspricht

LEFT$ (A$,2) gleich A$ (1 TO 2) oder auch
A$ (TO 2)

Beim ATARI wird die Schreibweise

A$ (n, n')

verwendet. PRINT A$ (1,2) druckt, beginnend beim ersten Zeichen, 2 Zeichen aus.

MID$(X$,Y, Z)

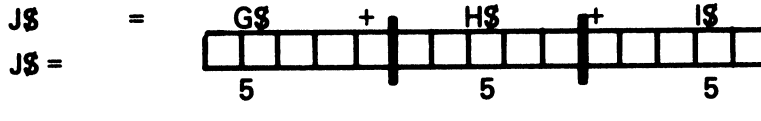
MID$(X$,Y,Z)

Der MID$-Befehl wird dazu verwendet aus einem String Z-Zeichen herauszuschneiden, welche Y-Zeichen vom äußerst linken Zeichen an beginnen.

```
                    8     4
PRINT MID$ ("ELCOMP–MAGAZINE",8,4)
```

würde auf dem Bildschirm MAGA erscheinen lassen.

Diesen Befehl verwendet man u. a. zum Auftrennen von Summenstrings. Sehen Sie bitte in diesen Zusammenhang den Befehl LEFT$ (X$,Y) noch einmal an. Nehmen wir an, wir haben für die dort verwendeten einzelnen Stringfelder mit je 5 Zeichen eine Summenstringvariable



gebildet.

Dieses Variable speichern wir auf Diskette oder Cassette. Anschließend lesen wir diese wieder ein und müssen sie wieder trennen, um unsere einzelnen Variablen wieder zu finden. Dazu benötigen wir den MID$ Befehl.

Beispiel:

```
10  REM  LEFT$(A$,X) DEMO
20 G$ = "2":H$ = "15":I$ = "105"
30 N$ = "******"
50 X$ = G$: GOSUB 100
60 X$ = H$: GOSUB 100
70 X$ = I$: GOSUB 100
75  PRINT
80 G$ =  LEFT$ (J$,5)
85 H$ =  MID$ (J$,6,5)
90 I$ =  RIGHT$ (J$,5)
95  PRINT G$: PRINT H$: PRINT I$
99  END
100 X$ =  LEFT$ (X$ + N$,5)
105  PRINT X$;
110 J$ = J$ + X$
120  RETURN

]RUN
2****15***105**
2****
15***
105**
```

Sehen Sie sich dieses Beispiel einmal ganz genau an. Diese Technik ist sehr wichtig für alle Geschäftsprogramme. Die Stringvariablen G$, H$ und I$ werden so aufbereitet, daß die jeweiligen Felder immer 5 Zeichen lang sind, gleich welche Zahl zwischen 0 und 99999 Sie hineinschreiben. Mit dem LEFT$ Befehl geschieht die Einteilung in 5 Gruppen. Anstelle der Asteriks (Sternchen) können Sie auch Leerzeichen im String N$ verwenden. Die einzelnen Fünfergruppen werden dann im Summenstring J$ durch Addition zusammengefasst.

Hierzu verwenden wir den LEFT$ Befehl für die erste Fünfergruppe.

Der MID$ Befehl wird für die zweite Fünfergruppe und der RIGHT$ Befehl für die dritte Fünfergruppe.

Anschließend sehen Sie, wie die Zahlen wieder in gleicher Weise erscheinen wie wir sie am Anfang aufbereitet haben. Diese Technik können Sie bei allen Geschäftsprogrammen wie z. B. Adressenliste, Lagerverwaltung etc. verwenden. Die einzelnen Datensätze werden dann in einem String zusammengefasst und auf Diskette gespeichert.

```
A$"          " 5 Zeichen für Kundennummer
B$"                                  " 20 Zeichen für Name
C$"                        " 12 Zeichen für Postleitzahl
D$"                                  " 20 Zeichen für Ort

E$ = " * * * * * "
F$ = " * * * * * * * * * * * * * * * * * * * * " 20 Asteriks
G$ = " * * * * * * * * * * * * " 12 Asteriks
A$ = LEFT$(A$+N$,5)
B$ = LEFT$(B$+F$,20)
C$ = LEFT$(C$+G$,12)
D$ = LEFT$(D$+F$,20)  usw.
```

Ein anderes Beispiel:

```
10 A$ = " HOFACKER VERLAG "
15 L$ = "              ": REM  10 LEERZEICHEN
20 X =  INT ( LEN (A$) / 2) + 2
30  FOR I = 1 TO  LEN (A$) / 2
40  PRINT  LEFT$ (L$,X - I + 1); MID$ (A$,X - I,2 * I - 1)
50  NEXT I

]RUN
          R
         ER
        KER V
       CKER VE
      ACKER VER
     FACKER VERL
    OFACKER VERLA
   HOFACKER VERLAG
```

Beim ZX81 und SPECTRUM

MID$ (AS,2,5) lautet hier
A$ (2 TO 5)

Beim ATARI

A$ (2,5)

RIGHT$(X$,Y)

RIGHT$(X$,Y)

Dieser Befehl arbeitet ähnlich wie der LEFT$(X$,Y), nur wird hier nicht von links, sondern von rechts her gezählt.

Sie können den RIGHT-Befehl für das gleiche Beispiel wie unter LEFT$(X$,Y) verwenden. Es ergäbe sich folgender Unterschied im Feldaufbau.

RIGHT$(X$,Y,5)

Feld

00002
00015
00105

LEFT$(X$,Y,5)

Feld

20000
15000
10500

0 = Blanks oder auch Sternchen

Beispiel für RIGHT$(X$,Y)

```
10  REM  RIGHT$(A$,X) DEMO
20 G$ = "2":H$ = "15":I$ = "105"
30 N$ = "*****"
50 X$ = G$: GOSUB 100
60 X$ = H$: GOSUB 100
70 X$ = I$: GOSUB 100
80  END
100 X$ =  RIGHT$ (N$ + X$,5)
105  PRINT X$;
110  RETURN

]RUN
****2***15**105
```

```
10 A$ = "HOFACKER VERLAG"
20  FOR I = 1 TO  LEN (A$)
30  PRINT  RIGHT$ (A$,I)
40  NEXT I

]RUN
G
AG
LAG
RLAG
ERLAG
VERLAG
 VERLAG
R VERLAG
ER VERLAG
KER VERLAG
CKER VERLAG
ACKER VERLAG
FACKER VERLAG
OFACKER VERLAG
HOFACKER VERLAG
```

Beim ZX81 und SPECTRUM entspricht RICHT$(A$,5)

A$ (LEN A$–5 TO LEN A$)

Beim ATARI

A$ (LEN(A$)–5, 5)

STR$(X)

STR$(X)

Dieser Befehl liefert einen String, welcher die numerische Zahl X repräsentiert. Sie können damit Zahlen in einen String umwandeln. Der Befehl wird oft dort verwendet wo Zahlen auf Diskette oder Cassette gespeichert werden müssen. Meist kann man jedoch nur Strings auf Diskette und Cassette speichern. Man verwandelt daher Zahlen in Strings oder behandelt sie von Anfang an als String. Erst wenn Berechnungen durchgeführt werden müssen, werden sie wieder in numerische Zahlen oder Variablen verwandelt. Wenn Sie z. B. die Anzahl der Stellen einer Zahl zählen wollen, verwandeln Sie diese zuerst einmal mit STR$(X) in einen String. Anschließend zählen Sie die Stellen mit LEN(X$).

Beispiel:

```
10  REM  STR$ DEMO
20  INPUT "BETRAG:";A
30 B = B + A
40 B$ =  STR$ (B)
45 B1$ =  LEFT$ (B$, LEN (B$) - 2)
47 B2$ =  RIGHT$ (B$,2)
49 B$ = B1$ + "." + B2$
50  HTAB 40 -  LEN (B$): PRINT B$
60  GOTO 20

]RUN
BETRAG:1000
                                    10.00
BETRAG:10000
                                   110.00
BETRAG:233
                                   112.33
BETRAG:
```

Im Beispielprogramm wird in Zeile 20 eine Zahl A eingegeben. Diese Zahl entspricht der Einheit Pfennig. Sie wird zur schon vorhandenen Zahl B addiert. Die Ausgabe soll nun rechtsbündig am rechten Bildschirmrand ausgegeben werden. Außerdem soll ein Dezimalpunkt bei der Ausgabe eingesetzt werden.

Die Zahl B wird deshalb in eine Zeichenkette B$ gewandelt und diese in zwei Teilketten B1$ und B2$ aufgeteilt.

In Zeile 49 wird zwischen B1$ und B2$ ein Dezimalpunkt eingefügt.

Die Ausgabe erfolgt in Zeile 50. HTAB für den APPLE muß durch TAB oder PRINT AT für andere Rechner ersetzt werden.

VAL(X$)

VAL(X$)

Dieser Befehl verwandelt die String-Variable X$ wieder zurück in eine Zahl. Er stellt die Umkehrung des Befehles STR$(X) dar.

```
10  REM  VAL(X$) DEMO
20  INPUT "BETRAG:";A$
22  FOR I = 1 TO  LEN (A$)
24  IF  MID$ (A$,I,1) = "." THEN 26
25  NEXT I: GOTO 20
26 A$ =  LEFT$ (A$,I - 1) +  RIGHT$ (A$, LEN (A$) - I)
30 B = B +  VAL (A$)
40 B$ =  STR$ (B)
45 B1$ =  LEFT$ (B$, LEN (B$) - 2)
47 B2$ =  RIGHT$ (B$,2)
49 B$ = B1$ + "." + B2$
50  HTAB 40 -  LEN (B$): PRINT B$
60  GOTO 20

]RUN
BETRAG:10.00
                                        10.00
BETRAG:122.50
                                       132.50
BETRAG:1234.56
                                      1367.06
BETRAG:10000.00
                                     11367.06
BETRAG:
```

VAL(X$) verwendet man immer dann, wenn Zahlen als String verarbeitet wurden und es so weit ist, daß man mit den Zahlen rechnen muß. Manipulieren Sie alle Zahlen so lange wie möglich als Strings und wandeln Sie erst vor der Rechenoperation in eine Zahl um.

Im Beispiel erfolgt die Eingabe als Zeichenkette. Vor dem Wandeln der Zeichenkette in eine Zahl wird der Dezimalpunkt entfernt. Ab Zeile 30 entspricht der Programmablauf dem Beispiel STR$.

BASIC-Grundkurs

BASIC-Grundkurs

Die folgende Einführung ist für den ersten Kontakt mit einem Computer geschrieben. Dabei werden die meisten Befehle, die in der Zusammenstellung aufgeführt sind, nochmals in kleinen Programmen vorgestellt. Es wird allerdings versucht, so wenig als möglich, rechnerspezifische Befehle zu verwenden. Als Grundlage für die Befehle soll die BASIC-Version von MICROSOFT dienen. Diese Version dürfte wohl die am weitesten verbreitete sein.

Der erste Kontakt:

Wenn man zum ersten Mal vor einem Computer sitzt, so sieht man die Tastatur und den Bildschirm. Auf letzterem ist ein weißes blinkendes Rechteck zu sehen. Dieses ist der Cursor, und zeigt an, daß der Computer auf eine Eingabe wartet. Ist kein Cursor zu sehen, so gibt es zwei Möglichkeiten. Entweder der Rechner bearbeitet ein Programm, oder er ist bei der Ausführung von Befehlen "im Wald" gelandet. Das bedeutet, der Rechner irrt zwischen Befehlen hin- und her und findet keinen Ausweg mehr. In diesem Fall ist es das Beste, den Rechner aus- und wieder einzuschalten. Die Hersteller von Rechnern führen an dieser Stelle einen ersten Intelligenztest mit dem Benutzer durch. Der Einschaltknopf ist meist so angebracht, daß er erst nach längeren Suchen gefunden wird.

Nach dem Einschalten des Rechners erscheint meistens ein Hinweis auf den Rechner und den Hersteller und der blinkende Cursor ist zu sehen. Nun kann ein erster Befehl eingegeben werden. Der Rechner soll als erstes, den Namen des Benutzers auf den Bildschirm ausgeben. Dazu wird der PRINT-Befehl verwendet. In den Rechner wird die Anweisung

```
PRINT "HEINRICH MEIER"
```

über die Tastatur eingegeben. Nach dem Drücken einer Taste, rückt der Cursor eine Stelle weiter. Ist das letzte Zeichen " eingegeben, so muß dem Rechner mitgeteilt werden, daß die Eingabe beendet ist. Dies geschieht mit der RETURN-Taste. Diese Taste veranlasst den Rechner, den eingegebenen Befehl auszuführen. In diesem Beispiel wird der Name HEINRICH MEIER auf den Bildschirm ausgegeben. Der blinkende Cursor zeigt an, daß eine neue Eingabe gemacht werden kann. Der Text zwischen den " Zeichen wird als Zeichenkette bezeichnet.

Das Wort Computer bedeutet Rechnen, und so soll dieser nun eine Rechenaufgabe ausführen. Mit der Anweisung

```
PRINT 3 * 1.25
```

führt der Rechner die Rechenaufgabe 3 * 1.25 nach Betätigen der RETURN-Taste aus und gibt das Ergebnis auf den Bildschirm aus.

Was muß man machen, wenn man sich bei der Eingabe vertippt. Anstatt PRINT wurde PRUNT eingegeben. Bei allen Rechnern kann der Cursor durch eine Taste nach links und nach rechts, meistens auch nach oben oder unten bewegt werden. Welche Tasten dies sind, muß in den einzelnen Handbüchern nachgeschlagen werden.

Wird der Cursor nach links bewegt, so gibt es zwei Möglichkeiten. Der Buchstabe unter dem Cursor wird gelöscht oder nicht gelöscht. Im ersten Fall wird man bei der Fehleingabe PRUNT die Buchstaben bis zum R Löschen und denn die Buchstaben INT eingeben. Im zweiten Fall wird der Cursor über das U gesetzt und dan ein I eingegeben. Danach wird der Cursor wieder nach rechts bis zum Ende der bisherigen Eingabe bewegt.

Die beiden Anweisungen werden vom Rechner sofort, direkt ausgeführt. BASIC kann zwei Arten der Befehlsausführung, die direkte Bearbeitung einer Anweisung oder die Bearbeitung von Anweisungen in einem Programm. Wird eine Zeile ohne Zeilennummer, wie z. B.

```
PRINT 3 + 4
```

eingegeben, so wird nach Beendigung der Eingabe durch RETURN, der Befehl ausgeführt. Das Ergebnis 7 wird auf dem Bildschirm ausgegeben. Wird dagegen die Zeile als

```
10 PRINT 3 + 4
```

eingegeben, so wird diese Anweisung nach der Eingabe von RETURN, als Programmzeile 10 im Rechner gespeichert.

Die Sprache BASIC ist ein INTERPRETER. Ein Interpreter liest

jeweils eine Programmzeile aus dem Programmspeicher, entschlüsselt diese und führt die Anweisungen aus. Ist diese Zeile fertig bearbeitet, so wird die Zeile mit der nächst höheren Zeilennummer geholt und ausgeführt. Es gibt aber besondere Anweisungen, die den Interpreter veranlassen, an eine andere Zeilennummer zu springen. Davon später.

Ein erstes Programm:

Ein Vorteil des Rechners ist, eine gestellte Aufgabe immer wieder zu wiederholen. Als Beispiel soll ein kleines Programm geschrieben werden, das den Namen HEINRICH MEIER 10 mal auf den Bildschirm ausgibt.

Man könnte natürlich ein Programm schreiben, das folgendermaßen aussieht:

```
 10 PRINT "HEINRICH MEIER"
 20 PRINT "HEINRICH MEIER"
  .
  .
  .
  .
  .
  .
  .
100 PRINT "HEINRICH MEIER"
110 END
```

Das stellt eine erhebliche Schreibarbeit dar. Einfacher geht es mit der

FOR . . . NEXT

Schleife.

Das Programm lautet dann

```
10  FOR I = 1 TO 10 STEP 1
20 PRINT "HEINRICH MEIER"
30 NEXT I
40 END
```

In Zeile 10 ist eine Variable, eine veränderliche Größe, I vereinbart.

Diese wird in der FOR . . . NEXT Schleife auch als Laufvariable bezeichnet. Der erste Wert von I ist Eins. Hat der Interpreter die Zeile 10 entschlüsselt, so wird Zeile 20 und danach Zeile 30 ausgeführt. In Zeile 30 wird die Variable I durch die Angabe STEP 1 um Eins erhöht. Nun prüft der Interpreter, ob der Wert von I größer als der Endwert 10 ist. Wenn nicht, dann wird mit I = 2 die Schleife nochmals durchlaufen. Des geschieht solange, bis in Zeile 30 der Wert von I größer als 10 ist. Das Programm hält dann in Zeile 40, wobei I den Wert 11 hat.

Zu diesem Programm müssen aber noch weitere Bemerkungen gemacht werden.

In allen BASIC-Versionen kann die Angabe STEP 1 weggelassen werden, wenn die Laufvariable immer um Eins erhöht wird.

Eine Schleife wird immer mindestens einmal durchlaufen. Wenn die Zeile 10 aus Versehen so geschrieben wurde

```
10 FOR I = 10 TO 1 STEP 1
```

so wird erst in Zeile 30 festgestellt, daß I größer als Eins ist. Die Zeile 20 wird also einmal durchlaufen.

Ein erster Programmlauf:

Ist ein Programm geschrieben und zeilenweise im Rechner gespeichert, so kann es gestartet werden. Die Anweisung dazu lautet RUN. Diese Anweisung wird direkt eingegeben. Das Programm beginnt zu laufen und der in Zeile 20 programmierte Name wird auf den Bildschirm ausgegeben. Ist der Rechner wieder zu einer Eingabe bereit, so wird durch

```
PRINT I
```

der Wert der Variablen I ausgegeben. In diesem Beispiel muß I den Wert 11 haben.

Ein nächstes Beispiel:

Es soll ein Programm geschrieben werden, das die Zahlen von Eins bis

Zehn zusammenzählt. Vor der Eingabe des neuen Programms wird mit

NEW

das alte Programm gelöscht.

Das Programm lautet:

```
10 S=0
20 FOR I=1 TO 10
30 S=S+I
40 NEXT I
50 PRINT S
60 END
```

S ist eine Variable, die zu Beginn des Programms den Wert Null hat. Diese Zeile kann in den meisten Fällen entfallen, da die Anweisung RUN alle Variablen zu Null macht.

In Zeile 20 beginnt eine Schleife, die in Zeile 40 endet. In dieser Schleife wird die Anweisung

30 S = S + I

ausgeführt. Diese Zeile scheint mathematisch falsch zu sein, den S kann niemals gleich S plus einer Zahl I sein. Das Gleichheitszeichen wird in BASIC in zwei verschiedenen Bedeutungen behandelt. In diesem Fall bedeutet es nicht "gleich" sondern "ersetzt durch". In Worten lautet Zeile 30: Der neue Wert der Variablen S wird ersetzt (ist gleich) dem alten Wert von S, plus der Zahl I. Beim ersten Durchlauf durch die Schleife hat S den Wert Null und I den Wert Eins. Nach Ausführung der Anweisung S = S + I hat S den Wert Eins.

Beim zweiten Schleifendurchlauf ist S gleich Eins und I gleich Zwei. Das neue S wird somit 3.

Nach dem Ende der Schleife wird das Ergebnis S = 55 ausgedruckt.

Anmerkung für den ZX81 und den SPECTRUM

Bei beiden Rechnern muß die Zeile 10 vorhanden sein. Der Interpreter dieser BASIC-Version überprüft in Zeile 30, ob der Variablen S auf der rechten Seite ein Wert zugewiesen wurde. Ohne Zeile 10 ist dies nicht der Fall und der Interpreter beendet das Programm in Zeile 30 mit einer Fehlermeldung.

Andere BASIC-Versionen machen diese Überprüfung nicht, was zu katastrophalen Fehlberechnungen führen kann.

Frage:

Welche Anweisung muß wo eingesetzt werden, damit bei jeden Schleifendurchlauf der neue Wert von S ausgdruckt wird.

Antwort:

```
35 PRINT S
```

Die Beantwortung dieser Frage zeigt auch, warum Zeilennummern meistens in Abständen von Zehn eingegeben werden. Dadurch ist Platz für weitere Anweisungen vorhanden. Beim Schreiben eines Programmes wird die Zeile an die richtige Stelle eingefügt.

Mit

```
LIST
```

wird ein Programm auf den Bildschirm ausgegeben. Dieses Einfügen einer Zeile kann leicht beobachtet werden, wenn das Beispielprogramm mit LIST auf den Bildschirm ausgegeben, danach die Zeile 35 eingefügt, und das Programm nochmals mit LIST ausgegeben wird.

Einzelne Zeilen können mit

```
LIST 10
```

bzw.

```
LIST 10,40 (Commodore-Rechner LIST 10—40)
```

angezeigt werden.

Soll eine Zeile gelöscht werden, so wird die Zeilennummer ohne eine Anweisung eingegeben.

Die Eingabe

```
35
```

gefolgt von einen RETURN löscht Zeile 35 wieder.

Eine schönere Ausgabe:

In dem Beispielprogramm soll jeweils die Laufvariable I und die Summe S ausgegeben werden. Eine Möglichkeit ist

```
35 PRINT I,S
```

Diese Anweisung bewirkt, daß der Wert von I und der Wert von S in einer Zeile ausgegeben wird. Durch das Komma werden zwischen beide Zahlen Leerzeichen eingefügt. Die Anzahl der Leerzeichen ist von Rechner zu Rechner verschieden.

Werden in einem Programm viele Werte von verschiedenen Variablen ausgegeben, so kann der Name der Variablen mit ausgegeben werden.

Zeile 35 lautet dann:

```
35 PRINT "I=";I;"  S=";S
```

Hier wird zuerst der Text I = und der Zwischenraum der Wert der Variablen I ausgedruckt. Der ; gibt an, daß kein Zwischenraum bei der Ausgabe eingefügt wird. Nach der Ausgabe des Wertes vor S folgt keine weitere Angabe (; oder ,) so daß dann eine neue Zeile angefangen wird. Dies wird auch durch eine leere PRINT-Anweisung (ohne Angabe eines Variablennamens) erreicht.

Noch mehr Schleifen:

Das folgende Programm zeichnet ein Dreieck aus *.

```
10 FOR I=1 TO 10
```

```
20 FOR J=1 TO I
30 PRINT "*";
40 NEXT J
50 PRINT
60 NEXT I
70 END
```

Hier werden zwei geschachtelte Schleifen mit den Laufvariablen I und J verwendet. Die Schleife J geht von Zeile 20 bis 40 und druckt I mal ein Sternchen aus. Die obere Grenze der inneren Schleife ist die Laufvariable I der äußeren Schleife.

Nach dem Programmstart ist I = 1 und J = 1. Nun wird die innere Schleife von J = 1 bis J = 1, also einmal durchlaufen. Danach wird in Zeile 50 eine neue Zeile auf dem Bildschirm aufgefangen. In Zeile 60 wird I = 2. Die innere Schleife wird nun von J = 1 bis J = 2, also zweimal, durchlaufen. Dies geschieht solange, bis I größer als 10 ist. Dann wird das Programm beendet.

Bei geschachtelten Schleifen muß darauf geachtet werden, daß sich die Schleifen nicht überschneiden. Dies wäre der Fall, wenn das Programm folgendermaßen falsch geschrieben wäre:

```
10 ..........
20
30
40 NEXT I
50
60 NEXT J
70
```

Die Schleifen I und J überschreiben sich.

Frage:
Was wird ausgegeben ?

Antwort:
10 Sternchen und die Fehlermeldung:
NEXT WITHOUT FOR ERROR IN 60.

Bei Schleifen muß auch noch auf etwas anderes geachtet werden. Innerhalb einer Schleife darf eine Laufvariable nicht verändert werden. Das folgende Programm zeigt dies:

```
10 FOR I=1 TO 10
20 I=I+I
30 PRINT I
40 NEXT I
50 END
```

In diesem Programm nimmt I nicht die Werte 1,2,3 usw. bis 10 an, sondern die Werte 2,6 und 14. Noch schlimmer ist es, wenn Zeile 20 in

```
20 I = I—I
```

geändert wird. Die obere Grenze 10 wird nie erreicht und das Programm ist "im Wald" und kann nur durch RESET wieder angehalten werden.

Eine kleine Spielerei:

Es soll ein kleines Programm geschrieben werden, bei welchen der Computer nach dem Namen des Benutzers fragt, und diesen dann mit "GUTEN TAG" gefolgt vor Namen begrüsst.

Das Programm lautet:

```
10 INPUT"DEIN NAME  ";A$
20 PRINT:PRINT
30 PRINT"GUTEN TAG ";A$
40 END
```

In Zeile 10 wird die INPUT-Anweisung verwendet. Damit wird der Wert einer Variablen über die Tastatur eingegeben. Bei den meisten BASIC-Versionen kann nach INPUT ein Text angeführt werden, der beim Programmlauf auf dem Bildschirm ausgegeben wird. Nach dem Text folgt nach einem Strichpunkt der Name der Variablen. Kann nach INPUT kein Text angegeben werden, so kann dies durch

```
PRINT "DEIN NAME:":INPUT A$
```

umgangen werden.

In Zeile 20 folgen zwei PRINT Anweisungen. Werden mehrere Anweisungen in eine Zeile geschrieben, so werden diese durch : getrennt.

In Zeile 30 wird nun der Text GUTEN TAG und der Inhalt der Textvariablen (String-Variable) A$ ausgegeben.

Entscheidungen und Sprünge:

In BASIC gibt es zwei Anweisungen, ein Programm zu verzweigen. Da ist einmal die bedingte Verzweigung mit IF . . . THEN und die unbedingte Verzweigung mit GOTO. Die letztere wird von vielen BASIC-Programmierern dazu verwendet, ungeniesbare Spaghetti-Programme zu erzeugen.

Das sind Progrmme, bei denen man zwar einen Anfang erkennen kann, der Programmablauf dann aber mit GOTO's irgendwohin verschwindet und nicht mehr erkennbar ist. Es tauchen hin und da wieder Anfänge und Enden von Programmteilen auf, aber ein übersichtlicher Programmaufbau ist nicht erkennbar. Trotzdem lassen sich mit GOTO übersichtliche Programme schreiben, wenn man gewisse Regeln einhält. Auf diese wird im weiteren Verlauf dieses Grundkurses noch eingegangen werden.

Als Beispiel für die bedingte Programmverzweigung soll ein kleines Zahlenratespiel programmiert werden. Meistens ist es so programmiert, daß der Computer eine Zahl sich ausdenkt, die vom Benutzer erraten werden soll. Das folgende Beispiel macht es umgekehrt. Der Benutzer denkt sich eine Zahl, die vom Computer erraten werden soll. Die Zahl darf zwischen Eins und 100 liegen. Das Programm sieht folgendermaßen aus:

```
10 PRINT"ICH RATE DEINE ZAHL":PRINT
20 J1=0:J2=100
25 N=INT((J1+J2)/2)
30 PRINT"IST DEINE ZAHL ";N;" ?"
40 PRINT"DEINE ANTWORT: ZU GROSS"
50 PRINT"               ZU KLEIN"
55 PRINT"               RICHTIG"
```

```
60 INPUT A$
70 IF A$="ZU GROSS" THEN J2=N:GOTO 100
80 IF A$="ZU KLEIN" THEN J1=N:GOTO 100
85 IF A$="RICHTIG" THEN END
90 GOTO 40
100 IF ABS(J1-J2)<>2  THEN GOTO 25
110 PRINT"DEINE ZAHL IST ";INT((J1+J2)/2)
120 END
```

Als Lösungsverfahren wird das klassische binäre Suchen verwendet. Zuerst wird das Intervall halbiert. Ist die vom Computer zu ratende Zahl größer als 50 wird "ZU KLEIN" andernfalls "ZU GROSS" oder "RICHTIG" eingegeben.

Wenn die vom Computer ausgegebene Zahl zu klein war, dann wird die obere Hälfte halbiert und ausgegeben. Dieses Verfahren wird solange fortgesetzt, bis die richtige Zahl gefunden oder das Intervall nur aus einer einzigen Zahl besteht. Dies ist dann die gesuchte Zahl.

Die erste bedingte Verzweigung ist in Zeile 70. Ist die Eingabe A$ = ZU GROSS, dann wird die obere Intervallgrenze J2 gleich N. In Zeile 25 wird das verbleibende Intervall halbiert, und die Zahl in der Mitte als neue Zahl ausgegeben. Bei einer bedingten Verzweigung mit IF werden nach THEN alle Anweisungen ausgeführt, die in der gleichen Zeile stehen. Die GOTO 100 Anweisung setzt das Programm in Zeile 100 fort. Dort ist wieder eine Abfrage mit IF programmiert. Diesmal wird untersucht, ob das Intervall nur noch eine Zahl enthält. Ist dies der Fall, sow wird diese Zahl ausgegeben und das Programm beendet. Sonst wird nach Zeile 25 zurückgesprungen.

Der Sprung in Zeile 90 wird dann ausgeführt, wenn bei der Eingabe von A$ nicht eines der drei vereinbarten Worte eingegeben wird.

Die in diesem Programm vorhandenen Sprungbefehle sind notwendig, da es in den meisten BASIC-Versionen nur eine Abfrage

WENN . . . DANN (IF . . . THEN)

gibt und kein Abfragen der Form

WENN . . . DANN (DIES) ODER (DAS) ENDE
(IF . . . THEN . . . ELSE . . . END)

Hier wird, wenn die Abfrage erfüllt ist, der Programmteil (DIES), und wenn die Abfrage nicht erfüllt ist, der Programmteil (DAS) ausgeführt.

In BASIC muß diese Verzweigung dann folgendermaßen programmiert werden.

```
100 IF (BEDINGUNG THEN (ANWEISUNGEN 1):GOTO 120
110 ANWEISUNGEN 2
120 PROGRAMM FORTSETZUNG
```

Wenn die Bedingung erfüllt ist, werden die ANWEISUNGEN 1 ausgeführt und dann nach Zeile 120 gesprungen. Ist die Bedingung nicht erfüllt, dann wird das Programm in Zeile 110 fortgesetzt und danach die Zeile 120 ausgeführt. In diesen Fällen sind die GOTO-Befehle notwendig und sinnvoll.

Das Programm verwendet zwei Funktionen. Dies sind die INT und die ABS-Funktion.
INT rundet immer auf die nächste kleinere Zahl ab.

```
PRINT INT(7.9/2) ergibt 3
PRINT INT(6/2)   ergibt auch 3
```

Anmerkung für ZX81 oder SPECTRUM

Die Anweisung PRINT 0.6*5 ergibt 3, dagegen ergibt
PRINT INT(0.6*5) 2.

Die Funktion ABS bilden den Absolutwert einer Zahl

PRINT ABS (–2) ergibt 2

Haupt- und Unterprogramme:

In einem BASIC-Programm ist der Programmablauf durch die Zeilennummern gegeben. Änderungen dieses Ablaufs werden durch Verzweigung mit dem GOTO-Befehl durchgeführt. Eine andere Art der Programmverzweigung ist ein Sprung in ein Unterprogramm. Anders als

beim GOTO-Befehl wird nach Beendigung des Unterprogramms an die Stelle im Hauptprogramm zurückgesprungen, an welcher dieses verlassen wurde.

```
 100 ...
 110 GOSUB 1000          Hauptprogramm
 120 ...

1000 ...
                         Unterprogramm
1900 RETURN
```

Im Beispiel wird in Zeile 110 in das Unterprogramm, das bei 1000 beginnt, gesprungen. Wird dort in Zeile 1900 die Anweisung RETURN gefunden, so wird im Hauptprogramm bei Zeile 120 das Programm fortgesetzt.

Wann werden Unterprogramme gebraucht ?

Unterprogramme werden immer dann verwendet, wenn an verschiedenen Stellen im Programm eine gleiche, festgelegte Folge von Anweisungen benötigt wird. Dazu das folgende Beispiel:

```
10 REM ROEMISCHE ZAHLEN
100 INPUT"Z=";Z:B=Z
120 IF B<=3000 THEN GOTO 500
130 PRINT "ZU GROSS":GOTO 100
500 F=1000:A$="M":GOSUB 1000
510 F=900:A$="CM":GOSUB 1000
520 F=500:A$="D":GOSUB 1000
530 F=400:A$="CD":GOSUB 1000
540 F=100:A$="C":GOSUB 1000
550 F=90:A$="XC":GOSUB 1000
560 F=50:A$="L":GOSUB 1000
570 F=40:A$="XL":GOSUB 1000
580 F=10:A$="X":GOSUB 1000
590 F=9:A$="IX":GOSUB 1000
600 F=5:A$="V":GOSUB 1000
610 F=4:A$="IV":GOSUB 1000
620 F=1:A$="I":GOSUB 1000
999 END
```

```
1000 B=INT(Z/F)
1010 IF B<>0 THEN GOSUB 1100
1020 Z=Z-B*F
1030 RETURN
1100 FOR I=1 TO B
1110 PRINT A$;
1120 NEXT I
1130 RETURN
```

Das Programm ROEM rechnet Dezimalzahlen kleiner gleich 3000 in roemische Zahlen um.

Die Zeilen 100 bis 130 enthalten die Eingabe. In den Zeilen 500 bis 620 sind die dezimalen Wertigkeiten und die entsprechenden Zahlensymbole vereinbart.

Das Unterprogramm ab Zeile 1000 bis 1030 berechnet eine Stelle des römischen Zahlensystems. Der Wert von B gibt an, wie oft das Zahlensymbol gedruckt werden muß. Wenn B ungleich Null ist, wird in ein weiteres Unterprogramm ab Zeile 1100 gesprungen. Die Schachtelung von Unterprogrammen ist möglich.

Zum Programmaufbau:

In diesem Beispiel wurde das Unterprogramme an das Ende des Programms nach dem Hauptprogramm gehängt (Ende des Hauptprogramms in Zeile 999).

Oft werden Unterprogramme vor dem eigentlichen Hauptprogramm aufgeführt. Dies hat in BASIC den folgenden Grund. Wird ein Unterprogramm aufgerufen, so fängt der Interpreter bei der kleinsten Zeilennummer an, nach der Zeilennummer des Unterprogramms zu suchen. Je früher er diese findet, desto schneller ist die Programmausführung.

Anmerkung für ZX81, SPECTRUM und ATARI

Bei diesen Rechnern sind Unterprogrammaufrufe mit Namen erlaubt.

Beispiel:

```
 10 MENU = 1000
100 GOSUB MENU
```

Durch diese Namensgebung kann ein Programm leserlicher gemacht werden.

Einige Anmerkungen zum Programmaufbau

Ein Programm sollte immer in einzelne Blöcke aufgeteilt sein. Für ein Inventur-Programm braucht man zum Beispiel folgende Teile:

1. Einen Menueteil
2. Eingabe von Artikeln
3. Ausgabe auf Drucker
4. Veränderung des Lagerbestandes

Der Programmaufbau kann dann folgendermaßen aussehen:

Zeilen	
10 – 99	Hinweis auf Autor, Programmart
100 – 199	Wertzuweisung an Variablen Dimensionierung von Feldern
200 – 999	Häufig gebrauchte Unterprogramme
1000 – 1999	Menueteil
2000 – 2999	Eingabe von Artikeln
3000 – 3999	Ausgabe auf Drucker oder Bildschirm
4000 – 4999	Verändern des Lagerbestandes
5000 – 5999	Unterprogramme, die Fehler bei der Bedingung des Programms überwachen.

Das Programm wird in Zeile 10 mit einem GOTO 1000, einem Sprung in Menue beginnen. Von dort werden die Programmteile über bedingte Verzweigungen als Unterprogramme angesprungen, und von dort wird wieder ins Menue zurückgesprungen.

Dieser modulartige Aufbau macht es leichter, ein Programm zu überschauen und auch zu testen.

Fehler:

Im Prinzip kann man davon ausgehen, daß kein Programm auf Anhieb fehlerfrei ist. Durch die Syntaxprüfung sind Schreibfehler schon bei der Eingabe gefunden worden. Logische Fehler werden erst im Programmlauf gefunden.

Einige der häufigsten Fehlerursachen sind die folgenden:

Oft wird vergessen, einer Variablen, die auf der rechten Seite einer Anweisung steht, einen Wert zuzuweisen. Nur beim ZX81 und beim SPECTRUM wird dieser Fehler erkannt. Bei allen anderen wird der Wert der Variablen als Null angenommen.

Rechenergebnisse sollten daher mit dem Taschenrechner einmal überprüft werden.

Der Variablenname I wird häufig als Laufvariable in FOR . . . NEXT-Schleifen verwendet. Manchmal vergisst man, daß man diesen Namen schon verwendet hat und programmiert eine neue Schleife mit dem gleichen Namen. Als Rechenergebnisse kommen dann reine Zufallszahlen auf.

Wenn man einen Variablennamen in einem Programm mehrfach benutzt, so muß man sicher sein, daß der augenblickliche Wert nicht später noch gebraucht wird.

Wenn ein Programm gestartet wurde, und es erscheint längere Zeit keine Ausgabe auf dem Bildschirm, so kann das Programm in einer unendlichen Scheife stecken. Das Programm kann dann durch die BREAK-Taste unterbrochen werden. Dann wird die Zeilennummer ausgegeben bei welcher das Programm unterbrochen wurde. In dieser Gegend muß man dann nach Möglichkeiten einer unendlichen Schleifenbildung suchen. Meistens ist es ein GOTO-Befehl mit einr falschen Zeilennummer, oder eine nicht erfüllte IF . . . THEN Bedingung wie im folgenden Beispiel:

```
10 I = 1
15 I = I + 2
20  IF I = 100 THEN  GOTO 50
30  GOTO 15
50  END
```

Die Bedingung I = 100 ist nie erfüllt, da I immer eine ungerade Zahl ist. Nach I = 99 wird I = 101. Damit stellt dies eine unendliche Schleife dar. Richtig wäre in Zeile 20 die Abfrage gewesen.

```
20 IF I > = 100 THEN GOTO 50
```

Damit wäre die Schleife mit I = 101 verlassen worden.

Meistens ist aber nicht klar ersichtlich, wo im Programm so eine Schleife auftritt. Dann wird man versuchen, durch PRINT-Anweisungen oder durch Einfügen des STOP-Befehls den Fehler einzukreisen.

BASIC ist eine interpretative Sprache. Das heißt, das eingegebene Programm oder Teile des Programms können sofort ausgeführt werden. Dies sollte man dazu benutzen, kleine Programmabschnitte (Module) sofort zu testen. Dies ist wesentlich einfacher, als ein vollständiges Programm auf Fehler zu untersuchen.

Leider führt dies aber auch dazu, daß Programme nur am Rechner entwickelt werden und nebenher keine Notizen zum Programmablauf gemacht werden. Bevor man mit dem Schreiben eines Programms beginnt, sollten die wesentlichen Grundzüge des Programmablaufs schriftlich festgehalten werden. Dies erleichtert wesentlich die Fehlersuche.

Auf eine weitere Fehlermöglichkeit, die Instabilität der angewendeten mathematischen Verfahren, soll hier nicht eingegangen werden.

Im Allgemeinen kann man sagen, daß die Fehlersuche mindestens ebenso lange dauert, wie das Schreiben des Programms.

Schluß:

Dieser Grundkurs sollte nur die notwendigsten Befehle eines BASIC-Programms erläutern. Ein Computer-Neuling sollte zu Beginn seiner Programmiererfahrung kurze und einfache Programme abtippen und daraus lernen. Das Abtippen langer Programme führt immer zur Frustration, da diese auf Anhieb nicht laufen und Fehler nicht gefunden werden. Selbst für einen versierten Programmierer ist es nicht leicht, einen Fehler in einem fremden Programm zu finden. Statt dem sturen Abtippen gibt es noch eine andere, bessere Methode. Soll ein längeres Programm in einen Rechner eingegeben werden, so sollte das Programm vorher untersucht werden, welche Teile es enthält. Dies geht meistens aus der Aufgabenstellung hervor. Dann sollten nur diese

Teile programmiert werden, wobei das ursprüngliche Programm als Vorlage dienen kann. Sobald sich eine Möglichkeit zeigt, einen Test eines Programmteils durchzuführen, dann sollte dies auch gemacht werden. Fehler sind in kleinen Programmteilen leichter zu finden, als in einem großen Programm. Fehler lassen sich leichter in einem ausgedruckten Protokoll als auf dem Bildschirm finden.

Bei abgetippten Programmen kann man bei der Fehlersuche auch so vorgehen, daß eine Person das Programm laut liest und eine andere Person dies auf dem Bildschirm vergleicht. So können auch lange Programme zum Laufen gebracht werden. Besser ist es allerdings, zu Versuchen, die Teile eines Programms zu verstehen um sie dann selbst zu programmieren.

Wie lernt man BASIC?

Wie lernt man BASIC ?

1. Allgemeines
Die Programmiersprache BASIC gehört heute zu den leichtesten höheren Programmiersprachen. Sie wurde ursprünglich im Jahre 1960 in einem amerikanischen College entwickelt und seit dieser Zeit haben sich unzählige Einzelversionen entwickelt. Diese einzelnen BASIC-Sprachen unterscheiden sich meist nur durch die Anzahl der Funktionen und in gewissen Besonderheiten der Computerhersteller. (DEC BASIC, HP-BASIC, WANG-BASIC, ALTAIR-BASIC, COMMODORE-BASIC, TANDY LEVEL I und LEVEL II BASIC, MICROSOFT BASIC u. v. a.)
Das genormte Standard BASIC (ANSI-BASIC) wird man jedoch in relativ wenig Systemen heute finden. Jeder Microcomputer hat seine besonderen Vorteile und auch diese werden meist durch geschickt gewählte BASIC-Befehle genutzt.

Für eine Einführung in die BASIC-Programmiersprache ist es jedoch nur von zweitrangiger Bedeutung, welches BASIC man benutzt. Wichtig ist nur, man lernt irgendein BASIC. Dann hat man sicher die Voraussetzungen, eine andere Version in kürzester Zeit zu durchschauen und zu verstehen.

BASIC besteht nur aus einer kleinen Anzahl von Befehlen. Sie sind meist Abkürzungen englischer Worte, die man leicht im Gedächtnis behalten kann.

Wie viele andere Programmiersprachen, kann man auch BASIC in zwei große Bereiche aufteilen.

1. Einfache Befehle für einfache Operationen
2. Hochentwickelte Befehle für komplizierte Operationen

Wir wollen uns hier nur mit Punkt 1 befassen. Der Vorteil liegt dabei auch darin, daß die einfachen, gebräuchlichen Befehle auch in allen BASIC-Versionen vorhanden sind. Beachten Sie deshalb dies bei Ihren Programmier-Experimenten. Je elementarer die Befehle beim Programmaufbau gewählt werden, um so größer ist die Anzahl der Versionen, die es verarbeiten kann. Die Manuals der Computerhersteller sollten auf jeden Fall benutzt werden. Und hier wären wir schon beim nächsten Punkt. Es ist unerlässlich,

daß man zum Erlernen von BASIC ein System zur Verfügung hat. Durch Lesen alleine dürfte es sehr schwer sein, sich einzuarbeiten. Man muß einfach einmal Programme eingeben und die Ergebnisse kontrollieren können.

Dies gilt übrigens für die gesamte Microcomputertechnik. Auch geht in den meisten Fällen Probieren übers Studieren. Diesen Bedürfnissen sind einige Hersteller ja heute wirklich entgegengekommen. TRS-80 TANDY und COMMODORE, um nur einmal die bekanntesten zu nennen. In ein paar Jahren werden es sicher mehr sein und die Preise werden auch anders aussehen.

2. Was kann man mit BASIC anfangen?

Sie können einfache mathematische Rechnungen durchführen (wie mit einem wissenschaftlichen Taschenrechner). Es ist aber auch möglich, kleine Programme zu entwickeln, die einem die Alltagsarbeit erleichtern. (Haushaltsfinanzen, Scheckbuchkontrolle, Heim- und Hobbyanwendungen). Viele BASIC-Versionen verfügen heute über eine phantastische Programmierbarkeit des Bildschirms. Sie können so Spiele und Graphik-Programme erstellen. (Computer-Kunst) Andere BASIC-Versionen erlauben eine vielseitige Ein-Ausgabe-Programmierung. Hier können sie

Steuerungen, Zeitschalter, Analog/Digitalwandler, Digital/Analogwandler, Spracherkennung, künstliche Spracherzeugung usw. programmieren.
Der Phantasie sind eigentlich keine Grenzen mehr gesetzt. „Fast" alles ist machbar.

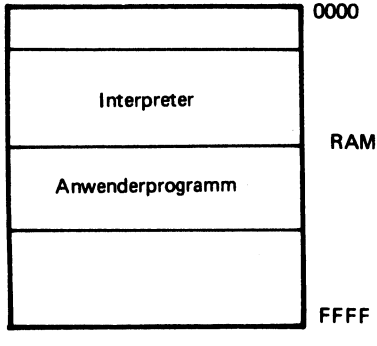
3. Wir fangen nun an zu programmieren

Wir nehmen an, Sie haben sich einen kleinen Computer (Heimcomputer etc.) besorgt und sitzen nun vor Ihrem Gerät. Nach dem Einschalten meldet sich der BASIC-Interpreter bei vielen Systemen bereits mit einem bestimmten Zeichen. Dieses Zeichen sagt uns meist, daß der Computer bereit ist, einen Befehl, Anweisungen oder ein Kommando von uns entgegenzunehmen.

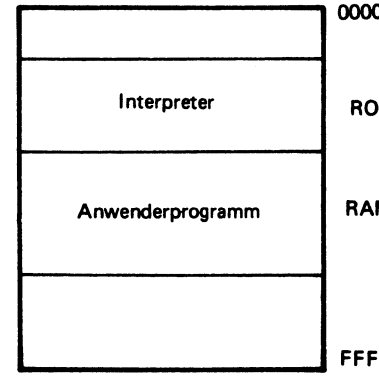
Andere Rechner wieder müssen erst gestartet werden, oder der BASIC-Interpreter muß erst von einem Massenspeicher (Cassette oder Floppy) in den Arbeitsspeicher geladen und anschließend aufgerufen werden. Die modernen Compact-Computer (wie Commodore-PET und TANDY TRS-80) haben ein residentes BASIC, d. h. es ist in Festwertspeichern (ROM) gespeichert und ist sofort nach Einschalten der Versorgungsspannung für Befehle aufnahmebereit. Da der BASIC-Interpreter resident ist, ergeben sich folgende Vorteile:

1. Keine Fehler beim Laden
2. Es wird kein Arbeitsspeicherbereich durch den Interpreter belegt. (ROM sind billiger als RAM)
3. Der gesamte Arbeitsspeicher (RAM) steht für meine Anwenderprogramme zur Verfügung.

1. Möglichkeit
BASIC wird geladen



2. Möglichkeit
BASIC in ROM



Grundsätzlich unterscheiden wir bei BASIC zwischen fünf verschiedenen Gruppen von Anweisungen und Befehlen.

1. Systemkommandos. Diese können (meist) nicht programmiert werden. Beispiele: RUN, LIST etc.

2. BASIC-Befehle (PRINT, LET, READ, GOTO usw.

3. Anweisungen (LOAD, SAVE usw.)

4. Funktionen. Arithmetische und sonstige Funktionen (SINUS, COSINUS, RND usw.)

5. Operatoren (arithmetische Operatoren)

Die wichtigsten Systemkommandos sind fast bei allen BASIC-Versionen gleich:

NEW: Eingabe eines neuen Programmes, alles wird aus dem Arbeitsspeicher gelöscht.

SCR: Gleiche Bedeutung wie NEW

LIST: Das im Speicher vorhandene Programm wird auf dem Bildschirm etc. ausgedruckt.

RUN: Übersetzung (compiling) des im Speicher befindlichen Programmes und Starten des Programmes.

Manche BASIC-Versionen erlauben die Möglichkeit, an einer bestimmten Zeile zu starten, z. B. RUN 200.

Die Befehle werden wir im folgenden genau besprechen.

Die bekanntesten Anweisungen sind:

SAVE: Abspeichern eines Programmes auf einen Massenspeicher (Cassette oder Diskette)

STORE: wie SAVE

LOAD: Laden von einem Massenspeicher her

VERIFY: Kontrolle des abgespeicherten Programmes

BYE: Verlassen des BASIC-Modes und Rückkehr in einen Monitor.

Die Anzahl der Funktionen in einem BASIC-Interpreter hängt meist von seiner Größe ab. Es gibt heute BASIC-Versionen von 2K, 4K, 8K, 12K bis hin zu 16K-Versionen. Dementsprechend ist meist die Anzahl der Funktionen. Die RND-Funktion ist meist auch in den kleinen Versionen zu finden, da sie sich besonders zum Programmieren von Spielen eignet. Die RND-Funktion erzeugt in BASIC eine Reihe von Zufallszahlen. In welchem Bereich, wird meist in einem Argument, welches unmittelbar der Funktion folgt, festgelegt.

z. B. RND (1)

Weitere Funktionen sind:

SIN (X)	Sinus Y
SGN (X)	Sinus X
ABS (X)	Absolutwert von X
TAB (X)	Ausdrucken von Reihen
INT (X)	Führt zu ganzen Zahlen
CHR$ (X)	Zeichenaufruf

u. v. a.

Operatoren (arithmetische Operatoren)

Arithmetische Berechnungen in einer höheren Programmiersprache, wie z. b. BASIC werden wie in der gewöhnlichen Arithmetik als Ausdrücke pro Zeile behandelt. Mit allen verfügbaren Operatoren, wie z. b. (+; –;*;/; ↑; etc.) können Formeln zusammengestellt werden. Die mathematischen Regeln werden dabei vom Computer automatisch beachtet.

a) Ausdrücke in Klammern werden zuerst ausgerechnet und dann später in weiteren Rechenoperationen berücksichtigt.

b) Zwei Operatoren können meist nicht aufeinanderfolgen.

c) Bei Berechnungen ohne Klammern heißt es auch hier Punktrechnung geht vor Strichrechnung, Potenzierung entsprechend noch vor der Punktrechnung.

d) Sind in einem arithmetischen Ausdruck keine Klammern enthalten, erfolgt die Berechnung von links nach rechts. (Punkt u. Strichrechnung werden natürlich berücksichtigt.

Beispiele:

(A + (3 * B)) ** 2	=	Das Produkt aus 3 * B wird zu A hinzuaddiert und quadriert.
A – B – C	=	B wird von A abgezogen und C wird vom Ergebnis der ersten Operation abgezogen.

Zum Vergleich zweier oder mehrer Größen gibt es in den meisten BASIC-Versionen auch vergleichbare Operatoren, wie z. B. die größer ⟩ und die ⟨ Kleiner-Zeichen. Sie werden oft in Verbindung mit IF-Befehlen verwendet.

= Gleichheitszeichen,	Beispiel A = B, bedeutet A gleich B,
⟨ kleiner Zeichen,	Beispiel A ⟨ B, bedeutet A kleiner B
⟩ größer Zeichen,	Beispiel A ⟩ B, bedeutet A größer B
⟨ = kleiner gleich,	Beispiel A ⟨ = B, bedeutet A kleiner gleich B

) = größer gleich,	Beispiel A) = B, bedeutet A größer gleich B
( ) nicht gleich,	A ( ) B; A nicht gleich B

Zeileneinheiten

In jedem BASIC-Programm muß jeder Befehl mit einer Zeilenangabe versehen werden. Die Anzahl der möglichen Zeilenzahlen hängt von der Arbeitsspeichergröße ab. Meist werden die Zeilenzahlen von 10 an aufwärts in Schritten von 5 oder 10 gewählt. Man hat dann später beim Korregieren noch Platz, einen Befehl irgendwo einzuschieben. Neue Zeilen werden vom BASIC-Programm genau dort ins Programm eingesetzt, wo sie hingehören.

Einfache BASIC-Befehle

Spätestens zu diesem Zeitpunkt sollten wir die Möglichkeit haben, einen BASIC-Computer zu benützen. Es ist von großer Wichtigkeit, daß man zum Erlernen einer Programmiersprache ein System zur Verfügung hat. Heute gibt es bereits BASIC-Computer zu recht günstigen Preisen, so daß auch der Amateur in der Lage ist, sich ein eigenes System anzuschaffen.

Wir wollen diesen BASIC-Kurs auf dem Commodore Computer PET 2001 durchführen und auch dessen BASIC-Version beschreiben. Sie können jedoch die meisten Befehle und Programme aus dieser Version auch auf andere Computersysteme übertragen.

Wir schalten nun unser Gerät ein und auf dem Bildschirm erscheint eine Angabe über die BASIC-Version, den maximal verfügbaren Speicherbereich, das Wort READY und ein blinkender Cursor.

Das Wort READY und der blinkende Cursor sagen uns, daß der Computer jetzt eine Eingabe über die Tastatur erwartet.

Das wollen wir jetzt auch tun und tippen ganz einfach einmal unseren Namen ein. Was wird der Computer jetzt tun?

Eingabe: OSWALD (Return-Taste drücken)

Es erscheint die Meldung „SYNTAX-ERROR". Dies bedeutet, daß der Computer uns nicht verstanden hat. Wir müssen also jetzt Worte verwenden, die der Computer versteht. Ganz zu Anfang hatten wir ja schon einmal einige BASIC-Befehle angedeutet.
Auch der Befehl PRINT wurde dort schon genannt. PRINT bedeutet „ Ausdrucken" und sagt dem Computer, daß er einen Text auf dem Bildschirm ausgeben soll. Der Text muß hinter dem Befehl in Anführungszeichen gesetzt werden.

Beispiel:
Geben Sie folgendes ein:

20 PRINT „ BASIC IST EINFACH" (Return)

Vergessen Sie nicht, nach jeder Eingabe die RETURN-Taste zu drücken, da der Computer die Eingabe erst nach Drücken dieser Taste in seinen Arbeitsspeicher aufgenommen hat.

Jetzt haben wir schon unser erstes Computer-Programm geschrieben und es steht im Speicher. Es ist zwar ein sehr kurzes und einfaches Programm, da es nur aus einer Zeile besteht, aber es ist ein Programm.

Jetzt müssen wir unser Programm auch noch starten. In BASIC geschieht dies u. a. durch die Eingabe des Kommandos RUN. Beachten Sie, daß wir zur Eingabe eines BASIC-Kommandos keine Zeilenzahl benötigen.

Also geben wir jetzt RUN ein.

Der Computer antwortet uns:

BASIC IST EINFACH

und sagt uns mit einem READY-Zeichen und dem blinkenden Cursor, daß das Programm ausgeführt wurde und er zur Aufnahme neuer Befehle und Kommandos bereit ist.

Durch die Eingabe des „RUN" Kommandos wird der Computer aufgefordert, an der niederwertigsten Zeilenzahl mit der Abarbeitung des Programmes zu beginnen. Er ging also zu Zeile 10 und hat den Befehl PRINT ausgeführt. Der Befehl PRINT „ BASIC IST EINFACH" sagte ihm, daß er den Text in Anführungszeichen auf dem Bildschirm ausdrucken soll. Man bezeichnet diesen Text zwischen zwei Anführungszeichen auch als „ String". Der PET-Com-

Unser kleines Programm bleibt jetzt im Speicher, bis wir unseren Computer ausschalten oder ihm durch ein Kommando sagen, daß diese Befehlszeile gelöscht werden muß. Der PET hat hier das Kommando NEW. Andere Computer haben SCR oder CLEAR etc. Wir geben nun NEW mit anschließendem Drücken der RETURN-Taste ein und probieren ein neues Programm.

puter hat jetzt noch einige kleine Vorteile. Man kann auch RUN 10 eingeben, um das Programm an einer von uns gewünschten Zeilenzahl zu starten. Weiterhin kann man sich beim Schreiben der Zeile 10 das zweite Anführungszeichen sparen. Bedingt ist jedoch, daß kein weiterer Text mehr nach dem letzten Wort in der gleichen Zeile folgt.

```
NEW
10 PRINT „ BASIC"
20 PRINT „ IST DOCH"
30 PRINT „ GANZ "
40 PRINT „ EINFACH"
RUN
BASIC
IST DOCH
GANZ
EINFACH
```

Der Computer arbeitet jetzt Zeile pro Zeile ab und druckt die Strings zwischen den Anführungszeichen aus. Machen Sie jetzt einige Versuche und geben Sie ganze Sätze oder Graphik-Symbole als Strings ein.

Wie lernt man BASIC,

Teil II

Wie lernt man BASIC? Teil II

Rückblick:
Im ersten Teil unseres kleinen BASIC-Kurses haben wir den PRINT-Befehl kennengelernt. Wir haben ein kleines Programm geschrieben, welches uns nach Eingabe von RUN einen kleinen Text auf dem Bildschirm ausdruckt.
Wollen wir uns jetzt unser Programm noch einmal ansehen, geben wir einfach LIST (Return-Taste drücken) ein. Der Computer druckt uns jetzt unser Programm noch einmal so aus, wie wir es eingegeben haben. Sie können jetzt Änderungen vornehmen, z. B. neue Zeilen eingeben oder auch vorhandene Zeilen löschen.

> Neu Eingeben geschieht einfach durch Schreiben einer neuen Zeile mit dem neuen Befehl oder der neuen Anweisung.

```
LIST
10 PRINT „ BASIC"
20 PRINT „ IST DOCH"
30 PRINT „ GANZ"
40 PRINT „ EINFACH"
READY
```

Nehmen wir einmal an, wir wollten jetzt unseren auszudruckenden Text ändern. Alles, was man nun tun muß, ist einfach die gewünschte Zeile neu eingeben.

```
30 PRINT „ SEHR"
LIST (Return)
```

Jetzt wird alles noch einmal ausgedruckt.

```
10 PRINT „ BASIC"
20 PRINT „ IST DOCH"
30 PRINT „ SEHR"
40 PRINT „EINFACH"
READY
```

Der PET bietet für solche Änderungen eine ganz praktische Methode.
Man braucht hier nicht die ganze Zeile noch einmal zu schreiben, sondern geht mit dem Cursor an die zu ändernde Stelle.
Mit den beiden Cursor-Tasten und der Shift-Taste kann der Cursor auf dem Bildschirm an jede beliebige Stelle gebracht werden. Von dieser Stelle aus kann ich dann den zu ändernden Text einfach überschreiben. Durch Drücken der SPACE-Taste wird ein vorhandenes Zeichen gelöscht. Mit der Insert-Funktion können auch Zeichen eingefügt werden. Wichtig ist, daß nach einer vorgenommenen Änderung in einer Zeile die RETURN-Taste gedrückt wird.

Nun wollen wir ein anderes kleines Programm mit PRINT-Befehlen schreiben.

```
10 PRINT 10 + 10
20 PRINT 5 + 5
30 PRINT 10 – 10

RUN

20
10
0
```

Wollen wir die Ergebnisse nebeneinander haben, geben wir folgendes ein:

```
NEW   (Sorgt dafür, daß der Speicher gelöscht
       wird)
READY
10 PRINT 10 + 10, 5 + 5, 10 – 10
RUN
20      10      0
```

Wir sehen hier, daß wir eine Verteilung der Ergebnisse auf einer bzw. zwei Zeilen vornehmen können.
Es können auch mehrere Operationen in eine Zeile eingegeben werden.

Übungen:

1. Schreiben Sie ein BASIC-Programm, welches Ihren Namen und Ihre Adresse ausdruckt. Der Ausdruck soll in vier Zeilen erfolgen.
2. Was muß eingegeben werden, um den Speicher des Computers zu löschen?
3. Welches Kommando muß man verwenden, um ein Programm aus dem Speicher ausschreiben zu lassen.

Die LET-Anweisung

Die LET-Anweisung wird dazu benutzt,

a) einen Wert (Zahl)
b) ein Ergebnis einer Berechnung

einer bestimmten Größe (Variable) oder auch mehreren Variablen zuzuordnen. Grundsätzlich sieht eine LET- oder auch Zuordnungsanweisung wie folgt aus:

100 LET (Variable) = (Zahl od. Ergebnis einer Berechnung)

In einigen BASIC-Versionen kann das Wort LET auch der Einfachheit wegen weggelassen werden.

100 LET Y = 2

Man kann also auch schreiben 100 Y = 2. Auch können in manchen Versionen mehrere Variable demselben Wert zugeordnet werden.

100 LET Y = X2 = W = 3

Auch können die Variablen auf beiden Seiten des Gleichheitszeichens geschrieben werden. Die Berechnung erfolgt dann so, daß Daten vor der Rechnung der Variablen links neben dem Gleichheitszeichen und Daten nach der Berechnung den Variablen rechts neben dem Gleichheitszeichen zugeordnet werden.

100 LET X = X + 1

nimmt den Anfangswert von X, addiert eines dazu und weist das Ergebnis als neuen Wert von X aus.

Die LET-Anweisung ist keine algebraische Gleichung. Sie ist lediglich ein Befehl, eine Berechnung durchzuführen und das Ergebnis einer vorher festgelegten Variablen zuzuordnen.

PROGRAMMBEISPIEL:

```
10 REM  DIESES PROGRAMM ZEIGT DIE
20 REM  ANWENDUNG DES LET-BEFEHLS
30 PRINT „ EINGABE EINER POSITIVEN,"
35 PRINT „GANZEN ZAHL"
40 INPUT X
60 LET   Y = X + 1
70 LET   A = LOG (X)
80 LET   B = SIN (X)
90 PRINT X, Y, A, B
100 END
```

Wir haben hier bei diesem Beispielprogramm jetzt drei neue Befehle kennengelernt, die man sehr oft beim Programmieren in BASIC benötigt.

REM, INPUT und END

Wir wollen diese Befehle deshalb jetzt gleich besprechen. Die Anweisung REM dient zur Dokumentation des Programmes. Alles, was hinter dem Befehl REM folgt, wird nur als Text im Programm angesehen und wirkt sich nicht auf den Ablauf aus. Der Befehl END sagt dem Computer, daß an dieser Stelle das Programm zu Ende ist. END sollte dann die höchste Zeilenzahl im Programm haben.

Die INPUT-Anweisung (Eingabe)

Die Input-Anweisung wird dazu verwendet, numerische oder String-Daten während des Programmablaufes in den Computer zu geben. Der Befehl besteht aus der Anweisung INPUT mit nachfolgenden Variablen. Diese Variablen müssen durch ein Komma getrennt sein. Numerische Variable und String Variable können in einem Input-Befehl vorkommen. String Variable (Zeichenketten) erlauben die Eingabe beliebiger Zeichen wie Buchstaben, Zahlen, Satzzeichen und graphische Zeichen).

Beispiel:

```
100 INPUT „ WIE ALT BIST DU", X
                    String
RUN
```

Diese Anweisung druckt die Frage: Wie alt bist Du? in die folgende Zeile. Das Fragezeichen bedeutet hier in erster Linie, daß der Computer auf eine Eingabe wartet. Erst wenn er diese Eingabe vom Programmierer bekommen hat, wird er im Programm fortfahren. Folgendes muß bei Anwendung des INPUT-Befehles beachtet werden:

1. Die einzugebenden Daten müssen zu den Variablen im INPUT-Befehl gehören.

2. Mehrere Daten müssen durch ein Komma getrennt werden.

3. Ein String sollte immer in Anführungszeichen gesetzt werden.

Treten große Mengen von Daten auf, sollten anstelle des Input-Befehls die Anweisungen READ und DATA benutzt werden.

PROGRAMMIERBEISPIEL:

```
10   REM   Dieses Programm zeigt Ihnen,
20   REM   wie der INPUT-Befehl arbeitet
30   INPUT X
40   LET   R = (X/3.14159) ↑ 0.5
50   PRINT „RADIUS ="; R
60   INPUT B, H
70   LET   A = (B*H) 12
80   PRINT „FLAECHE ="; A
90   INPUT K, J, L, M, N, O, P
100  LET   S = K + J + L + M + N + O + P
110  PRINT „ DIE SUMME IST ="; S
120  END
```

READ und DATA-Statements

Wenn der Computer mit einer großen Anzahl von Daten versorgt werden muß, kann die Eingabe über den INPUT-Befehl recht mühsam werden. In diesem Falle greift man dann zu den READ, DATA-Befehlen.

Der READ-Befehl legt die Variablen fest, deren Werte über das Programm in den Computer gegeben werden sollen. Dieser Befehl besteht aus dem Wort READ, gefolgt von einer Liste von Eingabevariablen. Diese Variablen müssen durch Komma getrennt werden. Die Liste kann Zahlen, Strings oder beides enthalten.

Der Sinn des DATA-Befehls ist es nun, den Variablen aus dem READ-Befehl die zugehörigen Werte zuzuordnen. Der DATA-Befehl besteht aus dem eigentlichen Befehl "DATA" gefolgt von den entsprechenden Zahlen oder Strings. (Jede Position muß wieder durch Komma getrennt werden).

Programmbeispiel:

```
10   READ  A, B, C, D
20   DATA  100, BERTA, CAESAR, 200
```

Bevor das Programm startet, nimmt der BASIC-Interpreter alle Daten aus den DATA-Anweisungen in der eingegebenen Reihenfolge von links nach rechts und speichert sie in einem großen Block. Wenn dann im Programm eine READ-Anweisung erscheint, wird der zugehörige Wert aus dem Datenblock geholt und zugeordnet.

Wenn mehr READ-Anweisungen im Programm vorkommen, als zugehörige Daten vorhanden sind, wird eine Fehlermeldung gegeben.

Die READ-Befehle werden im Normalfall zu Beginn des Programmes platziert. Die DATA-Befehle können im Programm an beliebiger Stelle eingeführt werden. Wichtig ist nur, daß die Reihenfolge stimmt. Viele Programmierer bevorzugen jedoch die DATA-Anweisungen am Programmende zu platzieren.

Merke:

1. Die Werte in den DATA-Statements müssen den zugehörigen Variablen in den READ-Anweisungen entsprechen.

2. Es müssen mindestens so viele Datenelemente in DATA-Anweisungen vorhanden sein, wie READ-Anweisungen gegeben werden. Zusätzliche Daten werden ignoriert.

3. Die Elemente in den DATA-Statements müssen durch Komma getrennt werden. Nach dem letzten Element folgt kein Komma.

4. Die Elemente in DATA-Statements müssen Daten oder Strings sein, keine Variablen oder Formeln.

5. Strings in der DATA-Anweisung, die ein Komma enthalten, oder mit einem Leerzeichen beginnen oder enden, müssen in Anführungszeichen gesetzt werden.

Der GOTO-Befehl

Der GOTO-Befehl wird dazu benutzt, an eine bestimmte Stelle (Zeile) im Programm zu springen. Normalerweise arbeitet das Programm von der Zeile mit der kleinsten Zeilennummer an, bis zur größten Zeilenzahl ab. Wenn ein unbedingter Sprung erforderlich ist, wird der GOTO-Befehl benutzt. Bei einem solchen unbedingten Sprung erfolgt keine logische Entscheidung.

Der Programmablauf wird durch den GOTO-Befehl in seiner Reihenfolge unterbrochen und das Programm geht an die im GOTO-Befehl angegebene Zeilenzahl. Von dort wird dann wieder in der bisherigen Reihenfolge weiter abgearbeitet.

Programmbeispiel:

```
20   REM   SPRUNG DURCH GOTO
30   REM   UNBEDINGTER SPRUNG
40   REM   DIESES PROGRAMM BE-
45   REM   RECHNET DEN MITTELWERT
50   REM   VON N ZAHLEN. UM DIE
60   REM   SCHLEIFE ZU VERLASSEN
70   REM   GEBEN SIE 0 EIN
80   LET   K = 0
90   LET   N = 0
100  INPUT J
110  IF J = 0 THEN 150
120  LET   K = K+J
130  LET   N = N + 1
140  GOTO  100
150  PRINT „SUMME IST = ";K
160  PRINT „DURCHSCHNITT IST ="; K/N
170  END
```

Im vorangegangenen Beispiel sind wir wieder auf eine unbekannte Anweisung gestoßen. (110 IF J = 0 THEN 150)

Der IF-Befehl

Der IF-Befehl wird für bedingte Sprünge verwendet. Mit ihm kann dann von der normalen Reihenfolge (d. h. Abarbeiten von der kleinsten Zeilenzahl zur höchsten Zeilenzahl) abgewickelt werden. Das Programm springt dann zu der im IF-Befehl definierten Zeilennummer, wenn die gestellte Bedingung erfüllt ist.

Beispiel:

```
100  IF X = 0 THEN 200
200  IF SIN ( X ) ⟨ = 0,5 THEN 100
300  LET Y = 1
```

Bei diesem Beispiel erfolgt ein Sprung des Programmes in Zeile 200, wenn der Wert für X null ist. Wenn SIN (X) kleiner gleich 0,5 ist, erfolgt ein Rücksprung nach Zeile 100. Wenn beide Bedingungen nicht erfüllt sind, geht das Programm zur nächsten Zeile (z. B. 300).

FOR .....NEXT-Schleifen

Wenn wir beim Programmieren wissen, wie oft ein Befehl oder eine Gruppe von Befehlen wiederholt werden muß, können wir den FOR-Befehl verwenden. Diese Anweisung legt fest, wie oft eine Programmschleife durchlaufen wird. Dem unmittelbaren Befehlswort FOR folgt die Ablaufvariable. Der Wert dieser Variable ändert sich nach jedem Durchlauf der Schleife. Die Anzahl der Durchläufe wird durch den Anfangs- und Endwert dieser Variablen festgelegt.

Beispiel:

```
100  FOR   X = 1 TO 100
200  FOR   Y = 10 TO 200
```

Beim Befehl hier in Zeile 100 wird zu Beginn des ersten Schleifendurchlaufes die Ablaufvariable auf 1 gesetzt. Dann wird diese nach jedem Durchlauf um eins erhöht, bis der Wert 100 erreicht wird. Die Ablaufvariable wird grundsätzlich um eins erhöht, es sei denn, es wird etwas anderes vorgeschrieben.
Eine solche Änderung kann mit dem STEP-Befehl herbeigeführt werden. Mit diesem Befehl können wir die Ablaufvariable um einen vorgegebenen Wert erhöhen oder auch vermindern.

```
100  FOR   Y = 1 TO 20 STEP 2
200  FOR   Y = 100 TO 10 STEP – 10
```

Generell gilt: FOR (Ablaufvariable) = Anfangswert) TO (Endwert) STEP (Schrittgröße)

Um eine FOR TO-Schleife abzuschließen, benötigen wir den NEXT-Befehl. Der NEXT-Befehl besteht aus einer Zeilenzahl der NEXT-Anweisung und der Ablaufvariablen. Auf jeden Fall muß die Ablaufvariable folgen, die zur gewünschten Schleife gehört.

```
100  FOR   X = A TO B
110  .....(beliebiger Befehl)
120  ..... (beliebiger Befehl)
130  .....(beliebiger Befehl)
140  NEXT  X
```

Merkregeln:

1. Die Ablaufvariable kann in einem weiteren Befehl innerhalb der Schleife erscheinen, kann aber nicht geändert werden.

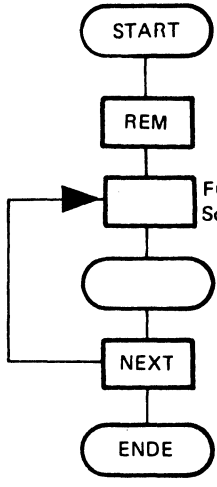
2. Wenn der Anfangswert und Endwert in einer Schleife gleich ist, und die Schritt-

größe nicht 0 ist, wird die Schleife nur einmal durchlaufen.

PROGRAMMBEISPIEL:

```
10   REM   DIESES   PROGRAMM   BE-
20   REM   NUTZT   EINE   FOR...NEXT
25   REM   SCHLEIFE
30   FOR   I = 1 TO 10
40   PRINT  I * 10
50   NEXT   I
60   END

RUN
10
20
30
40
50
60
70
80
90
100
```



Flußdiagramm:

Unterprogramme GOSUB

Wenn eine bestimmte Reihenfolge von Befehlen in einem Programm ständig benutzt werden soll, können diese Befehle als Unterprogramm geschrieben werden. Ein Unterprogramm ist ein komplettes, vollständiges Programm, welches vom Hauptprogramm oder von anderen Unterprogrammen aufgerufen werden kann. In BASIC erfolgt der Sprung in ein Unterprogramm durch den Befehl GOSUB mit nachfolgender Zeilenangabe, wo sich das Unterprogramm befindet. Der Computer merkt sich die Zeilennummer, von der er ins Unterprogramm gesprungen ist. Wenn das Unterprogramm abgearbeitet ist und ein RETURN-Befehl gegeben wurde, kehrt das Programm an die Zeile zurück, die als nächste unmittelbar dem GOSUB-Befehl folgt. Von dort aus wird dann im Hauptprogramm „weitergefahren".
Der RETURN-Befehl besteht einfach aus Zeilenzahl und RETURN. Z. B.

```
100 RETURN
```

PROGRAMMBEISPIEL:

```
10   REM     DIESES PROGRAMM ZEIGT
20   REM     DIE ARBEITSWEISE DES GO–
25   REM     SUB-BEFEHLES
30   PRINT   „EINGABE EINER ZAHL"
40   INPUT   I
50   IF I = 0 THEN 110
60   GOSUB 90
70   PRINT K
80   GOTO 30
90   LET     K = (I * 2) /3
100  RETURN
110  END


RUN
EINGABE EINER ZAHL
? 5
3. 3333
EINGABE EINER ZAHL
? 1256
837.333
EINGABE EINER ZAHL
? 0
READY
```

Strings (Zeichenketten)

Neben Zahlen als Variable können auch String Variable benutzt werden. Ein String (Zeichenkette) ist eine Folge von Zeichen (alphanumerisch), Sonderzeichen, Leerzeichen etc. Auf keinen Fall jedoch die Anführungszeichen. Sie bezeichnen Anfang und Ende einer Zeichenkette (String). Die Anzahl der Zeichen, die Sie in einem String verwenden können, hängt von Ihrer BASIC-Version ab.

Die Aufgabe von Strings ist es, nicht numerische Daten, wie Bezeichnungen und Erläuterungen zu repräsentieren. Eine Folge von Zahlen in einem String repräsentiert deshalb keine numerischen Daten!

PROGRAMMBEISPIEL:

```
10  REM     DIESES PROGRAMM ZEIGT
20  REM     DIE STRINGVERWENDUNG
30  PRINT   „GEBEN SIE EINFACH VIER
            BELIEBIGE BUCHSTABEN EIN
40  INPUT   J$(1), J$(2), J$(3), J$(4)
50  FOR     M = 1 TO 4
60  FOR     N = 1 TO 4
70  IF N = M THEN 140
80  FOR P = 1 TO 4
90  IF P = M THEN 130
100 IF P = N THEN 130
110 LET     R = 10 ( M + N + P )
120 PRINT   J$(M); J$(N); J$(P); J$(R)
130 NEXT    P
140 NEXT    10
150 NEXT    M
```

Einfaches Beispiel:

```
10  LET     C$ = „BASIC IST"
20  LET     E$ = „SEHR"
30  LET     F$ = „EINFACH"
40  PRINT   C$; E$; E$; F$

RUN
BASIC IST SEHR SEHR EINFACH
```

INPUT MIT STRINGS

Bei der Eingabe von Text wird ähnlich wie bei der Eingabe von Daten verfahren. Es wird dem Wert lediglich ein Dollarzeichen nachgestellt.

Beispiel:

```
10  INPUT   „WIE HEISST DU"; N$
20  PRINT   N$; „IST MEIN NAME."
30  INPUT   „WIE ALT BIST DU"; J
40  PRINT   N$; " ,DU BIST"; J ; " JAHRE
            ALT."
```

READ und DATA mit String

READ und DATA-Befehle arbeiten mit Strings (Zeichenketten) ähnlich wie mit Daten.

Beispiel 1:

```
10 READ A$
20 PRINT A$
30 DATA NEUNZEHN

RUN
NEUNZEHN
```

Beispiel 2:

```
10 READ B$
20 PRINT B$
30 DATA 19
RUN
19
```

Im zweiten Beispiel wird die Zahl 19 wie eine Textaussage aufgefaßt. Sie kann nicht weiterverarbeitet werden. Obwohl dem DATA-Befehl eine Zahl folgt.

Beispiel:

```
10 READ A$, B$, C$, D$
20 PRINT C$, B$, D$, A$
30 DATA,PETER,DER,SOLANG,ALTE
RUN

SOLANG DER ALTE PETER
```

Beachten Sie bitte bei diesen Beispielen, daß der READ-Befehl die Zeichenkette aus den DATA-Statements so zuordnet, wie Sie in der READ-Anweisung vorgegeben sind. Es spielt keine Rolle, wo die DATA-Anweisungen im Programm angeordnet werden.

Jetzt wollen wir einmal versuchen, in den Datastatements Strings und Werte (Daten) zu verwenden.

```
10 READ A$, B$, C, D
20 PRINT C; B$; D; A$; C + D

90 DATA  UND, IST, 1, 2
RUN
1 UND 2 IST 3
```

Wir sehen also, daß unser Computer numerische Variable und Stringvariable in READ und DATA-Befehlen verarbeiten kann.

Mehrfache Anweisungen in einer Zeile

Der PET-Computer bietet wie viele andere leistungsfähige Computersysteme die Möglichkeit, in einer Zeile mehrere Anweisungen anzuordnen.

```
100 A = 2 : B = 3 : C = 5:? A * B * C
RUN

30
```

Die einzelnen Statements werden durch Doppel-

punkt voneinander getrennt. Das Fragezeichen ist eine praktische Abkürzung für den PRINT-Befehl.

Die Quadrat- und Wurzelfunktion

Wie in der Einleitung bereits erwähnt, enthalten viele BASIC-Versionen automatische Berechnungsabläufe, genannt Funktionen. Oft bestehen diese Funktionen aus kompletten Formeln, die Zahlen sowie Zeichenketten handhaben können.

Die erste BASIC-Funktion, die wir heute besprechen wollen, ist die SQR (Squareroot = Quadratwurzel)-Funktion.

Beispiel:

```
10 LET A = 81
20 PRINT SQR (A)
```

Die Quadrierung von Zahlen erfolgt beim PET durch das „ ↑ " Zeichen. In anderen BASIC-Versionen findet man auch „ * * " (Zwei Sternchen hintereinander).

Das Beispiel:

```
10 PRINT A ↑ 2
```

liefert das Quadrat von A.

Die INT (X)-Funktion

Die INT (X)-Funktion liefert immer eine ganze, positive Zahl. (Also keine gebrochenen Zahlen) Die Anwendung dieser Funktion finden wir u.a. in Spielprogrammen, in der die Random (RND)-Funktion verwendet wird. Die RND-Funktion (Zufallsfunktion) liefert z. B. beim PET eine Zufallszahl zwischen 0 und 1. Da diese Zufallszahlen jetzt alle gebrochen sind, erreicht man durch die INT (A)-Funktion eine Auswahl der ganzen Zahlen.

Beispiel:

```
10 LET X = INT (100 * RND (1))
```

Dieses Statement erzeugt also nur Zufallszahlen im Bereich zwischen 1 und 100.

Beispiel:

```
10 PRINT INT (3, 1)
 3
20 PRINT SQR (52), INT (SQR (52))
   7.211              7
```

Sie sehen also, daß diese Funktion immer nur eine ganze Zahl liefert.

Die RND (X)-Funktion

In einem der obigen Beispiele sind wir wieder auf eine neue und recht wichtige Funktion gestoßen, die RND (X) oder auch Randomfunktion.
Sie ist nichts anderes als eine Zufallsfunktion. Es werden Zufallszahlen in einem bestimmten Bereich generiert.

Dies geschieht ähnlich wie in einer Zahlenlotterie.

Die RND-Funktion liefert, wie schon erwähnt, nur gebrochene Zahlen zwischen 0 und 1, niemals 0 und niemals 1, nur Zahlen dazwischen.

Beim Commodore PET liefern die Argumente in Klammern folgende Werte:

RND (X)	X kleiner 0. Bei jedem Aufruf wird die gleiche Zufallszahl generiert.
RND (X)	X = 0. Bei jedem Aufruf wird die die gleiche Reihe von Zufallszahlen generiert.
RND (X)	X größer 0. Bei jedem Aufruf wird eine neue Reihe von Zufallszahlen generiert.

Das Argument ist positiv

Beispiele:

```
10   X = 1
20   PRINT RND (X):GOTO 20
RUN

0.43481
0.69751
0.47892
0.86533
0.12411

BREAK IN 20
READY
```

Das Argument ist Null

```
10  X = 0
20  PRINT RND (X) : GOTO 20
RUN

0.21212
0.21212
0.21212
0.21212

BREAK in 20
READY
```

Argument ist negativ

```
10  X =–0,4
20  PRINT RND (X) : GOTO 20
RUN

0.90235
0.90235
0.90235

BREAK IN 20
READY
```

Beispiel:

```
05  REM ZUFALLSGENERATOR MIT WIE-
        DERHOLUNG
10  LET  X = RND (–0.3)
20  PRINT RND (1)
30  GOTO 20
RUN
0.56789
0.72154
0.31431
0.67891
STOP
RUN

0.56789
0.72154
0.31431
0.67891
STOP
```

In diesem Programm wird bei jedem Ablauf immer die gleiche Liste von Zufallszahlen generiert. Die Funktion RND (–0.3) erzeugt am Anfang immer die gleiche Zufallszahl. RND (1) erzeugt dann beliebige Zahlen.

Wie lernt man BASIC, Teil III

Wie lernt man BASIC? Teil III

Rückblick:
In den vorangegangenen zwei Lektionen sind wir bis zur RND (X) Funktion gekommen. Heute wollen wir gleich mit einem interessanten Beispiel zur RND(X) Funktion beginnen.Wie wir gelernt haben, erzeugt die Funktion RND (1) bei jedem Aufruf eine neue Reihe von Zufallszahlen.
Wir wollen nun einmal ein kleines Programm schreiben, welches uns eine Reihe von Zufallszahlen erzeugt.

```
READY.

 100 REM ZUFALLSZAHLEN
 120 PRINT" WIE VIEL ZAHLEN SOLL ICH
           ERZEUGEN";
 130 INPUT N
 140 PRINT
 150 FOR K=1TON
 160 PRINT INT(RND(1)*49) ;
 170 NEXT K
 180 PRINT
 190 PRINT
 200 GOTO 120
 999 END
READY.
```

```
RUN
WIEVIEL ZAHLEN SOLL ICH ERZEUGEN?10
1  4  7  9  3  2  1  5  6  9

WIEVIEL ZAHLEN SOLL ICH ERZEUGEN?
BREAK IN 200
READY
```

Versuchen Sie einige Experimente mit diesem Programm. Lassen Sie einmal das Semikolon in Zeile 160 weg. Entfernen Sie einmal die Zeilen 180 und 190. Beobachten Sie, was sich verändert.
Versuchen Sie einmal Zufallszahlen zwischen 0 und 49 (Lottozahlen) zu erzeugen.

Die TAB-Funktion

Eine andere wichtige Funktion ist die TAB-Funktion. Sie wird in PRINT-Anweisungen angewendet und arbeitet wie der Tabulator bei einer Schreibmaschine. Der Computer wird dadurch angewiesen, einen bestimmten freien Raum auf dem Bildschirm zu überspringen.

Hierzu einige Beispiele:

```
READY.

 10 PRINT"          A      A"
 30 END
 40 PRINT TAB(10);"A";TAB(16);"A"
 50 END
READY.
```

Wenn hier die Eingabe in Zeile 10 mit 10 Leertasten erfolgt, 10 PRINT" ← 10 Leertasten A ← 5 Leertasten → A, so kann mit Zeile 40 das gleiche erreicht werden.

Beim Commodore PET haben wir 40 Zeichen pro Zeile. TAB 20 würde dort das Zeichen in der Mitte der Bildschirmzeile bringen.

In der Klammer, die dem TAB(X)-Befehl folgt, kann eine Zahl, eine Variable oder ein mathematischer Ausdruck sein.

Beispiel: TAB (20), TAB (A), TAB (INT(10 *RND(1)))

```
READY.

 10 REM DEMOPROGRAMM FUER TAB
 20 FOR I=1 TO 23
 30 PRINT TAB(I)"ELCOMP"
 40 NEXT I
 50 END
READY.
```

```
RUN
ELCOMP
   ELCOMP
      ELCOMP
         ELCOMP
            ELCOMP
READY
```

Ein weiteres Beispiel:

```
10 REM TAB-DEMO II
20 READ Y
30 PRINT TAB (Y);Y
40 GOTO 10
50 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
99 END

RUN
1
   2
      3
         4
            5
               6
                  7
                     8
                        9
                           10
READY
```

Hier wird die Zahl Y, die nacheinander aus Data-Statements gelesen wird, um die entsprechende Anzahl von Schritten nach rechts versetzt. (Y = Zahl = Schritte nach rechts)

Ändern Sie einmal Zeile 30 in

30 PRINT TAB (Y);"*"

und starten Sie das Programm.

Die TAB-Funktion wird meist in der Computer-Graphik und zum Ausdrucken von mathematischen Funktionen angewendet.
Ein weiteres Beispiel:

```
READY.
 5 REM SORTIERPROGRAMM
 10 PRINT"UNTER 1000.-DM";TAB(18);"UNTER
        2000.-DM"
 20 READ X
 30 IF X<1000 THEN 60
 40 IF X<2000 THEN 80
 50 GOTO 20
 60 PRINT X
 70 GOTO 20
 80 PRINT TAB(18);X
 90 GOTO 20
 100 DATA 1000,1100,1500,1600,2500
 110 DATA 2600,900,700,2850,2900
READY.
```

Wir haben jetzt die wichtigsten Funktionen eines BASIC, wie SQR, INT, TAB und RND kennengelernt. Zum Schluß wollen wir Ihnen noch eine Anweisung zeigen, mit der Sie selbst ihre eigenen Funktionen festlegen können. Sie können somit selbst eine Rechnung, die immer wieder vorkommt als Funktion festlegen.
Sie werden als Anweisung in einem Programm geschrieben.

Beispiel:

```
10 DEF FNR (X) = (INT(X*100+0,5))/100
```

10 = Zeilenzahl
DEF = DEFINE-Anweisung
FN = Funktion
R =Genaue Bezeichnung der Funktion R = Runden. Es können alle Buchstaben von A bis Z als Kennzeichen verwendet werden.
(X) =Variable als Ersatz für alle kommenden Variablen.

Will man nun im Programm diese Funktion für eine Variable benutzen, wird die FNR-Funktion (Variable) geschrieben.

Beispiel eines Programmes zum Abrunden von Zahlen

```
05 REM DEF-DEMO
10 DEF FNR (X) = (INT (*100+0,5))/100
20 PRINT "ZAHL DIE ABGERUNDET WER-"
25 PRINT "DEN SOLL";
30 INPUT Y
40 PRINT Y; "GERUNDETE ZAHL AUF"
45 PRINT "2 DEZIMALSTELLEN=";FNR (Y)
50 PRINT
60 GOTO 20
70 END
```

ARRAYS (Felder)

Arrays sind Ansammlungen von Informationen im Speicher, welche sich an einer nummerierten Position befinden. Die Positionen (Speicherplätze) können alle möglichen Informationen enthalten. (Daten, Zahlen, Buchstaben etc.)

Die Inhalte gehören meist auf irgendeine Weise zusammen.

Beispiel für ein Eindimensionales Feld:VEKTOR

Position:

1	2	3	.....	n–2	n–1	n

n = letzte Position

Es gibt ein, zwei, drei- oder auch mehrdimensionale Felder.

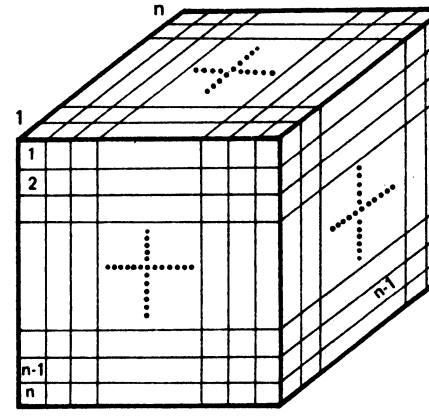
Wir wollen uns heute nur mit den ein- zwei- und dreidimensionalen Feldern beschäftigen.

Das eindimensionale Feld ist wie ein Vektor. (Eine Reihe von Positionen in eine Richtung)

Ein zweidimensionales Feld ist wie eine Matrix (Tabelle).

	1	2				n–1	n
1							
2							
n–1							
n							

Beispiel für ein zweidimensionales Array (Matrix). Ein dreidimensionales Feld ist wie ein Würfel.



Die einzelnen Elemente in solch einem Array (gleich welche Dimension) werden durch Indizierung angesprochen. Je nachdem, wieviele Dimensionen das Feld hat, wird es entsprechend indiziert (bezeichnet).

Eindimensionales Feld: VEKTOR
Zweidimensionales Feld: MATRIX

Beispiel: Nehmen wir an, wir haben ein eindimensionales Feld mit dem Namen NAM. Man kann in einem Programm nach dem dritten Element in diesem Array durch NAM (3) zugreifen. Eine weitere Möglichkeit ist es, wenn man anstelle der Zahl 3 eine Variable z. B. X verwendet und durch Verändern dieser Variable zu dem gesamten Feld zugreifen kann.

```
READY.

 10 REM ARRAY DEMO PROGRAMM
 20 DIM NAM(20)
 30 FOR I=1 TO 20
 35 PRINT"NAM(";I;")=";NAM(I)
 40 LET NAM(I)=I
 50 NEXT I
 60 END
READY.
```

Dieses kleine Demoprogramm erstellt ein eindimensionales Feld mit 20 Positionen und druckt den Inhalt der Felder aus. Dies bleibt auch so, bis es durch das Programm geändert wird. Beim Microsoft können Arrays bis 255 Elemente programmiert werden. Die Änderung einer Zelle kann jetzt ganz einfach erfolgen, indem wir die Zeile 40 durch

40 LET NAM (3) = 4

ersetzen. Damit schreiben wir in das Array mit dem Namen NAM in Zelle 3 die Zahl 4 ein.

Wollen wir in alle Zellen etwas hineinschreiben, so fügen wir folgendes ein.

```
READY.

 10 REM ARRAY DEMO PROGRAMM
 20 DIM NAM(20)
 30 FOR I=1 TO 20
 35 PRINT"NAM(";I;")=";NAM(I)
 36 FOR X=1TO 20
 37 LET NAM(X)=X-1
 38 NEXT X
 50 NEXT I
 60 END
READY.
```

Dieses Programm bringt uns die Zahlen von 0-19 in die Positionen 1 — 20 des Feldes mit dem Namen NAM.

Bei Arrays mit mehreren Dimensionen gilt das gleiche Prinzip. Man kann jedes Element im Array durch Angabe der Zahl in Klammern ansprechen.

Beispiel für ein zweidimensionales Array

Nehmen wir einmal an, wir haben eine Preisliste für verschiedene Artikel in unterschiedlichen Größen. Hieraus ergibt sich ein zweidimensionales Feld wie folgt:

Größe					
1					
2					
3					
4	31,--	42,--	54,--	56,--	67,--
5	20,--	30,--	40,--	50,--	60,--
6	10,--	20,--	30,--	40,--	50,--
	Typ 1	2	3	4	5

Wäre der Name dieses zweidimensionalen Feldes z. B. PREIS, so wäre der Preis von DM 54,--

PREIS (4,3)

da er sich in Reihe 4 und Zeile 3 befindet.

So, wie man Arrays mit Zahlen aufbauen kann, kann man auch Namen benutzen.

Sie haben nun in groben Zügen erfahren, wie Arrays aufgebaut sind. Was man damit machen kann, können Sie jetzt selbst ausprobieren.

Man kann Zahlen, Namen etc. sortieren und verschieben und hat immer leichten Zugriff zu jedem Element.

Beispiele zu der Feldanweisung DIM

Wir wollen mit unserem BASIC-Rechner einen Würfel simulieren. Dieser liefert uns Zufallszahlen zwischen 1 und 6. Damit wir nach dem Würfeln wissen, wie oft jede Zahl geworfen wurde, wollen wir die einzelnen Würfel notieren. Hierzu verwenden wir ein Feld (Array).

```
5 REMSIMULATIONSPROGRAMM
10 DIM(6)
15 INPUT"WIEVIELE WUERFE";W
20 FOR X=1 TO W
30 D=INT(6*RND(1))+1
40 LET T(D)=T(D)+1
50 NEXT X
60 FOR X=1TO6
70 PRINT X;"S:";T(X)
80 NEXT X
90 PRINT
91 FOR J=1TO6
92 LET T(J)=0
93 NEXT J
100 GOTO 15
110 END
```

Zerstören wir einmal die Zeile 10 und versuchen wir das Programm wieder zu starten. Was wird geschenen?
Siehe da, es läuft auch ohne diese Anweisung. Jetzt werden Sie denken „Alles umsonst, was ich bis jetzt über DIM gelernt habe". Aber warten wir ab!

Ändern Sie jetzt einmal das Programm wie folgt ab.

```
READY.

 5 REMSIMULATIONSPROGRAMM
 10 DIMT(49)
 15 INPUT"WIEVIELE WUERFE";W
 20 FOR X=1 TO W
 30 D=INT(49*RND(1))+1
 40 LET T(D)=T(D)+1
 50 NEXT X
 60 FOR X=1TO49
 70 PRINT X;"S:";T(X)
 80 NEXT X
 90 PRINT
 91 FOR J=1TO49
 92 LET T(J)=0
 93 NEXT J
 100 GOTO 15
 110 END
READY.
```

Dieses Programm simuliert die Lottozahlenmaschine und sagt Ihnen, wie oft bei mehreren Ziehungen die eine oder andere Zahl gezogen wurde.

Löschen Sie jetzt auch wieder Zeile 10 und versuchen Sie das Programm zu starten. Jetzt bekommen Sie eine Fehlermeldung.

Bei mehr als 10 Felder in einem Array oder bei mehreren Arrays pro Programm muß auf jeden Fall eine DIM(X) Anweisung gegeben werden. Unter zehn kann mit fortlaufenden Variablen gearbeitet werden (subscripted variables).

Auffüllen von Arrays aus DATA-Statements

Wir haben unsere Felder bei den vorangegangenen Beispielen immer durch die LET-Funktion aufgefüllt.

40 LET T (D) = T (D) + 1

Wir können die Felder natürlich auch aus DATA-Statements her auffüllen. Bei größeren Mengen von Daten ist dies wesentlich praktischer.

Beispiel:

Wir haben zum Beispiel in unserer Abteilung zwei Artikelgruppen 1 und 2. Wir wollen am Abend wissen, wieviele Artikel gesamt verkauft wurden. Wieviele davon waren 1 und wieviele davon 2.

```
READY.

 5 REM DIM UND DATA DEMO
 9 DIM T(2)
 10 READ I
 20 IF I=9999 THEN 50
 30 LET T(I)=T(I)+1
 40 GOTO 10
 50 PRINT"GESAMT:";T(1)+T(2)
 60 PRINT"PRODUKT1:";T(1)
 70 PRINT"PRODUKT2:";T(2)
 900 DATA 1,1,1,1,1,2,2,2,2,2,2,2,2,1
 904 DATA 1,2,2,2,2,1,1,1,1,1,1,1,1,1
 905 DATA 9999
READY.
```

Ich brauche jetzt immer nur die Artikel Nr. in die DATA-Statements einzugeben. Am Abend gebe ich RUN ein und habe die gesamte verkaufte Stückzahl und die Stückzahl nach Artikelnummern aufgegliedert.

Sie können dieses Programm bis zu 255 Artikelgruppen einfach erweitern.

Die Listings wurden über einen Drucker ohne Graphiksymbole ausgedruckt. Wir bitten Sie deshalb, bei der Eingabe in Ihren PET darauf zu achten.

Wie lernt man BASIC, Teil IV

Wie lernt man BASIC, Teil IV

Rückblick:
In der letzten Lektion (ELCOMP 1/79) sind wir bis zu den Arrays (Feldern) gekommen. Speziell haben wir uns zuletzt mit den Feldern beschäftigt, die mehr als 10 Elemente enthalten. Hierbei haben wir den Befehl DIM I(X) kennengelernt.
Mit diesem Befehl wird die Anzahl X der Elemente in einem Feld mit dem Namen I festgelegt.

Z. B. ist DIM I(10) ein Feld mit zehn Elementen. Es gibt auch BASIC-Versionen, die ein Element Null kennen. Dann haben wir es in diesem Fall mit 11 Elementen zu tun.

Die Zahl in Klammern (X) hinter dem DIM-Befehl kann niemals eine Variable sein. X kann nur eine ganzzahlige, ungebrochene Zahl sein.
Der Programmierer bevorzugt es, die DIM-Statements immer an den Anfang eines Programmes zu setzen. Hier hat man eine gute Übersicht und kann leicht kontrollieren, ob alle Felder richtig dimensioniert sind.

Wir wollen uns zur Vertiefung noch einige Beispiele ansehen. Ganz abgesehen davon, daß der DIM-Befehl beim Programmierer eine sehr wichtige Stellung einnimmt.

Zweidimensionale Arrays
Wir haben bis jetzt nur Felder kennengelernt, deren Elemente nur eine einzige Zuordnungsvariable (Subscript) haben. Oft ist es jedoch beim Programmieren notwendig, mit zwei Variablen zu arbeiten.

Man nennt diese Arrays, wie im letzten Kursabschnitt schon angedeutet, zweidimensionale Felder. Diese Felder eignen sich besonders zur Darstellung von Tabellen oder Variablen mit zwei Zuordnungswerten (Subscripts). Gleich ein Beispiel.

Nehmen wir einmal an, wir wollen eine Statistik (Aufzeichnung) von Artikelnummern pro Wochentag anfertigen. Im letzten Kurs hatten wir ein kleines Programm entworfen, welches uns pro Artikelgruppe eine Gesamtmenge angibt.

Jetzt wollen wir dieses Programm insoweit ausdehnen, daß wir Artikelgruppen pro Wochentag untersuchen wollen. Eine Liste könnte hierzu wie folgt aussehen:

Artikel-Nr.	Fr.	Sa.	Su
1	10	20	30
2	5	10	15

(100 Artikel sind hier möglich)
Diese Tabelle sagt uns z. B., daß am Freitag 5 Stück aus der Artikelgruppe 2 und 10 Stück aus der Artikelgruppe 1 verkauft wurden. Am Samstag waren es 10 aus Warengruppe 2 und 20 aus Warengruppe 1.

Wir wollen jetzt wieder ein kleines Programm entwerfen, welches uns die einzelnen Stückzahlen pro Artikel und Tag aufaddiert und am Ende eine Übersicht über unsere verkaufte Menge pro Artikelgruppe und pro Tag gibt. Wir haben in einem Beispiel nur zwei Tage genommen, da der Platz beim PET gerade über den Bildschirm reicht. Sie können jedoch auf beliebig viele Tage (max. 255) erweitern, indem Sie in Zeile 20 den Befehl DIM (100,4) entsprechend ändern, und die Frage nach Eingabe mit INPUT entsprechend ausdehnen.

```
READY.

 1 REM COPYRIGHT ING W HOFACKER GMBH
 2 REM DEMO FUER ARRAYS
 15 REM FUER ZWEIDIMENSIONALE FELDER
 20 DIM A(100,4)
 30 REM DATENEINGABE
 40 PRINT"GEBEN SIE BITTE DIE ANZAHL DER
 42 PRINT"ARTIKEL EIN! MAXIMAL 100"
 50 INPUT N
 60 PRINT"GEBEN SIE BITTE DIE ARTIKELNUM-
    MER EIN!  VON 1 BIS N (REIHENFOLGE)
 75 INPUT K
 77 FOR K=1 TO N
 82 PRINT"ANZAHL ARTIKELGRUPPE";K;"AM
          FREITAG
 90 INPUT A(K,1)
 91 NEXT K
 92 FOR K= 1 TO N
 96 PRINT"ANZAHL ARTIKELGRUPPE";K;"AM
          SAMSTAG
 97 INPUT A(K,2)
 98 LETA(K,3)=A(K,1)+A(K,2)
 99 NEXT K
 110 REM TABELLE
 120 PRINT"ART.NR.","FR","SA","SU"
 130 FOR J= 1 TO N
 140 PRINT J,A(J,1),A(J,2),A(J,3)
 155 NEXT J
 160 REM AUFSTELLUNG DER VERKAUFSZAHLEN
 999 END
READY.
```

Das Programm ließe sich noch dahingehend erweitern, daß man die verkauften Artikel dann nach Mengen ordnet, um die Renner von den Schleichern zu trennen.

Geben Sie dieses Beispielprogramm nun einmal in Ihren Commodore PET Computer und füllen Sie das Array mit zwei Artikelnummern (N=Z) und geben Sie Beispieldaten für verkaufte Stückzahlen ein. Z. B. Artikel-Nr. 1 Montag 20 Stück, Artikel-Nr. 2 Montag 32 Stück, Artikel-Nr. 1 Samstag 54 Stück und Artikel-Nr. 2 am Samstag 89 Stück. Nach der Eingabe erstellt der Computer sofort eine kleine Statistik mit der Gesamtstückzahl pro Artikelgruppe.

Sie können jetzt in die einzelnen Zellen des Arrays A(100,4) einsehen, indem Sie ?A(X,Y) mit Return eingeben. (X und Y sind die gewünschten Koordinaten).
Geben Sie einmal ?A(1,1) Return ein! Sie erhalten 20. Das ist die erste Eingabe (20 Stück von Artikelgruppe 1 am Freitag).
Sie sehen, wie man jetzt leicht jedes einzelne Element des Arrays ansehen kann. Sie können jetzt auch mit diesen Elementen arbeiten, indem Sie sie verknüpfen oder manipulieren.
Beispiel: ?A(1,1)+ A(2,1)

Stellen Sie jetzt selbst einige Versuche mit diesem Programm an, um etwas Erfahrungen zu sammeln.

Eingabe und Auslesen eines Feldes
Das nachfolgende kleine DEMO-Programm zeigt uns, wie man ein Array (Feld) auffüllen und auslesen kann. Wenn es sich wiederholen soll, geben Sie bitte in Zeile 230 GOTO 80 ein.

So, das soll nun vorerst über Felder (Arrays) genug sein. Sie können jetzt bereits einige kleine Programme selbst entwerfen und damit experimentieren.

```
READY.

 5 REM ARRAY DEMO
 10 DIM A(3,4)
 20 FOR X=1 TO 3
 30 FOR Y=1 TO 4
 40 INPUT A(X,Y)
 50 NEXT Y
 60 NEXT X
 70 PRINT"DAS ARRAY IST VOLL!!"
 80 PRINT"WELCHES ELEMENT WILLST DU SEHEN?"
 90 INPUT"WELCHE REIHE";X
 100 INPUT"WELCHE SPALTE";Y
 110 PRINT"A(";X;",";Y;")="A(X,Y)
 150 REM ORDNEN IN REIHENFOLGE
 160 PRINT" DER GESAMTINHALT DES FELDES
 170 FOR X=1 TO 3
 180 FOR Y=1 TO 4
 190 LET Z=A(X,Y)
 195 FOR Z=1 TO 4
 200 PRINT A(X,Y)
 210 NEXT Y
 220 NEXT X
 999 END
READY.
```

Der RESTORE-Befehl
Der RESTORE-Befehl ist ein wichtiges Hilfsmittel beim Arbeiten mit READ und DATA-Anweisungen. Der Zusammenhang zwischen der READ-Anweisung und den zugehörigen Daten in DATA-Statements wird durch einen internen Pointer hergestellt.
Was ist ein Pointer? Ein Pointer ist das zuletzt gespeicherte Wort im Stack, welches die Adresse einer anderen Speicherzelle beinhaltet. Meistens nach Springen in Unterprogramme etc.

Der interne Pointer zeigt bei den meisten BASIC-Versionen auf das nächste zu lesende Datenelement. Werden Strings und Zahlen in DATA-Statements abgelegt, müssen zwei Pointer bereitgestellt werden. Jedesmal, wenn ein DATA-Element gelesen wird, wird der Pointer um eins erhöht.

Wenn nun das gleiche Datenelement oder ein Block von Datenelementen noch einmal gelesen werden soll, wird der RESTORE-Befehl verwendet. Der Befehl besteht nur aus einem einzigen Wort „RESTORE". Er sorgt dafür, daß der Pointer wieder zurück zum ersten DATA-Element gebracht wird. Die nachfolgende READ-Anweisung beginnt dann wieder beim ersten DATA-Element.

Beispiel für READ-DATA:

```
10 READ A, B, C
20 PRINT A, B, C
30 DATA 2, 5, 10
```

Beispiel für READ-DATA mit RESTORE

```
10 READ A, B, C
20 PRINT A
25 RESTORE
27 READ B
30 PRINT B
40 PRINT C
50 DATA 2, 5, 10

RUN
2
2
10
```

Mit RESTORE und READ können Sie jetzt einmal einige Versuche mit diesem kleinen Programm machen. Sie sehen dann genau, wie Sie diesen Befehl in Zukunft nützlich einsetzen können. RESTORE arbeitet beim PET 2001 nur mit numerischen DATA-Elemente, nicht mit Strings.

Die Fortsetzung dieses BASIC-Kurses finden Sie in dem Buch

"BASIC für Fortgeschrittene"
Best.-Nr.: 122 39,– DM

vom Hofacker Verlag.

Literatur- und Quellenverzeichnis

1. BASIC, Robert L. Albrecht, Leroy Finkel und Jerald R. Brown, John Wiley + Sons. Inc. New York, London, Sydney, Toronto

2. FIRE BASIC-USER-GUIDE, Fairchild 464 Ellis Street, Mountain View, CA 94042, USA

3. The BASIC Cookbook, Ken Tracton, TAB Books (Hofacker Verlag, Holzkirchen)

4. BASIC Programmieren für Anfänger, Haase/Stucky, Bibliographisches Institut, Mannheim

5. North-Star-Horizon, Prospekt

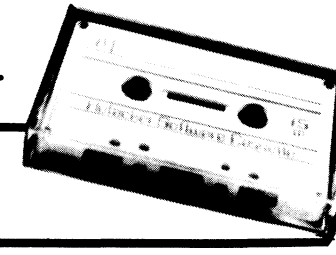
6. APPLE II Reference Manual, Apple Computer Inc. 10260 Brandley Dr. Cupertino, CA 95014, USA

7. BASIC für blutige Laien (Best.-Nr. 139), Hofacker Verlag GmbH, Holzkirchen

8. BASIC für Fortgeschrittene (Best.-Nr. 122), Hofacker Verlag GmbH, Holzkirchen

Besonderen Dank gilt Herrn Flögel, der dieses Buch sorgfältig überarbeitet hat.

Leercassetten für Microcomputer



C–10

Die ideale Cassettenlänge für Ihren Personalcomputer.
Praktisch – handlich und betriebssicher

Kassetten mit nur 10 Minuten Spieldauer (2 x 5 Minuten) haben sich zur Aufzeichnung von Daten im Mikrocomputerbereich bestens bewährt.

Vorteile der C-10 Computer Cassette vom HOFACKER Verlag:

- weniger Bandsalat
- kurze Rückspulzeiten
- schnelles Auffinden von Programmen
- bessere Gleichlaufeigenschaften
- einfache Programmverwaltung

Die C-10 HOFACKER Datencassette bietet weiterhin:

- extrem hoch aussteuerbares Bandmaterial (Agfa)
- hochwertiges Cassettengehäuse, 5fach verschraubt
- Tefloneinlage für gute Laufruhe
- Staubdichtes Glasfenster

Die C-10 HOFACKER Datencassette wird seit 1978 speziell für Microcomputeranwender produziert., Die Cassetten bieten ein Höchstmaß an Betriebssicherheit bezüglich fehlerfreier Aufnahme und Wiedergabe.

Hier eine kurze Übersicht über die Anzahl der Bytes, die Sie auf eine C-10 Cassette abspeichern können:

Computer	Speichermöglichkeit	Computer	Speichermöglichkeit
ATARI 400/800	16K	APPLE	36K
Sharp MZ-80	32K	APPLE II	16K
AIM 65	16K	Heathkit	36K
Ohio Scientific	10K	Kansas City Std.	16K
TRS-80	16K	KIM-1	12K
TRS-80 Color Computer	24K	NASCOM	12K
Video Genie	16K	Exidy Sorcerer	12K
Sinclair ZX80/81	16K	SYM-1	12K

BESTELLSCHEIN

Menge	Beschreibung	Preis/DM	Gesamt
	1 Cassette	**3,50**	
	10 Cassetten	**29,80**	
	100 Cassetten	**249,00**	

Lieferanschrift

Name. . . . . .
Straße . . . . . .
Ort . . . . . .

Datum. . . . . Unterschrift . . . . . .

HOFACKER

Ing. W. Hofacker GmbH
Tegernseerstr. 18
D–8150 Holzkirchen
Tel.: (0 80 24) 73 31

Weitere interessante Bücher von Hofacker:

Best.-Nr.	Titel	Preis/DM
	Bücher in deutscher Sprache aus dem Hofacker-Verlag	
1	Transistor Berechnungs- und Bauanleitungsbuch – 1	29,80
2	Transistor Berechnungs- und Bauanleitungsbuch – 2	19,80
3	Elektronik im Auto	9,80
4	IC-Handbuch, TTL, CMOS, Linear	19,80
5	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	IC-Schaltungen, TTL, CMOS, Linear	19,80
7	Elektronik Schaltungen	19,80
8	IC-Bauanleitungs-Handbuch	19,80
9	Feldeffekttransistoren	9,80
10	Elektronik und Radio	19,80
11	IC-NF Verstärker (i. V.)	9,80
12	Beispiele integrierter Schaltungen (BIS)	19,80
13	HEH, Hobby Elektronik Handbuch	9,80
15	Optoelektronik Handbuch	19,80
16	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	CMOS Teil 3, Entwurf und Schaltbeispiele	19,80
19	IC-Experimentier Handbuch	19,80
20	Operationsverstärker	19,80
21	Digitaltechnik Grundkurs	19,80
22	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	Progr. in Maschinensprache mit Z80, Band II	29,80
25	68000 Microcomputer Einführung (i. V.)	39,00
26	Mikroprozessor, Teil 2	19,80
27	BASIC-M Anwender-HB f. 6800/09/68000 (Motorola)	29,80
28	Lexikon + Wörterbuch f. Elektr. u. Mikroprozessor	29,80
29	Mikrocomputer Datenbuch	49,80
30	Floppy Disk Selbstbau-Handbuch (i. V.)	49,00
31	57 Programme in BASIC	39,00
33	Microcomputer Programmierbeispiele	19,80
34	TINY-BASIC Handbuch	19,80
35	Der freundliche Computer	29,80
103	Oszillographen-Handbuch	19,80
108	Rund um den Spectrum (Progr., Tips und Tricks)	29,80
109	6502 Microcomputer Programmierung	29,80
110	Programmierhandbuch für PET	29,80
111	Programmieren mit TRS-80 (Video Genie)	29,80
112	PASCAL-Programmier-Handbuch	29,80
113	BASIC-Programmier-Handbuch	19,80
114	Der Microcomputer im Kleinbetrieb	39,80
115	6809 Programmier Handbuch (i. V.)	49,00
116	Einführung 16-Bit Micorocomputer	29,80
117	FORTRAN für Heimcomputer	19,80
118	Programmieren in Maschinensprache mit dem 6502	49,00
119	Programmieren in Maschinensprache (Z80) Band I	39,00
120	Anwenderprogramme für TRS-80 u. Video Genie	29,80
121	Microsoft BASIC-Handbuch	29,80
122	BASIC für Fortgeschrittene	39,00
123	IEC-Bus Handbuch	19,80
124	Programmieren i. Ma.-Sprache mit Commodore-64	29,80
127	Einführung i. d. Microcmputer-Progr. mit 6800	49,00
128	Programmieren mit dem CBM	29,80
130	Programmierbeispiele für CBM	19,80
132	CP/M-Handbuch	19,80
133	Handbuch für MS/DOS (i. V.)	29,80
137	FORTH Handbuch	49,00
139	BASIC für blutige Laien	19,80
140	Progr. i. BASIC u. Maschinencode mit dem ZX81	29,80
141	Programme f. VC-20 (Spiele, Utilities, Erweiterungen)	29,80
143	35 Programme für den ZX81	29,80
144	33 Programme für den ZX-Spectrum	29,80
145	64 Programme für den Commodore 64	39,00
146	Hardware-Erweiterungen für den C-64 (i. V.)	39,00
147	Beherrschen Sie Ihren Commodore 64	19,80
148	Programmierhandbuch für SHARP	49,00
149	Programme für TI 99/4A	49,00
175	Astrologie auf dem ATARI 800	49,00
8029	Z-80 Assembler-Handbuch	29,80
	Bücher in englischer Sprache 1. Von ELCOMP Publishing, Inc., Los Angeles, CA.	
150	Care and Feeding of the Commodore PET	19,80
151	8K Microsoft BASIC Reference Manual	9,80
152	Expansion Handbook for 6502 and 6800	19,80
154	Complex Sound Generation using the SN76477	9,80
156	Small Business Programs	29,80
158	The Second Book of Ohio Scientific	19,80
159	The Third Book of Ohio Scientific	19,80
160	The Fourth Book of Ohio Scientific	29,80
161	The Fifth Book of Ohio Scientific	19,80
162	ATARI Games in BASIC	19,80
163	The Peripheral Handbook (i. V.)	29,80
164	ATARI-BASIC Learning by Using	19,80
166	Programming in 6502 Machinelanguage PET/CBM	49,00
169	How to Progr. your ATARI in 6502 Machinelanguage	29,80
170	FORTH on the ATARI – Learning by Using	29,80
171	See the Future with your ATARI (Astrology)	49,00
172	Hackerbook I (Tricks + Tips for your ATARI)	29,80
173	PD-Program Descriptions (ATARI)	9,80
174	ZX-81/TIMEX Progr. i. BASIC a. Machine Lang.	29,80
176	Programs + Tricks for VIC's	29,80
177	CP/M – MBASIC and the OSBORNE	29,80
178	The APPLE in Your Hand	39,00
182	The Great Book of Games Vol. I - Games f. the C-64	29,80
183	More on the Sixtyfour	39,00
	Riesenprogrammsammlung in BASIC	
8048	BASIC Software Vol. VI	199,00
8049	BASIC Software Vol. VII	159,00
8050	BASIC Software Vol. I	99,00
8051	BASIC Software Vol. II	99,00
8052	BASIC Software Vol. III	149,00
8053	BASIC Software Vol. IV	39,00
8054	BASIC Software Vol. V	39,00

Der Hofacker Verlag produziert und vertreibt neben einer sehr großen Auswahl an Fachbüchern für Elektronik und Microcomputertechnik noch:

– Leerplatinen und Bauanleitungen für Zusatzeinrichtungen für Ihren Personalcomputer, sowie
– Programme (Software) für die bedeutenden Personalcomputer.

(i. V. bedeutet: Buch ist in Vorbereitung)

HOFACKER

HOLZKIRCHEN **SINGAPORE** **LOS ANGELES**

ISBN 3-921682-48-7