
# EINFÜHRUNG

## in die Programmiersprache

# CBASIC-86

## in Verbindung mit dem

## Microcomputer-System

# OLYMPIA PEOPLE

Diese Einführung wurde nach den Anforderungen und mit Unterstützung des OLYMPIA-Bildungszentrums Wilhelmshaven entwickelt von

Dipl. Ing. W. Eggerichs / Wilhelmshaven

Wilhelmshaven, Oktober 1983

Diese Unterlage wird im Rahmen des Trainings CBASIC-86 als Schulungsunterlage ausgegeben und ist in den Trainingsgebühren enthalten.

Zusätzliche Exemplare können zum Preis von DM 48,-- + MWSt erworben werden, unter unserer Adresse

OLYMPIA WERKE AG
Bildungszentrum
Abt. S 22/1
Postfach 960

2940 Wilhelmshaven

Tel.: 04421 / 783170
oder 04421 / 783181

Hinweis:

In den Trainings CBASIC-86 wird außer dieser Unterlage auch ein Source-Listing aller hier beschriebenen Beispielprogramme ausgegeben.

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

# VORWORT

Bei dieser Einführung in die Programmiersprache CBASIC des Mikrocomputersystems PEOPLE handelt es sich um ein Schulungshandbuch für die Programmiersprache CBASIC, um eine Schulung dieser Programmiersprache im Rahmen eines Seminars oder eines Selbststudiums zu ermöglichen.
Die Reihenfolge der Befehlsdarstellung und auch die Schwerpunkte innerhalb der Darstellung weichen aus diesem Grund zum Teil erheblich von der englischsprachigen Original-Beschreibung ab.
Insbesondere wurde darauf Wert gelegt, daß erlernte Befehlsgruppen in kleinen Beispielprogrammen direkt am PEOPLE-System in die Praxis umgesetzt werden können.

Diese Schulungsunterlage soll und kann die Original-CBASIC-Beschreibung nicht vollständig ersetzen, da für die Darstellung des Leistungsumfangs der Programmiersprache CBASIC nur die Original-Beschreibung gültig ist.

Um die Kommando-Syntax der CBASIC-Anweisungen eindeutiger und verständlicher zu beschreiben, wurden innerhalb dieser Einführung alle Anweisungen in Syntax-Diagrammen dargestellt.
Syntax-Diagramme selber werden im Abschnitt 1. kurz erläutert.

Zusätzlich muß gesagt werden, daß einige CBASIC-Bereiche (wie z.B. die Dateiverwaltung) auf Grund des zeitlichen Rahmens von 40 Unterrichtsstunden nicht ausreichend in Beispielprogrammen behandelt werden können.

Einführung in CBASIC

## 1. Allgemeines zur Programmiersprache CBASIC und zu dieser Spracheinführung

- CBASIC ist eine kaufmännisch / verwaltungstechnisch orientierte Variante der allgemein bekannten Programmiersprache BASIC.
  Sie läuft auf dem PEOPLE-System als Compiler/Interpreter Version und erzielt hierdurch optimale Durchführungszeiten bei einem Minimum an Programm-Code.

- Jedes CBASIC-Programm besteht aus einer Sequenz von Anweisungen, die entsprechend ihrer Reihenfolge vom Rechner bearbeitet werden.

- Unterschiede zu anderen BASIC-Versionen:

  - Zeilennummern müssen nur dann angegeben werden, wenn sie als "Ansprungmarke" benötigt werden.
    Wenn Zeilenummern vergeben werden, brauchen sie nicht in aufsteigender Reihenfolge verwendet zu werden und können als Ganzzahl, als Gleitkommazahl oder in der Exponentialdarstellung eingegeben werden.
    Folgende Zeilennummern sind korrekt und nicht identisch:

    120 120.0 120.234 1.2E02

  - In CBASIC werden 2 numerische Datentypen unterschieden, die in den folgenden Paragraphen noch näher erläutert werden.

  - CBASIC enthält zusätzliche Sprachelemente zur strukturierten Programmierung.

  - Variablennahmen können in CBASIC max. 31 Zeichen lang sein und demzufolge selbstdokumentierend gewählt werden.

  - CBASIC-Anweisungen dürfen über mehrere Zeilen verteilt geschrieben werden, indem eine Zeile mit dem "Fortsetzungszeichen" (\) beendet wird.
    Auch hierdurch wird die Darstellung einer Programmstruktur erleichtert.

  - In einer Zeile dürfen mehrere CBASIC-Anweisungen stehen, wenn sie durch das Zeichen ":" getrennt werden.

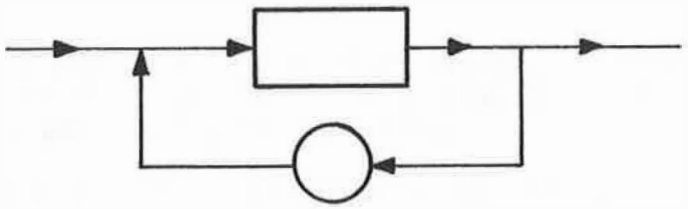
- Bei der Darstellung von Flußdiagrammen wurde abweichend von der DIN 66001 mit "Richtungspfeilen" gearbeitet.

- Diese Spracheinführung wurde als Unterrichtskonzept unter Einbeziehung praktischer Arbeit am PEOPLE-System nach methodischen und didaktischen Gesichtspunkten zusammengestellt.
  Aus diesem Grund sollte bei anschließenden Programmerstellungen zusätzlich die Original-CBASIC-Beschreibung benutzt werden.

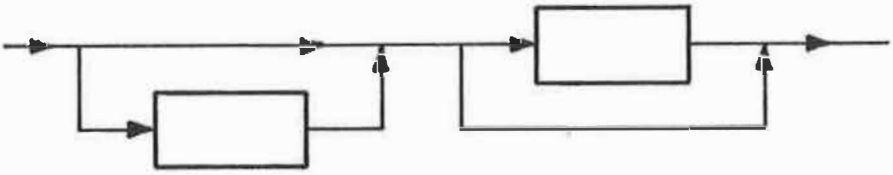
- Innerhalb dieser Spracheinführung erfolgt die Darstellung der CBASIC-Anweisungen in Form von Syntax-Diagrammen.

"leere" Beispiele zu Syntax-Diagrammen:

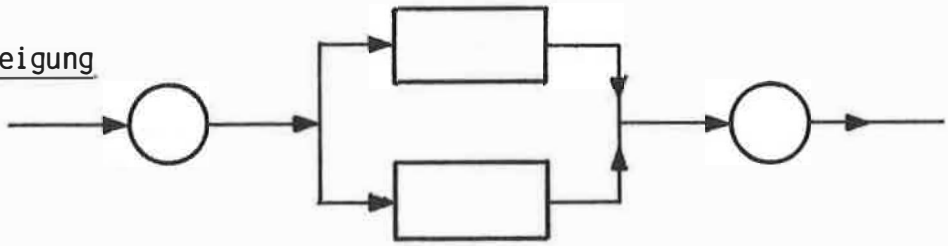
Wiederholung



Auslassung



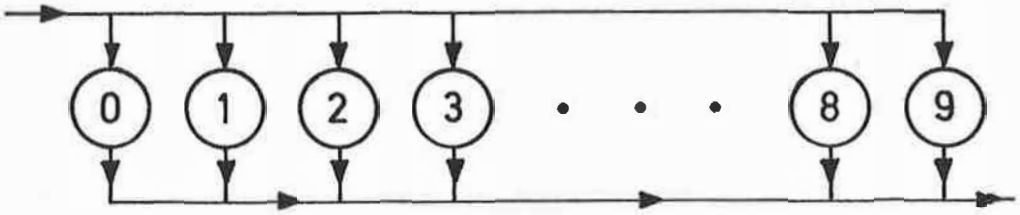
Verzweigung



alle in runden Flächen (Kreise oder Ovale) dargestellten Wortsymbole müssen in einem Kommando direkt übernommen werden.
Die in Rechtecken stehenden Begriffe verweisen auf andere Diagramme oder Erläuterungen und müssen durch aktuelle Begriffe des geforderten "Typs" ersetzt werden.

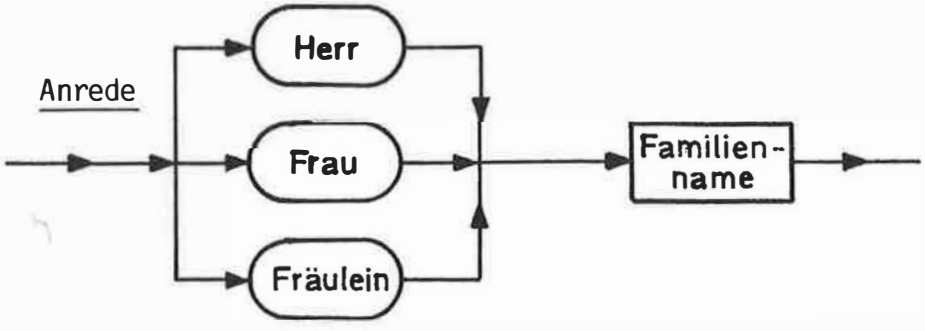
Beispiele für Syntax-Diagramme:

Ziffer




Postleitzahl


Anrede



# 2. Datentypen, Variablenbenennung, Ausdrücke und numerische Funktionen

## 2.1 Der Datentyp STRING

STRING-Konstanten sind Zeichenketten, die alle erlaubten alphanumerischen Zeichen enthalten können und durch das Zeichen (") eingeschlossen sind. STRING's dürfen zwischen 0 und 255 Zeichen lang sein.

Intern wird die Länge des STRING's (in einem Byte) und anschließend die Zeichenkette selber abgespeichert ( 1 Byte je Zeichen).

Beispiele:
```
"Vorname"   ,   "Datum : "
"Geben Sie Ihren Namen ein : "
"* 1450.76 DM"
" ZAHL = "
```

STRING-Variable müssen als letztes Namenszeichen ein $ enthalten.

## 2.2 Der Datentyp Numerisch

In CBASIC existieren die beiden numerischen Datentypen INTEGER und REAL.

INTEGER-Zahlen sind Ganzzahlen in einem bestimmten Zahlenbereich, während alle anderen Zahlen als REAL-Zahlen (Gleitkommazahlen) dargestellt werden.

### 2.2.1 INTEGER-Zahlen

INTEGER-Zahlen werden intern mit 16 Bit als Zweier-Komplement dargestellt. Hierdurch entsteht ein anwendbarer Zahlenbereich von

-32768 bis +32767 .

INTEGER-Konstanten können dezimal, hexadezimal oder binär angegeben werden.

Soll eine INTEGER-Konstante binär angegeben werden, muß der Konstanten der Buchstabe B folgen. Soll eine Konstante hexadezimal interpretiert werden, muß der Zahl der Buchstabe H nachgestellt werden. Dabei darf die erste Hexadezimalziffer kein Buchstabe sein.

Das negative Vorzeichen einer INTEGER-Konstanten muß mitgeschrieben werden, das positive Vorzeichen kann mitgeschrieben werden.

Beispiele:
```
3      ,   32 100  ,   10 1 1B
0FC 1H  ,   73F4H   ,   -234    ,   +5DE4H
- 100 1 1 1B  ,   +56
```

Der Name einer INTEGER-Variablen muß mit einem Prozentzeichen (%) zur Datentypkennung beendet werden.

### 2.2.2 REAL-Zahlen

REAL-Zahlen werden in CBASIC intern in 8 Byte dargestellt. Hierbei wird im ersten Bit des ersten Byte das Vorzeichen der Zahl abgelegt. Die restlichen 7 Bit enthalten den Exponenten zur Basis 10 einschließlich seines Vorzeichens ( 10 hoch +63 bis 10 hoch -64).
Die übrigen 7 Byte enthalten eine normierte, 14-ziffrige BCD-Zahl. Durch diese interne Darstellungsform von Zahlen kann sowohl die kommerzielle Forderung nach Zifferngenauigkeit in der Darstellung und der Arithmetik erfüllt werden, als auch dem technisch/ wissenschaftlichen Bedarf nach sehr großen und sehr kleinen Zahlen (z.B. 3.6E-24) nachgekommen werden.
Allerdings ist die Rechengeschwindigkeit bei umfangreichen mathematischen Ausdrücken mit REAL-Zahlen deutlich herabgesetzt.

Zahlen-Konstanten, die keinen Dezimalpunkt enthalten, jedoch nicht mehr im INTEGER-Zahlenbereich liegen, werden als REAL-Zahl abgelegt.

REAL-Zahlenbereich: 1.0 E-64 bis 0.99999999999999 E+63

Beispiele:
```
1.0     ,   345.6783    ,   ,-13467.80
-99.3   ,    7.5E3      ,    -1.5E-03
+56000  ,   -12300055    ,    0.00000067
```

Namen von REAL-Variablen benötigen kein besonderes Datentyp-Kennzeichen.

## 2.3 Variablen-Namen in CBASIC

Unter Variablennamen versteht man die symbolische Benennung von Speicherplätzen (im Arbeitsspeicher) für zu verarbeitende Daten.
Diese Namensvergabe unterliegt gewissen Regeln:

- Jeder Namen (Identifier) beginnt mit einem Buchstaben
- Anschließend können beliebig viele Buchstaben, Ziffern oder Punkte verwendet werden.
- Nur die ersten 31 Zeichen werden zur Unterscheidung des Namens und des Datentyps vom Compiler verwendet.
- Endet der Variablenname mit einem Prozentzeichen, handelt es sich um eine INTEGER-Variable, endet er mit einem Dollarzeichen ($), handelt es sich um eine STRING-Variable. Alle anderen Namen werden als REAL-Variable interpretiert.
- Der Variablenname darf nicht mit den Zeichen FN beginnen, da diese Zeichen für Funktionen reserviert sind.

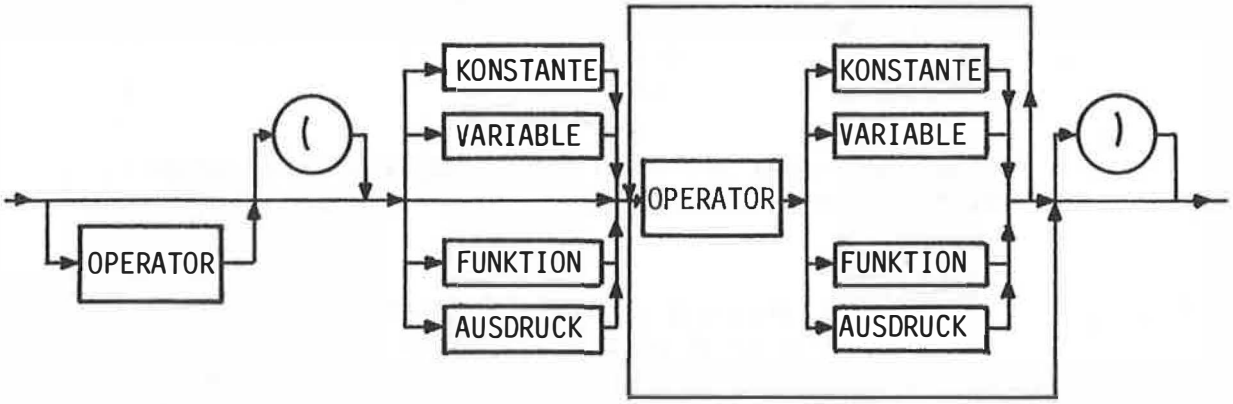
Aus dem Variablennamen sollte die Bedeutung (der Verwendungszweck) der Variablen erkennbar sein.

Beispiele:
```
NUMMER%     ,   ZAHL      ,   NAME$
TEXT$       ,   KUNDEN.Nr%    ,   X     ,   SUMME
DATEI.NAME$     ,   ORT$      ,   UMSATZ
STEUER      ,   NOTE%     ,   INDEX%
KLA.FAM.ART%
```

## 2.4 Bildung von Ausdrücken

Ausdrücke werden aus KONSTANTEN, VARIABLEN, FUNKTIONEN und OPERATOREN gebildet. Das Ergebnis eines Ausdrucks und auch die Parameter sind wiederum von einem der bereits erläuterten Datentypen INTEGER, REAL oder STRING.

AUSDRUCK:




Wird eine Klammer in einem Ausdruck geöffnet, muß sie auch wieder geschlossen werden!

Die Berechnungsreihenfolge innerhalb eines Ausdrucks ist durch die folgende Hierarchie der OPERATOREN festgelegt:

| Prior. | Darstellung | Datentyp | Bedeutung |
|---|---|---|---|
| 1 | (.....) | R, I, S | geklammerte Ausdrücke |
| 2 | ^ | R / I | Exponentation |
| 3 | * , / | R , I | Multiplikation + Division |
| 4 | + , -<br>+<br>+ , - | R , I<br>S<br>R , I | Addition und Subtraktion<br>Verknüpfung von STRING's<br>monadische Operatoren |
| 5 | <,<=,>,>=,=<br><> oder<br>LT,LE,GT,GE<br>EQ, NE | R , I , S | Vergleichsoperatoren<br>kleiner, kl./gleich, größer<br>gr./gleich, gleich, ungleich |
| 6 | NOT | I | log.Operator - Navigation |
| 7 | AND | I | log.Oper. UND (Konjunktion) |
| 8 | OR , XOR | I | log.Oper. ODER (Disjunktion)<br>log.Oper. Exclusiv-ODER |

- Werden INTEGER- und REAL-Zahlen mit einem für beide Datentypen zulässigen Operator verknüpft, wird die INTEGER-Zahl automatisch in eine REAL-Zahl umgewandelt und das Ergebnis ist vom Typ REAL. Bei diesen "Mixed-Mode-Operationen" wird zusätzliche Rechenzeit verbraucht.

- Bei der Exponentation darf die Basis nur dann negativ sein, wenn Basis und Exponent INTEGER-Zahlen sind.

- Sinnvollerweise darf ein Vergleich nur zwischen gleichen Datentypen stattfinden. Das Ergebnis eines Vergleichs ist immer vom Typ INTEGER.

  0 = FALSE
  -1 = TRUE

- Das Ergebnis von logischen Operationen ist immer vom Typ INTEGER.
  Sind Parameter vom Typ REAL, so werden sie zuerst nach INTEGER gewandelt.

- Findet bei einer REAL-Operation ein Überlauf (Overflow) des darstellbaren Zahlenbereichs statt, wird eine Warnung ausgegeben und die Programmausführung wird fortgesetzt, indem das Ergebnis auf die größte darstellbare REAL-Zahl gesetzt wird.
  Handelt es sich um eine INTEGER-Operation, erfolgt keine Überprüfung und keine Warnung im Falle eines Overflow.
  Das Ergebnis dieser Operation ist falsch!

Beispiele:

```
100,   2000
45.6 + 12 ,   1000 * 3
X ,   ANZ% + 1
ERG ,   -Nr ,   TEXT$
SUMME ,   ZAHL + X
NAME$ + " xxx" ,   WERT$ + ANT$
ZAHL * 3 - NR%
( X - Y )  + Z
"Name eingeben : " ,   "---------------"
- ( X ^4 )
10 < NR%   ,   ANZ% + NR% * ( A - 5)
( ZAHL1 * X / ZAHL2 ) ^ NR
ANTWORT$ <> "Nein"
WERT  >  1000 AND  WUNSCH$ = "ENDE"
```

## 2.5 Numerische Funktionen in CBASIC

Bei den hier aufgeführten Funktionen handelt es sich um die in der Programmiersprache CBASIC vordefinierten numerischen Funktionen, die immer dann verwendet werden können, wenn in einer Anweisung ein AUSDRUCK zugelassen ist.

| | Datentyp d. Fkt. | Datentyp d. Param. | Bedeutung | Beispiel |
|---|---|---|---|---|
| ABS(x) | REAL | REAL | Absolutwert von x | Y = ABS(ZAHL) |
| INT(x) | INT. | REAL | Ganzzahliger Teil von x. Nachkommastellen entfallen | I% = INT (ZAHL) |
| INT%(x) | INT. | REAL | Nachkommastellen werden gerundet | I% = INT% (ZAHL) |
| FLOAT(i) | REAL | INT. | Umwandlung INT. nach REAL | ERG=FLOAT(NR%) |
| SGN(x)<br>EXP(x)<br>SQR(x)<br>LOG(x) | INT.<br>REAL<br>REAL<br>REAL | R/In<br>REAL<br>REAL<br>REAL | Vorzeichen von x<br>e hoch x<br>Quadratwurzel von x<br>natürli. Logarithmus | VORZ%=SGN(ERG)<br>ERG=EXP(Y 1)<br>WURZ=SQR(SUMME)<br>XERG=LOG(Y) |
| SIN(x)<br>COS(x)<br>TAN(x)<br>ATN(x) | REAL<br>REAL<br>REAL<br>REAL | REAL<br>REAL<br>REAL<br>REAL | Sinus im Bogenmass<br>Cosinus im Bog.<br>Tangens im Bogenm.<br>Arcustangens | Y= SIN( 1+X*X)<br>AMPL=COS(PI-X)<br>WERT=TAN(Y 1+X 1)<br>YERG=ATN(TAN(WINK)) |
| RND<br>FRE | REAL<br>REAL | -<br>- | Zufallsfunktion<br>Freisprechergröße | ZUFALL=RND* 100<br>FREI=FRE- 1024 |

- Ist das Argument einer Funktion vom Typ INTEGER statt vom Typ REAL (oder umgekehrt), so findet vor der Berechnung eine automatische Umwandlung statt.

- Jede Funktion kann als Parameter in einer anderen Funktion aufgerufen werden, soweit die Datentypen miteinander verträglich sind.

- Außer den aufgeführten vordefinierten numerischen Funktionen können auch vom Programmierer selbst definierte Funktionen (siehe Abschnitt 4.12) oder auch vordefinierte String-Funktionen (siehe Abschnitt 4.13) im Programm verwendet werden.

Beispiele:

```
SIN (ZAHL) ,   FLOAT (INDEX%)
ABS ( SIN ( EXP ( 1 + X * WERT )))
INT ( SQR (1 + X * ))
```

## Ergebnisbeispiele zu Funktionsaufrufen

| Funktionsaufruf | Ergebnis |
|---|---|
| ABS ( -13.5 ) | 13.5 |
| INT ( 245.67 ) | 245 |
| INT ( -512.112 ) | -512 |
| INT ( 3.49 ) | 3 |
| INT ( 3.5 ) | 3 |
| INT ( -3.5 ) | -3 |
| INT% ( 245.67 ) | 246 |
| INT% ( -512.112 ) | -512 |
| INT% ( 3.49 ) | 3 |
| INT% ( 3.5 ) | 4 |
| INT% ( -3.5 ) | -4 |
| FLOAT ( 334 ) | 334 |
| SGN ( -12345 ) | -1 |
| SGN ( 33 ) | +1 |
| SGN ( 0 ) | 0 |
| EXP ( 1 ) | 2.718281.... |
| SQR ( 16 ) | 4 |
| SQR ( X * X ) | X |
| LOG ( EXP ( 1 )) | 1 |
| LOG ( 5 ) / LOG ( 10 ) | lg(5) = 0.69897 (zur Basis 10) |

kaufmännisches Runden:

```
FLOAT ( INT% (zahl * 100)) / 100

zahl = 3.494     >>>     3.49
zahl = 3.495     >>>     3.5
```

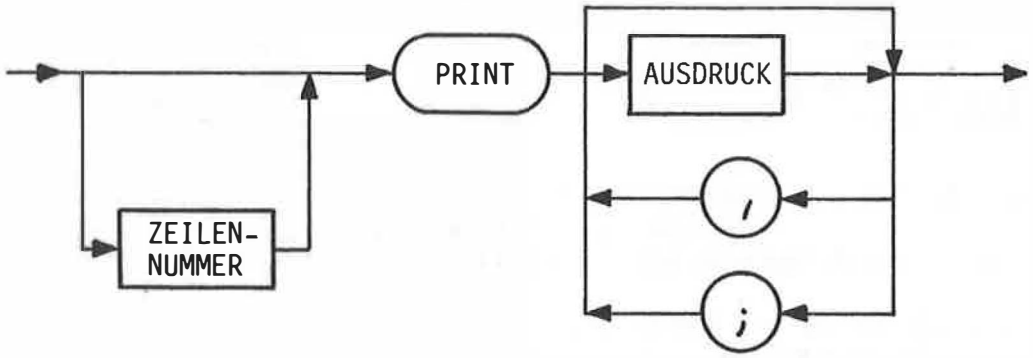
# 3. Anweisungen zur Ein- und Ausgabe

## 3.1 Die PRINT-Anweisung

Diese Anweisung dient der Ausgabe von Daten (Variable, Konstante, Funktionen und Ausdrücken) auf dem Bildschirm oder dem Drucker in einer wählbaren Form.

Anweisung:

[zeilennummer] PRINT [ausdruck] [trennzeichen] {ausdruck trennzeichen}




- Zahlen werden in Kolonnen zu je 15 Zeichen ausgegeben.
  Reicht dieser Bereich nicht aus, werden sie in wissenschaftlicher Notation ausgegeben.

- Ein Komma als Trennzeichen bewirkt die Weiterschaltung des Tabulators auf die nächste TAB-Position.
  TAB-Positionen sind alle Vielfachen von 20.
  Führt dies zu einem Überschreiten der maximalen Spaltenzahl, wird die Ausgabe auf der nächsten Zeile fortgesetzt.

- Ein Semikolon als Trennzeichen bewirkt ein Leerzeichen (BLANK) nach einer numerischen Ausgabe. Nach einem STRING (Variable oder Konstante) wird kein Leerzeichen ausgegeben.

- Jedes PRINT-Statement, das nicht mit einem Komma oder Semikolon endet, führt zu einem Zeilenvorschub.

- Wird ein PRINT-Befehl mit einem Komma oder Semikolon abgeschlossen, so wird diese Zeile nicht eher beendet, bis

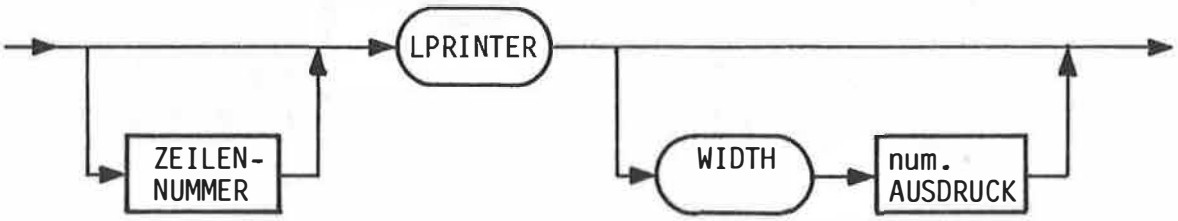
  a) ein weiterer PRINT-Befehl, der nicht mit einem (,) oder (;) endet, folgt.
  b) die Anzahl der Spalten pro Zeile geändert wird.
  c) ein LPRINTER oder CONSOLE-Statement folgt.
  d) ein STOP-Statement folgt.

Beispiele:

```
PRINT "Dieses ist ein Text"
PRINT   :   PRINT ,,
PRINT KLASSE.EINS%, ZAHL 1; ZAHL2;
PRINT "DATUM : ";TAG%;".";MONAT%;".";JAHR%
PRINT "Ergebnis = ";  A * B + Z
PRINT "Name       : ";NAME$
```

## 3.2 Die LPRINTER-Anweisung

Dieses Kommando bewirkt, daß alle folgenden PRINT-Anweisungen auf dem am PEOPLE-System angeschlossenen Drucker ausgegeben werden.




- Mit dem Parameter WIDTH kann die Druckbreite (Anzahl der Spalten pro Zeile) vom Programm aus definiert werden.
  Die implizite Voreinstellung für die Drucker-Ausgabe ist 132.

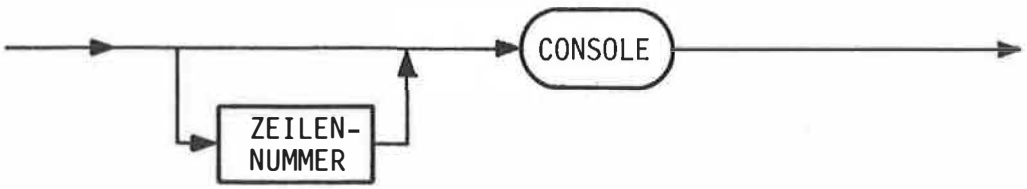
- Ist die Bildschirm-Cursor Position zum Zeitpunkt der LPRINTER-Anweisung ungleich 1, wird ein RETURN und ein Zeilenvorschub am Bildschirm ausgegeben.

Beispiele:

```
    LPRINTER
    LPRINTER WIDTH  BREITE%
300 IF  OUTPUT% THEN LPRINTER WIDTH 80
```

## 3.3 Die CONSOLE-Anweisung

Hiermit wird die Wirkung der LPRINTER-Anweisung wieder aufgehoben. Alle jetzt folgenden Ausgaben erscheinen wieder auf dem Bildschirm.




- Ist die Druckposition zum Zeitpunkt der CONSOLE-Anweisung ungleich 1, wird ein RETURN und ein Zeilenvorschub auf dem Drucker ausgegeben.

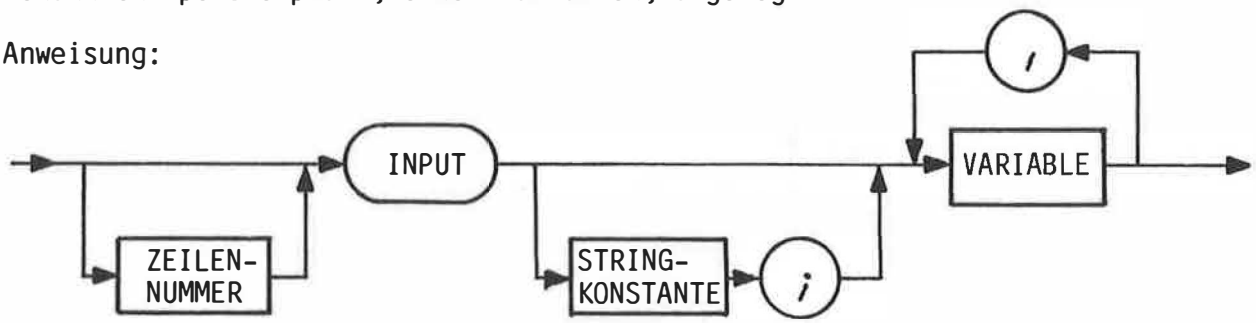
Beispiele:

```
    CONSOLE
410 CONSOLE
    IF  B% < 24  THEN  CONSOLE
```

## 3.4 Die INPUT-Anweisung

Dieser CBASIC-Befehl dient der Dateneingabe durch den Programmanwender während der Programmlaufzeit.
Die einzulesenden Daten (Strings und/oder Zahlen) werden auf einem symbolisch benannten Speicherplatz, einer Variablen, abgelegt.

Anweisung:




- Ist die STRING-Konstante in der Anweisung enthalten, wird sie am Bildschirm ausgegeben, andernfalls wird ein Fragezeichen ausgegeben.
  Danach wird in beiden Fällen ein Leerzeichen ausgegeben.
  Diese Ausgaben erfolgen immer am Bildschirm, auch wenn durch die Anweisung LPRINTER eine Umschaltung auf den Drucker aktiviert ist.

- Es können sowohl einfache Variable als auch indizierte Variable (siehe Abschnitt 4.8) eingelesen werden.

- Die gesamte Dateneingabe kann aus maximal 255 Zeichen bestehen.
  Hierbei werden alle (!!) eingegebenen Zeichen mitgezählt!

- Alle CP/M-Steuerzeichen (wie CTRL-C oder CTRL-Z) behalten auch im INPUT-Kommando ihre CP/M-Funktion.

- Vom Anwender einzugebende Daten müssen durch Kommata voneinander getrennt sein. Dies gilt sowohl für Zahlen als auch für Strings. Werden Strings durch das Zeichen (") eingeschlossen, können diese Strings auch die Zeichen (,) und (") enthalten.

- Wird eine INTEGER-Zahl erwartet und der Anwender gibt eine REAL-Zahl ein, so findet eine Rundung wie bei der Funktion INT% statt.

- Werden mehr oder weniger Daten eingegeben, als in der INPUT-Anweisung Variable gefordert werden, erfolgt eine Warnung am Bildschirm und die Eingabe der Daten muß wiederholt werden.
  Dies gilt auch dann, wenn nur < CR > als einzige "Dateneingabe" erfolgt.

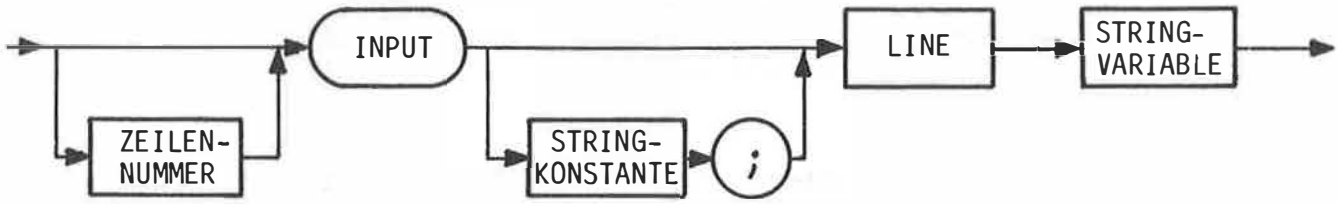
  SYSTEM-Warnung : < IMPROPER INPUT - REENTER >

Beispiele:

```
INPUT "Welches Ausgabegerät wünschen Sie (B/D) : ";AUS%
INPUT  ZAHL1%, ZAHL2%, OPZ$
INPUT "4 Zahlen eingeben : ";X1, X2, X3, X4
INPUT "Namen eingeben    : "; NAME$
```

## 3.5 Die INPUT-LINE-Anweisung

Diese Anweisung entspricht grundsätzlich der unter 3.4 beschriebenen INPUT-Anweisung.
Allerdings ist diese Anweisung dafür gedacht, um beliebige Eingabedaten in eine Stringvariable eingeben zu können.




- Nach < LINE > kann nur eine STRING-Variable stehen, der alle Eingabezeichen bis zum < RETURN > zugewiesen werden.

- Bei der Eingabe ist auch ein Null-String (Leer-String) zulässig, also die direkte Eingabe von < CR >.

- Auch die CP/M-Steuerzeichen wirken während der Dateneingabe.

Beispiele:
```
INPUT "Noch einmal  ? "; LINE  ANT$
INPUT "Weiter mit < RETURN > ! "; LINE  CON$
```

## 3.6 Die Funktion POS

Mit Hilfe dieser vordefinierten Funktion kann die nächste Ausgabeposition (Spalte) des Bildschirms oder des Druckers erfragt werden.


- Der von der Funktion POS gelieferte Zahlenwert kann von 1 bis zur augenblicklich definierten Zeilenbreite (WIDTH) reichen.

- Achtung: Es werden alle in dieser Zeile bisher ausgegebenen Zeichen mitgezählt, also auch entsprechende Bildschirm- oder Drucker-Steuerzeichen.

Beispiele:
```
IF  POS  > 50  THEN  PRINT
PRINT  "Dieses ist die "; POS ; ". Spalte"
IF  (POS+I%*2)  < DEV%  THEN PRINT  STRICH$
```

## 3.7 Die TAB-Funktion

Die Funktion TAB ermöglicht es dem Programmierer, im PRINT-Befehl eine bestimmte Ausgabeposition anzusteuern.


- Ist der Ausdruck nach TAB vom Typ REAL, findet zuerst eine Konvertierung nach INTEGER statt.
  Ist der errechnete INTEGER-Ausdruck kleiner oder gleich der momentanen Ausgabeposition, so wird ein RETURN und ein Zeilenvorschub durchgeführt und erst dann erfolgt die Positionierung auf die geforderte TAB-Position.
  Überschreitet der Ausdruck die definierte Zeilenbreite, so wird ein Fehler gemeldet.

Beispiele:

```
PRINT  TAB(22);"Vorname"
PRINT TAB(I1%); "Montag"; TAB(I2%); "Dienstag";
PRINT TAB(SIN(X+BAS)*0.02; "*"
```

## 3.8 Die Eingabe-Funktion CONCHAR%

Diese Funktion liest ein (1) Zeichen von der Konsole und übergibt den dezimalen Wert des eingelesenen ASCII-Zeichens.

Hiermit können z.B. Pausen programmiert werden, die erst dann beendet werden, wenn der Anwender eine beliebige Taste drückt oder es kann eine schnelle Überprüfung eines eingegebenen Zeichens erfolgen, ohne daß es einer zusätzlichen INPUT-Anweisung bedarf.

Auch bei dieser Funktion wirken die bekannten CP/M-Steuerzeichen wie z.B. CTRL-C oder CTRL-Z !

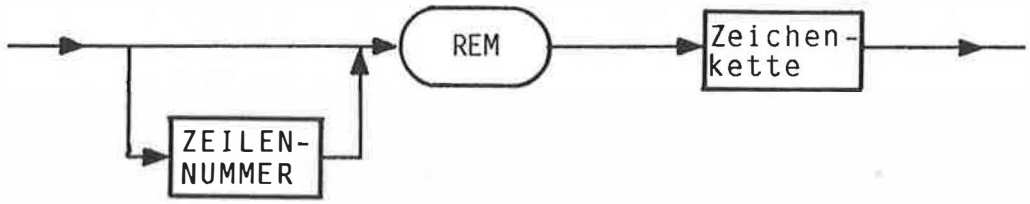
Beispiele:

```
    WEITER% = CONCHAR%
    IF  CONCHAR% = 13   THEN .....
20  IF  CONCHAR% <>  ASC ("J")   THEN  20
```

# 4. Allgemeine CBASIC-Anweisungen

## 4.1 Die REMARK-Anweisung

Diese Anweisung dient dem Einfügen von Kommentarzeilen in das Quellprogramm. Kommentare (Remarks) sollen das Programm erläutern und die "Lesbarkeit" erhöhen.




- Kommentare sind nur für den/die Programmierer von Bedeutung. Der Programm-Anwender bekommt sie nicht zu sehen.

- REM-Anweisungen sollten großzügig benutzt werden, da sich hierdurch der Aufwand für die Programmdokumentation verringert.

- Die eigentliche Programmgröße wird durch die Kommentarzeilen nicht beeinflußt, da dieses Statement nicht mit übersetzt wird.

- Wie aus dem Syntax-Diagramm erkennbar ist, dürfen REM-Anweisungen ebenfalls Zeilennummern enthalten, die von IF-, GOTO-, ON- und GOSUB-Anweisungen angesprochen werden können.

- Auch innerhalb einer Zeile dürfen REM-Statements jeder anderen Anweisung folgen, wenn vor dem REM keine Zeilennummer steht.

- Sollten mehrere Anweisungen, getrennt durch einen Doppelpunkt, in einer Zeile stehen, muß die REM-Anw. die letzte Anweisung in dieser Zeile sein.

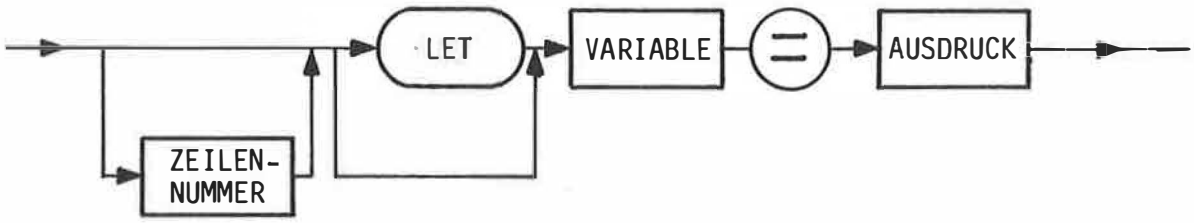
- Auch für den Befehl REM ist das Zeilen-Trennzeichen (\) anwendbar.

Beispiele:

```
      REM  Dieses ist ein Kommentar
      REM ---- ---- MODUL  "Ausgabeblock" ---------
      PRINT "........."    REM  Leerausgabe
      IF  ZEILE% > 70  THEN  PRINT  S$  REM  Seitenvorschub
   25 REM  AUFGABE 1  \
           In dieser Aufgabe sollen folgende Arbeiten  \
           erfüllt werden :  \
           a) ............................  \
           b) ..............................
```

## 4.2 Der Zuweisungsbefehl LET

Mit dieser Anweisung kann einer Variablen ein neuer Wert zugewiesen werden




- Der < AUSDRUCK > wird errechnet und der Variablen zugewiesen.
  Die Variable und der Ausdruck müssen beide vom gleichen Datentyp sein (String oder numerisch).

- Achtung: Es handelt sich hierbei um keine mathematische Gleichung.
  Die Anweisung X = X + 43.6 ist zulässig und korrekt.

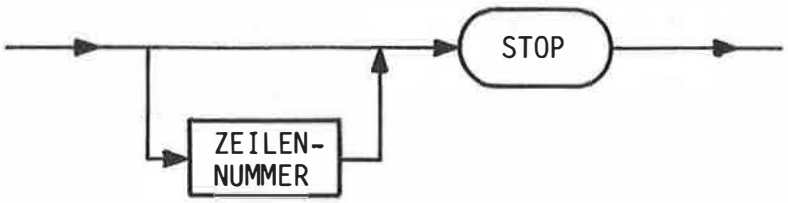
- Ist die Variable vom Typ INTEGER und der Ausdruck vom Typ REAL (oder umgekehrt), findet eine automatische Konvertierung in den Datentyp der Variablen statt.
  Wird einer INTEGER-Variablen ein REAL-Wert zugewiesen, werden vorhandene Nachkommastellen wie bei der Funktion INT% gerundet.

Beispiele:

```
      LET  A = B * WERT
200   LET  NAME$ = "MEIER"
      PI = 3.14159  :  STUNDE% = 0   :   E = EXP(1.)
      DATUM$ = TAG$ + "." + MONAT$ + "." + JAHR$
      LET  ANZ% = ANZ% + 1
      ERG = SIN(X) + COS (SQR (1 + X * X))
      NR% = INT (ERG)
```

## 4.3 Die STOP-Anweisung

Eine STOP-Anweisung bewirkt die Beendigung eines laufenden Programms.




- Diese Anweisung beendet das Programm, schließt alle offenen Dateien, beendet den Druckpuffer und übergibt die Kontrolle wieder an das Betriebssystem.

- Es können beliebig viele STOP-Statements im Programm enthalten sein. Aus Gründen der Übersichtlichkeit sollte jedoch nur eine STOP-Anweisung am Programmende stehen. Diese Anweisung kann bei Bedarf angesprungen werden.

## 1. Programmier-Aufgabe

Geben Sie das folgende Beispielprogramm mit Hilfe des EDITOR's in Ihren Mikrocomputer ein und führen das Programm anschließend durch.

Diese Programmier-Aufgabe soll Sie einerseits mit der Handhabung des PEOPLE-Systems vertraut machen und Ihnen andererseits die bisher erlernten CBASIC-Anweisungen in ihrer Wirkung demonstrieren. Achten Sie darum bitte besonders bei der (mehrfachen) Programmdurchführung auf die unterschiedlichen Reaktionen und Wirkungen der einzelnen CBASIC-Kommandos.

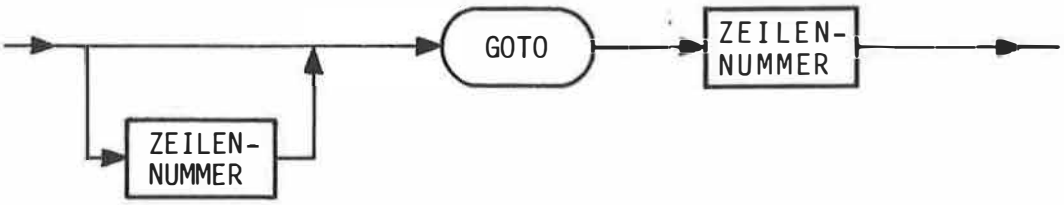
Achten Sie bereits bei der Programmeingabe auf jedes Komma und Semikolon und auf die entsprechenden Folgen während der Programmlaufzeit.

Aufgabe 1:

```
      REM ----------------- Aufgabe 1 --------------------
      REM  Diese Zeile ist eine Kommentarzeile
      PRINT "Dieses Programm liest Daten ein und gibt";  \
      " sie zur Kontrolle wieder aus."
      PRINT:PRINT:PRINT
      INPUT "ZAHL eingeben : ";ZAHL 1
      PRINT "1.ZAHL =";ZAHL 1,INT%(ZAHL 1)
20    REM diese Zeilennummer wird nicht gebraucht
      INPUT "Beliebige Zahl eingeben :";ZAHL2%
      PRINT "2.ZAHL = ";ZAHL2%
      LET ERG=ZAHL 1+ZAHL2% : LET ERG%=INT(ZAHL 1)+ZAHL2%
      LPRINTER WIDTH 50
      PRINT "Die Summe der beiden Zahlen = ";ERG%;" bzw. "  \
      ;ERG
      LET T$="#"
      INPUT "beliebiges Zeichen eingeben : ";Z$
      PRINT Z$;Z$;Z$,Z$;Z$;T$;
      PRINT Z$,Z$;Z$;Z$,T$
      PRINT "Umschaltung zurück auf den Bildschirm"
      CONSOLE
      PRINT "Ausgabe erfolgt jetzt wieder auf dem Bildschirm"
      STOP
```

## 4.4 Der absolute Sprungbefehl GOTO

Durch diese Anweisung ist es möglich, den sequentiellen Programmablauf zu verlassen und die Programmdurchführung an einer bestimmten Programmzeile fortzusetzen.




- Statt dem Wortsymbol GOTO kann auch GO TO geschrieben werden.

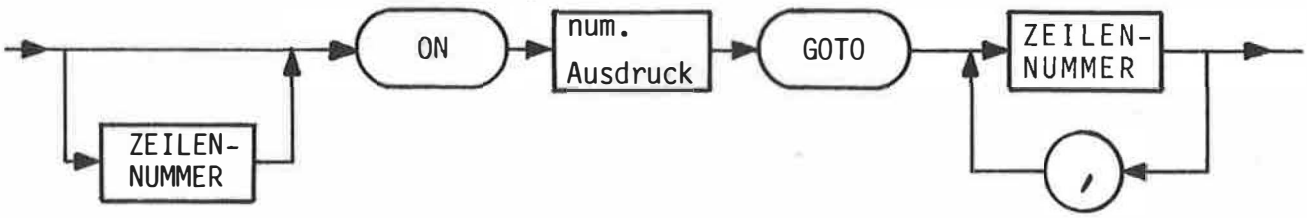
- Dieser absolute Sprungbefehl sollte so wenig wie möglich verwendet werden, da durch häufige und unkritische Benutzung dieser Anweisung ein Programm unverständlich und unübersichtlich gemacht werden kann.
  Die sparsame Verwendung der GOTO-Anweisung gilt für CBASIC ganz besonders, da bei Anwendung der in der Sprache implementierten Struktur-Anweisungen IF-THEN-ELSE (siehe 4.6) und WHILE-WEND (siehe 4.8) der absolute Sprungbefehl in den meisten Fällen überflüssig ist.

- Die hinter GOTO stehende Zeilennummer muß selbstverständlich vor der Anweisung stehen, die entsprechend der Programmlogik als nächste durchgeführt werden soll.

Beispiele:
```
    GOTO 100
    GO TO 565.34
20  GOTO 100
    GOTO  34.5E 10
```

## 4.5 Der berechnete Sprungbefehl ON-GOTO

Beim "berechneten Sprung" findet in Abhängigkeit vom Wert eines Ausdrucks eine Verzweigung zu einer bestimmten Zeilennummer statt.

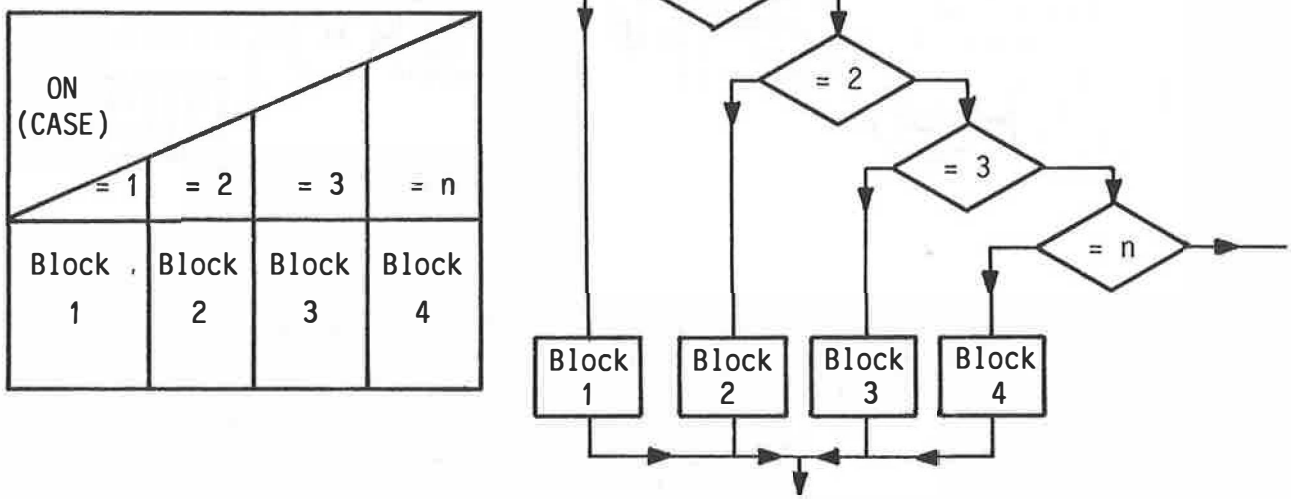



- Der hinter ON stehende numerische Ausdruck wird ausgewertet.
  Handelt es sich um einen REAL-Ausdruck, findet eine Umwandlung zum Datentyp INTEGER statt.
  Ist der Ausdruck gleich 1, wird zur ersten Zeilennummer nach GOTO verzweigt. Ist der Ausdruck gleich 2, zur zweiten Anweisungsnr. usw.

- Ist der Wert des Ausdrucks kleiner als 1 oder als die Anzahl der angegebenen Zeilennummern, erfolgt eine Fehlermeldung.

- Für die Anzahl der hinter GOTO möglichen Zeilennummern gibt es keine starre Grenze. Man kann jedoch von mindestens 32 möglichen Zeilenummern ausgehen, so daß keinerlei Programmiereinschränkungen zu erwarten sind.

Logische Darstellung:




Beispiele:

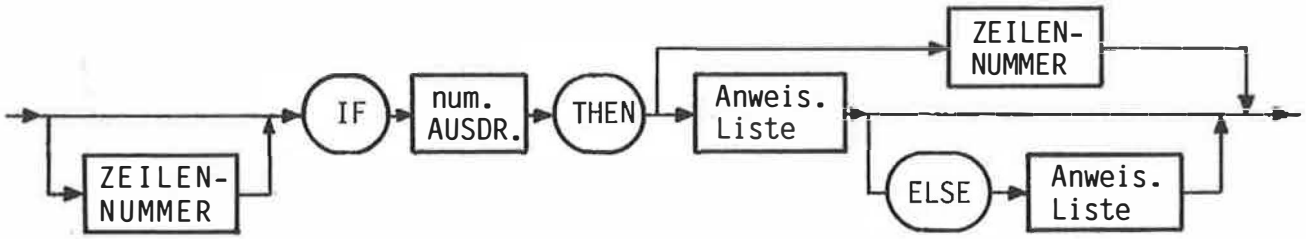
```
    ON  I% * 3 - NR% GOTO 55, 10, 200, 90
    ON  NUMMER%  GOTO  100, 200, 200, 400, 100
135 ON  ANTWORT + ISI  GOTO  1000, 40.5, 10.2
```

Programmbeispiel:

```
 10 INPUT "Gewünschte Modulnummer ? ";NR%
    IF NR% < 1  OR  NR% > 4   THEN GOTO 10
    ON  NR%   GOTO  50,100,150,200
 50 REM  Programm-Modul 1
    .....................
    GOTO 999
100 REM  Programm-Modul 2
    ...........................
    .....................
    GOTO 999
150 REM  Programm-Modul 3
    LET ..............
    PRINT ..............
    GOTO 999
200 REM  Programm-Modul 4
    LET ............
    IF ..............
    INPUT ..........
    GOTO 999
    ..............................
999 STOP
```

## 4.6 Die Verzweigungsanweisung IF-THEN-ELSE

Mit Hilfe dieser Anweisung kann eine Verzweigung in 2 (!) verschiedene Programmblöcke erfolgen.




- Beim < AUSDRUCK > nach IF handelt es sich üblicherweise um einen logischen Ausdruck, in der Regel um einen Vergleichs-Ausdruck.
  Ist der Wert des Ausdrucks ungleich 0 (ungleich FALSE), wird die nach dem THEN stehende Anweisungsliste abgearbeitet.
  Ansonsten wird die Anweisungsliste nach ELSE (wenn vorhanden) oder die dem IF folgende Anweisung durchgeführt.

- Eine Anweisungsliste besteht aus einer oder mehreren Anweisungen. Besteht die Liste aus mehreren Anweisungen, so sind diese Anweisungen durch Doppelpunkt (:) voneinander zu trennen.

- Wird nach dem THEN direkt eine Zeilennummer angesprungen, handelt es sich um die in BASIC allgemein übliche Anweisungsform des IF.

- Der < AUSDRUCK > darf nicht vom Typ STRING sein.
  Handelt es sich um einen REAL-Ausdruck, wird das Ergebnis nach INTEGER gewandelt.
  Die sofortige Verwendung eines INTEGER-Ausdrucks reduziert die Ausführungszeit dieser Anweisung.

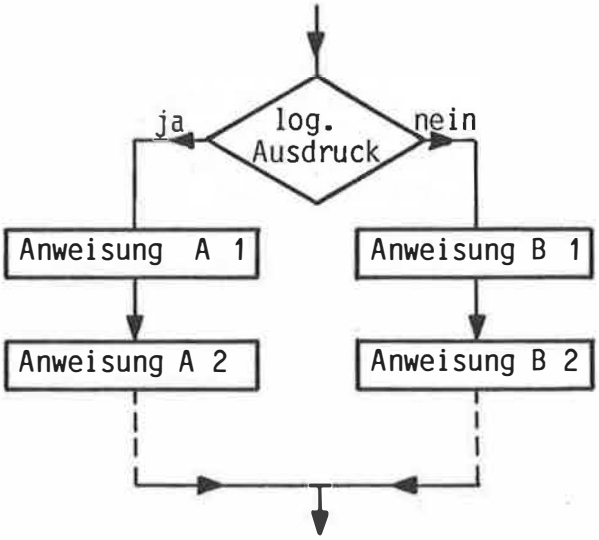
- In der Befehlsliste nach THEN bzw. nach ELSE sind folgende CBASIC-Anweisungen nicht zulässig:

  - DATA-Anweisung
  - DEF-Anweisung
  - IF-Anweisung
  - IF-END-Anweisung

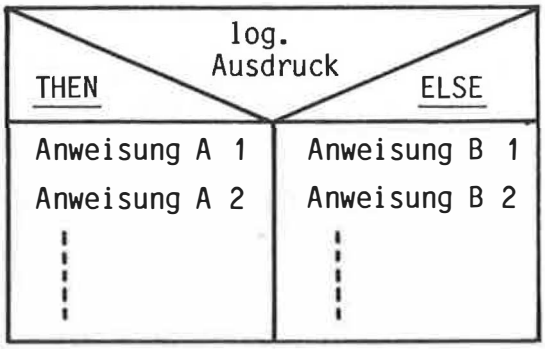
- Die in CBASIC mögliche Syntaxform der IF-Anweisung bietet sich ideal zur Umsetzung eines strukturierten Programmablaufplans an.
  Geschachtelte IF-Anweisungen dürfen hierbei allerdings nicht vorkommen.

Logische Darstellung:

FLUSSDIAGRAMM




STRUKTOGRAMM




Beispiele:

```
33 IF    (SUMME < 1000.) \
    THEN  INPUT "Zahl eingeben : "; ZAHL :\
          SUMME = SUMME + ZAHL : \
          ANZ% = ANZ% + 1 : \
          GOTO 33 \
    ELSE  MITTEL = SUMME / ANZ% : \
          PRINT "Der Mittelwert = "; MITTEL

IF  INDEX% + NR% <>   24  THEN  PRINT

IF  (NAME$ = "ENDE")  THEN  1000

IF  ANTWORT$ <>  "JA"   THEN GOTO 55 \
                        ELSE GOTO 66
```

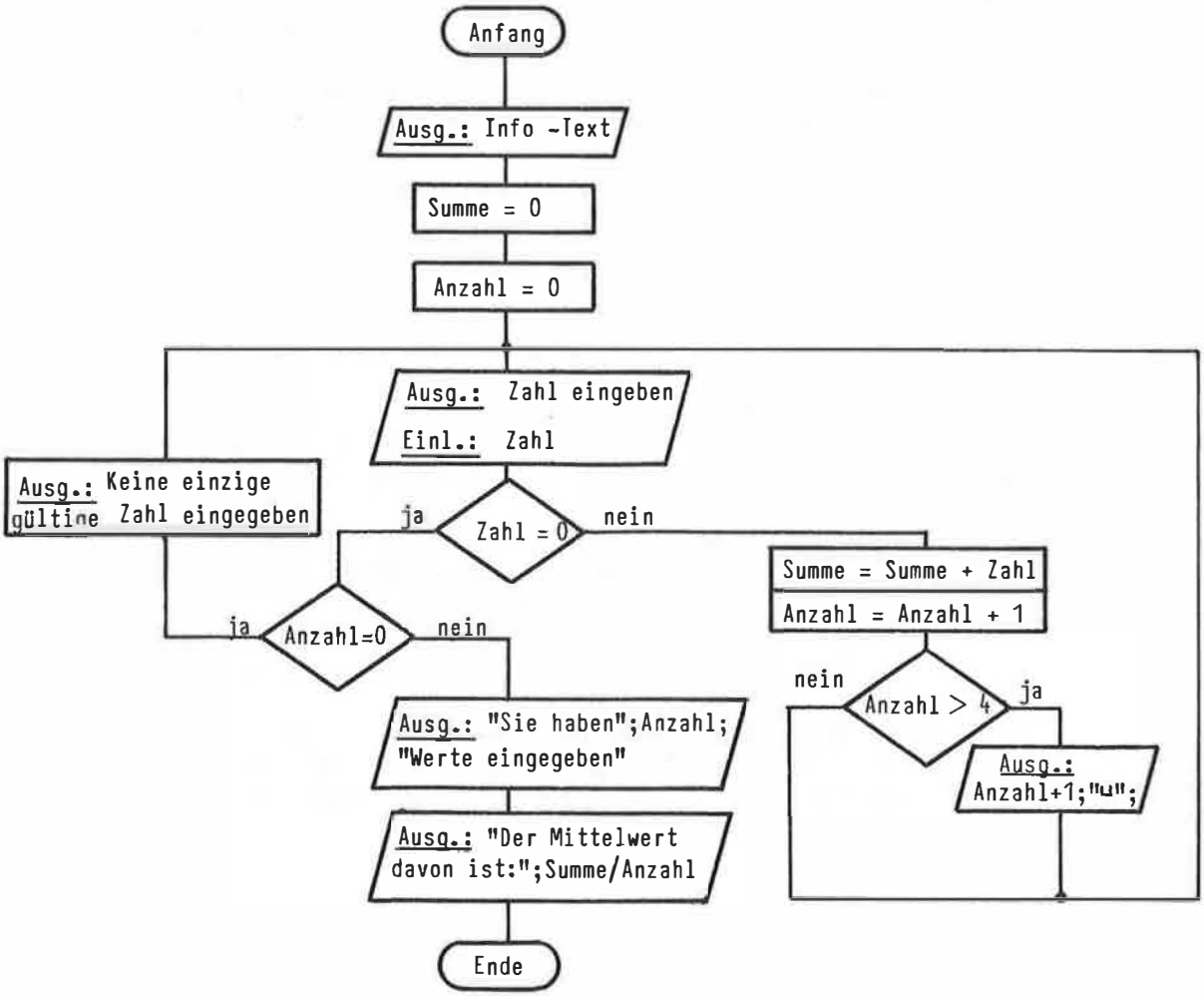
## 2. Programmier-Aufgabe

Ein Programm soll den arithmetischen Mittelwert einer vom Programm-Anwender einzugebenden Zahlenreihe errechnen.
Um den Anwender die Programmbenutzung zu erleichtern, soll ab der 6. Zahleneingabe nicht nur die Aufforderung zum Eingeben einer Zahl erfolgen, sondern auch die "Nummer" der Zahl am Bildschirm erscheinen.
Z.B. "8. Zahl eingeben : "

Das Einlesen weiterer Zahlen soll beendet werden, wenn vom Programmbenutzer eine NULL eingegeben wird.
Gibt der Anwender keine einzige gültige Zahl ein (also gleich als erste Zahl eine NULL), so soll eine entsprechende Fehlermeldung erfolgen und die erneute Eingabe einer Zahl gefordert werden.

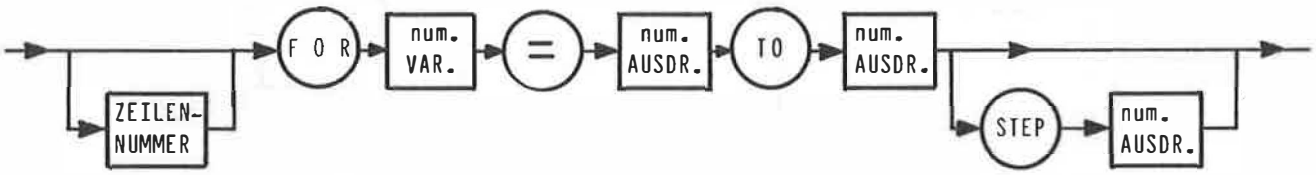
Verwenden Sie entsprechend dem Zahlenbedarf die Datentypen INTEGER und REAL.

FLUSSDIAGRAMM



## 4.7 Die Wiederholungsanweisung FOR-NEXT

Alle Anweisungen zwischen der FOR-Anweisung und der zugeordneten NEXT-Anweisung werden so oft wiederholt, bis die "Laufvariable" den Endwert erreicht hat.




- Die Variable nach FOR muß eine nicht indizierte (dimensionierte) Variable (siehe Abschnitt 4.9) sein.

- Der Ausdruck vor TO gibt den Anfangswert an, der der "Laufvariablen" zugewiesen werden soll. Der Ausdruck nach TO gibt den Endwert an, der von der "Laufvariablen" maximal angenommen wird.

- Wird der Parameter STEP (Schrittweite) nicht angegeben, wird automatisch nach jedem Schleifendurchlauf eine Erhöhung der Laufvariablen um 1 durchgeführt.
  Ansonsten wird sie um den Wert des hinter STEP stehenden Ausdrucks modifiziert.

- Die Laufvariable und alle auftretenden Ausdrücke müssen vom gleichen Datentyp sein.
  Die Verwendung des Datentyps INTEGER verkürzt die Laufzeit.

- Die FOR-Schleife wird grundsätzlich mindestens einmal durchgeführt, selbst wenn der Anfangswert höher als der Endwert sein sollte. Eine Überprüfung der Laufvariablen erfolgt erst nach jedem Durchlauf.

- Sowohl der Ausdruck nach TO als auch der Ausdruck nach STEP werden nach jedem Schleifendurchlauf neu berechnet.
  Alle in diesen Ausdrücken enthaltenen Variablen können innerhalb der Schleife also noch verändert werden.
  Von diesem Programmstil wird jedoch abgeraten!

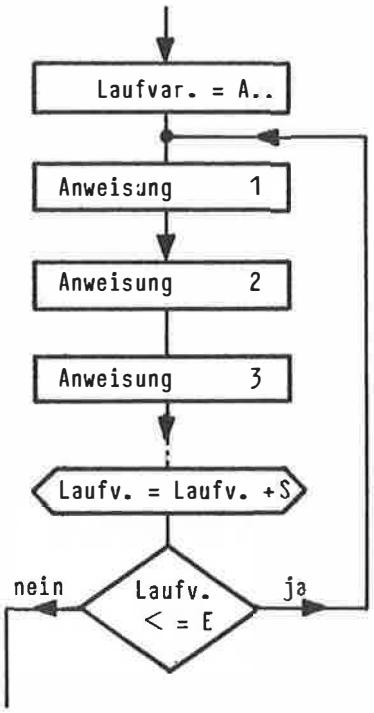
- Auch der Inhalt der "Laufvariablen" kann innerhalb der Schleife modifiziert werden. Auch von dieser Möglichkeit sollte der Programmierer keinen Gebrauch machen, um eine übersichtliche Programmstruktur zu behalten.

Beispiele:

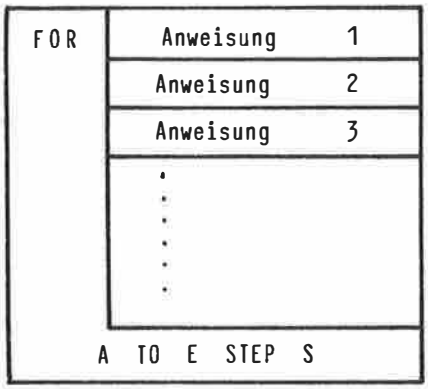
```
FOR  I% = 1 TO 100
FOR  INDEX% = ANF% TO END% STEP 5
FOR  DIF = 0.1  TO MAX STEP 0.2
FOR  NUMMER% = 100 TO 1 STEP -1
```

Logische Darstellung:

FLUSSDIAGRAMM

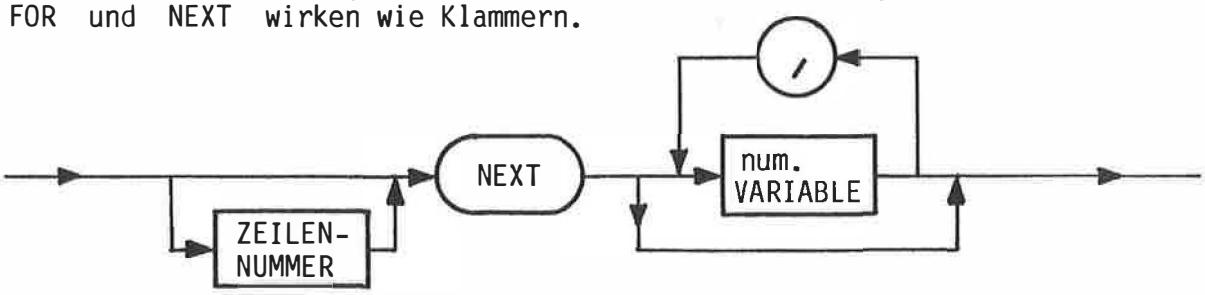



STRUKTOGRAMM




## Die NEXT-Anweisung

Mit der NEXT-Anweisung wird das Ende der FOR-Schleife gekennzeichnet.
FOR und NEXT wirken wie Klammern.




- Bei der hinter NEXT stehenden Variablen handelt es sich um die in der FOR-Anweisung festgelegten "Laufvariablen".
  Sie sollte immer angegeben werden.

- Es können mehrere FOR-NEXT-Schleifen ineinander geschachtelt werden. Dabei ist allerdings auf die richtige Struktur zu achten (siehe Beispielprogramm).

- Mit einer NEXT-Anweisung können mehrere FOR-Anweisungen abgeschlossen werden, wenn hinter dem NEXT die entsprechenden "Laufvariablen" gelistet werden.

Beispiele:

```
FOR  I% = 1  TO 10
   PRINT  ...........
   INPUT  ..............
   NEXT   I%


FOR  ZAHL% = ANFANG% TO ENDE% STEP I%
   ..............
   ..............
   IF  X < ERG  THEN 55
  ..............
 55 NEXT
```

Schachtelung von FOR-Schleifen

richtig

```
FOR  NR% = 1 TO ENDE%
   PRINT .........
   LET ..............

   FOR  LAUF=0.1 TO MAX STEP X
      LET ..........
      ...................
      NEXT LAUF

   NEXT NR%
```

falsch

```
FOR  NR% = 1 TO ENDE%
   PRINT ................
   LET ............

  FOR  LAUF=0.1 TO MAX STEP X
      LET .........
      ..................
      NEXT NR%

  NEXT LAUF
```

oder

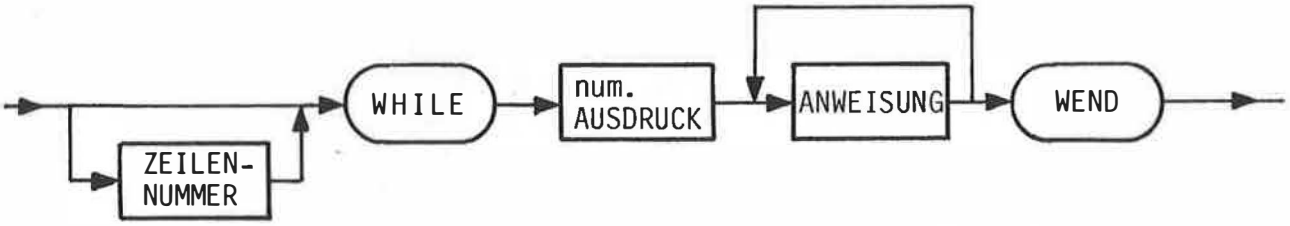
```
FOR  NR% = 1 TO ENDE%
   PRINT .........
   LET ..............

   FOR  LAUF=0.1 TO MAX STEP X
      LET ............
      ..............

    NEXT  LAUF,  NR%
```

## 4.8 Die Wiederholungsanweisung WHILE-WEND

Auch mit dieser Anweisung läßt sich die Wiederholung von Anweisungen (eines Anweisungsblockes) programmieren.
Allerdings wird bei der WHILE-Anweisung vor Beginn der Schleife geprüft, ob der < AUSDRUCK > Null ist.
Im Gegensatz zu einer FOR-Schleife wird eine WHILE-Schleife eventuell keinmal durchgeführt.




- Der Ausdruck muß vom Typ INTEGER sein und ist üblicherweise ein logischer Ausdruck bzw. ein Vergleichs-Ausdruck.
  Ein REAL-Ausdruck wird automatisch nach INTEGER konvertiert.

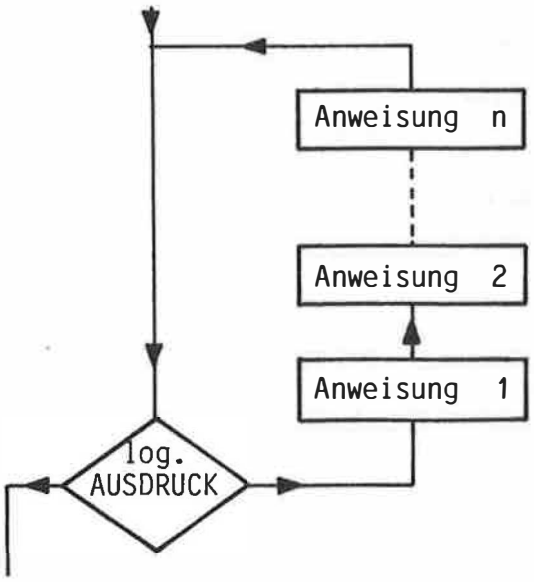
- Genau genommen handelt es sich hierbei um mindestens 3 Anweisungen!
  - Die WHILE-Anweisung
  - Die Anweisung(en) innerhalb der Schleife
  - Die WEND-Anweisung

  Logisch gesehen handelt es sich um einen "Anweisungs-Verbund".

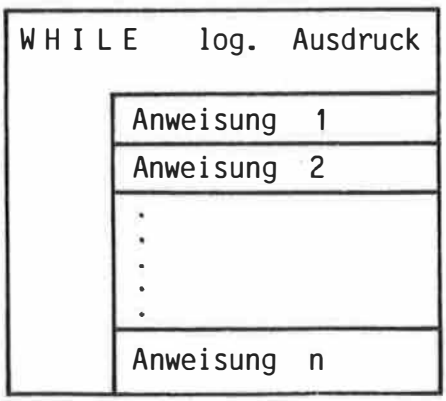
- Nach der CBASIC-Syntax kann auch vor WEND eine Zeilennummer stehen und angesprungen werden, die Wirkung ist jedoch die gleiche wie ein Sprung auf die Zeilennummer vor WHILE.

Logische Darstellung:

FLUSSDIAGRAMM




STRUKTOGRAMM



- Grundsätzlich kann gesagt werden, daß eine WHILE-Anweisung immer dann verwendet werden sollte, wenn die Anzahl der Schleifendurchläufe beim Eintritt in die Schleife noch nicht bekannt ist.

- Der Struktogrammblock WHILE kann in CBASIC direkt übernommen werden. Von dieser Möglichkeit sollte auch immer Gebrauch gemacht werden, da hierdurch ein Programm besonders übersichtlich und "lesbar" gestaltet werden kann.

Beispiele:

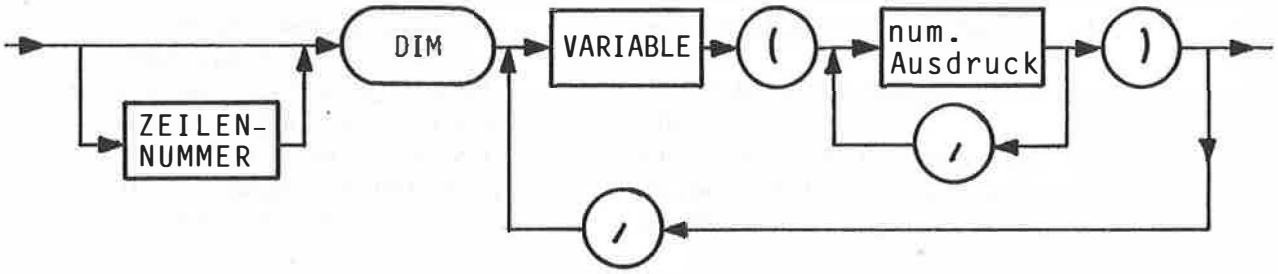
```
LET  ANTWORT$ = "JA"
WHILE  (ANTWORT$ = "JA")
   INPUT .............
   LET .........
   ....................
   ....................
   INPUT "Noch einmal (JA/NEIN) ? "; ANTWORT$
   WEND
```

```
WHILE  X > Z
   PRINT X
   LET  X = X + DIF
   WEND
```

```
WHILE  (INDEX% < = 100)
   PRINT .............
   ...................
   WHILE  (X < 0)
      LET .........
      ....................
      LET X = X * ANZ% + LOG(Y)
      WEND \  innere Schleife
   IF  TEXT$ <> "N" \
      THEN  PRINT ........  \
      INPUT.................  \
      ..................  \
      ELSE  INDEX% = INDEX% + NR%  \
      LET ....................
      ................
   WEND  \  äußere Schleife
```

## 4.9 Deklaration von Tabellen mit DIMENSION

Die DIM-Anweisung erzeugt in CBASIC ein Variablenfeld (eine Variablentabelle). Diese Tabelle kann ein- oder mehrdimensional sein.
Mit der DIM-Anweisung wird eine feste Anzahl gleicher Komponenten (Plätze gleichen Datentyps) erzeugt.




- Mit dieser Anweisung wird für eine Feldvariable die Anzahl der Dimensionen und der jeweilige maximale Index definiert.
  Der kleinste Index ist 0.

- Komponenten einer String-Tabelle können die Länge von 0 bis 255 Zeichen annehmen. Nach der Definition durch DIM werden die Komponenten als Null-String initialisiert.
  Numerische Komponenten werden mit 0 initialisiert.

- Im Gegensatz zu anderen BASIC-Versionen handelt es sich in CBASIC um eine durchführbare Anweisung, die zur Laufzeit ein entsprechendes Feld eröffnet. Existiert bereits ein Feld gleichen Namens, so wird dieses Feld vorher gelöscht!!!

  Ein String-Feld sollte jedoch nur gelöscht werden, nachdem alle Komponenten vorher als Null-String (Leer-String) gesetzt wurden. Diese Maßnahme ist notwendig, da Stringvariablen keine festen Speicheradressen haben, sondern bei jeder Zuweisung neu angelegt werden und somit immer mehr Speicher benötigen würden.
  Hierdurch kann es zu Programmlaufzeitverschlechterungen kommen, wenn bei Zuweisungskommandos Speicherreorganisationen erforderlich werden.

- Innerhalb eines Ausdrucks in der DIM-Anweisung darf keine "Rückwärtsreferenz" auf die gleiche Feldvariable stattfinden,
  wie z.B. DIM X(A, X(2,5)).

- Jede Feldvariable muß explizit in CBASIC definiert werden.
  Es gibt keine impliziten Systemeinstellungen für Felder, wie in anderen BASIC-Versionen.

- Es kann der gleiche Variablenname sowohl für eine Feldvariable als auch für nicht indizierte Variable verwendet werden.

  Z.B. X, X(10) und X(2,3)
  sind 3 verschiedene Variable(n-Bereiche).

  Hiervon sollte jedoch kein Gebrauch gemacht werden, um Programmierfehler zu vermeiden.

Beispiele:

```
DIM  ZAHL( 10) ,  X(50) ,  SATZ$( 10)
DIM  NAME$(30) ,  VORNAME$(30) ,  PLZ$(30)
DIM  ERG( 10,40) ,  BOOL%(5, 10,3)
DIM  MATRIX%(20,NR%) ,  FELD$(LEN(NAME$))
DIM  ERG(IX%,IY%) ,  UMSATZ(ANZ%, 12)
```

- Bei mehrdimensionalen Feldern werden schlagartig große Speicherbereiche reserviert.
  Zum Beispiel benötigt die Anweisung DIM X(99,299)
  100 mal 300 mal 8 Byte = 240.000 Byte.
  Der durch eine DIM-Anweisung benötigte Bedarf ist vom Programmierer aus diesem Grund vorher zu prüfen. Gegebenenfalls müssen solche Daten dann in einer externen Datei abgelegt werden.

Programmbeispiel:

```
DIM  UMSATZ ( 11,49) ,  NAME$ (49)

10 INPUT "Anzahl der zu erfassenden Kunden (max.50) ? "; ANZ%
   IF  ANZ% < 1  OR  ANZ% > 50 THEN  10

  FOR  I% = 0 TO ANZ%-1
      PRINT "Geben Sie den ";I%+1; ". Kundennamen ein : ";
      INPUT " "; NAME$ (I%)  : PRINT

      FOR  K% = 0 TO 11
          PRINT "Umsatz des "; K%+1; ". Monats : ";
          INPUT " "; UMSATZ (K%,I%)
          NEXT K%

      PRINT
      NEXT  I%
```

## 3. Programmier-Aufgabe

Schreiben Sie ein Programm, das ab einer vom Programmbediener wählbaren Anfangszahl zwischen 1 und 1000 und für die folgenden 10 Zahlen die "Grundzahlen", die Quadratzahlen und die Kubikzahlen in einer zweidimensionalen Tabelle (Feld) abspeichert, um diese Werte für die Weiterverarbeitung jederzeit im Zugriff zu haben.

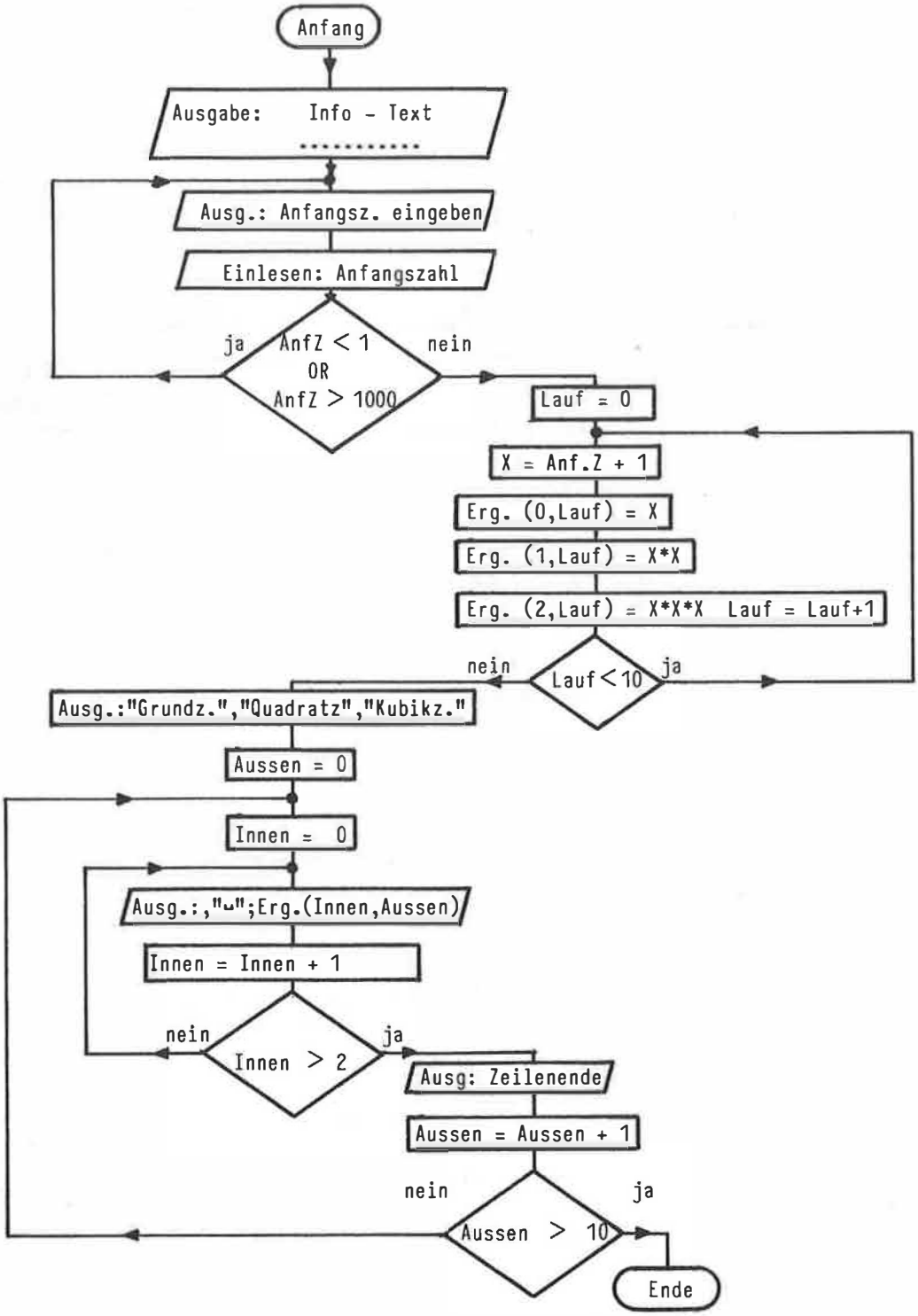
Anschließend sollen alle Werte in einer Tabelle der folgenden Form ausgegeben werden:

| GRUNDZAHL | QUADRATZAHL | KUBIKZAHL |
|---|---|---|
| n | n * n | n * n * n |
| n+1 | ..... | ..... |
| .... | ..... | ..... |
| usw. | | |

Struktogramm

```
Erzeugung eines zweidimensionalen Feldes :  ERG(2,10)
AUSGABE: Dieses Programm berechnet das Quadrat und .....
WHILE  (Anzahl < 1)  OR  (Anzahl > 1000)
       AUSGABE:  Anfangszahl eingeben ( 1-1000) :
       EINGABE:  Anfangszahl
FOR    Zahl = Anfangszahl + Lauf
       ERG (0,Lauf)= Zahl
       ERG ( 1,Lauf) = Zahl * Zahl
       ERG (2,Lauf) = Zahl * Zahl * Zahl
   Lauf = 0  TO  10
AUSGABE:  ,"GRUNDZAHL","QUADRATZAHL","KUBIKZAHL"
          ,"_____"
FOR    FOR    PRINT , " "; ERG (Innen,Aussen);
           Innen = 0  TO 2
       AUSGABE:   neue Zeile
  Aussen = 0  TO  10
```

## Darstellung der Aufgabe 3 im Flußdiagramm

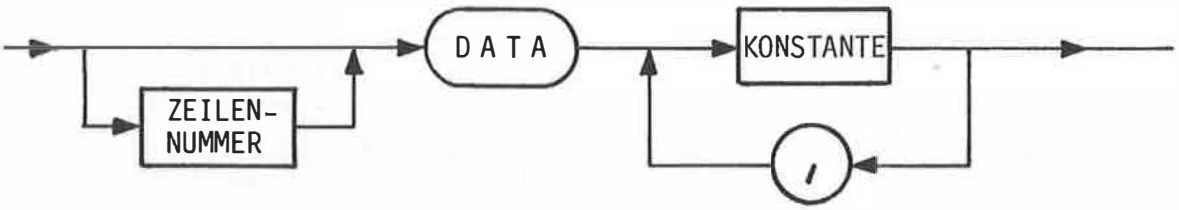


## 4.10 Die Anweisungen DATA, READ und RESTORE

Diese drei Anweisungen müssen als Befehlsgruppe gesehen werden, da keine der Anweisungen (sinnvollerweise) alleine in einem Programm auftreten kann.
Die Bedeutung dieser Anweisungen ist vorwiegend historisch zu sehen.
Mit Hilfe der LET-, und/oder der INPUT- und der GOSUB-Anweisung (siehe Abschnitt 4.11) läßt sich die gleiche Programmlogik erzielen.

### Die DATA-Anweisung

Data-Anweisungen sind nicht ausführbare Anweisungen mit denen Datenlisten, bestehen aus INTEGER-, REAL oder /und STRING-Konstanten, definiert werden können.




- Eine DATA-Anweisung kann nur alleine in einer Zeile stehen und sie darf nicht durch das "Zeilen-Trennzeichen" verlängert werden.
  Es dürfen jedoch beliebig viele DATA-Anweisungen in einem Programm enthalten sein.

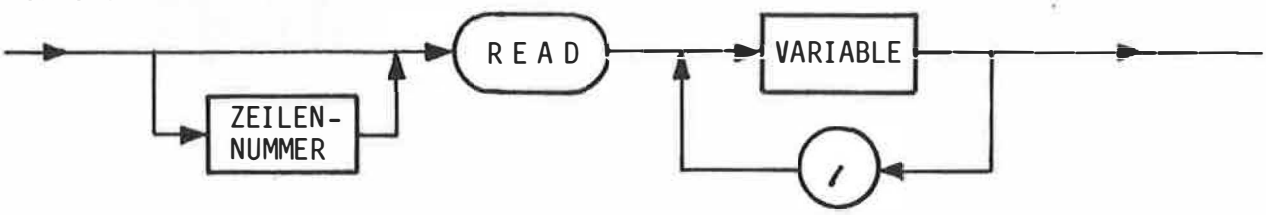
- Alle in einem Programm enthaltenen DATA-Anweisungen werden wie eine große DATA-Liste behandelt.
  Das bedeutet, daß ein interner "DATA-Zeiger" auf die gerade aktuelle Konstante der gesamten DATA-Liste zeigt.

- Werden STRING-Konstante von dem Zeichen (") eingeschlossen, können sie auch die Zeichen (,) und (") enthalten.
  Ansonsten erfolgt die Konstanten-Beschreibung in der DATA-Anweisung wie die Dateneingabe im INPUT-Befehl.

Beispiele:
```
DATA  12  ,  34.36  ,  "Überschrift"
DATA  "Zeile , Spalte"  ,  Meier  ,  600
```

### Die READ-Anweisung

Mit der READ-Anweisung können Konstante der DATA-Liste Variable zugeordnet werden.



- Jede READ-Anweisung liest sequentiell so viele Konstante ein, wie Variable hinter READ stehen. Sind weniger Konstanten in der DATA-Liste als Variablen im READ, wird ein Fehler gemeldet.

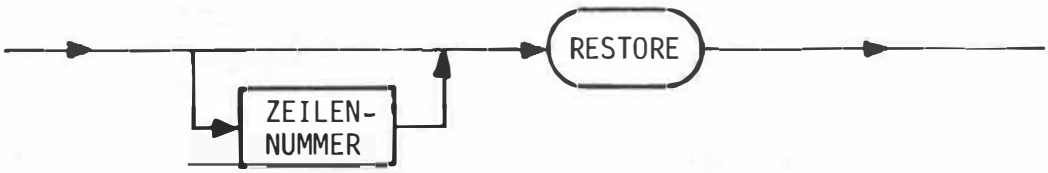
- Variablen und zugeordnete Konstanten müssen vom gleichen Datentyp sein. In Bezug auf Verträglichenkeiten und Grenzen für INTEGER- und REAL-Zahlen gelten die gleichen Regeln wie beim INPUT-Befehl.

- Nach jedem Lesen einer Konstanten zeigt der interne DATA-Zeiger automatisch auf die nächste Konstante der Liste.
  READ- und DATA-Anweisungen brauchen also nicht die gleiche Anzahl von Parametern (Variable bzw. Konstante) enthalten.

Beispiele:
```
READ X , NAME$ , NR% , TEXT1$(7)
READ   PI  ,  MAX  ,  ANZ%  ,  GRUNDWERT
```

## Die RESTORE-Anweisung

Diese Anweisung ermöglicht das Rücksetzen des internen DATA-Zeigers auf die erste Konstante der ersten (!) DATA-Anweisung.




- Die nach RESTORE folgende READ-Anweisung arbeitet die DATA-Liste wieder von vorne ab.

Programmbeispiel:

```
    DATA ------------------------------ , " ..........,.."
    DATA ====================== , ..............., "xxxxxxxxxx,xx"
    LPRINTER
    READ  PUFFER$
 10 FOR  I= 0 TO 5
       INPUT "Name eingeben : "; NAME$
       PRINT PUFFER$;"  ";NAME$
       NEXT I
    LET ANTW$="JA"
    WHILE ANTW$="JA"
       READ PUFFER$
       PRINT PUFFER$,
       ..............................
       READ PUFFER$
       PRINT PUFFER$,
       INPUT "Noch einmal ? : ";ANTW$
       IF  ANTW$="JA" THEN RESTORE : \
                           READ PUFFER$
       WEND
    READ PUFFER$
    usw.
```

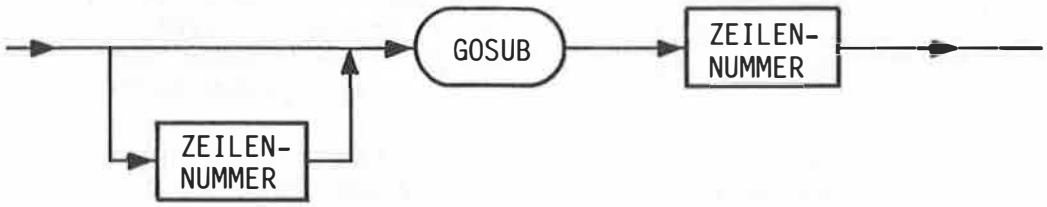
## 4.11 Unterprogramme in CBASIC

Durch die Verwendung von Unterprogrammen (Subroutines) können in CBASIC Programmsegmente von verschiedenen Stellen aus aufgerufen werden und die Programmkontrolle kehrt nach Durchführung des gewünschten Unterprogramms an die "richtige" Programmzeile zurück.

Hierdurch kann der Programmieraufwand zum Teil erheblich reduziert werden. Zusätzlich wird die Übersichtlichkeit verbessert und bei Korrekturen braucht nur ein Programmblock verändert zu werden.

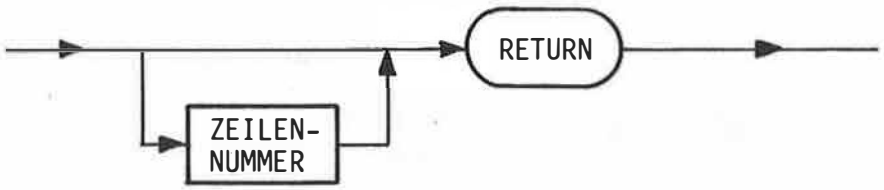
### Die Anweisung GOSUB

Diese Anweisung bewirkt einen Sprung in das gewünschte Unterprogramm.




- Statt GOSUB kann auch GO SUB geschrieben werden.
- Bei Durchführung dieser Anweisung wird die Rückkehradresse (nächste Zeile nach GOSUB) im STACK-Register gespeichert, anschließend findet ein Sprung auf die gewünschte Zeilenummer statt.
- In einem Unterprogramm kann ein weiteres Unterprogramm aufgerufen werden.
  Die Schachtelungstiefe von Unterprogrammen darf maximal 20 betragen.

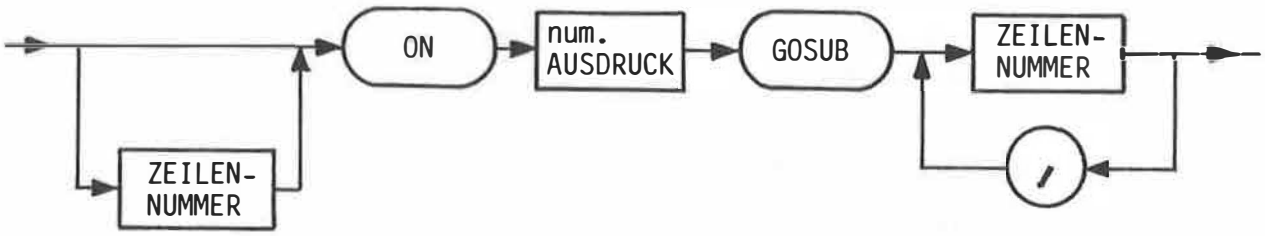
### Die "Rückkehr-Anweisung" RETURN




- Mit dieser Anweisung wird das Ende eines Unterprogramms gekennzeichnet. Die Programmdurchführung wird hierbei an der zuletzt im STACK-Register gespeicherten Rückführadresse fortgesetzt.
- Außer bei Unterprogrammen wird mit dieser Anweisung auch das Ende einer mehrzeiligen Funktion (siehe Abschnitt 4.12) gekennzeichnet.

## Die Anweisung ON-GOSUB

Mit Hilfe dieser Anweisung kann ein "berechneter Sprung" (siehe auch Abschnitt 4.5) in Abhängigkeit vom Wert eines AUSDRUCKS zu einer bestimmten Unterprogramm-Startzeile erfolgen.
Die Regeln für den Rücksprung aus dem Unterprogramm sind die gleichen wie bei der GOSUB-Anweisung.




- Der hinter ON stehende Ausdruck wird ausgewertet. Handelt es sich um einen REAL-Ausdruck, findet eine Umwandlung nach INTEGER statt.
  Ist der Ausdruck gleich 1, wird zum ersten angegebenen Unterprogramm (Statement-Nummer) nach GOSUB verzweigt. Ist der AUSDRUCK gleich 2, zum zweiten Unterprogramm usw..

- Ist der Wert des Ausdrucks kleiner 1 oder größer als die Anzahl der angegebenen Zeilennummern, erfolgt eine Fehlermeldung.

- Für die Anzahl der hinter GOSUB möglichen Zeilennummern gibt es keine starre Grenze. Man kann jedoch von mindestens 32 Zeilennummern ausgehen, so daß keine Programmiereinschränkung zu erwarten ist.

Logische Darstellung: siehe unter ON-GOTO

Beispiele:

```
        GOSUB 300
        LET  X= .....
        ..............
-------------------------------------------------
     300 REM  Unterprogramm Ebene 1
        .................
        GOSUB  1000
        INPUT ..............
        ...................
        GOSUB  1000
        ................
        RETURN    \  Ende Unterprogramm Ebene 1
-------------------------------------------------
     1000 REM  Unterprogramm Ebene 2
        ................
        ...................
        RETURN    \  ENDE Unterprogramm Ebene 2
-------------------------------------------------
```

Beispiel zu ON-GOSUB analog zu Beispiel ON-GOTO

## 4.12 Definition von Funktionen

Wird eine umfangreichere Berechnung eines Wertes in einem Programm häufiger benötigt, so kann diese Programmierung einmalig in einer eigenen Funktion erfolgen.
Vom Programmierer definierte Funktionen können anschließend in allen Ausdrücken genauso verwandt werden, wie die bereits implementierten Standardfunktionen.

Bei der Vergabe eines Funktions-Namens sind folgende Regeln zu beachten:

- Die Benennung einer Funktion erfolgt nach den gleichen Regeln wie die Benennung einer Variablen, außer daß die beiden ersten Namenszeichen "FN" sein müssen!!

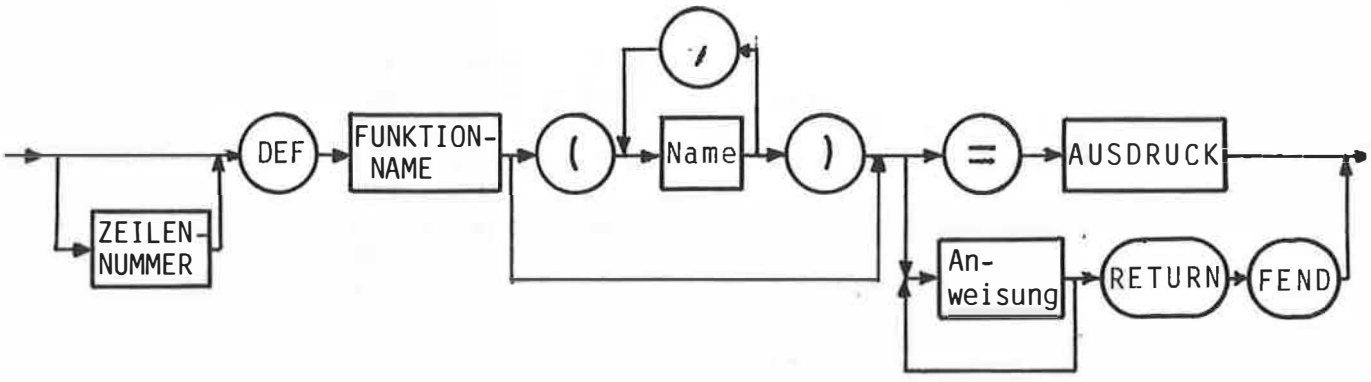
- Der Datentyp der Funktion wird wie bei den Variablen durch das letzte Zeichen festgelegt.

  Prozentzeichen = INTEGER
  Dollarzeichen = STRING
  kein Zeichen = REAL

Der Funktions-Name wird sowohl bei der Definition der Funktion als auch beim Aufruf benötigt.

### Die DEF-Anweisung

Mit dieser Anweisung können ein- oder mehrzeilige Funktionen vom Programmierer definiert werden.




- Die Funktion muß vom gleichen Datentyp sein wie der AUSDRUCK.

- Bei den anzugebenden "Namen" handelt es sich um die sogenannten "Formal-Parameter", die beim aktuellen Funktionsaufruf durch die "Aktual-Parameter" ersetzt werden.
  Durch diese Regelung kann ein Ausdruck in Abhängigkeit von verschiedenen Variablen berechnet werden.

- Die Namen der "Formal-Parameter" sind reine "Platzhalter", sie haben keinen Einfluß und keinen Bezug auf im Programm auftretende gleiche Variablennamen.

- Im AUSDRUCK können Variable, Konstante und auch andere Funktionen aufgerufen werden.
  Rekursive Aufrufe (z.B. FN.EINS(X) = X*FN.EINS(Y) ) sind nicht erlaubt.

- Der Datentyp der Parameter ist unabhängig vom Datentyp der Funktion. Die jeweiligen "Aktual-Parameter" und "Formal-Parameter" müssen jedoch vom gleichen Datentyp sein.

Beispiele:

```
DEF  FN.L$ (A$,I%) = LEFT$ (A$,I%-100)
DEF  FN.FUNK (X,Y) = SIN (X) * SQR ( 1+X*Y)
DEF  FNMILLI (X) = X / 0.25
.............................
LET TEXT$ = FN.L$ (BASIS$,K%)
.....................
LET ERGEB = X1 * FN.FUNK (A2,X1(L%))
...............
LET MASS = FNMILLI (EINGABE)
```

- Die Definition einer Funktion kann in CBASIC auch aus mehreren Zeilen (Anweisungen) bestehen (siehe Syntax-Diagramm).
  Die letzte Funktions-Zuweisung erzeugt dann den Funktionswert.

- Besteht eine Funktion aus mehreren Zeilen, muß sie mit RETURN und FEND abgeschlossen werden.
  RETURN und FEND sind selbständige Anweisungen, vor denen eine Zeilennummer stehen darf.

Beispiel:

```
DEF  FN.EINGABE% (ZAHL)
10 INPUT "Zahl eingeben : "; ZAHL
   IF  ZAHL > 32767  OR  ZAHL < -32768   \
      THEN  PRINT "Zahl zu groß oder zu klein" : \
            GOTO 10
   FN.EINGABE% (ZAHL) = INT (ZAHL)
   RETURN
   FEND

.................................
ANZAHL% = FN.EINGABE% (WERT)

......
```

## 4.13 Vordefinierte String-Funktionen in CBASIC

| Funktion | Typ der FKT. | Beschreibung der Funktion |
|---|---|---|
| ASC (A$) | INTEGER | erzeugt den ASCII-Wert des ersten Zeichens von A$ |
| CHR$ (I%) | STRING | erzeugt das ASCII-Zeichen, das dem Wert von I% entspricht |
| LEFT$ (A$,I%) | STRING | erzeugt einen String, der die ersten I% Zeichen von A$ umfaßt |
| RIGHT$ (A$,I%) | STRING | erzeugt einen String aus den letzten I% Zeichen von A$ |
| MATCH(A$,B$,I%) | INTEGER | gibt die Position des ersten Auftretens von A$ in B$ beginnend ab Position I% (sonst = 0). Hierbei können "Maskenzeichen" verwendet werden :<br># = alle Ziffern<br>! = alle kl. + gr. Buchstaben<br>? = beliebiges Zeichen<br>\ = Maskenfkt. aufgehoben |
| MID$(A$,I%,J% | STRING | erzeugt String, der J% Zeichen von A$ ab Position I% enthält |
| LEN (A$) | INTEGER | gibt die Länge von A$ in Byte (Zeichen) an |
| UCASE$ (A$) | STRING | wandelt alle kl. Buchstaben von A$ in große Buchstaben um |
| STR$ (X) | STRING | wandelt den REAL-Wert X in einen String um |
| VAL (A$) | REAL | wandelt A$ in eine REAL-Zahl um. Die Konvertierung wird beendet, bei String-Ende bzw. unerlaubten Zeichen. Ist das erste Zeichen ungleich +, - oder Ziffer, wird die Funktion zu Null gesetzt. |

Weitere String-Funktionen mit vorwiegend systeminternen Zugriffsoperationen sind der Original CBASIC-Beschreibung zu entnehmen.

## Ergebnisbeispiele zu STRING-Funktionsaufrufen

| Funktionsaufruf | Ergebnis |
|---|---|
| ASC ("A") | 65 |
| ASC ("CBA") | 67 |
| CHR$ (66) | "B" |
| CHR$ (49) | " 1" |
| LEFT$ ("TEXT",2) | "TE" |
| LEFT$ ("TEXT",6) | "TEXT" |
| RIGHT$ ("TEXT",2) | "XT" |
| MATCH ("X","MAXIMUM", 1) | 3 |
| MATCH ("X","WERT", 1) | 0 |
| MATCH ("##","JULI 1980",4) | 6 |
| MID$ (": ....", 1,4) | ": .." |
| MID$ ("Nr. 03 14",4,4) | "03 14" |
| LEN ("TITEL") | 5 |
| UCASE$ ("Nein") | "NEIN" |
| UCASE$ ("neiN | "NEIN" |
| STR$ (3.14 159) | "3. 14 159" |
| VAL ("-2.56AB3") | -2.56 |

Alle Beispiele wurden aus Gründen der Übersichtlichkeit mit Konstanten "besetzt.

Selbstverständlich können auch alle Funktionen mit variablen Parametern aufgerufen werden.

## 4.14 Bildschirm-Ansteuerung in CBASIC

Bildschirm-Ansteuerung in CBASIC läßt sich mit der bereits jetzt bekannten Anweisung PRINT und der String-Funktion CHR$ durchführen.
Der jeweils erforderliche Bildschirm-Steuercode kann direkt in die CHR$-Funktion eingegeben werden und mit der PRINT-Anweisung ausgegeben.

Folgende wichtige Steuercodes sollten dem Programmierer bekannt sein:

| dezimal | hexadezimal | Bedeutung |
|---|---|---|
| 07 | 07 | Ring Bell |
| 10 | 0A | Line Feed (Zeilenvorschub) |
| 12 | 0C | Clear Screen (Bildschirm löschen) |
| 27+74 | 1B+4A | Löschen bis Bildschirmende |
| 27+102<br>+Spalte<br>+Zeile | 1B+66 | Anwahl Cursor-Adressierung<br>Die Spaltennummer hat einen Offset von 32<br>Die Zeilennummer hat einen Offset von 32 |

Weitere Steuercodes können den PEOPLE-Unterlagen entnommen werden.

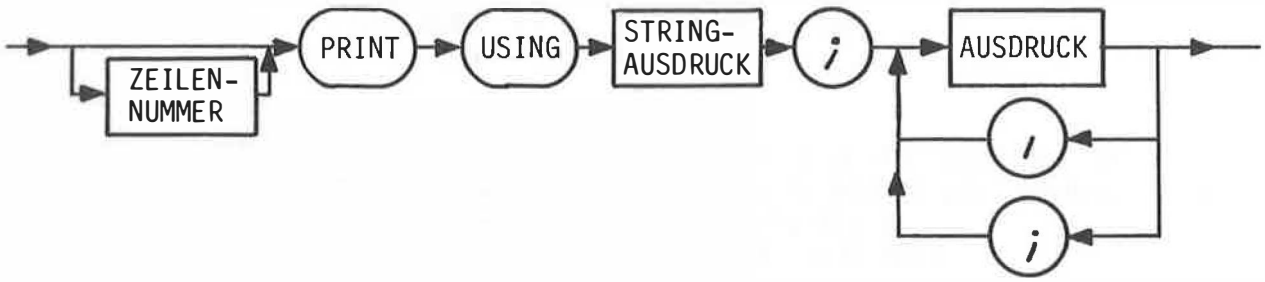
Beispiel zur Ausgabe einer Text-Maske:

```
LET RB$ = CHR$(07)
LET CS$ = CHR$(0CH)

LET XY$ = CHR$( 1BH) + CHR$(66H)
POS 1$= XY$+ CHR$(32+ 10)+ CHR$(32+5)    :REM Spalte 10, Zeile 5
POS2$= XY$+ CHR$(32+ 10)+ CHR$(32+8)     :REM Spalte 10, Zeile 8
POS3$= XY$+ CHR$(32+ 10)+ CHR$(32+ 15)   :REM Spalte 10, Zeile 15
POS4$= XY$+ CHR$(32+50)+ CHR$(32+ 15)    :REM Spalte 50, Zeile 15
REM
PRINT CS$; POS 1$; "Erfassen von Kundendaten     : 1"
PRINT      POS2$; "Listen von Kundendaten        : 2"
PRINT POS3$; "Gewünschte Segmentnummer  ?"; RB$; POS4$
INPUT " "; ANTW$
```

## 4.15 Formatierte Ausgaben mit PRINT-USING

Bei Verwendung dieser Anweisung lassen sich in CBASIC formatierte Ausgaben erzeugen.




- Der STRING-Ausdruck nach USING bestimmt mit seinen "Format-Zeichen" die Form der Ausgabe der nachfolgenden Ausdrücke.

- Anders als im normalen PRINT-Befehl bewirkt ein Komma nach einem Ausgabe-Ausdruck keinen automatischen Sprung auf die nächste Tabulatorposition.

- Innerhalb des STRING-Ausdrucks (Formats) können mehrere Datenformate enthalten sein. Sie beziehen sich jeweils auf einen der auszugebenden Ausdrücke.
  Ist die Anzahl der Datenformate kleiner als die Anzahl der auszugebenden Ausdrücke, findet eine automatische Formatwiederholung des gesamten STRING-Ausdrucks statt.

- Alle im Format-String enthaltenen Zeichen, die keine spezielle Bedeutung als Format-Zeichen haben, werden direkt ausgegeben.

Folgende Zeichen haben eine spezielle Bedeutung innerhalb des USING-String's:
(Die "spitzen Klammern" in den Beispielen stellen die Ausgabe dar. Sie werden nicht mit ausgegeben ! )

`!` definiert einen 1-Zeichen-String
z.B. `PRINT USING"!x !"; "TAG", "NACHT"`
bewirkt : < Tx N >

`&` definiert eine variable String-Länge.
Der String wird in seiner tatsächlichen Länge ausgegeben.
z.B. `PRINT USING "& &"; "Heute","ist Montag"`
bewirkt : < Heute ist Montag >

`/.../` Hierdurch wird eine String-Ausgabe auf n Zeichen begrenzt.
Die auszugebende String-Länge wird durch die Anzahl der Zeichen bestimmt (einschl. der Schrägstriche.)
Zwischen den Schrägstrichen können beliebige Zeichen stehen, sie haben keine Auswirkung auf die Ausgabe. Ist die String-Länge kleiner als die Anzahl der Zeichen, werden zusätzliche Blanks ausgegeben.
z.B. `PRINT USING "/234/"; "ABCDEFGH"`
bewirkt: < ABCDE >

#  definiert eine Ziffer eines numerischen Ausdrucks. Eine Zahl kann durch mehrere # formatiert werden. Führende Nullen werden durch Blanks ersetzt.
Ist der Ausdruck negativ, wird vor der höchstwertigen Ziffer ein Minuszeichen ausgegeben.
z.B. PRINT USING "#### ##"; 2113 , 14 , -12 , 4
bewirkt : < 2113 14 -12 4 >

.  Innerhalb eines Zahlenformats darf ein (!) Dezimalpunkt auftreten. Sollten weniger Nachkommastellen ausgegeben werden als vorhanden sind, findet ein automatisches auf- oder abrunden statt.
z.B. Print USING "###.##"; 12.678 , 4.3 , -14.1047
bewirkt : <12.68 4.30 -14.10>

^  Folgt einem numerischen Format ein oder mehrere "Hochpfeile" (uparrows), so wird diese Zahl in der Exponentialform ausgegeben.
Die Exponentialzahl wird bei der Ausgabe so normiert, daß alle geforderten führenden Formatziffern besetzt sind.
z.B. PRINT USING "####.##^"; -1.345678 , 4560
bewirkt : <-134.57E-02 4560.00E 00 >

**  Stehen vor einem numerischen Format 2 führende Sternchen, so werden alle führenden Nullen statt durch Blanks durch Sterne ersetzt.
Hierbei handelt es sich um die typische Schecksicherung.
Dieses Formatzeichen darf nicht im Zusammenhang mit der Exponentialdarstellung verwendet werden.
z.B. PRINT USING "**#######.##"; 123.6 , 0.4 , -5876.95
bewirkt : <***123.60 *****0.40 *-5876.95 >

\  Soll eines der festgelegten Formatzeichen innerhalb eines USING selber ausgegeben werden, kann die Formatbedeutung dieses Zeichens durch einen vorangehenden nach links gerichteten Schrägstrich (\) aufgehoben werden.
z.B. PRINT USING "\# ##"; 74
bewirkt : < # 74 >

Reicht ein gewähltes USING-Format nicht aus, um alle relevanten Ziffern einer Zahl darzustellen, so wird das Ausgabeformat automatisch so vergrößert, daß der Wert der Zahl nicht verändert wird.
Dieser "System-Eingriff" wird durch die führende Ausgabe eines Prozentzeichens sichtbar gemacht.
z.B. PRINT USING "#.#"; 233.456 , 7.87
bewirkt : <% 233.5 7.9 >

Weitere (in Deutschland normalerweise nicht notwendigen) Möglichkeiten des USING-Formats müssen der Orignal-CBASIC-Beschreibung entnommen werden.

## 4. Programmier-Aufgabe

Ein Gastwirt will Gäste nur dann in sein Lokal lassen, wenn sie essen oder trinken wollen und wenn sie Barzahler und nicht betrunken sind.
Ein Programm soll diese logischen Entscheidungen nachvollziehen, indem die entsprechenden Fragen vom Programmbenutzer mit "ja" oder "nein" beantwortet werden können.
Anschließend soll die Antwort des Gastwirtes erfolgen, indem der Text
"Lieber Gast, kommen Sie doch bitte herein!" oder
"In meinem Lokal ist kein Platz für Sie!" auf dem Bildschirm erscheint.

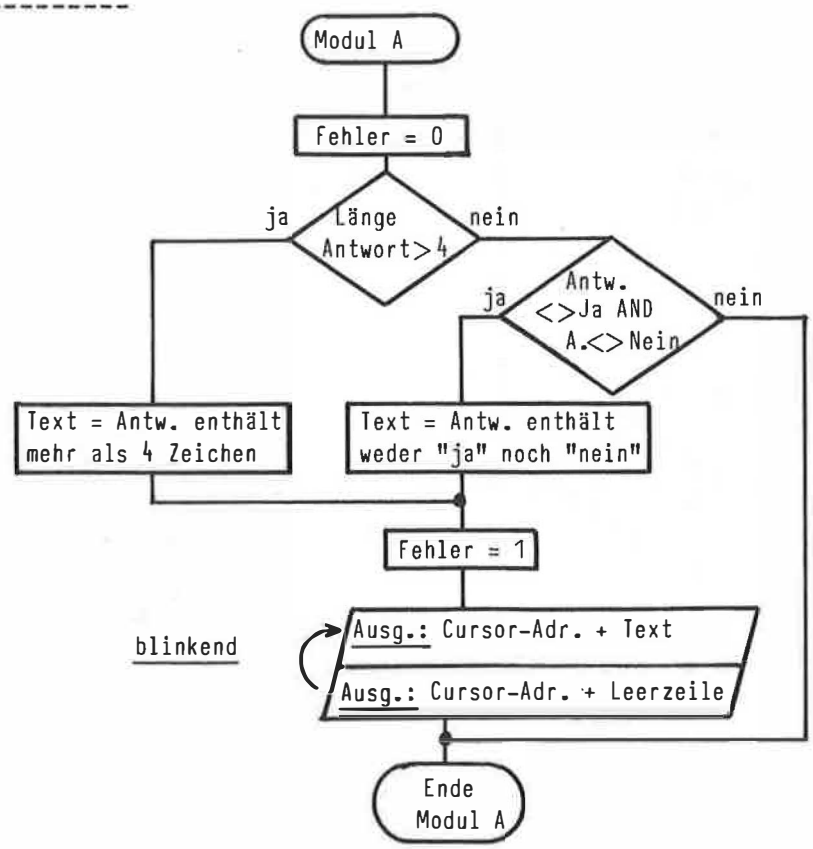
- Die Abprüfung, ob der Programm-Bediener nicht mehr als 4 Zeichen eingegeben hat und ob die Eingabe tatsächlich "ja" oder "nein" war, soll in einem Unterprogramm (Modul A) erfolgen.

- Dieses Unterprogramm soll bei fehlerhaften Eingaben einen entsprechenden Fehlertext initialisieren und diesen Fehlertext als Fehlermeldung auf dem Bildschirm blinkend darstellen.
  Außerdem muß im Unterprogramm ein "Fehlermerker" gesetzt werden.

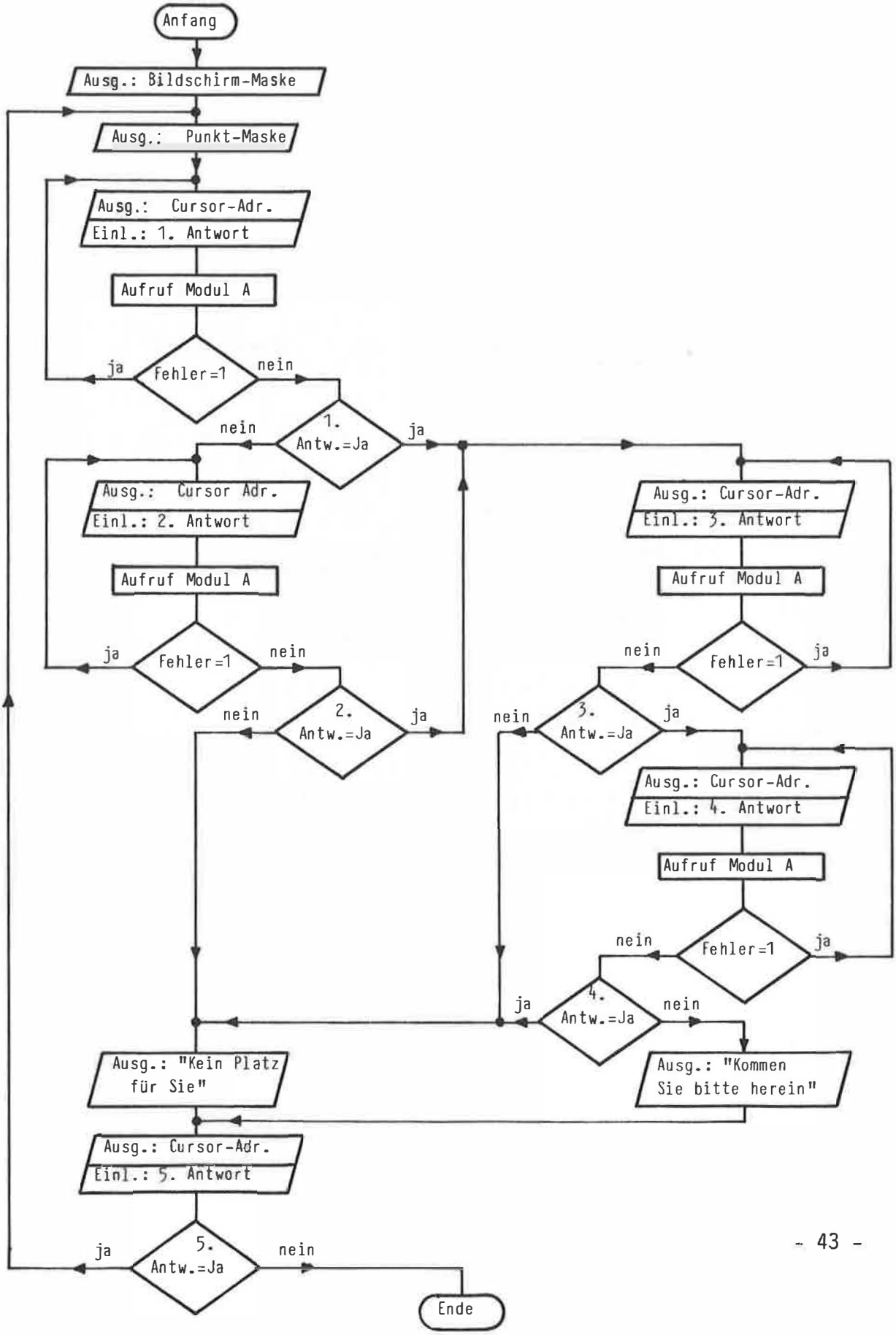
- Das Programm soll beliebig oft wiederholbar sein.

- Das Programm soll mit den beigefügten, feststehenden Bildschirm-Masken arbeiten. Es sind also die dargelegten Bildschirm-Steuercodes zu verwenden.

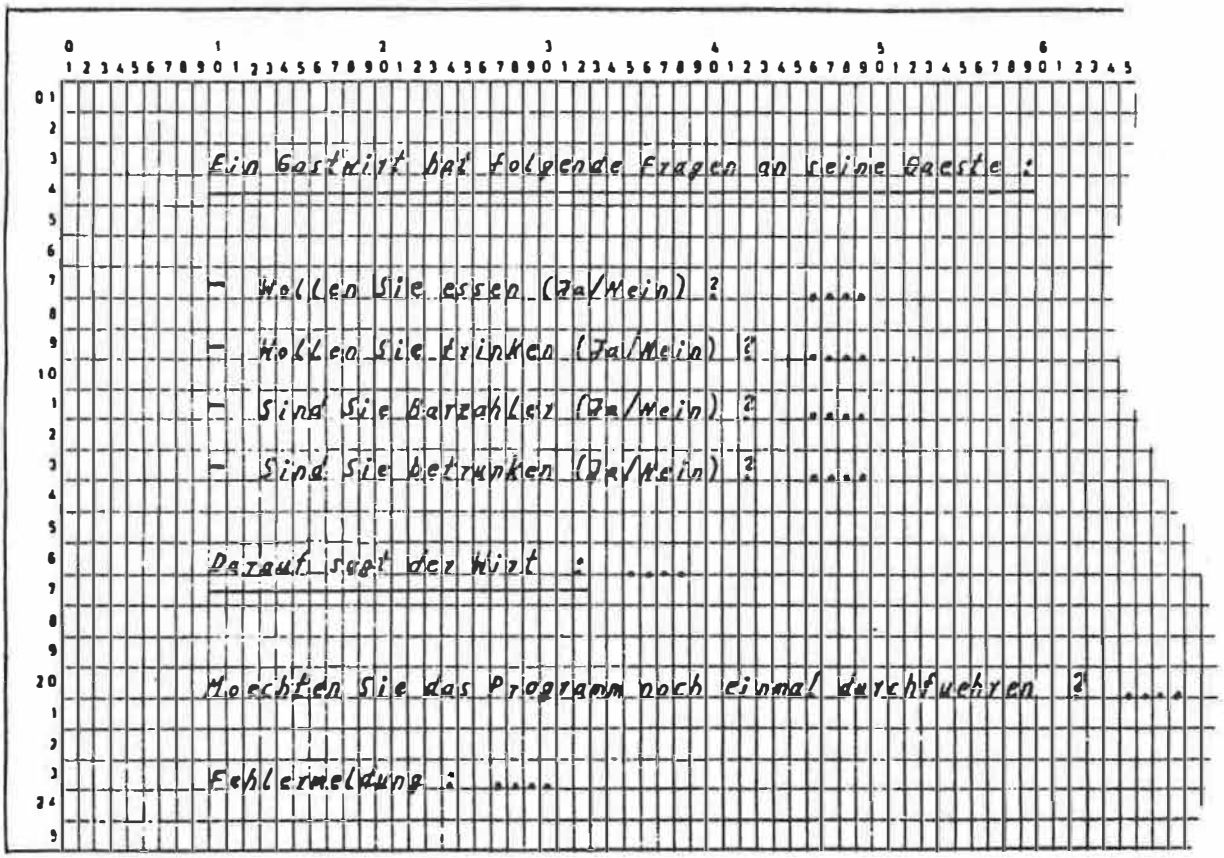
Flussdiagramm MODUL A



## Darstellung der Aufgabe 4 im Flussdiagramm



Bildschirmentwurfsblatt zu Aufgabe 4
==========



## 5.0 Dateiorganisation in CBASIC

### 5.1 Allgemeines zur Dateiverwaltung

Sobald größere Probleme per Computer bearbeitet werden sollen, müssen irgendwelche Daten (Kundendaten, Artikeldaten, Meßdaten usw.) längerfristig gespeichert werden bzw. können wegen Speicherplatzmangel nicht permanent im Arbeitsspeicher gehalten werden. Diese Daten werden dann in Dateien abgelegt.

In der Programmiersprache CBASIC sind auf der Basis des CP/M-86 Betriebssystems 2 verschiedene Dateizugriffsverfahren implementiert. Es wird sowohl die sequentielle Dateistruktur als auch die satzorientierte Dateistruktur (Random-Zugriff) unterstützt. Sequentielle Dateizugriffe können mit variablen oder festen Satzlängen erfolgen, satzorientierte Dateizugriffe nur mit fester Satzlänge.

Schaubild einer RANDOM-Datei

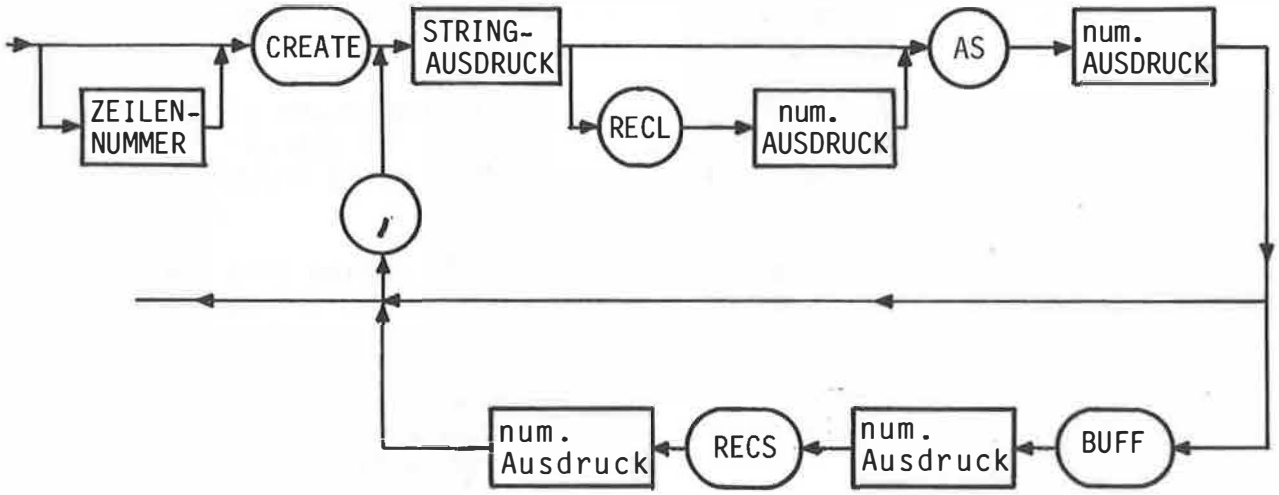

Jede zu bearbeitenden Datei muß auf der Magnetplatte (Diskette) vorhanden sein und im Programm angemeldet (geöffnet) werden.
In einem Programm können maximal 20 Dateien geöffnet sein. Dabei wird für jede geöffnete Datei eine sogenannte logische Arbeitsnummer vergeben, die zwischen 1 und 20 liegt.
Da für jede in der "OPEN-Tabelle" eingetragene Datei Arbeitsspeicher benötigt wird, kann es zweckmäßig sein, nicht mehr benötigte Dateien zu schließen (abzumelden).

Auf Grund der insgesamt zur Verfügung stehenden Unterrichtzeit im Rahmen dieser Einführung wurden auch die Erläuterungen zu den Datei-Anweisungen auf das Notwendigste begrenzt.
Bei praktischer Programmierung mit Dateien sollte darum auf jeden Fall die Original CBASIC-Beschreibung mit zu Rate gezogen werden.

## 5.2 Dateierzeugung und Dateieröffnung mit der CREATE-Anweisung

Mit dieser Anweisung wird eine Datei mit einem wählbaren Namen auf einer gewünschten Diskette erzeugt (kreiert) und gleichzeitig geöffnet.

ACHTUNG: Mit dieser Anweisung wird eine existierende Datei zuerst gelöscht und dann wird eine neue Datei gleichen Namens erzeugt.




- Der STRING-AUSDRUCK nach CREATE bezeichnet den Datei-Namen einschl. eines gewünschten Disketten-Laufwerks.
  Der Datei-Name muß den CP/M-86 Konventionen für erlaubte Datei-Namen entsprechen.
  Wird kein Laufwerk angegeben, wird das momentan aktive Laufwerk angewählt.

- Die (optionale) Definition von RECL (RECORD-Length) muß immer dann gegeben werden, wenn absolute Dateiorganisation (satzorientierter Zugriff) erfolgen soll.
  Mit RECL wird eine feste Satzlänge in Byte definiert, die größer Null sein muß.
  Ist der Ausdruck vom Typ REAL, findet eine automatische Umwandlung nach INTEGER statt.
  Näheres zur RECORD-Länge siehe nächste Seite.

- Der numerische Ausdruck nach AS bezeichnet die logische Arbeitsnummer, unter der das System die physikalische Datei führen soll. Nur über diese logische Arbeitsnummer kann im Programm auf diese Datei zugegriffen werden.
  Es können maximal 20 Arbeitsnummern (1-20) vergeben werden.

- Mit der Angabe der Puffergröße (Buffer-Size) kann die Anzahl der Plattensektoren festgelegt werden, die jeweils im Arbeitsspeicher als Datei-Zwischenpuffer benutzt werden.
  Die System-Voreinstellung ist 1 Sektor.
  Bei RANDOM-Zugriff muß die Puffergröße gleich 1 sein.

- Der Ausdruck nach RECS wird momentan nicht ausgewertet und hat somit keine aktuelle Bedeutung.

## Erläuterungen zur RECORD-Länge

Alle Daten werden von CBASIC mit ASCII-Zeichen auf die Diskette gespeichert. Das gilt auch für die numerischen Datentypen REAL und INTEGER!
Die Sätze in einer RANDOM-Datei werden grundsätzlich in der geforderten Länge angelegt, auch wenn die Satzinhalte die geforderte Byte-Anzahl nicht ausnutzen.

- Um die erforderliche Byteanzahl für numerische Werte zu berechnen, muß die maximale Anzahl von Ziffern, ein eventuelles negatives Vorzeichen und bei REAL-Zahlen der "Gleitpunkt" berücksichtigt werden.

- Sowohl in sequentiellen als auch in RANDOM-Dateien wird jeder Satz durch CBASIC mit den Zeichen "Carriage Return" und "Line Feed" abgeschlossen. Diese beiden Zeichen (Byte) müssen bei Angabe der Satzlänge mitgezählt werden.

- Desweiteren werden alle STRING's durch doppelte Hochkommas (") eingeschlossen, so daß die notwendige STRING-Länge plus 2 Byte berücksichtigt werden muß.

- Alle Satzkomponenten werden durch Kommas voneinander getrennt.
  Auch die hierfür notwendigen Byte müssen vom Programmierer bei Angabe der Satzlänge mit einbezogen werden.

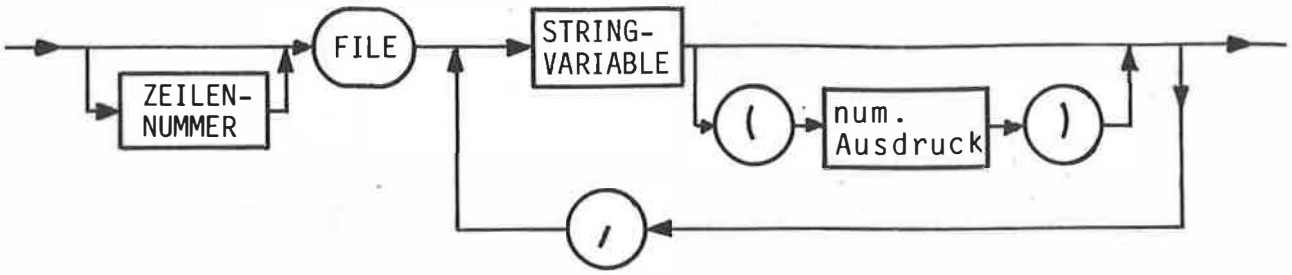
## Beispiel zur RECORD-Länge mit 5 Satzfeldern:

| | | | | | | |
|---|---|---|---|---|---|---|
| - | 6-stellige Artikelnummer (STRING) | : | 6 Byte | + 2 (") | + 1 = | 9 |
| - | 20-stelliger Artikelname | : | 20 Byte | + 2 (") | + 1 = | 23 |
| - | Stückzahl, num. max. 10000 | : | 5 Byte | | + 1 = | 6 |
| - | Artikelpreis, max. 100000.00 | : | 9 Byte | | + 1 = | 10 |
| - | 3-stell. Kennzeichen (STRING) | : | 3 Byte | + 2 (") | = | 5 |
| | | | | + <CR> + <LF> | = | 2 |
| erforderliche Satzlänge in Byte | | : | | | = | 55 |

```
Beispiele:        CREATE  "MERKSATZ.DAT"  RECL 20  AS 5
----------        CREATE  DATEI$   AS NR%
                  CREATE  NAME$   AS 20
                200 CREATE  "DATEI 1.DAT"  RECL ANZ%  AS INDEX%
                  CREATE "DAT 1.DAT"  AS 7  ,  "DAT2.DAT"  AS 3
```

## 5.3 Dateierzeugung oder/und Dateieröffnung mit der FILE-Anweisung

Hiermit wird die auf der Diskette befindliche Datei des gewünschten Namens geöffnet.
Ist keine Datei mit dem entsprechenden Namen vorhanden, wird sie mit der eventuell geforderten Satzlänge erzeugt (kreiert) und geöffnet.




- Die Variable beinhaltet den Dateinamen einschl. der eventuell gewünschten Laufwerksbezeichnung.
  Es darf keine indizierte Variable verwendet werden!

- Der numerische Ausdruck entspricht der Definition einer festen Satzlänge wie im CREATE-Befehl.

- Die logische Arbeitsnummer wird bei dieser Anweisung automatisch vergeben, indem die nächste freie Arbeitsnummer, beginnend bei 1, dieser Datei zugeordnet wird.
  Wurden bereits alle 20 erlaubten Arbeitsnummern vergeben, wird ein Fehler gemeldet.
  Arbeitet der Programmierer mit dieser Anweisung, muß er die vom System vergebene logische Arbeitsnummer selber "errechnen" da nur über diese log. Arbeitsnummer ein Zugriff auf die Datei erfolgen kann.

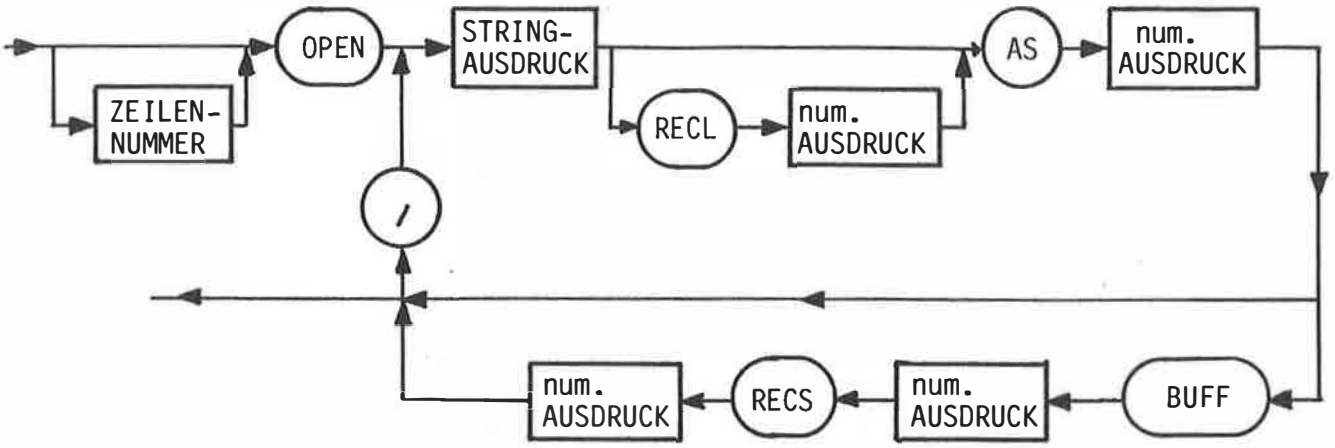
- Die FILE-Anweisung ist für permanente Dateien zweckmäßig, da keine Löschung bereits vorhandener Dateien stattfindet.

Beispiele:
----------

```
FILE  NAME$
FILE  DATEI$ ( 128)
FILE  TEXT$  ,  KUNDEN.DAT$ (85)  ,  NAME 1$
FILE  DAT 1$  ,  DAT2$  ,  DAT3$
```

## 5.4 Die OPEN-Anweisung

Mit dieser Anweisung wird eine auf der Diskette befindliche Datei im System angemeldet (in die "OPEN-Tabelle" eingetragen).
Diese Anweisung ist nur erforderlich, wenn keine CREATE- , FILE- oder CLOSE-Anweisung mit dem gleichen Dateinamen vorausging.

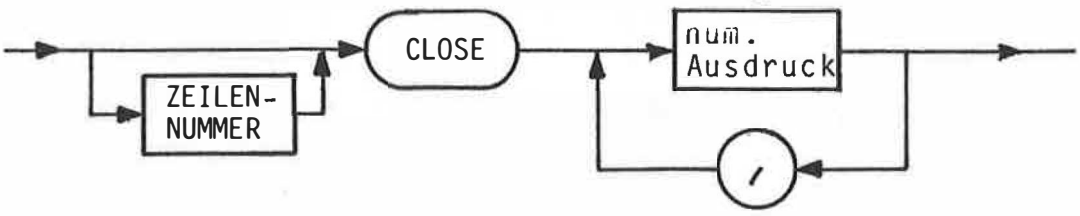



- Die Befehlssyntax und die Bedeutung der Parameter der OPEN-Anweisung entspricht der CREATE-Anweisung.
  Die dort beschriebenen Erläuterungen gelten entsprechend.

```
Beispiele:      OPEN  "KUNDEN.DAT"   RECL 74   AS 8
---------       OPEN  DATEI$   AS  DATEI.NR%
                OPEN  DAT1$   AS 1  ,  DAT2$   AS 2  ,  DAT3$  AS 3
```

## 5.5 Die CLOSE-Anweisung

Mit dieser Anweisung können eine oder mehrere durch CREATE-, OPEN- oder FILE-Anweisungen geöffnete (aktivierte) Dateien wieder geschlossen werden.




- Der numerische Ausdruck nach CLOSE gibt die logische Arbeitsnummer an, unter der die Datei geöffnet ist.

- Nach Durchführung dieser Anweisung ist die vorher belegte log. Arbeitsnummer wieder frei und kann z.B. für eine andere Datei verwendet werden.

- Der durch die Datei belegte Pufferbereich im Arbeitsspeicher steht nach Durchführung dieser Anweisung wieder allgemein zur Verfügung.

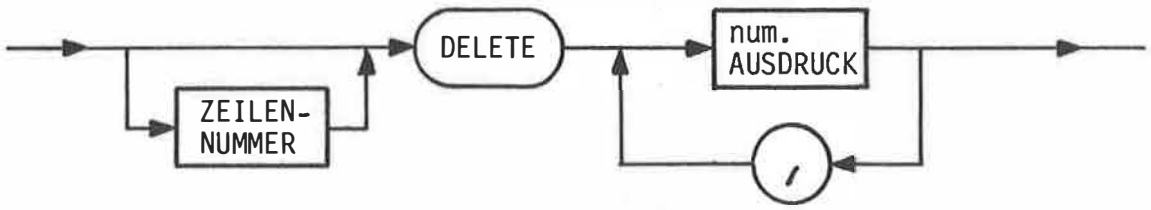
- Soll die mit CLOSE geschlossene Datei erneut lesend oder schreibend bearbeitet werden, muß ein erneutes OPEN- oder FILE-Kommando erfolgen.

- Alle IF-END-Anweisungen (siehe Abschnitt 5.7) dieser logischen Arbeitsnummer(n) haben keine Auswirkung mehr.

Beispiele:
```
CLOSE  3
CLOSE  NR%
CLOSE  7  ,  11  ,  14  ,  I%
CLOSE  INDEX%  ,  1+X*N%
```

## 5.6 Löschen von Dateien mit der DELETE-Anweisung

Diese Anweisung löscht eine oder mehrere Dateien aus dem Inhaltsverzeichnis der Diskette(n).




- Der numerische Ausdruck nach DELETE bezeichnet die logische Arbeitsnummer, unter der die Datei geöffnet ist.
  Das bedeutet, daß nur geöffnete Dateien gelöscht werden können.

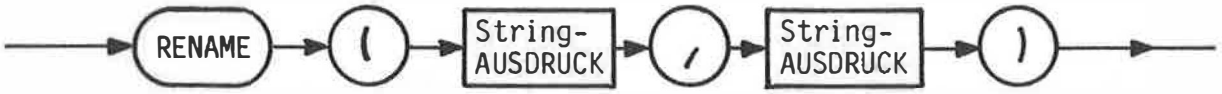
- Die Ausdrücke müssen im Bereich von 1 bis 20 liegen.
  REAL-Ausdrücke werden nach INTEGER gewandelt.

- Alle aktiven IF-END-Anweisungen (siehe Abschnitt 5.7) mit dieser (diesen) Arbeitsnummer(n) haben keine Auswirkung mehr.

Beispiele:
```
   DELETE  2  ,  14
   DELETE  INDEX%
25 DELETE  NR%  ,  20  ,  I%
   DELETE  ANZ%-1  ,  ZAHL*2-2
```

## 5.7 Die Datei-Funktion RENAME

Mit Hilfe dieser Funktion läßt sich vom CBASIC-Programm her ein Dateiname auf der Diskette ändern.




- Der erste String-AUSDRUCK beinhaltet den neuen Dateinamen, der zweite String-AUSDRUCK bezeichnet den zu ändernden Dateinamen.

- Die Funktion RENAME gibt eine -1 (=TRUE) zurück, wenn die Umbenennung erfolgreich war, ansonsten eine Null (=FALSE).

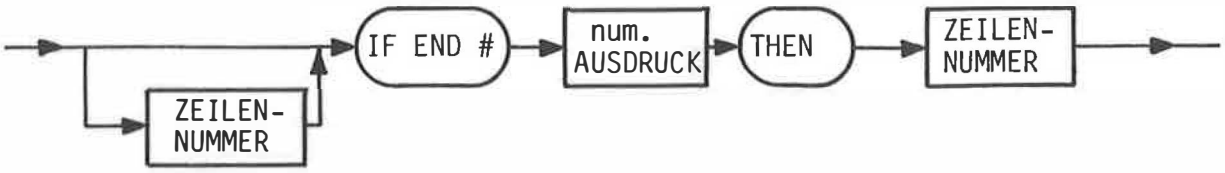
- Beim Umbenennen einer Datei sollte diese noch geschlossen sein oder aber mit CLOSE geschlossen werden, da es sonst zu Fehlern kommen könnte.

Beispiele:
---------

```
FEHLER% = RENAME ("DATEI2.DAT","DATEI1.DAT")
IF  RENAME ("NEU.DAT","ALT.DAT")    THEN 100
IF  RENAME (FILE1$,FILE$)   THEN  ........  \
                            ELSE  ........
```

## 5.8 Die IF-END-Anweisung

Mit dieser Anweisung kann das Ende einer Datei (End of file) "abgefangen" werden, um Laufzeitfehler im Programm zu vermeiden.




- Der numerische Ausdruck nach IF-END # beinhaltet wiederum die logische Arbeitsnummer, die mit einer OPEN-, CREATE- oder FILE-Anweisung vergeben wurde.

- Wird während der Programmdurchführung das Ende einer Datei erreicht, so wird ein Laufzeitfehler gemeldet, wenn keine IF-END-Anweisung für diese Datei programmiert wurde.
  Existiert eine entsprechende IF-END-Anweisung, so findet ein Sprung auf die nach THEN stehende Zeilenummer statt.

- Die IF-END-Anweisung muß die einzige Anweisung einer Zeile sein.
  Sie darf nicht Element einer Anweisungsliste sein.

- Es können beliebig viele IF-END-Befehle bezogen auf eine logische Arbeitsnummer in einem Programm vorkommen.
  Jeweils die letzte IF-END-Anweisung ist aktiviert.

- Wird diese Anweisung beim Beschreiben einer Datei aktiviert, so ist die Ende-Bedingung erst dann gegeben (TRUE), wenn in der Datei kein weiterer Platz zum Beschreiben existiert.

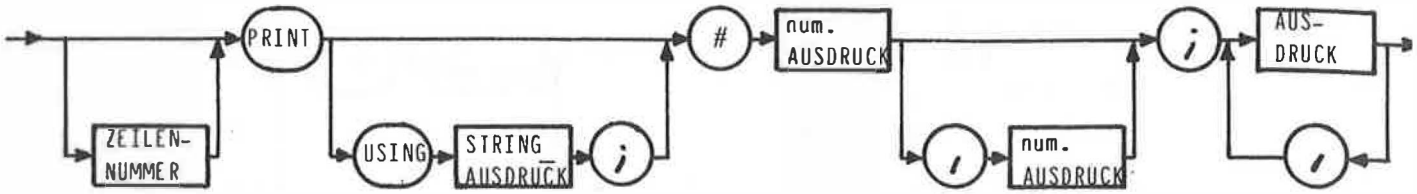
Beispiele:
```
IF END #   3  THEN 350
IF END #   NR%  THEN  20
IF END #   ZAHL*3-X   THEN  45
```

Programmbeispiel:

```
        IF END #   DATEI%  THEN 400
        OPEN  "B:KUNDEN.DAT"  RECL  100   AS  DATEI%
        ..........
        GOTO 40
   400  CREATE  "B:KUNDEN.DAT"   RECL  100  AS  DATEI%
    40  IF END #  DATEI%  THEN 500
        .......
```

## 5.9 Beschreiben einer Datei mit der PRINT-#-Anweisung

Diese Anweisung ermöglicht es dem Programmierer, eine Datei zu beschreiben.




- Der erste numerische Ausdruck nach # bezeichnet die logische Arbeitsnummer unter der die gewünschte Datei aktiviert wurde.

- Der zweite numerische Ausdruck muß nur bei Random-Dateien angegeben werden und enthält die gewünschte Satznummer innerhalb der Datei.
  REAL-Ausdrücke werden in INTEGER-Ausdrücke verwandelt und intern als (positive) Dualzahlen dargestellt.
  Hierdurch können Satznummern von 1 bis 65535 zur Anwendung kommen.

- Die nach dem Semikolon anzugebenden Ausdrücke werden durch Komma voneinander getrennt und müssen den Regeln für Ausdrücke entsprechen.
  String-Konstante sind durch das Zeichen (") einzuschließen.

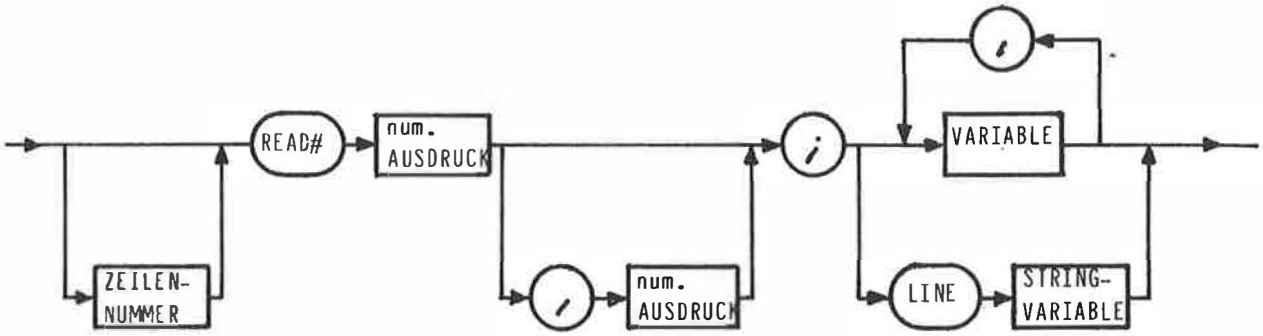
- Das Beschreiben von Dateien kann auch formatiert erfolgen.
  Hierzu ist die Befehlserweiterung USING zu verwenden.
  Die Bedeutung der im String-Ausdruck nach USING anwendbaren Zeichen können den Erläuterungen zum allgemeinen PRINT-USING-Befehl (siehe Abschnitt 4.14) entnommen werden.

Beispiele:
---------

```
PRINT #  7; "Montag : " , NAME$
PRINT #  NR%; STRASSE$ , PLZ$ , ORT$
PRINT #  DATEI%,INDEX%; KUNDEN.NR% , NAME$ , UMSATZ
PRINT #  3 , K% ; X , Y , ERG
PRINT  USING FORMAT$ ; # NR% , SATZ%; VERZEICHNIS$ , ART$
PRINT  USING ZAHL$ ; # DATEI% ; X1 , X2 , X3 , X4
```

## 5.10 Lesen aus einer Datei mit der READ-#-Anweisung

Diese Anweisung ermöglicht es dem Programmierer, Daten aus einer Datei in den Arbeitsspeicher zu lesen




- Der erste numerische Ausdruck nach # bezeichnet die logische Arbeitsnummer, unter der die gewünschte Datei aktiviert wurde.

- Der zweite numerische Ausdruck muß nur bei Random-Dateien angegeben werden und enthält die gewünschte Satznummer innerhalb der Datei.
  REAL-Ausdrücke werden in INTEGER-Ausdrücke verwandelt und intern als Dualzahlen dargestellt.
  Hierdurch können Satznummern von 1 bis 65535 zur Anwendung kommen.

- Die nach dem Semikolon anzugebenden Variablen werden durch Komma voneinander getrennt.
  Die zu lesenden Variablen müssen natürlich vom gleichen Datentyp sein, wie die entsprechenden Ausdrücke, die mit der PRINT # - Anweisung in diesen Satz geschrieben wurden, um identische Daten zu erhalten.

- Eine Sonderform der READ-#-Anweisung stellt der Parameter LINE dar. Hierdurch kann der Inhalt eines ganzen Satzes (bis zum nächsten RETURN und LINE-FEED) in eine String-Variable eingelesen werden.
  Diese Sonderform ist möglich, da alle Daten (auch numerische Daten) im ASCII-Format in die Dateien geschrieben werden.

Beispiele:
---

```
READ #  7; TEXT$ , NAME$
READ #  NR%; STRASSE$ , PLZ$ , ORT$
READ #  DATEI%,INDEX%; KUNDEN.NR% , NAME$ , UMSATZ
READ #  3 , K% ;  X , Y , ERG
READ #  7,25; LINE ZEILE$
```

## 5. Programmier-Aufgabe

Dieses Beispiel soll Sie mit den Anweisungen zur Dateiverwaltung vertraut machen.

Schreiben Sie ein Programm, das die 4 Teilaufgaben einer Kundendaten-Verwaltung:

- erfassen
- listen
- ändern

und - löschen übernimmt.

Die Kundendaten sollen in einer RANDOM-Datei mit fester Satzlänge abgelegt werden, um direkten Zugriff zu jeder Kundennummer zu ermöglichen.

Es sollen nur folgende Kundendaten erfaßt und gespeichert werden :

- Kundennummer (zwischen 1 und 100)
- Kundenname (max. 20 Zeichen)
- Umsatz des Kunden (als REAL-Zahl)

Obwohl die Kundennummer in diesem Beispiel gleich der Satznummer innerhalb der Datei ist, soll sie mit im jeweiligen Kundensatz gespeichert werden.

Denken Sie daran, daß alle Daten im ASCII-Format auf der Diskette abgespeichert werden und berücksichtigen Sie dies bei der Definition der Satzlänge. Sie müssen also die maximal erforderliche Anzahl an Byte berechnen (siehe 5.2 - Record-Länge).

Dem Bediener soll ein Menue zwischen

- Erfassen von Kundendaten = 1
- Listen von Kundendaten = 2
- Ändern von Kundendaten = 3
- Löschen von Kundendaten = 4
- Programm-Ende = 5

angeboten werden.

Alle 4 Programmblöcke sollen jeweils zu Beginn den Bildschirm löschen, ansonsten ist (aus Zeitgründen) keine besondere Bildschirm-Maske vorgeschrieben.

MODUL "Erfassen von Kundendaten"

Das "Erfassen von Kundendaten" wird solange wiederholt, bis der Bediener als Kundennummer eine NULL eingibt.
Fehlerhafte Kundennummern (kleiner 0 oder größer 100) oder bereits eingetragene Kundennummern werden vom Programm erkannt und es wird eine entsprechende Fehlermeldung ausgegeben.

MODUL "Listen von Kundendaten"

Beim "Listen von Kundendaten" kann der Bediener zwischen den Ausgabegeräten Bildschirm und Drucker wählen.
Nach der Ausgabe aller eingetragenen Kundendaten findet nach der Eingabe eines beliebigen Zeichens ein Rücksprung zum Menue statt. Bei Ausgabe der Kundendaten müssen die Kundennummern und die Umsatzzahlen rechtsbündig untereinander stehen (PRINT USING).

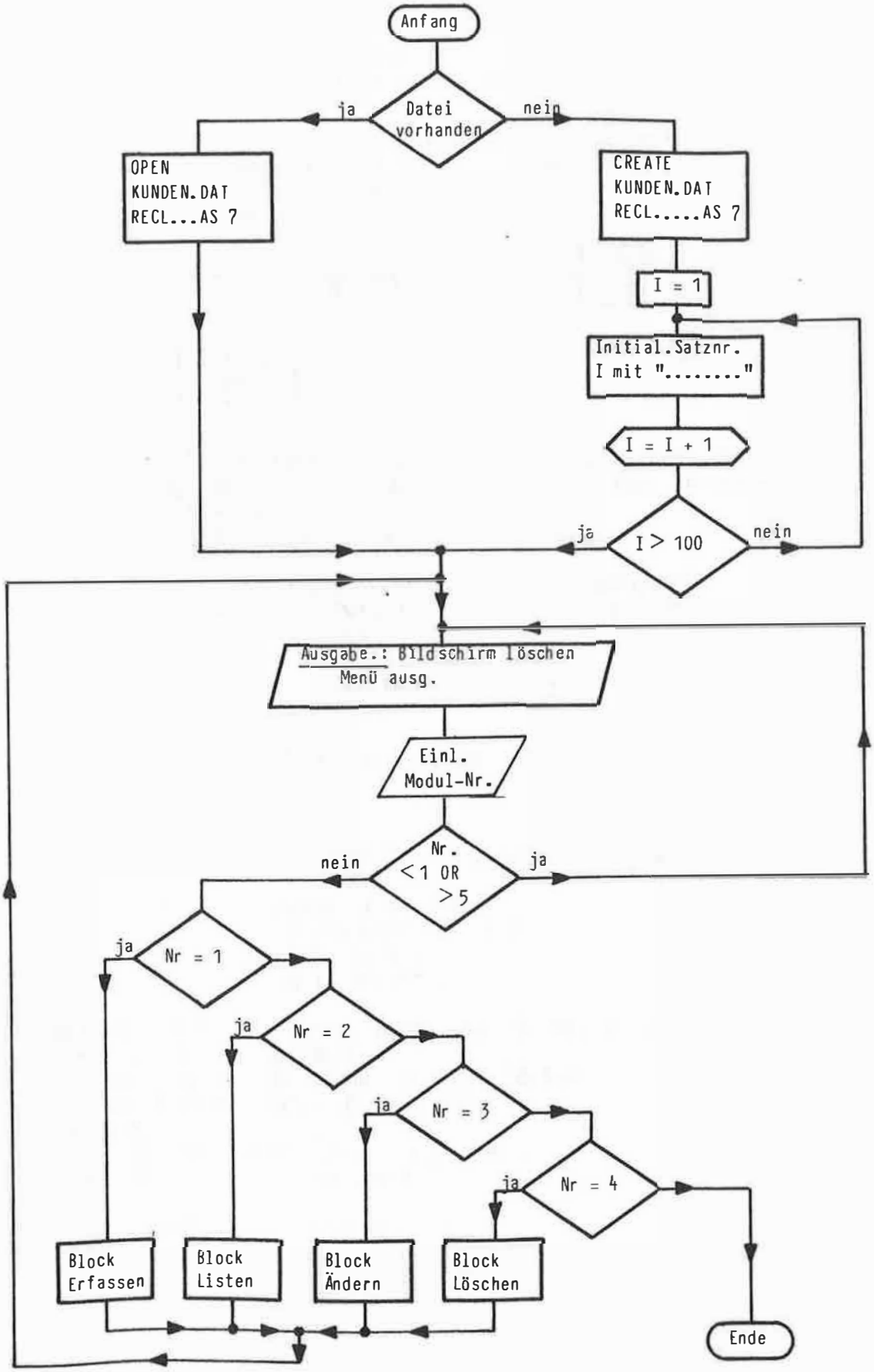
MODUL "Ändern von Kundendaten"

Vom Programmbediener muß die Nummer des zu "ändernden" Kunden eingegeben werden.
Hierauf werden vom Programm die zu dieser Kundennummer gehörenden Daten auf dem Bildschirm angezeigt. Nun kann der Bediener den Kundennamen und / oder den Kundenumsatz ändern.
Will der Bediener eine Information nicht (!) ändern, so muß er als erstes Zeichen die Taste <CR> drücken.
Nach dem Einlesen einer oder beider Änderungen werden die neuen Werte in die Kundendatei auf der Diskette geschrieben.

MODUL "Löschen von Kundendaten"

Vom Programmbediener muß die Nummer des zu "ändernden" Kunden eingegeben werden.
Hierauf werden vom Programm die zu dieser Kundennummer gehörenden Daten auf dem Bildschirm angezeigt.
Anschließend wird der Bediener aus Sicherheitsgründen noch einmal gefragt, ob diese Kundeneintragung gelöscht werden soll.
Nur wenn diese Frage mit "JA" beantwortet wird, erfolgt die Löschung bzw. Initialisierung dieser Kundennummer.

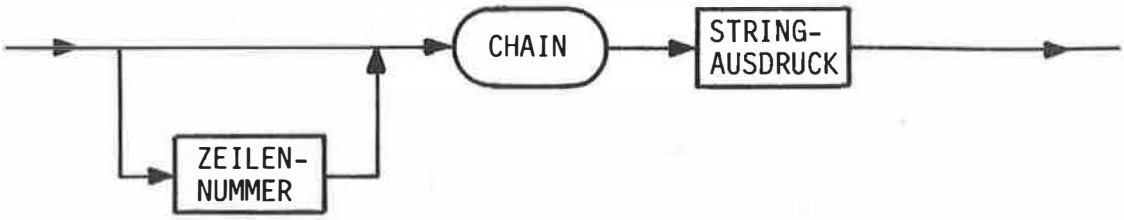
Darstellung der Aufgabe 5 im Grobflussdiagramm



# 6. Programmverkettung, Variablenübergabe und Diskettenwechsel

## 6.1 Programmverkettung mit CHAIN

Hiermit können übersetzte CBASIC-Programme gekettet werden.
Man kann mit dieser Anweisung aus einem Programm ein anderes Programm von der Diskette aufrufen und durchführen.
Das aufrufende Programm wird dabei überlagert.




- Der hinter CHAIN stehende String-AUSDRUCK bezeichnet das übersetzte (!) CBASIC-Programm, das von der Diskette in den Speicher geladen und durchgeführt werden soll.
  Das Programm muß also vom File-Typ "INT" sein.

- ACHTUNG: Das rufende Programm wird vom aufgerufenen Programm überlagert. Es handelt sich hierbei um keine Segmentierung oder Overlaytechnik.
  Sollen Speicherdaten an das neue Programm übergeben werden, so muß die Anweisung COMMON verwendet werden (siehe Abschnitt 6.2).

- Durch eine CHAIN-Anweisung werden alle offenen Dateien automatisch geschlossen.

- Während der Programmdurchführung existieren 4 Bereiche im Speicher:

  - Konstanten-Bereich
  - Programmcode-Bereich
  - DATA-Bereich
  - Variablen-Bereich

  Die vom Programm jeweils benötigten Bereiche werden nach dem Übersetzungslauf mit ausgegeben.
  Ist einer dieser Bereiche im durch CHAIN aufgerufenen Programm größer als im Hauptprogramm (Erstprogramm), so führt dies zu einer Fehlermeldung.
  Mit Hilfe der %CHAIN-Anweisung kann der Programmierer dafür sorgen, daß die Speicherbereiche in der maximal benötigten Größe reserviert werden.

  Finden mehrere "Verkettungen" durch CHAIN statt, so muß jeweils der größte Bereich (siehe folgendes Beispiel) in der %CHAIN-Anweisung eingesetzt werden.

| Beispiel: | 1. Progr. | 2. Progr. | 3. Progr. |
|---|---|---|---|
| Konst.-B. | 8 | 32 x | 16 |
| Code-B. | 1648 | 960 | 2438 x |
| DATA-B. | 0 | 16 | 48 x |
| Var.-B. | 142 x | 96 | 84 |

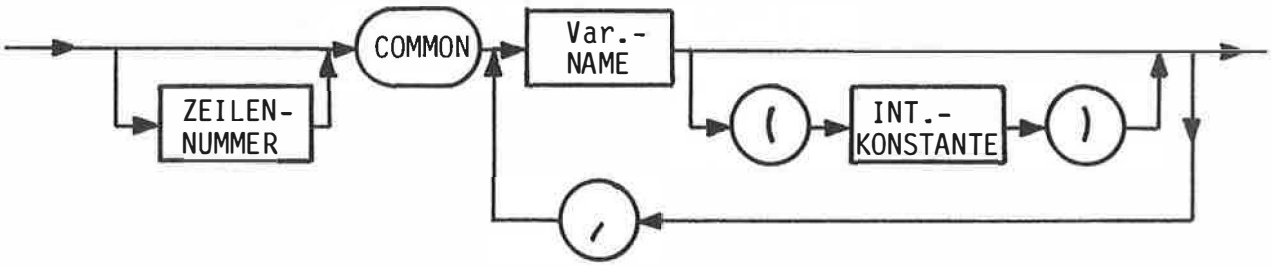
Der erforderliche %CHAIN-Befehl im Hauptprogramm muß heißen:

```
%CHAIN  32, 2438, 48, 142
```

Diese Anweisung muß ab der Spalte 1 stehen!

## 6.2 Übergabe von Variablen mit COMMON

Wird eine Programmverkettung mit der CHAIN-Anweisung durchgeführt, so werden alle im Arbeitsspeicher befindlichen Daten des aufrufenden Programms zerstört. Mit Hilfe der COMMON-Anweisung können Variable als globale Variable definiert werden, auf die dann alle Programme zugreifen können.




- Die COMMON-Anweisung muß die erste Anweisung im Programm sein.
  Nur REM-Anweisungen sind von dieser Regel ausgenommen.

- Steht eine COMMON-Anweisung im "Hauptprogramm", so muß jedes mit CHAIN aufgerufene Programm eine entsprechende COMMON-Anweisung enthalten.
  Das bedeutet, daß die Anzahl und der Datentyp der Variablen in allen COMMON-Anweisungen identisch sein muß.
  Die Variablen-Namen können jedoch in jedem Programm frei gewählt werden.

- Auch dimensionierte Variable müssen einander entsprechen.
  Anders als in anderen Programmiersprachen muß bei einer dimensionierten Variablen die Anzahl der Dimensionen (und nicht die Feldgröße) in der COMMON-Anweisung mit angegeben werden.
  Diese Angabe ist nicht identisch mit den Angaben in der DIM-Anweisung.

```
z.B.  COMMON  X(2)
      DIM  X (10,25)
```

## Programmbeispiel zu den Anweisungen CHAIN und COMMON

### "Hauptprogramm"

```
   REM  Hauptprogramm zum "Setzen" einer Tabelle
   REM        Programm-Name:       TEST
   COMMON  I%(2),  ANTW$,  E1%,  E2%
   REM ----------------------------
   DIM  I% (4,9)        \     nur im Hauptprogramm
20 INPUT "Ausgabe auf Drucker oder Bildschirm (D/B) : ";  ANTW$
   IF  ANTW$ <> "D"  AND  ANTW$ <>  "B"   THEN 20
   LET  E1% = 9 :  E2% = 4

   FOR  J1% = 0  TO  E1%
   FOR  J2% = 0  TO  E2%
   I% (J1%,J2%) = J1% * 10 + J2%
   NEXT  J2%,  J1%

   CHAIN  "TEST2"       \     Aufruf von  TEST2.INT
   STOP
   END
```

### "Kett-Programm"

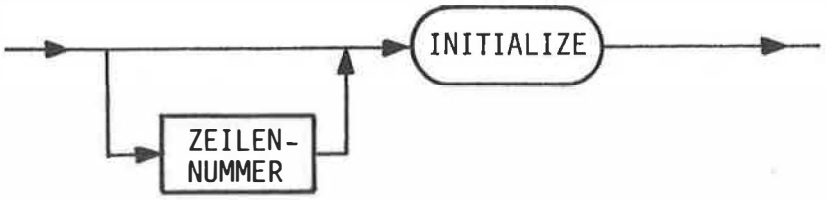
```
   REM  Quellprogramm zum Ausdrucken der Tabelle
   REM        Programm-Name :  TEST2
   COMMON  ZAHL%(2),  JA$,  A%,  E%
   -------------------------------
   IF  JA$ = "D"     THEN  LPRINTER

   FOR  K1% = 0  TO  A%
   FOR  K2% = 0  TO  E%
   PRINT  ZAHL% (K1%,K2%) ;
   NEXT  K2%
   PRINT
   NEXT  K1%

   STOP
   END
```

## 6.3 Diskettenwechsel mit INITIALIZE

Diese Anweisung ist immer dann notwendig, wenn während der Programmdurchführung eine Diskette gewechselt werden muß, um z.B. zusätzliche Daten im Zugriff zu haben.




- Das Betriebssystem führt ein internes Disketten-Inhaltsverzeichnis.
  Wird ein Diskettenwechsel dem System nicht "mitgeteilt", so arbeitet es mit falschen Informationen weiter und es kann zu gravierenden Fehlern kommen.
  Mit der INITIALIZE-Anweisung wird dem System ein Diskettenwechsel mitgeteilt und es kann die internen Disketteninformationen aktualisieren.

- Bevor ein INTIALIZE-Kommando durchgeführt wird, müssen grundsätzlich alle Dateien durch das Anwenderprogramm geschlossen werden.
  Wird dieser Grundsatz nicht beachtet, kann es zum Verlust aller neu eingefügten Daten kommen.

- Die INITIALIZE-Anweisung bezieht sich auf alle "eingeschalteten" Laufwerke. Der Programmierer muß also kein bestimmtes Laufwerk definieren.
  Das angewählte Laufwerk bleibt auch nach der Durchführung der INITIALIZE-Anweisung im Zugriff.

- Denken Sie daran, daß Sie dem Anwender einen notwendigen Diskettenwechsel anzeigen müssen und ihm anschließend auch entsprechende Zeit zur Verfügung stellen, damit er den Wechsel durchführen kann.
  Erst danach, wenn also die neue Diskette im Laufwerk und dieses eingeschaltet ist, darf das Kommando INITIALIZE durchgeführt werden.

Beispiele:
---

```
123 INITIALIZE
    IF NUMMER% 5000 THEN INITIALIZE
    INITIALIZE
```

## 7.0 Weitere CBASIC-Anweisungen

In diesem Kapitel werden weitere zweckmäßige und mögliche CBASIC-Anweisungen bzw. -Funktionen knapp beschrieben.
Aus Zeitgründen lassen sich diese Anweisungen nicht mehr in ausführlicher Form in das vorliegende 40-Unterrichtsstunden umfassende Schulungskonzept integrieren.

Bei Verwendung dieser Anweisungen muß die Original-CBASIC-Beschreibung zu Hilfe genommen werden.

### RANDOMIZE

Diese Anweisung startet den Zufallsgenerator von neuem, so daß keine reproduzierenden Zahlen erzeugt werden.

### SAVEMEM

Reservierung von Arbeitsspeicher für ein Maschinencode-Unterprogramm und laden dieses Unterprogramms in den Datenbereich.

### PEEK

Diese Funktion liefert den Inhalt einer absoluten Speicheradresse aus dem Datenbereich.

### POKE

Hiermit kann der Inhalt einer absoluten Speicheradresse gesetzt werden. Dieses gilt auch für Adressen des Betriebssystems.

### SIZE

Diese INTEGER-Funktion liefert die Größe einer auf der Diskette vorhandenen Datei.

Olympia Werke AG — Bildungszentrum — Postfach 960, 2940 Wilhelmshaven