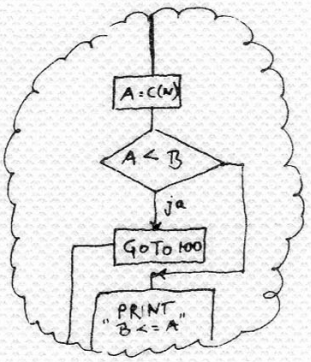
Basic Lehrbuch für SHARP Computer



FISCHEL GMBH

ISBN 3-924327-09-2

VON BERNHARD HARTMANN UND JÜRGEN BRENNER-HARTMANN

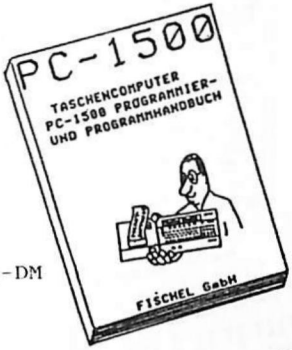
FISCHEL

BETRIEBSWIRTSCHAFTLICHER BERATUNGS- UND PROGRAMMIERDIENST GMBH

Die Unternehmensberatung

für SHARP-Computer

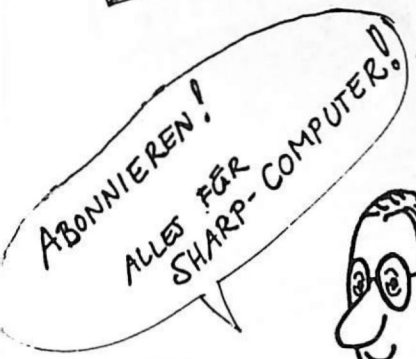
alle Preise incl. 7% MWST:



,-DM

39,-DM

49,-DM



49,-DM

39,-DM

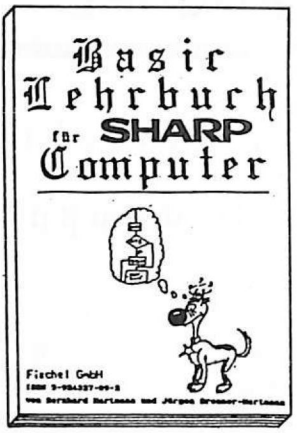
29,-DM

User-Club Deutschland

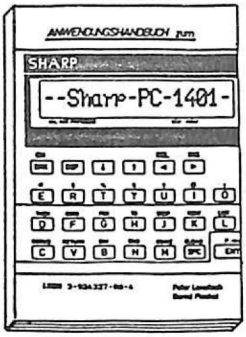
DURCH INFORMATION VORN

L-KFM. B.FISCHEL-KAISER-FRIEDRICH-STR.54A-1000BERLIN12-TEL.(030)323 60 29

65,- DM



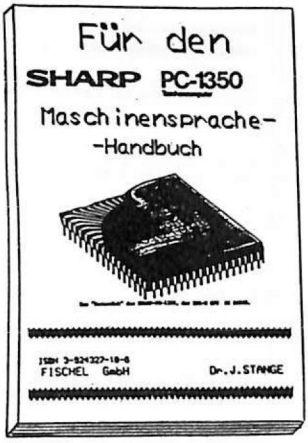
BASIC-Lehrbuch für Sharp Computer
160 Seiten; Preis: 49,-DM

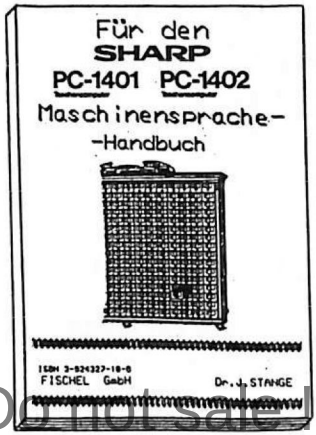


39,-DM

ALLE VERGANGENEN HEFTE SIND NOCH LIEFERBAR !!
Bitte siehe Bestellschein für ein Abonnement. 6,-DM pro Heft

49,-DM





59,-DM

49,-DM

FISCHEL GMBH, KAISER-FRIEDRICH STR. 54a, 1000 BERLIN 12
Tel. 030/3236029

Bernhard Hartmann und
Jürgen Brenner-Hartmann (Autoren)
Fischel GmbH (Hrsg.) :

BASIC-Lehrbuch für SHARP Computer

Berlin, 1985
ISBN 3-124327-09-2

WICHTIGER HINWEIS

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle technischen Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und wiedergegeben. Trotzdem sind Fehler nicht ganz auszuschließen. Der Herausgeber und die Autoren sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch eine juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind die Autoren jedoch jederzeit dankbar.

SHARP-COMPUTER
FISCHEL GMBH
KAISER-FRIEDRICH-STR. 54A
1000 BERLIN 12

TEL.: 030 / 323 60 29

DRUCK: Offsetdruckerei Gerhard Weinert
Friedrichstraße 224, 1000 Berlin 61

INHALT

V O R W O R T

Gibt es denn nicht schon genug BASIC-Bücher? Wir meinen - nein! Sie werden uns sicher zustimmen, zumal, wenn Sie Besitzer eines Pocket-Computers sind. Die überwiegende Mehrzahl der bisher erschienenen BASIC-Einführungen setzen ein ausgewachsenes Bildschirmgerät voraus und wie oft schauten Sie dabei in die Röhre. Anders hier: Alle in diesem Buch wiedergegebenen Programme laufen auch auf den PCs von SHARP. Sollten Sie stolzer Besitzer eines "Großen" sein, brauchen Sie deshalb das Buch nicht enttäuscht wegzulegen. Was die Pockets können, kann Ihrer allemal. Die verwendeten BASIC-Befehle sind zudem so ausgewählt, daß alle SHARPs parieren.

Die Programme, die Sie in dieser Einführung finden, sind sämtlich von uns auf dem PC-1500 A getestet. Wo es Unterschiede zwischen den einzelnen Rechnertypen geben könnte, ist darauf hingewiesen. In diesen Fällen hilft Ihnen immer Ihre Gebrauchsanleitung weiter.

Einige PCs verfügen über einen sogenannten CALC-Mode, der es erlaubt, sie wie normale Taschenrechner zu benutzen. Da Sie jedoch das Programmieren lernen wollen, schalten Sie sie auf den BASIC-Mode, wenn Sie mit diesem Buch arbeiten.

Sie brauchen keine Vorkenntnisse über Computer oder Programmiersprachen, wenn Sie mit dem vorliegenden Lehrbuch BASIC lernen wollen. Die Kapitel sind so angeordnet, daß eins aufs andere aufbaut. In dieser Reihenfolge sollten Sie sie auch bearbeiten. Legen Sie Ihren Computer dabei immer neben sich, denn "ohne Wasser hat noch niemand Schwimmen gelernt". Gleiches gilt für die Übungsaufgaben, die Ihnen Gelegenheit geben sollen, in der Anwendung das Gelernte zu überprüfen und zu vertiefen.

Am Ende dieses Buches sollten Sie über die wichtigsten Werkzeuge des Programmierens verfügen. Ob Sie damit geniale Programme zimmern, hängt wesentlich auch von Ihnen und Ihrer Fähigkeit zur kreativen Problemlösung ab. Um bildlich zu sprechen: Wir können Ihnen Pinsel und Farbe geben, ein Kunstwerk müssen Sie daraus machen.

Wir sind jedoch sicher, daß Sie die Lust am Programmebasteln nicht mehr loslassen wird, wenn Sie erst einmal "pinseln gelernt" haben, denn mit der Computerei ist es wie bei so vielem: Der Appetit kommt mit dem Essen.

In diesem Sinne - wohl bekomm's!

Bad Boll

Bernhard Hartmann
Jürgen Brenner-Hartmann

1. Kapitel

EINE MASCHINE STELLT SICH VOR
- Der Microcomputer und seine Komponenten -

1.1 Was ist ein Computer?

Haben Sie das auch schon mal in einer Werbeanzeige für ein Automobil gelesen: "Digitale Motorelektronik - Eine komplette Computersteuerung des Triebwerks"? Hier treffen Sie auf das Wort "Computer" in einer der alltäglichsten Sachen der Welt - dem Auto.

Und es gibt noch mehr Geräte, in denen computerähnliche Apparate am Werk sind: In der Waschmaschine, der Hifi-Stereoanlage, dem Taschenrechner und der elektronischen Schreibmaschine. Um zum Auto zurückzukommen: Eine Bremsanlage, die nicht blockiert (ein sog. Antiblockiersystem), und der Katalysator wären ohne diese kleinen elektronischen Hilfsgeister nicht denkbar.

Diese "intelligenten" Geräte kontrollieren, steuern, regeln und, im Falle des Taschenrechners, rechnen sie auch noch. Eines haben diese Apparate gemeinsam: Sie sind meistens für eine spezielle Aufgabe konstruiert, z.B. um eine Waschmaschine zu steuern oder eben die Bremsanlage eines Autos zu regeln. Wehe, wenn die Elektronik der Waschmaschine mit der eines Antiblockiersystems vertauscht würde! Vielleicht käme das Auto ins Schleudern und die Waschmaschine würde ständig und blockadefrei bremsen?

Sie sehen: Für jede neue Aufgabe braucht man eine neue Elektronik, ein neues "computerähnliches System". Ich habe bewußt den Ausdruck "computerähnliches System" gebraucht, denn ein Computer ist doch noch etwas anderes. Ein Computer kann nämlich alle Aufgaben bewältigen, die die "computerähnlichen Systeme" lösen, ohne dafür die Elektronik wechseln zu müssen. Ein Computer ist also eine Maschine, die für jede Anwendung im Bereich des Messens, Steuerns, Regelns und der Informationsverarbeitung programmierbar ist. Verändert sich die Aufgabe, muß lediglich das Programm geändert werden.

In diesen Rahmen fallen auch folgende "Jobs", die der Computer erledigen kann. Ich will Ihnen nur einige Schlagworte aufzählen: Texte verarbeiten, Adressen verwalten, Experimente auswerten, Buch- und Kontoführung, Kfz-Diagnose, Steuerung von Schiffen und Flugzeugen und, wenn Sie sich langweilen, können Sie sogar mit Ihrem Computer spielen.

Wie Sie Ihren Microcomputer in der Sprache BASIC programmieren, damit er das tut, was Sie von Ihm verlangen, will Ihnen dieses Buch zeigen.

1.2 Zahlensysteme

Bevor wir uns näher mit dem Computer befassen, habe ich ein kleines

mathematisches Attentat auf Sie vor. Es geht dabei um verschiedene Zahlensysteme, genauer gesagt, um das Dezimal-, das Dual- und das Hexadezimalsystem. Da Sie vermutlich Ihr Leben lang im Dezimalsystem gerechnet haben, der Computer hingegen nur das Dualsystem kennt, scheint mir diese Attacke gerechtfertigt.

Beginnen wir mit dem Dezimalsystem. Dieses Zahlensystem, Sie wissen es seit Ihren ersten Schuljahren, hat 10 Ziffern:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Sie nannten sie damals "Einer". Die nächste Zahl ist die 10, d. h. die "Zehner" beginnen. Sie zählen weiter bis ...99, 100 - die "Hunderter" treten auf den Plan. Bei ...999, 1000 sind sie wahrscheinlich längst eingeschlafen, trotzdem fangen jetzt die Tausender an, usw., usw....

Die Ziffern der Zahl 6 348 z.B. können Sie so in Einer-, Zehner-, Hunderter- und Tausenderkästchen einsortieren:

Tausender	Hunderter	Zehner	Einer
6	3	4	8

Statt Einer, Zehner, Hunderter usw. hat man in der Mathematik die Potenzschreibweise eingeführt. Dabei ist:

10^0 = 1 (Einer)

10^1 = 10 (Zehner)

10^2 = 10x10 = 100 (Hunderter)

10^3 = 10x10x10 = 1000 (Tausender)

Bei einer Zahl wie 10^3 nennt man die 3 "Hochzahl" oder "Exponent" und die 10 heißt "Basis". Das Dezimalsystem ist also ein System zur Basis 10. Jetzt können wir die 6 348 auch so schreiben:

10^3	10^2	10^1	10^0
6	3	4	8

Oder in der Summenschreibweise:

$$6\,348 = 6x10^3 + 3x10^2 + 4x10^1 + 8x10^0$$

Nun schauen wir uns mal das Dual- oder Binärsystem an. Hier gibt es nur zwei Ziffern: 0 und 1. Zählen Sie weiter, gelangen sie zur 2, die wird im Dualsystem aber so dargestellt: 10 (lies: "Eins-Null", nicht

"Zehn"). Statt der 3 erhalten wir 11 (also "Eins-Eins", nicht "Elf"), aus der vier wird 100. Die folgende Tabelle zeigt Ihnen die Zahlen von 0 bis 9 in Dezimal- und Dualschreibweise.

dezimal	0	1	2	3	4	5	6	7	8	9
dual	0	1	10	11	100	101	110	111	1000	1001

Die Basis des Dualsystems ist 2, die Ziffern einer Dualzahl werden also als Potenzen von 2 geordnet. Bei einer Zahl wie z.B. 10110100 sieht das so aus:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	1	0	1	0	0

oder in der Summenschreibweise:

$$10110100 = 1x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$$

Es ergeben sich also folgende 2er Potenzen:

$2^0 = 1$

$2^1 = 2$

$2^2 = 2x2 = 4$

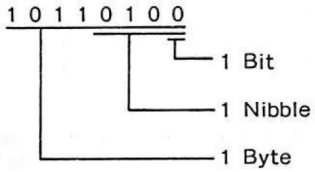
$2^3 = 2x2x2 = 8$

usw.

Bevor wir mit dem Hexadezimalsystem loslegen, müssen wir einige grundlegende Begriffe der Datenverarbeitung klären. Der Computer arbeitet bekannterweise mit binären Daten, etwa mit Zahlen, die sich im Dualsystem als Folge von Nullen und Einsen darstellen lassen. Die kleinste Informationseinheit, d.h. eine "Stelle" einer Binärzahl (egal, ob 0 oder 1), wird ein Bit genannt (von Binary Digit). Die Zahl 10110100 besteht also aus 8 Bit. Wenn der Computer fordert: "Bitte, ein Bit", dürstet ihn nicht nach Bier, sondern nach besagter kleinsten Dateneinheit.

Die nächstgrößere Einheit ist 1 Nibble. 1 Nibble hat 4 Bit. Von grösserer Bedeutung ist das Byte. 1 Byte besteht aus 2 Nibble oder 8 Bit. Wir werden dem Byte bald wieder begegnen, wenn wir das Innenleben des

Computers genauer unter die Lupe nehmen. Schauen Sie sich diese Einheiten noch einmal an, bevor wir weiter machen:



Als Maß für die Speicherkapazität eines Computers wird oft das Kbyte verwandt. Dabei ist:

$$1\ \text{Kbyte} = 2^{10}\ \text{Byte} = 1024\ \text{Byte}$$

Das ist also etwas mehr als 1 Kilobyte = 1000 Byte. (Oft sagt man aber "1 Kilobyte" und meint 1024 Byte).

Und nun, wie versprochen, das Hexadezimalsystem oder "Sechzehnersystem". Der Name läßt nichts Gutes ahnen, und tatsächlich gibt es hier sechzehn Ziffern, wie folgende Tabelle zeigt:

dezimal	0	1	2	3	4	5	6	7	8	9
hexadezimal	0	1	2	3	4	5	6	7	8	9
dezimal	10	11	12	13	14	15	16	17		
hexadizimal	A	B	C	D	E	F	10	11		

Für die Dezimalzahlen 10, 11, ..., 15 stehen also die Hexadezimalziffern A, B, ..., F. Und noch etwas ist herb: Die Ziffern einer Hexadezimalzahl ordnet man nach Potenzen von 16. Die Hexadezimalzahl A74B z.B. wird so in die mittlerweile bekannten Kästchen einsortiert:

16^3	16^2	16^1	16^0
A	7	4	B

Als Summe schreibt sich das:

$$A74B = Ax16^3 + 7x16^2 + 4x16^1 + Bx16^0$$

Die zugehörigen 16er Potenzen berechnet man auf diese Weise:

$16^0 = 1$

$16^1 = 16$

$16^2 = 16 \times 16 = 256$

$16^3 = 16 \times 16 \times 16 = 4\,096$

Nun wäre es praktisch, wenn wir die Zahlensysteme ineinander umrechnen könnten, wie verschiedene Währungssysteme auch. Damit wir aber dann noch wissen, welche Zahl aus welchem System stammt, geben wir ihnen einen Index, das ist eine Art Nummernschild, welches der Zahl rechts unten angehängt wird. So ist 10110100_2 eine Dualzahl, $6\,348_{10}$ eine Dezimalzahl und $A74B_{16}$ eine Hexadezimalzahl.

Jetzt können wir getrost mit der Umwandlung beginnen, wobei zuerst Zahlen des Dual- und des Hexadezimalsystems in solche des Dezimalsystems umgeformt werden. Immer, wenn Sie eine Zahl aus einem beliebigen System in eine Dezimalzahl verwandeln, gehen Sie nach diesem Schema vor:

1. Schreiben Sie die betreffende Zahl als Summe von Potenzen ihrer Basis (hier: 2) auf:

$$10110100_2 = 1x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 2^7 + 2^5 + 2^4 + 2^2$$

2. Rechnen Sie die Glieder der Summe aus:

$$10110100_2 = 128 + 32 + 16 + 4$$

3. Bilden Sie die Summe:

$$10110100_2 = 180_{10}$$

Damit Sie sich an dieses Schema gewöhnen, rechnen wir gleich die Hexadezimalzahl $A74B_{16}$ in eine Dezimalzahl um:

1. $A74B_{16} = Ax16^3 + 7x16^2 + 4x16^1 + Bx16^0$

2. Der zweite Schritt wird jetzt etwas komplizierter, da die hexadezimale A der dezimalen 10 entspricht und die B_{16} der 11_{10}.

$$A74B_{16} = 10x4\,096 + 7x256 + 4x16 + 11x1$$

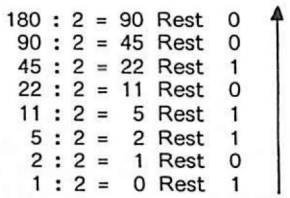
$$A74B_{16} = 40\,960 + 1\,792 + 64 + 11$$

3. $A74B_{16} = 42\,827_{10}$

Der zweite Akt der Umwandlungen kann beginnen. Jetzt formen wir Zahlen des Dezimalsystems in Dual- bzw. Hexadezimalzahlen um. Auch dafür gibt es einen Trick, der Ihnen Dezimalzahlen in Zahlen aller möglichen an-

deren Systeme verwandelt. Probieren wir ihn doch gleich aus, indem wir aus der dezimalen 180 wieder eine Dualzahl machen. Dazu verwenden wir

1. das folgende Schema:



```
180 : 2 =  90 Rest  0
 90 : 2 =  45 Rest  0
 45 : 2 =  22 Rest  1
 22 : 2 =  11 Rest  0
 11 : 2 =   5 Rest  1
  5 : 2 =   2 Rest  1
  2 : 2 =   1 Rest  0
  1 : 2 =   0 Rest  1
```

Ich glaube, das Schema erklärt sich von selbst.

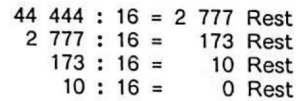
2. Lesen Sie die Reste von unten nach oben (in Pfeilrichtung) und schreiben Sie sie von links nach rechts auf:

$1\ 0\ 1\ 1\ 0\ 1\ 0\ 0_2$

So erhalten Sie, oh Wunder, die Binärdarstellung von 180_{10} .

Ob auch die Verwandlung z.B. der Dezimalzahl 44 444 in eine Hexadezimalzahl klappt? Natürlich! Also:

1. Arbeiten Sie wieder mit obigem Schema.



```
44 444 : 16 = 2 777 Rest  12
 2 777 : 16 =   173 Rest   9
   173 : 16 =    10 Rest  13
    10 : 16 =     0 Rest  10
```

2. Lesen Sie die Reste von unten nach oben und schreiben Sie sie von links nach rechts auf:

10 13 9 12

Was jetzt noch wie ein verkrüppelter Lottotip aussieht, ist tatsächlich eine Hexadezimalzahl, denn es war und ist: $10_{10} = A_{16}$, $13_{10} = D_{16}$ und $12_{10} = C_{16}$. Dann wird:

$10\ 13\ 9\ 12 = AD9C_{16}$

Na - ist das nicht eine schöne Hexadezimalzahl? Sie entspricht der Dezimalzahl 44 444. Wenn Sie mir nicht glauben, wandeln Sie sie doch wieder in eine Dezimalzahl um.

Jetzt müssen Sie nur noch lernen, wie man Dualzahlen in Hexadezimalzahlen und umgekehrt überführt. Mit folgender Tabelle geht dies recht einfach:

binär	dezimal	hexadezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Tab. 1: Die ersten 16 Zahlen des Dual-, Dezimal- und Hexadezimalsystems

Angenommen, Sie wollen aus der Binärzahl 1010110111001 eine Hexadezimalzahl machen. Dann müssen Sie

1. die Dualzahl von rechts nach links in 4er Gruppen, also Nibble, unterteilen:

   0001 0101 1011 1001

   4. Nibble 3. Nibble 2. Nibble 1. Nibble

   Beim 4. Nibble fällt Ihnen auf, daß es ja gar keine vier Bits besitzt, weshalb wir es einfach mit Nullen aufgefüllt haben.

2. Wie Sie sicher schon längst bemerkten, sind in Tab. 1 die Dualzahlen bereits als Nibble aufgelistet. Suchen Sie jetzt aus der

Tabelle für jedes Nibble in 1. die entsprechende Hexadezimalziffer aus Tab. 1, dann erhalten Sie:

1. Nibble = 1001 = 9

2. Nibble = 1011 = B

3. Nibble = 0101 = 5

4. Nibble = 0001 = 1

3. Schreiben Sie die so erhaltenen Hexadezimalziffern in der gleichen Reihenfolge, wie die entsprechenden Nibble auf.

$0001\ 0101\ 1011\ 1001_2 = 15B9_{16}$

Das war's!

Wie Sie sehen, ist die Hexadezimaldarstellung sehr viel übersichtlicher als die langen Dualzahlen. Und nicht nur das: Kennt man die Ziffer einer Hexadezimalzahl, weiß man sofort, wie das entsprechende Nibble der Dualzahl aussieht. Das ist wichtig, wenn man in Maschinensprache programmiert, weshalb das Hexadezimalsystem auch geschaffen wurde.

Genauso leicht geht die Rückverwandlung einer Hexadezimalzahl in eine Dualzahl. Dabei müssen Sie nur beachten, daß einer Ziffer im Hexadezimalsystem ein Nibble (also vier Ziffern) im Binärsystem entspricht (siehe Tab. 1).

Aufgabe 1.1

Versuchen Sie doch mal, die Hexadezimalzahl ACDC in eine Dualzahl zu verwandeln!
(Die Lösung finden Sie im Anhang)

1.3 Der Computer intim - aus dem Innenleben einer Maschine

Oft wird der Computer mit dem Menschen verglichen, vielleicht deshalb, weil er von allen Maschinen noch die menschenähnlichste ist. Zwar liegen zwischen Mensch und Computer Welten, aber trotzdem ist so ein Vergleich ganz nützlich, um wichtige Begriffe aus dem Bereich der Microelektronik zu erklären.

"Hardware" ist der erste von ihnen. Wenn Sie in einem älteren English/Deutsch-Wörterbuch nachschlagen, werden Sie die Übersetzung Eisenwaren dafür finden. Damit ist schon angedeutet, was man unter "Hardware" versteht: Etwas Handfestes nämlich. Was beim Menschen der Körper mit seinen Organen, Gliedern, Nerven usw. ist, ist beim Computer die Elektronik, das Gehäuse aber auch die Peripherie, wie Monitor, Drucker, Kassetten, Disketten und deren Laufwerke. Jetzt hat es nur so Fremdworte gehagelt. Erschrecken Sie nicht, es wird alles erklärt.

Fangen wir erst mit dem eigentlichen Computer an. Dazu öffnen Sie mal in Gedanken (aber bitte wirklich nur in Gedanken) Ihren Computer. Was

würden Sie da sehen? Einige Widerstände, ein paar Kondensatoren und vor allem etliche rechteckige "Klötzchen" mit vielen Füßchen dran - Chips nennen sie die Fachleute. Das sind Siliziumscheiben, auf denen irrsinnig viel elektronische Bausteine untergebracht sind. Das Ganze ist zum Schutz gegen äußere Einflüsse in ein Kunststoffgehäuse eingegossen. Alle Bauteile (Chips, Kondensatoren, Widerstände usw.) sind auf eine Platine gelötet und mit einer verwirrenden Vielfalt von Leiterbahnen untereinander verbunden. Ein sehr, sehr grober Schaltplan Ihres Computers sieht etwa so aus:

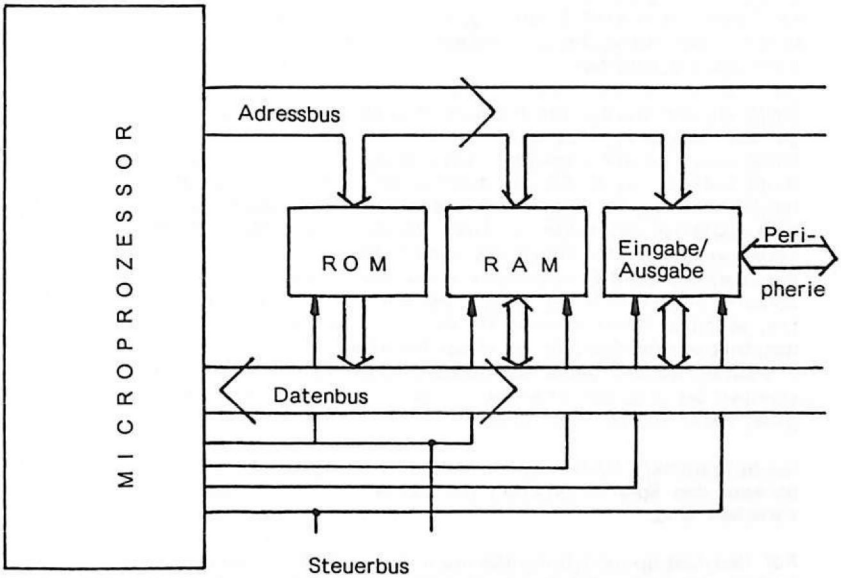


Abb. 1: Prinzipieller Aufbau eines Microcomputers

Und jetzt folgt eine sehr, sehr grobe Erklärung der einzelnen Komponenten des Computers. Sie werden dann aber immerhin wissen, wie in etwa der Microcomputer funktioniert.

Das wichtigste Teil des Computers ist der Microprozessor, auch CPU (Abkürzung von Central Processing Unit = Zentralrecheneinheit) genannt. Der Microprozessor ist der berechnende bzw. datenverarbeitende Teil des "Computergehirns". Er entspräche dem Teil des menschlichen Hirns, mit dem wir unsere (Milchmädchen-)Rechnungen machen oder einfach nur äußere Eindrücke verarbeiten (z.B. sagt Ihnen dieser Gehirnteil, daß Sie vor einer roten Ampel halten müssen). Aber, im Unterschied zum Menschen, kann der Computer nur zwei Zustände unterscheiden, nämlich, ob

in seinen Schaltkreisen eine Spannung anliegt oder nicht. Daher müssen alle Daten (Buchstaben, Zahlen usw.) als Nullen (keine Spannung) oder Einsen (Spannung vorhanden), also als Dualzahlen, codiert werden. Das ist der Grund, warum ich Sie mit dem Dualsystem gequält habe. Aber woher erhält der Microprozessor die zu verarbeitenden Daten? Und woher bekommt er die Anweisungen, die ihm sagen, wie er die Daten zu verwursteln hat? Beides wird ihm aus dem Speicher, in Abb. 1 mit RAM und ROM bezeichnet, geliefert. Dabei ist ROM eine Abkürzung für Read Only Memory, auf deutsch: "Nur-Lese-Speicher". Aus diesem Speicher kann der Microprozessor zwar Daten holen, selbst aber nichts abspeichern. Meistens befindet sich das Betriebssystem mit dem BASIC-Interpreter (diese beiden Begriffe erläutere ich Ihnen später) im ROM. Das ROM ist eine "festverdrahtete" Schaltung, kann also nie gelöscht werden, auch durch Ausschalten des Computers nicht. Es ist mit unserem Langzeitgedächtnis vergleichbar.

RAM ist das Kürzel für Random Access Memory, was sehr frei (aber sinngemäß) übersetzt wird mit: "Speicher mit wahlfreiem Zugriff". In das RAM kann die CPU sowohl Daten ablegen als auch wieder herauslesen. Das RAM enthält das BASIC-Programm, die Daten und Variablen. Ein besonderer Bereich ist für die Bildschirmgestaltung vorgesehen. Im Gegensatz zum ROM ist das RAM löschbar. Wenn Sie also Ihren Computer ausschalten, verschwindet in der Regel der Inhalt des RAM, es sei denn, daß dies durch eine Batterie verhindert wird. Aber auch während ein Programm abläuft, ändert sich der Inhalt des RAM, z.B. durch abgespeicherte Daten, ständig. Somit können Sie es mit dem menschlichen Kurzzeitgedächtnis vergleichen, in welchem Sie etwa abspeichern, was Sie heute einkaufen wollen. Sollte Sie jemand eine Woche später danach fragen, könnten Sie ihm nur erwiedern, daß es auf dumme Fragen dumme Antworten gebe, mehr wüßten Sie hierzu nicht zu sagen.

Beide Speicher, RAM und ROM, können Sie sich in Schubladen unterteilt denken, den Speicherplätzen, die mit einer "Hausnummer", einer Adresse versehen sind.

Für den Daten- und Befehlsaustausch zwischen Prozessor und Speicher besitzt der Computer drei Leitungssysteme: Daten-, Adress- und Steuerbus (Bus ist ein Fachausdruck für parallele Leitungen). Beim Menschen entsprechen diese Busse den Nervensträngen, auf denen bestimmte Reize weitergeleitet werden. Mit Hilfe des Adressbusses spricht der Microprozessor die gewünschte Speicherstelle im RAM oder ROM an, so wie Sie mit einer Telefonnummer einen anderen Fernsprechteilnehmer anwählen. Die Anzahl der Leitungen, die im Adressbus parallel zueinander laufen, bestimmt die Speicherkapazität Ihres Computers. Um dieses zu veranschaulichen, stellen Sie sich mal einen Adressbus vor, der nur aus einer Leitung besteht. Mit dieser könnten Sie sich nur zwei (2^1) Adressen aussuchen, je nachdem, ob diese Leitung Spannung führt (entsprechend einer binären 1) oder nicht (das wäre die binäre 0). Nehmen Sie nun einen Adressbus mit zwei Leitungen, dann sind 4 (2^2) Speicherzellen adressierbar, da jetzt die Kombinationen 00, 01, 10 und 11 möglich sind.

Die Adressbusse der meisten Heimcomputer besitzen 16 Leitungen. Damit sind Sie in der Lage, maximal 2^{16} = 65 536 Speicherplätze anzusprechen. Da Sie in jeder Speicherzelle ein Datenwort von 8 Bit, gleich 1 Byte,

ablegen können, verfügen Sie über eine RAM-Kapazität von 65 536 Byte. Wenn Sie sich jetzt noch erinnern, daß ein Kbyte = 1 024 Byte ist, errechnen Sie eine Speicherkapazität von 64 Kbyte, wie sie in vielen Heimcomputern üblich ist. Auf dem Adressbus werden übrigens Signale nur in einer Richtung übertragen: Von der CPU zum RAM/ROM bzw. den Eingabe/Ausgabe-Bausteinen.

Der über den Adressbus angesprochene Speicherplatz weiß nun, daß er gemeint ist. Aber jetzt stellt sich die Frage: Sollen Daten aus der Speicherzelle gelesen oder in sie hineingeschrieben werden? Das teilt ihr der Microprozessor über den Steuerbus mit. Da aus dem ROM nur gelesen werden kann, ist für ihn eine Steuerleitung ausreichend, während das RAM zwei davon benötigt (siehe Abb. 1).

Nun müssen aber noch die Daten zwischen CPU und den Speichern übermittelt werden. Dafür ist der Datenbus zuständig. Das ist ein Leitungssystem mit meistens acht Leitungen. Auch hier wird pro Leitung ein Bit transportiert, d.h. über den gesamten Datenbus kann gerade 1 Byte auf einmal geschickt werden. Es ist einsichtig, daß diese Datenübertragung um so schneller geht, je mehr Leiterbahnen zur Verfügung stehen. So braucht ein Datenbus mit vier Leitungen für die Übermittlung mindestens doppelt so lange wie sein 8-adriger Kollege. Daher kann man sagen, daß die Anzahl der Leitungen im Datenbus (also seine "Breite") für die Geschwindigkeit der Datenübertragung maßgebend ist. Im Gegensatz zum Adressbus, kann der Datenbus in zwei Richtungen "befahren" werden, "bidirektional" heißt das im Fachchinesisch.

Mit dem inneren Aufbau der CPU und dem internen Datenverarbeitungsmodus möchte ich Sie nicht auch noch belasten. Es könnte sonst vorkommen, daß Sie mir aus dem Buch aussteigen, bevor wir überhaupt ein Wort programmieren konnten.

Zwei Dinge muß ich allerdings noch erwähnen, bevor ich Sie aus diesem Kapitel entlasse. Zum einen ist dies der Taktgenerator, zum anderen die Eingabe/Ausgabe-Bausteine.

Daß Ihr Computer ein taktvolles Gerät ist, haben Sie vielleicht bisher noch gar nicht bemerkt. Trotzdem ist so ein Computer ein Ding, das zyklisch arbeitet, d.h. die Operationen dieses Geräts werden immer zu bestimmten Zeitpunkten vollzogen. Diese Zeitmarken müssen genau festgelegt werden. Damit er weiß, was die Stunde geschlagen hat, braucht er einen Taktgeber. Das ist eine elektronische Schaltung mit einem Schwingquarz als Kernstück. Dieser Quarz wird wie eine Stimmgabel zum Schwingen angeregt, und diese Schwingungen benutzt der Computer zur Zeitmessung. Der Taktgenerator bestimmt das Arbeitstempo der CPU, denn die Dauer der Zyklen für die einzelnen Operationen werden durch die Frequenz des Taktgebers festgelegt. Bei vielen Heimcomputern liegt sie zwischen 1 - 4 Megahertz (das sind 1 - 4 Millionen Schwingungen pro Sekunde). So hat z.B. der SHARP MZ 700 eine Taktfrequenz von 3,5 MHz.

Daten speichern und rechnen im Takt - alles schön und gut, aber was hülfe es dem Programmierer, wenn sein Computer die schönsten Berechnungen anstellte, aber das Ergebnis nicht mitteilen könnte? Noch schlimmer: Zu solchen Berechnungen könnte es erst gar nicht kommen, wenn der Bediener keine Möglichkeit hätte, seiner Maschine kundzutun,

welche Daten sie verarbeiten soll. Für den notwendigen Kontakt zwischen Computer und Peripherie, also auch zu Ihnen, sorgen die Ein- und Ausgabebausteine. Wie Sie anhand von Abb. 1 erkennen können, sind auch sie mit dem Daten-, Adress- und Steuerbus verbunden. Der Microprozessor kann sie also auch ansprechen wie einen Speicher. Es kommt aber ein weiterer Anschluß hinzu: Sogenannte Ein/Ausgabe-Kanäle, die den Computer über die Eingabe/Ausgabe-Einheit mit den Peripheriegeräten (Drucker, Bildschirm, Tastatur usw.) verbinden. An dieser Stelle gilt es, eine wichtige Unterscheidung zu machen: Es gibt nämlich E/A-Einheiten und -kanäle mit 8-Bit paralleler Datenübertragung und solche mit seriellem Datenfluß. Den parallelen Modus kennen Sie schon vom Datenbus her. Das geht schön schnell; wie auf einer achtspurigen Autobahn rasen 8 Bits nebeneinander her. Eine serielle Datenübertragung hingegen ist viel langsamer, da nur eine einspurige Straße zur Verfügung steht, d.h. hier kann nur ein Bit nach dem andern passieren. Weil aber der Datenstrom im Computer sonst immer 8-Bit parallel ist, kommt es zwangsläufig zu einem Stau, wenn Daten vom Datenbus auf einen seriellen Ausgabekanal stoßen. Um diesen Stau sicher abfangen zu können, bedarf es eines Zwischenspeichers, des sog. Puffers. Dort warten die Bits, bis sie im Gänsemarsch entweichen können. Kommen umgekehrt Bits vom seriellen Kanal in den Computer, so werden sie im Puffer gesammelt, bis acht Bits beisammen sind. Erst dann geht die Post auf dem Datenbus ab.

Ob Daten von der Peripherie in den Computer eingegeben oder auf die Peripherie ausgegeben werden sollen, das teilt der Microprozessor den E/A-Bausteinen über den Steuerbus mit.

Da nicht nur ein, sondern gleich mehrere E/A-Kanäle zur Verfügung stehen, hat die CPU die Qual der Wahl, über welchen Kanal sie senden oder empfangen will. Der auserwählte Kanal wird dabei über den Adressbus angesprochen.

2. Kapitel

DER RECHNER SUCHT KONTAKT
- Schnittstellen und Peripherie -

Wenn wir in Gedanken einen Datenstrom aus dem Computer hinausverfolgten, würden wir, nachdem wir die E/A-Einheit passiert haben, auf das Interface treffen. Was, um Himmels Willen, ist das? Schauen wir uns zuerst in Ruhe die deutsche Übersetzung an: "Schnittstelle". Diese Bezeichnung ist irreführend, weil mit einem Interface nicht etwas geschnitten, sondern im Gegenteil etwas verbunden werden soll. Die Daten laufen nämlich von den Peripheriegeräten über das Interface zum Computer und zurück.

Wie bei den E/A-Bausteinen gibt es Schnittstellen mit serieller Datenübertragung (der Standard ist hier die amerikanische RS 232 C- bzw. die fast identische europäische V 24-Schnittstelle) und solche mit parallelem Datentransport (die berühmteste unter ihnen hat den Namen Centronics).

Ein Interface leistet aber mehr, als nur Daten zu übertragen, dazu hätte schließlich ein Kabel gereicht. Es hat vielmehr die Aufgabe, den Computer und die Peripherie elektronisch anzupassen, so daß sie sich untereinander verständigen können. Zum Beispiel teilt der Computer über das Interface der Peripherie mit, daß er Daten senden will. Dies geschieht durch ein sog. Start-Bit. Dann folgen die Bits des Datenworts und ein Stop-Bit kommt hinterdrein, wenn die Übertragung des Zeichens zu Ende ist. Das langsame Peripheriegerät kann seinerseits mit Hilfe der Schnittstelle den schnellen Computer auffordern, zu warten, wenn es mit seinem Tempo nicht mehr mithält. Auch das Interface besitzt einen Puffer, in dem die Daten aus dem Computer zwischengespeichert werden, bis sie das Peripheriegerät (z.B. der Drucker) abgearbeitet hat.

Jetzt wird es endlich Zeit, daß ich Ihnen erkläre, worum es sich bei den Peripheriegeräten eigentlich handelt. Wenn Sie das Wort "Peripherie" im Duden nachschlagen, so finden Sie dort die Bedeutung "Randgebiet oder Umkreis einer Großstadt". Peripheriegeräte sind für den Computer etwas ähnliches - es sind die Apparate, die um ihn "herumstehen" wie z.B. externe Speicher, der Drucker, der Monitor oder auch die Tastatur (bei vielen Rechnern ist die Tastatur allerdings im Gehäuse integriert).

Nachdem Sie Ihren Computer eingeschaltet haben, ist sie das erste, was Sie bedienen. Schließlich müssen Sie ja das Gespräch mit Ihrem Rechner beginnen, indem Sie ihm irgendwelche Anweisungen eintippen. Die Tastatur der meisten Heimcomputer sieht fast so aus wie die einer Schreibmaschine. Unterschiede finden sich z.B. im Fehlen der Umlaute und bei der Anordnung der Buchstaben "Z" und "Y". Anstatt der bekannten Folge Q-W-E-R-T-Z in der ersten Buchstabenreihe, finden Sie in der Regel: Q-W-E-R-T-Y, weswegen man auch von einer QWERTY-Tastatur spricht. Die Tasten der Taschencomputer erinnern eher an die eines Taschenrechners, aber auch sie sind nach dem QWERTY-Muster aufgebaut.

Drücken Sie eine dieser Tasten, so wird das ihr zugeordnete Zeichen in einen Binärcode umgewandelt. Für jedes Zeichen, das Sie auf Ihren Tasten finden, ist ein anderer Code reserviert. Nur so können Sie sich Ihrem elektronischen Helfer verständlich machen.

Um kontrollieren zu können, ob er Sie wirklich begriffen hat, brauchen Sie nur auf den Monitor bzw. die Flüssigkristallanzeige (Liquid Crystal Display = LCD) zu blicken. Dort müßte nämlich das von Ihnen gewünschte Zeichen wieder erscheinen. Und wie, bitte schön, gelangen die Zeichen auf die Mattscheibe? Durch Hexerei? Nein! Trotzdem gibt es dafür zwei zauberhafte Verfahren. Das erste heißt Memory Mapped Display. Der Computer ruft dabei aus seinem ROM ein fertiges Punktmuster des von Ihnen gedrückten Zeichens auf und gibt es auf dem Bildschirm oder dem Display aus. Das zweite Verfahren nennt sich Bitmapping. Hierbei wird im RAM ein Bit-Muster erzeugt, das Punkt für Punkt auf dem Bildschirm ausgegeben wird. Ist ein solches Bit "Eins", so erscheint der entsprechende Punkt auf dem Bildschirm hell. Bei einer "Null" bleibt er dunkel. Das zweite Verfahren zeichnet sich durch eine bessere Grafikfähigkeit aus, braucht aber mehr Speicherplatz.

Wenn Sie allerdings Ihr Programm schwarz auf weiß nach Hause tragen wollen, brauchen Sie dafür ein weiteres Ausgabegerät - den Drucker. Der häufigste Typ ist der Matrixdrucker. Wie der Bildschirm formt er die Zeichen aus Punkten, die zu einer Matrix zusammengefaßt sind. Unter einer Matrix vesteht man ein rechteckiges Punkteraster. In Abb. 2 ist am Beispiel des Buchstaben "A" dargestellt, wie in einer 5x7-Matrix ein Zeichen erzeugt wird.

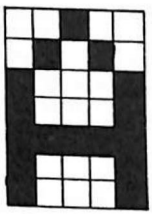


Abb. 2: Punktemuster von "A" in einer 5x7-Matrix

Dieses Punktemuster wird mit Hilfe eng neben- und übereinander liegender Nadeln, die auf ein Farbband schlagen, gebildet.

An viele Taschencomputer können Sie einen Ballpendrucker anschließen. Dieser Typ verfügt über ein revolverartiges Magazin, in dem vier verschiedenfarbige Kugelschreiberminen stecken. Mit einem besonderen BASIC-Befehl können Sie sich die Farbe aussuchen. Ballpendrucker sind, was die Schreibgeschwindigkeit betrifft, recht gemütliche Gesellen, doch können Sie mit Ihnen zeichnen (oder "plotten", wie die Computerfans sich auszudrücken pflegen). Im Reich der Drucker gibt es noch eine ganze Reihe anderer Typen, so z.B. die Thermodrucker, Tintenstrahldrucker und Typenraddrucker.

Als letzte Peripheriegeräte stelle ich Ihnen zwei externe Speicher vor - so heißen sie jedenfalls im Computerjargon. Es sind also Speicher, die sich, im Gegensatz zu RAM und ROM, außerhalb Ihres Computers befinden. Sie sind dazu da, um wichtige Programme abzuspeichern, bevor Sie Ihren Computer ausschalten oder neu programmieren. Häufig finden Sie auch die Bezeichnung Massenspeicher, weil sie damit eine ganze Menge Programme abspeichern können.

Das erste Gerät ist ein alter Bekannter; denn, wenn Sie ein schönes Musikstück aufnehmen wollen, was tun Sie da? Richtig, Sie legen eine Cassette in den Recorder und los geht's. Nichts anderes machen Sie, wenn Sie ein BASIC-Programm konservieren möchten. In Ihrem Computer spielt sich dabei folgendes ab: Die Bits fließen durch eine Schaltung, die aus Nullen und Einsen Töne erzeugt und den parallelen Datenstrom in einen seriellen umwandelt, weil auf einer Cassette brav ein Bit nach dem andern gespeichert wird. Laden Sie Ihr Programm von der Cassette in den Computer, geschieht genau das Umgekehrte.

Der Vorteil des Speichermediums Cassette liegt im geringen Preis, doch ist es sehr langsam. Die Datenübertragungsraten betragen etwa 300 - 2 400 Baud (1 Baud = 1 Bit/sec). Der SHARP MZ 700 z.B. hat eine solche von 1200 Baud. Außerdem ist nur ein serieller Zugang zu den Daten möglich. Sie dürfen also ganz schön spulen, wenn Sie ausgerechnet Ihr wichtigstes Programm an den Schluß einer Cassette gespeichert haben.

Praktischer sind deshalb Disketten. Das sind Scheiben, die aussehen wie eine unterentwickelte Schallplatte mit Schutzhülle. Diese Hülle brauchen die Disketten, denn sie sind sehr empfindlich. Auf ihrer Oberfläche befindet sich eine magnetisierbare Oxydschicht, die ein Schreib/Lesekopf im Diskettenlaufwerk bitmusterartig magnetisiert, wenn ein Programm oder Daten vom Computer auf die Diskette geschrieben werden. Sollen wiederum Daten oder Programme von der Diskette gelesen werden, so werden diese magnetischen Bitmuster durch den Schreib/Lesekopf in einen Strom von Nullen und Einsen zurückverwandelt.

Gegenüber der Cassette hat die Diskette zwei wesentliche Vorteile: Erstens geht die Datenübertragung viel, viel schneller - 250 000 Baud bis 500 000 Baud sind gängige Werte. Zweitens ist ein beliebiger Zugriff auf Daten und Programme möglich (statt des seriellen Zugriffs auf der Cassette), d.h. die Reihenfolge, in der Sie Ihre Programmme abgespeichert haben, ist unbedeutend.

Übrigens: Die Diskette wird, weil sie sehr weich und biegsam ist, auch "Floppy Disk" (zu deutsch: "Schlappe Scheibe") genannt.

3. Kapitel

HARTE SCHALE, WEICHER KERN
- Von der Hardware zur Software -

Der Begriff "Software" ist schon schwerer zu erklären als "Hardware". Software ist nämlich nichts Greifbares. Und dennoch ist sie da. Hier tut wieder ein Vergleich Computer - Mensch not. Beim Menschen würde man als Software seine Gedanken, Gefühle, Erinnerungen, die Erziehung und Prägung bezeichnen. Es gibt also beim Menschen verschiedene Arten von "Software" und genauso ist es beim Computer. Man kann für ihn geradezu eine Softwarehierarchie aufstellen, von maschinennaher Software, die den engsten Kontakt zum Rechner hat, bis zur anwendernahen Software, mit der Sie am engsten in Berührung kommen.

Das erste Glied dieser Kette ist das Betriebssystem, auch Systemsoftware genannt. Der Sitz dieses Betriebssystems ist entweder im ROM oder man lädt es von einer Diskette in das RAM. Und wozu ist es gut? Wieder hilft uns ein Vergleich mit dem Menschen weiter. Wir alle haben uns doch gewisse Routinefähigkeiten erworben, die, egal vor welche Probleme wir gestellt werden, immer die gleichen bleiben. Wir führen sie aus, ohne großartig darüber nachdenken zu müssen. Zum Beispiel gehen Sie ganz automatisch, gleichgültig, ob Sie zu einem Liebesrendevouz oder zum Zahnarzt wollen.

Auch Ihr Rechner hat Routinetätigkeiten zu verrichten, die immer dieselben sind, was Sie auch mit ihm anstellen. Das geht schon mit dem ersten Tastendruck los. Der Computer muß diese Mitteilung von Ihnen entgegennehmen und er muß auch im gleichen Moment auf dem Bildschirm oder Display anzeigen, daß er Sie richtig verstanden hat. Wenn Sie also eine "1" gedrückt haben, sollte auch eine "1" auf der Mattscheibe erscheinen. Das Betriebssystem sorgt für den reibungslosen Ablauf dieses Vorgangs.

Eine andere Aufgabe der Systemsoftware ist die Steuerung des Diskettenlaufwerks. So können Sie z.B. mit ihrer Hilfe ein Programm von der Diskette in den Computer laden und starten.

Eines der bekanntesten Betriebssysteme ist CP/M (Controlprogram/Microcomputer) der Firma Digital Research. Dieses System läuft auch auf dem SHARP MZ 700. Dabei ist es keine Selbstverständlichkeit, daß ein bestimmtes Betriebssystem für einen Computer tauglich ist, da jeder Microprozessor "sein" Betriebssystem fordert. Weil der MZ 700 mit einem Z80 A-Prozessor ausgerüstet ist und CP/M für diesen Prozessortyp entwickelt wurde, geht in diesem Fall alles klar.

Die nächste Softwarestufe könnte jetzt die Programmiersprache sein, wenn es da nicht ein kleines Problem gäbe: Der Computer versteht nämlich keine Programmiersprachen. Ja, Sie haben richtig gelesen. Wie bringt er es dann aber fertig, daß er trotzdem das erledigt, was Sie Ihm z.B. in BASIC programmieren? Gegenfrage: Was würden Sie tun, wenn Ihnen jemand Anweisungen auf Kisuaheli gäbe? Sie würden sich das übersetzen lassen? Eine brillante Idee - genau das geschieht im Computer auch!

Die nächste Sprosse in der Leiter der Softwarehierarchie sind also die Anweisungen, die den Computer befähigen, Ihre Programme in einen für ihn verständlichen Maschinencode zu übersetzen. Zwei Möglichkeiten stehen ihm zur Verfügung: der Interpreter und der Compiler.

Zuerst zum Interpreter: Das ist ein entweder im ROM gespeichertes oder über Cassette in den RAM ladbares Programm, das die in einer Programmiersprache geschriebene Software in Maschinensprache überführt. Der Interpreter übersetzt Wort für Wort, Befehl für Befehl und führt jeden übersetzten Befehl sogleich aus (falls Sie beim Programmieren keinen Fehler gemacht haben, sonst begnügt er sich mit einer Fehlermeldung). Das ist lästig und zeitraubend bei Schleifen und Unterprogrammen, also Programmteilen, die sich mehrmals wiederholen (keine Bange, diese Begriffe werden Ihnen noch erklärt). Bei jedem Durchlauf einer Schleife oder eines Unterprogramms muß nämlich von neuem übersetzt werden.

Ganz anders der Compiler. Der überträgt das ganze Programm in einem Aufwasch und erst danach wird es ausgeführt. Eine Schleife wird also nur einmal übersetzt. Das spart Zeit, und daher sind "compilierte" Programme schneller als "interpretierte". Da aber ein Interpreter die billigere Lösung ist, werden Sie ihm in Heim- und Taschencomputern fast ausschließlich begegnen.

So, jetzt können wir getrost die nächste Stufe der Softwareleiter erklimmen und gelangen zu den Programmiersprachen. Ein Vergleich von Computer und Mensch fällt hier nicht schwer, weil wir Menschen ebenfalls in allen möglichen Sprachen reden. Auch für Computer wurden verschiedene Sprachen entwickelt. Da gibt es "niedere" Sprachen, die computernah sind, bei denen also der Computer wenig Mühe hat, sie in den für ihn brauchbaren Maschinencode umzuwandeln. Dafür ist es für den Menschen recht schwierig, in einer solchen Sprache zu programmieren. Ein bekanntes Beispiel ist die Assemblersprache.

Auf der anderen Seite gibt es "hohe" Programmiersprachen wie z.B. Fortran (eine naturwissenschaftlich-technisch orientierte Sprache), Cobol (für kommerzielle Anwendungen), Pascal, Forth oder Basic. Diese Sprachen sind anwenderfreundlich; dem Menschen bereitet es also verhältnismäßig wenig Schwierigkeiten, diese Sprachen zu erlernen und zu verstehen, dafür hat aber der Computer seine liebe Übersetzungsnot damit.

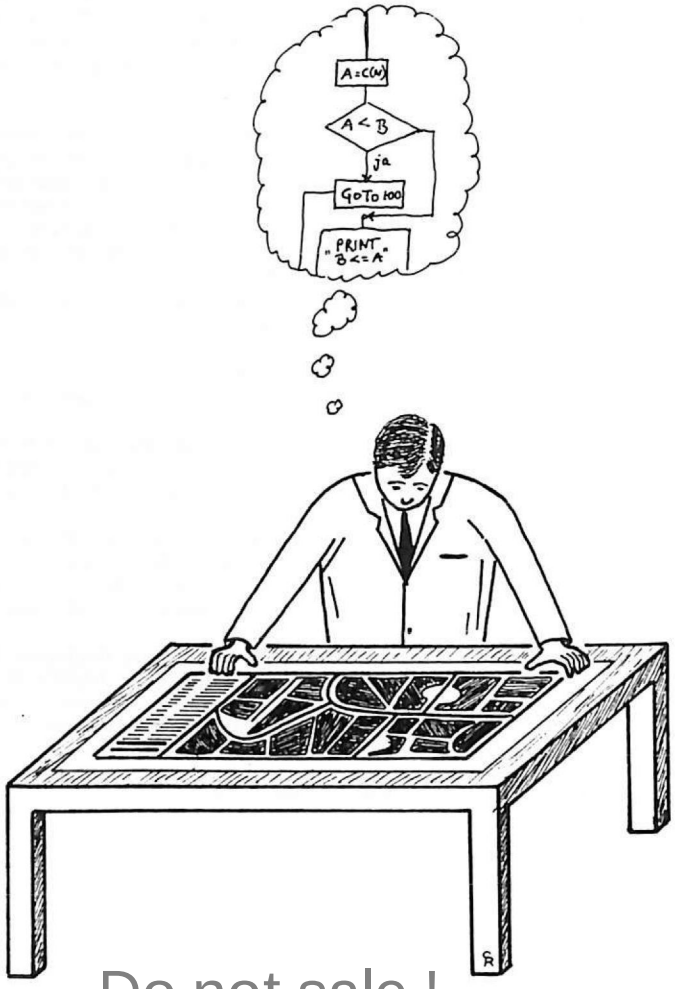
Für BASIC intressieren wir uns hier besonders, ist das doch die am meisten verbreitete Sprache für Heim- und Taschencomputer. Der Name BASIC ist die Abkürzung für **B**eginners **A**ll purpose **S**ympolic **I**nstruction **C**ode - eine Sprache für Anfänger also, was schon mal ganz positiv ist.

Nun dürfen Sie sich nicht der trügerischen Hoffnung hingeben, es gäbe "das BASIC". Im Gegenteil! Soviele Computermarken, soviele BASIC-Dialekte gibt es auch. Um die Verwirrung komplett zu machen, haben zum Teil auch Rechner ein und desselben Herstellers verschiedene BASIC-Versionen. Dieses Buch ist aber so geschrieben, daß Sie wenigstens alle SHARP-Computer damit füttern können. Daneben gibt es doch tatsächlich auch Befehle, die auf allen Computerfabrikaten laufen.

Nach den Programmiersprachen erreichen wir den Gipfel der Software-

pyramide: das Programm. Das ist eine Folge von Anweisungen an den Computer, die in einer beliebigen Programmiersprache geschrieben sind. Hat der Computer das Programm geschluckt, weiß er, was er wie zu tun hat. Für ihn ist dies das gleiche, als ob Ihnen frühmorgens der Chef über den Weg läuft. Nach dieser Begegnung wissen Sie auch, was Sie wie zu tun haben.

Man kennt so viele Programme wie es Anwendungsmöglichkeiten für Computer gibt: Textverarbeitungsprogramme, naturwissenschaftlich-technische Programme, Branchenlösungen (von A wie Apotheker bis Z wie Zahnarzt), Programme zur Steuerung von Maschinen, zur Auswertung von Labordaten usw., usw.. Das ist die Art von Software, mit der ein reger Handel betrieben wird. Und dann gibt es noch Anfängerprogramme, die für den interessierten Laien den Einstieg auf dem Weg zum genialen Programmierer bedeuten.



4. Kapitel

AUSSEN ZWERG UND INNEN ADAM RIESE
- Der Mikrocomputer als Taschenrechner -

Bevor wir loslegen, eine bescheidene Frage: Haben Sie die Bedienungsanleitung Ihres Computers gründlich durchgelesen? Nein?! Dann tun Sie es bitte jetzt! Er ist Ihnen für jede unterlassene Fehlbedienung dankbar. Bitte machen Sie sich dabei gleich mit der Tastatur Ihres Rechners vertraut, da ich in diesem Buch nicht für jeden Computer im Detail darauf eingehen kann.

So, jetzt schalten Sie Ihren Rechner ein. Das erste, was Sie entdekken, ist der CURSOR. Das kann ein Pfeil, ein Rechteck oder einfach ein Strich sein. In jedem Fall befindet er sich zu Anfang im linken oberen Eck des Bildschirms oder Displays. Sobald Sie mit Ihren Eingaben beginnen, wandert der Cursor mit und zeigt Ihnen an, an welche Stelle beim nächsten Tastendruck das gewünschte Zeichen gesetzt wird.

Zu Beginn wollen wir uns an die Fähigkeiten Ihres Computers herantasten, indem wir ihn wie einen Taschenrechner gebrauchen. Bei den kleinen Pocket-Computern (im folgenden PCs genannt) ist dies möglich, ohne auch nur ein Wort BASIC zu verstehen. Probieren wir's doch gleich aus und addieren die Zahlen 4 und 3. In den Taschenrechner würden Sie einfach eingeben:

```
4 + 3 =
```

und als Ergebnis erschiene:

```
7
```

Das machen Sie mal mit Ihrem Computer! Sie könnten bis zur nächsten Eiszeit warten - es bleibt bei:

```
4 + 3 =
```

Solange will ich Sie nicht hängen lassen. Versuchen Sie es doch mal der ENTER-Taste (bzw. der CR-Taste) anstelle von "=":

```
4 + 3 ENTER
```

Mit einem PC haben Sie schon Glück. Es erscheint die 7 auf der Anzeige. Nicht so bei den großen Computern, denn ohne den zugehörigen BASIC-Befehl bleiben sie sehr zugeknöpfte Gesellen. Das ändert sich, wenn Sie folgende Eingabe machen:

```
PRINT 4 + 3 ENTER
```

Der BASIC-Befehl P R I N T bedeutet soviel wie "Gebe aus". Sie geben also im Klartext die Anweisung: "Addiere 4 und 3 und gib das Ergebnis auf dem Bildschirm/Display aus." Diese Anweisung akzeptieren die PCs

genauso. Bei einigen Taschencomputern (PC 1245, PC 1401, PC 1421) kann der PRINT-Befehl durch Druck auf eine dafür vorgesehene Taste eingegeben werden. Bitte brechen Sie nicht in Panik aus, wenn Sie eine solche Taste nicht auf Ihrem Rechner finden, sondern basteln Sie sich "ihr PRINT" aus den Buchstaben P, R, I, N, T zusammen. Das gleiche gilt auch für alle anderen BASIC-Wörter, für die keine eigene Taste vorgesehen ist.

Sicher ist Ihnen schon aufgefallen: Alle Anweisungen oder auch Daten werden mit ENTER verabschiedet. Wenn Sie sich die deutsche Übersetzung von "enter" anschauen, wird Ihnen auch klar, warum. Sie lautet: "eintreten in". Erst wenn Sie Ihrem Rechner diesen Befehl erteilen, geben Sie die Daten oder Anweisungen, die auf der Anzeige stehen, auch ein. Da dieser Befehl so häufig vorkommt, besitzen alle Computer eine spezielle ENTER-Taste. Lassen Sie sich nicht verwirren, wenn bei Ihrem Computer CR oder RETURN auf ihr steht: Das bedeutet alles dasselbe. In den nächsten Beispielen werden wir Sie noch daran erinnern, die ENTER-Taste zu bedienen, später denken Sie bitte selbst daran. Mit den beiden waagrechten Linien oberhalb und unter dem Wort ENTER wollen wir andeuten, daß es sich um eine Taste handelt. Diese Markierung soll auch für zukünftige Beispiele gelten.

Bevor wir mit dem nächsten Beispiel fortfahren, machen wir den Bildschirm sauber. Dies geschieht entweder durch Druck auf die CL-Taste oder in dem Sie den BASIC-Befehl CLS (**CL**ear **S**creen) über die Tastatur eingeben. Sollte sich danach nichts rühren, liegt es vermutlich daran, daß Sie ENTER vergessen haben. Der Cursor ist nach diesem Reinigungsakt wieder in seine Ausgangsposition zurückgekehrt.

Geben Sie jetzt

```
PRINT 4 - 3  ENTER
```

ein. Die Ausgabe ist, wie nicht anders zu erwarten:

```
1
```

Subtrahieren kann Ihr Rechner also auch. Lassen Sie Ihren Computer jetzt multiplizieren und tippen Sie:

```
PRINT  4 * 3  ENTER

12
```

antwortet er Ihnen. Wie Sie oben sehen können, ist aus dem Malpunkt ein Stern (*) geworden. Das gilt für alle Multiplikationen auf allen Computern. Die Division wird durch einen Schrägstrich (/) dargestellt:

```
PRINT  4 / 3  ENTER

1.333333333
```

Sie können an diesem Beispiel sehen, wie viele Stellen hinter dem Komma Ihr Computer berücksichtigt. Apropos, Komma: Statt des uns bei Dezimal-

zahlen vertrauten Kommas, setzt der Computer einen Dezimalpunkt. Bitte tun Sie das bei Ihren Eingaben ebenso. Bei Divisionen müssen Sie auf folgendes noch achtgeben: Teilen Sie nie durch Null! Ihr Programm wird sonst durch eine Fehlermeldung abgebrochen.

Potenzieren tun Sie so:

```
PRINT  2 ^ 3

8
```

Sie haben doch nicht etwa die ENTER-Taste vergessen? 2 ^ 3 lesen Sie dabei als "2 hoch 3", also die dritte Potenz von zwei. Die Form des nach oben zeigenden Pfeils kann von Computertyp zu Computertyp verschieden sein.

Ein Beispiel für Klammerrechnung :

```
PRINT 3 * (8 + 12)
```

Mit dem Resultat:

```
60
```

Sie wollen die Wurzel aus einer beliebigen Zahl wissen? Kein Problem. Dafür kennt BASIC die mathematische Funktion SQR. Das ist die Abkürzung für **SQ**uare **R**oot, auf deutsch: Quadrat Wurzel.

```
PRINT SQR(144)
```

Zieht Ihnen die Wurzel aus 144, folglich muß

```
12
```

herauskommen. Bei einigen PCs können Sie ganz komfortabel eine √-Taste drücken, anstatt SQR einzutippen. Auch bezüglich des Wurzelziehens folgt der Computer streng den Regeln der Mathematik. Das bedeutet, daß die Wurzel aus einer negativen Zahl nicht erlaubt ist. Handeln Sie diesem Gesetz zuwider, werden Sie erbarmungslos mit einer Fehlermeldung bestraft.

Selbstverständlich können Sie auch mehrere dieser mathematischen Operationen mit einem PRINT-Befehl ausführen, Sie müssen dabei aber beachten, daß es unter dieses Operationen eine Rangfolge gibt, die so aussieht:

höchster Rang ↓

1. Klammerausdrücke berechnet der Computer immer zuerst. Bei ineinander verschachtelten Klammern wird mit der innersten begonnen. Z.B. wird der Ausdruck
   `3*(4*(8+2))`
   so abgearbeitet:
   `3*(4*(8+2)) = 3*(4*10) = 3*40 = 120`

2. Danach wird der Wert der Funktionen ermittelt. Der Ausdruck

SQR(16)*2

wird so errechnet:

SQR(16)*2 = (SQR(16))*2 = 4*2 = 8

3. Ausführung der Exponentiation. Beispiel:
   4∧2/2 = 16/2 = 8

4. Vorzeichen + und -.

5. Jetzt sind die Multiplikationen und Divisionen an der Reihe (* und /). Untereinander sind sie gleichberechtigt.

6. Addition und Subtraktion (+ und -). Auch sie sind einander gleichgestellt.

Man kann 5. und 6. zu einer Merkregel zusammenfassen: "Punktrechnung vor Strichrechnung".

7. Die Vergleichsoperatoren <,>,<=,>=,<>, =.
   Das Hühnchen mit den Vergleichsoperatoren rupfen wir, wenn die entsprechenden BASIC-Programme drankommen.

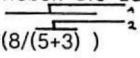
niedrigster Rang

8. Die letzten in dieser Hitparade sind die logischen Operatoren AND, OR, NOT. Auch von ihnen werden Sie noch hören.

Noch ein Hinweis: Haben Sie in einer Formel mehrere gleichrangige Operationen nebeneinanderstehen, so werden sie von links nach rechts abgearbeitet, so wie Sie die Zeitung lesen. Beispiel:

36/3*2 = 12*2 = 24

Und eine Zusatzbemerkung darf nicht fehlen. Haben Sie eine Klammer geöffnet, müssen Sie diese auch wieder schließen. In verschachtelten Klammerausdrücken wird die zuerst geöffnete Klammer zuletzt geschlossen, die zuletzt geöffnete schließen Sie zuerst. Ein Beispiel:



Verdeutlichen wir uns obige Regeln einmal an einem komplizierten Ausdruck:

4+3*(8+3)-2*4∧(2+1)+SQR(9)*12/2

Nach den Rangfolgeregeln wird dieser Bandwurm so abgefertigt:

1. Zuerst alle Klammerausdrücke, also:
   4+3*11-2*4∧3+SQR(9)*12/2
2. Dann alle Funktionsoperatoren, hier ist also SQR(9) = 3 am dransten:
   4+3*11-2*4∧3+3*12/2
3. Jetzt werden alle Exponentialausdrücke berechnet, hier also 4∧3 = 64:
   4+3*11-2*64+3*12/2
4. Alle Multiplikationen und Divisionen werden jetzt ausgeführt und zwar von links nach rechts:
   4+33-128+36/2 = 4+33-128+18
5. Es bleiben nur noch die Additionen und Subtraktionen übrig:
   4+33-128+18 = -73 Geschafft!

Geben Sie zur Probe diese Formel in den Computer ein:

```
PRINT 4+3*(8+3)-2*4↑(2+1)+SQR(9)*12/2
```

Dabei werden Sie feststellen (zumindest wenn Sie einen PC-1500 A besitzen), daß Sie die 26 Zeichen, die die Anzeige aufnehmen kann, überschreiten. Dadurch wandern die Zeichen von rechts nach links aus dem Display hinaus. Das hat für den Rechenvorgang aber keine Bedeutung, denn der PC-1500 A kann bis zu 80 Zeichen pro Zeile verarbeiten. Solange Sie nicht die ENTER-Taste gedrückt haben, können Sie Ihre Eingabe kontrollieren, indem Sie mit der Taste "Cursor nach links" den Text zurückrollen lassen.

Sie werden sich vielleicht schon wundern, warum ich bisher meistens ganze Zahlen für die Rechenbeispiele verwendet habe. Das hat seinen besonderen Grund: Rechnungen mit ganzen Zahlen können Sie leicht im Kopf nachvollziehen und so prüfen, ob auch das rauskommt, was Sie sich vorgestellt haben. Dabei bekommen Sie ein Gefühl dafür, wie Ihr Rechner arbeitet, also ob z.B. die Operatorenrangfolge, die ich Ihnen angegeben habe, tatsächlich so stimmt. Das Nachvollziehen ist besonders wichtig bei komplizierten mathematischen Formeln, die Sie so auf ihre Richtigkeit testen können. Ich gebe Ihnen daher die warme Empfehlung: Testen Sie mathematische Ausdrücke und Programme immer anhand einfacher Beispiele, die Sie noch leicht überblicken können. Selbstverständlich rechnet der elektronische Mathematiker auch mit Kommazahlen, doch müssen Sie es schon gewaltig auf der Pfanne haben, um ihm da noch folgen zu können.

Damit Sie sehen, wie flink Ihr Computer selbst mit komplizierten Zahlen fertig wird, servieren Sie ihm dies:

```
PRINT 4.98802*0.2↑(3.84+2.39)-(8.11-5.773)/1.615
```

Es sollte

```
-1.446838356
```

dabei herauskommen. Man nennt diese Art von Zahlenausgabe "Gleit"- oder "Fließkommadarstellung", auch wenn Sie einen Dezimalpunkt statt eines Kommas verwenden. Daneben gibt es die "Exponentialdarstellung" einer Zahl, auch "wissenschaftliches Format" genannt, z.B.

```
4.3469132E-02
```

Das E steht für "10 hoch", es ist also

$$4.3469132E\text{-}02 = 4.3469132 * 10^{-2} = 0.043469132$$

Sie können somit diese Zahl entweder in Gleitkomma- oder Exponentialschreibweise in den Rechner eingeben. Ihr Computer gibt auch im wissenschaftlichen Format aus, wenn ihm die Stellen des Ergebnisses einer Rechenaufgabe über den Kopf wachsen. Füttern Sie ihn deshalb mal mit dieser Aufgabe:

```
PRINT 9.335627/190
```

Das Resultat ist:

4.913487895E-02

Sofern Sie über einen Drucker verfügen und das Ergebnis schwarz auf weiß haben wollen, steht dem nichts im Wege. Tippen Sie einfach

LPRINT 9.335627/190

ein und schon steht obiges Ergebnis auf dem Papier. LPRINT bewerkstelligt somit das gleiche wie der PRINT-Befehl, nur werden die Ergebnisse auf den Drucker statt der Anzeige ausgegeben.

Zum Schluß des Kapitels gebe ich Ihnen noch diese Nuß zu knacken:

Aufgabe 4.1

Berechnen Sie folgenden Mathematischen Ausdruck nach der Rangordnung für mathematische Operatoren und zwar bitte, bevor Sie diese Formel in den Computer eingeben oder sich die Musterlösung im Anhang anschauen:

4+2*(2+3)/\2-SQR(4)*8/4*3

Aufgabe 4.2

Lassen Sie Ihren Rechner die folgenden Formeln ausrechnen. Bedenken Sie dabei, daß Sie sie vorher in eine Schreibweise umwandeln müssen, die er versteht.

a) $\frac{1}{5} + \frac{1}{4} - \frac{1}{20}$

b) $3 \times 1\frac{1}{4}$

c) $\sqrt{25+9} + 3^{(1+2)}$

Ermitteln Sie die Ergebnisse aus a) - c) zur Kontrolle immer erst einmal von Hand.

5. Kapitel

WOVON ER BESONDERS SCHWÄRMT, WENN ES WIEDER AUFGEWÄRMT

- Das erste Programm -

Als Lockerungsübung ist der Ge- (oder Miß-)brauch des Computers als Taschenrechner ganz nett - aber, seien Sie mal ehrlich, auf die Dauer wäre das doch genauso hirnrissig, als ob Sie mit einer Harley-Davidson ständig in der Fußgängerzone rumkurven würden. Das will besagen: Sie schöpfen die Möglichkeiten Ihres Computers bei weitem nicht aus, wenn Sie ihn nur als Taschenrechner ver(sch)wenden.

Stellen Sie sich einmal vor, Sie müßten hundertmal die Mehrwertsteuer von irgendwelchen Artikeln berechnen. Mit einem Taschenrechner gehen Sie da so vor:

1. Eingabe des Preises der Ware (ohne MWSt.)
2. Berechnung der Mehrwertsteuer (von % in DM)
3. Addition der MWSt. zum Warenpreis
4. Sie schreiben sich den Preis incl. MWSt. auf

Ich garantiere Ihnen: Irgendwann werden Sie dabei zum Hirsch. Mit dem Computer (diesmal wirklich als Computer verwendet) tun Sie nur dies:

Eingabe des Warenpreises ohne MWSt. sowie des Prozentsatzes

Der Computer (nicht Sie) geht dann so vor:

1. Berechnung der Mehrwertsteuer (von % in DM)
2. Addition der MWSt. zum Preis
3. Ausgabe des Preises incl. MWSt. auf dem Drucker (oder der Anzeige)

Die Wahrscheinlichkeit, daß Sie zum Hirsch werden, hat sich erheblich vermindert.

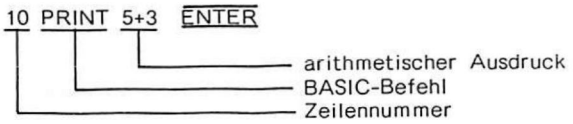
Jedoch: Ihr Computer braucht ein Programm, um diese Aufgabe lösen zu können. Und ein Programm, das ist eine Folge von Anweisungen, ... Ja, Sie haben es schon im 3. Kapitel "Harte Schale, weicher Kern" gelesen.

Wir schreiben unsere Programme in BASIC und das bedeutet: die Folge von Anweisungen wird in Form von Befehlszeilen, auch Statements genannt, aufgeschrieben. Damit der Computer weiß, in welcher Reihenfolge er diese Statements zu bearbeiten hat, werden sie mit einer Zeilennummer versehen. Der Programmablauf beginnt bei der niedrigsten und endet bei der höchsten Nummer, es sei denn, Sie verändern diese Reihenfolge durch Programmsprünge (darüber erfahren Sie im 7. Kapitel mehr).

Um nun Ihrer Vorstellung auf die Sprünge zu helfen, wählen wir wieder ein ganz, ganz einfaches Beispiel. Sie wollen zwei Zahlen addieren und das Ergebnis auf dem Display ausgeben.

Halt, beinahe hätte ich es vergessen: Wenn Sie Programmieren wollen, dann müssen Sie darauf achten, daß sich Ihr Rechner im PRO-Mode oder im BASIC-Mode befindet, je nach PC-Typ. Für den Fall, daß Sie mit einem Computer der MZ-700 bzw. MZ-800 Serie arbeiten, brauchen

Sie jetzt gar nichts zu tun. Ihr Rechner merkt es an der Art der Eingabe, ob Sie ihn programmieren wollen. Jetzt geben Sie mal folgendes in die Maschine:



Sie sehen: Zuerst kommt die Zeilen- oder Statementnummer, sie kann bei 1 anfangen, muß es aber nicht. Ich habe in diesem Beispiel die 10 gewählt. Die Zeilennummer darf aber nicht kleiner als 1 sein (0, -1, -2 usw. sind demnach verboten) und es muß eine ganze Zahl sein. Wenn Sie z.B. 4.9 als Zeilennummer wählen, wird der Computer Sie für einen Witzbold halten und, allerdings gar nicht zum Scherz, eine Fehlermeldung bringen. Außerdem darf die Zeilennummer eine Höchstgrenze, die von Computertyp zu Computertyp variiert, nicht überschreiten. Beim PC-1500 A beträgt sie beispielsweise 65 279. Nach der Nummer folgt das Statement selbst, hier ein PRINT-Befehl mit einem arithmetischen Ausdruck. Obwohl es Ihnen z.B. der PC-1500 A nicht verübelt, wenn Sie zwischen der Zeilennummer und dem BASIC-Befehl kein Leerzeichen setzen, empfielt sich dies trotzdem, denn nicht alle Computer sind so gutmütig. Aus dem gleichen Grund gewöhnen Sie es sich bitte an, auch zwischen einem BASIC-Befehl und einem arithmetischen Ausdruck ein Zeichen frei zu lassen.

Und noch etwas: Eine Programmzeile darf nicht beliebig lang sein. Der PC-1500 A erlaubt Ihnen 80 Zeichen/Zeile.

Ich erlaube mir jetzt eine Zwischenfrage: Was ist eigentlich geschehen, seitdem Sie

10 PRINT 5+3 ENTER

eingegeben haben? Nichts? Doch, etwas hat sich verändert. Bei den PCs hat sich zwischen der Zeilennummer und dem PRINT ein Doppelpunkt eingefunden, so daß die Zeile so aussieht:

10:PRINT 5+3

Bei den großen Computern blieb alles beim alten, lediglich der Cursor ist auf den Anfang der nächsten Zeile gesprungen.

Was jetzt? Zunächst schalten Sie Ihren Taschencomputer vom PRO-Mode in den RUN-Mode (dasselbe gilt für Rechner im BASIC-Mode) um, bei den MZs drehen Sie so lange Däumchen. Immer noch nichts. Aber jetzt kommt das "Sesam-öffne-Dich-Kommando". Tippen Sie nur diese drei Buchstaben ein:

R U N

Verfügt Ihr kleiner Liebling über eine RUN-Taste, brauchen Sie sogar nur diese zu drücken. In beiden Fällen segnen Sie aber das RUN-Kommando noch mit einem Druck auf die obligate ENTER-Taste ab. Und siehe da,

eine

8

erscheint auf der Anzeige. Also merken Sie sich: Programme werden mit

R U N ENTER

gestartet. Und das Wichtigste: Programme können beliebig oft mit RUN wiederholt werden.

Wenn Sie jetzt sagen, es sei blödsinnig gewesen, wegen so einer Miniaufgabe so einen Staatsakt zu machen, stimme ich Ihnen unumwunden zu. Dennoch haben Sie einige wichtige Prinzipien über Programme lernen können.

Machen wir uns jetzt an eine Aufgabe heran, die das Programmieren schon eher lohnt. Erinnern Sie sich noch an die Sache mit der Mehrwertsteuer? Wir bauen uns nun ein Programm, das uns die Mehrwertsteuer zum Warenpreis addiert. Es soll uns zuerst auffordern, den Preis ohne MWSt. in DM und die MWSt. in % einzugeben, dann soll die MWSt. in DM ausgerechnet und zum Preis addiert werden. Das Ergebnis soll auf der Anzeige erscheinen.

Bevor wir mit dem neuen Programm beginnen, machen wir erst die Anzeige sauber. Dem Befehl dafür sind Sie schon kurz begegnet:

CLS

Schalten Sie wieder auf den PRO-Mode um. Das alte Programm brauchen Sie nicht mehr, also nichts wie hinaus damit. In BASIC gibt es für diesen Hinauswurf den Befehl

NEW

Er löscht alle im Speicher vorhandenen Programme und Variablen. Diesen Befehl müssen Sie bei allen Computern mit Hilfe der Buchstabentasten eingeben, d.h. es gibt keine NEW-Taste. Und das aus gutem Grund: Sie könnten sonst aus Versehen an eine solche Taste herankommen und ungewollt Ihr Programm zur Minna machen. Das NEW-Kommando wird übrigens im RUN-Mode nicht akzeptiert.

Nun zu unserem neuen Programm. Was brauchen wir da alles? Um mit beliebigen Zahlen rechnen zu können, sind erst mal numerische Variablen notwendig. In der Mathematik sind das Größen, die beliebige, sich ändernde Werte annehmen können. Für den Computer ist eine Variable eine Größe, für die er einen bestimmten Speicherplatz im RAM zu reservieren hat. Sie dient sozusagen als Platzkarte für eine Zahl, die Sie in diesen RAM-Speicherplatz ablegen wollen. Es ist aber nicht so, daß diese Zahl die Speicherzelle für alle Zeiten gepachtet hat. Der Computer behandelt die Variablenwerte nämlich nach dem K.-o.-Prinzip. Weist er einer Variablen einen neuen Wert zu, kommt dieser in die Speicherstelle, wodurch der alte Wert erbarmungslos "aus dem Ring" fliegt, d.h. er wird gelöscht. Damit der Computer auch immer weiß, welche Variable und somit welche

Speicherzelle gemeint ist, müssen Sie der Variablen einen Namen geben, z.B. X oder A usw.

Für numerische Variablen sind etliche Regeln erdacht worden, einige davon gelten doch tatsächlich für alle Computer und die nenne ich Ihnen zuerst.

Regel 1: Der Name einer numerischen Variable besteht aus einer Folge von Buchstaben und Zahlen. Beispiele: AA, A1, X2, Y usw. Nicht erlaubt sind z.B.: A&", A? und dergleichen.

Regel 2: Der Variablenname muß mit einem Buchstaben anfangen, erlaubt sind demnach: Z1, B2 usw., nicht aber 1B, 24A ...

Regel 3: Der Variablenname darf mit keinem BASIC-Schlüsselwort identisch sein, deshalb sind Namen wie IF, SQR, LN, PI, PRINT oder LET nicht zulässig.

Regel 4: Einer numerischen Variablen darf entweder eine Zahl, ein Funktionswert oder arithmetischer Ausdruck zugewiesen werden. Erlaubt sind z.B. diese Zuweisungen:
A = 3.475 C = 1.45837967E02
Y = SQR(73) Z = (Y-A)/2
Für numerische Variablen Tabu sind jedoch:
F = Donnerstag G = 4 plus 3 mal 7
H = ??? Und ganz bestimmt nicht erlaubt ist:
I = "Lieber ueber Nacht versumpfen, als im Sumpf uebernachten"

Regel 5: Sie dürfen folgende Zuweisung machen: A = A+1 Was in der Mathematik ein Frevel wäre, bedeutet für den Computer lediglich, in den Speicherplatz von A den Wert von A+1 zu setzen. Der alte Wert von A wird dadurch gelöscht (siehe oben)

Diese fünf Regeln gelten, man lese es und staune, also für alle Computer. Das Gesetz, das ich Ihnen jetzt präsentiere, ist für alle Computer im Prinzip gültig, unterscheidet sich aber von Rechner zu Rechner im Detail.

Regel 6: Der Variablenname darf eine bestimmte Höchstzahl von Zeichen nicht überschreiten, wenn Sie der Computer noch voneinander unterscheiden oder nicht mit einer Fehlermeldung abbrechen soll.

Der PC-1500 A kann beispielsweise die verschiedenen Variablen nur auf Grund der beiden ersten Zeichen auseinanderhalten. Sie dürfen zwar mehr als zwei Zeichen verwenden, aber die zusätzlichen Zeichen werden vom Rechner ignoriert. So sind für ihn Variablennamen wie MISTER und MISS bzw. KUCHEN und KUHFLADEN ein und dasselbe. Dennoch kann der Gebrauch von mehr als zwei Zeichen pro Name sinnvoll sein, wenn dieser für den Programmierer erklärende Funktion hat. Die meisten Taschencomputer von SHARP können eins, höchstens zwei Zeichen einer Variablen voneinander unterscheiden, bei Personal-Computern sind es jedoch mehr. Zu guter Letzt noch eine Regel, die für die meisten PCs gilt:

Regel 7: Variablennamen dürfen nur Großbuchstaben enthalten. Erlaubt sind somit A, AA, ... aber nicht a, aa, Aa, ...

Nachdem Sie jetzt wissen, welchen Variablen Sie welche Werte zuweisen können, möchte ich Ihnen nicht verschweigen, wie Sie mit einem Befehl so gut wie alles über den Haufen werfen können. Die Anweisung

```
CLEAR
```

läßt Ihren Rechner alle Variablennamen und deren Werte vergessen. Auch die Variablenfelder, von denen Sie noch später hören werden, sind davon betroffen.

Falls Sie vor lauter Regeln das Programm nicht mehr sehen: Wir sind immer noch dabei, Mehrwertsteuern zu berechnen. Eine Frage muß noch geklärt werden. Wie bekommt man die Variablenwerte in den Computer? Eine Methode, etwas in den Computer "hinein"-zubekommen, ist die Anweisung:

```
INPUT
```

Mit ihr können wir die zwei Variablen P für Preis und M für Mehrwertsteuer so eingeben:

```
10 INPUT P

20 INPUT M
```

Was bewirkt nun die Zeile: 10 INPUT P ? Sie reserviert erstens für die Variable P einen Speicherplatz, zweitens fragt der Computer, welchen Zahlenwert Ihre Durchlaucht in den Speicherplatz zu schreiben wünschen. Er zeigt dies durch ein "?" am linken Rand des Displays (Bildschirms) an und wartet mit der Fortsetzung des Programms so lange, bis Sie einen Wert für P über die Tastatur eingetippt haben. Die von Ihnen eingegebene Zahl, wie sollte es auch anders sein, wird erst nach Druck der ENTER-Taste geschluckt. Mit dem Statement: 20 INPUT M geschieht dasselbe.

Der Rechner hat nun den beiden Variablen P und M die von Ihnen eingegebenen Werte zugewiesen. Jetzt soll der Preis einschließlich MWSt. berechnet und dann ausgegeben werden. Wie machen Sie das? Nun, wenn M die Mehrwertsteuer in % ist, dann ist doch der MWSt.-Anteil des Preises P:

```
P*M/100
```

Diesen Anteil addieren Sie zum Preis P:

```
P+P*M/100
```

Jetzt ziehen Sie P vor die Klammer:

```
P*(1+M/100)
```

So soll auch das Programm den Preis incl. MWSt. berechnen und ausgeben. Auf die Schnelle geht das so:

```
30 PRINT P*(1+M/100)
```

Sie können also mit Variablen genauso rechnen wie oben mit den Zahlen.

Da genügt ein LIST und das ganze Programm ist auf der Mattscheibe:

```
10 INPUT P

20 INPUT M

30 LET I = P*(1+M/100)

40 PRINT I

50 END
```

(Der Doppelpunkt zwischen Zeilennummer und Statement fehlt bei den Großen).

Sie können sich aber auch einzelne Zeilen ausgeben lassen, indem Sie die entsprechende Zeilennummer hinter den LIST-Befehl setzen, etwa so:

```
LIST 20
```

wodurch

```
20:INPUT M
```

auf der Anzeige erscheint. Sollte es allerdings keine Zeile 20 in Ihrem Programm geben, dann listet ein Computer wie der PC-1500 A die nächsthöhere (in diesem Fall 30:LET I = P*(1+M/100)). Und wenn es keine nächsthöhere Zeile gibt? Pech gehabt, dann schauen Sie sich zur Abwechslung eine Fehlermeldung an.

Anmerkung: Sie können eine Zeile auch durch eine Textmarke kennzeichnen und sie durch Aufruf dieser Marke listen lassen. Wenn Sie z.B. Zeile 20 so programmiert haben:

```
20:"MARKE":INPUT M
```

dann wird sie durch das Kommando LIST "MARKE" genauso ausgegeben wie durch LIST 20. Sie dürfen nur nicht vergessen, die Textmarke in Anführungszeichen zu setzen und sie durch einen Doppelpunkt vom restlichen Statement zu trennen. Pro Marke sind maximal 16 Zeichen erlaubt.

Theoretisch haben wir nun das ganze Programm durchgekaut, jetzt wird es Zeit, ihm Beine zu machen. Schalten Sie deshalb in den RUN-Mode um und starten Sie es mit RUN ENTER. Ihr Computer antwortet mit einem Fragezeichen:

```
?
```

Keine Angst, Sie haben nichts falsch gemacht, das erste INPUT fordert Sie lediglich dazu auf, einen Wert für die Variable P (den Preis also) einzugeben. Tippen Sie z.B. 100 ein (sollte sich noch nichts abspielen, dann liegt es daran, daß Sie vergessen haben, die ENTER-Taste zu drükken). Danach erscheint wieder ein Fragezeichen, denn 20:INPUT M erwartet sehnsüchtig einen Wert für die Mehrwertsteuer. Befriedigen Sie es

z.B. mit 14. Das Ergebnis der Rechnung aus Zeile 30 folgt sogleich, es wird durch PRINT I in Zeile 40 ausgegeben:

114

Eines werden Sie sich aber noch fragen: Wieso hat er die Zeilenfolge

```
10:INPUT P

20:INPUT M

30:LET I = P*(1+M/100)

40:PRINT I

50:END
```

und nicht

```
1:INPUT P

2:INPUT M

3:LET I = P*(1+M/100)

4:PRINT I

5:END
```

verwendet? Das hat einen ganz praktischen Grund. Wenn Sie nämlich irgendwo zwischen den Zeilen ein zusätzliches Statement einschieben wollen, ja, wo fügen Sie das dann ein, wenn Sie nach der zweiten Art programmiert haben? Beispielsweise müssen Sie immer noch CLS eintippen, wenn Sie Ihr Programm auf einer sauberen Anzeige neu starten wollen. Auf die Dauer wird Ihnen das auf den Geist gehen. Bei der ersten Programmversion können Sie das Übel bequem abschaffen, indem Sie die Zeile

```
5 CLS
```

an den Anfang stellen. Der Rest des Programms bleibt dadurch unbehelligt. Bei der zweiten Version ist dies nicht möglich. Es hilft nichts, Sie müssen das ganze Programm nochmals schreiben:

```
1:CLS

2:INPUT P

3:INPUT M

4:LET I = P*(1+M/100)

5:PRINT I

6:END
```

Das ist doch ärgerlich, nicht wahr? Deshalb wählen Sie für die Zeilennummern immer einen Zehnerabstand.

Übrigens: Es macht rein gar nichts aus, wenn Sie die Zeile 5 erst nach den Zeilen 10, 20 usw. programmieren. Der Computer ordnet die Zeilen automatisch nach aufsteigenden Zeilennummern und führt auch die Befehle in der richtigen Reihenfolge aus.

Das Mehrwertsteuer-Programm läuft also - schön und gut - aber nun stellen Sie sich einmal vor, jemand anderes als Sie wollte mit diesem Programm arbeiten. Selbst wenn er es zum laufen brächte und wüßte, daß ein Zahlenwert einzugeben ist, sobald ein Fragezeichen auftaucht, er könnte trotzdem das Programm nicht richtig bedienen. Zum Beispiel würde er das erste INPUT für den Preis mit dem Wert 12 beantworten, was noch im Rahmen läge, aber das INPUT für die Mehrwertsteuer, INPUT M, eventuell mit 200 erwidern und damit die Mehrwertsteuer auf 200 % setzen! Für Sie ist eine solche Steuer ein Alptraum - dem Computer ist das so egal wie der Dreck unter Ihrem Fingernagel. Brav errechnet er gemäß der Anweisung 30:LET I = P*(1+M/100) einen Preis von 36 Mark inclusive Mehrwertsteuer aus. Was fehlt, ist eine "Gebrauchsanweisung", die dem Anwender sagt, was Trumpf ist.

Nun kennt BASIC mehrere Möglichkeiten, ein Programm zu kommentieren. Eine davon ist der Kommentar mit PRINT. Dazu brauchen Sie nur alles, was als Kommentar gelten soll, in Anführungszeichen zu setzen, so wie bei einer wörtlichen Rede. Zum Beispiel faßt Ihr Rechner die Zeile

```
100:PRINT "DIES IST NUR EIN KOMMENTAR"
```

als eine solche "wörtliche Rede" auf, d.h. er gibt nur die Zeichen zwischen den Anführungszeichen auf der Anzeige aus und macht sonst nichts mit ihnen. Damit haben wir ein hervorragendes Mittel zur Hand, einem "Reingeschmeckten" den richtigen Gebrauch unseres Edelprogramms mitzuteilen.

Zurück zu unserem Programm: Tippen Sie folgende Zeile in den Rechner:

```
7 PRINT "PREIS IN DM"
```

Der Computer kann jetzt seinen Benutzer schriftlich dazu auffordern, einen Warenpreis einzugeben. Den Zahlenwert für die Mehrwertsteuer erbittet sich der Computer so:

```
15 PRINT "MEHRWERTSTEUER IN %"
```

Nun weiß wenigstens jeder, der sich mit Mehrwertsteuern ein bißchen auskennt, daß nur die Zahlen 7 oder 14 in Frage kommen.

Weil wir auch das Ergebnis (den Preis inclusive MWSt.) nicht kommentarlos ausgeben wollen, frisieren wir Zeile 40 so um:

```
40 PRINT "INCLUSIVPREIS IN DM";I
```

Wie Sie sehen, steht hinter dem Text "INCLUSIVPREIS IN DM" ein Se-

mikolon gefolgt von der Variablen I. Das Semikolon setzen Sie immer, wenn Sie die darauffolgende Ausgabe (hier den Wert von I) an die vorangehende (hier der Text INCLUSIVPREIS IN DM) anschließen wollen. Lassen Sie das Semikolon fort und programmieren Sie z.B. so

```
40 PRINT "INCLUSIVPREIS IN DM"

45 PRINT I
```

dann wird der Text INCLUSIVPREIS IN DM gelöscht, bevor der Wert von I ausgegeben wird. Die Zeile

```
40 PRINT "INCLUSIVPREIS IN DM" I
```

(also ohne Semikolon) bringt Ihnen nichts als eine Fehlermeldung ein.

Nebenbei bemerkt: Wiederum erkennen Sie, welch ein Segen die Zeilennummerierung in Zehnerschritten ist, denn Sie können mühelos die beiden zusätzlichen Zeilen einfügen.

Sie können aber auch mehrere BASIC-Statements in einer Zeile unterbringen, wenn Sie diese durch einen Doppelpunkt trennen. Zeile 7 und Zeile 10 etwa lassen sich so zusammenfassen:

```
10 PRINT "PREIS IN DM" : INPUT P
```

Dies leistet dasselbe wie:

```
7 PRINT "PREIS IN DM"

10 INPUT P
```

Eine Zeile, in der mehrere Statements auftauchen, wird auch Multistatement genannt. Um die Übersicht nicht zu verlieren, sollte der Anfänger Multistatements möglichst vermeiden.

Nun wollen wir aber endlich das Programm starten. Erwartungsgemäß wird die Anzeige gelöscht (wegen 5:CLS) und die Worte

```
PREIS IN DM
```

(ohne Anführungszeichen) erscheinen auf dem Display. Bei den Großen von SHARP ist links unter diesem Text auch ein Fragezeichen zu sehen und Sie können den Preis sofort eingeben. Bei einem kleinen Rechner, vor allem denen mit einzeiligem Display, müssen Sie erst die ENTER-Taste drücken, bevor Sie den Preis eintippen, da das Programm nach einer PRINT-Anweisung so lange ausharrt, bis Sie Ihm mit ENTER wieder die Sporen geben. Jetzt erst taucht das Fragezeichen auf und Sie können den Preis eingeben. Danach wird die Zeile

```
MEHRWERTSTEUER IN %
```

angezeigt. Nach einem erneuten ENTER sehen Sie wieder ein Fragezeichen und Sie sollten Ihrem Rechner auch den MWSt.-Satz nicht vorenthalten.

Füttern Sie ihn wieder mit 14 und Sie erhalten als Resultat brav den Preis incl. 14% MWSt. angezeigt.

Sollten Sie es lästig finden, Ihr Programm nach jedem PRINT durch ENTER wieder anzutreiben, können Sie dafür auch die BASIC-Anweisung

WAIT

verwenden. Diese Instruktion ermöglicht es Ihnen, ein Zeitintervall vorzugeben, nach dessen Ablauf das Programm hinter einer PRINT-Anweisung automatisch wieder in Gang kommt. Das Statement

```
100 WAIT 64
```

bewirkt dies: Alle nach dieser Zeile folgenden PRINT-Anweisungen, werden für die Dauer von einer Sekunde auf dem Display angezeigt, worauf das Programm von selbst wieder fortfährt. Sie sehen also: Die Zahl hinter dem WAIT-Befehl gibt das Zeitintervall in Einheiten von 1/64 sec an. Vergessen Sie es, hinter WAIT eine Zeitangabe zu machen, war Ihre Mühe umsonst, und Sie müssen wieder nach jedem PRINT die ENTER-Taste bedienen. Ganz sinnvoll kann ein leerer WAIT-Befehl sein, wenn Sie während des Programmlaufs diesen PRINT-Modus ausschalten wollen. Ansonsten setzen Sie hinter das WAIT eine Zahl zwischen 0 und 65 535, wobei Sie bei Verwendung von 65 535 ruhig Kaffee trinken gehen können, denn Ihre Maschine wartet geschlagene 17 Minuten.

Angewandt auf unser Programm könnte das so aussehen:

```
5:CLS
6:WAIT 200
7:PRINT "PREIS IN DM"
10:INPUT P
15:PRINT "MEHRWERTSTEUER IN %"
20:INPUT M
30:LET I = P*(1+M/100)
40:PRINT "INCLUSIVPREIS";I
50:END
```

Nach Ausführung der PRINT-Anweisungen in den Zeilen 7, 15 und 40 wartet der Computer jeweils etwa 3 Sekunden, dann läßt er das Programm weiterlaufen.

Ist Ihnen die Sache mit dem WAIT zu aufwendig? Keine Bange, sie läßt sich mit Hilfe der

PAUSE

Anweisung umgehen. Diese funktioniert genauso wie der PRINT-Befehl, mit einer Ausnahme: Nachdem der Computer den PAUSE-Befehl befolgt hat, startet er nach einer gewissen Zeitspanne (beim PC-1500 A sind es ca. 0.85 sec) das Programm automatisch und zwar ohne eine WAIT-Anweisung. Werfen Sie also das WAIT in Zeile 6 wieder aus dem Programm raus. Das tun Sie, indem Sie (im PRO-Mode) eingeben:

6 ENTER

Dadurch löschen Sie Zeile 6. Ersetzen Sie jetzt in den Zeilen 7 und 15 das PRINT durch PAUSE:

```
7 PAUSE "PREIS IN DM"

15 PAUSE "MEHRWERTSTEUER IN %"
```

Lassen Sie Ihr Programm wieder laufen. Sie werden feststellen, daß die Anzeigen

PREIS IN DM

und

MEHRWERTSTEUER IN %

durch die PAUSE-Instruktion recht flott über das Display huschen, denn die Wartezeit von 0.85 Sekunden ist doch recht kurz.

Aber nicht nur mit PRINT und PAUSE, sondern auch mit INPUT können Sie Ihren Senf dazugeben. Genauso wie bei den ersten beiden Anweisungen, müssen Sie den Kommentar wieder in Anführungszeichen setzen und wenn Sie die Variablenwerte gleich dahinter eingeben wollen, dann ist nach dem Kommentar ein Semikolon und der Variablenname erforderlich. Modeln Sie die Zeilen 10 und 20 also so um:

```
10 INPUT "PREIS IN DM ";P

20 INPUT "MEHRWERTSTEUER IN % ";M
```

Zeile 7 und 15 sind überflüssig. Sie können Ihnen eiskalt den Buckel runterrutschen, indem Sie sie mit 7 ENTER und 15 ENTER verabschieden. Das neue Programm hat dann diese Gestalt:

```
5:CLS
10:INPUT "PREIS IN DM ";P
20:INPUT "MEHRWERTSTEUER IN % ";M
30:LET I = P*(1+M/100)
40:PRINT "INCLUSIVPREIS IN DM";I
50:END
```

Übrigens: Die Leerzeichen hinter DM und % sind pure Absicht, weil sonst die Eingaben für Preis und Mehrwertsteuer zu dicht auf den Text folgen würden. Die PRINT-Anweisung in Zeile 40 hat sowas nicht nötig - sie läßt zwischen Kommentar und INCLUSIVPREIS-Ausgabe automatisch ein Zeichen frei (bei positivem Vorzeichen).

Starten Sie jetzt das Programm und Sie werden sehen: Es leistet ähnliches wie vorher die Programme mit PRINT-WAIT bzw. mit PAUSE, nur daß jetzt der Variablenwert direkt hinter dem Kommentar eingetippt werden kann und bei manchen Computern (wie z.B. dem PC-1500 A) kein Fragezeichen mehr erscheint. Dennoch erwartet der Rechner Ihre Zahlen.

Sollten Sie jetzt etwas ärgerlich werden, weil Sie daran denken, wie einfach alles gewesen wäre, wenn ich Ihnen die letzte Programmversion gleich präsentiert hätte, so darf ich Sie daran erinnern: Sie wollten doch BASIC lernen, nicht wahr? Na also! Zum Sharp-BASIC gehören aber Befehle wie WAIT und PAUSE dazu. Damit Sie sich Ihr Mütchen kühlen, gleich noch ein paar Anweisungen hinterher.

Eventuell verspüren Sie jetzt Lust, Ihr Prográmmchen auf Papier zu verewigen. Wenn Sie außer Lust noch einen Drucker haben, läßt sich dies mit dem Befehl

```
LLIST
```

ohne weiteres einrichten. Er funktioniert sogar im PRO- und im RUN-Mode. Geben Sie diese Anweisung ohne Zusätze ein, werden alle Programmzeilen über den Drucker aufgelistet. Verwenden Sie diesen mit einer Zeilennummer, z.B.

```
LLIST 30
```

so wird nur die Zeile auf dem Drucker ausgegeben, deren Nummer hinter dem LLIST steht, in diesem Fall also Zeile 30:

```
30:LET I = P*(1+M
   /100)
```

Weil das Druckerpapier so schmal ist, muß das Statement in zwei Zeilen aufgelistet werden.

Der Befehl

```
LLIST 20,40
```

bewirkt den Ausdruck von Zeile 20 bis 40 (je einschließlich):

```
20:INPUT "MEHRWER
   TSTEUER IN %"
   ;M
30:LET I = P*(1+M
   /100)
40:PRINT "INCLUSI
   VPREIS IN DM";
   I
```

Das Kommando

```
LLIST 20,
```

läßt Ihren Drucker von Zeile 20 bis zum Schluß alles ausgeben,

```
LLIST ,40
```

druckt von Anfang bis Zeile 40 alles aus. Am besten probieren Sie es mit verschiedenen Zeilennummern einfach aus!

Im übrigen können Sie den LLIST-Befehl mit Zeilenmarken statt Zeilennummern genauso verwenden. Haben Sie beispielsweise die Zeilen 10 und 40 so programmiert:

```
10:"LOS":INPUT "PREIS IN DM ";P
40:"ENDE":PRINT "INCLUSIVPREIS IN DM";I
```

dann werden Ihnen durch das Kommando

```
LLIST "LOS","ENDE"
```

die Zeilen zwischen "LOS" und "ENDE", also die Zeilen 10 bis 40 (je einschließlich) ausgedruckt. Auch hier gilt: Probieren geht über Studieren!

Da wir schon beim Ausdrucken sind: Mit der Instruktion

```
LPRINT
```

können Sie die Ergebnisse der Mehrwertsteuerrechnung zu Papier bringen. Füttern Sie mal Ihren Rechner mit dem hier:

```
40:LPRINT "INCLUSIVPREIS     IN DM:";I
```

Die fünf Leerzeichen zwischen INCLUSIVPREIS und IN sind dazu da, damit der Ausdruck nicht mitten im Text in die nächste Zeile springt, sondern so erfolgt:

```
INCLUSIVPREIS
IN DM: 53.7
```

sofern Sie 50DM für den Preis und 7% für die Mehrwertsteuer gewählt haben. Das soll zum Thema LPRINT vorerst genügen. Halt! Eine Anmerkung fehlt noch: Wenn Sie in Zeile 40 aus einem PRINT ein LPRINT gemacht haben, indem Sie den Buchstaben L davorsetzten, kann es sein, daß der Computer mit einer Fehlermeldung aussteigt. Ich habe mir so beholfen, daß ich Zeile 40 völlig neu programmiert habe.

Zwei Befehle möchte ich Ihnen noch mit auf den Weg geben, bevor wir dieses Kapitel verlassen. Der erste lautet:

```
BREAK
```

Möglich, daß Sie ihn vom Boxsport her schon kennen. Dort ruft der Ringrichter:"BREAK", wenn er mit dem Verlauf eines Kampfes unzufrieden ist und schon unterbrechen die beiden Kontrahenten ihr Techtelmechtel. Sie drücken die BREAK-Taste und schon stopt Ihr Programm. Auf dem Display lesen Sie die Worte

```
BREAK IN ....
```

und dann die Zeilennummer, in der unterbrochen wurde, z.B.

```
BREAK IN 20
```

Da Ihr Programm erst einmal laufen muß, bevor Sie es unterbrechen können, ist klar, daß Sie diesen Befehl nur im RUN-Mode eingeben können.

"Und wie krieg ich mein Programm wieder flott?", werden Sie sich vielleicht jetzt fragen. Ganz einfach! Tippen Sie nur

CONT ENTER

in den Rechner und es fährt fort, als wäre nichts gewesen ("alle Zustände des Rechners ... bleiben erhalten", wie es im PC-1500 A-Handbuch heißt). Dabei ist das BASIC-Wort CONT eine Abkürzung für **CONT**inue (=fortsetzen). Bitte wenden Sie es nur im RUN-Mode an.

Für Sie war das Durcharbeiten dieses Kapitels hoffentlich nicht so, als wäre nichts gewesen, so daß Sie mit den folgenden Übungsaufgaben CONTinuen können, ohne vorher bei den Musterlösungen im Anhang zu kibitzen. Sollten Sie dennoch Schwierigkeiten mit den Übungen haben, schlagen Sie lieber erst die entsprechende Stelle in diesem Kapitel nach. Doch vorher gönnen Sie sich mal eine kleine Pause.

Aufgabe 5.1

Welche der folgenden Namen für numerische Variablen sind nicht erlaubt und warum?

a) AS b) Y1 c) 1X d) FF e) SQR
f) "I" g) IF

Aufgabe 5.2

Welche Zuweisungen für numerische Variablen sind nicht erlaubt? Bitte begründen Sie auch hier Ihre Entscheidung.

a) A = 1.23456
b) X = 2*A+34.5
c) Y = "SQR(4+3)"
d) Z = 3*(A+X)^2-33.4/X*1.2
e) C+2 = 88.4-A/4
f) F = "HOLLAREIDULLIÖÖH"

Aufgabe 5.3

Was müssen Sie als Besitzer eines PCs von SHARP bei diesen Variablennamen beachten?

a) 1. Variablenname: PREIS
   2. Variablenname: PROST
b) 1. Variablenname: PR
   2. Variablenname: Pr

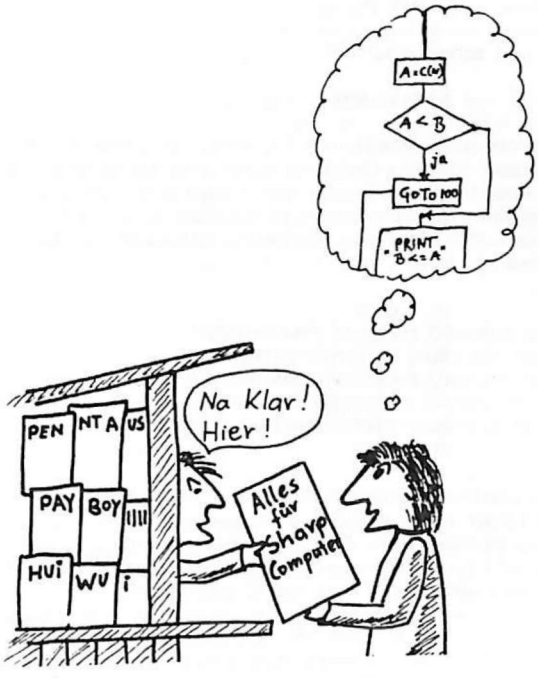
Aufgabe 5.4

Auf der nächsten Seite finden Sie ein kleines Programm zur Berechnung einer Fläche. Es sind absichtlich einige Fehler eingebaut. Finden Sie bitte heraus, wo der Wurm drin ist. Können Sie's herausfinden, bevor Sie es in den Rechner eingeben?

```
 10 : CLS
 20 : "MARKE" INPUT "EINGABE LAENGE";L
 30 : INPUT "EINGABE BREITE ; B
 40 : PASE "FLAECHENBERECHNUNG"
 50 : LET FLAECHE1 = L*B
 60 : LET FLAECHE2 = L*L
 70 : LET FLAECHE3 = B*B
 80 : PAUSE "FLAECHE1 =";FLAECHE1
90.4 : END
```

Aufgabe 5.5

Schreiben Sie bitte ein Programm, welches Ihnen den Benzinpreis pro km errechnet. Dabei soll das Programm die Eingabe des Benzinverbrauchs Ihres Wagens pro 100km in Litern und des Benzinpreises in DM pro Liter verlangen. Bitte versehen Sie Ihr Programm mit Kommentaren, so daß auch ein Außenstehender etwas damit anfangen kann.



6. Kapitel

KEIN GRAUS VOR GAUSS
- Die mathematischen Funktionen -

Hand aufs Herz - wie fühlen Sie sich, wenn Sie sich an Ihren Mathe-Unterricht erinnern? Nicht so gut? Dachte ich mir's doch. Das wird nun insofern anders, als Sie jetzt einen Mathe-Schutzengel haben - Ihren Computer. Er wird Sie vor schwierigen Zahlenrechnungen behüten. Allerdings: da er ein programmierbarer Schutzengel ist, müssen Sie sich vorher mit dem Programmieren von mathematischen Funktionen vertraut machen.

Fangen wir ganz, ganz leicht an. Angenommen, Sie wollen den Wert der Zahl π wissen, jener legendären Zahl, der Sie immer dann begegnen, wenn's in der Mathematik rundgeht: Bei Kreis- und Kugelberechnungen z.B.. Das BASIC-Wort für den Aufruf von π ist

```
PI
```

Geben Sie

```
PRINT PI
```

ein und schon erscheint:

```
3.141592654
```

Anstelle der Eingabe von PI, kann bei vielen Rechnern auch eine π-Taste gedrückt werden. Die Zahl π ist eine Konstante, das ist in der Mathematik eine in Ewigkeit und drei Tagen gleichbleibende Zahl. Einer Berechnung von Kreisflächen, Kugelvolumen etc. steht nun nichts mehr im Wege. Füttern Sie Ihren elektronischen Dreikäsehoch doch mal mit diesem Programm:

```
10 : CLS
20 : INPUT "RADIUS";R
30 : LET F = PI*R  2
40 : LET V = 4*PI*R^3/3
50 : PRINT "KREISFL F = ";F
60 : PRINT "KUGELVOL V = ";V
70 : END
```

Die mathematischen Formeln in Zeile 30 und 40 hat Ihr Micro-Schutzengel leider nicht parat, Sie müssen sie wie eh und je in einer Formelsammlung nachschlagen. Probieren Sie das kleine Programm doch gleich mal aus und lassen Sie seine Frage nach dem Radius nicht unbeantwortet. Sie werden sehen, wie flink der Computer Ihnen die Kreisfläche und das Kugelvolumen berechnet. Damit Sie sich an den Ergebnissen in aller Ruhe ergötzen können, habe ich den Befehl PRINT statt PAUSE gebraucht, so daß das Programm hinter den PRINT-Anweisungen erst nach Druck der ENTER-Taste weiterläuft.

Im großen und ganzen müssen Sie mir doch zustimmen: Die Sache mit PI war doch nicht schwer, oder? Einfach gehts auch weiter und zwar mit der Quadratwurzel (die kennen Sie ja schon). Die Taschencomputer haben für sie die Taste √‾. Tippen Sie die Folge

√‾4

ein, sollten Sie auf diese Weise die Quadratwurzel aus 4 errechnen wollen. Daß das Ergebnis 2 beträgt, brauche ich Ihnen nicht zu sagen. Weil wir aber BASIC lernen wollen, wählen wir auch hier das BASIC-Wort für die Quadratwurzel:

SQR(4)

Jetzt muß ich einige wichtige Dinge bezüglich der mathematischen Funktionen loswerden:

1. Die Zahl oder der mathematische Ausdruck, auf die eine Funktion angewandt wird, nennt man Argument. In obigem Beispiel ist 4 das Argument von SQR.
2. Als Argument für numerische Funktionen dürfen Sie (sowohl in der Mathematik als auch in der Computerei) Zahlen, numerische Variablen und mathematische Ausdrücke verwenden. Argumente, die aus mehr als nur einer Zahl oder einer Variablen bestehen, müssen Sie in Klammern setzen. Da dies von allen Computern etwa anders gehandhabt wird, empfehle ich Ihnen, sich anzugewöhnen, das Argument einer Funktion immer in Klammern zu setzen.

Zugegeben, das hört sich alles sehr theoretisch und schulisch an. Ich gebe Ihnen wieder am besten ein paar Beispiele, damit es Ihnen leichter die Hirnwindungen hinunter rutscht. Wie Sie eine Zahl als Argument einsetzen, haben Sie oben schon gesehen. Wenn Sie SQR(X) wählen, dann haben Sie die Variable X im Argument der Quadratwurzel stehen. In den Ausdrücken SQR(4+3) oder SQR(5*A-2/B) haben Sie ein Beispiel für arithmetische Ausdrücke im Argument einer Funktion. Selbstverständlich dürfen Sie auch eine weitere Funktion als Argument verwenden. SQR(SQR(16)) ist dafür ein Beispiel.

3. Es gibt Funktionen, bei denen nicht alle Zahlenwerte als Argument zugelassen sind. Wieder ist SQR dafür ein gutes Beispiel. Sie dürfen nämlich keine Quadratwurzel aus einer negativen Zahl ziehen. So würden Ihnen alle Computer der Welt den Ausdruck SQR(-1) mit einer Fehlermeldung beantworten. Ebenso falsch ist:

```
10 : X = -56
20 : A = SQR(X)
```

   da X ja einen negativen Wert angenommen hat.
4. Bei numerischen Funktionen dürfen Sie keine Textausdrücke oder Textvariablen (die lernen Sie auch noch kennen) als Argument verwenden. Sowas wie SQR("APRIL, APRIL") oder SQR(A$) führt zu einer prompten Fehlermeldung.

So, jetzt sind Sie reichlich fit für die anderen Funktionen. Was ist denn z.B., wenn Sie es sich in den Kopf gesetzt haben, nicht die Quadrat- sondern die Kubikwurzel zu ziehen? Das ist die dritte Wurzel aus einer Zahl. Dafür gibt es leider keine spezielle BASIC-Funktion und Sie müssen auf die Potenzschreibweise ausweichen. In der Mathematik können Sie nämlich "dritte Wurzel aus X" auch so scheiben:

$$\sqrt[3]{X} = X^{1/3}$$

Programmiert wird dies so:

```
X^(1/3)
```

Vergessen Sie bloß die Klammer nicht, sonst wird das Ihr Computer so interpretieren: X 1/3 = (X 1)/3. Diese Potenzschreibweise können Sie natürlich auch für die 4., 5. usw. Wurzel verwenden.

Die nächste Funktion, mit der wir uns beschäftigen, heißt:

```
ABS
```

Keine Angst, das ist keine neue ansteckende Krankheit wie AIDS, sondern eine Funktion, die Ihnen den **ABS**olutbetrag einer Zahl ermittelt. Das ist der Wert einer Zahl ohne Berücksichtigung des Vorzeichens. Geben Sie mal

```
10 : PRINT ABS(-10)
20 : PRINT ABS(10)
30 : PRINT ABS(-3+4)
40 : PRINT ABS(-8*2)
```

in den Rechner ein. Nachdem Sie das Programm gestartet haben, erscheint eine 10 auf der Anzeige. Das ist der Wert von ABS(-10). Drükken Sie nun ENTER, bleibt die 10 stehen, da auch ABS(10) = 10 ist. ENTERn Sie weiter, so kommen nacheinander 1 (von ABS(-3+4)) und 16 (von ABS(-8*2)) ins Bild. Selbstverständlich sind auch Variablen als Argument von ABS erlaubt. Die ABS-Funktion ist für alle Zahlen zu haben.

Eine typische "Computer-Funktion" ist

```
INT
```

Es handelt sich dabei um die **INT**egerfunktion, die sich aus einer Zahl immer nur den ganzzahligen Anteil herauspickt. Im Falle eines positiven Arguments ist das einfach der Teil vor dem Dezimalpunkt. Beispielsweise wird nach

```
PRINT INT(2.3844)
```

eine 2 ausgegeben. Dies ist auch bei

```
PRINT INT(2.9999)
```

der Fall. Diese Funktion rundet also nicht, sondern "hackt" die Stellen

nach dem Dezimalpunkt rigoros ab. Wenn Sie ihr allerdings eine negative Kommazahl zu futtern geben, dann wird eine negative Zahl daraus gemacht, deren Betrag größer, mindestens aber gleich dem des Arguments ist. Damit Sie sich darunter etwas vorstellen können, gleich ein Beispiel: Die Anweisung

```
PRINT INT(-3.2)
```

hat das Resultat:

```
-4
```

Ein letztes mal möchte ich Sie darauf aufmerksam machen, daß wie bei jeder mathematischen Funktion auch Variablen und arithmetische Ausdrücke als Argument erlaubt sind. Auch diese Funktion kann auf alle Zahlen losgelassen werden. Vielleicht gehören Sie aber zu den sensiblen Menschen, die eine Zahl lieber ordentlich runden, anstatt ihr die Kommastellen einfach abzuhauen. Für diesen Fall bietet sich die folgende kleine Routine an, in der auch die INT-Funktion vorkommt:

```
10 : CLS
20 : INPUT "ZU RUNDENDE ZAHL ";Z
30 : INPUT "WELCHE STELLE ";S
40 : Z = INT(Z*10^S+0.5)/10^S
50 : PRINT "ERGEBNIS ";Z
60 : END
```

Um zu sehen, wie es funktioniert, verfolgen wir den Weg, den eine Zahl Z durch das Programm geht. Angenommen, Sie geben für Z den Wert 1.2467 ein und Sie wollen nun diese Zahl auf drei Stellen nach dem Dezimalpunkt runden. Dann wählen Sie für S die 3. In Ihrem Kopf haben Sie diese Rundung natürlich schon längst durchgeführt: Es muß Z = 1.247 herauskommen. Zeile 40 ist das Kernstück des Programms. Hier wird die Zahl Z ins Argument von INT genommen, aber gleichzeitig mit 10^S, also hier 10^3 mutipliziert. Dadurch werden alle Ziffern, einschließlich der Stelle, auf die gerundet werden soll, vor den Dezimalpunkt geholt und Zeile 40 sieht in diesem Moment so aus:

```
40 : Z = INT(1246.7+0.5)/1000
```

"Was soll denn das +0.5 in der Klammer?", werden Sie sich jetzt vielleicht fragen. Stellen Sie sich einmal vor, es wäre nicht da, dann würde INT aus 1246.7 den Wert 1246 ergeben. Das führte zu der falschen Rundung 1.246 statt der richtigen 1.247. Durch 1246.7+0.5 = 1247.2 erreicht man, daß die Zahl durch INT zu 1247 gestutzt wird. Die Division durch 1000 rückt den Dezimalpunkt wieder an seinen Platz zurück. Zeile 50 braucht Ihnen nur noch

```
ERGEBNIS  1.247
```

auszugeben und die Rundung ist vollbracht. Ich glaube, dieses Programm ist es wert, auf dem Drucker aufgelistet zu werden. Tun Sie das doch mit LLIST und tun Sie's zu den Akten. Selbstverständlich sollten Sie es auch mal probelaufen lassen.

Hat sich vorhin die ABS-Funktion nur um den Betrag einer Zahl geschert, das Vorzeichen ließ sie unter den Tisch fallen, so ist es bei der Funktion

SGN

genau umgekehrt. Sie schaut sich nur das Vorzeichen einer Zahl an, der Rest ist ihr Wurst. Ist eine Zahl X positiv (>0 heißt das in der Sprache der Mathematik), so ist SGN(X) = 1. Für X = 0 wird SGN(X) = 0 und für negative X (X<0) ist SGN(X) = -1. Ein Beispiel möchte ich Ihnen hier geben:

```
10 : LET A = -20
20 : LET B = SGN(3*4+A)
30 : PRINT B
```

ergibt -1 auf der Anzeige. Sie dürfen ungestraft alle Zahlen als Argument der SGN-Funktion verwenden.

Jetzt kommen zwei exotischere Geräte auf Sie zu. Das eine von ihnen wird "Exponentialfunktion"genannt und im Mathe-Jargon so geschrieben: e^x. Das bedeutet, daß die Eulersche Zahl e = 2.718281828 x-mal mit sich selbst multipliziert wird. Ein Aha-Erlebnis wird in Ihnen aufsteigen, hatten Sie dies nicht schon mal in diesem Buch gelesen? Natürlich, das war bei den 10er-, 2er- und 16er-Potenzen der Fall. Während dort die Basis 10, 2 oder 16 war, ist sie hier eben e. Allerdings wird die Exponentialfunktion nicht als E∧X in den Computer gegeben, sondern so:

EXP(X)

Für X dürfen Sie beliebige Zahlen wählen. Natürlich darf auch hier ein Beispiel nicht fehlen. Tippen Sie doch mal

PRINT EXP(6.843)

in Ihren Rechner ein. Ergebnis:

937.2968115

Die Funktion, die ich Ihnen jetzt vorstellen möchte, heißt "natürlicher Logarithmus", in BASIC mit

LN

bezeichnet. Sie ist im wahrsten Sinne des Wortes das Gegenstück zur Exponentialfunktion. Die Mathematiker tauften solche Gegenstücke auf den Namen "Umkehrfunktion". Die Umkehrfunktion tut genau das Gegenteil von dem, was eine Funktion macht. Wenn Sie also beide gegeneinander antreten lassen, bleibt das Argument so wie es am Anfang war. Ich kann es Ihnen nachfühlen, ein Beispiel wäre Ihnen jetzt recht. Hier ist es:

PRINT LN(EXP(100))

ergibt schlicht und ergreifend: 100. Es ist also alles beim alten geblieben.

Das gleiche gilt auch für den Ausdruck

```
PRINT EXP(LN(100))
```

d.h. EXP ist seinerseits die Umkehrfunktion von LN. Die Zahl, Variable oder der arithmetische Ausdruck im Argument der LN-Funktion muß aber immer positiv sein.

```
PRINT LN(-1)
```

würde daher mit einer Fehlermeldung erwidert. Hier noch einige korrekte Beispiele:

```
10 : PRINT LN(10)
20 : PRINT LN(100)
30 : PRINT LN(1000)
```

Wenn Sie von PRINT zu PRINT brav die ENTER-Taste drücken, erhalten Sie nacheinander die Ergebnisse:

```
2.302585093
4.605170186
6.907755279
```

Der **LOG**arithmus zur Basis 10 wird im SHARP-BASIC mit

```
LOG
```

bezeichnet. Er ist die Umkehrfunktion zu 10∧X. Auch sie läßt nur positive Argumente gelten. Beispiele:

```
10 : PRINT LOG(2.15)
20 : PRINT LOG(10∧5)
30 : PRINT 10∧(LOG(5))
40 : PRINT LOG(-35)
```

Das Ergebnis dieser Zeilen ist:

```
3.324384599E-01
5
5
ERROR 39
```

Letzteres gilt zumindest beim PC-1500 A, andere Computer haben andere Fehlermeldungen.

So weit, so gut. Wenden wir uns jetzt einem neuen Funktionentyp zu. Gemeint sind die Winkel- oder Kreisfunktionen. Ein dritter Name für diese Dinger ist "trigonometrische Funktionen". Sicher haben Sie schon mal was von Sinus, Cosinus, Tangens & Co. gehört. Und wozu braucht man sie? Um sie auf Winkel anzuwenden. Das sagt Ihnen vielleicht auch noch nicht viel. Jedenfalls kommen diese Funktionen häufig in der Geometrie, dem Vermessungswesen und der Schwingungslehre vor. Schon wenn Sie auch nur

die Saite einer Gitarre zupfen, hagelt es nur so von Sinus- und Cosinusschwingungen. Damit wir uns richtig verstehen - ich behandle diese Funktionen nicht, um Ihnen die Freude am Gitarrenspiel zu verderben, sondern weil sie auf Ihrem Computer programmierbar sind.

Der Ärger geht schon mit den Winkeln los, also den Argumenten bzw. Ergebnissen der trigonometrischen Funktionen. Es gibt nämlich insgesamt drei Einheiten, in welchen man Winkel messen kann.

Die erste Einheit ist Ihnen sicher geläufig, es handelt sich um Grad (^{o}). Ein Kreis hat einen Winkel von 360^o, ein Halbkreis hat 180^o, ein Viertelkreis 90^o usw.. Der 90^o-Winkel wird auch "Rechter Winkel" genannt. Wenn Sie also nachts wegen eines Mathealptraums senkrecht im Bett stehen, dann wissen Sie ab jetzt, daß dabei Ihre Körperachse mit der Ihres Bettes einen Winkel von 90^o bildet. Wollen Sie den Computer die Winkel in dieser Einheit berechnen lassen, dann geben Sie das Kommando

DEGREE

ein. Der Rechner arbeitet dann im DEG-Mode und zeigt Ihnen das auch auf dem Display an. Eine andere Bezeichnung für Grad ist "Altgrad".

Wo ein Altgrad ist, muß auch ein Neugrad sein. In der Tat, dem ist so. In der Neugrad-Einteilung hat der rechte Winkel 100^g und nicht mehr 90^o. 180 alte Grad entsprechen dann 200 neuen Graden (200^g) und ein voller Kreis hat 400^g (statt bisher 360^o). Soll Ihr Computer alles, was mit Winkeln zu tun hat, in Neugrad-Einheiten erledigen, tippen Sie

GRAD

ein. Wieder können Sie auf der Anzeige erkennen, in welchem Modus Ihr Rechner sich befindet.

Aber aller guten Dinge sind drei. Das müssen sich wohl auch die Erfinder der Winkeleinheiten gedacht haben und so bescherten Sie uns das Bogenmaß. Das Bogenmaß mißt nicht in Grad, sondern die Länge des Kreisbogens eines Winkels. Einem Winkel von 360^o entspricht der gesamte Kreisumfang. Der Einfachheit halber setzt man den Radius dieses Kreises gleich 1, woraus sich ein Bogenmaß von 2π für den 360^o-Winkel ergibt. Der Bogen eines Halbkreises ist folglich halbso lang und mißt π, was 180^o entspricht. Der Winkel von 90^o hat im Bogenmaß den Wert $\pi/2$. Wollen Sie in dieser Einheit rechnen, so befehlen Sie dem Computer

RADIAN

und schon werkelt er im RAD-Mode dahin. Natürlich teilt er Ihnen das auf der Anzeige mit. Zur besseren Übersicht stelle ich Ihnen nochmals alle Winkeleinheiten zusammen:

DEGREE	0^o - 360^o	DEG-Mode (Altgrad)
GRAD	0^g - 400^g	GRAD-Mode (Neugrad)
RADIAN	0 - 2π	RAD-Mode (Bogenmaß)

Und zur Veranschaulichung lassen Sie Ihren Computer noch dieses Pro-

gramm ausführen:

```
10 : CLS
20 : DEGREE
30 : INPUT "WINKELEINGABE ";A
40 : PAUSE "SINUS VON ",A;" ALTGRAD"
50 : PRINT SIN(A)
60 : GRAD
70 : PAUSE "SINUS VON ";A;" NEUGRAD"
80 : PRINT SIN(A)
90 : RADIAN
100 : PAUSE "SINUS VOM BOGENMASS ";A
110 : PRINT SIN(A)
120 : END
```

Geben Sie für A nun z.B. 90 ein. Solange der Computer in Altgrad rechnet, wird der Sinus von 90^{o} = 1 sein. Geht der Computer in Zeile 60 in Neugrad über, dann wird SIN(A) = 9.876883406E-01, denn der Sinus von 90^{g} ist eben etwas anders als der von 90^{o}. Nach dem Umschalten auf den RAD-Mode, wird es etwas komplizierter, das Ergebnis (8.939966634E-01) zu interpretieren. Das Bogenmaß 90 ist eine unhandliche Größe, denn es entspricht etwa $28.65*\pi$,was wiederum 14 vollen Kreisdurchläufen und einem Restwinkel mit Bogenmaß $0.65*\pi$ entspricht.

Im obigen Programm sind Sie bereits auf die erste trigonometrische Funktion gestoßen, den **SIN**us. Es stehen Ihnen aber noch weitere zur Verfügung, nämlich **COS**inus und **TAN**gens. Ihre BASIC-Worte sind:

```
SIN
COS
TAN
```

Sollte es Ihnen nicht mehr geläufig sein, wie diese Funktionen definiert sind, dann schlagen Sie in Ihren alten Mathematikbüchern nach. Hier nur soviel: Für SIN und COS dürfen Sie alle Zahlen verwenden, ohne daß Ihr Computer auch nur den Muckser einer Fehlermeldung macht, beim TAN sehen Sie sich vor, wenn Winkel von 90^{o} (=100^{g} bzw. $\pi/2$), 270^{o} (=300^{g} oder $3\pi/2$) usw. auftauchen, da sonst der Funktionswert von TAN unendlich große positive bzw. negative Werte annimmt und der Computer deshalb mit einer Fehlermeldung rebelliert.

Zum Aufwärmen Ihrer Schulmathematik dürfen Sie wieder eine kleine Fingerübung machen und dieses Prográmmchen in Ihren Rechner tippen:

```
10 : CLS
20 : DEGREE
30 : INPUT "EINGABE WINKEL ";A
40 : PRINT "SIN VON A = ";SIN(A)
50 : PRINT "COS VON A = ";COS(A)
60 : PRINT "TAN VON A = ";TAN(A)
70 : END
```

Mit Zeile 20 legen Sie sich auf die Einheit Altgrad fest. Zeile 30 erbittet von Ihnen die Eingabe eines Winkels, und wie ich Sie einschätze, wer-

den Sie dieser Bitte auch prompt nachkommen. Versuchen Sie es mal mit einem Winkel von 90° d.h. geben Sie die Zahl 90 ein. Danach spuckt Ihnen Zeile 40 das aus:

```
SIN VON A =  1
```

und Zeile 50 präsentiert Ihnen:

```
COS VON A =  0
```

Aber was meinen Sie wohl, bietet Ihnen Zeile 60? Eine Fehlermeldung? Oh, wie recht Sie doch haben! Erinnern Sie sich noch? Der Wert 90 als Argument für den Tangens ist (wenn Sie die Winkel in Altgrad angeben) nicht erlaubt. Probieren Sie das Programm ruhig mit einer Reihe von anderen Winkeln aus. Bei A = 45 dürfte nichts mehr schief gehen, es sei denn, das Ergebnis paßt nicht mehr auf das Display. In diesem Falle kürzen Sie einfach den Kommentar der PRINT-Anweisungen (z.B. schreiben Sie "SIN(A)=" statt "SIN VON A = "). Testen Sie auch die anderen Winkeleinheiten, indem Sie in Zeile 20 den entsprechenden Befehl für die Rechnung in Bogenmaß oder Neugrad eingeben (die Anweisung dafür kennen Sie doch sicher noch, oder?).

Selbstverständlich gibt es für SIN, COS und TAN auch die zugehörigen Umkehrfunktionen, die ich Ihnen kurz vorstellen möchte. Die Umkehrfunktion für SIN ist der **ArcusSiNus**, sein BASIC-Wort heißt:

```
ASN
```

Für COS ist das der **ArcusCoSinus** mit dem BASIC-Wort:

```
ACS
```

Für den TAN haben wir den **ArcusTaNgens**. Das BASIC-Wort hierfür ist:

```
ATN
```

Wie war das noch? Damit eine Umkehrfunktion wirklich eine Umkehrfunktion ist, muß sie, auf die Funktion angewandt, wieder das ursprüngliche Argument herzaubern. Es muß also z.B. die Funktion ASN angewandt auf SIN(X) wieder X ergeben. Andererseits sind SIN, COS und TAN die Umkehrfunktionen zu ASN, ACS und ATN. Probieren Sie es mit folgendem Programm aus:

```
10 : CLS
20 : DEGREE
30 : PRINT ASN(SIN(45))
40 : PRINT SIN(ASN(0.7))
50 : END
```

Ich will Rumpelstilzchen heißen, wenn Sie nicht die Ergebnisse

```
45
0.7
```

erhalten - es sei denn, Sie haben falsch programmiert (aber das ist schon eine sehr verwegene Vermutung). Bleibt noch nachzutragen, daß die Funktionen ASN und ACS nur Argumente zwischen -1 und +1 vertragen. Ansonsten werden Ihnen ASN, ACS und ATN kaum begegnen, es sei denn, Sie sind Ingenieur oder Physiker.

Bisher haben wir immer mit vollen Graden gerechnet (gemeint sind im weiteren die Altgrad). Aber vielleicht ist Ihnen sowas schon mal begegnet:

$$20^{o}\ 43'\ 34.5''$$

Die erste Zahl mit der obengestellten verkümmerten Null ist Ihnen hinreichend bekannt. Und die 43' ? Das sind 43 Minuten, wobei 1 Grad (1^{o}) in 60 Minuten (60') unterteilt ist. Dann kann es sich bei 34.5'' eigentlich nur noch um Sekunden handeln? Genau so ist es! Dabei ist 1 Minute (1') gleich 60 Sekunden (60''). Ein Grad hat demnach 3600'' (Sek.). Die Ziffern hinter dem Dezimalpunkt stellen die Sekundenbruchteile dar. Zur besseren Orientierung eine kleine Übersicht:

$$1^{o} = 60' = 3600''$$

$$1' = 60''$$

Sie haben es schon richtig begriffen, die Einteilung der Winkelgrade in Minuten und Sekunden ist genau die gleiche wie die der zeitlichen Stunde. Zum Beispiel bedeutet die Angabe

$$1^{h}\ 15^{min}\ 03.6^{sec}$$

1 Stunde, 15 Minuten und 3.6 Sekunden. In der Gebrauchsanleitung zum PC-1500 A ist das folgende Schema zu finden. Sie können es für Zeit- und Winkeleinheiten gleichermaßen gebrauchen.

$$\underbrace{g_1\ g_2 \cdots g_n}_{\text{Grad/Stunden}}\quad \underbrace{m_1\ m_2}_{\text{Minuten}}\quad \underbrace{s_1\ s_2}_{\text{Sekunden}}\ .\ \underbrace{b_1\ b_2 \cdots b_n}_{\text{Sek.-Bruchteile}}$$

Wegen der Aufteilung der Grad/Stunden in 60 Minuten bzw. der Minuten in 60 Sekunden bezeichnet man derartige Angaben als Sexagesimalzahlen. Das braucht uns nicht weiter zu stören, sind wir doch die sexagesimalen Zeiteinheiten von Kind auf gewohnt. Aber was ist, wenn irgend ein Scherzkeks unsere 20^{o} 43' 34.5'' so schreibt:

$$20.72625^{o}$$

Sie werden lachen, er hat damit auch noch recht! Er hat nämlich die Sexagesimalzahl 20^{o} 43' 34.5'' in die Dezimalzahl 20.72625^{o} umgewandelt. Wie er das wohl gemacht hat? Vermutlich hat er auch einen SHARP-Computer und dieses Programm eingegeben:

```
10 : CLS
20 : INPUT "SEXAGES.?";A
30 : PRINT "DEZIMAL:";DEG(A)
40 : END
```

Ich rate Ihnen: Tun Sie desgleichen. Sobald Sie das Programm in Zeile 20 dazu auffordert, einen sexagesimalen Wert einzugeben, tippen Sie die 20^{o} 43' 34.5" so in den Rechner ein:

20.43345

Sie lesen dann auf der Anzeige:

DEZIMAL: 20.72625

Die Umwandlung von Sexagesimal- in Dezimalzahlen wird also durch die Anweisung

DEG

vollbracht. Genauso können Sie mit dem Beispiel für eine Zeitangabe verfahren. Geben Sie die entsprechende Zahlenreihe so in den Computer:

1.15036

DEZIMAL: 1.251

lautet wahrheitsgemäß das Ergebnis. Wenn Sie die Dezimaldarstellung wieder in Sexagesimalform haben wollen, ändern Sie Zeile 20 und 30 so um:

```
20 : INPUT "DEZIMAL? ";A
30 : PRINT "SEXAGES.: ";DMS(A)
```

Füttern Sie den Rechner diesmal mit

20.72625

SEXAGES.: 20.43345

wird Ihnen als Ergebnis angeboten, wobei Sie diese Zahl bitte so interpretieren:

20^{o} 43' 34.5"

Die Funktion

DMS

rechnet also Dezimalwinkel und -stunden in die gebräuchlichere Sexagesimaldarstellung um. Testen Sie es doch auch mit 1.251. Erwartungsgemäß kommt 1.15036 heraus.

Sie werden aufatmen, wir sind bei der vorläufig letzten Funktion angelangt. Hatten Sie je vermutet, daß Ihr Computer etwas mit Fortunas Amphore gemein hat? Oh doch, das hat er. Sie können ihn nämlich Schicksal spielen lassen, indem Sie die Funktion

RND

aufrufen. Ihre Aufgabe ist es, Zufallszahlen zu erzeugen. Um zu sehen, wie RND arbeitet, programmieren Sie mal das:

```
10 : CLS
20 : INPUT "BELIEBIGE ZAHL ";Z
30 : PRINT "RND(Z): ";RND(Z)
40 : END
```

Zuerst wird Sie natürlich das Programm mit den Worten BELIEBIGE ZAHL dazu auffordern, irgendeine Zahl einzugeben. Wählen Sie eine Zahl Z zwischen 0 und 0.999999999 (also kleiner 1), und RND(Z) erzeugt Ihnen eine Zufallszahl, die auch zwischen 0 und 0.999999999 liegt. Haben Sie sich aber für ein Z entschieden, das größer oder gleich 1 ist, zieht RND(Z) eine Zahl zwischen 1 und INT(Z). Sie werden bemerken, daß jetzt immer nur ganze Zufallszahlen erzeugt werden.

Wie kommt Ihr exakter Computer eigentlich dazu, den unvorhersehbaren Zufall heraufzubeschwören? Meist aus dem gleichen Grund wie Sie - weil er spielen will. Wie das zugeht, demonstriert Ihnen das folgende kleine Programm:

```
10 : CLS
20 : WAIT 200
30 : PRINT "SPIELER 1 WUERFELT EINE";RND(6)
40 : PRINT "SPIELER 2 WUERFELT EINE";RND(6)
50 : END
```

Interessant sind nur die Zeilen 30 und 40, in denen der Computer zwei Spieler gegeneinander würfeln läßt. Mit RND(6) wird eine Zufallszahl zwischen 1 und 6 produziert, genauso wie beim richtigen Würfelspiel. Hat das Programm für Spieler 1 eine 3 und für Spieler 2 eine 5 auserkoren, so lesen Sie auf dem Display dies:

```
SPIELER 1 WUERFELT EINE 3

SPIELER 2 WUERFELT EINE 5
```

Starten Sie das Programm mehrmals und Sie werden sehen, daß immer andere Zahlen zwischen 1 und 6 erscheinen. Intern geht das so vor sich, daß der Rechner beim ersten Aufruf von RND(6) eine lange Folge von zufällig angeordneten Zahlen zwischen 1 und 6 bildet. Dieser Folge bedient er sich jedesmal, wenn er erneut auf RND(6) trifft. Sie bleibt auch beim Ausschalten des Rechners erhalten. Wollen Sie ganz sicher gehen und verhindern, daß sich Ihre Zufallszahlen irgendwann wiederholen, so geben Sie zu Beginn des Programms dem Computer die Anweisung

```
RANDOM
```

Diese sorgt dafür, daß eine neue Zufallszahlenfolge initiiert wird. Sie finden den RANDOM-Befehl auch in unserem Roulette-Programm in Kapitel 16.

Nachdem Sie sich mit diesen Spielereien etwas von den mathematischen Funktionen erholen konnten, sind Sie sicher fit für ein paar neue Übungs-

aufgaben. Vielleicht schließen Sie mit sich selbst eine Wette ab, daß Sie es schaffen, alle zu lösen, ohne vorher auf die Musterlösungen zu spicken.

Aufgabe 6.1

Welche Argumente in den folgenden Funktionen führen zu einer Fehlermeldung?

a) SIN(175)
b) SQR(-4)
c) INT(-80043.75)
d) COS("ALPHA")
e)
```
10 : RADIAN
20 : PRINT TAN(PI/2)
30 : END
```
f)
```
10 : LET A = 32
20 : LET B = 8
30 : LET A = A/B*4-20
40 : PAUSE LN(A)
50 : END
```

Versuchen Sie, die Aufgaben "im Kopf" zu lösen, bevor Sie sie in den Computer eintippen.

Aufgabe 6.2

Welche Ergebnisse liefern die folgenden arithmetischen Ausdrücke (bitte rechnen Sie auch hier zuerst im Kopf)

a) SQR(4+12)
b) INT(4.799)*8
c) ABS(-18/2)
d) LN(EXP(7.347))
e) 10^(LOG(2))

Aufgabe 6.3

Welches Ergebnis liefert Ihnen dieses Rundungsprogramm, wenn Sie für Z den Wert -3.1442 eingeben und auf die 3.Stelle (also S = 3) gerundet werden soll?

```
10 : CLS
20 : INPUT "ZU RUNDENDE ZAHL? ";Z
30 : INPUT "WELCHE STELLE? ";S
40 : Z = INT(Z*10^S+0.5)/10^S
50 : PRINT "ERGEBNIS ";Z
60 : END
```

Bitte lösen Sie diese Aufgabe, ohne Ihren Computer vorher zu befragen.

Aufgabe 6.4

Bitte schreiben Sie ein Programm, das Ihnen die Oberfläche und das Volumen einer Kugel berechnet. Dabei soll das Programm folgendes leisten: Zuerst soll die Anzeige sauber gemacht werden, dann soll mit einem Kommentar ein Wert für den Radius R der Kugel angefordert werden. Als nächstes soll das Programm die Kugeloberfläche O und das Volumen V nach diesen Formeln berechnen:

$$O = 4\pi R^2 \qquad V = \frac{4\pi R^3}{3}$$

Am Schluß sollen die Ergebnisse für die Oberfläche und das Volumen mit einem erklärenden Text ausgegeben werden.

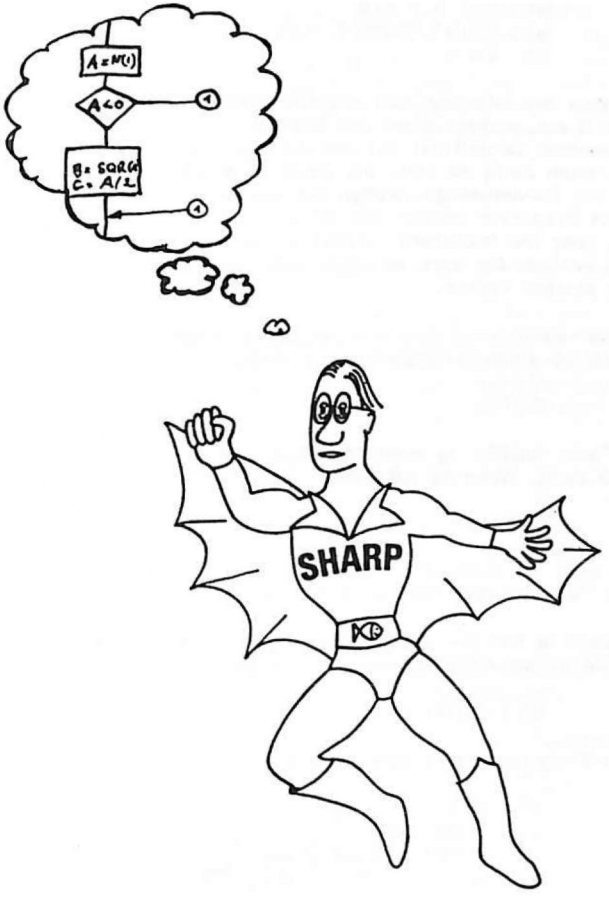
Aufgabe 6.5

Schreiben Sie bitte ein Programm, das Ihnen die Fläche F eines Dreiecks nach der Formel

$$F = \frac{1}{2} ABsinG$$

berechnet. A und B sind Seiten dieses Dreiecks und G ist der Winkel, den sie einschließen. Der Winkel G soll im DEG-Modus eingegeben werden. Bitte schmücken Sie Ihr Programm mit Kommentaren aus und testen Sie es mit folgenden Daten:

A = 5cm B = 4cm G = 50^{o}



7. Kapitel

BRINGEN SIE IHR PROGRAMM ZUM ROTIEREN
- Von Schleifen und weiteren Befehlen -

Bisher haben wir den Computer unsere Programme stur von der ersten bis zur letzten Zeile durchackern lassen. Wollten wir nach einem Programmdurchlauf wieder anfangen, mußten wir das Programm mit RUN neu starten. Ein kleines Beispiel soll uns das nochmal verdeutlichen:

```
10 : CLS
20 : WAIT 320
30 : INPUT "ERSTE ZAHL ";A
40 : INPUT "ZWEITE ZAHL ";B
50 : LET S = A+B
60 : PRINT "SUMME A+B = ";S
70 : END
```

Sie haben die Situation voll erkannt, dieses Programm liest zwei Zahlen A und B ein, addiert diese und präsentiert dann ihre Summe mit entsprechendem Begleittext auf der Anzeige. Dabei wird immer zuerst Zeile 10, dann Zeile 20 usw., bis Zeile 70 abgearbeitet. Da ist dann das Ende der Fahnenstange. Wollen Sie wieder zwei Zahlen addieren, müssen Sie das Programm wieder mit RUN anwerfen. Nun kenne ich Ihre Geduld nicht, aber bei höchstens hundert solcher Summen wird das auch Ihnen lästig werden. Da wäre es schon ganz nett, wenn das Programm von selbst starten könnte.

Na, dann wollen wir ihm mal auf die Sprünge helfen. Dafür gibt es doch tatsächlich einen Sprungbefehl. Er heißt

```
GOTO
```

also "gehe nach" - ja wohin denn? Zu der Zeilennummer, die hinter dem GOTO steht. Wenn es z.B. heißt

```
100 : GOTO 10
```

dann weiß der Computer, daß jetzt ein Sprung nach Zeile 10 fällig ist (statt "Sprung" sagt man auch "Verzweigung").

Das paßt ja wie die besagte Faust auf's Auge in unsrer Routine. Schieben Sie in das Additionsprogramm diese Zeile zwischen Zeile 60 und 70:

```
65 : GOTO 10
```

Unser Programm sieht dann so aus:

```
10 : CLS
20 : WAIT 320
30 : INPUT "ERSTE ZAHL ";A
40 : INPUT "ZWEITE ZAHL ";B
```

```
50 : LET S = A+B
60 : PRINT "SUMME A+B = ";S
65 : GOTO 10
70 : END
```

Bis Zeile 60 läuft die Kiste ab wie gehabt, aber dann kommt Zeile 65 : GOTO 10 und schon springt Ihr Computer nach Zeile 10 zurück, löscht die Anzeige und fordert wieder zwei Zahlen zur Addition an. Die geben Sie ein, wieder wird die Summe gebildet und auf dem Display angezeigt, abermals springt, GOTO 10 sei Dank, das Programm auf Zeile 10 zurück und ... merken Sie was? Sie drehen sich immer im Kreis, ständig wird das Programm wiederholt, das END in Zeile 70 wird nie erreicht. In der Computersprache heißt so ein Kreis "Schleife" und weil er dauernd durchlaufen wird, nennt man ihn auch "Endlosschleife". Da der Sprung GOTO 10 in jedem Fall, also bedingungslos, durchgeführt wird, hört er auf den Namen "unbedingter Sprung".

Jetzt ist eine kleine Zwischenbemerkung nötig. Der Befehl GOTO mit einer nachfolgenden Zeilennummer hat nämlich die wunderbare Eigenschaft, daß er so für alle Computer verständlich ist. Ein Taschencomputer wie der PC-1500 A kann allerdings mit ein paar Spezialitäten aufwarten. Sein GOTO springt nicht nur Zeilennummern, sondern auch Zeilen mit einer Marke an. Sie kennen das schon vom LIST-Befehl. Beispielsweise ist das hier möglich:

```
100 : "MARKE":PAUSE "HIER IST ZEILE 100"
150 : GOTO "MARKE"
```

Der GOTO-Befehl springt nun die Zeile 100 mit der "MARKE" genauso an, als hätten Sie 150 : GOTO 100 programmiert. Wieder müssen Sie lediglich beachten, daß Sie für den Markennamen nicht mehr als 16 Zeichen verwenden und ihn in Anführungszeichen setzen. Vergessen Sie auch nicht, ihn durch einen Doppelpunkt vom Rest der Zeile zu trennen. Beim PC-1500 A dürfen Sie auch Variablennamen und arithmetische Ausdrücke als Sprungadressen verwenden, es ist also z.B. folgendes machbar:

```
100 : "MARKE":PAUSE "HIER IST ZEILE 100"
110 : LET A = 20
120 : LET C = 100
150 : GOTO 5*A
```

oder:

```
150 : GOTO C
```

Sicher, geistreich ist dieses Beispiel nicht gerade, aber beidesmal wird, wie in den vorhergehenden Fällen, die Zeile 100 angesprungen.

Alles schön und gut, aber Sie sitzen immer noch in einer Endlosschleife drin und geben nach wie vor zwei Zahlen für die Addition ein. Wenn Sie jetzt einwenden, daß dies nicht Ihr Lebensinhalt sein kann, gebe ich Ihnen völlig recht - nur: Wie kommen Sie wieder heraus? Den Computer ausschalten ist eine Möglichkeit, gewiß, auch der Druck auf die BREAK-Taste bereitet dem Programmablauf ein jähes Ende. Aber,ehrlich, beide

Methoden sind etwa so genial wie der Sprung aus dem dritten Stock, wenn Sie zum Einkaufen das Haus verlassen. Dagegen wäre es doch sehr komfortabel, wenn wir dem Computer mit Hilfe des Programms mitteilen könnten, wann wir keine Lust mehr haben, ständig Zahlen zu addieren. In der Tat, BASIC hat auch dafür eine Möglichkeit, nämlich die logische Abfrage mit

IF ... THEN

oder in deutscher Menschensprache: "Wenn ... dann".

Allerdings können Sie nicht einfach programmieren: IF ich keine Lust mehr habe THEN hör gefälligst auf mit dem GOTO. Sie müssen zwischen IF und THEN schon etwas einschieben, was der Computer versteht. Ein Vergleichsausdruck ist da ideal. Dieser Vergleichsausdruck wird durch IF hochnotpeinlich auf "wahr" oder "falsch" geprüft. Nun steht hinter dem THEN eine Anweisung, die dann ausgeführt wird, wenn der Vergleichsausdruck wahr ist. Ist er falsch, geht der Computer sofort zu der Programmzeile über, die auf die IF ... THEN-Zeile folgt, ohne sich um das Statement hinter dem THEN zu kümmern.

Aber zunächst: Was ist ein Vergleichsausdruck? Ein Vergleichsausdruck ist ein Ding, in dem eine Variable mit einer Zahl oder mit einer anderen Variablen oder einem arithmetischen Ausdruck verglichen wird. Für diesen Vergleich braucht man Vergleichsoperatoren, von denen Sie einen schon oft benutzt haben: "=", was soviel heißt wie "gleich". Zum Beispiel haben Sie ihn benutzt, wenn Sie A=4 gesetzt haben. A=4 ist seinerseits ein Beispiel für einen Vergleichsausdruck. Natürlich gibt es noch mehr Vergleichsoperatoren mit verschiedenen Bedeutungen, wie Sie in Tabelle 4 sehen können:

Symbol	Bedeutung
=	gleich
>	größer als
<	kleiner als
>=	größer oder gleich
<=	kleiner oder gleich
<>	ungleich

Tab. 4: Vergleichsoperatoren und ihre Bedeutung

Selbstverständlich will ich Ihnen ein paar Beispiele für Vergleichsausdrükke nicht vorenthalten:

```
B>5
A<=C+D
F>=SIN(PI/4)*E
X<>Y
```

Jetzt wollen wir mal schauen, wie so ein logischer Ausdruck in einer IF ... THEN-Abfrage gebraucht wird. Dazu dieses Beispiel:

```
100 : INPUT B
110 : IF B>5 THEN PRINT "B>5 IST RICHTIG"
120 : END
```

Zeile 110 prüft nach, ob der Vergleichsausdruck B>5 wahr ist. Haben Sie für B z.B. den Wert 8 eingegeben, dann ist dies der Fall, also wird die Anweisung nach dem THEN (PRINT "B>5 IST RICHTIG") ausgeführt und Sie sehen

```
B>5 IST RICHTIG
```

auf dem Display. Erst dann geht das Programm mit Zeile 120 zu Ende. Hätten Sie aber beispielsweise eine 2 für B eingetippt, dann wäre der Vergleich B>5 falsch, die PRINT-Anweisung nach dem THEN würde nicht ausgeführt und das Programm würde durch Zeile 120 sofort beendet.

Statuieren wir gleich nochmal ein Beispiel, diesmal mit Sprüngen:

```
120 : INPUT C
130 : IF C>7 THEN GOTO 160
140 : PAUSE "C>7 IST FALSCH"
150 : GOTO 170
160 : PAUSE "C>7 IST RICHTIG"
170 : END
```

Ob der Vergleich C>7 stimmt, hängt wieder von Ihrer Eingabe ab. Für C = 8 z.B. wird die GOTO-Anweisung hinter dem THEN ausgeführt, das Programm verzweigt nach Zeile 160 und gibt den Text

```
C>7 IST RICHTIG
```

aus. Ist C>7 falsch, ignoriert der Computer die Anweisung hinter THEN und es geht mit Zeile 140 weiter, also erscheint

```
C>7 IST FALSCH
```

auf der Anzeige. Damit jetzt aber nicht doch noch fälschlicherweise C>7 IST RICHTIG ausgegeben wird, muß Zeile 160 übersprungen werden. Das geschieht in Zeile 150: GOTO 170. Sie können, wie Sie an diesem Beispiel sehen, gleich einem Känguruh kreuz und quer durch das Programm hüpfen.

Übrigens: Das GOTO nach dem THEN dürfen Sie auch fortlassen. Sie können Zeile 130 deshalb auch so schreiben:

```
130 : IF C>7 THEN 160
```

Kehren wir zu unserem Additionsprogramm zurück. Mit Hilfe der logischen Abfrage können Sie Ihr Programm auf elegante Art und Weise (sozusagen durch die Haustür) verlassen, indem Sie es durch die Zeilen 65 bis 67 so ergänzen:

```
10 : CLS
20 : WAIT 320
30 : INPUT "ERSTE ZAHL ";A
40 : INPUT "ZWEITE ZAHL ";B
50 : LET S = A+B
60 : PRINT "SUMME A+B = ";S
65 : PRINT "GEBEN SIE EINE 1 EIN, WENN"
66 : PRINT "DAS PROGRAMM STOPPEN SOLL"
67 : INPUT C
70 : IF C<>1 THEN GOTO 10
80 : END
```

Bis Zeile 60 läuft alles wie gewohnt ab, aber dann werden Sie in den Zeilen 65 und 66 dazu aufgefordert, eine 1 einzugeben, falls das Programm aufhören soll. Wollen Sie es weiterlaufen lassen, brauchen Sie bloß eine beliebige andere Zahl einzutippen. Zeile 70 springt nun nicht mehr automatisch nach Zeile 10 zurück wie das früher der Fall war, sondern nur, wenn sich der Vergleich C<>1 als wahr erweist. Haben Sie jedoch Lust verspürt, das Programm zu beenden und deshalb für C den Wert 1 gewählt, dann ist C<>1 falsch und die Anweisung GOTO 10 wird ignoriert. Das Programm geht sofort zu Zeile 80 : END über.

Ein GOTO, das an eine Bedingung geknüpft ist (in unsrem Beispiel wurde C<>1 vorausgesetzt), nennt man einen bedingten Sprung. Das bedeutet: nur für den Fall, daß die Bedingung erfüllt ist, wird gesprungen.

Wie Sie sehen, können wir mit Hilfe der IF ... THEN-Abfrage eine Endlosschleife in eine ganz "normale" Schleife verwandeln, also in eine Schleife, die unter bestimmten Voraussetzungen wieder verlassen werden kann.

Allerdings haben wir noch längst nicht alle Möglichkeiten der IF ... THEN-Abfragerei ausgeschöpft, beispielsweise ist dies hier machbar:

```
200 : IF X THEN PRINT "ALLES KLAR"
```

Das bedeutet: Sofern X ungleich Null ist, erblicken Sie den Schriftzug

```
ALLES KLAR
```

auf dem Display. Selbstverständlich dürfen Sie auch arithmetische Ausdrücke ins IF ... THEN stecken, wie Sie in den folgenden Zeilen sehen:

```
100 : INPUT "A EINGEBEN "; A
110 : INPUT "B EINGEBEN "; B
120 : IF A*B>10 THEN GOTO 200
```

wobei nach Zeile 200 nur dann verzweigt wird, wenn der Ausdruck A*B>10 ist.

Bisher haben wir hinter IF nur einen logischen Vergleich verwendet. Aber niemand hindert Sie daran, zwei oder mehr von diesen Ausdrücken ins Spiel zu bringen, falls Sie sie durch sogenannte " logische Operatoren" verknüpfen. Die wichtigsten von ihnen sind

AND

(auf deutsch "und") und

OR

("oder"). Dazu gleich ein Beispiel:

```
100 : IF A = 1 AND B>0 THEN GOTO 300
```

Hier wird der Befehl hinter THEN nur unter der Voraussetzung durchgeführt, daß beide Ausdrücke, also A = 1 und B>0 wahr sind. D.h., falls Sie zwei logische Ausdrücke Vergleich 1 und Vergleich 2 mit AND verknüpfen, dann müssen beide, Vergleich 1 und Vergleich 2, wahr sein, wenn Vergleich 1 AND Vergleich 2 auch stimmen soll. Ist auch nur einer der beiden Vergleiche falsch, so ist es auch der ganze Ausdruck. Anders ist es, wenn Sie mit OR verknüpfen, z.B. so:

```
100 : IF A = 1 OR B>0 THEN GOTO 300
```

Die Anweisung nach dem THEN (GOTO 300) wird ausgeführt, wenn:

1. Beide Vergleiche wahr sind oder
2. Nur einer von ihnen stimmt

Der Ausdruck Vergleich 1 OR Vergleich 2 liefert also auch dann noch das Ergebnis "wahr", wenn Vergleich 1 oder Vergleich 2 richtig ist. Allerdings: Sind beide Vergleiche falsch, dann ist es auch Vergleich 1 OR Vergleich 2. Treffen also in unserem Beispiel A = 1 und B>0 nicht zu, so wird GOTO 300 auch nicht in die Tat umgesetzt.

Bislang haben wir hinter THEN nur ein Statement gesetzt, so wurde z.B. durch die Zeile

```
110 : IF B>5 THEN PRINT "B>5 IST RICHTIG"
```

nur B>5 IST RICHTIG auf das Display gebracht, falls B>5 zutraf. Es besteht aber auch die Möglichkeit, mehrere Statements durchführen zu lassen, wenn ein Vergleichsausdruck vor dem THEN wahr ist. Um dies zu demonstrieren, flößen Sie mal Ihrem elektronischen Freund dieses Miniprogramm ein:

```
200 : INPUT P
210 : INPUT R
220 : IF P*R = 9 THEN LET K = 3: GOTO 240
230 : K = 5
240 : PRINT K
250 : END
```

Sollte er dann noch für P und R jeweils den Wert 3 erhalten, so wird er Ihnen zum Dank nicht nur K = 3 setzen, sondern auch einen Freudensprung nach Zeile 240 machen und eine 3 auf die Anzeige PRINTen. Sie können demnach hinter THEN mehrere Befehle ausführen lassen, nur müssen diese

1. Durch einen Doppelpunkt voneinander getrennt sein
2. In ein und derselben Zeile mit dem THEN stehen (im Beispiel also in Zeile 220), denn nur auf diese bezieht sich das THEN

Und noch etwas Wichtiges müssen Sie sich zu Herzen nehmen: Sie dürfen das LET in einer Zuweisung nicht weglassen,falls sie hinter einem THEN steht, programmieren Sie deshalb nicht ... THEN K = 3, sondern

... THEN LET K = 3

Das ist aber auch der einzige Fall, in welchem LET nicht unter den Tisch fallen sollte. In Zeile 250 ist wieder K = 5 statt LET K = 5 erlaubt. Einige Computer sind da allerdings etwas toleranter als z.B. der PC-1500 A. Die meckern nicht, wenn Sie das LET nach einem THEN vergessen.

Bisher bestimmte der Grad Ihrer Motivation, wie oft eine Schleife durchlaufen wurde. Sie können die Anzahl der Schleifendurchläufe aber auch vom Wert einer Variablen abhängen lassen. Angenommen, Sie wollen die Summe der Zahlen von 1 bis 20 bilden. Dieses bewerkstelligen Sie beispielsweise mit diesem Programm:

```
10 : CLS
15 : CLEAR
20 : LET A = A+1
30 : LET S = S+A
40 : IF A<20 THEN GOTO 20
50 : PRINT "SUMME VON 1 BIS 20: ";S
60 : END
```

Ich weiß, es sind wieder einmal ein paar erklärende Worte nötig. Nachdem Sie das Programm gestartet haben, werden durch die CLEAR-Anweisung in Zeile 15 zunächst alle Variablen gelöscht bzw. auf Null gesetzt. Das mag für den ersten Programmdurchlauf noch ohne Belang sein, da die Variablen dann ohnehin noch gleich Null sind. Das ändert sich jedoch in diesem Programm spätestens in der zweiten Runde, denn dann sitzen in den Speicherplätzen von A und S noch die Werte aus dem ersten Durchlauf. Werden diese durch den CLEAR-Befehl nicht gelöscht, so addieren Sie ungewollt zur neuen Summe noch den Wert der alten hinzu. Die CLEAR-Instruktion empfiehlt sich auch dann, wenn Sie mehrere Programme mit identischen Variablennamen im Speicher haben. Dabei kann es durchaus passieren, daß ein Programm einen Variablenwert aus einem anderen Programm aufschnappt und mit diesem Wert munter vor sich hinrechnet. Am Schluß bleibt Ihnen nur noch das Staunen ob der merkwürdigen Ergebnisse, die Ihr Rechner ausspuckt. Mit einem CLEAR passiert Ihnen das nicht.

Nachdem nun alles klar ist, wird in Zeile 20 A = A+1 gesetzt. Einem Mathematiker würde es dabei den Magen herumdrehen, nicht aber dem Computer. Sie wissen natürlich, warum: Die Zuweisung A = A+1 bedeutet lediglich, daß die Zahl A+1 in den Speicherplatz kommt, in welchem vorher A war. Oben habe ich es schon erwähnt: Beim ersten Durchlauf

eines Programms oder nach einem CLEAR-Befehl sind alle Variablen gleich Null. Das gilt auch für unser A, aber nur solange, bis die Anweisung "= A+1" erfolgt. Dann kommt auf den Speicherplatz von A statt der 0 eine 1. Das gleiche Spielchen wiederholt sich in Zeile 30: Auf dem Speicherplatz von S sitzt anfangs eine 0, die aber durch S+A einer 1 weichen muß. Zeile 40 fragt, ob A<20 ist, was für A = 1 zutrifft, also wird die Anweisung nach THEN, GOTO 20, sofort erledigt, das Programm landet wieder in Zeile 20. Abermals wird der Wert von A um 1 erhöht, so daß A = 2 wird. A = 2 wird zur Summe hinzugezählt, die damit auf S = 3 steigt. Die Gretchenfrage (A<20?) stellt sich wieder in Zeile 40. Auch für A = 2 kann sie bejaht werden, demnach steht dem Sprung nach Zeile 20 nichts mehr im Wege, usw., usw. bis A = 19 ist. Die Frage: A<20? wird zum letztenmal mit Ja beantwortet, noch einmal springt das Programm nach Zeile 20, aus A = 19 wird A = 20, dieses wird zur Summe S addiert. Und dann fragt das Programm in Zeile 40 nochmal: Ist A<20? Nein heißt es diesmal, daher gibt es keine Verzweigung nach Zeile 20 mehr, sondern das Programm geht zu Zeile 50 über und sogleich erblicken Sie

SUMME VON 1 BIS 20: 210

auf dem Display. Danach ist Schluß (Zeile 60). Die Schleife wurde genau zwanzigmal durchlaufen.

Einen Ausdruck wie A = A+1 nennt man einen "Zähler". Nun können Sie auch einen Zähler "hinunter-" statt "hinaufzählen", indem Sie z.B. alle Zahlen von -1 bis -20 addieren lassen. Dazu müssen Sie lediglich die Zeilen 20, 40 und 50 so abändern:

```
20 : LET A = A-1
40 : IF A>-20 THEN GOTO 20
50 : PRINT "SUMME VON -1 BIS -20: ";S
```

In Zeile 40 müssen Sie IF A>-20 programmieren, denn die Zahlen -1, -2, ... -19 sind alle größer als -20 und für diese muß die Schleife durchlaufen werden. Beim letzten Durchlauf wird A auf den Wert -20 gesetzt und zur Summe addiert. Testen Sie das Programm einmal! Es muß

SUMME VON -1 BIS -20: -210

herauskommen.

Ihnen ist sicher schon eines aufgefallen: Die IF ... THEN-Schleife ist nicht immer leicht zu handhaben. Gibt es denn keine bequemere Möglichkeit, eine Schleife in die Welt zu setzen? Klar gibt es die! Es handelt sich dabei um die

FOR ... NEXT-Schleife

Im allgemeinen sieht sie so aus:

```
100 : FOR Zählvariable = A TO E STEP Schrittweite
110 : Anweisung
```

```
120 : Anweisung
      :
      :
150 : Anweisung
160 : NEXT Zählvariable
```

Die Zähl- oder Schleifenvariable läuft vom Anfangswert A bis zum Endwert E. Bei jedem Schleifendurchlauf erhöht sie sich um die Schrittweite (die auch Inkrement genannt wird). So eine dicke Theoriepille schlucken Sie am besten mit einem Beispiel hinunter und zwar mit einem, das Sie schon kennen, nämlich die Addition der Zahlen von 1 bis 20. Mit Hilfe der FOR ... NEXT-Schleife wird diese Summe so berechnet:

```
10 : CLS
15 : CLEAR
20 : FOR A = 1 TO 20 STEP 1
30 : LET S = S+A
40 : NEXT A
50 : PRINT "SUMME VON 1 BIS 20: ";S
60 : END
```

Im Prinzip funktioniert sie genauso wie die IF ... THEN-Schleife. Zeile 20 schafft die Schleifenvariable A ("initiiert" heißt das im Computerdeutsch), die von 1 bis 20 laufen soll und zwar mit einer Schrittweite von 1, was durch STEP 1 kundgetan wird. Wie in der IF ... THEN-Schleife wird nun die Summe S = S+A gebildet. S ist zu Anfang also auch gleich Null, aber nicht lange, denn sofort kommt A = 1 hinzu (Zeile 30). Das Kommando NEXT A in Zeile 40 erhöht A um die Schrittweite, hier also um 1. Das geht so lange, bis A = 20 zu S addiert wird. Noch einmal erhöht NEXT A den Wert von A um 1, damit ist A = 21, d.h. der Endwert von 20 wurde überschritten, doch die Schleife wird nicht mehr durchlaufen. Sie können das leicht nachprüfen, indem Sie Zeile 55 einfügen:

```
55 : PRINT "A = ";A
```

Wenn Sie nach dem PRINT in Zeile 50 die ENTER-Taste drücken, steht tatsächlich A = 21 da. S hat natürlich den gleichen Wert wie im Programm mit der IF ... THEN-Schleife: 210. Übrigens rate ich Ihnen auch hier die Zeile 15 : CLEAR nicht zu vergessen, weil sonst bei jedem Programmdurchlauf der Wert der alten Summe zur neuen addiert wird.

Natürlich ist auch die FOR ... NEXT-Schleife nicht ohne Regeln, hier sind sie:

1. Wer FOR sagt, muß auch NEXT sagen, d.h. wurde eine Schleife mit FOR eröffnet, muß sie durch NEXT wieder geschlossen werden. Sowohl nach dem FOR als auch hinter dem NEXT muß der Name der gleichen Schleifenvariable stehen (und natürlich würden Sie auch nie auf die Idee kommen, den Anfangs- und den Endwert, sowie die Schrittweite der Schleife zu vergessen).
2. Ist die Schrittweite 1, können Sie die Anweisung STEP 1 getrost weglassen. Ihr Rechner versteht Sie in diesem Fall

auch so.

Zeile 20 aus dem vorherigen Programm hat nach dieser Regel folgende Gestalt:

```
20 : FOR A = 1 TO 20
```

Bei allen anderen Schrittweiten müssen Sie aber STEP S programmieren, wobei S die Schrittweite ist.

Und diese Regel wird Ihnen inzwischen bekannt vorkommen:

3. Für den Anfangs- und Endwert, sowie für die Schrittweite einer FOR ... NEXT-Schleife dürfen Sie Zahlen, numerische Variablen und arithmetische Ausdrücke verwenden.

Sie können also z.B. sowas programmieren:

```
10 : LET A = 10
20 : LET B = 30
30 : LET S = 2
40 : FOR I = A TO B+14 STEP 2*S
50 : PAUSE I
60 : NEXT I
70 : END
```

Dieses Programm gibt Ihnen vom Anfangswert A = 10 bis zum Endwert B+14 = 30+14 = 44 in Viererschritten (wegen 2*S = 2*2 = 4) eine Zahl aus, also 10, 14, 18, ... bis 42. Jawohl, nur bis 42, denn beim nächsten Schritt ist I = 46, und somit größer als der Endwert von 42.

4. Bei positiven Schrittweiten muß der Anfangswert einer Schleife kleiner als der Endwert sein. Umgekehrt muß bei negativen Schrittweiten (diese sind erlaubt!) der Anfangswert größer als der Endwert sein.

Beispiel mit positiver Schrittweite:

```
10 : FOR I = 10 TO 20 STEP 2
20 : PAUSE I
30 : NEXT I
```

Hätten Sie aber in Zeile 10 FOR I = 20 TO 10 STEP 2 stehen, könnten Sie die Ausführung der Schleife in den Wind schreiben.

Beispiel mit negativer Schrittweite:

```
10 : FOR I = 20 TO 10 STEP -1
20 : PAUSE I
30 : NEXT I
```

Vergessen Sie hier ja nicht, STEP -1 zu programmieren, nur für die Schrittweite +1 dürfen Sie es weglassen!

5. Die Schrittweite 0 ist nicht zulässig, denn Sie produzieren damit eine Endlosschleife – ein Computer wie der PC-1500 A wehrt sich dagegen mit einem ERROR 19 und auch andere Computer giften Sie mit einer Fehlermeldung an.
6. Sie dürfen mehrere FOR ... NEXT-Schleifen ineinanderschachteln (bis zu 16 beim PC-1500 A), nur geben Sie jeder Schleife ihre eigene Schleifenvariable (Sie würden z.B. Ihre Zahnbürste ja auch mit niemandem teilen wollen). Achten Sie auch darauf, daß sich die Schleifen nicht gegenseitig überschneiden. D.h.: Die zuerst geöffnete Schleife wird zuletzt geschlossen, die zuletzt geöffnete Schleife wird zuerst wieder geschlossen.

So z.B. programmieren Sie richtig:

```
10 : FOR I = 1 TO 10
20 : FOR J 1 TO 200
30 : NEXT J
40 : PAUSE I
50 : NEXT I
60 : END
```

Nebenbei bemerkt: Bei den Zeilen 20 und 30 handelt es sich nicht um einen Scherzartikel, sondern um eine Zeitschleife. Der Computer macht dabei nichts weiter, als von 1 bis 200 zählen und verbrät dadurch natürlich etwas Zeit, während der der Programmablauf verzögert wird. Solche Zeitschleifen sind ein beliebtes Mittel, um beispielsweise eine Textausgabe zum Lesen ein Weilchen auf der Anzeige anzuhalten, bevor im Programm fortgefahren wird. So werden Sie im obigen Beispiel rasch feststellen, daß die Ausgabe der I-Werte deutlich länger dauert als sonst beim PAUSE üblich.

Ein Exempel für eine falsch verschachtelte Schleife sollen Sie jetzt auch noch haben:

```
10 : FOR I = 1 TO 10
20 : FOR J = 1 TO 200
30 : PAUSE I
40 : NEXT I
50 : NEXT J
60 : END
```

Sollten Sie diese kleine Routine starten, so wird ein Rechner wie der PC-1500 A noch die Freundlichkeit besitzen, Ihnen die I's von 1 bis 10 auf dem Display zu zeigen, aber Sie bemerken zunächst, daß die Zeitschleife in Zeile 20 nicht mehr zur Wirkung kommt (NEXT J wird nicht mehr erreicht), die I-Werte erscheinen also wieder im 0.85 Sekundenabstand, so wie Sie es beim PAUSE-Befehl gewohnt sind. Danach aber wird Sie Ihr Computer mit einer ERROR-Anzeige frusten, nämlich dann, wenn er in Zeile 50 auf die Instruktion NEXT J stößt und dabei registriert, daß Sie die beiden Schleifen ineinander verknotet haben.

7. Machen Sie sich und Ihren Computer nicht unglücklich, indem Sie mit einem Programmsprung (z.B. mit GOTO) in eine FOR ... NEXT-Schleife hüpfen.

Dieses kleine Programm soll Ihnen als abschreckendes Beispiel dienen:

```
80 : GOTO 110
90 : PAUSE "ICH HABE SIE GEWARNT"
100 : FOR I = 1 TO 10
110 : PAUSE "BLEIBEN SIE DRAUSSEN!"
120 : NEXT I
```

Wenn man schon eine Schleife von außen anspringt, dann sollte man es nur über die FOR ... TO-Zeile tun, z.B. so:

```
80 : GOTO 100
90 : PAUSE "DAS DUERFEN SIE"
100 : FOR I = 1 TO 10
110 : PAUSE "SO IST'S RECHT"
120 : NEXT I
```

Es ist auch erlaubt, aus einer Schleife hinauszuspringen, wie diese Zeilen zeigen:

```
200 : FOR K = 1 TO 5
210 : PAUSE K
220 : IF K = 3 THEN GOTO 240
230 : NEXT K
240 : PRINT "DIESER SPRUNG IST ERLAUBT"
```

8. In einer FOR ... NEXT- Schleife dürfen alle möglichen BASIC-Befehle stehen, z.B. PRINT, PAUSE, INPUT, Funktionennamen usw.

Beispiel:

```
100 : FOR I = 1 TO 10
110 : INPUT "EINE ZAHL BITTE";Z
120 : LET K = Z↑3
130 : PAUSE "Z↑3 = ";K
140 : NEXT I
```

An diese Regeln halten sich alle Computer, jedoch gibt es noch ein paar Gesetze, die von Rechner zu Rechner recht individuell gehandhabt werden, ich will sie Ihnen nicht vorenthalten:

9. Auch Schleifenvariablen haben ihre Grenzen, so müssen sie beim PC-1500 A Werte zwischen -32768 und +32767 annehmen. Werden diese Grenzen überschritten, setzt es eine ERROR-Meldung. Bitte schauen Sie in der Gebrauchsanleitung nach, wie weit Sie bei Ihrem Computer gehen dürfen.
10. Sie können auch Kommazahlen für die Anfangs- und End-

werte und die Schrittweite wählen.

Beispiel:

```
100 : FOR Z = 1.5 TO 4.5 STEP 0.5
110 : PAUSE Z
120 : NEXT Z
```

Manche Computer schlucken das nicht nur, sondern führen dies auch exakt aus, so daß Sie die Zahlen 1.5, 2.0, 2.5, 3.0, 3.5, 4.0 und 4.5 auf der Anzeige erblicken. Aber: Ein Taschencomputer, wie z.B. der PC-1500 A, verwendet von diesen Ausdrücken nur den ganzzahligen Anteil, mit der üblen Konsequenz, daß die Schrittweite gleich 0 wird. Für ihn ist das ein willkommener Anlaß, Ihnen mal wieder eine Fehlermeldung zu präsentieren (ERROR 19 IN 100). Er wird Zeile 100 jedoch akzeptieren, wenn Sie sie so programmieren:

```
100 : FOR Z = 1.5 TO 4.5 STEP 1.5
```

Es werden der Reihe nach die Zahlen 1.5, 2.5 und 3.5 auf dem Display erscheinen. Sie sehen: Er macht Einserschritte, statt solche mit 1.5. Die Zahl 4.5 gönnt er Ihnen nicht, denn bei der ganzzahligen 4 und nicht bei 4.5 ist für ihn schon Feierabend.

Nach dieser Berieselung wollen Sie sicher wieder kreativ tätig werden. Die nächsten Übungsaufgaben warten schon auf Ihre Geistesblitze.

Aufgabe 7.1

Sicher kennen Sie noch das Programm, welches Ihnen die Zahlen von 1 bis 20 mit Hilfe einer IF ... THEN-Schleife addiert. Die untere Grenze dieser Zahlenreihe war 1, die obere war 20. Schreiben Sie nun das Programm so um, daß alle Zahlen zwischen einer beliebigen unteren Grenze U und einer oberen Grenze O addiert werden. Eine IF ... THEN-Abfrage soll testen, ob die obere Grenze schon erreicht ist und für diesen Fall die Addition abbrechen, d.h. die obere Grenze soll die letzte Zahl sein, die addiert wird. Bitte vergessen Sie auch nicht, die Summe mit Begleittext auf dem Display erscheinen zu lassen.

Aufgabe 7.2

Schreiben Sie bitte das Programm aus Aufgabe 7.2 nochmal, jedoch mit einer FOR ... NEXT- statt einer IF ... THEN-Schleife. Eines kommt aber noch hinzu: Lassen Sie diesmal auch die Schrittweite variieren, d.h. sorgen Sie dafür, daß eine beliebige Schrittweite eingelesen und in der Schleife verwendet werden kann.

Aufgabe 7.3

Welche Zahlen werden für A und B durch das folgende Programm auf der Anzeige ausgegeben? Bitte beantworten Sie diese Frage, ohne vorher das Programm in den Rechner zu tippen.

```
10 : CLS
15 : CLEAR
20 : WAIT 200
30 : A = 14:B = 9
40 : A = A+4:B = B+3
50 : PRINT "B = ";B
60 : IF A<30 THEN 40
70 : PRINT "A = ";A
80 : END
```

Aufgabe 7.4

Sicher haben Sie irgendwann einmal die Zinsrechnung in der Schule durchgenommen. Dann ist Ihnen diese Formel schon begegnet:

$$K = A*(1+P/100)^N$$

Dabei ist A das Anfangskapital, welches Sie auf die Sparkasse getragen haben, in DM, P der Ihnen gewährte Zinssatz in % und K das Endkapital in DM nach N Jahren Sparkassenaufenthalt. Schreiben Sie bitte ein Programm, das Ihnen A, P und N einliest und das Endkapital K dafür ausgibt. Gestalten Sie Ihr Programm mit Hilfe der IF ... THEN-Abfrage so, daß es nur Zinssätze von P>= 2% und P<= 10% akzeptiert. Wird ein Wert eingegeben, der außerhalb dieser Grenzen liegt, soll das Programm den Benutzer darauf aufmerksam machen und zu der Zeile zurückspringen, in der P eingelesen wird, damit ein neuer Wert für den Zinssatz eingegeben werden kann.

8. Kapitel

VOM PROBLEM ZUM PROGRAMM
- Wie Sie systematisch programmieren -

Hatten wir nicht schon Probleme genug? Und es gab doch soviel Programme, wie Probleme da waren. Was, werden Sie sich fragen, soll jetzt dieses Kapitel? Sicher, programmiert haben Sie schon viel, aber nicht mit System. Bisher lief doch das immer so ab: Wir übersetzten unsere Probleme direkt ins BASIC und trichterten Sie dem Computer ein. Bei kleinen Programmen ist dies auch nicht weiter schlimm, bei großen verlieren Sie aber auf diese Weise rasch den Überblick. Eines haben Sie sicher schon mitbekommen: Man muß dem Computer die Lösung eines Problems Schritt für Schritt beibringen. So mußten Sie sich erst darüber klar werden, was für Variablen Sie brauchten, diese mußten dem Computer eingegeben werden. Dann kam der Teil des Programms, in welchem diese Variablen verarbeitet wurden. Und sogar die Ergebnisausgabe mußten Sie Ihrem Computer noch befehlen, sonst hätte er Ihnen zwar etwas ausgerechnet, das Resultat aber fein still für sich behalten. Wie Sie sehen, war es Ihre Aufgabe, einen Lösungsweg für Ihr Programm zu finden. Diesen "Lösungsweg für ein Programm" nennt man **Algorithmus.**

Sie haben ihn bisher mehr oder weniger bewußt und nur so nebenbei ausgeführt, jetzt soll er zum festen Bestandteil Ihrer Programmierkunst werden. Fangen wir mit einer ganz einfachen Übung an, der Addition zweier Zahlen. Wie gehen Sie nach dem systematischen Weg vor?

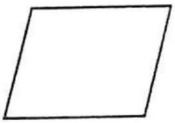
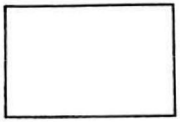
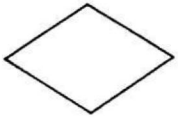
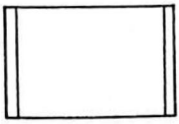
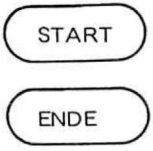
1. Sie erarbeiten eine Problemstellung und analysieren das Problem.

In unserem Beispiel gibts nicht viel zu analysieren. Zwei beliebige Zahlen sollen eingegeben, addiert und das Ergebnis auf der Anzeige ausgegeben werden, fertig!

2. Sie entwickeln eine Lösungsstrategie, den Algorithmus. Ein bewährtes Hilfsmittel hierzu ist der **Programmablaufplan,** kurz **PAP** genannt.

Hier sagt der Name schon alles: Dieser Plan zeigt Ihnen genau, wie ein Programm abläuft, was Sie aus einem ausgelisteten Programm, dem **Listing,** nicht so ohne weiteres erkennen können. Für solche Programmablaufpläne gibt es besondere Symbole (den DIN-Norm-Fetischisten unter Ihnen wird es eine Freude bereiten, zu wissen, daß diese Symbole nach **DIN 66001** genorm sind). Die wichtigsten von ihnen finden Sie in Tab. 3.

Zeichnen Sie Programmablaufpläne immer von oben nach unten (aus diesem Grund können Sie die Richtungspfeile weglassen, das gilt auch für Ablaufrichtungen von links nach rechts). Immer wenn Sie von dieser Richtung abweichen, verwenden Sie die sogenannten Konnektoren. Mit Hilfe von Zahlen oder Buchstaben in diesen Konnektoren werden z.B. Anfang und Ende eines Sprungs markiert. Sie können sie natürlich auch dann

SYMBOL	BEDEUTUNG	BASIC
	Ein- oder Ausgabe	INPUT PRINT PAUSE
	Ausgabe auf Drucker	LPRINT
	Programmverarbeitung	verschie- den
	logische Verzweigung	IF ... THEN FOR ... NEXT
	Konnektor	
	Unterprogrammaufruf	GOSUB RETURN
	Ablauflinien	
	Grenzstellen, markieren Anfang und Ende eines Programms	

Tab. 3: Symbole des Programmablaufplans

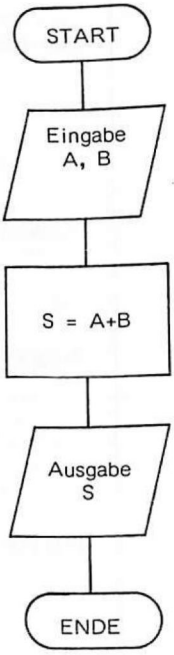


Abb. 3: PAP für ein Additionsprogramm

benutzen, wenn Ihnen einfach der Platz auf dem Papier nicht gereicht hat und Sie auf der nächsten Seite fortsetzen wollen.

Für die Addition zweier beliebiger Zahlen ist der Lösungsweg so:

a) Eingabe zweier Zahlen A und B
b) Addition von A und B. Ergibt die Summe S
c) Ausgabe der Summe S auf der Anzeige

Den Programmablaufplan dazu sehen Sie in Abb. 3.

3. Jetzt erst programmieren Sie, d.h. Sie übersetzen das PAP ins BASIC.

Den PAP aus Abb. 3 ins BASIC zu übersetzen, bereitet keine Schwierigkeiten:

```
10 : CLS
20 : INPUT "ERSTE ZAHL";A
```

```
30 : INPUT "ZWEITE ZAHL";A
40 : LET S = A+B
50 : PAUSE "SUMME S = ";S
60 : END
```

So ein PAP hat den Vorteil, daß er für alle Programmiersprachen der gleiche bleibt, egal, ob Sie in BASIC, FORTRAN oder einer anderen Sprache programmieren.

4. Testen Sie Ihr Programm zuerst im Kopf und mit einfachen Zahlen, so daß Sie die Rechnung noch nachvollziehen können. Dann starten Sie einen Probelauf auf dem Computer.
5. Wollen Sie ganz perfekt sein, dann schreiben Sie für Ihr Programm noch eine Dokumentation.

Eine Dokumentation ist eine Gebrauchsanleitung oder Funktionsbeschreibung für das Programm. Sie ist besonders dann nützlich, wenn andere mit Ihrem Programm arbeiten oder Sie selbst nach einem halben Jahr vergessen haben, was Sie da eigentlich programmiert haben. Für einfache Routinen wie dieses Additionsprogramm kann sie unter den Tisch fallen. Da reichen entsprechende Kommentare im Programm zur Dokumentation völlig aus.

Sollten Sie jetzt einwenden, daß, wer so programmiert, selbst aus Nasenbohren noch eine Doktorarbeit macht, gebe ich Ihnen in diesem Fall noch recht, biete Ihnen aber ein anderes Beispiel, nämlich die Lösung einer quadratischen Gleichung. Da werden Sie die Vorteile des sytematischen Programmierens rasch einsehen.

Das fängt schon beim ersten Punkt an, denn: Erarbeiten Sie doch mal eine Problemstellung und analysieren Sie das Problem dann. Wie sieht denn so eine quadratische Gleichung überhaupt aus? Gut, eine mathematische Formelsammlung hilft Ihnen hier geschwinde aus der Klemme, die Formel für eine quadratische Gleichung lautet:

$$AX^2 + BX + C = 0$$

Diese Gleichung hat in der Regel zwei Lösungen, d.h. es gibt zwei Werte für X, die diese Gleichung erfüllen. Nun ist so eine Formelsammlung ein ganz praktisches Buch, das die Lösungen gleich mitliefert:

1. Lösung: $X1 = \frac{-B+\sqrt{B^2-4AC}}{2A}$

2. Lösung: $X2 = \frac{-B-\sqrt{B^2-4AC}}{2A}$

Jetzt brauchen wir nur noch die Formeln mit etwas Kommentar in den Computer zu prügeln? Oh nein! Die Filigranarbeit der Problemanalyse beginnt erst.

Daß zunächst einmal die drei Variablen A, B und C eingelesen werden müssen, ist jedem klar, aber dann taucht schon die erste Schwierigkeit auf: Einige SHARP-Taschencomputer können die Variablen X1 und X2

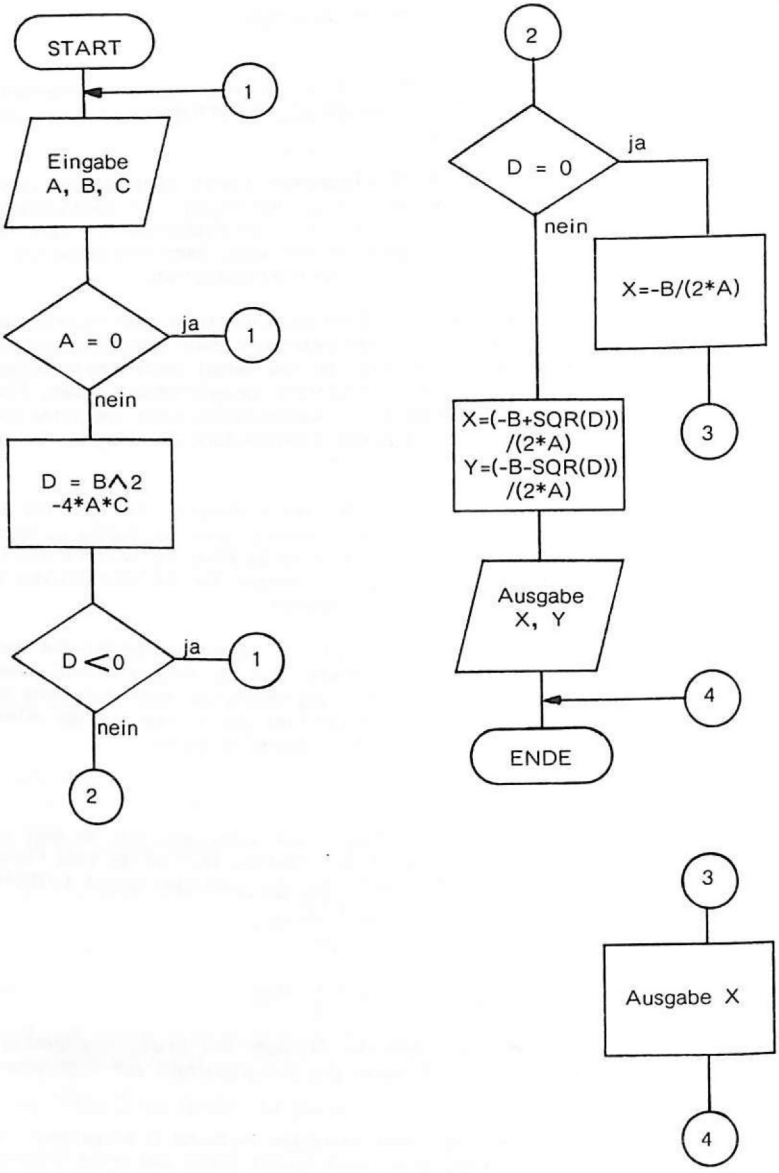


Abb. 4: PAP für Programm zur Lösung einer Quadratischen Gleichung

nicht voneinander unterscheiden, da sie nur das erste Zeichen einer Variablen registrieren. Also taufen Sie die Variablen um. Statt X1 schreiben Sie X und aus X2 wird Y.

Der zweite Haken steckt unter der Wurzel. Aus dem Kapitel über mathematische Funktionen wissen Sie sicher noch, daß das Argument in der Wurzel nicht negativ sein darf, also muß $B^2-4AC>=0$ sein. Ihr Programm darf deshalb nur dann weiterrechnen, wenn diese Bedingung erfüllt ist, sonst muß es zurückspringen und neue Werte für A, B und C anfordern (nach entsprechendem Kommentar für den Benutzer versteht sich).

Dann kann es Ihnen passieren, daß $B^2-4AC = 0$ wird und Sie nur die eine Lösung

$$X = Y = \frac{-B}{2A}$$

erhalten. Auch das müssen Sie in Ihrem Programm vorsehen.

Der letzte Sonderfall entsteht, wenn A = 0 wird. Dann geht es im Computer heiß her, weil durch Null nicht dividiert werden darf. Auch für diesen Fall muß das Programm vorsorgen und neue Zahlen für A, B und C verlangen.

Nun zu den rechentechnischen Dingen. Damit Sie nicht bei jedem Aufruf von B^2-4AC diesen langen Ausdruck von neuem in den Rechner tippen müssen, setzen Sie ihn auf eine Variable D: $D = B^2-4AC$ und weil ihn sicher auch der Computer kein einziges Mal umsonst berechnen will, stellen Sie lieber vorher die Frage, ob A = 0 ist und sich somit der Rest des Programms erübrigt. Für den Fall, daß alles glatt über die Bühne geht, sollten die Lösungen X und Y auch berechnet und ausgegeben werden. Na, das sah schon eher nach Problemanalyse aus, nicht wahr?

Den Programmablaufplan (er gehört zu Punkt 2 in unsrem Programmiersystem) können Sie in Abb. 4 betrachten. Ein paar erklärende Worte dazu sollen auch nicht fehlen. Gleich nach der Eingabe von A, B und C wird abgefragt, ob A = 0 ist. Lautet die Antwort "ja", dann zeigt der Konnektor mit der 1 rechts neben der Abfrage an, daß zum Anfang des Programms (ebenfalls durch einen Konnektor mit einer 1 gekennzeichnet) zurückgesprungen wird. Dann sind neue Werte für A, B und C fällig. Ist die Antwort "nein", setzt das Programm mit D = B^2-4*A*C fort. Wie Sie sehen, wurde die mathematische Formel gleich ins BASIC übersetzt. Die nächste Frage lautet Ist D<0? Wenn ja, wird wieder zum Anfang des Programms zurückgesprungen (wieder erkennen Sie das am Konnektor mit der 1). Im Falle eines Nein geht das Programm zur nächsten Abfrage über: Ist D = 0? Falls ja, dann gibt es nur eine Lösung X, diese soll berechnet und ausgegeben werden. Konnektor 3 führt uns zielsicher zu dieser Ausgabe und nachdem diese erledigt ist, wird zum Programmende gesprungen (Konnektor 4 ist der Wegweiser dorthin). Ist D nicht gleich Null und auch nicht kleiner Null, bleibt nur noch D>0 übrig. Die quadratische Gleichung hat dann zwei Lösungen X und Y, die der Computer auch brav berrechnet und auf dem Display zeigt.

Danach kommt Punkt 3, das für Sie und den Computer wohlverdiente Programm. Wir schreiben das Programm auf Papier auf, indem wir das PAP in BASIC übersetzen:

```
 5 : REM QUADRATISCHE GLEICHUNG
10 : CLS
15 : CLEAR
20 : WAIT 200
30 : INPUT "WERT FUER A ";A
40 : INPUT "WERT FUER B ";B
50 : INPUT "WERT FUER C ";C
60 : IF A = 0 THEN PRINT "A MUSS<>0 SEIN" : GOTO 30
70 : LET D = B^2-4*A*C
80 : IF D<0 THEN PRINT " D MUSS> = 0 SEIN" : GOTO 30
90 : IF D = 0 THEN LET X = -B/(2*A) : PRINT " EINZIGE
     LOESUNG X = ";X : GOTO 140
100 : LET X = (-B+SQR(D))/(2*A)
110 : LET Y = (-B-SQR(D))/(2*A)
120 : PRINT "ERSTE LOESUNG X = ";X
130 : PRINT "ZWEITE LOESUNG Y = ";Y
140 : END
```

Der Befehl

REM

in Zeile 5 ist für Sie noch neu. Er ist die Abkürzung für **REM**ark, was soviel wie "Bemerkung" bedeutet. Die REM-Anweisung und die restliche Zeile dahinter ist für den Computer Luft, d.h. er führt sie niemals aus. Sie dient lediglich dazu, das Programm mit zusätzlichen Kommentaren zu erklären. Sie haben schon Kommentare mit PRINT, PAUSE und INPUT kennengelernt, aber während diese Kommentare innerhalb des Programmablaufs auf der Anzeige erscheinen, finden Sie einen Kommentar mit REM nur beim Auflisten des Programms, nicht aber, wenn das Programm ausgeführt wird. Selbst eine Anweisung wie z.B.

```
10 : REM SETZE A GLEICH 1 : LET A = 1
```

wird nie erledigt, d.h. A wird nicht 1 gesetzt, obwohl die Anweisung LET A = 1 durch einen Doppelpunkt von REM SETZE A GLEICH 1 getrennt ist. Anders ist es, wenn Sie die Zeile so formulieren:

```
10 : LET A = 1 : REM SETZE A GLEICH 1
```

Hier erhält A tatsächlich den Wert 1, weil LET A = 1 vor der REM-Anweisung steht. REM kann an jeder beliebigen Stelle des Programms stehen.

Auch zum INPUT gibt es was zu bemerken. Sie haben in den Zeilen 30-50 für jede der Variablen A, B und C ein INPUT verbraucht. Dasselbe können Sie auch mit einem INPUT erledigen. Probieren wir es doch gleich aus und programmieren Sie:

```
30 : INPUT "WERTE FUER A, B, C ";A,B,C
```

statt der Zeilen 30-50. Sie müssen lediglich die Variablennamen A, B und C durch Kommata voneinander trennen. Nach dem Starten des Programms geben Sie für A einen Wert (z.B. 1) ein und schicken ihn, wie sonst auch, mit ENTER ab. Danach taucht ein Fragezeichen auf dem Display auf, der Wert von B wird verlangt. Auch ihn geben Sie ein, verabschieden ihn mit ENTER und wenn Sie desgleichen mit C gemacht haben, kann Ihr Computer loslegen.

Haben Sie das vollständige Programm schon im Rechner? Gut, dann können wir mit Punkt 4, Testen des Programms, fortfahren. Wählen Sie dazu folgende Werte für A, B und C:

a) A = 0, B = 1, C = 1

Wetten, daß ihr Computer A = 0 moniert und Sie zur Eingabe von neuen Werten für A, B, C auffordert? Bedienen Sie Ihn dann mit

b) A = 1, B = 1, C = 1

Diesmal wird D = B∧2-4*A*C kleiner 0, Ihr Computer wird auch das kritisieren und wieder neue Zahlen für A, B und C fordern. Besänftigen Sie Ihn mit

c) A = 1, B = 2, C = 1

Dadurch wird D = 0 und Sie erhalten die einzige Lösung X = -1. Starten Sie das Programm erneut und geben Sie

d) A = 1, B = 6, C = 5

ein. Sie werden nacheinander die zwei Lösungen X = -1 und Y = -5 auf der Anzeige sehen.

Den Punkt 5, die Dokumentation des Programms, können Sie durch den obigen Text als bereits erledigt betrachten. Noch nicht erledigt sind allerdings die Übungsaufgaben, die jetzt auf Sie zukommen.

Aufgabe 8.1:

Abb. 5: PAP für das Programm ZINSRECHNUNG

Wenn Sie die Abb. 5 hier vergeblich suchen, hat das seinen guten Grund. Den Programmablaufplan zu Aufgabe 7.4 sollen nämlich Sie selbst zeichen. Machen Sie zudem eine Problemanalyse zu dieser Aufgabe. Sollte es mit dem Zeichnen nicht so recht gelingen, ist Abb. 5 im Anhang zu finden.

Aufgabe 8.2:

Schreiben Sie bitte ein Programm, das Ihnen die Summe S einer geometrischen Reihe berechnet. Unter einer geometri-

schen Reihe versteht man eine Folge von Zahlen, deren Glieder stetig um einen Faktor Q zunehmen (z.B. sich immer verdoppeln). Ist also das erste Glied A, dann wird das zweite Glied A*Q, das dritte $A*Q^2$ ... das n-te Glied wird $A*Q^{n-1}$. Die Summe ist

$$S = A+A*Q+A*Q^2 \ldots +A*Q^{n-1}$$

Ein Blick in die Mathematische Formelsammlung genügt, um Ihnen zu zeigen, daß sich S auch so berechnen läßt:

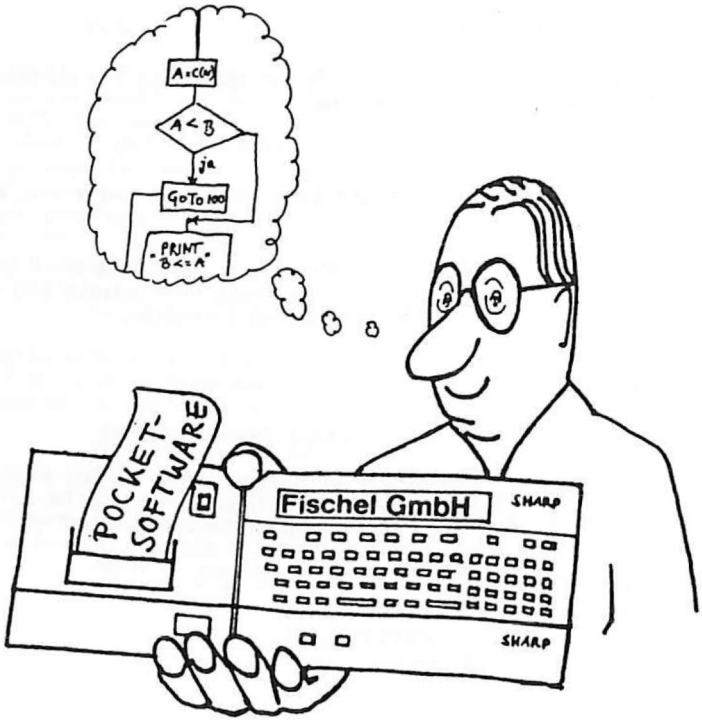
$$S = \frac{A*(1-Q^n)}{1-Q} \quad , \text{ wenn } Q<>1$$

und $S = n*A$, wenn $Q = 1$

Das Programm soll die Summe für beide Fälle berechnen und ausgeben. Lösen Sie diese Aufgabe nach unserem Schema: 1. Problemanalyse, 2. Programmablaufplan, 3. Programm schreiben und 4. Programm testen. Probieren Sie es mit diesen Werten:

a) A = 1, Q = 1, N = 10
b) A = 1, Q = 2, N = 10

Die Lösung mit PAP (Abb. 6) finden Sie wie üblich im Anhang.



9. Kapitel

AUS GUTEM GRUND IM UNTERGRUND
- Unterprogramme -

Stellen Sie sich einmal vor, Sie wollten ein Programm schreiben, welches Ihnen zwei Zahlen einliest und dann die vier Grundrechenarten (+, -, *, /) an ihnen exerziert. Sie programmieren also munter:

```
10 : REM VIER GRUNDRECHENARTEN
20 : CLS : CLEAR
30 : WAIT 200
40 : INPUT "ERSTE ZAHL ";A
50 : INPUT "ZWEITE ZAHL ";B
60 : LET C = A+B
70 : PRINT "ERG. ADD.: ";C
80 : LET D = A-B
90 : PRINT "ERG. SUB.: ";D
100 : LET E = A*B
110 : PRINT " ERG. MUL.: ";E
120 : LET F = A/B
130 : PRINT "ERG. DIV.: ";F
140 : END
```

Ausgezeichnet, Sie haben Ihre vier Grundrechenarten, aber gleich alle vier auf einmal, obwohl Sie vielleicht nur eine Multiplikation wollten. Ist das der Sinn der Sache? Die Antwort liegt auf der Hand - nein! Also muß ein Programm her, mit dem Sie sich eine Grundrechenart aussuchen können. Dieses "Aussuchen" geschieht so:

```
245 : PRINT "BITTE WAEHLEN SIE"
250 : PRINT "1 FUER ADDITION"
260 : PRINT "2 FUER SUBTRAKTION"
270 : PRINT "3 FUER MULTIPLIKATION"
280 : PRINT "4 FUER DIVISION"
290 : INPUT X
300 : IF X = 1 THEN ...
310 : IF X = 2 THEN ...
320 : IF X = 3 THEN ...
330 : IF X = 4 THEN ...
```

Im Klartext heißt das: Sie sollen eine 1 eintippen, falls Ihnen nach Addition zumute ist, wenn Sie die Subtraktion favorisieren, tippen Sie auf die 2 usw. (Zeile 250-280). Diese Zahl merkt sich der Computer mit Hilfe der Variablen X (Zeile 290). Die Zeilen 300-330 fragen den Wert von X ab. Sofern also X = 1 ist, THEN ... ja dann soll doch die Addition ausgeführt werden:

```
50 : LET C = A+B
60 : PRINT "ERG. ADD.: ";C
```

Sieht das Ganze nicht wie ein schnuckeliges Programm aus? In der Tat! Und weil es ein kleines Programm innerhalb eines größeren Programms (hier die vier Grundrechenarten) ist, wird es **Unterprogramm** genannt.

Wie aber bedient man sich seiner? Antwort: Indem man es zu Hilfe ruft. Der BASIC-Hilferuf nach einem Unterprogramm lautet:

```
GOSUB
```

GOSUB ist die Abkürzung für **GO**to **SUB**routine, auf deutsch: "Gehe ins Unterprogramm". Direkt hinter den GOSUB-Befehl wird die Zeilennummer gesetzt, mit der das Unterprogramm anfängt. Nun können Sie die Abfrage IF X = 1 THEN ... so beantworten:

```
300 : IF X = 1 THEN GOSUB 50
```

und rufen dadurch das Additionsunterprogramm auf:

```
50 : LET C = A+B
60 : PRINT "ERG. ADD.: ";C
70 : RETURN
```

Es funktioniert wie ein selbstständiges Programm, d.h. Zeile 50 berechnet C = A+B, Zeile 60 teilt Ihnen das Ergebnis auf der Anzeige mit. Der Befehl In Zeile 70

```
RETURN
```

ist Ihnen neu. Er ist sehr wichtig, denn er sagt dem Programm, daß es das Wiederkommen nicht vergessen soll (return = kehre zurück). Sobald das Programm auf ein RETURN stößt, springt es zur Instruktion zurück, die dem GOSUB folgt. In unserem Beispiel ist das die Zeile 310. Vergessen Sie deshalb niemals, Ihre Subroutinen mit RETURN abzuschließen, sonst gibt es für Ihren Computer keine Wiederkehr.

Gut, wir sind heil auf Zeile 310 gelandet. Sie lautet:

```
310 : IF X = 2 THEN GOSUB 90
```

Sie ahnen es schon, mit GOSUB 90 wird das zweite Unterprogramm heraufbeschworen:

```
 90 : LET D = A-B
100 : PRINT "ERG. SUB.: ";D
110 : RETURN
```

Genau so können wir mit der Multiplikation und der Division verfahren:

```
130 : LET E = A*B
140 : PRINT "ERG. MUL.: ";E
150 : RETURN
```

und

```
170 : LET F = A/B
180 : PRINT " ERG. DIV.: ";F
190 : RETURN
```

Schön, die Unterprogramme für die Grundrechenarten stehen jetzt. Aber wo bringen wir Sie in unserem Programm unter? Es gibt Programmierer, die ihre Unterprogramme an das Ende des Programms stecken und solche, die sie lieber am Anfang sehen. Bevor Sie deshalb einen Glaubenskrieg vom Zaun brechen, die kleinen Zeilennummern deuten es auch schon an: Unterprogramme sind am besten am Anfang eines Programms aufgehoben und zwar deshalb, weil die meisten BASIC-Versionen so beschaffen sind, daß der Computer nach jedem Subroutinenaufruf an den Beginn des Programms springt und von dort aus nach dem gewünschten Unterprogramm sucht. Die Suche ist um so schneller beendet, je eher der Computer auf das betreffende Unterprogramm stößt.

Damit Sie den Überblick nicht verlieren, hier das gesamte Listing:

```
 10 : REM VIER GRUNDRECHENARTEN
 20 : CLS : CLEAR
 30 : WAIT 200
 35 : GOTO 230
 40 : REM ADDITION
 50 : LET C = A+B
 60 : PRINT "ERG. ADD.: ";C
 70 : RETURN
 80 : REM SUBTRAKTION
 90 : LET D = A-B
100 : PRINT "ERG. SUB.: ";D
110 : RETURN
120 : REM MULTIPLIKATION
130 : LET E = A*B
140 : PRINT "ERG, MUL.: ";E
150 : RETURN
160 : REM DIVISION
170 : LET F = A/B
180 : PRINT "ERG. DIV.: ";F
190 : RETURN
220 : REM HAUPTPROGRAMM
230 : INPUT "ERSTE ZAHL ";A
240 : INPUT "ZWEITE ZAHL ";B
250 : PRINT "BITTE WAEHLEN SIE"
260 : PRINT "1 FUER ADDITION"
270 : PRINT "2 FUER SUBTRAKTION"
280 : PRINT "3 FUER MULTIPLIKATION"
290 : PRINT "4 FUER DIVISION"
300 : INPUT X
310 : IF X = 1 THEN GOSUB 50
320 : IF X = 2 THEN GOSUB 90
330 : IF X = 3 THEN GOSUB 130
340 : IF X = 4 THEN GOSUB 170
350 : END
```

Zu diesem Programm bin ich Ihnen noch einige Erklärungen schuldig. So wird in Zeile 35 mit GOTO 230 über die Subroutinen hinweg zunächst ins Hauptprogramm gesprungen. Das geschieht deshalb, weil ein Unterprogramm erst dann dran ist, wenn es durch eine GOSUB-Anweisung angesprungen wird. Rauschen Sie ohne diesen Befehl in eine Subroutine hinein, stößt Ihr Computer auf einen RETURN-Befehl. Nun soll aber RETURN das Programm immer zu der Instruktion zurückspringen lassen, die dem dazugehörigen GOSUB folgt. Da es jedoch vorher kein GOSUB gab, weiß auch der Computer nicht, wohin er zurückspringen soll. Und was ein Computer nicht weiß, macht ihn nicht heiß - eiskalt serviert er Ihnen eine Fehlermeldung.

Gut, Sie befinden sich jetzt im Hauptprogramm (Zeile 230). Was unser elektronischer Adam Riese dort tut, wissen Sie bereits: Er sucht sich, nachdem er von Ihnen zwei Werte A und B bekommen hat, das gewünschte Unterprogramm, z.B. die Addition. Ist er von dort mit RETURN zurückgeschickt worden, arbeitet er sinnloserweise die restlichen IF-Abfragen zuende. Zugegeben, eine recht holperige Art zu programmieren, aber ich kann Sie trösten, Sie werden bald Befehle kennenlernen, die solche Probleme eleganter lösen.

Zur besseren Übersicht habe ich das Programm mit einigen REM-Zeilen versehen. Sicher ist Ihnen aufgefallen, daß ich beim Aufruf der Subroutinen nie eine REM-Zeile angesprungen habe. Es führt zwar zu keiner Fehlermeldung, eine REM-Zeile anzuspringen, da aber diese Zeile sowieso nicht ausgeführt wird, wäre es immerhin Zeitverschwendung.

Wollen Sie Ihr Programm mehrmals hintereinander benutzen, ist es lästig, es jedesmal von neuem starten zu müssen. Schaffen Sie Abhilfe und programmieren Sie noch folgende Zeilen dazu:

```
350 : PRINT "NOCH EINE RECHNUNG?"
360 : INPUT " 1 FUER JA";Y
370 : IF Y = 1 THEN GOTO 230
380 : END
```

Jetzt haben Sie's in der Hand, wann Sie aufhören wollen.

Eine wichtige Bemerkung fehlt noch: Subroutinen zeigen vor allem dann ihre Stärke, wenn sie in längeren Programmen sich ständig wiederholende Aufgaben erledigen müssen, etwa das Ausdrucken von Zwischenergebnissen.

Nachdem Sie die Unterprogramme anhand eines Beispiels beschnuppern durften, gebe ich Ihnen noch ein paar gute Regeln für den Umgang mit Subroutinen mit auf den Weg. Einige davon haben Sie schon kennengelernt. Zunächst die, welche für alle Computer gelten.

1. Jedes Unterprogramm muß mit GOSUB aufgerufen und mit RETURN beendet werden.
2. Aus Regel 1 folgt: Springen Sie ja nicht mit GOTO in ein Unterprogramm.

Der Grund: Der Computer trifft auf ein RETURN und weiß nicht wohin. Wegen Regel 2 verbietet es sich auch, ein Unterprogramm so aufzurufen:

```
100 : IF X = 5 THEN 130
```

also das Weglassen eines GOSUB hinter dem THEN, denn eine fortgelassene Sprunganweisung wird als GOTO und nicht als GOSUB interpretiert. Zeile 100 muß richtig lauten:

```
100 : IF X = 5 THEN GOSUB 130
```

3. Springen Sie nie mit GOTO aus einem Unterprogramm zurück, sondern immer nur mit RETURN.
4. Innerhalb eines Unterprogramms dürfen Sie den GOTO-Befehl natürlich verwenden, wie auch jede andere BASIC-Instruktion.

Hierzu ein kleines Beispiel:

```
100 : GOSUB 300

      ...

290 : REM UNTERPROGRAMM
300 : A = A+1
310 : S = S+A
320 : IF A<10 THEN GOTO 300
330 : RETURN
```

5. GOSUB darf an jeder Stelle eines Programms stehen.
6. Sie dürfen Unterprogramme schachteln (bis zu 32 Subroutinen beim PC-1500 A).

Es kann Ihnen dabei gar nichts passieren, wenn Sie es z.B. so machen:

```
 10 : REM GESCHACHTELTE UNTERPROGRAMME
 20 : WAIT 200
 30 : GOSUB 100
 40 : PRINT "GESCHAFFT!"
 50 : END
100 : REM ERSTES UNTERPROGRAMM
110 : PRINT "GEHT IN ORDNUNG"
120 : GOSUB 200
130 : PRINT "NA SEHEN SIE, WAR DOCH"
140 : PRINT "GAR NICHT SO SCHWER"
150 : RETURN
200 : REM ZWEITES UNTERPROGRAMM
210 : PRINT "AUCH DAS IST O.K."
220 : RETURN
```

Und wie funktioniert's? Der Sprung ins erste Unterprogramm geschieht

in Zeile 30: GOSUB 100. Der erste Befehl dieser Subroutine lautet: Nun PRINT mal schön "GEHT IN ORDNUNG". Sie haben diesen Sprung ja auch nach allen Regeln der Programmierkunst richtig gemacht. Der zweite Befehl dieses Unterprogramms ist GOSUB 200. Damit wird im Unterprogramm eine zweite Subroutine aufgerufen. Schnell hüpft der Computer ins zweite Unterprogramm und nach der REM-Zeile, um die er sich nicht zu scheren braucht, erscheinen die Worte AUCH DAS IST O.K. auf dem Display. Das folgende Kommando, RETURN, befiehlt dem Programm, zu der Instruktion, die nach GOSUB 200 kommt, zurückzukehren. Das ist Zeile 130 im ersten Unterprogramm (wir sind noch nicht wieder im Hauptprogramm). Jetzt werden Sie ein bischen aufgemuntert. Mit dem Satz NA SEHEN SIE, WAR DOCH GAR NICHT SO SCHWER klopft Ihnen der Computer symbolisch auf die Schulter. Danach folgt in Zeile 150 das RETURN des ersten Unterprogramms und Sie landen wieder sicher hinter dem ersten Unterprogrammaufruf GOSUB 100, also auf Zeile 40. GESCHAFFT! muß nun auf Ihrer Anzeige stehen. Das END in Zeile 50 ist nötig, weil sonst das Programm unerlaubt in die erste Subroutine gelangt. Wie Sie sehen, kann END auch mitten im Programm stehen.

In diesem Beispiel treffen Sie wieder auf die alte Programmierregel, der Sie schon bei der Klammersetzung und bei den verschachtelten Schleifen begegnet sind: Was zuerst begonnen hat, hört zuletzt auf und was zuletzt angefangen wurde wird zuerst beendet.

Einige besondere Süppchen werden noch auf Computern wie z.B. dem PC-1500 A gekocht. Mit solchen Geräten können Sie ein Unterprogramm nicht nur durch

```
GOSUB Zeilennummer
```

sondern auch mittels

```
GOSUB I+J
```

also mit einem arithmetischen Ausdruck hinter GOSUB, dessen Wert eine Zeilennummer ergeben muß, oder durch

```
GOSUB "MARKE"
```

aufrufen. Wie das läuft, haben Sie schon beim GOTO-Befehl kennengelernt.

Aufgabe 9.1

Bitte schreiben Sie ein Programm, welches Ihnen eine Zahl X einliest, von der dann wahlweise der Funktionswert EXP(X), LN(X), LOG(X) oder 10^X berechnet und das Ergebnis auf dem Display ausgegeben wird. Verwenden Sie dazu die Unterprogrammtechnik, wie Sie sie in dem Programm VIER GRUNDRECHENARTEN kennengelernt haben.

10 . K a p i t e l

FRÄULEIN, ZUM DIKTAT!
- Die String-Variablen -

COMPUTE heißt ins deutsche übersetzt "rechnen, zählen" und im Sprachgebrauch von Computerbenutzern wird diese Maschine gerne auch einfach "Rechner" genannt. Dies ist wohl auch die Aufgabe, für die Computer eigentlich konstruiert worden sind, nämlich um Mathematikern, Physikern und Statistikern ihre Arbeit zu erleichtern, bzw. um Ihnen neue Arbeitsgebiete zu erschließen.

Sie haben im ersten Kapitel dieses Buches erfahren, daß ein Computer, egal ob Taschenrechner oder Rechenzentrum, im wesentlichen 1 und 1 zusammenzählt. Im 4.-9. Kapitel haben Sie sehen können, wie diese simple Fähigkeit dank findiger Elektronikingenieure und Systemprogrammierer und, last not least, dank Ihrer Programmierkenntnisse dazu ausgenutzt werden kann, schwierigste mathematische Probleme zu lösen.

Daß der warme Regen der computerunterstützten Arbeitshilfen jedoch nicht auf die "rechnenden Berufe" beschränkt bleibt, wissen Sie spätestens, seit Sie die Sekretärin aus der Nachbarschaft vom endgültigen Aussterben des Tippfehlers in der deutschen Geschäftsbriefewelt schwärmen hörten. Sie spielte damit nicht etwa auf bisher ungekannte orthographische Qualitäten bei den neuen Schulabschlußjahrgängen an, sondern dachte eher an die frisch eingeführte Textverarbeitungsmaschine in ihrem Büro. Vorbei die Zeiten der überfüllten Papierkörbe und des Suchens nach einem noch brauchbaren Tipp-ex-Steifen: Der Computer wirds schon richten!

Textverarbeitung, also der Umgang mit Worten, Sätzen und ganzen Texten gehört auch zu den Fähigkeiten von Microprozessorsystemen. Moderne Büroschreibmaschinen verdienen oft schon die Bezeichnung "Terminal" oder Microcomputer, auch wenn sie kein BASIC verstehen und die Sekretärin für ihre Bedienung nicht programmieren können muß. Denn war eine mechanische Schreibmaschine nur dazu geeignet, den angetippten Buchstaben gut leserlich aufs Papier zu bringen, also den Bleistift zu ersetzen, tritt diese Eigenschaft bei größeren elektronischen Schreibmaschinen oder gar Textverarbeitungssystemen fast in den Hintergrund. Ihre Fähigkeiten bestehen darin, Texte zu speichern, zu korrigieren, beliebig oft auszudrucken und evtl. gar zu verknüpfen.

Schon kleine, verhältnismäßig preiswerte (unter 1000.- DM) elektronische Scheibmaschinen, die mit dem kostengünstigen Thermodruckkopf arbeiten, verfügen über diese Eigenschaft. Die Zeiten, in denen gerade ungeübte Tipper jeden Brief dreimal schreiben mußten, bis er endlich fehlerfrei war, bzw. bis die Nerven nicht mehr mitmachen wollten, könnten also bald vorbei sein.

Die Computer und ihre Programmierer wollen aber nicht damit zufrieden sein, diese relativ eintönigen, routinemäßigen Aufgaben zu übernehmen. Schon kursiert in den Vereinigten Staaten (wo auch sonst?) ein Programm, PRACTER ist sein Name, das sich darin versucht hat, ein Buch selbst zu

schreiben, welches dann auch tatsächlich veröffentlicht wurde. Es handelt sich dabei nicht etwa um ein mathematisches Fachbuch, nein, der Computer schrieb Kurzgeschichten und Gedichte!!! Wenn Sie nun glauben, solche Leistungen könne wohl nur ein gigantisches Großrechner"hirn" bewältigen, muß ich Sie enttäuschen. PRACTER ist ein BASIC-Programm, geschrieben für einen Microcomputer mit 64 K-Speicher und einem Z80-Microprozessor, also einen Home-Computer wie etwa den SHARP MZ-821. Ein Beispiel für die dichterische Kunst von PRACTER kann in der ZEIT vom 3. 5. 1985 nachgelesen werden. Dabei fällt allerdings auf, daß der Text zwar den grammatikalischen Regeln entspricht, Sie den Sinn jedoch selbst hineininterpretieren müssen. Doch das kann Ihnen auch beim Lesen moderner Gedichte von menschlichen Autoren so ergehen.

Ob all diese Fertigkeiten des Microprozessors für die Menschheit nun einen Segen bedeuten oder ob sie eine Katastrophe heraufbeschwören, soll uns hier ausnahmsweise mal nicht interessieren. Uns interessiert vielmehr die Frage: "Wie bringt es eine Rechenmaschine überhaupt fertig, mit Texten umzugehen?"

Die Grundlage hierfür stellt ein Ihnen noch vorenthaltener Variablentyp dar: die String-Variable. Mit ihr wollen wir uns in diesem Kapitel ausführlich beschäftigen.

10.1 Die String-Variablen

Wenn wir Menschen mit unserem "Bio-Computer", den wir Gehirn nennen, Sprache produzieren, geschieht dies, indem wir eine Reihe von Zeichen miteinander zu einer Bedeutungseinheit verknüpfen. Bei diesen Zeichen kann es sich um gesprochene Laute, sogenannte Phoneme, handeln oder beim lautlos niedergeschriebenen Text um Zeichen, die den Phonemen annähernd analog sind, die Buchstaben. Hierbei haben wir als Kind mühevoll lernen müssen, die Bedeutung einer solchen Kette von Zeichen zu verstehen und schließlich selbst produzieren zu können. Als Erwachsener ist es uns nunmehr fast unmöglich, eine Buchstabenkette wie

H, A, U, S,

zu betrachten, ohne sofort den Sinn dieser Kette zu verstehen, also automatisch das Bild eines Hauses vor Augen zu haben. Dieser unwillkürlichen Verknüpfung von der Gestalt einer Zeichenkette und ihrer Bedeutung müssen wir uns entledigen, wenn wir uns den oben angekündigten Stringvariablen unseres Computers zuwenden.

Der Variablentyp der String-Variable zeichnet sich gerade dadurch aus, daß der Rechner den Inhalt, mit dem die Variable "gefüllt" ist, nicht "versteht". Die Stringvariable soll für uns nur eine Reihe beliebiger Zeichen aufbewahren. Daher auch der Name 'String", was nichts anderes bedeutet als "Schnur, Reihe oder Kette". Eine solche Reihe von Zeichen kann etwa der Name ROSI sein. Versuchen Sie jedoch die Zuweisung

```
LET N = ROSI
```

tut Ihr Rechner das mit ERROR 1 ab. Das liegt nicht etwa daran, daß Ihr PC doch nicht so sehr zur Textverarbeitung geeignet ist. Vielmehr erkennt der Computer N als den Namen einer numerischen Variable und seit dem 5. Kapitel wissen Sie, daß diese nie den Wert ROSI annehmen kann. Was schließen wir daraus? Richtig, der Rechner muß bereits am Variablennamen erkennen, daß es sich um eine String-Variable handelt. Dies geschieht, indem sie hinter den Namen der Variablen ein **$-Zeichen** setzen. Und damit Sie sich merken, daß der Inhalt, der der Stringvariablen zugewiesen wird, keinen "Wert" hat, muß dieser in Gänsefüßchen gestellt werden. Lassen Sie uns die Zuweisung jetzt richtig machen:

```
LET N$ = "ROSI"
```

So sieht das also dann aus. Und natürlich können Sie sich den Inhalt dieser schönen Variable auch jederzeit ausgeben lassen:

```
PRINT N$
```

Wenn Sie glauben, mit ROSI nichts anfangen zu können, ohne auch noch ihre Telefonnummer zu wissen, machen Sie doch einfach:

```
LET TN$ = "ROSI: TEL. 32168"
```

Ich denke, Ihr PC schluckt auch dies noch. Probieren Sie's einfach aus. An diesem Beispiel sehen Sie gleich: Der Variablenname kann, abhängig von Ihrem Computertyp, auch aus zwei Buchstaben bestehen plus dem obligatorischen $ am Schluß. Wie bei den numerischen Variablen kann das zweite Zeichen des Variablennamens entweder ein Buchstabe oder eine Zahl sein. Als zulässige Variablennamen sind also erlaubt:

```
A$, B$, C$, ... Z$
AA$, BB$, ... ZZ$
A1$, A2$, B1$, ... Z9$
```

Noch etwas können Sie dem Beispiel oben entnehmen. Die Variable TN$ beinhaltet nicht nur Buchstaben, sondern auch Satzzeichen und sogar eine Zahlenreihe. Deshalb scheint mir auch die oft gebrauchte Bezeichnung "Textvariable" etwas irreführend, denn die Stringvariablen sind nicht darauf beschränkt, Wörter zu speichern, sondern nehmen eben jedes Zeichen aus dem Zeichenvorrat der Tastatur, solange Sie es nur in Anführungsstriche setzen. Eine Stringvariable mit dem Inhalt

```
T$ = "79358"
```

ist also durchaus zulässig, wenn Sie auch berücksichtigen müssen, daß der Computer nicht "weiß", welche Zahl im Wert von T$ steht und Sie deshalb besser nicht versuchen, mit T$ zu rechnen.

Doch lassen Sie uns zu unserer Variablen TN$ zurückkehren. Sie wissen jetzt, daß der Rechner jeden Nonsens als Wert einer Stringvariablen akzeptiert, Sie wissen noch nicht, daß dieser Nonsens nicht länger als 16 Zeichen sein darf (gilt für den PC-1500 A). Die höchstzulässige Länge einer Stringkonstante ist bei jedem Rechner festgelegt. Schauen Sie für Ihren in der Bedienungsanleitung nach.

Wieviel Zeichen, würden Sie sagen, beinhaltet unsere Stringvariable TN$? Zwölf? Falsch, Sie haben Punkt und Doppelpunkt vergessen mitzuzählen. Also vierzehn? Wieder falsch, denn zwischen ROSI: und TEL sowie zwischen TEL. und 32... befindet sich jeweils ein Leerzeichen und dies muß, so unscheinbar es auch sein mag, als Inhalt der Variablen mitgezählt werden. Also, nicht vergessen, denn jetzt hat TN$ schon 16 Zeichen und ist damit voll.

Das sind nun also diese mysteriösen String-Variablen, die den Schlüssel zur Textverarbeitung darstellen. Falls Sie jedoch im Moment einen Pocket-Computer vor sich liegen haben, leuchtet Ihnen sicher auch ein, daß dieses gezielt kompakt konstruierte Ding keinen idealen Ersatz oder gar einen Fortschritt gegenüber einer Schreibmaschine darstellen kann. Niemand möchte gerne Briefe auf einer so engen Tastatur tippen müssen und dies ist wohl auch kaum die Aufgabe, die Sie Ihrem SHARP-PC zugedacht haben. Trotzdem: Grundsätzlich spricht nichts dagegen, daß Sie es tun könnten.

Um Ihnen den Umgang mit Strings an einem Programm zu illustrieren, habe ich mir jedoch ein passenderes Beispiel ausgedacht. Vorher muß ich allerdings erst eine grundsätzliche Bemerkung zu den meisten Beispielen der nächsten Kapitel loswerden. Ein sinnvoller Umgang mit Textvariablen und selbst die einfachste Textverarbeitung setzen ein Medium voraus, auf dem der Text dargestellt werden kann. Hierzu mag ein Bildschirm noch ganz annehmbar sein, ein einzeiliges Display stößt jedoch sehr schnell an seine Grenzen. Die folgenden Beispielprogramme sind deshalb für den PC-1500 A mit dem Drucker/Cassetten-Interface CE-150 geschrieben, damit auch ein Text entstehen und schwarz auf weiß gelesen werden kann. Wenn Sie über keinen Drucker verfügen, behalten die Programme trotzdem ihre prinzipielle Gültigkeit und erfüllen auch ihren pädagogischen Zweck. Sie müssen allerdings für einen Probelauf alle Druckerbefehle (etwa LPRINT) durch Bildschirmbefehle (etwa PRINT oder PAUSE) ersetzen. Einverstanden?

10.2 Programm "TUERSCHILD"

Sind Sie auch schon mal vor einer verschlossenen Bürotür gestanden, an die ein Zettel geheftet war: "Bin in einer halben Stunde zurück."? Und haben Sie sich dann auch gefragt, ob Sie jetzt am Beginn oder schon am Ende dieser halben Stunde sind und ob es sich nun lohnt, zu warten? Im Innern dieser Büros klebt dann meist ein wahres Reservoir derartiger Zettel, für jede Eventualität ein anderer. Lassen Sie uns zusammen ein Programm schreiben, das Ihnen diese Zettelwirtschaft erspart und Ihren Computer den jeweils gewünschten, aktuellen Zettel ausdrucken läßt. Zuerst müssen Sie sich entscheiden, welche Information auf so einem Zettel stehen soll, eine Frage, die ich nun zwangsläufig für Sie beantworte:

1. Daß gerade keiner da ist
2. Wo Sie sind
3. Wann Sie zurück sind
4. Eine Anrede und ein Schlußsatz

Damit hätten wir zwei Arten von Inhalten: Punkt 1 und Punkt 4 stellen

Aussagen dar, die bei jedem Ausdruck unverändert bleiben können, Punkt 2 und Punkt 3 sind ständig wechselnde Informationen. Wie Sie die ersten beiden Punkte aufs Papier bringen, brauche ich Ihnen nicht zu erklären, das sind simple PRINT-Anweisungen. Für die Punkte 2 und 3 braucht das Programm jedoch die jeweils neuesten Informationen, um sie in den Text irgendwo zwischen An- und Abrede einzufügen. Dazu benötigen wir die String-Variablen! Und natürlich einen INPUT-Teil in unserem Programm. Sehen Sie sich hierzu mal folgendes Programm an:

```
 10:CLS
 20:INPUT WO$
 30:INPUT ZT$
100:LPRINT "LIEBER
     BESUCHER,"
115:LPRINT "DAS BU
    ERO IST ZUR ZE
    IT LEIDER UNBE
    SETZT. ICH BIN
    "
125:LPRINT WO$
135:LPRINT "GEGEN"
145:LPRINT ZT$
155:LPRINT "BIN IC
    H VORAUSSICHTL
    ICH WIEDER ZUR
    UECK."
165:LPRINT "VIELEN
     DANK FUER IHR
     VERSTAENDNIS!
    "
170:END
```

Das Programm besteht eigentlich nur aus zwei verschiedenen Anweisungen, INPUT und LPRINT. In der Zeilen 20 und 30 erfolgt die Eingabe der Stringdaten in die beiden Variablen WO$ und ZT$, die dann in Zeile 125 und 145 wieder ausgegeben werden und zwar zwischen dem Text der Zeilen 100, 115, 135, 155 und 165. Die Variable WO$ soll den Ort Ihres Aufenthalts enthalten, also z.B. "ZUHAUSE", "IM ARCHIV" oder "BEIM CHEF". In die Variable ZT$ schreiben Sie den Zeitpunkt Ihrer voraussichtlichen Rückkehr, so etwa "13.00 UHR".

Wollen wir mal sehen, ob's funkitoniert. Probieren Sie's auf Ihrem Rechner auch mal aus. Bei mir ist nach Eingabe von "BEIM ESSEN" und von "13.45 UHR" folgender Ausdruck erschienen:

```
LIEBER BESUCHER,
DAS BUERO IST ZUR
ZEIT LEIDER UNBESE
TZT. ICH BIN
BEIM ESSEN.
GEGEN
13.45 UHR
BIN ICH VORAUSSICH
TLICH WIEDER ZURUE
CK.
VIELEN DANK FUER I
HR VERSTAENDNIS!
```

Sie sehen, es klappt. Der Zettel und das Programm erfüllen ihren Zweck. Das ist aber auch schon alles. Es ist weder ein schönes Programm, noch ein ansehnlicher Ausdruck. Da müssen wir noch was machen. Überlegen Sie, bevor Sie weiterlesen, mal selbst, was Sie wie verbessern könnten und basteln Sie ruhig etwas an dem Programm herum. Mal sehen, ob Ihnen ein schönerer Ausdruck gelingt, als mir.

Im Folgenden habe ich meinen Vorschlag aufgelistet und erläutert:

```
 4:REM **********
 5:REM TUERSCHILD
 6:REM **********
10:CLS
20:INPUT "WO SIND
    SIE?";WO$
30:INPUT "WANN SI
   ND SIE ZURUECK
   ?";ZT$
100:COLOR 3:LPRINT
    "LIEBER BESUCH
    ER,"
105:LF 1
110:COLOR 0
115:LPRINT "DAS BU
    ERO IST ZUR ZE
    IT LEIDER UNBE
    - SETZT. ICH B
    IN    "
120:COLOR 1
123:LF 1
125:LPRINT WO$
128:LF 1
130:COLOR 0
135:LPRINT "GEGEN
    ";
140:COLOR 2
145:LPRINT ZT$;" U
    HR"
150:COLOR 0
155:LPRINT "BIN IC
    H VORAUS-   SI
    CHTLICH WIEDER
      ZURUECK."
160:COLOR 3
163:LF 1
165:LPRINT "VIELEN
     DANK FUER  IH
    R VERSTAENDNIS
    !"
170:END
```

Nun, was hat sich verändert? Als erstes hat das Kind einen Namen bekommen (Zeile 4, 5, 6), so daß Sie es auf dem Listing gleich identifizieren können und gegebenenfalls auf einer Cassette wiederfinden können. Dann wurden die INPUT- Abfragen mit den dringend nötigen Kommentaren versehen, damit Sie auch nächste Woche noch wissen, was die Fragezeichen auf dem Display eignetlich abfragen (Zeilen 20, 30). Auffallend sind die neu ins Programm gekommenen Befehle

COLOR

LF

Diese sind nur auf dem PC-1500 A mit Interface CE-150 anwendbar und bewirken folgendes: Mit COLOR kann die Farbe des Ausdrucks bestimmt werden, eine Spielerei, die ich Ihnen hier leider nicht demonstrieren kann; LF bewirkt einen Zeilenvorschub, wodurch der Ausdruck übersichtlicher gestaltet werden kann (LF 1 = 1 Zeile weiter, LF 2 = 2 Zeilen weiter ...), Bei Rechnern ohne LF-Befehl kann ein Zeilenvorschub auch einfach durch einen "leeren" LPRINT-Befehl erreicht werden. Hier müssen Sie allerdings für jede gewünschte Leerzeile auch eine LPRINT-Zeile programmieren.

In unserem Beispiel kann das dann so aussehen:

```
100 LPRINT "LIEBER BESUCHER,"
110 LPRINT
115 LPRINT "DAS BUERO IST ... "
```

Eine weitere wichtige Veränderung sieht man dem Listing nicht gleich an, Sie werden es sofort merken, wenn Sie sich das Ergebnisbeispiel ansehen:

```
LIEBER BESUCHER,

DAS BUERO IST ZUR
ZEIT LEIDER UNBE-
SETZT. ICH BIN

IM ARCHIV

GEGEN 15.00 UHR
BIN ICH VORAUS-
SICHTLICH WIEDER
ZURUECK.

VIELEN DANK FUER
IHR VERSTAENDNIS!
```

Waren im ersten Beispiel noch Wörter am Zeilenende willkürlich an den unmöglichsten Stellen getrennt wie bei:

```
VIELEN DANK FUER I
HR VERSTAENDNIS!
```

so kommt das jetzt nicht mehr vor und das ist kein Zufall. Denn schon beim Programmieren der LPRINT-Anweisungen wurde bedacht, daß der Drucker nur 18 Zeichen pro Zeile ausdruckt, obwohl ja in die LPRINT-Anweisung mehr Zeichen pro Programmzeile geschrieben werden können (beim PC-1500 A sind das 72 Zeichen, also genau 4 Druckzeilen). Sie sollten also Ihre LPRINTs mittels Leerzeichen oder Trennungsstrichen so gestalten, daß ein Ausdruck entsteht, der den üblichen Schreib- und Trennungsregeln entspricht.

Noch was. Mir war es etwas lästig, bei der Zeitangabe immer das Wort UHR mit eingeben zu müssen, weshalb ich in Zeile 145 diesen Zusatz vom Programm ausdrucken lasse. Die beiden Semikolons in Zeile 135 und 145 bewirken übrigens, wie Ihnen bekannt sein dürfte, daß das Schreibwerk (bzw. der Cursor bei PRINT-Anweisungen) an der Stelle stehen bleibt, wo es mit der Ausführung des letzten LPRINT-Befehles fertig war. Das ist immer dann wichtig, wenn man verschiedene Textbausteine nahtlos zusammenfügen möchte. Sie dürfen dabei allerdings das obligatorische Leerzeichen (Blank) zwischen zwei Worten nicht vergessen, das Sie entweder am Ende des ersten Bausteins oder am Anfang des zweiten berücksichtigen müssen. Also nie vergessen: **Auch Blanks sind Daten!** Wir werden in einem späteren Kapitel noch genauer darauf eingehen, wie Sie Ihre

Ausdrucke gestalten oder besser gesagt formatieren können. Fürs erste soll's jedoch genug sein, und Sie können mit dem, was Sie in diesem Kapitel gelernt haben üben, alle möglichen Briefchen oder Notizen ausdrucken zu lassen. Wenn Sie sich fit genug fühlen, wagen Sie sich an die Übungsaufgabe unten 'ran:

Aufgabe 10.1

Entwerfen Sie ein kleines Programm, das Ihnen Merkzettel für Termine ausdruckt. Die veränderlichen Angaben sollen dabei sein:

1. Mit wem Sie sich treffen wollen
2. Wo das geschehen soll
3. Wann das geschehen soll

Ich will Ihnen keine weiteren Hinweise und Einschränkungen geben. Sie können also mal versuchen, Ihr individuelles Programm zu stricken. Einen Lösungsvorschlag finden Sie im Anhang.

11 . K a p i t e l

DIE GEHEIMSPRACHE DER COMPUTER
- Der ASCII-Code -

Wie eine Sting-Variable definiert ist und wie sie auszusehen hat, das haben Sie im letzten Kapitel erfahren. Die Frage, die wir uns am Anfang dieses Kapitels gestellt hatten, warum nämlich eine Rechenmaschine überhaupt mit alphabetischen Zeichen umgehen kann, ist allerdings damit noch nicht befriedigend beantwortet. Denn, Sie erinnern sich: Ein Computer versteht nur Binärzahlen sonst gar nichts!

Fällt es noch recht leicht, sich vorzustellen, daß die dezimal eingegebenen Zahlen vor der Verarbeitung im Microcomputer in Binärzahlen umgewandelt werden, stoßen wir bei dem Versuch, dies mit Worten oder Buchstaben zu tun, an die Grenzen unserer Schulmathematik. Trotzdem wird man nicht umhinkönnen, wenn man einen "Rechner" zum Umgang mit Sprache befähigen will.

Da es nun also keinen definierten Zusammenhang zwischen alphabetischen Zeichen und Binärzahlen gibt, hilft nur eins: man muß einen definieren. Zum Glück ist es nicht jedem Computerhersteller selbst überlassen, dies zu tun. Sonst könnten wir den Begriff "Kompatibilität" völlig aus unserem Wortschatz streichen. Was wir brauchen, ist eine allgemeinverbindliche Regelung. Und tatsächlich: Es existiert ein international gebrauchter Code, nach dem jedes Zeichen, das über die Tastatur angetippt wird, in eine achtstellige Binärzahl (oft auch nur in eine siebenstellige) umgewandelt wird. Besser als der Begriff Binärzahl ist in diesem Zusammenhang die Bezeichnung **8-Bit-Wort**, denn der numerische Wert dieser Binärzahl ist erstmal völlig unwichtig.

Aus der Länge von 8 Bit oder einem Byte, ergibt sich, daß uns maximal 256, also 2^8, verschiedene Zeichen zur Verfügung stehen können, die mit dem Binärzahlen 0000 0000 bis 1111 1111 codiert sind. Dezimal dargestellt ist das 0 bis 255.

Jetzt ist aber Zeit, Ihnen mitzuteilen, wie dieser überaus sinnvolle Code denn nun heißt. Die abgekürzte Bezeichnung, unter der er auch allgemein bekannt ist, habe ich Ihnen schon in der Überschrift zu diesem Kapitel verraten: ASCII-Code. Ausgeschrieben bedeutet das: **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. Zu deutsch: Amerikanischer Standard Code für den Informationsaustausch. Obwohl dieser Code für die Computerei einen unschätzbaren Wert darstellt, wurde er gar nicht für diese Maschinen erfunden. Das Wort "Informationsaustausch"deutet es schon an: Der ASCII war einst für den amerikanischen Fernschreiberverkehr entwickelt worden.

Eigentlich hätte uns schon viel früher die Notwendigkeit eines solchen Codes beschäftigen müssen, denn sind kommentierte INPUT-Abfragen oder PRINT-Ausgaben etwa keine verarbeiteten Texte?

Da ich annehme, daß Sie ein neugieriger Mensch sind und ganz gerne

ab und zu wüßten, welche Code-Nummer ein bestimmtes Zeichen bekommen hat, will ich Ihnen verraten, wie Sie das ganz einfach herauskriegen können. Fragen Sie doch Ihren Computer, der wird doch wohl wissen, womit er die ganze Zeit umgeht. Es gibt zwei Funktionen, mit denen Sie Ihm dieses Wissen entlocken können:

ASC

CHR$

Wie das geht, will ich Ihnen auch noch sagen. Interessiert Sie z.B. der ASCII-Code von A, genügt die Eingabe

ASC("A")

und Sie erhalten die Auskunft

65

auf dem Display. Ihnen wird die ASCII-Codenummer also dezimal mitgeteilt. Achten Sie bitte auch darauf, daß das Argument von ASC ein String ist und deshalb in Anführungszeichen gepackt werden muß. Machen Sie gleich noch ein paar Versuche:

```
ASC("B")
ASC("C")
ASC("Z")
```

Sie erhalten die Ergebnisse:

```
66
67
90
```

Diese Beispiele zeigen bereits, daß es nicht ganz willkürlich ist, welches Zeichen nun welche Code-Nummer erhalten hat. Vielmehr scheinen die Zeichen A - Z die Codes 65 - 90 in aufsteigender Reihenfolge für sich beansprucht zu haben. Daraus ergibt sich ein interessanter Zusammenhang:

ASC("A") < ASC("B")

Dieser Zusammenhang wird uns später noch sehr nützlich sein. Der ASCII-Code verfügt aber bekanntermaßen über mehr Zeichen, als nur die Codes 65 - 90. Sie könnten also auf die Idee kommen, sich zu fragen, welche Zeichen denn mit den kleineren und größeren Nummern verschlüsselt werden. Was versteckt sich beispielsweise hinter Nr. 91? Da fast jede Funktion auch eine Umkehrfunktion hat, stellt dies auch hier kein Problem dar:

CHR$(91)

flugs eingetippt und der Spannung wird ein Ende bereitet:

√

Sie haben also ein Sonderzeichen erwischt. Machen Sie den Versuch auch noch mit:

```
CHR$(99)
CHR$(63)
CHR$(49)
CHR$(33)
CHR$(32)
```

und betrachten Sie die Ergebnisse:

```
c
?
1
!
```

Außer Großbuchstaben gibt es also auch Kleinbuchstaben, Satzzeichen, Ziffern und mathematische Operatoren im Lieferumfang des ASCII-Codes. Auf die Aufforderung CHR$(0) - CHR$(32) schweigt der Rechner jedoch beharrlich. Dies liegt daran, daß diese Nummern nicht benötigt werden, bis auf eine Ausnahme. Probieren Sie mal:

```
ASC("SPACE") ENTER
```

Sollten Sie mit einer Fehlermeldung gerechnet haben, haben Sie hier mal unrecht, denn ohne mit der Wimper zu zucken liefert der Rechner Ihnen das Ergebnis dieser Funktion:

```
32
```

Man kann es nicht oft genug sagen: Auch das Blank ist ein Zeichen und hat sogar eine Nummer. Daß der ASCII-Code doch noch einige Ausnahmen bei den Nummern 0-32 macht, kann ich Ihnen leider nicht so einfach beweisen. Hier sind nämlich eine ganze Reihe nicht darstellbarer Funktionen untergebracht. Die ASCII-Nummer 24 steht etwa für "Cursor um eine Position zurück", und ASCII-Code 13 ist die ENTER-Taste. In diesem Zusammenhang soll nicht unerwähnt bleiben, daß der ASCII-Code nicht bei allen Zeichen für alle Computer identisch ist. Dies trifft vor allen auf die Graphik-Zeichen zu, die in der oberen Hälfte des ASCII-Codes untergebracht sind (Codenr. 127-255)

Das folgende kleine Programm erlaubt es Ihnen, auf Ihrem Rechner mal durch den ASCII-Code zu wandern und sich anzusehen, was es da so alles gibt. Geht Ihnen dieser Spaziergang zu schnell, ersetzen Sie einfach den PAUSE-Befehl durch ein PRINT.

```
10 : CLS : CLEAR
20 : FOR I = 33 TO 255
30 : PAUSE I;"     ";CHR$(I)
40 : NEXT I
50 : END
```

Sollten Sie über einen Drucker verfügen, versuchen Sie doch mal, sich

eine eigene ASCII-Codetabelle herstellen zu lassen. Wie wärs mit Hilfe dieses Programms?

```
10 : CLS : CLEAR
20 : LPRINT "ZEICHEN      ASCII"
30 : LF 2
40 : FOR I = 33 TO 122
50 : LPRINT CHR$(I);"           ";I
60 : NEXT I
70 : END
```

Ich habe mich auf die Zeichen 33 bis 122 beschränkt, weil nur sie auf dem Ballpen-Drucker des CE-150 darstellbar sind. Für alle, denen es an einem Drucker gebricht, ist hier meine Tabelle:

ZEICHEN	ASCII				
!	33	?	63	Π	93
"	34	@	64	^	94
#	35	A	65		95
$	36	B	66		96
%	37	C	67	a	97
&	38	D	68	b	98
'	39	E	69	c	99
(	40	F	70	d	100
)	41	G	71	e	101
*	42	H	72	f	102
+	43	I	73	g	103
,	44	J	74	h	104
-	45	K	75	i	105
.	46	L	76	j	106
/	47	M	77	k	107
0	48	N	78	l	108
1	49	O	79	m	109
2	50	P	80	n	110
3	51	Q	81	o	111
4	52	R	82	p	112
5	53	S	83	q	113
6	54	T	84	r	114
7	55	U	85	s	115
8	56	V	86	t	116
9	57	W	87	u	117
:	58	X	88	v	118
;	59	Y	89	w	119
<	60	Z	90	x	120
=	61	↓	91	y	121
>	62	↑	92	z	122

Betrachten Sie sich die Tabelle ruhig eine Zeitlang genau, denn immer, wenn Sie mit den Funktionen ASC und CHR$ arbeiten wollen, sollten

Ihnen einige Merkmale dieser Liste geläufig sein. So ist der Code für "7" nicht etwa 7, sondern eben 55. Oder: ASC("A") < ASC("a"). Alle Ziffern haben eine niederere Codenummer als die Buchstaben usw.

Diese Merkmale werden wichtig, wenn man mit String-Variablen mehr anfangen will, als sie nur zu speichern und wieder auszudrucken. Es gibt nämlich einige, wenn auch wenige, Operationen, die Sie mit Strings ausführen können:

Merke: Mit String-Variablen sind die Operationen

= + < >

durchführbar.

Durch das Zeichen = können Sie im Programm überprüfen lassen, ob zwei Stringvariablen identisch sind. Aber Vorsicht! Sie müssen haargenau gleich sein um diese Bedingung zu erfüllen:

"OTTO" ≠ "Otto"

Die Operation + erlaubt es, zwei Strings aneinanderzuheften, es ist also keine mathematische Addition, sondern nur eine Verknüpfung zweier Textketten. Ich demonstriere es Ihnen am besten:

```
10 : CLS : CLEAR
20 : LET A$ = "LINA "
30 : LET B$ = "IST "
40 : LET C$ = "DOOF"
50 : PRINT A$+B$+C$
60 : END
```

Sollte sich eine Lina unter meinen Leserinnen befinden, dann blättern Sie doch jetzt bitte gleich weiter, denn wir machen jetzt einen Probelauf und drücken RUN:

LINA IST DOOF

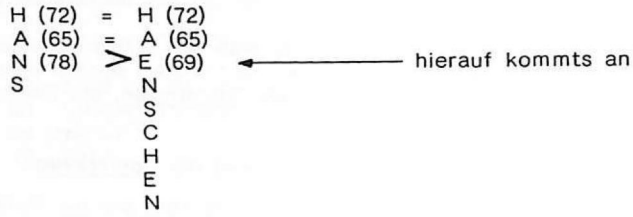
erscheint auf der Anzeige und demonstriert, daß Computer nicht allwissend sind (oder kennen Sie eine doofe Lina?).

Interessanter wird es jetzt, wenn wir uns überlegen, was man wohl mit den Vergleichsoperatoren < und > anfangen kann. Ist da nun "Hans" größer als "Haenschen" oder verhält es sich umgekehrt, weil ja "Haenschen" der längere String ist? Ich will Sie nicht länger auf die Folter spannen: Es gilt

"HANS" > "HAENSCHEN"

Jedoch beileibe nicht, weil der Computer etwa den Inhalt der String-Variablen verstünde. Bei dem Größenvergleich zwischen String-Variablen hält sich unser Microcomputer nämlich an den ASCII-Code, der uns mittlerweile ja auch in Fleisch und Blut übergegangen ist. Er vergleicht Zeichen um Zeichen der beiden Variablen und die Variable, die als erste

ein Zeichen mit einer höheren ASCII-Codenummer enthält, ist dann "größer als" die andere. In unserem Fall läuft der Vergleich so ab:



Dieses Vergleichsprinzip ist Ihnen allen geläufig, denn nach dieser Methode sind alle Nachschlagewerke alphabetisch geordnet. Ein Fachausdruck hierfür ist lexigraphische Ordnung. Es ist deshalb nicht weiter verwunderlich, daß das Sortieren nach dem Alphabet der wichtigste Anwendungsbereich der Vergleichsoperationen bei String-Variablen ist.

Noch was: Nehmen Sie die beiden Zeichen < und > zusammen zu einem <>, so bedeutet das "ungleich" und Sie haben eine fünfte für Strings zulässige Operation.

Damit Sie mal nach Herzenslust Ihren Computer vergleichen und verknüpfen lassen können, habe ich Ihnen ein kleines Programm geschrieben, mit dem Sie tüchtig rumspielen sollten, damit Sie die Stings bald kennen wie Ihre Hosentasche:

```
 4:REM *********
 5:REM ALPHATEST
 6:REM *********
10:CLS :CLEAR
20:WAIT 250
30:INPUT "1. WORT
   ? ";A$
40:INPUT "2. WORT
   ? ";B$
50:IF A$=B$THEN
   GOTO 140
60:IF A$<B$THEN
   GOTO 90
70:PRINT B$;" KOM
   MT VOR ";A$
80:GOTO 110
90:PRINT A$;" KOM
   MT VOR ";B$
100:GOTO 110
110:LET C$=A$+B$
120:PRINT "BEIDE Z
    US.: ";C$
130:GOTO 10
140:PRINT "BEIDE W
    ORTE SIND GLEI
    CH"
150:GOTO 10
```

Das Programm erklärt sich eigentlich von selbst, so daß ich mich (und Sie) damit nicht aufhalten möchte. Hier einige Tips für interessante Vergleiche. Konfrontieren Sie mal:

KOHL	mit	STRAUSS
SEX	mit	SIEBEN
"OTTO"	mit	OTTO
GROESSE 38	mit	GROESSE: 36
007	mit	08/15

Interpretieren Sie die Ergebnisse bitte nur nach den Regeln der Programmierkunst, also anhand Ihrer ASCII-Codetabelle und dem lexigraphischen Ordnungsschema.

12 . K a p i t e l

WORTSPALTEREIEN
- Die String-Funktionen -

Bei unseren bisherigen Betrachtungen der Textvariablen stand im Mittelpunkt, daß der Rechner befähigt sein sollte, mit alphabetischen Zeichenfolgen, nämlich mit Worten und Texten, zu arbeiten. Daß die Zeichenfolge einer Stringvariablen jedoch genausogut aus Ziffern bestehen kann, trat dabei in den Hintergrund. Es gibt nämlich eine ganze Reihe von Fällen, wo der numerische Wert einer Ziffernfolge von keiner oder von nur zweitrangiger Bedeutung ist. Denken wir nur an Telefonnummern, Postleitzahlen oder Autokennzeichen. In unserem Programm TUERSCHILD war die Variable ZT$ so ein Beispiel, denn bei der Zeitangabe kommt es auf den mathematischen Wert von 17.30 UHR nicht an und wir hatten in unserem Programm ja auch nicht die Absicht, mit der eingegebenen Zahl zu rechnen. Man wird also immer dann, wenn die korrekte Darstellung einer Zahlenkette wichtiger ist, als die Möglichkeit, mit ihr zu rechnen, den Typ der String-Variable wählen um sie in einem Programm zu verarbeiten.

Hätten wir in unserem TUERSCHILD-Programm anstelle der String-Variablen ZT$ die numerische Variable ZT benutzt, hätten wir einige Nachteile in Kauf nehmen müssen. Eine Eingabe der Uhrzeit incl. der Bezeichnung UHR wäre ausgeschlossen und was hätten wir für einen Ausdruck zu erwarten gehabt, wenn wir etwa den Wert 14.30 eingegeben hätten? Auf unserem Schildchen wäre zu lesen gewesen:

GEGEN 14.3 UHR BIN ICH ...

Denn als mathematische Größe ist 14.3 mit 14.30 identisch und der Computer arbeitet lieber rationell und läßt die Null am Schluß weg. Zwar wäre auch dieses Problem zu lösen gewesen (mit einer formatierten PRINT-Anweisung), aber es ist einfacher, gleich den geeigneteren Variablentyp zu wählen um sich derartige Schwierigkeiten von Anfang an zu ersparen.

Ein anderes Beispiel: Sie wollen die Telefonnummer 0711/36 99 99 als numerische Variable TN abspeichern, etwa:

```
TN = 0711.369999
```

Abgesehen davon, daß Sie regelmäßig die Null von der Vorwahl unterschlagen bekämen, könnte es Ihnen bei einer internen Rechengenauigkeit von 6 Stellen passieren, daß daraus 0711.370000 wird oder einfach nur 711.37. Basta! Dieses Beispiel ist zwar etwas konstruiert und unsere Rechner sind zu genau, um schon vor 6 Stellen zu kapitulieren, aber es illustriert den Unterschied zwischen einer mathematischen Größe und einer String-Variablen. Für die mathematisch etwas bewanderten unter Ihnen: Strinvariablen haben Nominalskalenniveau. Zu deutsch: Mit String-Variablen kann man, auch wenn sie wie normale Zahlen aussehen sollten, nicht rechnen. Deutlich wird dies auch am Beispiel der Postleitzahlen. München ist nicht viermal so groß wie Hamburg, nur weil es die Postleit-

zahl 8000 hat und Hamburg nur 2000. Es macht keinen Sinn, Postleitzahlen zu multiplizieren oder voneinander zu subtrahieren. Vielleicht fällt Ihnen an dieser Stelle ein, daß da doch die Operation " + " mit Strings erlaubt war, doch das war keine mathematische Operation. Um den Unterschied noch mal zu verdeutlichen, ein Beispiel:

```
10 : LET A$ = "12.5"
20 : LET B$ = "7.5"
30 : PRINT A$+B$
```

Das Ergebnis ist nicht etwa "20", sondern:

```
12.57.5
```

Mathematisch völliger Unsinn, aber eine korrekte Verknüpfung der beiden Strings A$ und B$.

12.1 Die Funktionen VAL und STR$

Soweit, so gut. Manchmal wäre es trotzdem einfach praktisch, wenn der Computer den mathematischen Wert eines Strings nehmen könnte, kurz damit eine Rechenoperation durchführen würde, und danach wieder alles beim alten bliebe. Nehmen wir wieder unsere Uhrzeit aus dem TUER-SCHILD- Programm und nehmen wir an, Ihnen sei es lieber, wenn Sie eingeben könnten, wie lange Sie wegbleiben und der Rechner könnte für Sie ermitteln, wann Sie wieder zurück sind. Wozu hat man denn so ein Ding! Nun, ich will's Ihnen nicht verheimlichen: es geht. Mit der Funktion

```
VAL
```

können Sie einer numerischen Variablen (z.B. ZT) den mathematischen Wert einer Textvariablen (z.B. ZT$) oder einer beliebigen Textkonstanten (z.B. "14.30 UHR") zuweisen. Es ist also folgendes möglich:

```
LET ZT = VAL(ZT$)

LET ZT = VAL("14.30 UHR")
```

Die numerische Variable ZT hat jetzt den Wert ZT = 14.3 und damit können Sie rechnen, soviel Sie wollen. Wenn Sie nach Beendigung dieses Rechenvorgangs das Ergebnis dann wieder zu einer harmlosen Textvariablen machen wollen, bitte schön:

```
LET ZT$ = STR$(ZT)
```

Die Funktion VAL besitzt also auch eine Umkehrfunktion und die heißt:

```
STR$
```

Die Rückverwandlung hat nur einen Haken: Die schöne Form mit zwei Stellen hinterm Komma und der Bezeichnung UHR ist verlorengegangen. Letzteres können Sie glattbügeln, indem Sie einfach eine zusätzliche Variable U$ = " UHR" nehmen und dann schreiben:

```
LET ZT$ = STR$(ZT)+U$
```

Bevor ich an einem Programmbeispiel beweisen möchte, daß man mit diesen Funktionen auch tatsächlich etwas anfangen kann, müssen Sie sich allerdings noch näher mit der Arbeitsweise von VAL vertraut machen, der wichtigeren der beiden Funktionen. Geben Sie Ihrem Rechner im Direktmodus folgende Aufgaben:

```
VAL("12.Mai 1985")
VAL("12.05.1985")
VAL("TEL: 321908")
VAL("0711/32 19 08")
VAL("32 19 08")
VAL("-12 GRAD CELSIUS")
```

Und schauen Sie sich nun die Ergebnisse an:

```
12
12.051985
0
711
321908
-12
```

Haben diese Sie etwas verwirrt? Macht nichts, wird sich alles aufklären. Folgendes müssen Sie über VAL wissen:

1. Die Umwandlung beginnt mit dem ersten Zeichen des Strings und wird abgebrochen, sobald der Rechner auf ein Zeichen stößt, das er nicht als numerischen Wert interpretieren kann. So geschehen in den Beispielen 1, 3 und 4.
2. Leerstellen werden überlesen, also ignoriert. Siehe Beispiel 5.
3. Ein Punkt wird als Dezimalpunkt interpretiert. Ist mehr als ein Punkt vorhanden, werden die dem ersten folgenden ignoriert. Dazu: Beispiel 2
4. Die Zeichen + und - werden am Anfang des Strings als Vorzeichen interpretiert, mitten im String führen sie zum Abbruch des Konvertierungsvorgangs. Siehe Beispiel 6.

Alles klar? Gut, dann können wir uns an das versprochene Anwendungsbeispiel heranwagen. Unser altes TUERSCHILD-Programm soll nun so umgestaltet werden, daß der Rechner aus den Angaben wie spät es im Moment ist und wie lange Sie weg bleiben wollen Ihre Rückkehrzeit selbstständig ermittelt. Dabei soll die Funktion VAL eingesetzt werden. Sehen wir uns in Abb.7 einmal das Flußdiagramm für das alte Programm an. Es soll so wenig wie möglich daran verändert werden, damit es die neu gestellte Aufgabe erfüllen kann. Der gesamte mühsam gestaltete PRINT-Teil kann eigentlich so bleiben, wie bisher. Mit einer kleinen, aber wichtigen Ausnahme, wie Sie noch sehen werden. Verändern müssen wir den INPUT-Teil, wobei Zeile 20 bestehen bleiben kann.

An die Stelle der Abfrage "WANN SIND SIE ZURÜCK?" in Zeile 30 soll nun jedoch gefragt werden "WIE LANGE SIND SIE WEG?". Und damit Ihr Rechner überhaupt eine Chance hat, die Rückkehrzeit auszurechnen,

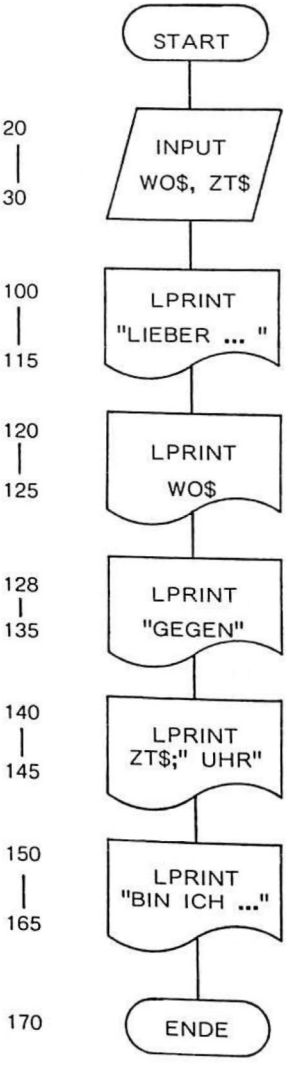


Abb.7a

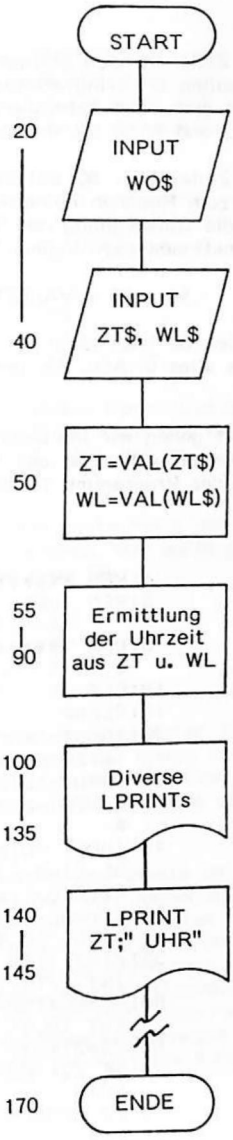


Abb.7b

Abbildung 7: Flußdiagramme zu den Programmen TUERSCHILD (Abb.7a) und TUERSCHILD II (Abb.7b)

soll in Zeile 40 noch gefragt werden, "WIE SPAET IST ES?". Beide Angaben sollen in Stringvariablen eingelesen werden, zu Übungszwecken, versteht sich. Ich habe die beiden Textvariablen ZT$, für die jetzige Uhrzeit, und WL$, für die Dauer Ihrer Abwesenheit, genannt.

In den Zeilen 50 - 90 soll die Rückkehrzeit errechnet werden. Da wir jedoch zum Rechnen numerische Variablen benötigen, muß dort als erstes die Umwandlung der Strings in numerische Variablen erfolgen. Die Funktionen dazu kennen Sie:

```
50 : ZT = VAL(ZT$) : WL = VAL(WL$)
```

Ab Zeile 100 folgt dann der (beinahe) unveränderte PRINT-Teil. Sie können das alles in Abb. 7b, dem Flußdiagramm des neuen Programms verfolgen.

So, jetzt gehen wir ins Detail. Dabei hilft uns das Flußdiagramm nicht mehr weiter und es ist das Beste, ich bringe Ihnen gleich das komplette Listing des Programms TUERSCHILD II :

```
 4:REM **********
 5:REM TUERSCHILD
       ----II----
 6:REM  *********
   *
10:CLS
15:CLEAR
20:INPUT "WO SIND
    SIE?";WO$
30:INPUT "WIE SPA
   ET IST ES?";ZT
   $
40:INPUT "WIE LAN
   GE SIND SIE WE
   G?(STD)";WL$
50:ZT=VAL (ZT$):W
   L=VAL (WL$)
55:Z1=ZT*100-100*
   INT (ZT)
60:W1=WL*100-100*
   INT (WL)
65:Z2=Z1+W1
70:IF Z2<=60THEN
   GOTO 85
75:Z2=((Z2-60)/10
   0)+1
80:GOTO 90
85:Z2=Z2/100
90:ZT=INT (ZT)+
   INT (WL)+Z2
100:COLOR 3:LPRINT
    "LIEBER BESUCH
    ER"
105:LF 1
110:COLOR 0
115:LPRINT "DAS BU
    ERO IST ZUR ZE
    IT LEIDER UNBE
    - SETZT. ICH B
    IN"
120:COLOR 1
123:LF 1
125:LPRINT WO$
128:LF 1
130:COLOR 0
135:LPRINT "GEGEN"
    ;
140:COLOR 2
143:USING "###.##"
145:LPRINT ZT;" UH
    R"
150:COLOR 0
155:LPRINT "BIN IC
    N VORAUS-   SI
    CHTLICH WIEDER
      ZURUECK."
160:COLOR 3
163:LF 1
165:LPRINT "VIELEN
     DANK FUER  IH
    R VERSTAENDNIS
    !"
168:LF 5
170:END
```

Bevor wir uns ansehen, was das Programm im "Rechen-Teil" mit unseren Variablen ZT und WL anstellt, sind noch einige Vorüberlegungen wichtig. In welcher Form wollen Sie die Variablen ZT$ und WL$ in Zeile 30 und 40 eingeben? Und zweitens: Was geschieht dann damit in Zeile 50?

Variable ZT$: Sie waren bisher gewohnt, die Zeit so einzugeben, wie sie üblicherweise auch geschrieben wird und wie sie dann Ihr Drucker auch ausgeben soll, nämlich "13.00 UHR" oder "14 UHR" usw. In Zeile 50 wird dann daraus:

```
ZT$ = "14 UHR"              ZT = 14
ZT$ = "13.30 UHR"    ----->  ZT = 13.3
ZT$ = "14 UHR 30"           ZT = 14      !!!
```

Also, hüten Sie sich vor einer Eingabe in der letzten Form. Die beiden anderen Beispiele repräsentieren in einer eindeutigen Form die Uhrzeit, auch wenn sie nicht so formschön dargestellt ist, aber das ist zum rechnen auch nicht erforderlich.

Variable WL$: Hier müssen wir uns auf eine bestimmte Form der Eingabe einigen, mit der wir dann weiterrechnen können. Wie wichtig das ist, zeigen einige Beispiele zum Geschehen in Zeile 50:

```
WL$ = "1.30 STD"            WL = 1.3
WL$ = "90 MIN"              WL = 90
WL$ = "1 1/2 STD"    ----->  WL = 11
WL$ = "1 STD 30 MIN"        WL = 1
```

Sie sehen, obwohl WL$ immer die gleiche Zeitspanne beinhaltet, hat WL viermal einen anderen Wert bekommen. Hierbei sind die ersten beiden Umwandlungen brauchbar, die letzten beiden liefern offensichtlich verfälschte Werte. Es hat sich gelohnt, sich mit der VAL-Funktion vertraut zu machen, oder?

Ob Sie sich in Ihrem Programm nun für die erste oder zweite der möglichen Eingabeformate entscheiden, ist nur insofern von Bedeutung, als Sie sich im weiteren Programmierablauf konsequent daran halten müssen. Rechnen können Sie mit beiden Formen. Ich habe mich für die erste Form entschieden, weil Sie es bei der Benutzung des Programms ja bequem haben wollen. Oder wissen Sie auf Anhieb, wieviele Minuten 2.30 h haben? Damit Sie das nicht vergessen, steht im INPUT-Kommentar die Bemerkung (STD), also "Angabe in Stunden" (sexagesimal, versteht sich). Damit das auch klappt, hier einige Beispiele für die Korrekte Eingabe:

Abwesenheit:	Eingabe:
1 Std. 10 Min.	1.10 STD
50 Min	0.50 STD
1 Std. 5 Min	1.05 STD (nicht: 1.5)

So, und jetzt stürzen wir uns auf den Rechenvorgang in den Zeilen 55-90. Im Prinzip wollen Sie folgende einfache Rechnung ausführen:

$$ZT_{(neu)} = ZT_{(alt)} + WL$$

Es handelt sich also um eine simple Addition der Zeit , die Sie weg sind,zur momentanen Uhrzeit. In manchen Fällen geht das auch so einfach. Etwa in folgendem Beispiel:

```
ZT$ = "14.20 UHR"          WL$ = "0.30 STD"
ergibt:
ZT = 14.2                  WL = 0.3
wir addieren:
ZT = ZT + WL = 14.2 + 0.3 = 14.5
ZT = 14.5
das entspricht:
14.50 UHR                  Stimmt!
```

Daß das allerdings nicht immer so einfach funktioniert, wird sofort klar, wenn wir uns verdeutlichen, daß wir die Zeiten sexagesimal eingeben, der Rechner die Variablen ZT und WL aber als Dezimalzahlen behandelt, die in unserem Programm nun eine Uhrzeit repräsentieren sollen. Die beiden nächsten Beispiele zeigen, was dabei schief gehen kann:

```
ZT$ = "12.50 UHR"          WL$ = "0.50 STD"
ergibt:
ZT = 12.5                  WL = 0.5
wir addieren:
ZT = ZT + WL = 12.5 + 0.5 = 13
das entspricht:
13.00 UHR                  Falsch!
```

ein anderer Fall:

```
ZT$ = "14.50 UHR"          WL$ = 1.30 STD"
ergibt:
ZT = 14.5                  WL = 1.3
wir addieren wieder:
ZT = ZT + WL = 14.5 + 1.3 = 15.8
das entspricht:
15.80 UHR                  kompletter Unsinn!!
```

Sie erkennen sofort: So geht's nicht! Eine Möglichkeit, wie's geht, können Sie in den Zeilen 55 - 90 mitverfolgen. Ich habe hier diesen Weg gewählt, weil er Sie mal wieder dazu zwingt, genau den Weg der Variablen und ihre inhaltlichen Metamorphosen zu verfolgen. Gleichzeitig gibt diese Methode Ihnen Gelegenheit, die INT-Funktion in Aktion zu sehen. Bevor Sie weiterlesen, versuchen Sie doch, aus dem Programmlisting schlau zu werden.

Gehen wir jetzt gemeinsam die Programmzeilen durch:
Zeile 55 und 60: Hier werden den Variablen ZT und WL die Nachkommastellen, also die Minutenanteile, abgetrennt, und den Hilfsvariablen Z1 und W1 zugewiesen. Ein Beispiel verdeutlicht den Vorgang:

```
ZT$ = "12.50 UHR"          ZT = 12.5
INT(ZT) = 12               100*INT(ZT) = 1200
100*ZT = 1250
100*ZT - 100*INT(ZT) = 1250 - 1200 = 50 (=Minutenanteil)
```

Zeile 65: Die Minutenanteile von WL und ZT, also W1 und Z1, werden addiert und einer weiteren Hilfsvariablen Z2 zugewiesen.
Zeilen 70 und 85: Ist Z2 kleiner als 60, genügt es, Z2/100 zu bilden, um die korrekte Minutenzahl für das Ergebnis hinter dem Komma zu erhalten.
Zeile 75: Ist Z2 jedoch größer als 60, würden wir unsinnige Minutenangaben erhalten. Z2 muß also in einen Stunden- und einen Minutenteil aufgespalten werden. Hier ein Beispiel:

```
Z2 = 90                          Z2 - 60 = 30
(Z2-60)/100 = 0.3
((Z2-60)/100)+1 = 1.3
entspricht:
1Std 30 Min = 90 Min             Stimmt!
```

Zeile 90: Hier kümmern wir uns jetzt um die Stundenanteile von ZT und WL. Diese sind einfacher zu erhalten, als die Minutenanteile. Es genügt, den Integer zu nehmen, und diese beiden Integeranteile dann zu addieren, also:

```
ZT = INT(ZT) + INT(WL)
```

Wir sollten allerdings den mühsam ermittelten Minutenwert nicht unterschlagen:

```
ZT = INT(ZT) + INT(WL) + Z2
```

Zeile 145: Die Variable ZT$, die im vorherigen Programm hier stand, wird nun durch ZT ersetzt.
Zeile 143: Was ist denn das für ein Ding? Es handelt sich um eine Anweisung, die Sie immer dann verwenden, wenn Sie mit der Art und Weise, wie Ihr Rechner die Ergebnisse ausdruckt nicht zufrieden sind und Ihm da etwas Manieren beibringen wollen. In unserem Fall zaubert die Anweisung

```
USING
```

uns die Null hinters 14.3, so daß ein ordentliches 14.30 UHR daraus werden kann. Mittels der seltsamen "###.##"-Zeichen teilen Sie dem Rechner mit, daß er immer zwei Stellen vor und zwei Stellen nach dem Komma ausdrucken soll, auch wenn er nur eine hat. Fehlende Stellen hinterm Komma werden mit Nullen aufgefüllt, vor dem Komma werden sie freigelassen. Das dritte "#" vor dem Dezimalpunkt muß in unserem Falle stehen, denn ein Platz vor dem Komma wird immer fürs Vorzeichen reserviert. USING "##.###" würde also bedeuten: 1 Stelle vor und drei nach dem Komma sollen ausgedruckt werden. Mit den Stellen vor dem Komma müssen Sie aufpassen, denn wenn eine Variable hier mehr aufweist, als für die Ausgabe vorgesehen sind, gibts'nen ERROR.

Aufgabe 12.1

Ändern Sie das Programm TUERSCHILD II in den Zeilen 55 - 90 so ab, daß Sie die Rückkehrzeit nicht mehr umständlich mit INT-Rechnungen, sondern mit DMS und DEG ermitteln.

12.2 Die Textfunktion LEN

Mit Hilfe von LEN können Sie in einem Programm die Länge (engl.: **LEN**gth) einer Textvariablen, einer Textkonstanten oder eines Textausdruckes ermitteln. Unter der Länge einer Textvariablen versteht man die Anzahl der Zeichen, aus der sie besteht, natürlich inclusive der Leerstellen, Kommas usw. Da auf dem PC-1500 A eine Textvariable (vorerst) nicht mehr als 16 Zeichen beinhalten darf, gilt für ihn:

```
0 <= LEN X$<= 16
```

Das ist selbstverständlich von Computer zu Computer verschieden und von der jeweils zulässigen Zeichenzahl einer String-Variablen abhängig. Um ein Gefühl für die LEN-Funktion zu bekommen, testen Sie sie doch mal im Direktmodus:

```
Eingabe:                    Anzeige:

A$ = "1."
B$ = "HALTE"
C$ = "STELLE"

LEN(A$)                     2
LEN(B$)                     5
LEN(C$)                     6
LEN(A$+B$+C$)               13
LEN("1. HALTESTELLE")       14
```

Und erneut konnten Sie feststellen: Der Computer übersieht kein Blank! Doch bevor Sie das Ganze jetzt als unnötige Spielerei oder gar blanken Unsinn abtun, wollen wir ein letztes mal unser Programm TUERSCHILD hervorkramen und mit Hilfe von LEN eine weitere Verbesserung einbauen. Ist es Ihnen schon mal passiert, daß Sie auf die Frage "WO SIND SIE?" wahrheitsgetreu

```
BEIM ZWEITEN FRUEHSTUECK
```

eingegeben haben und auf dem Türschild der Blödsinn

```
ICH BIN
BEIM ZWEITEN FRU
```

erschien, was Witzbolde schnell als zweiten Frühling interpretierten? Nein? Glück gehabt. Warum das geschen kann, ist kein großes Rätsel: Die Eingabe hatte mehr als 16 Zeichen. Da es jedoch lästig ist, sich immer erst das komplette Türschild ausdrucken zu lasssen, bevor man solche Fehler bemerkt, haben Sie beschlossen, Ihr Rechner soll Sie doch bitte schön gleich nach der Eingabe auf den Fauxpas aufmerksam machen. Wie können Sie Ihm das beibringen?

Nun, wozu haben wir die LEN-Funktion? Es muß also nur noch nach der Eingabe von WO$ überprüft werden, ob LEN(WO$) größer als 16 ist. Ist das der Fall, lassen wir Konsequenzen folgen, nämlich die Anzeige "EINGABE ZU LANG" auf dem Display.

Probieren wir's doch mal und ergänzen unser Programm wie folgt:

```
23 : IF LEN(WO$) > 16 THEN GOTO 17
```

Damit das geschehen kann, muß auch eine Zeile 17 existieren:

```
17 : WAIT 100
18 : PRINT "EINGABE ZU LANG"
```

Und da wir das nicht bei jedem Programmlauf automatisch dargeboten bekommen wollen, programmieren wir:

```
16 : GOTO 20
```

Wenn Sie es nicht schon beim Programmieren festgestellt haben, dürfte Ihnen spätestens bei den ersten Probeläufen aufgefallen sein, daß da was nicht stimmen kann. Egal was Sie auch eingeben, nie meldet Ihr Rechner

```
EINGABE ZU LANG
```

Der Fehler liegt in Zeile 23. Die Bedingung LEN(WO$) > 16 wird nie erfüllt, da Ihnen der Rechner den String automatisch auf 16 Zeichen beschneidet. Damit wir eine derartige Kontrolle überhaupt ausführen können, müssen wir uns darauf beschränken, Ortsangaben mit 15 oder weniger Zeichen zu machen. Jetzt können wir in Zeile 23 eine neue Bedingung setzen:

```
23 : IF LEN(WO$) = 16 THEN GOTO 17
```

Wenn Sie dies geändert haben, klappt's. Probieren Sie's doch aus. In der Abbildung 8 finden Sie den Anfang des geänderten Programms nochmal als Listing und daneben den zugehörigen Programmablaufplan.

12.3 Die Funktionen LEFT$, RIGHT$ und MID$

Die mangelne Rechentauglichkeit der String-Variablen hätte ihnen ein bescheidenes Schattendasein beschert, wenn sich nicht ein paar schlaue Leute die in der Überschrift aufgeführten Textfunktionen ausgedacht hätten. Mit der Hilfe von LEFT$, RIGHT$ und MID$ können Sie Ihre Strings ganz schön manipulieren. Es handelt sich dabei nicht etwa um politische Manipulationen, obwohl die Funktionsnamen so klingen, als könnte man damit prüfen, ob ein String linkslastig oder rechtsextrem ist oder sich aber der politischen Mitte zugehörig fühlt. Es wird alle Verfassungsschützer enttäuschen, aber es handelt sich bei unserem Dreiergespann nur um nüchterne, wohldefinierte Funktionen, deren Bedeutung ich Ihnen nun im einzelnen erläutern möchte.

```
LEFT(X$,Y)
```

schneidet dem String X$ eine bestimmte Anzahl (Y) von Zeichen am linken Ende ab und stellt sie zur Weiterverarbeitung zur Verfügung. Anstelle von einer Textvariablen kann auch ein Textausdruck oder eine Text-

```
 4:REM **********
 5:REM TUERSCHILD
       ----II----
 6:REM **********
10:CLS
15:CLEAR
16:GOTO 20
17:WAIT 100
18:PRINT "EINGABE
    ZU LANG"
20:INPUT "WO SIND
    SIE?";WO$
23:IF LEN (WO$)=1
   6THEN GOTO 17
30:INPUT "WIE SPA
   ET IST ES?";ZT
```

Abb 8a

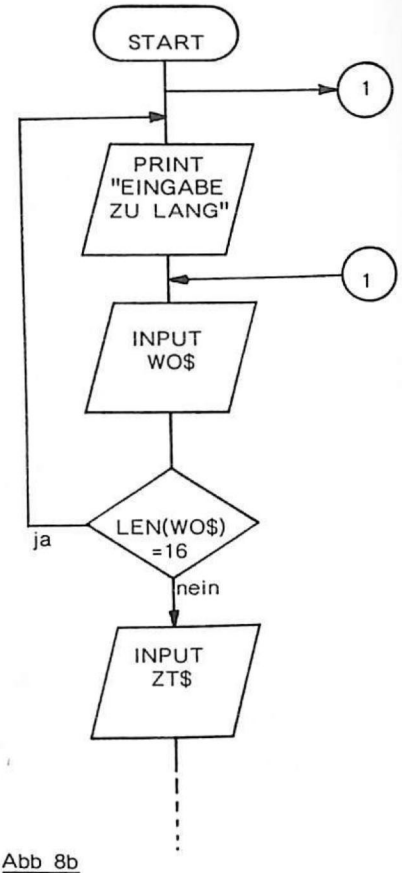


Abb 8b

Abbildung 8: Teil-Listing und Programmablaufplan vom Programm TUERSCHILD II

konstante stehen. Sehen wir uns ein Beispiel an:

```
10 : LET X$ = "ROSI FREI"
20 : LET A$ = LEFT$(X$,4)
30 : PRINT A$
```

Als Ergebisausdruck erhalten Sie für A$ die ersten vier Zeichen von X$:

ROSI

Daß anstelle einer Textvariablen in der Funktion LEFT$ eine Textkonstante als Argument gewählt wird, dürfte kaum vorkommen, weil Sie diese

einfacher "von Hand" abschneiden, bevor Sie sie einer Variablen zuweisen. Wenden wir uns nun der Funktion

RIGHT$(X$,Y)

zu. Sie hat die selbe Aufgabe wie LEFT$, nur, daß sie eine Anzahl (Y) Buchstaben von rechts gezählt dem String abknipst. Versuchen wir das gleiche Beispiel wie vorhin, nur eben mit RIGHT$ statt LEFT$:

```
10 : LET X$ = "ROSI FREI"
20 : LET A$ = RIGHT$(X$,4)
30 : PRINT A$
```

Sie ahnen das Ergebnis schon voraus? Hier ist es:

FREI

Das ist doch alles in allem keine komplizierte Sache, oder? Ein wenig komplexer wird das Ganze mit der dritten Funktion im Bunde:

MID$(X$,Z,Y)

Es hat sich offensichtlich noch eine dritte Größe ins Argument geschlichen. Gleich geblieben ist, daß Sie sich auch mit dieser Funktion einen Teil des Strings X$ zur weiteren Verwendung herauspicken können. Sie sind nun jedoch nicht mehr auf die linken oder rechten Endstücke der Zeichenkette beschränkt, sondern Sie können jetzt in die Vollen greifen. Mit Y geben Sie weiterhin an, wie lang Ihr Segment sein soll, das neu hinzugekommene Z definiert nun, wo dieses Stückchen denn beginnen soll, und zwar von links gerechnet. Sie haben Recht, ohne ein Beispiel kann das ja keiner verstehen:

```
10 : LET X$ = "ROSI-MAUSI FREI"
20 : LET A$ = MID$(X$,6,5)
30 : PRINT A$
```

Wenn Sie genau mitgezählt haben, kennen Sie schon den Ausdruck. Er beginnt mit der 6. Stelle und ist 5 Stellen lang. Richtig:

MAUSI

nennt unser Computer neckisch unsere, uns schon lieb gewordene, Frau Frei. Etwaige Ähnlichkeiten mit lebenden Personen sind natürlich rein zufällig, versteht sich.

Was bisher noch wie eine muntere Spielerei aussieht, wird immer dann zu einer unverzichtbaren Möglichkeit der Stringmanipulation, wenn Sie größere Datenmengen abgespeichert haben und Daten nach bestimmten Gesichtspunkten aus Dateien auswählen oder sortieren möchten. Darauf kommen wir später ausführlicher noch zurück. Anhand eines kleinen, grob vereinfachten Beispiels kann ich Ihnen das hier jedoch einmal illustrieren.

Wollen Sie etwa aus einer Adressenliste die Stuttgarter Adressen heraus-

fischen, könnte das unter Zuhilfenahme des folgenden kleinen Programmbruchstückes geschehen:

```
10 : INPUT "PLZ, ORT";O$
20 : LET PZ$ = LEFT$(O$,4)
30 : IF PZ$ = "7000" THEN GOTO 60
40 : PRINT " ADRESSE NICHT AUS STUTTGT"
50 : GOTO 10
60 : PRINT " ADRESSE IST AUS STUTTGART"
70 : GOTO 10
```

Die Eingabe in Zeile 10 muß also in postalisch üblicher Form geschehen, also etwa 7000 STUTTGART 40. Sie können es auch noch weiter treiben, und sich entschließen, nur die Adressen aus Stuttgart-Bad Cannstatt gelten zu lassen. Ergänzen wir also das Programm:

```
 5 : WAIT 100
70 : LET OT$ = RIGHT$(O$,2)
80 : IF OT$ = "50" THEN GOTO 110
90 : PRINT "ABER NICHT AUS CANNSTATT"
100 : GOTO 10
110 : PRINT "UND AUS BAD CANNSTATT"
120 : GOTO 10
```

Damit Sie den Vorgang in Zeile 80 verstehen, müssen Sie natürlich wissen, daß Cannstatt die Postanschrift 7000 STUTTGART-50 hat.

Ein anderer Bereich, wo man mit String-Funktionen ganz nette Effekte erzielen kann, ist die graphische Gestaltung von Ausdrucken. Ich möchte Ihnen hierfür ein kleines Beispiel geben und in Aufgabe 12.2 finden Sie noch mehr Anschauungsmaterial, dessen Entstehung Sie sich dann allerdings selbst erklären sollten.

```
B
BA
BAS
BASI
BASIC
BASIC-
BASIC-L
BASIC-LE
BASIC-LEH
BASIC-LEHR
BASIC-LEHRB
BASIC-LEHRBU
BASIC-LEHRBUC
BASIC-LEHRBUCH
```

Das zugehörige Programm bekommen Sie sofort nachgeliefert, doch überlegen Sie zuerst, ob Sie von selbst auf die Lösung kommen.

```
20:A$="BASIC-LEHR
   BUCH"
30:FOR I=1TO 14
40:LET B$=LEFT$ (
   A$, I)
50:LPRINT B$
60:NEXT I
```

Wenn Sie sich vorstellen, Sie würden wie ein Computer das Programm abarbeiten und müßten 14 mal die Schleife in den Zeilen 30 - 60 durchlaufen, verstehen Sie das Prinzip des Programms am schnellsten. Von zentraler Bedeutung ist, was der Wert I in Zeile 40 anrichtet.

Nachdem Ihnen das Programm klar geworden ist, dürften Sie mit den Übungsaufgaben eigentlich auch keine Schwierigkeiten mehr haben. Sie sind alle nach dem gleichen Schema aufgebaut. Nur beim 4. Beispiel müssen Sie etwas umdenken. Ein Hinweis: Auch hier wird Zeile für Zeile ausgedruckt.

Aufgabe 12.2

Sehen Sie sich die vier Ausdrucke genau an, und versuchen Sie ein Programm zu schreiben, das diese Ausdrucke erzeugt (Pro Beispiel ein Programm)

a)

```
H
CH
UCH
BUCH
RBUCH
HRBUCH
EHRBUCH
LEHRBUCH
-LEHRBUCH
C-LEHRBUCH
IC-LEHRBUCH
SIC-LEHRBUCH
ASIC-LEHRBUCH
BASIC-LEHRBUCH
```

b)

```
H    BASIC-LEHRBUCH
CH   BASIC-LEHRBUC
UCH  BASIC-LEHRBU
BUCH BASIC-LEHRB
RBUCH BASIC-LEHR
HRBUCH BASIC-LEH
EHRBUCH BASIC-LE
LEHRBUCH BASIC-L
-LEHRBUCH BASIC-
C-LEHRBUCH BASIC
IC-LEHRBUCH BASI
SIC-LEHRBUCH BAS
ASIC-LEHRBUCH BA
BASIC-LEHRBUCH B
```

c)

```
L
LE
-LE
-LEH
C-LEH
C-LEHR
IC-LEHR
IC-LEHRB
SIC-LEHRB
SIC-LEHRBU
ASIC-LEHRBU
ASIC-LEHRBUC
BASIC-LEHRBUC
BASIC-LEHRBUCH
```

d)

```
BB B  B  B  B
AA A  A  A  A
SS S  S  S  S
II I  I  I  I
CC C  C  C  C
-- -  -  -  -
LL L  L  L  L
EE E  E  E  E
HH H  H  H  H
RR R  R  R  R
BB B  B  B  B
UU U  U  U  U
CC C  C  C  C
HH H  H  H  H
```

13 . K a p i t e l

EIN FELD WIRD BESTELLT
- Indizierte Variablen und Variablenfelder -

13.1 Alphabetisches Sortieren

Haben Sie sich schon mal gewünscht, Ihr Computer könnte für Sie eine lange Liste von Namen, Buchtitel oder ähnliches in eine alphabetische Ordnung bringen? Daß das im Prinzip gehen müßte, wissen Sie, seit wir uns mit dem ASCII-Code beschäftigt haben, und Sie in unserem Kurzprogramm ALPHATEST erfahren konnten, daß der Rechner sehr wohl weiß, daß MEIER <MEYER ist. Wie Sie allerdings vorgehen müssen, wenn Ihre Liste aus mehr als nur zwei Namen besteht, haben wir dabei nicht überlegt.

Wir benötigen dazu einen Sortieralgorithmus, den ich Ihnen im folgenden vorstellen möchte, bevor wir uns dem eigentlichen Thema dieses Kapitels, den Variablenfeldern, widmen. Warum wir nicht gleich zur Sache kommen? Nun, beim Sortieren werden Ihnen nochmals deutlich die Eigenschaften der Ihnen bislang bekannten Variablen vor Augen geführt und Sie sehen vor allem die Begrenztheit ihrer Einsatzmöglichkeiten.

Erinnern Sie sich: Das Gleichheitszeichen beim Befehl LET A = 5 unterscheidet sich wesentlich vom mathematischen Gleichheitszeichen. Es soll nicht ausdrücken, daß der Ausdruck links und rechts davon identisch sind, sondern es weist der linksstehenden Variable (hier A) den rechtsstehenden Wert (hier 5) zu. Anstelle des Gleichheitszeichens stünde also besser ein kleiner Pfeil, etwa so:

LET A ← 5

Da man in der Regel das Befehlswort LET sogar weglassen darf, und einfach A=5 schreiben kann, sind Verwirrungen, im wahrsten Sinn des Wortes, vorprogrammiert. Um für die weiteren Ausführungen gewappnet zu sein, stellen Sie sich am besten jede Variable (egal ob numerisch oder String) als kleines Fach in einem großen Regal vor, das mit einem Namensschild (etwa A, B, C... oder A$, B$...) versehen ist und nur darauf wartet, mit einem Wert (Zahl oder Text) gefüllt zu werden. Dabei gilt: Wenn Sie vorne ins Regalfach einen Wert hineinstecken, fällt der Wert, der eventuell bisher darin lag, hinten raus und ist damit für immer verschwunden.

Soviel als Gedächtnisstütze, lassen Sie uns jetzt aber mit dem Ordnen unserer Namensliste beginnen. Für den Anfang geben wir uns bescheiden, und sortieren nur drei Namen. Programmieren Sie:

```
 10 : LET N1$ = "ESCHER"
 20 : LET N2$ = "GOEDEL"
 30 : LET N3$ = "BACH"
210 : PRINT N1$
220 : PRINT N2$
```

```
230 : PRINT N3$
```

Wenn Sie über einen Drucker verfügen, ersetzen Sie PRINT durch LPRINT und Sie erhalten eine übersichtliche Liste auf dem Papier. Aus der Zeilennummerierung können Sie ersehen, daß zwischen den Zuweisungen und der Ausgabe also die Sortierarbeit geleistet werden soll. Es soll bewirkt werden, daß uns in den Zeilen 210-230 eine alphabetisch sortierte Liste ausgegeben wird. Am Ende muß also gelten:

```
N1$ = "BACH"
N2$ = "ESCHER"
N3$ = "GOEDEL"
```

Das bedeutet, daß der Inhalt der Variablen N1$, N2$ und N3$ ausgetauscht worden ist. Lassen Sie uns das jetzt mal im Programm versuchen. Als erstes suchen wir den Namen, der im Alphabet ganz vorne steht und nachher N1$ sein soll. Wie wärs damit:

```
110 : IF N2$ <N1$ THEN LET N1$ = N2$
120 : IF N3$ <N1$ THEN LET N1$ = N3$
```

Wir sind aufs Ergebnis gespanntund tippen gleich RUN ein:

```
BACH
GOEDEL
BACH                          Hoppla!
```

Wir haben es also geschafft, Herrn Bach an die ihm gebührende erste Stelle zu hieven, haben dabei aber Herrn Escher "aus dem Regal geschupst" und können ihn nicht wiederfinden. Wenn wir's richtig machen wollen, müssen wir Herrn Escher in ein Reserveregal stellen, bevor wir Herrn Bach in das Regel Nr. N1$ befördern. Danach können wir ihn ja in das freigewordene[1] Regal Nr. N3$ von Herrn Bach zurücksetzen. Programmiert wird das so:

```
110 : IF N1$ <N2$ THEN GOTO 130
120 : LET RE$ = N1$ : N1$ = N2$ : N2$ = RE$
130 : IF N1$ <N3$ THEN GOTO 210
140 : LET RE$ = N1$ : N1$ = N3$ : N3$ = RE$
```

Wir erhalten das Ergebnis:

```
BACH
GOEDEL
ESCHER                        Na also!
```

Es ist uns offensichtlich gelungen, Herrn Bach an die erste Stelle rücken, ohne dabei einen der anderen Herren unter den Tisch fallen zu lassen.

[1] "freigeworden" ist im Prinzip eine falsche Bezeichnung, denn auch wenn man einer Variablen B den Wert von A zuweist (B=A), bleibt in der Variablen A der Wert so lange weitergespeichert, bis ein neuer darübergeschrieben wird. Es gibt also keine leeren Regale, höchstens solche, die bewußt 0 gesetzt werden.

Verdeutlichen wir uns, wie das erreicht wurde:

Zeile 110: Es wird geprüft, ob ESCHER kleiner als GOEDEL ist. Das trifft zu, also erfolgt ein Sprung nach Zeile 130
Zeile 130: Ist ESCHER auch kleiner als BACH? Nein, also gehts weiter in Zeile 140.
Zeile 140: Hier wird die Position vertauscht. ESCHER kommt in die Reservevariable RE$, BACH bekommt ESCHERs füheren Platz (N1$) und zuletzt wird ESCHER auf BACHs Platz (N3$) geschoben.

Bevor wir jedoch allzu stolz auf unser Ergebnis sind, dürfen wir einen Schönheitsfehler in unserer Liste nicht übersehen: Der zweite und der dritte Platz sind nicht in alphabetischer Folge. Schaffen wir schnell Abhilfe:

```
150 : IF N2$ <N3$ THEN GOTO 210
160 : LET RE$ = N2$ : N2$ = N3$ : N3$ = RE$
```

In Zeile 130 müssen wir noch die Sprungadresse ändern:

```
130 : IF N1$ <N3$ THEN GOTO 150
```

Schnell ein Blick auf das erzielte Ergebnis:

```
BACH
ESCHER
GOEDEL
```

Ich glaube, wir können zufrieden sein. Wenn Sie jetzt noch ein übriges tun, und die LET-Befehle in 10-30 durch INPUTs ersetzen, können Sie jede beliebige Liste von drei Strings alphabetisch sortieren lassen. Da der Austauschalgorithmus recht häufig gebraucht wird, ist er hier gleich nochmal zu bewundern:

Austauschen zweier Variablenwerte A und B:
LET R = A : A = B : B = R

Und damit Sie das auch nie wieder vergessen, trainieren Sie es bitte mit der folgenden Übungsaufgabe:

Aufgabe 13.1

Versuchen Sie, das Sortierprogramm so zu erweitern, daß es befähigt wird, eine 4-Wort-Liste alphabetisch zu ordnen.

Die Lösung dürfte Ihnen eigentlich nicht schwer gefallen sein. Mich würde es jedoch nicht wundern, wenn Sie das Gefühl beschlichen hätte, daß Ihnen Ihr Computer mehr (Programmier-) Arbeit macht, als er Ihnen abnimmt. Waren zum Sortieren von drei Variablen noch sechs Befehlszeilen ausreichend, benötigen wir bei vier Variablen schon 12 Zeilen. Bei fünf Variablen wären schon 20 Zeilen erforderlich. Allgemein ausgedrückt

braucht man zum Sortieren von N Variablen N*(N-1) Befehlszeilen. Das würde bedeuten, um die Telefonliste Ihrer 36 Bekannten zu ordnen, müßten Sie 1260 programmieren, wobei die je 36 INPUT- und PRINT-Anweisungen noch nocht berücksichtigt sind.

Das kann unmöglich der Programmierkunst letzter Stand sein! Recht haben Sie, und ich denke, Sie sind jetzt motiviert genug, sich mit den indizierten Variablen zu beschäftigen, die uns eine ganze Menge Arbeit abnehmen können.

13.2 Die indizierten Variablen

Bei unserem Sortierprogramm mußten wir einige wenige Befehlstypen ständig neu programmieren, wobei sich nur der Variablenname veränderte. Diese Arbeit könnte man sich sparen, wenn der Rechner in einer FOR ... NEXT-Schleife den Befehl immer wieder ausführen könnte und dabei selbsttätig den jeweils richtigen Variablennamen einsetzte. Nun, wenn wir indizierte Variablen verwenden, geht das.

Indizierte Variablen haben hinter dem Variablennamen in Klammern einen Index. Dieser Index kann entweder eine Zahl oder, und das ist der springende Punkt, wieder eine Variable sein. Am einfachsten erläutere ich Ihnen das an unserem Beispielprogramm. Die vier zu sortierenden Namen hatten wir bisher in die Variablen N1$, N2$, N3$ und N4$ eingelesen. Im Programm sah das so aus:

```
10 : INPUT N1$
20 : INPUT N2$
30 : INPUT N3$
40 : INPUT N4$
```

Wir wollen sie jetzt durch indizierte Variablen ersetzen, die die Namen N$(1), N$(2), N$(3) und N$(4) erhalten sollen. Der neue Variablentyp erlaubt uns eine Umstrukturierung des INPUT-Teils:

```
20 : FOR E = 1 TO 4
30 : INPUT N$(E)
40 : NEXT E
```

Sie sehen, in der Zeile 30 tritt anstelle der einzelnen Indizes 1-4 einfach die Schleifenvariable E, so daß wir nur noch eine INPUT-Anweisung programmieren mußten. Da sich E mit jedem Schleifendurchlauf um 1 erhöht, tauscht der Rechner also selbsttätig die Variablennamen aus. Genau das hatten wir uns ja gewünscht.

Doch bevor Sie jetzt mit Eifer darangehen, Ihr Programm mit dem neuen Variablentyp zu bestücken, ist noch etwas zu klären. Sollten Sie Ihr Programm mit dem neuen INPUT-Teil nämlich mal probelaufen lassen, setzt's eine wüste Fehlermeldung:

```
ERROR 6 IN 30
```

steht da beispielsweise auf dem PC-1500 A. Blättern Sie im Handbuch nach, finden Sie die Erklärung: "Eine Feldvariable wurde ohne DIM-Instruktion benutzt." Was soll denn das heißen, Feldvariable? Und was ist eine DIM-Instruktion? Ich werd's Ihnen erklären.

Indizierte Variablen haben die Eigenart, daß auf ein und denselben Variablennamen (in unserem Beispiel N$) mehrere Werte gespeichert werden sollen. Unter einem Namen werden eine ganze Liste von Werten zusammengefaßt, die mittels der Indizes durchnummeriert sind. Eine solche Liste von Textkonstanten oder numerischen Konstanten nennt man ein Feld oder auch Array. Die indizierte Variable, mit der wir es in unserem Beispiel zu tun haben, stellt ein eindimensionales Feld dar. Hatten wir bisher eine Variable in unserem Programm verwendet, mußten wir nur darauf achten, daß der Name zulässig war und daß wir, im Falle einer Textvariablen, nicht zu viele Zeichen hineingeschrieben haben. Den Platz im Speicher des Rechners, in dem die Variablenwerte aufbewahrt werden sollten, hat sich der Computer selbst besorgt, und zwar in dem Moment, in dem er erstmals auf den neuen Variablennamen getroffen ist. Für die Variablen A - Z und A$ - Z$ ist im Standardvariablenspeicher sogar vorsorglich Platz reserviert.

Bei indizierten Variablen verhält sich das anders. Hier kann der Rechner in dem Moment, wo er erstmals eine derartige Variable im Programm antrifft, nicht feststellen, wieviele Werte unter diesem Namen im Speicher abgelegt werden sollen. Gibt es also, wie in unserem Beispiel, nur vier Variablen mit dem Namen N$(...) oder hat unsere Liste vielleicht den Umfang eines Adressbuchs, so daß 50 Werte mit den Namen N$(1) - N$(50) gespeichert werden sollen? Dann müßte im Hauptspeicher auch entsprechend viel Platz reserviert werden.

Eben diese Platzreservierung müssen deshalb wir im Programm vornehmen. Dies geschieht mit der bereits erwähnten Instruktion

DIM

die das Feld für die Variable N$ **DIM**ensioniert, also Platz schafft für das, was so alles unter dem Namen N$ auftauchen wird. In unserem Programm wollen wir das nun gleich erledigen:

```
10 : DIM N$(4)
```

Wir setzen die DIM-Anweisung immer an den Anfang des Programms, damit wir sicher sein können, daß im Programm nicht eine indizierte Variable auftaucht, für die noch kein Feld existiert. Die 4 in der Klammer gibt an, daß wir beabsichtigen, 4 Strings in das Feld N$(...) einzulesen. Die DIM-Anweisung muß nur einmal gegeben werden und behält für den gesamten Programmlauf ihre Gültigkeit.

Jetzt möchten Sie aber Ihre neue Errungenschaft endlich in unser Sortierprogramm einbauen. Dann will ich Sie nicht länger warten lassen. Das Programm besteht aus drei Teilen:

1. Eingabeteil
2. Sortierteil
3. Ausgabeteil

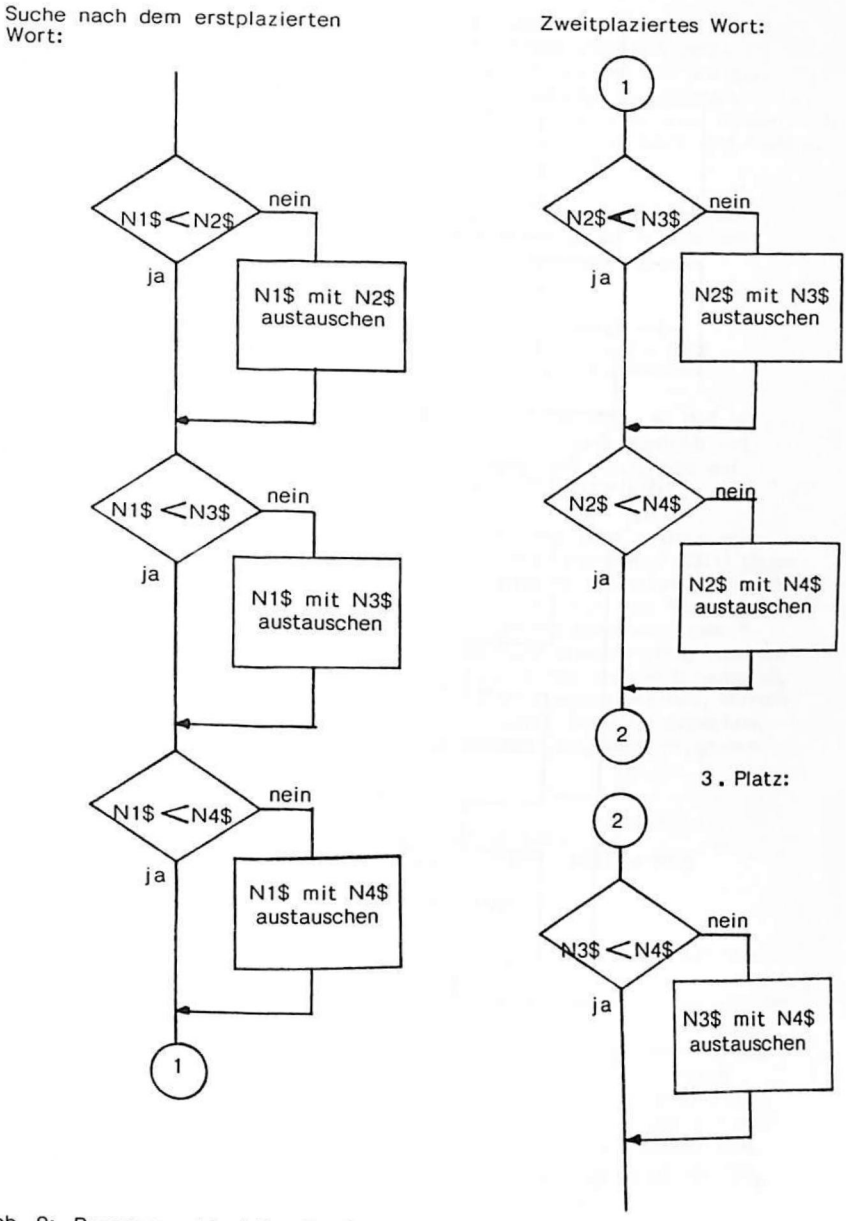


Abb. 9: Programmablaufplan Sortierprogramm (alt), Sortierteil

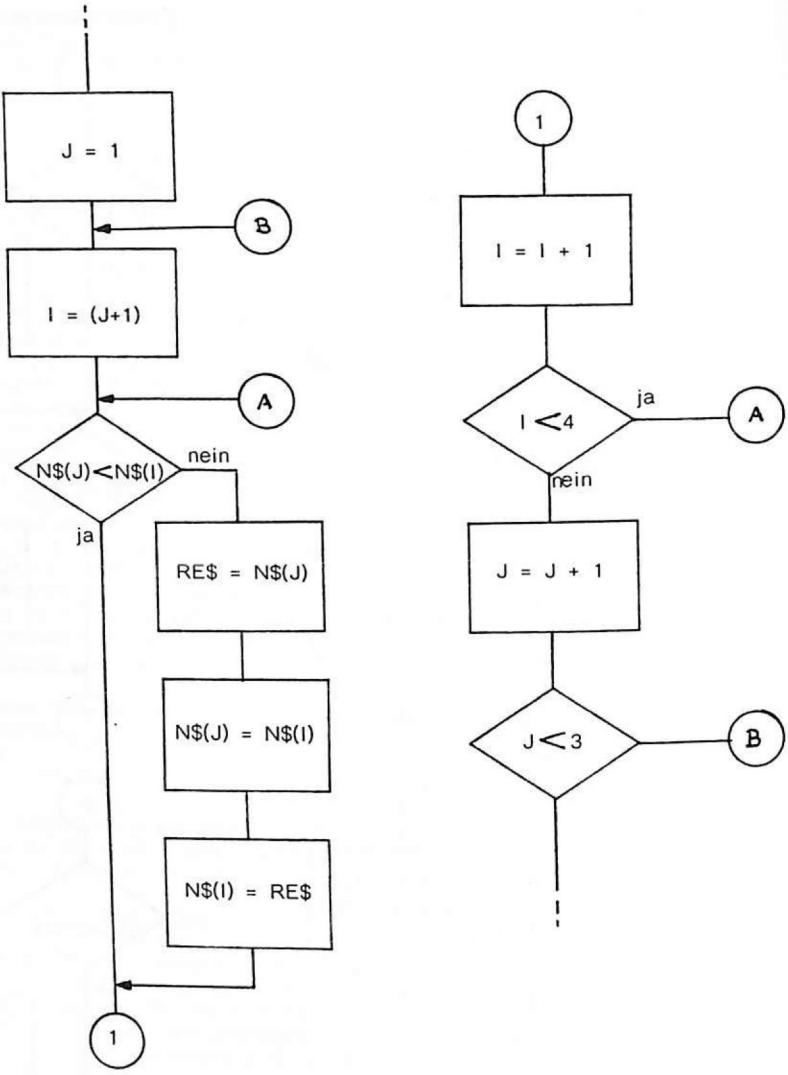


Abb. 10: PAP des Sortierteils vom Sortierprogramm (neu)

Der dritte Teil wird analog zum ersten Teil verändert, es steht anstelle der INPUT-Anweisung nur eine PRINT-Anweisung. Mit dem zweiten Teil, dem Kernstück unseres Programms, werden wir uns etwas näher beschäftigen müssen. In Abbildung 9 sehen Sie nochmals den Aufbau des alten Sortierteils in einem Programmablaufplan. Diesen umständlich zu programmierenden Ablauf wollen wir mit Hilfe von FOR ... NEXT-Schleifen straffen.

Beginnen wir mit der Suche nach dem erstplazierten Wort. In Abb. 9 können Sie sehen, daß wir dreimal vergleichen und ggf. vertauschen müssen. Demnach sind drei Schleifendurchläufe erforderlich.

```
110 : FOR I = 2 TO 4
120 : IF N$(1) <N$(I) THEN GOTO 140
130 : LET RE$ = N$(1) : N$(1) = N$(I) : N$(I) = RE$
140 : NEXT I
```

Der Index I nimmt nacheinander die Werte 2, 3 und 4 an, so daß in Zeile 120 N$(1) einmal mit N$(2) dann mit N$(3) und schließlich mit N$(4) verglichen wird. In Zeile 130 finden dann, falls nötig, die entsprechenden Tauschprozesse statt.

Um das zweitplazierte Wort zu finden, könnten wir jetzt einfach eine entsprechend umgeänderte Schleife anhängen. Man müßte nur N$(1) gegen N$(2) tauschen und 'FOR I = 3 TO 4"schreiben. Eine derartige Aneinanderkettung von Schleifen mag noch angehen, wenn wir nur vier Worte sortieren wollen, bei längeren Listen wäre der Programmieraufwand jedoch wieder zu umfangreich. Lassen Sie es uns deshalb gleich richtig machen. Bei der Suche nach dem 2. und 3. Platz läuft ja das gleiche Schema ab, wie für den ersten. Da wir also wieder dreimal dasselbe machen, können wir doch unsere Austausch-Schleife in eine weitere Schleife einpacken, und einfach 3-mal ablaufen lassen. Dabei entsteht folgendes Programm:

```
100 : FOR J = 1 TO 3
110 : FOR I = (J+1) TO 4
120 : IF N$(J) <N$(I) THEN GOTO 140
130 : LET RE$ = N$(J) : N$(J) = N$(I) : N$(I) = RE$
140 : NEXT I
150 : NEXT J
```

Was hat sich alles verändert? Wir haben den Schleifenanfang (und damit die Anzahl der Schleifendurchläufe) in Zeile 110 von der Schleifenvariablen J der Außenschleife abhängig gemacht. Von J hängt auch ab, welches N$(...) in Zeile 120 verglichen und in 130 vertauscht werden soll. Wenn Sie nicht gleich verstehen, was in den Schleifen abläuft, spielen Sie mal wieder selbst Computer und gehen Sie die Schleifen mit den jeweils wechselnden Indizes durch und vergleichen das mit unserem alten Programm. Sie werden sehen, es passiert genau das selbe, nur daß wir eben mit sechs Programmzeilen anstatt zwölf wie bisher ausgekommen sind. Ein Vorteil, der sich mit der Länge unserer Liste multipliziert. In Abb. 10 können Sie den neuen Sortierteil im PAP nochmal bewundern.

Vertiefen Sie Ihr neu gewonnenes Wissen jetzt noch, indem Sie sich munter an die Übungsaufgaben zu diesem Teil heranmachen. Viel Erfolg!

Aufgabe 13.2

Verändern Sie das Programm so, daß Sie eine beliebig lange Liste von Wörtern damit alphabetisch ordnen können. Die Listenlänge soll in Zeile 5 so abgefragt werden:

```
5 : INPUT "LAENGE DER LISTE?";L
```

Aufgabe 13.3

Machen Sie Ihren Programmausdruck mittels REM-Anweisungen übersichtlicher.

Aufgabe 13.4

Geben Sie in Ihr Sortierprogramm anstelle von Worten die Zahlenstrings "97", "195", "1875" und "24 045" ein. Als Ergebnis erscheint die geordnete Liste:

```
1875
195
24 045
97
```

Wie erklärt sich das?

Aufgabe 13.5

Wie können Sie erreichen, daß unser Programm Zahlen mathematisch richtig, also nach ihrem Wert sortiert?

13.3 Variablenfelder

In den letzten beiden Unterkapiteln konnten Sie an einer konkreten Aufgabenstellung die Notwendigkeit von indizierten Variablen erkennen. Sie haben auch den Begriff "Variablenfeld" und die damit notwendig gewordene Anweisung DIM kennengelernt. Ich hatte mich jedoch auf die Information beschränkt, die für die Lösung unseres Problems notwendig war. In diesem Abschnitt möchte ich Ihnen einen Überblick über den Umgang mit Variablenfeldern geben und auch nochmal auf die zugehörige Deklaration mit DIM zu sprechen kommen.

Ganz am Rande war zu lesen, daß es sich bei dem von uns benutzten Feld um ein eindimensionales Feld handele. Das hört sich ganz so an, als gebe es da auch noch zweidimensionale Felder. Stimmt!

Nehmen wir einmal an, Sie möchten eine Preisliste für verschiedene Produkte abspeichern. Es soll aber für jedes Produkt nicht nur der Brutto-Verkaufspreis (VK), sondern auch der Netto-VK sowie der Einkaufspreis (EK) mit und ohne Mehrwertsteuer abgespeichert werden. Aus einer eindimensionalen Liste wird so eine zweidimensionale Tabelle. In Tab. 4 sehen Sie ein Beispiel dafür. Jeder Wert, der in der Tabelle steht, hängt also von zwei Faktoren ab: Zum einen von der Ware und zum anderen, welchen Preis wir nachsehen möchten. Wenn wir jetzt eine Variable PR einführen, die diese Preise enthalten soll, müssen wir sie mit zwei Indizes versehen. Tab. 5 zeigt, wie das aus-

	EK 1	EK (+14%) 2	VK 3	VK (+14%) 4
Ware 1	12.50	14.25	17.37	19.80
Ware 2	15.90	18.13	21.75	24.80
Ware 3	23.40	26.68	30.53	34.80
Ware 4	29.70	33.86	39.30	44.80
etc.	etc.	etc.	etc.	etc.

Tab. 4: Beispiel einer Preistabelle

J I	Preis 1	Preis 2	Preis 3	Preis 4
Ware 1	PR(1,1)	PR(1,2)	PR(1,3)	PR(1,4)
Ware 2	PR(2,1)	PR(2,2)	PR(2,3)	PR(2,4)
Ware 3	PR(3,1)	PR(3,2)	PR(3,3)	PR(3,4)
Ware 4	PR(4,1)	PR(4,2)	PR(4,3)	PR(4,4)
...	...	...	...	...
Ware I	...	...	...	PR(I,J)

Tab. 5: Beispiel eines Variablenfeldes zu Preisliste in Tab. 4

sieht. Auch eine solche zweifach indizierte Variable können Sie mit der DIM-Anweisung deklarieren. Wenn wir annehmen, unsere Preisliste enthielte 12 verschiedene Waren, müßten Sie programmieren:

```
10 : DIM PR(12,4)
       ↑ ↑
       | └─Spalten
       └──Zeilen
```

Dabei werden immer zuerst die benötigten Zeilen und dann die erforderlichen Spalten angegeben. Noch was: Mit DIM(12,4) reservieren Sie eigentlich 13 Zeilen und 5 Spalten, denn der Rechner beginnt bei 0 zu zählen und reserviert deshalb die Zeilen 0-12 und die Spalten 0-4. Wenn Sie sich also auch entschließen, Ihre Tabelle mit 0 beginnend durchzunummerieren, sparen Sie Speicherplatz. Aber es geht natürlich auch so wie angegeben. Die unnötigerweise reservierten Plätze PR(0,0)-PR(0,4) sowie PR(1,0)-PR(12,0) bleiben übrigens nicht einfach leer, sondern werden mit dem Wert 0, bzw dem ASCII-Code für 0 bei Textfeldern, gefüllt. Das sollten Sie beim Umgang mit Arrays im Auge behalten, denn das geschieht mit allen Plätzen, die Sie nicht belegt haben.

Nach so viel neuen Variablentypen ist es nun an der Zeit, alle, die Sie bisher kennen, in einer Übersicht zusammenzustellen und zu vergleichen. In Tab. 6 habe ich das für Sie getan.

VARIABLENTYP		ZULÄSSIGE NAMEN	DEKLARATION DES SPEICHERPL.
einfache Variablen	numerisch	A-Z;AA-ZZ;A1-Z9	automatisch
	String	A$-Z$; AA$-Z9$	automatisch
eindimen. Feldvar.	numerisch	A(z) - Z9(z)	DIM A(Z)
	String	A$(z) - Z9$(z)	DIM A$(Z)*K
zweidim. Feldvar.	numerisch	A(z,s) - Z9(z,s)	DIM A(Z,S)
	String	A$(z,s) - Z9$(z,s)	DIM A$(Z,S)*K

z = Index einer Variablen (Zeile)
s = 2. Index einer Variablen (Spalte)
Z = Anzahl der Zeilen
S = Anzahl der Spalten
K = Zeichenkapazität eines Textvariablenelements

Tab. 6: Übersicht über verschiedene Variablentypen.

Die obenstehende Tabelle muß noch durch einige Erläuterungen ergänzt werden, die Sie im Umgang mit Variablenfeldern noch sicherer machen sollen.

1. Es ist bei indizierten Variablen nur eine begrenzte Menge von Indizes erlaubt. Beim PC-1500 A sind das 256, d.h. die Indizes müssen zwischen 0 und 255 liegen.

2. Bei der Dimensionierung von Variablenfeldern kann es geschehen, daß Sie an die Speicherkapazitätsgrenze Ihres Rechners geraten, was vor allem bei den kleinen PCs schnell passiert ist. Ein numerisches Feld von 90 Zeilen und 90 Spalten enthält 8100 Werte und dafür benötigt man ca 64 KByte Platz.

3. Bei der Verwendung von Textfeldern haben Sie bei der Deklaration zusätzlich die Möglichkeit, die Anzahl der Zeichen anzugeben, die ein Element haben darf. Die Beschränkung auf 16 Zeichen können Sie also mit einer DIM-Anweisung aufheben und beim PC-1500 A beispielsweise auf max. 80 Zeichen erweitern. Sie können aber auch Speicherplatz sparen, indem Sie weniger als 16 Zeichen deklarieren. Geben Sie nämlich nichts an, werden automatisch 16 Zeichen pro Element des Textfeldes reserviert und das ist eine Menge,

wenn Sie es mit der Anzahl der Zeilen und Spalten multiplizieren, um Ihren Speicherbedarf zu ermitteln. Kommen Sie bei einem Textfeld von 50 Zeilen und 10 Spalten mit 8 anstelle von 16 Buchstaben pro Wort aus, haben Sie 4000 Byte gespart. Die Deklaration dieses Feldes sähe so aus:

```
10 : DIM N$(50,10)*8
            |  |  |
            |  |  Zeichen pro Element
            |  Spalten
            Zeilen
```

4. Variablenfelder können auch im Programm mit Hilfe von anderen Variablen dimensioniert werden. Wenn Sie also die Anzahl der Waren in unserer Preisliste nicht ein für alle mal festlegen wollen, sondern lieber vor dem Programmablauf eingeben wollen, können Sie das tun:

```
 5 : INPUT "WIE VIELE WAREN?";AN
10 : DIM PR(AN,4)
```

B	H	L
BA	CH	LE
BAS	UCH	-LE
BASI	BUCH	-LEH
BASIC	RBUCH	C-LEH
BASIC-	HRBUCH	C-LEHR
BASIC-L	EHRBUCH	IC-LEHR
BASIC-LE	LEHRBUCH	IC-LEHRB
BASIC-LEH	-LEHRBUCH	SIC-LEHRB
BASIC-LEHR	C-LEHRBUCH	SIC-LEHRBU
BASIC-LEHRB	IC-LEHRBUCH	ASIC-LEHRBU
BASIC-LEHRBU	SIC-LEHRBUCH	ASIC-LEHRBUC
BASIC-LEHRBUC	ASIC-LEHRBUCH	BASIC-LEHRBUC
BASIC-LEHRBUCH	BASIC-LEHRBUCH	BASIC-LEHRBUCH

L	H BASIC-LEHRBUCH	BB B B B B
LE	CH BASIC-LEHRBUC	AA A A A A
-LE	UCH BASIC-LEHRBU	SS S S S S
-LEH	BUCH BASIC-LEHRB	II I I I I
C-LEH	RBUCH BASIC-LEHR	CC C C C C
C-LEHR	HRBUCH BASIC-LEH	-- - - - -
IC-LEHR	EHRBUCH BASIC-LE	LL L L L L
IC-LEHRB	LEHRBUCH BASIC-L	EE E E E E
SIC-LEHRB	-LEHRBUCH BASIC-	HH H H H H
SIC-LEHRBU	C-LEHRBUCH BASIC	RR R R R R
ASIC-LEHRBU	IC-LEHRBUCH BASI	BB B B B B
ASIC-LEHRBUC	SIC-LEHRBUCH BAS	UU U U U U
BASIC-LEHRBUC	ASIC-LEHRBUCH BA	CC C C C C
BASIC-LEHRBUCH	BASIC-LEHRBUCH B'	HH H H H H

14 . K a p i t e l

DIE ERNTE WIRD EINGEFAHREN
- DATA-Listen und READ-Befehle -

Um einer Variablen in einem Programm einen Wert zuzuweisen, kennen wir bisher zwei Wege: Entweder können sie mit einer LET-Anweisung durch das Programm versorgt werden, oder über den INPUT-Befehl direkt von Ihnen. Einen dritten Weg möchte ich Ihnen nun vorstellen.

Sie können nämlich mit der Instruktion

DATA

in Ihrem Programm eine Liste von Daten fest installieren, aus der Sie bei Bedarf mit Hilfe der Anweisung

READ

die benötigten Werte herauslesen. Bevor Sie lange rätseln, wie das wohl in der Praxis aussehen mag, gleich ein Beispiel:

```
10 : CLEAR : CLS
20 : READ A
30 : PRINT "DAS QUADRAT VON ";A;" IST ";A^2
40 : GOTO 20
50 : DATA 1,2,3,4,5,6,7,8,9,10
```

In der Tat, Sie erhalten mit diesem Programm nacheinander die Anzeigen 1, 4, 9, 16, 25, 36, 49, 64, 81 und 100, also exakt die Quadratzahlen der Zahlen aus der Liste in Zeile 50. Als letzte Anzeige ist bei mir dann allerdings ein unfreundliches

ERROR 4 IN 20

erschienen. Dazu kommen wir später. Jetzt wollen Sie sicher erst mal wissen, wie das funktioniert hat. Was Sie vielleicht stutzig macht, ist, daß der Rechner, von dem Sie ja wissen, daß er brav Zeile für Zeile abarbeitet, Ihnen in Zeile 30 einen Wert ausgibt, den er aus Daten ermitteln mußte, die in Zeile 50 stehen, zumal er diese Zeile aufgrund des Rücksprungbefehls in Zeile 40 nie erreichen dürfte. Nun, damit sind Sie schon einem wesentlichen Merkmal der READ ... DATA- Anweisung auf der Spur.

Immer, wenn der Rechner in einem Programm auf eine READ-Instruktion trifft, springt er selbsttätig zur nächstbesten DATA-Liste, nimmt sich dort einen Wert, weist ihn der Variablen in der READ-Zeile zu und setzt das Programm an der Stelle fort, wo er es verlassen hatte. In unserem kleinen Beispielprogramm schicken wir ihn nach jedem PRINT wieder zu READ A und wie Sie sehen können, nimmt er dann jedesmal das nächste Datum aus der Liste. Der Rechner bedient sich also der

DATA-Liste wie eines Abreißkalenders: Was einmal gelesen wurde, ist weg! Ergänzen Sie unser Programm nun um die Zeile

```
15 : DATA 11,12,13,14,15,16,17,18,19,20
```

Starten Sie und staunen Sie. Aus dem Ergebnis läßt sich entnehmen, daß der Rechner erst zur Liste in Zeile 15 zurückgesprungen ist, und dann, nachdem diese geleert war, bei Zeile 50 mit dem Lesen der 1 begonnen hat. Wenn ich also oben noch behauptet habe, der Rechner springe zur nächstbesten DATA-Liste, war das unpräzise ausgedrückt. Richtig muß es heißen:

> Beim Antreffen einer READ-Instruktion im Programm springt der Rechner zuerst zu der DATA-Liste mit der niedrigsten Zeilennummer.

Kommt der Rechner im Programmablauf zu einer DATA-Liste, ohne vorher einen READ-Befehl gesehen zu haben, ignoriert es sie einfach und springt drüber weg. Ändern wir unser Programm nochmals um:

```
10 : CLEAR : CLS
20 : FOR I = 1 TO 10
30 : READ A,B
40 : PRINT A;A^2;B;B^2
50 : NEXT I
60 : DATA 1,2,3,4,5,6,7,8,9,10
70 : DATA 11,12,13,14,15,16,17,18,19,20
```

Es erscheinen nacheinander folgende Anzeigen:

```
1 1 2 4
3 9 4 16
5 25 6 36
7 49 8 64
9 81 10 100
11 121 12 144
13 169 14 196

    usw.
```

Schauen Sie sich dazu unsere READ- und PRINT-Anweisungen genauer an. Diesmal lesen wir gleich zwei Variablen mit READ A,B. Auch hier gilt das Prinzip des Abreißkalenders, so daß beim ersten Schleifen-Durchlauf A=1 und B=2 werden, beim zweiten dann A=3 und B=4.

Es ist dabei völlig gleichgültig, ob Sie Ihre Daten in einer DATA-Zeile oder in mehreren aufeinanderfolgenden unterbringen. Es dürfen zwischen den einzelnen DATA-Zeilen sogar andere Programmzeilen stehen, ohne daß dies einen Enfluß auf den Lesevorgang hätte. Worauf Sie allerdings achten müssen ist, daß Sie Ihre Daten immer durch ein Komma trennen und daß am Ende einer DATA-Liste keine Komma mehr stehen darf. Der Übersicht dient es natürlich, wenn Sie alle DATA-Zeilen am Ende des Programms zusammenfassen. Sie dürfen sie natürlich auch zu Beginn

aufstapeln, wie's Ihnen am besten gefällt.

Ein unverzeihlicher Fehler ist es, in einem Programm mehr READ-Befehle als Daten in den DATA-Listen zu haben. Das ist uns bei unserem ersten Beispiel passiert und daher rührte auch die Fehlermeldung ERROR 4. Sie können einen derart unrühmlichen Ausstieg aus dem Programm vermeiden, wenn Sie sich angewöhnen, das Ende der DATA-Listem immer mit einem bestimmten Wert, etwa Ø, zu markieren und nach jedem READ abfragen, ob etwa Ø gelesen wurde.

Wenn Ihre DATA-Liste von Anfang bis Ende gelesen worden ist, ist, bildlich gesprochen, der Abreißkalender leer. Damit Sie Ihre mühsam einprogrammierten Daten jedoch nicht nur einmal benutzen können, gehört zu READ .. DATA noch ein dritter Befehl, nämlich

RESTORE

Mit dieser Instruktion erreichen Sie, daß der Rechner bei der nächsten READ-Anweisung wieder von vorne, also bei der ersten DATA-Liste, anfängt, zu lesen. Sie haben also, um in unserem Bild zu bleiben, die Kalenderblätter nicht abgerissen, sondern nur umgeblättert. Ergänzen wir unser Programm um diesen Befehl:

```
20 : FOR I = 1 TO 20
45 : IF I=10 THEN RESTORE
```

So, jetzt können Sie sich zweimal Ihre Quadratzahlen ansehen.

Mit dem RESTORE-Befehl können Sie allerdings nicht nur alle DATA-Listen auf den Anfang zurücksetzten, sondern Sie können gezielt die READ-Anweisung am Beginn einer bestimmten DATA-Zeile weiterlesen lassen. Versuchen Sie gleich ob's funktioniert:

```
55 : RESTORE 70
57 : READ X : PRINT X
```

Alles klar? Es versteht sich fast von selbst, daß als Daten in einer DATA-Liste auch Strings stehen können. Diese müssen dann in Anführungszeichen gesetzt werden und selbstverständlich muß in der READ-Instruktion eine String-Variable stehen. Auch können Sie nach Herzenslust numerische und Textkonstanten in der DATA-Liste abwechseln, solange Sie den Überblick bei den READ-Befehlen nicht verlieren. Alles was zwischen zwei Kommas steht, wird als Datum gewertet und der Variable in READ zugewiesen. Das gilt auch für mathematische Ausdrücke.

Die Beispiele, die Sie bisher kennengelernt haben, mögen nützlich gewesen sein, Ihnen zu verdeutlichen, wie die READ ... DATA-Instruktion zu handhaben ist, dürften Sie jedoch kaum davon überzeugt haben, daß sie besonders sinnvoll ist. DATA-Listen finden dort ihren berechtigten Einsatz, wo Sie mit der selben Liste von Daten immer wieder etwas anstellen wollen. Dann wäre es nämlich zu umständlich, diese Daten jedesmal neu einzulesen und übersichtlicher als 100 LET-Anweisungen ist eine DATA-Liste allemal. Doch am besten sehen Sie das an einem Beispiel:

14.1 Anwendungsbeispiel PREISLISTE

Ein Großhändler möchte für eine Warengruppe Preislisten erstellen, die er seinen Kunden mitgeben kann. Diese Listen sollen die Artikelbezeichnung, die Artikelnummer, die Stückzahl pro Verpackungseinheit sowie den VK mit und ohne USt. enthalten. Hinzu kommt noch, daß der Händler mit jedem seiner Kunden einen individuellen Rabatt aushandelt. Auf den Preislisten soll jetzt aber dieser Rabatt vom VK schon abgezogen sein, so daß der Kunde seinen EK gleich vor Augen hat.

Das Programm soll diese Aufgabe mit Hilfe der READ ... DATA-Anweisung meistern. Als erstes erstellen wir die DATA-Liste, mit der wir arbeiten wollen. Im Beispiel beschränken wir uns dabei auf wenige Posten.

```
200 : DATA "HOLZSCHRAUBEN","0040","25",5.30
210 : DATA "GEWINDESCHRAUBEN","0041","50",11.20
220 : DATA "SCHRAUBENMUTTERN","0042","50",6.80
230 : DATA "STAHLSTIFTE","0050","100",7.90
```

Wir haben also eine Datenliste, in der sowohl Strings als auch numerische Daten vorhanden sind. Um die Liste übersichtlicher zu halten, wurde für jeden Artikel eine eigene Zeilennummer gewählt. Damit daraus das gewünschte Programm wird, benötigen wir im wesentlichen noch drei Teile:

A) Die zugehörigen READ-Instruktionen
B) Den Kundenrabatt
C) Die erforderlichen LPRINT-Befehle

A) Die READ-Instruktionen:

Da wir die Länge unserer DATA-Liste genau kennen und sie sehr übersichtlich aufgebaut haben, können wir auf die Kennzeichnung des Listenendes verzichten, weshalb sich auch beim READ-Prozeß eine Abfrage nach einem entsprechenden Stichwort (etwa "ENDE") erübrigt. Wenn wir eine Schleife zum Einlesen der Daten benutzen, können wir am einfachsten kontrollieren, daß nur die erforderliche Anzahl von READ-Befehlen gegeben wird. Programmieren Sie:

```
30 : For I = 1 TO 4
40 : READ A$,NR$,ST$,VK
50 : NEXT I
```

Warum ich übrigens die Artikelnummer und die Stückzahl pro Packung als Stringvariablen verwende, werde ich Ihnen bei der Besprechung des LPRINT-Teiles verdeutlichen.

Ist Ihnen aufgefallen, daß die Schleife, so wie sie oben steht, einen gravierenden Fehler aufweist? Überlegen Sie einmal, was geschieht. Bei jedem Durchlauf der Schleife liest der Rechner die Werte der DATA-Liste in die entsprechenden Variablen ein, um sie beim nächsten Durchlauf dann wieder rauszuwerfen und durch die neuen zu ersetzen. Am Schluß haben wir dann nur die Werte der letzten Zeile zur Verfügung, etwas

mager für eine komplette Preisliste. Das Problem läßt sich auf zwei Weisen lösen. Haben Sie eine Idee, welche?

Erstens können wir unsere Variablen als eindimensionale Felder deklarieren, um dann in der Schleife die Variablen A$(I), NR$(I), ST$(I) und VK(I) einzulesen. Damit könnten wir dann, etwa in einer LPRINT-Schleife, weiterarbeiten.

Zweitens können wir (fast) alles so lassen wie es ist. Dann müssen wir allerdings vor jedem Schleifenende schon alles erledigt haben, was wir mit den Variablenwerten anfangen wollten.

In unserem Beispiel würde ich zur zweiten Lösung raten, denn wir möchten die Werte ja im wesentlichen nur ausdrucken und das können wir ja auch innerhalb der READ-Schleife erledigen. Wir ersparen uns damit, eine zweite Schleife zu programmieren und, was wichtiger ist, eine ganze Menge Speicherplatz. Denn wenn Ihre Warenliste länger als nur vier Posten ist, wirds oft nicht mehr reichen, zusätzlich zur langen DATA-Liste auch noch umfängliche Variablenfelder zu deklarieren. Damit wir in der Schleife auch genügend Platz haben, ändern Sie bitte

```
50 : NEXT I
```

in

```
100 : NEXT I
```

um.

B) Eingabeteil Kundenrabatt:

Den Rabatt, den der einzelne Kunde erhalten soll, müssen wir wissen, bevor wir mit dem Programmieren des LPRINT-Teiles beginnen können. Geben Sie deshalb ein:

```
20 : INPUT "KUNDENRABATT IN PROZENT?";RB
```

C) LPRINT-Teil:

Erinnern wir uns nochmal, was wir alles ausdrucken wollten:

> Artikelbezeichnung, Art.Nr., Stückzahl pro Packung, Nettopreis abzüglich Kundenrabatt und den daraus resultierenden VK incl. Umsatzsteuer

Die ersten drei Positionen können wir so übernehmen, wie wir sie aus der DATA-Liste herauslesen. Den VK und den VK incl. USt. müssen wir vor dem Ausdrucken allerdings individuell errechnen:

```
50 : LET VK = VK*RB/100
```

Jetzt kann's ans Ausdrucken gehen. Wir beginnen mit der Artikelbezeichnung, der Art.Nr. und der Verpackungseinheit:

```
60 : LPRINT A$;NR$;ST$
```

Und schließen gleich den VK mit und ohne USt. an:

```
70 : LPRINT VK;VK*1.14
```

So, fertig! Jetzt können die Kunden kommen. Machen Sie mal einen Probelauf. Sind Sie mit dem Ergebnis zufrieden? Also, ich find's furchtbar. Auf meinem CE-150-Drucker sieht das (bei 7% Rabatt) so aus:

```
HOLZSCHRAUBEN  00
4025
 4.929 5.61906
GEWINDESCHRAUBEN00
4150
 10.416 11.87424

      usw.
```

Diese Preisliste könnte von einem Geheimdienst stammen, ist aber wohl keinem Kunden zuzumuten. Ich möchte es Ihnen überlassen, mit den uns bisher bekannten Mitteln die LPRINTs so übersichtlich zu gestalten, daß auch Leute die Liste verstehen, die nicht bei der Entstehung dieses Programms dabei waren. Einen Hinweis will ich Ihnen vorher noch geben. Gestalten Sie den Preisausdruck so, daß höchstens zwei Stellen hinter dem Komma auftauchen. Nehmen Sie dazu vorerst den Befehl

```
65 USING  "###.##"
```

Aufgabe 14.1:

Bearbeiten Sie die LPRINT-Anweisungen so, daß alle Daten übersichtlich und kommentiert ausgedruckt werden. Setzten Sie auch eine Überschrift an den Kopf der Preisliste. Lassen Sie zudem den berücksichtigten Rabatt im Kopf der Liste angeben. Geben Sie dem Kind (Programm) einen Namen.

Aufgabe 14.2:

Durch die Ausgabeanweisung USUNG "###.##" erreichen wir, daß der Preis formal richtig in DM und Pf. ausgedruckt wird. Gleichzeitig bauen wir aber einen Rundungsfehler mit ins Programm ein, da die USING-Anweisung einfach den "Schwanz abschneidet", ohne auf Rundungsregeln zu achten. Da Sie als Kaufmann jedoch einen Betrag von 5.419 DM gerne als 5.42 DM auf der Liste hätten, sollten Sie die USING-Instruktion durch einen Rundungsalgorithmus ersetzen.

Aufgabe 14.3:

Erstellen Sie mit Hilfe von DATA-Listen ein Programm, das Ihnen als Telefon und Adressverzeichnis dienen kann. Auf die Eingabe des Nach- und Vornamen Ihres Bekannten soll dessen Adresse und Telefonnummer ausgegeben werden. Machen Sie sich bitte die Mühe, selbst ein Programm von Grund auf zu entwerfen und bis zum letzten Befehl zu programmieren, bevor Sie im Lösungsteil meinen Vorschlag anschauen.

14.2 Gestaltung der Ausgabe mit dem TAB-Befehl

Zugegeben, wir haben es bisher immer noch geschafft, daß ein auszudruckendes Ergebnis so auf dem Papier erschienen ist, wie wir es haben wollten, ohne daß wir den Befehl

TAB

gekannt hätten. Wir können uns die Arbeit jedoch erleichtern und unsere gestalterischen Möglichkeiten erweitern, wenn uns diese Anweisung zusätzlich zur Verfügung seht. Wenn! Denn die TAB-Anweisung ist meines Wissens bei den SHARP PCs nur auf dem PC-1500 A vorhanden, so daß die Besitzer anderer Pockets gleich weiterblättern sollten, um nicht neidisch zu werden. Größere Bildschirmgeräte hingegen verfügen in der Regel über den TAB-Befehl, der dort auch die Gestaltung der Bildschirmausgabe mit übernimmt.

"Die TAB-Instruktion definiert die Schreibposition". So steht es schlicht und ergreifend in der Bedienungsanleitung des PC-1500 A. Gemeint ist die Schreibposition des Druckerschreibwerkes. Also: Auch alle PC-1500 A Besitzer können umblättern, wenn sie keinen Drucker besitzen. Ist jetzt überhaupt noch jemand dabei? Gut.

Da TAB unschwer als Abkürzung von **TAB**elle zu erkennen ist, leuchtet ein, daß die Hauptaufgabe dieses Befehls die Darstellung von Tabellen sein dürfte. Bisher konnten Sie gestalterisch wirken, indem Sie z.B. zwischen zwei auszudruckende Variablen ein LPRINT " " programmierten. Damit kann man, wenn auch etwas umständlich, nach Herzenslust Tabellen gestalten. Schwierig wirds nur, wenn die Länge der zuerst ausgedruckten Variable auch variabel ist und die zweite trotzdem akkurat unter den anderen zweiten stehen soll.

Daß auch das jetzt kein Problem mehr darstellt, will ich Ihnen gleich demonstrieren. Vorweg jedoch noch ein kurzes Wort zum TAB-Befehl selbst: Sie können ihn als einzelne Befehlszeile ins Programm setzen und er wirkt sich auf den als nächsten folgenden LPRINT-Befehl aus, etwa so:

```
120 : TAB(10)
130 : LPRINT A
```

Oder Sie können TAB mit in den LPRINT-Befehl aufnehmen, für den er gelten soll:

```
130 : LPRINT TAB(10);A
```

Das ist meiner Meinung nach wesentlich sinnvoller und es kann nicht passieren, daß Sie versehentlich einen anderen LPRINT-Befehl dazwischenprogrammieren. Die Zahl in der Klammer gibt eben die gewünschte Schreibwerkposition an. Sie müssen also wissen, wieviele Positionen Sie zur Verfügung haben und wo Sie mit dem Ausdruck beginnen wollen. Die Anzahl der verfügbaren Positionen hängt natürlich vom Drucker selbst und von der (mit CSIZE) vorprogrammierten Schriftgröße ab. Auch bei Bildschirmwiedergabe können Sei in der Regel zwischen 20, 40 und

80 Zeichen wählen. Dementsprechend viele Positionen haben Sie zur Auswahl.

Doch bleiben wir für die folgenden Beispiele wieder beim PC-1500 A mit Drucker CE-150 und der normalen Schriftgröße CSIZE 2. Auf Ihren Papierstreifen passen dann 18 Zeichen. Lassen Sie mal folgendes kleines Programm laufen:

```
10 : FOR I = 1 TO 18
20 : LPRINT TAB(I);"A"
30 : NEXT I
```

Schön, wie sich die Druckposition diagonal übers Blatt verteilt. Weniger schön, daß auf der Anzeige ERROR 19 IN 20 erscheint! Warum? Nun, unser Rechner zählt mal wieder von 0-17 und nicht von 1-18, das ist das ganze Geheimnis. Also jetzt richtig:

```
10 : FOR I = 0 TO 17
```

Aber wenden wir uns der eigentlichen Aufgabe des TABs zu, dem Ausdruck von Tabellen. Nehmen wir an, Sie möchten eine Tabelle der Zahlen 1-20 sowie ihrer zweiten und dritten Potenz. Schnell ein NEW getippt und munter losprogrammiert:

```
 5 : CLEAR : CLS
10 : LPRINT "POTENZTABELLE"
15 : LF 2
20 : LPRINT" X";"  X↑2";"  X↑3"
30 : FOR I = 1 TO 20
40 : LPRINT I;I↑2;I↑3
50 : NEXT I
60 : END
```

Nun gut, man kann mit dem Ergebnis schon was anfangen, aber besonders tabellarisch sieht es nicht aus. Bedienen wir uns jetzt mal unseres neuen Instruments, der TAB-Anweisung:

```
40 : LPRINT I;TAB(5);I↑2;TAB(12);I↑3
```

Damit auch die Überschriften der Spalten am rechten Platz sind, ändern wir Zeile 20 entsprechend:

```
20 : LPRINT "X";TAB(5);"X↑2";TAB(12);"X↑3"
```

In Tab. 7a können Sie das Ergebnis bewundern. Wenn Sie allerdings genau abzählen, merken Sie, daß die Zahlen gar nicht an der Position stehen, die in den TABs angegeben sind, sondern immer um eine Position nach rechts verschoben sind. Das liegt daran, daß der Rechner jede Zahl mit Vorzeichen erfaßt und auch so die benötigten Stellen für den Ausdruck ermittelt. Nur wird ein positiven Vorzeichen dann unterdrückt und nur eine Leerstelle "ausgedruckt". Wir müssen in unserem Fall nun eben die Überschriften auch noch um eine Stelle weiter nach rechts verschieben, damit sie exakt über den Zahlenkolonnen beginnen.

POTENZTABELLE

X	X^2	X^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000
11	121	1331
12	144	1728
13	169	2197
14	196	2744
15	225	3375
16	256	4096
17	289	4913
18	324	5832
19	361	6859
20	400	8000

Tab. 7a

POTENZTABELLE

X	X^2	X^3
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000
11	121	1331
12	144	1728
13	169	2197
14	196	2744
15	225	3375
16	256	4096
17	289	4913
18	324	5832
19	361	6859
20	400	8000

Tab.7b

Tab.7: Ausdruck einer Potenztabelle auf dem CE-150. Fassung 1 (Tab.7a) und Fassung 2 (Tab.7b)

Wenn Sie ein besonders ordentlicher Mensch sind und meinen, Zahlen müssen rechtsbündig untereinander stehen, bitte schön:

```
35 : USING "#####"
```

Die Überschriften sollten dann allerdings etwas nach rechts gerückt werden:

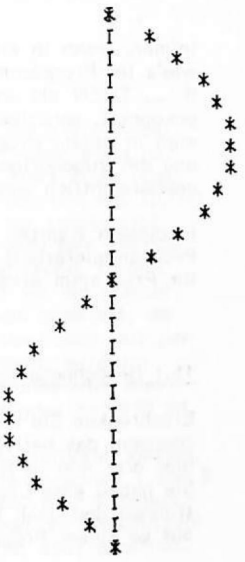
```
10 : LPRINT TAB(3);"POTENZTABELLE"
20 : LPRINT TAB(4);"X";TAB(7);"X ^2";TAB(14);"X ^3"
```

Ganz passabel der endgültige Ausdruck, oder? In Tab.7b strahlt er Ihnen entgegen. Nebenbei bemerkt: Bei Verwendung von USING-Formatierungen in Verbindung mit TAB-Anweisungen kann man leicht durcheinanderkommen und sich verzählen. Hier gilt oft: Probieren geht über Kapitulieren!

Ich möchte Sie aus dem TAB-Kapitel jedoch nicht entlassen, ohne wenigstens kurz auf die (etwas bescheidenen) graphischen Möglichkeiten dieses Befehls hinzuweisen. Wer seinen Rechner so richtig zum Malen bringen will, sei auf das **Graphikhandbuch** von **Klaus Schreiner** verwiesen, das auch bei der FISCHEL GmbH erschienen ist.

Erinnern Sie sich noch an den diagonalen Ausdruck der "A" am Anfang des Abschnitts? Dann können Sie jetzt sicher Ihre Kreativität kaum noch bremsen. Tun Sie's trotzdem und schauen Sie sich zu guter Letzt noch folgendes Programm und den damit erzeugten Ausdruck an:

```
10 : FOR I = 0 TO 360 STEP 15
20 : LET X = SIN(I)*8.9+9
30 : LPRINT TAB (X);"*";TAB(9);"I"
40 : NEXT I
```



Gar nicht so übel als illustrative Graphik. Finden Sie nicht auch? Wie ich auf die Werte in Zeile 20 gekommen bin? Nun, ich geb's zu, etwas herumprobieren war schon auch dabei. Daß ich den Sinus ausgerechnet mit 8.9 multipliziere, liegt daran, daß so einerseits verhindert wird, daß X den Wert 18 annimmt (TAB(18) gibt es ja nicht), und ich zum andern erreiche, daß die Werte für SIN(I) <1 doch so weit wie möglich ausgebaucht werden. Wenn Sie graphische Funktionsdarstellungen machen wollen, müssen Sie wissen, daß TAB(12) dasselbe ergibt wie TAB(12.999...). Es werden also nur ganzzahlige Druckpositionen angesteuert. Das was Sie jetzt gelernt haben, können Sie auch gleich in einer Aufgabe anwenden.

Aufgabe 14.4:

Erweitern Sie das Programm für die Sinuskurve so, daß gleichzeitig (evtl. in einer anderen Farbe) auch eine Cosinusfunktion in korrekter Phasenlage mitgedruckt wird.

15 . K a p i t e l

DER KALKULIERTE SEITENSPRUNG
- Berechnete Sprünge und Menütechnik -

Immer, wenn in einem Programm eine Entscheidung darüber anstand, wie's im Programm weitergehen soll, haben wir bisher die Anweisung IF ... THEN als unverzichtbar kennengelernt. War sie mit einem GOTO gekoppelt, sprachen wir von einem "bedingten Sprung". Mitunter muß man in einem Programm eine ganze Reihe solcher IF ... THEN-Befehle und die zugehörigen Sprungadressen untereinanderschreiben, was leicht unübersichtlich werden und zu Programmierfehlern führen kann.

In diesem Kapitel will ich Ihnen zeigen, wie Sie in einem solchen Fall Programmierarbeit sparen und gleichzeitig eine bessere Übersicht über Ihr Programm erhalten können.

15.1 Berechnete Sprünge

Erschrecken Sie nicht bei dieser Überschrift, Sie müssen gar nichts rechnen, das betrifft nur Ihren Computer. Aber lassen Sie uns auch hier erst mal in Ruhe ein Beispiel betrachten. Nehmen Sie mal an, Sie haben eine Liste von Ortschaften, die Sie nach Postleitbezirken trennen und dann in einem Programm weiterverarbeiten. Ein Ausschnitt aus so einem Programm könnte etwa so aussehen:

```
100 : READ A$
105 : IF A$ = "ENDE" THEN GOTO 1100
110 : LET PZ = VAL A$
120 : IF PZ< 2000 THEN GOTO 300
130 : IF PZ< 3000 THEN GOTO 400
140 : IF PZ< 4000 THEN GOTO 500
150 : IF PZ< 5000 THEN GOTO 600
160 : IF PZ< 6000 THEN GOTO 700
170 : IF PZ< 7000 THEN GOTO 800
180 : IF PZ< 8000 THEN GOTO 900
190 : GOTO 1000
200 : DATA "7325 BAD BOLL","5427 BAD EMS"
210 : DATA "2357 BAD BRAMSTEDT","3490 BAD DRIBURG"

                    ...

290 : DATA "6702 BAD DÜRKHEIM","ENDE"

                    ...

1100 : END
```

In den Zeilen 300, 400, ..., 1000 werden die Adressen der einzelnen

Postleitbezirke dann nach Wunsch weiterverarbeitet. Uns interessieren jedoch nur die Zeilen 120-190 und ich möchte Ihnen zeigen, wie Sie diese 7 Befehle auf eine Zeile reduzieren können. Diese lautet:

```
120 ON X GOTO 300, 400, 500, 600, 700, 800, 900, 1000
```

Das muß ich Ihnen erklären: Der Befehl

```
ON ... GOTO
```

bewirkt, daß sich der Rechner die Sprungadresse aus der Liste hinter GOTO selbst aussucht. Dies geschieht nicht willkürlich oder etwa der Reihe nach wie bei READ, sondern ist abhängig vom Wert einer numerischen Variablen (in unserem Beispiel: X), die zwischen ON und GOTO steht. Ist X=1, nimmt er die erste Adresse, für X=2 die zweite und wenn X=8 ist, eben die achte. Der Computer springt also sehr berechnend, weshalb wir auch von einem **berechneten Sprung** reden. Nimmt unser X nun einen Wert an, der kleiner 1 oder größer als die Anzahl der vorhandenen Sprungadressen ist, wird GOTO einfach ignoriert, der Rechner überspringt also den Sprungbefehl. Was geschieht aber mit den "krummen" X-Werten? Hat X eine Dezimalzahl zum Wert, wird zur Berechnung der Sprungadresse der INT(X) herangezogen. Es wird also die dritte Adresse hinter GOTO gewählt, egal ob X=3 oder X=3.979 ist.

Damit wir diesen Befehl in unserem Beispielprogramm einführen können, brauchen wir noch ein solches X, das immer die Werte zwischen 1 und 8 annimmt, die die gewünschte Adresse anspringen lassen. Kein Problem, denn wir haben ja unsere Variable PZ, die abhängig von der Postleitzahl des gelesenen Ortes, Werte zwischen 1000 und vielleicht 8941 annimmt. Ergänzen wir also unser Programm:

```
115 LET X = PZ/1000
```

Jetzt bekommen alle Orte zwischen 7000 Stuttgart und 7990 Friedrichshafen einen X-Wert zwischen 7 und 7.99, der so immer die 7-te Sprungadresse anvisiert. Das ist Zeile 900. Hier könnte etwa stehen:

```
900 : PRINT A$;" LIEGT IN BADEN-WÜRTTEMBERG"
910 : GOTO 100
```

Daß Sie jetzt die Zeilen 130-190 ersatzlos streichen können, dürfte klar sein.

Sie können die ON ... GOTO-Anweisung auf Ihrerm Rechner mal durchchecken, wenn Sie dieses einfache Programm eintippen, in das Sie dann unterschiedliche Werte für unser X (das natürlich auch A oder Z9 heißen darf) eingeben können, und den erzielten Effekt mitgeteilt bekommen:

```
10 : CLEAR : CLS
20 : INPUT "SPRUNGINDEX";X
30 : ON X GOTO 50, 60, 70, 80, 90, 100
40 : PRINT "KEINEN SPRUNG AUSGEFÜHRT!"
45 : GOTO 20
```

```
 50 : PRINT "SPRUNG IN ZEILE 50"
 55 : GOTO 20
 60 : PRINT "SPRUNG IN ZEILE 60"
 65 : GOTO 20

                ...

100 : PRINT "SPRUNG IN ZEILE 100"
110 : GOTO 20
```

Da es sich immer empfiehlt, frisch Gelerntes möglichst sofort anzuwenden, habe ich mir auch gleich noch eine Übungsaufgabe dazu ausgedacht:

Aufgabe 15.1:

Sie haben das folgende kleine Programm zur Unterstützung Ihrer Haushaltsführung in den Computer eingegeben:

```
 10 : INPUT "KONTOSTAND? (IN DM);KT
 20 : INPUT "MIETE? (IN DM);MT
 30 : INPUT "NEBENKOSTEN ?";NK
 40 : INPUT "HAUSHALTSGELD?";HA
 50 : INPUT "AUTOKOSTEN?";AU
 60 : LET KT = KT-(MT+NK+HA+AU)
 70 : IF KT >0 THEN GOTO 90
 80 : PRINT "KEINE WEITEREN AUSGABEN!" : GOTO 200
 90 : INPUT "ANSCHAFFUNGEN? (DM)";AN
200 : END
```

Programmieren Sie zwischen Zeile 90 und 200 mit Hilfe von ON ... GOTO so weiter, daß der Rechner Ihnen je nach Höhe der beabsichtigten Anschaffung mitteilt, ob Sie Ihr Konto überziehen, ob es auf Null aufgeht oder ob noch ein Restbetrag übrig geblieben ist. (Hinweis: Schauen Sie sich davor noch einmal die Funktion SGN auf Seite 46 an.)

15.2 Die Menütechnik

Die Menütechnik ist eine Methode der Programmgestaltung, die im wesentlichen dazu dienen soll, das fertige Programm benutzerfreundlich zu machen. Es handelt sich dabei um eine Kombination einer INPUT-Abfrage mit einem berechneten Sprung, die den weiteren Programmlauf von einer Entscheidung des Benutzers abhängig macht.

Vielleicht erinnern Sie sich noch an das kleine Programm zu Beginn des 9. Kapitels. Da sollten Sie entscheiden, ob der Rechner Additionen, Subtraktionen, Multiplikationen oder Divisionen durchführen sollte. Je nach Ihrer Entscheidung ist er dann zu einer bestimmten Subroutine gesprungen. Sehen Sie die Ähnlichkeit zu unserem ON ... GOTO-Befehl? Wenn wir den dazu benutzen könnten, nicht nur bestimmte GOTOs auszuführen, sondern auch unter bestimmten Bedingungen in Subroutinen zu springen,

wäre das eine interessante Möglichkeit. Nun, Sie haben es sich sicher schon gedacht, diese Möglichkeit existiert. Wir benötigen dafür allerdings einen etwas anderen Befehl:

```
ON ... GOSUB
```

heißt das dann. Lassen Sie uns doch diesen Befehl in das Programm GRUNDRECHENARTEN aus dem 9. Kapitel einbauen. Da heißt es bisher in den Zeilen 250-340:

```
250 : PRINT "BITTE WAEHLEN SIE"
260 : PRINT "1 FUER ADDITION"
270 : PRINT "2 FUER SUBTRAKTION"
280 : PRINT "3 FUER MULTIPLIKATION"
290 : PRINT "4 FUER DIVISION"
300 : INPUT X
310 : IF X = 1 THEN GOSUB 50
320 : IF X = 2 THEN GOSUB 90
330 : IF X = 3 THEN GOSUB 130
340 : IF X = 4 THEN GOSUB 170
```

Jetzt kann ich Sie sicher nicht mehr länger daran hindern, sofort den neu gelernten Befehl hier einzusetzen. Also:

```
310 : ON X GOSUB 50, 90, 130, 170
```

Die Zeilen 320 bis 340 können Sie jetzt getrost aus dem Programm werfen.

Da Ihnen der Rechner in den Zeilen 250-290 wie auf einer Speisekarte seine kalkulatorischen Spezialitäten angeboten hat, wird dieses Verfahren der Benutzerentscheidung auch **Menütechnik** genannt. Daß dieses Wählen à la carte auf einem Bildschirmgerät natürlich sehr viel schöner anzuwenden ist, wenn man das gesamte Menü auf einen Blick vor sich hat, darf die Besitzer eines größeren Computers jetzt ruhig die Brust vor Stolz schwellen lassen. Die große Zeit des PC-Besitzers kommt dann wieder, wenn er seinen Computer im Büro geschwind aus der Tasche ziehen kann, während so mancher Große "at home" bleiben mußte.

16 . K a p i t e l

SO RICHTIG NETT ISTS BEIM ROULETTE
- Der Computer will spielen -

Wenn der Computer zum Spielzeug wird, steckt in der Regel eine ganze Menge Arbeit dahinter. Diese Arbeit wird den Anwendern in der überwiegenden Mehrheit der Fälle jedoch von findigen Programmierern abgenommen und dann auch meist als "festverdrahtetes Programm" in Spielcassetten für spezielle Computerspielgeräte angeboten. Von Schlümpfen, die über den Bildschim hopsen, bis zur fast perfekten Simulation eines Space-Shuttle-Flugs scheint die Fertigkeit der Computer, richtiger ihrer Programmentwickler, keine Grenzen zu kennen.

Obwohl unsere Programmiererfahrung am Ende dieses Buches noch in den Kinderschuhen steckt, und uns die PCs, auf die wir den Schwerpunkt gelegt haben, keinen attraktiven Mehrfarbbildschirm bieten, möchte ich Ihnen in diesem Kapitel zeigen, daß Ihre Fähigkeiten durchaus ausreichen, um ein nettes Spielchen mit Ihrem Computer zu wagen.

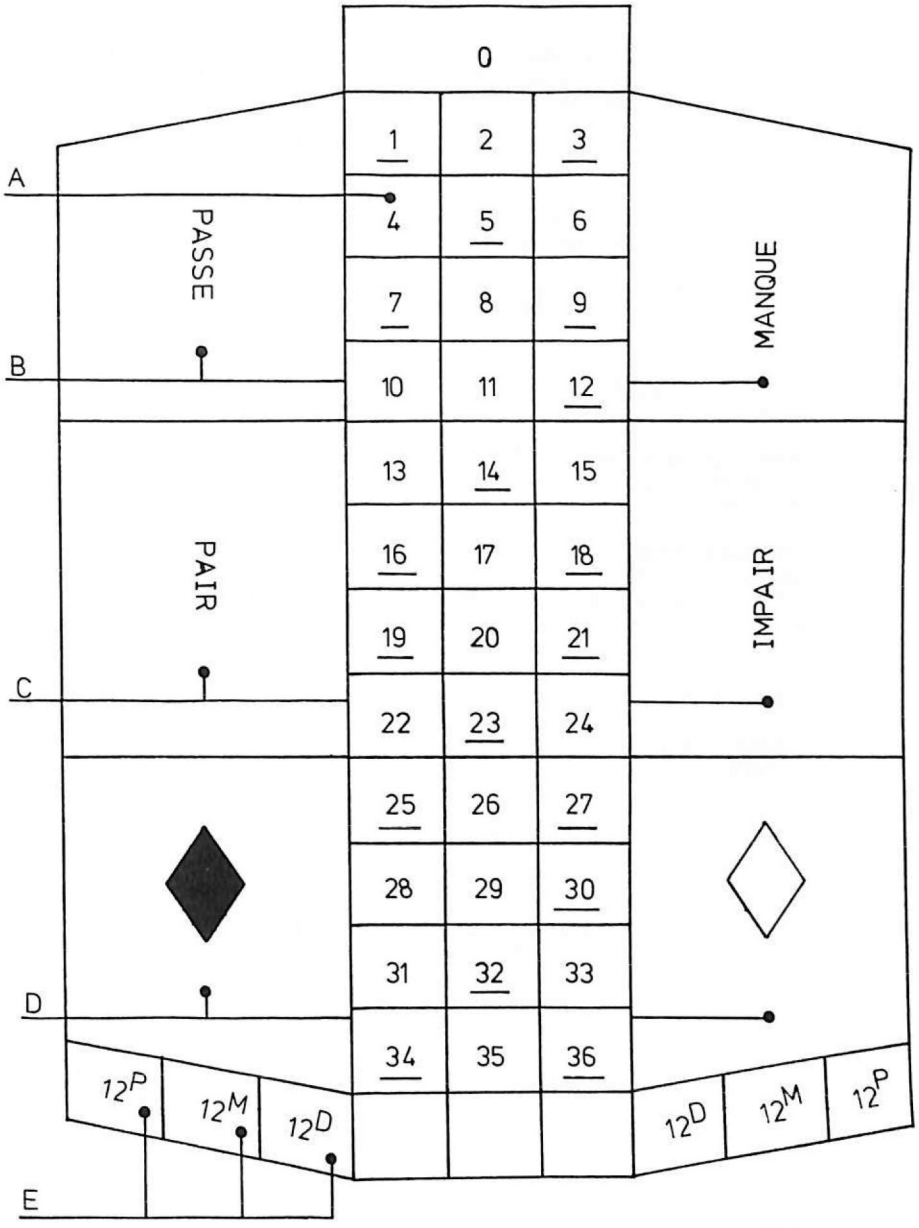
Ich habe für den PC-1500 A ein **Roulette-Programm** geschrieben, das, so hoffe ich, leicht verständlich ist und ohne Insiderkniffe oder gar Maschinensprache ein ganz brauchbares Ergebnis liefert. Ich möchte Sie dabei ein wenig an meiner Programmplanung teilnehmen lassen und hoffe, Sie können das bisher Gelernte so nochmals vertiefen. Sie brauchen also diesmal, bis auf eine unbedeutende Ausnahme, nichts Neues hinzuzulernen.

16.1 Die Regeln des Roulette-Spiels

Das Roulettespiel ist eine Art Wettspiel, wo Sie darauf wetten, daß eine Kugel, die in das "Roulette" geworfen wird, auf einer bestimmten Zahl liegen bleibt. Es gibt die Möglichkeiten 1 bis 36 sowie die "Zero", also die Null. Sie spielen gegen die Bank, die, wenigstens in unserem Programm, über unbegrenzte Geldmittel verfügt. Haben Sie auf die richtige Zahl gesetzt, bekommen Sie den 36-fachen Einsatz zurück, wenn nicht, haben Sie Ihren Einsatz verloren.

Da es auf Dauer unbefriedigend ist, in 35 von 36 Spielen zu verlieren, gibt es noch andere Wettmöglichkeiten mit höheren Gewinnchancen. In Abb. 11 finden Sie ein Roulette-Spielfeld, anhand dessen ich Ihnen die Wettmöglichkeiten erklären möchte. Auf ein derartiges Spielfeld setzen die Spieler im Casino ihre Chips, die sie vorher an der Kasse für bares Geld erworben haben. Mit dem Setzen der Chips auf bestimmte Stellen wird dem Bankhalter (Croupier) angezeigt, worauf man wetten möchte. Da ich mich im Programm der Übersichtlichkeit halber auf nur einen Teil der insgesamt 18 verschiedenen Wettmöglichkeiten beschränkt habe, möchte ich auch nur diese hier erklären. Eine Übersicht finden Sie in Tab. 8. Aus ihr geht hervor, daß die Gewinnchancen im statistischen Mittel immer bei der Bank liegen, die damit ihr Casino betreiben kann.

Abb.11: Spielfeld beim Roulette



	WETTE	CHANCE	AUSZAHLUNG
A)	Volle Zahl	1 : 36	36-facher Einsatz
B)	Erste oder zweite Hälfte der Zahlen (Passe oder Manque)	18 : 19	2-facher Einsatz
C)	Gerade oder ungerade (Pair oder Impair)	18 : 19	2-facher Einsatz
D)	Rot oder Schwarz (Rouge oder Noir)	18 : 19	2-facher Einsatz
E)	1., 2. oder 3. Dutzend (12^P, 12^M oder 12^D)	12 : 25	3-facher Einsatz

Tab. 8: Wettmöglichkeiten beim Roulette

Das liegt daran, daß die "Zero" eine Sonderrolle spielt. Sie gehört in keine der Gruppen B)-E) und auch in Gruppe A) gibt es mit ihr 37 Zahlen und nur den 36-fachen Einsatz zu gewinnen.

Gruppe D) entsteht dadurch, daß die Zahlen 1-36 in unregelmäßigem Wechsel zur Hälfte aus roten und aus schwarzen Zahlen bestehen. In Abb. 11 habe ich die roten Zahlen unterstrichen.

16.2 Das ROULETTE-Programm

Stellen wir zuallererst die Aufgaben zusammen, die unser Programm erfüllen muß, damit der Computer an die Stelle des Croupiers treten kann:

1. Die wichtigste Aufgabe besteht darin, das "Roulette" zu ersetzen und die Gewinnzahl zu ermitteln.

2. Es soll uns zudem mitteilen, ob diese Zahl rot oder schwarz, gerade oder ungerade ist, in der ersten (manque) oder zweiten (passe) Hälfte des Zahlenfeldes liegt und, ob sie zum 1., 2. oder 3. Dutzend der Zahlen gehört.

3. Es soll abfragen, worauf der Spieler wieviele Chips wetten möchte. Maximal fünf Spieler sollen gleichzeitig setzen können.

4. Der Einsatz der Spieler soll gespeichert und der Gewinn

eines jeden soll ermittelt werden.

5. Den Spielern soll am Anfang eine bestimmt Menge an Chips "verkauft" werden und der Rechner soll die Chipkonten eines jeden Spielers verwalten und den Kontostand nach dem Setzen sowie nach der Gewinnauschüttung mitteilen.

6. Es sollen demnach die Gewinne den Konten gutgeschrieben und Verluste abgezogen werden.

Schauen Sie sich einfach mal das fertige Programm in dem Listing auf den nächsten Seiten an. Erschrecken Sie nicht über den Umfang, sondern versuchen Sie einmal herauszufinden, was Sie davon jetzt schon verstehen. Es ist nur ein Befehl dabei, den Sie nicht kennen:

```
15 : GCURSOR I
```

Wie er sich auswirkt sehen Sie am besten, wenn Sie Zeile 10-25 schon einmal eingeben und mit RUN starten. GCURSOR stellt die Entsprechung des TAB-Befehls für das Display dar. Der Unterschied liegt nur darin, daß das Display nicht nur 18 Positionen hat, sondern 156 (0-155). Wie damit eine Laufschrift erzeugt wird, sehen Sie im Roulette-Listing. Lassen Sie den WAIT-Befehl in der Schleife weg, müssen Sie "von Hand kurbeln", indem Sie dauernd die ENTER-Taste drücken.

Bevor wir die einzelnen Programmteile von ROULETTE besprechen, soll Ihnen eine Liste der verwendeten Variablenfelder die Orientierung im Programm erleichtern. Einfache Variablen sind nicht aufgeführt. Die Variablennamen sind so gewählt, daß die Bedeutung ihres Inhalts annähernd zu erkennen ist. Die Indizes entsprechen der Dimensionierung in Zeile 60.

	FELD	BEDEUTUNG	INHALT
1.	Z(N)	Z(0): gezogene Zahl Z(1-5): gesetzt Zahl der Spieler 1-5	0-36
2.	RN$(N)	**R**ouge/**N**oir RN$(0): ermitteltes Ergebnis RN$(1-5): Wetten der Spieler	R,S,Ø
3.	GU$(N)	**G**erade/**U**ngerade Index analog zu RN$(N)	G,U,Ø
4.	PM$(N)	**P**asse/**M**anque	P,M,Ø
5.	DZ(N)	**D**ut**Z**end	1,2,3,Ø
6.	E(N,5)	**E**insatz der Spieler (1-N) Indizes: E(N,1): Einsatz des N-ten	beliebige numerische Konstante

```
 1:REM **********
 2:REM
 3:REM  ROULETTE
 4:REM
 5:REM **********
 6:REM
 9:CLS :CLEAR
10:FOR I=155TO 1
   STEP -1
12:WAIT 1
15:GCURSOR I
20:PRINT "******R
   OULETTE-SPIEL*
   *****"
25:NEXT I
28:WAIT 150
30:PRINT "******R
   OULETTE-SPIEL*
   *****"
40:RANDOM
43:REM **********
44:REM
45:REM FELDDEKLAR
46:REM ATION NACH
47:REM SPIELERN
48:REM
49:REM **********
50:INPUT "WIEVIEL
   E MITSPIELER?
   ";N
55:IF N>5THEN
   PAUSE "ES SIND
    NUR 5 MOEGLIC
   H":GOTO 50
60:DIM Z(N),PM$(N
   ),GU$(N),RN$(N
   ),DZ(N),E(N,5)
   ,KT(N),G(N)
63:REM **********
64:REM
65:REM KONTEN AUF
66:REM FUELLEN
67:REM
68:REM **********
70:INPUT "WIEVIEL
    CHIPS HAT JED
   ER? ";X
80:FOR I=1TO N
85:LET KT(I)=X
90:NEXT I
92:GOTO 100
95:PRINT "NEUES S
   PIEL - NEUES G
   LUECK"
100:REM **********
101:REM
102:REM N SPIELER
103:REM SETZEN AUF
104:REM ZAHL, FARBE
105:REM GERADE/UNG
106:REM PASSE/MAN.
107:REM UND DUTZD
108:REM JE E CHIPS
109:REM
110:REM **********
115:PRINT "FAITES
    VOS JEUX!"
120:FOR J=1TO N
130:PRINT "DER ";J
    ;"-TE SPIELER
    SETZT!"
140:INPUT "GESETZT
   E ZAHL? (0-36)
   ";Z(J):INPUT "
   WIEVIEL CHIPS?
   ";E(J,1)
150:INPUT "FARBE R
   OT/SCHWARZ? (R
   /S)";RN$(J):
   INPUT "WIEVIEL
    CHIPS?";E(J,2
   )
160:INPUT "GERADE
   OD. UNGERADE?(
   G/U)";GU$(J):
   INPUT "WIEVIEL
    CHIPS?";E(J,3
   )
170:INPUT "PASSE O
   DER MANQUE? (P
   /M)";PM$(J):
   INPUT "WIEVIEL
    CHIPS?";E(J,4
   )
180:INPUT "WELCHES
    DUTZEND? (1,2
   ,3)";DZ(J):
   INPUT "WIEVIEL
    CHIPS?";E(J,5
   )
200:FOR K=1TO 5
210:LET KT(J)=KT(J
   )-E(J,K)
220:NEXT K
230:PRINT "SIE HAB
   EN NOCH";KT(J)
   ;" CHIPS."
250:NEXT J
280:GOSUB 700
285:REM **********
286:REM
287:REM RUECKKEHR
288:REM AUS SUBROU
289:REM TINE------
290:REM
291:REM ERGEBNIS
292:REM MITTEILUNG
293:REM
294:REM **********
295:RESTORE
299:WAIT 100
300:PRINT "MESDAME
   S ET MESSIEURS
   !"
310:PRINT "DAS ERG
   EBNIS LAUTET:"
311:WAIT 200
315:IF Z(0)=0THEN
   GOTO 350
320:PRINT Z(0);" /
    ";RN$;" / ";G
   U$
330:PRINT "    ";PM
   $;" / ";DZ$
340:GOTO 400
350:PRINT "ZERO, D
   IE NULL"
390:REM **********
391:REM
392:REM GEWINN DER
393:REM N SPIELER
394:REM WIRD ERMIT
395:REM TELT UND
396:REM DEM KONTO
397:REM ZUADDIERT
399:REM
400:REM **********
410:FOR K=1TO N
420:LET G(K)=0
430:IF Z(0)=Z(K)
   THEN LET G(K)=
   36*E(K,1)
440:IF RN$(0)=RN$(
   K)THEN LET G(K
   )=G(K)+2*E(K,2
   )
450:IF GU$(0)=GU$(
   K)THEN LET G(K
   )=G(K)+2*E(K,3
   )
460:IF PM$(0)=PM$(
   K)THEN LET G(K
   )=G(K)+2*E(K,4
   )
```

```
470:IF DZ(0)=DZ(K)
    THEN LET G(K)=
    G(K)+3*E(K,5)
480:LET KT(K)=KT(K
    )+G(K)
490:REM **********
491:REM
492:REM MITTEILUNG
493:REM VON GEWINN
494:REM UND KONTO-
495:REM STAND
496:REM
497:REM **********
500:PRINT "GEWINN
    SPIELER";K;":"
505:IF G(K)=0THEN
    GOTO 550
510:PRINT G(K);" C
    HIPS!"
520:PRINT "HERZLIC
    HEN GLUECKWUNS
    CH!"
530:PRINT "JETZT H
    ABEN SIE";KT(K
    );" CHIPS"
540:GOTO 590
550:PRINT "SIE PEC
    HVOGEL! ALLES
    WEG!"
560:PRINT "SIE HAB
    EN NOCH";KT(K)
    ;" CHIPS"
570:PRINT "KOPF HO
    CH!"
580:PRINT "VON NUN
     AN, VIEL GLUE
    CK!"
590:NEXT K
600:INPUT "NOCH EI
    N SPIEL? (J/N)
    ";W$
610:IF W$="J"OR W$
    ="JA"THEN GOTO
    95
620:PRINT "NA DENN
    , TSCHUESS!"
650:END

700:REM **********
701:REM
702:REM ----------
703:REM SUBROUTINE
704:REM ----------
705:REM
706:REM ZIEHEN DER
707:REM ZAHL UND
708:REM ERMITTELN
709:REM VON PM$(0)
710:REM GU$(0),
711:REM DZ(0) UND
712:REM RN$(0)
713:REM
714:REM **********
730:PRINT "RIEN NE
     VA PLUS!"
735:WAIT 300
740:PRINT "DIE KUG
    EL ROLLT!"
750:LET Z(0)=RND 3
    7
760:IF Z(0)=37THEN
    LET Z(0)=0
765:IF Z(0)=0THEN
    GOTO 940
770:IF Z(0)<=18
    THEN GOTO 790
780:LET PM$(0)="P"
    :PM$="PASSE":
    GOTO 800
790:LET PM$(0)="M"
    :PM$="MANQUE"
800:IF Z(0)/2-INT
    (Z(0)/2)=0THEN
    GOTO 820
810:LET GU$(0)="U"
    :GU$="UNGERADE
    ":GOTO 830
820:LET GU$(0)="G"
    :GU$="GERADE"
830:IF Z(0)<13THEN
    GOTO 860
840:IF Z(0)<25THEN
    GOTO 870
850:LET DZ(0)=3:DZ
    $="3.DUTZEND":
    GOTO 900
860:LET DZ(0)=1:DZ
    $="1.DUTZEND":
    GOTO 900
870:LET DZ(0)=2:DZ
    $="2.DUTZEND"

900:READ Y
910:IF Y=Z(0)THEN
    GOTO 940
915:IF Y=100THEN
    GOTO 930
920:GOTO 900
930:LET RN$(0)="S"
    :RN$="SCHWARZ"
    :GOTO 950
940:LET RN$(0)="R"
    :RN$="ROT":
    GOTO 950
945:LET PM$(0)="0"
    :LET GU$(0)="0
    ":RN$(0)="0":D
    Z(0)=0
950:RETURN
 1000:DATA 1,3,5,7
      ,9,12,14,16,
      18,19,21,23,
      25,27,30,32,
      34,36,100
```

		Spielers auf Zahl E(N,2):Einsatz auf Farbe E(N,3):Einsatz auf Gerade/Ung. E(N,4):Einsatz auf Passe/Manque E(N,5):Einsatz auf Dutzend	
7.	KT(N)	Kontostand des N-ten Spielers	beliebige numerische Konstante
8.	G(N)	"Gewinn" des N-ten Spielers	beliebige numerische Konstante
9.	N	Anzahl der Mitspieler	1-5

Die als "Gewinn" bezeichnete Variable G(N) stellt allerdings keinen Reingewinn dar, sondern nur die "Einnahmen" nach Ablauf des Spiels. Hiervon müssen die insgesamt gesetzten Chips abgezogen werden, wenn ein Spieler seinen tatsächlichen Reingewinn ermitteln will. Im Programm wird dies nicht vorgenommen.

Jetzt kommen wir aber zum Programmaufbau. Die Aufgaben 1 und 2 werden vom Programm in einer Subroutine durchgeführt (Zeile 700 - 1000). Es ist eigentlich nicht erforderlich, hier ein Unterprogramm zu verwenden, ich empfand es allerdings als übersichtlicher. Was geschieht nun in dieser Subroutine?
Zeile 750: Ziehen der Gewinnzahl mit RND und Zuweisen des Wertes auf Z(0). Da RND(X) nur Zahlen zwischen 1 und X ermittelt, mußte RND(37) genommen werden und in
Zeile 760 die eventuell gezogene 37 in eine 0 verwandelt werden.
Zeile 765: Da bei "Zero" weder Farbe noch sonst ein anderes Merkmal ermittelt werden muß, erfolgt ein Sprung nach
Zeile 945, wo diesen Merkmalsvariablen der Wert 0 zugewiesen wird. War das Ergebnis der Ziehung nicht "Zero", müssen nun die anderen Variablen mit dem richtigen Wert versehen werden:
Zeile 770 - 790 ermittelt den Wert von PM$(0) und PM$. PM$ ist eine Hilfvariable, die nur zum Ergebnisausdruck dient.
Zeile 800 - 820: Der Wert von GU$(0) wird festgestellt, analog zu oben auch der der Hilfsvariablen GU$.
Zeile 830 - 870: Ermittlung von DZ(0) und DZ.
Zeile 900 - 940: Der Wert der Variablen RN$(0) wird aus einer DATA-Liste gelesen, da es keine formale Regel dafür gibt, wann ein Wert rot oder Schwarz ist. In der DATA-Liste in
Zeile 1000 befinden sich alle Zahlen, die rot sind und am Schluß eine 100 als Abschlußkriterium der Liste.
Zeile 950 enthält den Rücksprungbefehl ins Programm. Das Ergebnis wird dann auch sofort in den
Zeilen 295-350 mitgeteilt.

Die Aufgabe 3, also das Setzen der Spieler zu überwachen, übernimmt das Programm in den Zeilen 100-250. Wie beim echten Roulette auch sollen sich die Spieler selbst entscheiden, worauf sie setzen wollen und

worauf nicht. Es werden aber in jedem Fall alle Setzmöglichkeiten abgefragt. In der Klammer des INPUT-Kommentars wird jeweils angegeben, was der Spieler in den Rechner eingeben soll. Beispiel:

ROT ODER SCHWARZ (R/S)

bedeutet, daß der Spieler seine Wahl mit R oder S angeben soll. Will ein Spieler nicht auf Farbe setzen, kann er hier, wie bei allen anderen Abfragen auch, eine beliebige Taste drücken und muß dann bei der Frage

WIEVIELE CHIPS

eine Ø eingeben. Das Programm ist in diesem Teil folgendermaßen aufgebaut:
Zeile120 und 250: Schleife für J = 1 bis N Spieler.
Zeile 140-180: Auffüllen der Variablenfelder Z(J), also bei N=5 Spielern der Variablen Z(1) bis Z(5), RN$(J), GU$(J), PM$(J), DZ(J) und jeweils gleichzeitig E(J,1) bis E(J,5) mit den Wetten und den Einsätzen der Spieler.

Die Aufgabe 4, also die Ermittlung des Gewinns jedes Spielers wird nach der Ergebnismitteilung in den Zeilen 390-590 angegangen.
Zeile 410 und 590: Schleife für K = 1 bis N Spieler
Zeile 420 Das Gewinnkonto G(K) jedes Spielers wird Null gesetzt.
Ziele 430: Z(0), also die Gewinnzahl, wird mit der Wette Z(K) jedes Spielers verglichen. Bei Gleichheit wird G(K) um den Einsatz E(K,1) erhöht.
Zeile 450: Die Farbe wird verglichen. Ggf. Erhöhung von G(K) um den 2-fachen Einsatz.
Zeile 450-470: Analoge Vorgehensweise für die Variablen GU$(K), PM$(K), DZ(K). Bei DZ(K) allerdings Erhöhung um den 3-fachen Einsatz.
Zeilen 500-550: Mitteilung des Gewinns G(K). (Kein Reingewinn!)

Für die Spieler muß noch das jeweilige Chip-Konto verwaltet werden (Aufgabe 5). Auch das geschieht in den Schleifen J und K, die wir gerade genauer angesehen haben. Verantwortlich ist die Variable KT(N). Sie wird zu Beginn des Programms in
Zeile 70-90 für jeden Spieler mit der gleichen Anzahl von Chips gefüllt. Die Verrechnung von Gewinn und Verlust erfolgt in zwei Stufen. In
Zeile 200-230 nach dem Setzen und in
Zeile 480 nach der Ermittlung des Gewinns. Mitgeteilt wird der Kontostand dem Spieler ebenfalls nach dem Setzen und am Ende des Spiels.
Zeile 230,530 und 560 übernehmen diese Aufgabe.

Als schmückendes Beiwerk, das Spiel soll ja auch Casino-Atmosphäre vermitteln, sind eine ganze Reihe von Kommentaren hinzugefügt, die sich von selbst erklären. Wenn Sie jetzt nach geduldiger und gelehrsamer Lektüre der letzten Seiten das Programm endlich selbst in Ihren Rechner eingeben, lassen Sie die ganzen REM-Zeilen, die Ihnen das Verständnis des Programms erleichtern sollten, einfach weg, denn die haben ja auf den Programmablauf keinen Einfluß. Gleich können Sie sich Ihrem Spieltrieb hingeben, nicht jedoch, ohne in seriöse Kleidung geschlüpft zu sein (Jacket und Krawatte sind beim Roulette nun mal vorgeschrieben).

ANHANG I

Musterlösungen

Lösung 1.1

$ACDC_{16} = 1010\ 1100\ 1101\ 1100_2$

Lösung 4.1

4+2*(2+3)∧2-SQR(4)*8/4*3 = 4+2*5∧2-2*8/4*3 =

4+2*25-2*8/4*3 = 4+50-12 = 42

Lösung 4.2

a) 1/5 + 1/4 + 1/20 = 10/20 =0.5

b) 3*(1+1/4) = 3 + 3/4 = 3.75

c) SQR(25+9) + 3∧(1+2) = SQR(34) + 3∧3 = 32.83095189

Lösung 5.1

Nicht erlaubte Variablennamen sind:
1X (s. Regel 2, S.28) SQR (s. Regel 3) "I" (s. Regel 2)
IF (s. Regel 3)

Lösung 5.2

Nicht erlaubte Zuweisungen sind:
Y = "SQR(4+3)" (s. Regel 4, S.28, numerische Ausdrücke stehen nicht in Anführungszeichen)
C+2 = 88.4-A/4 (s. S. 30)
F = "HOLLAREIDULLIHÖÖH" (s. Regel 4)

Lösung 5.3

Zu a): PREIS und PROST können nicht unterschieden werden (s.S. 28)
Zu b): Keine Kleinbuchstaben in Variablennamen! (s. Regel 7, S.29)

Lösung 5.4

Richtig muß es heißen:

```
20 : "MARKE": INPUT "EINGABE LAENGE";L
30 : INPUT "EINGABE BREITE"; B
40 : PAUSE "FLAECHENBERECHNUNG"
50 : LET X = L*B
60 : LET Y = L*L
70 : LET Z = B*B
80 : PAUSE "FLAECHE 1 =";X
90 : END
```

Zu Zeile 50-70: Bedenken Sie bitte, daß Ihr Computer in der Regel nur max. die ersten beiden Zeichen eines Variablennamens voneinander unterscheiden kann.

Lösung 5.5

```
10 : CLS
20 : INPUT "VERBRAUCH/100 KM IN L?";V
30 : INPUT "PREIS IN DM PRO L?";P
40 : LET K = V*P/100
50 : PRINT "KILOMETERGELD K = ";K
60 : END
```

Lösung 6.1

Nicht erlaubte Argumente:
SQR(-4) COS("ALPHA")
Zu e): TAN(PI/2) im RAD-Mode nicht erlaubt
Zu f): In Zeile 30 nimmt A einen negativen Wert an. Negatives Argument von LN in Zeile 40 nicht zulässig.

Lösung 6.2

a) 4 b) 32 c) 9 d) 7.347 e) 2

Lösung 6.3

Ergebnis: -3.144
Rechengang in Zeile 40:
Z = INT(-3.1442*1000+0.5)/1000 = INT(-3144.2+0.5)/1000 =
INT(-3143.7)/1000 = -3144/1000 = -3.144

Lösung 6.4

```
10 : CLS
20 : INPUT "RADIUS? ";R
30 : WAIT 200
40 : LET O = 4*PI*R^2
50 : LET V = 4*PI*R^3/3
60 : PRINT "OBERFLAECHE ";O
70 : PRINT "VOLUMEN ";V
80 : END
```

Lösung 6.5

```
10 : CLS
20 : DEGREE
30 : INPUT "A IN CM? ";A
40 : INPUT "B IN CM? ";B
50 : INPUT "G IN ALTGRAD? ";G
60 : LET F = A*B*SIN(G)/2
70 : PRINT "F IN CM^2 = ";F
80 : END
```

Ergebnis Testlauf: F IN CM^2 = 7.66044443

Lösung 7.1

```
10 : CLS
15 : CLEAR
20 : INPUT "UNTERE GRENZE ";U
30 : INPUT "OBERE GRENZE ";O
```

```
40 : LET S = S+U
50 : LET U = U+1
60 : IF U<= O THEN GOTO 40
70 : PRINT "SUMME S = ";S
80 : END
```

Sie müssen zuerst S = S+U, dann U = U+1 programmieren, weil sonst die untere Grenze U nicht zu S addiert wird. Um O in die Summe miteinzubeziehen, müssen Sie U<=O statt U<O programmieren.

Lösung 7.2

```
10 : CLS
15 : CLEAR
20 : INPUT "UNTERE GRENZE ";U
30 : INPUT "OBERE GRENZE ";O
40 : INPUT "SCHRITTWEITE ";I
50 : FOR A = U TO O STEP I
60 : S = S+A
70 : NEXT A
80 : PRINT "SUMME S = ";S
90 : END
```

Lösung 7.3

Folgende Zahlen werden ausgegeben:
Für A: 30 Für B: 12, 15, 18, 21

Lösung 7.4

```
10 : CLS
15 : CLEAR
20 : WAIT 200
30 : INPUT "ANFANGSKAPITAL IN DM ";A
40 : INPUT "JAHRE ";N
50 : INPUT "ZINSSATZ IN % ";P
60 : IF P<2 OR P>10 THEN PRINT "ES MUSS 2<= P<= 10
     SEIN " : GOTO 50
70 : LET K = A*(1+P/100)^N
80 : PRINT "ENDKAPITAL IN DM: ";K
90 : END
```

Es empfiehlt sich, den Zinssatz P erst in Zeile 50 einzulesen, da sonst bei einer Fehleingabe alle anderen INPUTs mit wiederholt werden müssen.

Lösung 8.1

Problemanalyse:
a) Einlesen von A, N und P
b) Abfage mit OR, da zwei Bedingungen: P<2 und P>10
c) Berechnung des Endkapitals K
d) Ausgabe des Endkapitals K
Programmablaufplan (PAP) siehe Abb. 5
Programm siehe Lösung 7.4

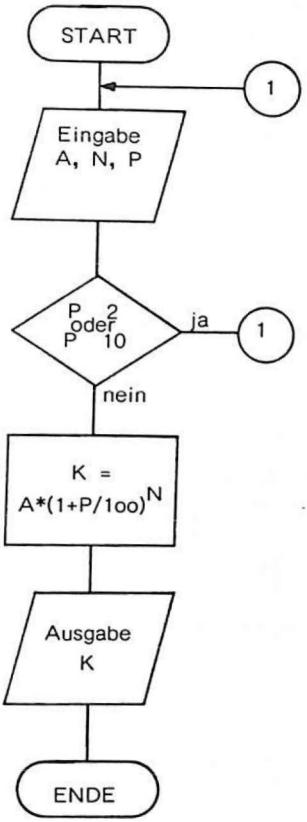


Abb. 5: PAP für das Programm ZINSRECHNUNG

Lösung 8.2

Problemanalyse:
a) Eingabe A, N und Q
b) Abfrage ob Q = 1. Wenn ja: S = N*A, wenn nein:
S = A*(1-Q∧N)/(1-Q)
c) Ausgabe S
Programmablaufplan siehe Abb. 6
Programm:

```
 5 : REM SUMME GEOMETRISCHE REIHE
10 : CLS
15 : WAIT 200
20 : INPUT "ANFANGSGLIED A ";A
```

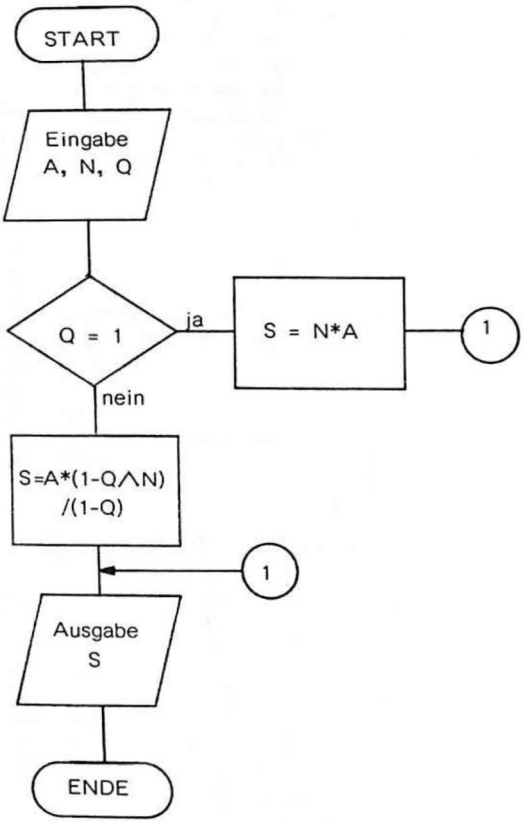


Abb. 6: PAP zum Programm GEOMETRISCHE REIHE

```
30 : INPUT "ZAHL DER GLIEDER ";N
40 : INPUT "FAKTOR Q ";Q
50 : IF Q = 1 THEN LET S = N*A : GOTO 70
60 : LET S = A*(1-Q∧N)/(1-Q)
70 : PRINT "SUMME S = ";S
80 : END
```

Ergebnis Testlauf:
a) S = 10 b) S = 1023

Lösung 9.1

```
 10 : REM FUNKTIONEN EXP, LN, 10^X, LOG
 20 : CLS: CLEAR
 30 : WAIT 200
 40 : GOTO 220
 50 : REM UNTERPROGRAMM EXP
 60 : LET Y = EXP(X)
 70 : PRINT "EXP(X) = ";Y
 80 : RETURN
 90 : REM UNTERPROGRAMM LN
 95 : IF X<= 0 THEN PRINT "X MUSS>0 SEIN" : RETURN
100 : LET Y = LN(X)
110 : PRINT "LN(X) = ";Y
120 : RETURN
130 : REM UNTERPROGRAMM 10^X
140 : LET Y = 10^X
150 : PRINT "10^X = ";Y
160 : RETURN
170 : REM UNTERPROGRAMM LOG
175 : IF X<= 0 THEN PRINT "X MUSS>0 SEIN" : RETURN
180 : LET Y = LOG(X)
190 : PRINT "LOG(X) = ";Y
200 : RETURN
210 : REM HAUPTPROGRAMM
220 : INPUT "WERT FUER X ";X
230 : PRINT "BITTE WAEHLEN SIE"
240 : PRINT "1 FUER EXP, 2 FUER LN"
250 : PRINT "3 FUER 10^X, 4 FUER LOG"
260 : INPUT I
270 : IF I = 1 THEN GOSUB 60
280 : IF I = 2 THEN GOSUB 95
290 : IF I = 3 THEN GOSUB 140
300 : IF I = 4 THEN GOSUB 175
310 : PRINT "NOCH EINE RECHNUNG?"
320 : INPUT "1 FUER JA ";K
330 : IF K = 1 THEN GOTO 220
340 : END
```

Beachten Sie bitte, daß bei den IF-Abfragen in Zeile 95 und 175 der Rücksprung mit RETURN erfolgen muß!

Lösung 10.1

Mehrere Lösungen denkbar, hier mein Vorschlag:

```
10 : REM **********
12 : REM TERMINE
14 : REM **********
15 : CLS : CLEAR
20 : INPUT "MIT WEM? ";WR$
30 : INPUT "WELCHES DATUM? ";DT$
40 : INPUT "UHRZEIT? ";ZT$
50 : INPUT "TREFFPUNKT? ";WO$
```

```
 60 : LPRINT "A C H T U N G !"
 70 : LPRINT "WICHTIGER TERMIN!"
 90 : LF 2
100 : LPRINT "TREFFEN MIT"
110 : LPRINT WR$
120 : LF 1
130 : LPRINT "AM:  ";DT$
140 : LPRINT "UM:  ";ZT$
150 : LPRINT "IM:  ";WO$
160 : LF 5
170 : END
```

Lösung 12.1

```
55 : LET Z1 = DEG(ZT) : W1 = DEG(WL)
60 : LET Z2 = Z1+W1
70 : LET ZT = DMS(Z2)
```

Allerdings:
Bei ZT$ = "12.50 UHR" und WL$ = "0.50 STD" erhalten Sie als Ergebnis: GEGEN 13.39 UHR BIN ICH .... Denn ZT hatte den Wert 13.395999. Wir brauchen also eine Rundungsroutine:

```
80 : LET ZT = INT(ZT*100+0.5)/100
```

Lösung 12.2

a)

```
20:A$="BASIC-LEHR
   BUCH"
30:FOR I=1TO 14
40:LET B$=RIGHT$
   (A$,I)
50:LPRINT B$
60:NEXT I
```

b)

```
20:A$="BASIC-LEHR
   BUCH"
30:FOR I=1TO 14
35:LET J=15-I
40:LET B$=RIGHT$
   (A$,I):C$=
   LEFT$ (A$,J)
50:LPRINT B$;"
   ";C$
60:NEXT I
```

c)

```
20:A$="BASIC-LEHR
   BUCH"
30:FOR I=1TO 14
35:LET J=INT (8-I
   /2)
40:LET B$=MID$ (A
   $,J,I)
50:LPRINT B$
60:NEXT I
```

d)

```
20:A$="BASIC-LEHR
   BUCH"
30:FOR I=1TO 14
40:LET B$=MID$ (A
   $,I,1)
50:LPRINT B$;B$;"
    ";B$;"  ";B$;
   "   ";B$;"
   ";B$
60:NEXT I
```

Lösung 13.1

```
 40 : INPUT N4$
150 : IF N1$ <N4$ THEN GOTO 160
155 : LET RE$ = N1$ : N1$ = N4$ : N4$ = RE$
160 : IF N2$ <N3$ THEN GOTO 170
165 : LET RE$ = N2$ : N2$ = N3$ : N3$ = RE$
170 : IF N2$ <N4$ THEN GOTO 180
175 : LET RE$ = N2$ : N2$ = N4$ : N4$ = RE$
180 : IF N3$ <N4$ THEN GOTO 210
185 : LET RE$ = N3$ : N3$ = N4$ : N4$ = RE$
240 : PRINT N4$
```

Listing enthält nur die veränderten und ergänzten Befehle

Lösung 13.2

Nur veränderte und ergänzte Befehlszeilen:

```
  5 : INPUT "LAENGE DER LISTE? ";L
 10 : DIM N$(L)
 20 : FOR E = 1 TO L
100 : FOR J = 1 TO (L-1)
110 : FOR I = (J+1) TO L
200 : FOR A = 1 TO L
```

Die Klammern bei (L-1) und (J+1) können bei den meisten Rechnern weggelassen werden.

Lösung 13.3

Hier gibt es keine richtige oder falsche Lösung. Empfehlenswert ist es, Schleifen graphisch abzusetzen, um ihren Anfang und ihr Ende zu verdeutlichen. Auch sollten Sie Ihrem Programm unbedingt einen Namen mit aufs Listing geben.

Lösung 13.4

Strings werden nicht nach dem numerischen Wert ihres Inhalts, sondern lexigraphisch nach dem ASCII-Code ihrer Zeichen sortiert. Schauen Sie nochmal auf Seite 98 dazu nach.

Lösung 13.5

Tauschen Sie einfach die String-Variable N$(...) überall gegen die numerische Variable N(...) aus. Jetzt können Sie Zahlen, aber keine Wörter sortieren.

Lösung 14.1

Siehe Abb. 12 auf S. 156. (Listing und Beispielausdruck)

Lösung 14.2

Nur veränderte und ergänzte Zeilen:

```
55 : LET VK = INT(VK*100+0.5)/100
85 : LET BR = VK*1.14 : BR = INT(BR*100+0.5)/100
90 : LPRINT "    INCL. USt:";BR
```

```
 5:REM **********
 6:REM PREISLISTE
 7:REM **********
10:LPRINT "PREISL
   ISTE EISEN-"
11:LPRINT "WAREN"
12:LF 1
13:LPRINT "UNTER
   BERUECKSICH-TI
   GUNG VON ";
20:INPUT "KUNDENR
   ABATT IN PROZE
   NT?";RB
25:LPRINT RB;" PR
   O-ZENT RABATT"
27:LF 3
30:FOR I=1TO 4
40:READ A$,NR$,ST
   $,UK
50:LET UK=UK-UK*R
   B/100
55:USING "###.##"
60:LPRINT A$;"
   ARTIKELNR. ";N
   R$
70:LPRINT "    ";S
   T$;" STUECK"
80:LPRINT "    IHR
    EK:   ";UK
90:LPRINT "    INC
   L.USt:";UK*1.1
   4
95:LF 1
100:NEXT I
200:DATA "HOLZSCHR
   AUBEN      ","0
   040","25",5.30
210:DATA "GEWINDES
   CHRAUBEN   ","0
   041","50",11.2
   0
220:DATA "SCHRAUBE
   NMUTTERN   ","0
   042","50",6.80
230:DATA "STAHLSTI
   FTE        ","0
   050","100",7.9
   0
```

Beispielausdruck:

```
PREISLISTE EISEN-
WAREN

UNTER BERUECKSICH-
TIGUNG VON  7 PRO-
ZENT RABATT



HOLZSCHRAUBEN
 ARTIKELNR. 0040
   25 STUECK
   IHR EK:     4.92
   INCL.USt:   5.61

GEWINDESCHRAUBEN
 ARTIKELNR. 0041
   50 STUECK
   IHR EK:    10.41
   INCL.USt:  11.87

SCHRAUBENMUTTERN
 ARTIKELNR. 0042
   50 STUECK
   IHR EK:     6.32
   INCL.USt:   7.20

STAHLSTIFTE
 ARTIKELNR. 0050
   100 STUECK
   IHR EK:     7.34
   INCL.USt:   8.37
```

Abb. 12: Listing und Beispielausdruck zu Programm PREISLISTE aus Aufgabe 14.1

Lösung 14.3

Als Lösungsbeispiel genüge ein Verzeichnis mit 3 Adressen. Das Programm bleibt dasselbe, wenn Sie die DATA-Liste entsprechend Ihrer Bedürfnisse erweitern.

```
10 : DIM NA$(0)*25, ST$(0)*25, SD$(0)*25, N$(0)*25
15 : RESTORE
20 : INPUT "NAME, VORNAME? ";N$(0)
30 : READ NA$(0),TL$,ST$(0),SD$(0)
40 : IF NA$(0) = "Ø" THEN GOTO 150
50 : IF NA$(0) = N$(0) THEN GOTO 70
60 : GOTO 30
70 : LPRINT NA$(0)
75 : LPRINT "TEL.: ";TL$
78 : LF 1
80 : LPRINT "ADRESSE:"
```

```
 85 : LF 1
 90 : LPRINT ST$(0)
 95 : LPRINT SD$(0)
 98 : LF 5
100 : INPUT "NOCH EINE AUSKUNFT? (J/N)";A$
110 : IF A$ = "N" THEN GOTO 300
120 : GOTO 15
150 : PAUSE "NAME NICHT IN LISTE" : GOTO 15
200 : DATA "MUELLER, HANS","0711/ 22 50 10",
      "HERDWEG 12","7000 STUTTGART 1"
210 : DATA "FREI, ROSI","089/ 32 16 8","VOR DER STADT 1",
      "8000 MUENCHEN 40"
220 : DATA "LANGENTALER, BIRGER";"07141/78 05 67",
      "SCHUSTERSTR. 24","7140 LUDWIGSBURG"
230 : DATA "Ø","Ø","Ø","Ø"
300 : END
```

Einige kurze Erläuterungen:
Zeile 10: Damit der Name, die Staße und die Stadt länger als 16 Zeichen sein dürfen, wurden eindimensionale Felder mit 25 Zeichen Länge deklariert. Zeile 20: Vergleichsname wird eingegeben. Zeile 40: "Ø" ist Abbruchkriterium für READ-Befehl. Zeile 50: Der Vergleichsname aus dem INPUT wird so lange mit den Namen in der DATA-Liste verglichen, bis er identisch ist oder das Abbruchkriterium erreicht ist. In Zeile 70-95 erfolgt der Adressenausdruck bei positivem Vergleich. Zeile 100 fragt, ob Programmlauf wiederholt werden soll. In Zeile 200-240 steht die DATA-Liste.

Lösung 14.4

Geänderte oder ergänzte Befehle:

```
30 : LPRINT TAB(X);"*";TAB(9);"I";
35 : COLOR 3
40 : LET Y = COS(I)*8.9+9
50 : LPRINT TAB(Y);"*"
55 : COLOR 0
60 : NEXT I
```

Lösung 15.1

Lösungsvorschlag (nur ergänzte Zeilen):

```
100 : LET X = SGN(KT) + 2
110 : ON X GOTO 120, 130, 140
120 : PRINT "KONTO UEBERZOGEN!!!"
125 : PRINT "MIT";ABS(KT);"IN DEN ROTEN" : GOTO 200
130 : PRINT "REICHT GERADE NOCH!" : GOTO 200
140 : PRINT "KONTO NOCH VOLL"
150 : INPUT "NOCH WAS KAUFEN? (J/N)";A$
160 : IF A$ = J THEN GOTO 90
```

Da SGN(KT) in Zeile 100 die Werte -1,0 oder +1 annehmen kann, wird X = 1, 2 oder 3. Genau richtig für ON ... GOTO.

ANHANG II

Stichwortverzeichnis

ABONNEMENT

Wenn es Ihnen Spaß gemacht hat, diese Ausgabe von "Alles für Sharp Computer" zu lesen, und Sie sich auch in Zukunft durch unsere interessante Zeitschrift über alles Wissenswerte zum Thema Sharp Computer informieren wollen, dann sollten Sie nicht länger zögern, "Alles für Sharp Computer" jetzt im regelmäßigen Bezug per Post zu bestellen. Sichern Sie sich eine lückenlose Information und schicken Sie den Bestellabschnitt am besten noch heute ab. "Alles für Sharp Computer" kommt dann regelmäßig jeden Monat ins Haus, ohne daß Ihnen zusätzliche Kosten entstehen.

Alles für SHARP Computer

Bestellschein Bitte vollständig und lesbar ausfüllen, unterschreiben und einsenden an Fischel GmbH, Kaiser-Friedrich-Str. 54a, D-1000 Berlin 12

☐ Ich abonniere die Zeitschrift "Alles für Sharp Computer"von der nächsten erreichbaren Ausgabe an (Preis pro Jahr 72 DM, Ausland 84 DM, Luftpostzuschlag 12 DM).

☐ Ich abonniere die Zeitschrift "Alles für Sharp Computer"von der Ausgabe ... (Monat) ... (Jahr) an (Preis pro Jahr 72 DM, Ausland 84 DM, Luftpostzuschlag 12 DM).

Das Abonnement verlängert sich um ein Jahr zu den dann jeweils gültigen Bedingungen, wenn es nicht 2 Monate vor Ablauf schriftlich gekündigt wird.

☐ Ich bestelle folgende schon erschienene Exemplare von "Alles für Sharp Computer" (Stückpreis 6 DM, Ausland 7 DM):
Heftnr.: ... , ... , ... , ... , ...

Alle Preise incl. 7 % Mwst.

Der Gesamtbetrag von ..... DM

☐ liegt bar bei

☐ liegt als Verrechnungsscheck bei (schnellste Erledigung)

☐ wurde am ..... auf das Postgirokonto der Fischel GmbH, Kontonr. 461533-103, BLZ 10010010, Postgiroamt Berlin überwiesen (Bearbeitung nach Zahlungseingang)

☐ liegt (nur bei kleineren Beträgen) in Briefmarken oder internationalen Antwortscheinen bei.

Name,Vorname .....

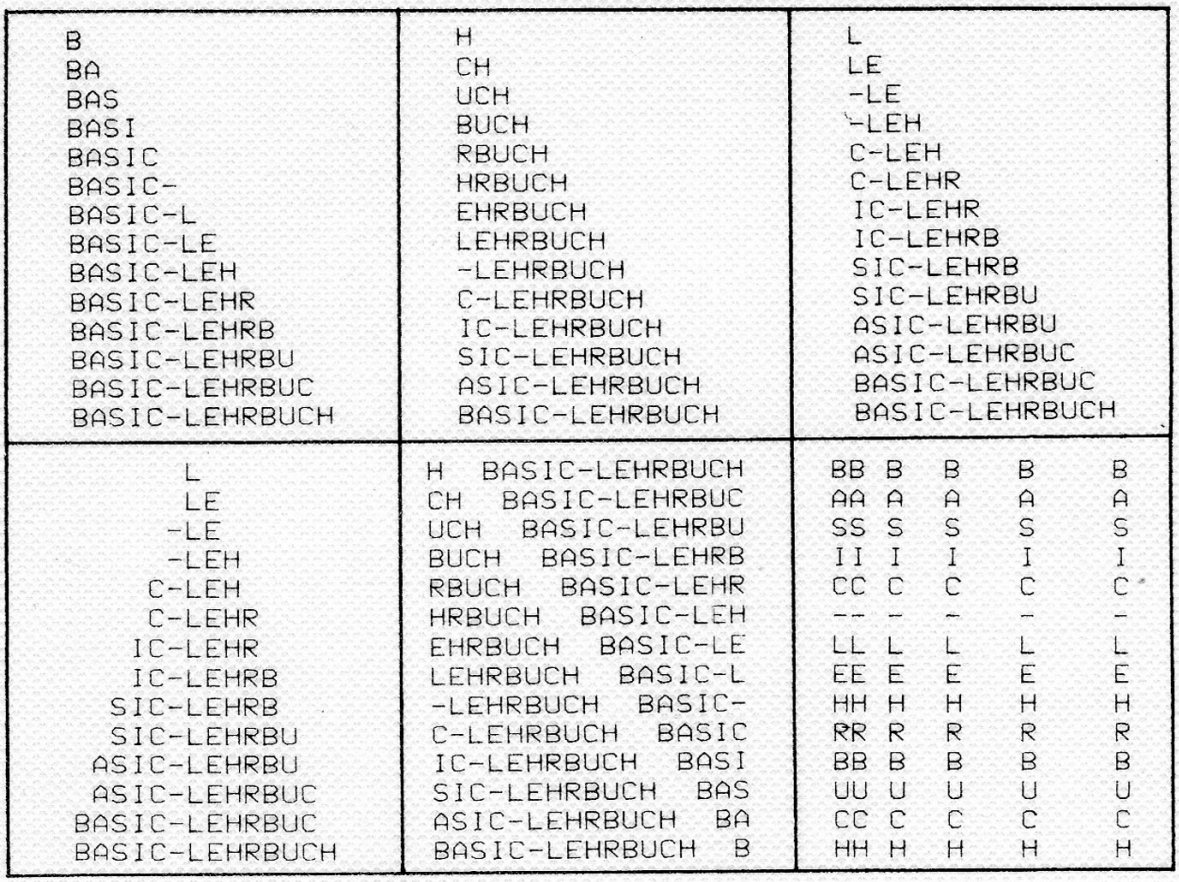
Straße .....

PLZ/Ort .....

Datum,Unterschrift .....

Mir ist bekannt, daß ich diese Bestellung innerhalb von 8 Tagen bei der Bestelladresse widerrufen kann. Zur Wahrung der Frist genügt die rechtzeitige Absendung. Ich bestätige dies durch meine zweite Unterschrift.

Datum,Unterschrift .....



Basic-Lehrbuch

Eine leicht verständliche Einführung
in das Programmieren mit BASIC

Bernhard Hartmann

und

Jürgen Brenner-Hartmann

FISCHEL GMBH

(ISBN: 3-924327-09-2)