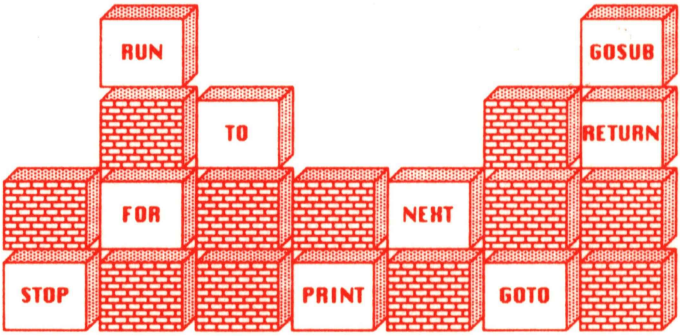
Kampow

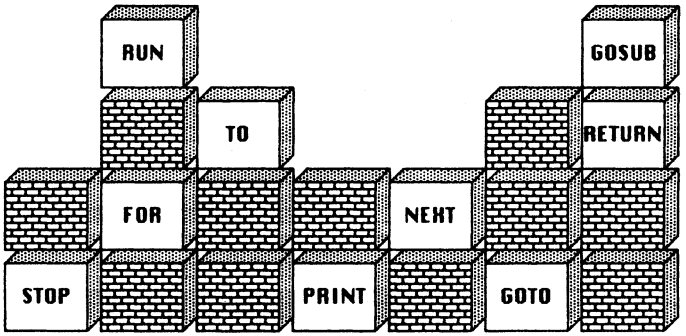
DAS BASIC TRAININGSBUCH ZUM COMMODORE 64



EIN DATA BECKER BUCH

Kampow

DAS BASIC TRAININGSBUCH ZUM COMMODORE 64



EIN DATA BECKER BUCH

ISBN 3-89011-023-1

2. Auflage

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 Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

VORWORT

Viele, die einen COMMODORE-64 besitzen, verwenden Standardprogramme oder tippen lange BASIC-Listings ab. Schon bei der Fehlersuche fangen aber die Probleme an. Da geht´s ohne solide BASIC-Kenntnisse nicht mehr. Hier bietet Ihnen dieses Buch eine echte Hilfe.

Das neue BASIC-Trainingsbuch ist eine ausführliche, didaktisch gut geschriebene Einführung in das CBM-BASIC-V2. Aber genau wie beim Erlernen einer Fremdsprache geht es nicht ohne Arbeit und Fleiß. Aber es lohnt sich, das Buch gründlich durchzuarbeiten, wobei Sie das letzte Kapitel auch später noch als Nachschlagewerk verwenden können. Viele praktische Tips werden gleich mitgeliefert, z.B. die Programmierung komfortabler Eingaberoutinen und bedienerfreundlicher Menues. Immer wird dabei auf strukturierten und modularen Aufbau der Programme Wert gelegt.

Viel Erfolg beim Lernen!

Dr. Achim Becker

INHALTSVERZEICHNIS

Inhaltsverzeichnis.....3
Einleitung.....7

1. KAPITEL

1. Grundlagen des Programmierens.....9
1.1 Allgemeine Erläuterungen.....9
1.2 Datenfluß- und Programmablaufpläne.....11
1.2.1 Datenflußpläne.....13
1.2.2 Programmablaufpläne.....16
1.3 Ascii-Codes.....21
1.4. Zahlensysteme.....22
1.4.1 Das Dualsystem.....23
1.4.2 Bit und Byte.....24
1.4.3 Das Hexadezimalsystem.....25
Aufgaben.....29
Lösungen.....30

2. KAPITEL

2. Einführung in das Programmieren mit Basic.....31
2.1 Das erste Basic-Programm.....31
2.1.1 Eingabe von Werten mit INPUT.....34
2.1.2 Ausgabe mit PRINT.....36
2.1.3 Kommentare mit REM.....40
2.2 Variablen und deren Benutzung in Programmen...41
2.2.1 Rechenoperationen mit Variablen.....43
Aufgaben.....45
Lösungen.....46
2.3 Numerische Funktionen.....49
2.3.1 Funktionen.....54
2.3.2 Zufallszahlen.....54
2.3.3 ASC(X$) und CHR$(X).....56
Aufgaben.....59
Lösungen.....60
2.4 TAB und SPC.....63

2.5 Strings....65
Aufgaben....75
Lösungen....76

3. KAPITEL

3. Erweiterte Programmstrukturen....77
3.1 Unbedingte Programmsprünge....77
3.2 Bedingte Programmsprünge....80
3.2.1 IF...THEN....80
Aufgaben....85
Lösungen....86
3.2.2 FOR...TO...NEXT....90
3.3 Berechnete Sprungbefehle....101
3.3.1 Beispielprogramm "Rechenlehrgang"....105
Aufgaben....115
Lösungen....116
3.4 Programmablaufsteuerung mit GET....118
3.4.1 Eingabe von Daten mit GET....118
3.4.2 Funktionstastenbelegung mit GET....125
3.5 FRE, POS, SYS, USR(X), WAIT....127
3.6 PEEK und POKE....130
3.7 READ, DATA und RESTORE....133

4. KAPITEL

4. Komplexere Basic-Programme....141
4.1 Felder....141
4.1.1 Eindimensionale Felder....141
4.1.2 Anwendungsbeispiele zu
eindimensionalen Feldern....149
Aufgaben....159
Lösungen....160
4.1.3 Mehrdimensionale Felder....163
Aufgabe....171
Lösung....172
4.2 Unterprogramme....174
Aufgabe....191
Lösung....192
4.3 Menütechniken....195

4.3.1 Verwendung von GET-Routinen im Menü.....200
4.4 Sortierverfahren.....220
4.5 Das Prinzip der Dateiverwaltung.....223

5. KAPITEL

5. Beschreibung der BASIC-Befehle.....229

ANHANG

1. Ascii-Codes.....295
2. Sachregister.....305

Dieses Buch ist, wie der Titel schon angibt, ein Trainingsbuch. Es wendet sich an all diejenigen, die sich - vor kurzem vielleicht - einen Commodore 64 angeschafft haben und meinen: "Nun kann es ja losgehen mit dem Programmieren."
Ganz so einfach ist die Sache allerdings nicht. Es gehört schon etwas dazu, sich den Computer nutzbar zu machen (wenn man nicht gerade auf die Software zurückgreift, die es zu kaufen gibt). Dieses Buch will Sie daher auf ganz systematischem Wege in das Programmieren mit Basic einweisen. Sie werden lernen, wie man ein bestimmtes Problem in ein Programm umsetzt und wie man rationell und durchschaubar dieses Programm schreibt.

Der 1.Teil befaßt sich zunächst mit den allgemeinen Grundlagen des Programmierens. Wie erreicht man einen guten Programmierstil? Wie dokumentiert man seine Programme? Auf diese Fragen werden Sie eine Antwort erhalten. Außerdem erfahren Sie ein wenig über die wichtigsten theoretischen Grundlagen der Datenverarbeitung.

Im 2. und 3.Teil geht es dann an die eigentliche Programmierarbeit. Zunächst lernen Sie anhand vieler Beispiele, wie bestimmte Basic-Befehle zu verwenden sind und wozu. Die Beispielprogramme sind übrigens - mit wenigen Einschränkungen - auch auf andere Rechner übertragbar, die über den gleichen Basic-Befehlssatz verfügen. Daher wurde in den Programmen auf eine übermäßige Verwendung der Befehle PEEK und POKE verzichtet. Diese Befehle beziehen sich auf rechnerspezifische Speicheradressen, die nicht ohne weiteres übertragbar sind.
Im Anschluß an die einzelnen Kapitel finden Sie Aufgaben, die Sie lösen sollten. Damit können Sie überprüfen, ob Sie die Schritte bis dahin nachvollziehen konnten. Die Lösungen sind natürlich auch angegeben und eingehend erklärt.

Der 4.Teil befaßt sich dann mit komplexeren Problemstellungen und damit auch mit komplexeren Programmen. Wie man auch damit zurechtkommt, will Ihnen dieser Teil zeigen. Auch hier finden Sie wieder viele Beispiele, außerdem Aufgaben - denn Sie sollen ja "trainieren" und nicht nur lesen - und Lösungen.

Im 5.Teil dieses Buches werden noch einmal alle Basic-Befehle alphabetisch aufgeführt und kurz erklärt. Diesen 5.Teil können Sie dann später wie ein "Nachschlagewerk" benutzen.

Und nun bleibt eigentlich nur noch, Ihnen viel Spaß und viel Erfolg bei der Arbeit mit diesem Buch zu wünschen. Und nicht verzweifeln, wenn es einmal nicht sofort klappt! Erstens ist noch kein Meister vom Himmel gefallen. Und zweitens: Die Programmierarbeit verlangt auch ein wenig Ausdauer und Spaß am "Tüfteln".

1. GRUNDLAGEN DES PROGRAMMIERENS

1.1. ALLGEMEINE ERLÄUTERUNGEN

In diesem Kapitel geht es zunächst um die Grundlagen des Programmierens. Bevor anhand einfacher und später komplexerer Aufgaben das Programmieren mit den BASIC-Befehlen gezeigt wird, wird hier zunächst Grundsätzliches zur Programmierung gesagt, d.h. es wird erklärt, wie man ein Problem in ein Programm umsetzt. Dieses bißchen Theorie mag zwar anfangs trocken erscheinen, ist aber hilfreich und notwendig, um später auch mit komplexeren Programmen zurechtzukommen.

Was heißt eigentlich Programmieren ?

Sie müssen davon ausgehen, daß ein Computer nach dem Einschalten im Prinzip "dumm" ist, d.h. er hat zwar irgendeine Programmiersprache fest eingebaut, aber Sie können nicht einfach über die Tastatur eingeben "Berechne die Oberfläche einer Kugel.". Wollen Sie dieses Problem durch den Computer lösen lassen, so müssen Sie ihm vorher in der Sprache des Computers in eindeutiger, logisch bestimmter Reihenfolge mitteilen, was er zu tun hat. Den Lösungsweg, den Sie dadurch bestimmen, nennt man ALGORITHMUS. Die gesamte Folge von Anweisungen nennt sich dann PROGRAMM.

Die Sprache ist im Falle des COMMODORE 64 BASIC. BASIC wurde im Jahre 1961 am Darthmouth College in New Hampshire (USA) entwickelt und setzt sich aus den Anfangsbuchstaben von BEGINNER´S ALL purpose SYMBOLIC INSTRUCTION CODE zusammen, was soviel heißt wie "Symbolischer Allzweck Befehlscode für Anfänger".
BASIC wurde aus der Programmiersprache FORTRAN entwickelt. Inzwischen haben sich allerdings auf den verschiedenen Computern verschiedene BASIC-Dialekte herausgebildet, sodaß das BASIC des COMMODORE 64 nicht direkt auf andere Computer anwendbar ist. Es unterscheidet sich zwar meistens nur in Kleinigkeiten, dennoch sollte man wissen, daß die mit dieser

BASIC-Version erstellten Programme bei Bedarf angepaßt werden müssen.

Der Computer versteht nun allerdings die einzelnen BASIC-Befehle nicht direkt. Diese müssen erst in einen entsprechenden Code, die sogenannte Maschinensprache, übersetzt werden, mit dem der Computer dann arbeiten kann. Diese Übersetzung der BASIC-Befehle übernimmt der BASIC-INTERPRETER im Computer. Geben Sie nun einen BASIC-Befehl über die Tastatur in den Computer ein und drücken die RETURN-Taste, so wird dieser Befehl erst über den INTERPRETER geleitet, dort in den computereigenen Code umgewandelt und dann erst ausgeführt.
Zusammenfassend kann man also sagen, daß man unter Programmieren die Übersetzung eines ALGORITHMUS in eine Programmiersprache, in unserem Falle BASIC, versteht.

Nun wird aber von Anfängern, jedoch auch von vielen Fortgeschrittenen, meistens in der folgenden Art und Weise vorgegangen:
Herr Müller möchte sich zum Beispiel bei vorgegebenem Radius den Rauminhalt einer Kugel für 20 verschiedene Radien berechnen lassen. Die Formel wird ruckzuck aus der Formelsammlung abgelesen, demnach ist das Volumen einer Kugel $V=4\pi r^3/3$, und in den Computer gehämmert. Das Programm könnte dann ungefähr so aussehen:

```
10 FOR I=1 TO 20
20 INPUT"WELCHER RADIUS (IN CM)";R
30 V=4*π*R↑3/3
40 PRINT"DAS VOLUMEN BETRAEGT ";V;" ccm"
50 NEXT I
```

Das Programm läuft dann zur vollsten Zufriedenheit, Herr Müller hat seine Ergebnisse; was, werden Sie fragen, will er mehr? Herr Müller hat ja einen Algorithmus für sein Problem gefunden und diesen auch in BASIC übersetzt. Bei solch kleinen Programmen wird man immer wieder dazu verleitet, auf diese Weise vorzugehen. Ich muß Ihnen unter Vorbehalt Recht geben, wenn Sie jetzt sagen:" Warum soll man denn noch mehr Aufwand treiben?"

Sobald jedoch die Problemstellungen und somit die Programme komplexer werden, rächt sich diese Einstellung, da Sie den DATENFLUß und den PROGRAMMABLAUF nicht mehr auf Anhieb überblicken können. So kann es z.B. passieren, daß ein Programm falsch abläuft, womit Sie dann ganz einfach "falsche" Ergebnisse bekommen. Sie haben dann irgendwo einen logischen Fehler im Programm eingebaut und das Programm läuft nicht so ab, wie Sie es sich vorgestellt haben. Das liegt ganz einfach daran, daß der Mensch im allgemeinen Schwierigkeiten hat, sich in die Arbeitsweise eines Computers hineindenken zu können. Damit der Computer für uns ein Problem lösen kann, müssen wir es in viele kleine Einzelschritte zerlegen, die der Computer dann erst der Reihe nach abarbeiten kann. Gerade bei dieser Zerlegung und der Zusammenstellung der Reihenfolge dieser Teile unterlaufen uns immer wieder Fehler. Es ist also von der Aufgabenstellung bis zum fertigen Programm doch komplizierter, als es zuerst den Anschein hatte. Deswegen legt man i.A. einen Zwischenschritt ein, in dem man festlegt, was der Computer in welcher Reihenfolge tun soll.

1.2 DATENFLUß- UND PROGRAMMABLAUFPLÄNE

Es wurden nun zwei neue Begriffe verwendet, nämlich DATENFLUß und PROGRAMMABLAUF. Wie Sie sicherlich schon ahnen, stehen diese Begriffe mit dem o.a. Problem im direkten Zusammenhang. Der erwähnte Zwischenschritt besteht nun in der Erstellung von DATENFLUß- und PROGRAMMABLAUFPLÄNEN nach DIN 66001. Dieses Kapitel soll Ihnen eine kurze Einführung in diese Technik geben.
Zur Erstellung von Datenfluß- und Programmablaufplänen werden Symbole benutzt, die auf einer sogenannten Programmierschablone verfügbar sind. Diese Schablonen erhalten Sie in Schreibwarengeschäften, wo auch andere Zeichenschablonen erhältlich sind. Eine solche Schablone sehen Sie in Bild 1 abgebildet. Auf ihr findet man alle wichtigen Symbole für die Datenfluß- und Programmablaufpläne.

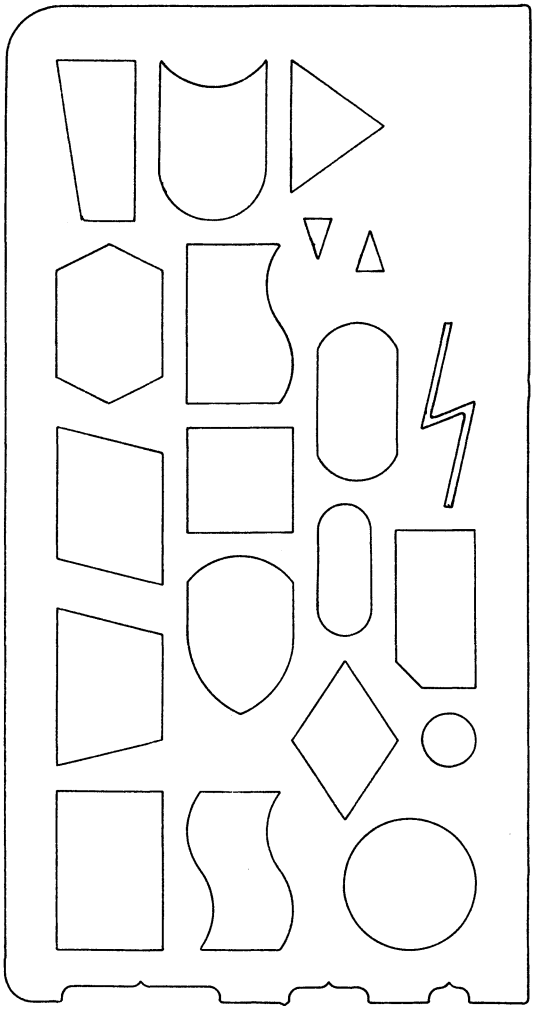
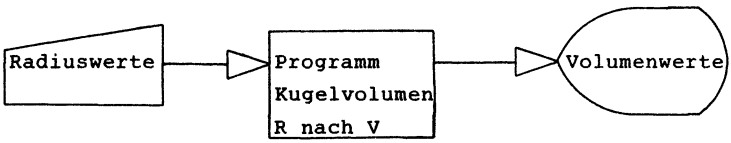


Bild 1

1.2.1 DATENFLUßPLÄNE

Datenflußpläne sollen, wie der Name schon sagt, den Datenfluß innerhalb eines Programms verdeutlichen. Genaugenommen sollen sie zeigen, welche Daten (z.B. Radiuswerte) wie in den Computer gelangen (z.B. per Hand über die Tastatur), durch welche Programme die Daten verarbeitet werden (z.B. Berechnung Kugelvolumen), und wie diese Daten wieder ausgegeben werden (z.B. auf dem Bildschirm). Anhand des Programms, welches bei gegebenem Radius das zugehörige Kugelvolumen berechnet, will ich Ihnen nun zeigen, wie der entsprechende Datenflußplan dazu aussieht.



Eingabe über Tastatur | Bearbeitung | Anzeige auf Bildschirm

Bild 2

Sie sehen, selbst für ein solch kleines Programm zur Volumenberechnung einer Kugel läßt sich ein Datenflußplan erstellen. Das mag Ihnen zwar lächerlich erscheinen, trotzdem sollte Ihnen diese Prozedur in Fleisch und Blut übergehen. Bei größeren Programmen werden Sie diese Datenflußpläne nicht mehr missen wollen. Diese Pläne können bei großen Programmen durchaus mehrere Seiten lang sein. Die Wege, auf denen die Daten verarbeitet werden, lassen sich dann anhand dieser Pläne mühelos nachvollziehen. Sie müssen zugeben, daß aus dem Listing solch großer Programme die Daten nur noch mit großer Mühe zu verfolgen sind und dann auch wahrscheinlich nur vom Programmierer selber. Haben Sie sich also rechtzeitig an die Erstellung solcher Datenflußpläne gewöhnt, so fällt es Ihnen umso leichter, sie auf größere Programme anzuwenden.
Die Bedeutung der einzelnen Symbole für die Datenflußpläne entnehmen Sie bitte dem Bild 3.

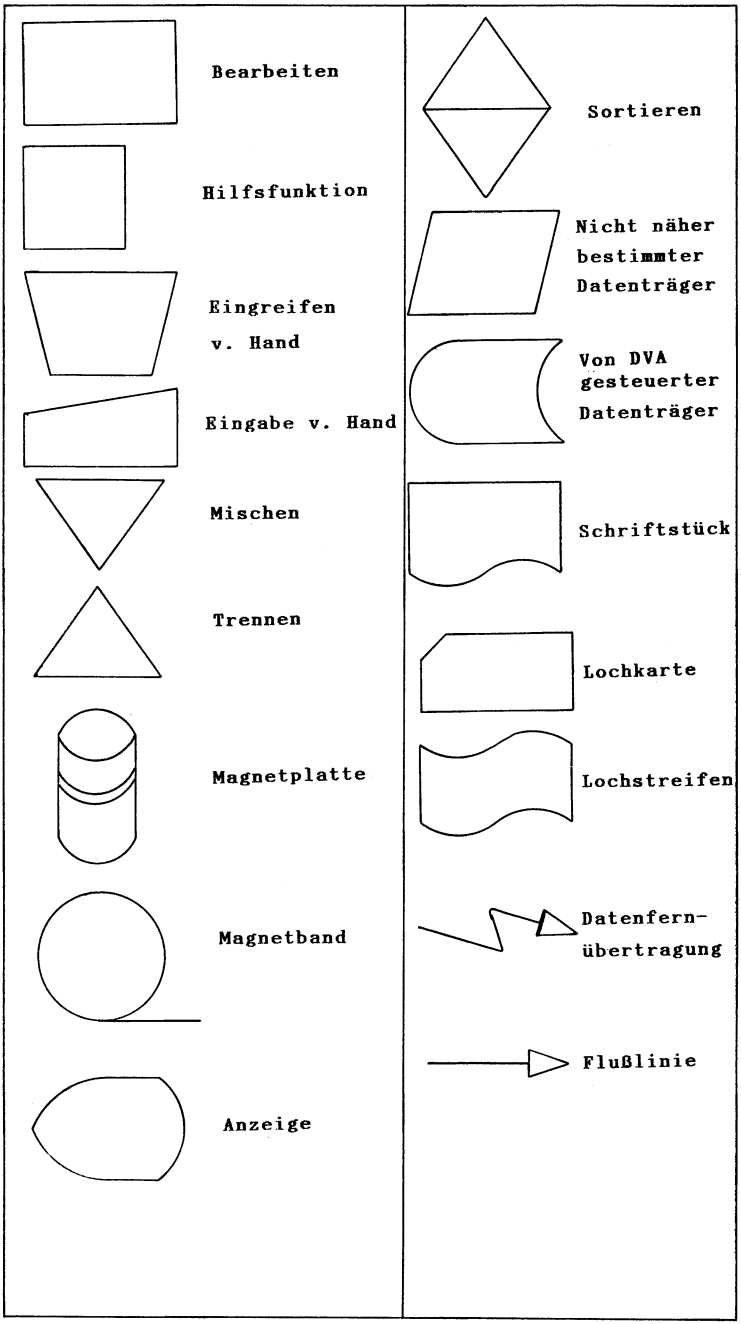

Bearbeiten
Hilfsfunktion
Eingreifen
v. Hand
Eingabe v. Hand
Mischen
Trennen
Magnetplatte
Magnetband
Anzeige
Sortieren
Nicht näher
bestimmter
Datenträger
Von DVA
gesteuerter
Datenträger
Schriftstück
Lochkarte
Lochstreifen
Datenfern-
übertragung
Flußlinie

Bild 3

Versuchen Sie nun einmal, einen Datenflußplan für ein Programm zu erstellen, welches Ihnen Meilen in Kilometer umrechnet und das Ergebnis auf dem Bildschirm anzeigt. Nun, Sie hatten die Lösung sicherlich schnell zur Hand. Vergleichen Sie Ihren Datenflußplan aber trotzdem mit dem Lösungsvorschlag in Bild 6.

Wir wir in diesem Kapitel also gelernt haben, dienen Datenflußpläne der übersichtlichen Darstellung, welche Daten auf welchen Datenträgern in den Computer gelangen, durch welche Programme diese Daten zu anderen Daten verarbeitet werden und auf welchen Datenträgern die Daten zur Ausgabe gelangen.

Wir wollen uns nun mit der zweiten Stufe des vorhin erwähnten Zwischenschritts befassen, dem PROGRAMMABLAUFPLAN, abgekürzt PAP genannt. Da der Datenflußplan ja nicht Auskunft darüber gibt, wie z.B. die Radiuswerte in die Volumenwerte umgerechnet werden, benötigen wir noch eine zweite Form der symbolischen Darstellung, die uns sagt, in welchen Einzelschritten der Rechner ein Problem lösen soll.

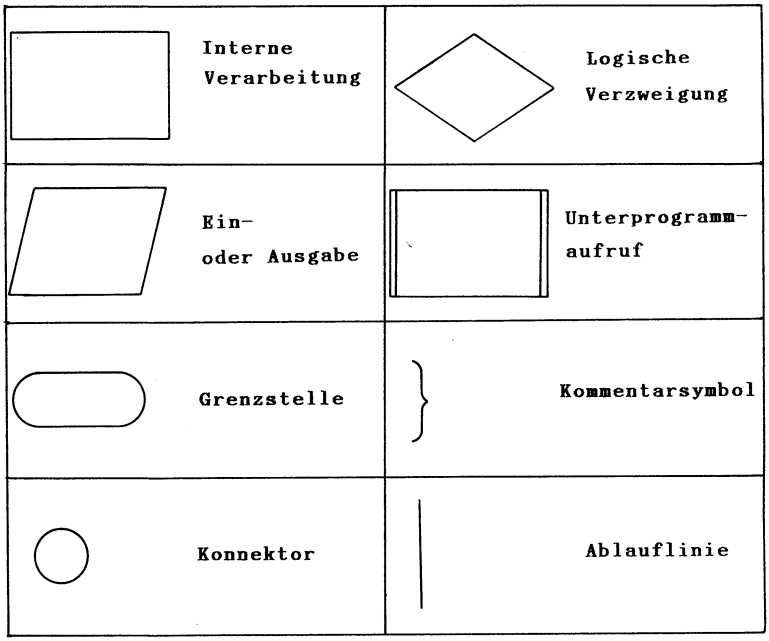


Bild 4

1.2.2 PROGRAMMABLAUFPLÄNE

Im Datenflußplan für die Berechnung des Kugelvolumens wird unter dem Punkt "Bearbeitung" lediglich "Programm Kugelvolumen R nach V" angeführt. Daraus geht aber nur hervor, was mit den eingegebenen Daten geschieht. Das eigentliche Problem wurde noch nicht in Einzelschritte zerlegt. Diese Aufgabe übernehmen nun die Programmablaufpläne. Sie sollen in überschaubaren Einzelschritten zeigen, was ein Computer machen soll, um ein bestimmtes Problem zu lösen. Auch bei den Programmablaufplänen werden wieder Symbole verwendet, die der DIN 66001 entsprechen und die auch auf der Programmierschablone zu finden sind. Diese Symbole sind in Bild 4 näher erläutert.
An unserem bekannten Beispiel wollen wir nun an die Erstellung unseres ersten Programmablaufplans gehen. Selbst bei solch kleineren Programmen sollte man ruhig dazu übergehen, sich PAPs zu erstellen, damit man nicht später bei komplexeren Programmen Schwierigkeiten mit der Umsetzung bekommt. Auch hier gilt der alte Spruch "Übung macht den Meister".

Programmablaufpläne werden immer von oben nach unten gezeichnet. Erreichen sie das untere Ende des Blattes, so wird rechts daneben der sich daran anschließende Teil gezeichnet. Gewöhnen Sie sich es erst gar nicht an, diese Trennstellen durch Linien zu verbinden. Für solche Fälle gibt es den sogenannten KONNEKTOR. Dieses Verbindungssymbol (siehe Bild 4) wird an das untere Ende des Plans gesetzt und mit einer Zahl oder einem Buchstaben gekennzeichnet. Der zweite Konnektor wird mit dem gleichen Buchstaben bezeichnet und an den Anfang des zweiten Teils gesetzt. Dazu schauen Sie sich nun bitte das folgende Beispiel eines Programmablaufplans in Bild 5 an.

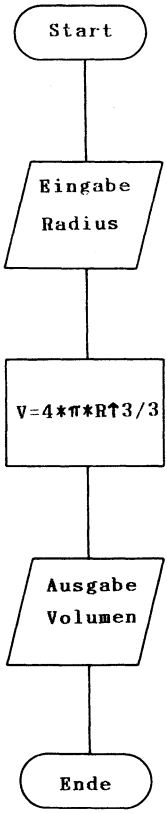


Bild 5

Das Start- bzw. Ende-Symbol kann man nicht in Basic übersetzen. Das Eingabesymbol "EINGABE RADIUS" kann man mit dem BASIC-Befehl INPUT übersetzen. Dieser kann noch mit einem Kommentar versehen werden wie "WELCHER RADIUS IN CM". Die Formel für die Berechnung des Kugelvolumens kann direkt in das Symbol für die interne Verarbeitung übernommen werden. Für das Ausgabesymbol "Ausgabe Volumen" benutzen wir den PRINT-Befehl, der mit einem entsprechenden Text versehen wurde. Im Gegensatz zu unserem früheren Beispielprogramm wurde hier keine FOR-NEXT Schleife benutzt. Sie sehen, daß bei einem Programmablaufplan, wenn er einen gewissen Grad der Verfeinerung erreicht hat, im Prinzip die einzelnen Symbole nur noch in die entsprechende Programmiersprache übersetzt werden brauchen.

Haben Sie diesen Stand bei der Programmierung erreicht, so können Sie an den ersten Testlauf Ihres Programms denken. Dieser findet zuerst auf dem Papier statt, d.h. Sie verfolgen noch einmal die Daten anhand des Datenflußplanes und überprüfen den Programmablauf anhand des PAP's. Fällt alles zu Ihrer Zufriedenheit aus, können Sie daran gehen, das Programm mit RUN zu starten.

Versuchen Sie jetzt bitte, einen eigenen PAP für die folgende Problemstellung zu schreiben: Sie wollen Celsiuswerte von einem Programm in Fahrenheitwerte

umrechen lassen. Die Formel dazu lautet: "F=1.8*C+32". Sie werden die Lösung sicher rasch gefunden haben. Vergleichen Sie sie aber trotzdem wieder mit dem Lösungsvorschlag in Bild 7.

Bei größeren Programmen werden die Vorteile dieser Programmablaufpläne erst richtig deutlich. Durch ihre graphische Darstellung sind sie leicht überschaubar, was man von einem Programmlisting nicht unbedingt behaupten kann. Ein anderer Vorteil, den man oft übersieht oder nicht hoch genug einschätzt, ist der, daß Programmablaufpläne unabhängig von einem bestimmten Rechner sind. Das bedeutet im Endeffekt, daß Ihr einmal erstellter PAP auf jeden beliebigen Rechner umsetzbar ist. Weiterhin stellen sie ein nicht zu unterschätzendes Dokumentationshilfsmittel für Ihre Programme dar.
Hier wurde soeben ein neuer Begriff verwendet, nämlich DOKUMENTATION. Die Programmdokumentation wird von vielen Programmierern sträflich vernachlässigt. Das Ende vom Lied ist dann, daß irgendwann eine Programmveränderung vorgenommen werden soll und dann ist es schon geschehen, daß einige Programmierer ihr eigenes Programm nicht mehr verstanden haben. Das liegt ganz einfach daran, daß sich kaum jemand an Kleinigkeiten erinnern kann, die er vielleicht vor einem Jahr in seinem Programm untergebracht hat. Deshalb sollte man es sich angewöhnen, zu seinem Programm eine Dokumentation zu erstellen. Diese sollte so gehalten sein, daß man das Programm auch noch nach mehreren Monaten versteht.
Soweit die Kapitel zu Datenfluß- und Programmablaufplänen. Wollen Sie sich mehr mit dieser Materie befassen, so sei hier auf die entsprechende Fachliteratur verwiesen. Im "64 für Profis", aus dem Hause DATA BECKER, wird dieses Thema ausführlicher behandelt.

Wir wollen jetzt noch einmal zusammenfassen, aus welchen Stufen sich das eigentliche Programmieren zusammensetzen sollte.

1. Definition des Problems (Erarbeiten der Problemstellung, Problemanalyse)

2. Entwurf des Algorithmus´ zur Lösung (Datenfluß- und Programmablaufpläne)

3. Umsetzen des Algorithmus´ in eine Programmiersprache (Erstellen des Programms)

4. Testlauf des Programms

5. Dokumentation

Lösungsvorschlag Bild 6

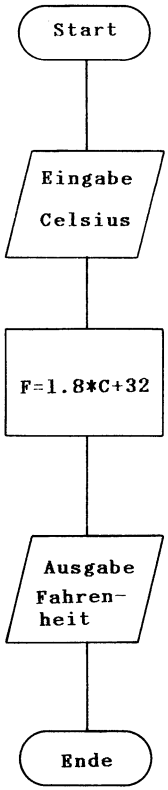


Bild 7

1.3 ASCII-CODES

Der COMMODORE 64 kann, wie Sie vielleicht wissen, die Zeichen, die Sie über die Tastatur eingeben, nicht direkt verarbeiten. Diese werden in einen Zahlencode übersetzt. Der gebräuchlichste Zahlencode ist der ASCII-Code. ASCII steht für "American Standard Code for Information Interchange", was soviel heißt wie "Amerikanischer Standardcode für den Informationsaustausch". Er wurde entwickelt, um einen Datenaustausch auch zwischen verschiedenen Informationsträgern zu gewährleisten, d.h. daß z.B. das Zeichen "A" im ASCII-Code immer den Wert 65 hat. Wird nun diese Zahl an einen Computer oder Drucker gesendet, der ebenfalls mit dem ASCII-Code arbeitet, wird dieser Wert immer als das Zeichen "A" interpretiert. Dabei hat die Entfernung zwischen Sender und Empfänger keinerlei Bedeutung. Ob Sie nun über die Tastatur Zeichen in den Computer eingeben - diese werden ja ebenfalls über eine Leitung an den Rechner weitergeleitet - oder ob Sie über ein Telefonmodem Ihre Daten, z.B. nach Amerika, übertragen, sobald der Empfänger den Wert 65 erhält, wird dieser in ein "A" übersetzt. Der Standard-ASCII-Code benutzt die Werte von 0 bis 127.
Die meisten Computerhersteller haben sich allerdings für einen erweiterten ASCII-Code entschlossen, um auch Zeichen nach eigenem Belieben darstellen zu können. Dieser Code wird auch ASCII-Code genannt, obwohl er mit dem Standard-ASCII-Code nicht in allen Werten übereinstimmt. Beim Standard-ASCII-Code werden die Zahlen 32-90 für Großbuchstaben und die Zahlen 91-127 für Kleinbuchstaben und einige andere Zeichen verwendet. Der COMMODORE 64 ASCII-Code ist nur im Bereich zwischen den Zahlen 32-91 und noch mit wenigen anderen Zahlen mit dem Standard-ASCII-Code identisch. Genaueres entnehmen Sie bitte den Tabellen im Anhang.

1.4 ZAHLENSYSTEME

Der Computer kann nur zwei Zustände in seinen elektronischen Schaltkreisen unterscheiden, nämlich "AN" und "AUS". Diese beiden Zustände mußten nun in ein Zahlensystem übertragen werden. Was lag da näher als das DUALSYSTEM. Im Dualsystem werden die Zahlen, die wir vom Dezimalsystem her kennen, nur mit den Zahlen 0 und 1 dargestellt. Dabei steht die 1 für den Zustand "EIN" und die 0 für den Zustand "AUS". Zur Erklärung des Dualsystems gehen wir vom bekannten Dezimalsystem aus. Man kann jede Dezimalzahl in ein beliebiges anderes Zahlensystem umwandeln. Wir können im Dezimalsystem für die Zahl 5678 auch folgendes schreiben:

$$5678 = 5*1000 + 6*100 + 7*10 + 8*1$$

oder auch

$$5678 = 5*10^3 + 6*10^2 + 7*10^1 + 8*10^0$$

Anmerkung: In der Mathematik hat eine beliebige Zahl hoch Null immer den Wert 1.

Im Dezimalsystem können also die Zahlen in einer Summe von einzelnen Produkten zur Basis 10 dargestellt werden. Jede Ziffer ist einer bestimmten Zehnerpotenz zugeordnet.

10^3	10^2	10^1	10^0
5	6	7	8

Diese Zahl kann noch zusätzlich mit dem Index 10 gekennzeichnet werden, um sie dem Dezimalsystem zuzuordnen und um sie in diesem Kapitel von anderen Zahlen unterscheiden zu können.

$$(5678)_{10}$$

1.4.1 DAS DUALSYSTEM

Das Dualsystem basiert auf dem gleichen Prinzip, nur mit dem Unterschied, daß die Basis 2 ist. Daraus ergibt sich dann, daß nur die Ziffern 0 und 1 Verwendung finden. Um nun die Dualzahl 1011_2 in eine Dezimalzahl umzuwandeln, gehen wir wie folgt vor:

Die Stellen der einzelnen Ziffern entsprechen wie beim Dezimalsystem wieder den einzelnen Potenzen, in diesem Falle den 2er-Potenzen. Wollen wir nun die Dualzahl umwandeln, schreiben wir jede Ziffer unter ihre zugehörige 2er-Potenz. Das Ganze wird zum Schluß nur noch addiert und schon haben wir unsere Dezimalzahl.

2^3	2^2	2^1	2^0
1	0	1	1

Somit ergibt sich folgende Summe mit den Teilprodukten:

$$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 11$$

$$1*8 + 0*4 + 1*2 + 1*1 = 11$$

Als Ergebnis erhalten wir die Dezimalzahl 11. Wollen Sie nun eine Dezimalzahl in eine Dualzahl umwandeln, so gehen Sie wie folgt vor:

Nehmen wir an, Sie wollen die dezimale Zahl 167 in eine Dualzahl umwandeln, so überlegen Sie, welche höchste Potenz von 2 sich in dieser Zahl unterbringen läßt. In unserem Falle ist das $2^7 = 128$. Dieser Wert wird von der umzurechnenden Zahl subtrahiert. Bei dem Rest von 39 wird in der gleichen Art verfahren. Höchste Potenz von 2 ist hier $2^5 = 32$ Rest 7. Höchste Potenz von 2 ist dann $2^2 = 4$ Rest 3 usw. Haben wir so alle vorkommenden Potenzen von 2 ermittelt, schreiben wir eine 1 unter die Potenz von 2, die in der Zahl enthalten ist. Unter alle anderen Potenzen wird eine Null geschrieben. Das sieht dann wie folgt aus:

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

Bilden wir jetzt wieder die Summe mit den Teilprodukten der 2er-Potenzen, unter denen eine 1 steht, so erhalten wir wieder unsere dezimale Zahl, von der wir ausgegangen sind, nämlich 167.

1.4.2 BIT UND BYTE

Es wurde oben bewußt eine dezimale Zahl genommen, die kleiner als 255 ist. Es genügen nämlich somit zur Darstellung im Dualsystem 8 Ziffern bzw. 8 Potenzen zur Basis 2. Die kleinste Informationseinheit, die ein Computer verarbeitet, nennt man BIT. Ein Bit kann zwei Zustände oder Werte haben: 0 oder 1. Man spricht auch von einem gesetzten Bit beim Wert von 1, oder von einem nicht gesetzten Bit beim Wert von 0. Der COMMODORE 64 besitzt einen 8-Bit Prozessor, d.h. daß er in einer Speicherstelle maximal einen dezimalen Wert von 255 ablegen kann. Sie sehen jetzt, warum im obigen Beispiel mit 8 Ziffern gearbeitet wurde. Jede Ziffer entspricht einem Bit. Alle acht Bits zusammengefaßt nennt man BYTE. Sind nun alle acht Bits "gesetzt", so erhält man einen dezimalen Wert von 255. Das sind insgesamt 256 mögliche Werte, nämlich 0-255. Der COMMODORE 64 besitzt aber insgesamt 65535 Speicherstellen. Wie kann er diese Stellen erreichen, wenn er in einer Speicherstelle nur den Wert 255 ablegen kann?

Nun, dieser Wert wird einfach in zwei "Hälften" aufgeteilt. Man nennt diese beiden Teile LOW-Byte und HIGH-Byte oder zu Deutsch niederwertiges Byte und höherwertiges Byte. Diese beiden Bytes werden nun in zwei Speicherstellen abgelegt. Das High-Byte errechnet sich aus der Division der Speicherstelle mit 256. Ein Beispiel soll Ihnen das verdeutlichen.

Nehmen wir an, Sie wollten die Speicherstelle 53280 in ein High- und ein Low-Byte zerlegen. Sie dividieren

53280/256 und erhalten 208 Rest 32. Somit ist der Wert des High-Bytes 208 und der des Low-Bytes 32. So speichert auch der Rechner intern Werte ab, die größer als 255 sind, und zwar zuerst das Low-Byte und dann das High-Byte. Sprechen Sie also in Ihrem Programm eine Speicherstelle an, z.B. durch den POKE-Befehl, so wird diese Speicherstelle erst durch den Rechner in ein Low- und High-Byte zerlegt. Werte, die größer als 255 sind, benötigen also zur Darstellung mindestens 2 Bytes.
Mit dieser Art der internen Darstellung bzw. Verarbeitung von Zahlen wird noch ein anderes Zahlensystem notwendig: das HEXADEZIMALSYSTEM.

1.4.3 DAS HEXADEZIMALSYSTEM

Im Hexadezimalsystem verwendet man als Basis die Zahl 16. Somit benötigt man auch 16 verschiedene "Ziffern". Um nun die Ziffern, die Werte größer als 10 darstellen sollen, unterscheiden zu können, bedient man sich der Buchstaben A-F. Damit sieht dann die dezimale Ziffernfolge

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 usw.

in hexadezimaler Schreibweise wie folgt aus:

1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 usw.

An einigen Beispielen wollen wir nun den Umgang mit diesem Zahlensystem üben. Wir wandeln zunächst hexadezimale Zahlen in dezimale Zahlen um. Zur Kennzeichnung der hexadezimalen Zahlen verwenden wir den Index 16.

$$2\,E\,0\,C_{16} = 2*16^{3} + 14*16^{2} + 0*16^{1} + 12*16^{0}$$

$$= 2*4096 + 14*256 + 0*16 + 12*1 = 11788_{10}$$

Sie sehen, auch hier wurde den Ziffern 2E0C jeweils eine ganz bestimmte Basis 16 mit Exponent zugeordnet, wie wir es schon von den vorherigen Zahlensystemen kennen. Zur Verdeutlichung noch ein weiteres Beispiel:

$$0\,A\,B\,C_{16} = 0*16^3 + 10*16^2 + 11*16^1 + 12*16^0$$

$$= 0*4096 + 10*256 + 11*16 + 12*1 = 2748_{10}$$

Ein nicht zu unterschätzender Vorteil der hexadezimalen Schreibweise liegt darin, daß man das Low- und High-Byte fast direkt ablesen kann. Betrachten wir unser voriges Beispiel OABC. Der erste Teil ist unser High-Byte. Dieses darf ja maximal den Wert 255 annehmen, welches in hexadezimaler Schreibweise FF wäre. OA ist in dezimaler Schreibweise 10. Somit hat unser High-Byte den Wert 10. Das Low-Byte lautet BC und hat den dezimalen Wert 188 (11*16+12). Schon haben wir unser Low- und High-Byte ermittelt. Sie brauchen also zur Ermittlung dieser beiden Werte nur noch mit einem maximalen Exponent 1 zur Basis 16 zu rechnen. Somit ist auch die Umwandlung von Dualzahlen kein großes Problem mehr, wenn wir den "Umweg" über die hexadezimalen Zahlen gehen. Folgende Beispiele sollen dies verdeutlichen.

BEISPIELE

$$0101\ 1011_2 = 5B_{16} = 5*16^1 + 11*16^0 = 91_{10}$$

$$1100\ 0011_2 = C3_{16} = 12*16^1 + 3*16^0 = 195_{10}$$

$$1010\ 1010_2 = AA_{16} = 10*16^1 + 10*16^0 = 170_{10}$$

Sicher haben Sie bemerkt, daß die Dualzahlen in zwei Hälften unterteilt wurden. Jede Hälfte wurde nun für sich zuerst in eine hexadezimale Zahl umgewandelt. Im ersten Fall waren in der linken Hälfte das erste und dritte Bit gesetzt. Das ergibt eine 5_{16}. In der rechten Hälfte waren das erste, zweite und vierte Bit gesetzt, was ein B_{16} ergibt. Somit erhalten wir den hexadezimalen Wert von 5B. Jede Hälfte der Dualzahl kann ja maximal den dezimalen Wert 15 bzw. den hexadezimalen Wert F einnehmen. Die zweistellige Hexadezimalzahl dürfte dann leicht in eine Dezimalzahl zu überführen sein (siehe Beispiel oben).

Anmerkung: Diese Hälften zu je vier Bits nennt man auch NIBBLE.

Anhand dieser Beispiele können Sie ablesen, wie Sie bei der Umwandlung von Zahlen in ein anderes Zahlensystem vorzugehen haben. Zum Schluß will ich Ihnen noch zeigen, wie Sie dezimale Zahlen in hexadezimale Zahlen umwandeln können. Der Weg ist vom Prinzip her genau der gleiche wie bei der Umwandlung von Dezimalzahlen in Dualzahlen. Nehmen wir an, Sie wollen die Zahl 49153 in ihr hexadezimales Äquivalent umwandeln. Sie überlegen wieder, welche höchste Potenz von 16 sich gerade noch in dieser Zahl unterbringen läßt. Das ist in unserem Fall 16^3 oder 4096. Nun wird 49153 durch 16^3 dividiert. Ergibt in unserem Beispiel 12 Rest 1. Damit sind wir fast am Ziel. Die Werte von 16^2 und 16^1 lassen sich nicht mehr unterbringen. Bleibt also nur noch 16^0, das einmal vorkommt. Zur Verdeutlichung nochmal die Schreibweise in der Zahlendarstellung:

$$49153 = 12*16^3 + 0*16^2 + 0*16^1 + 1*16^0$$

12_{10} entspricht hexadezimal C

0_{10} entspricht hexadezimal 0

0_{10} entspricht hexadezimal 0

1_{10} entspricht hexadezimal 1

Damit haben wir unsere Hexadezimalzahl, sie lautet:

$C001_{16}$

So, nun habe ich vorerst genug von mir gegeben. Es wird Zeit, daß Sie etwas zur Übung tun müssen. Lösen Sie bitte die Aufgaben auf der folgenden Seite. Sollten Sie an einer Stelle unsicher sein, so schlagen Sie noch einmal im entsprechenden Kapitel nach. Die Lösungen stehen auf

der übernächsten Seite. Seien Sie ehrlich vor sich selbst und lösen Sie die Aufgaben, ohne auf der Lösungsseite nachzuschauen. Viel Erfolg!

AUFGABEN

1. Wandeln Sie die folgenden Dualzahlen in Hexadezimalzahlen um:

   a) 01101100 b) 10010010
   c) 10111010 d) 11110000
   e) 00001100 f) 11001001

2. Wandeln Sie die folgenden Hexadezimalzahlen in Dezimalzahlen um:

   a) FOCA b) 1268
   c) 35AO d) 0255
   e) FOOO f) 0800

3. Wandeln Sie die folgenden Dualzahlen in Dezimalzahlen um:

   a) 10110111 b) 00110011
   c) 11111110 d) 00010101
   e) 01010101 f) 10101010

4. Wandeln Sie die folgenden Dezimalzahlen in Hexadezimalzahlen um:

   a) 63280 b) 24576
   b) 32769 d) 43981
   e) 65534 f) 18193

LÖSUNGEN

1. a) 6C b) 92
   c) BA d) FO
   e) C f) C9

2. a) 61642 b) 4712
   c) 13728 d) 597
   e) 61440 f) 2048

3. a) 183 b) 51
   c) 254 d) 21
   e) 85 f) 170

4. a) F730 b) 6000
   c) 8001 d) ABCD
   e) FFFE f) 4711

2. EINFÜHRUNG IN DAS PROGRAMMIEREN MIT BASIC

In diesem Kapitel soll die Verwendung der im ersten Teil des Buches vorgestellten Basic-Befehle anhand einfacher Basic-Programme gelernt werden. Das erste Programm soll genau nach den fünf aufgestellten Grundregeln erstellt werden. Danach wollen wir uns hauptsächlich mit dem dritten Punkt befassen, nämlich mit dem Umsetzen des Algorithmus´ in Basic.

2.1 DAS ERSTE BASIC-PROGRAMM

Wir nehmen an, daß Herr Müller nun anstatt des Kugelvolumens die Kugeloberfläche für 10 verschiedene Radien berechnen möchte. Da auch er inzwischen dazugelernt hat, hält er sich genau an die Anweisungen. Er definiert also zuerst das Problem bzw. macht eine Problemanalyse.

1. Definition des Problems

Sein COMMODORE 64 soll ihm zu gegebenen Radien, die in der Maßeinheit cm in den Rechner gelesen werden, die Oberfläche S einer Kugel berechnen. Die Formel dazu lautet:

$$S = 4\pi r^2$$

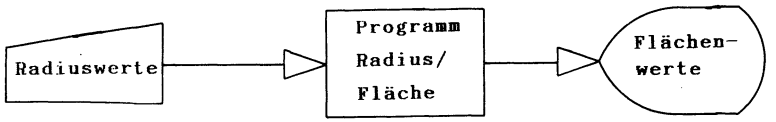
2. Entwurf des Algorithmus zur Lösung

1. Start
2. Eingabe von r
3. Berechnung von $S = 4\pi r^2$
4. Ausgabe von S auf Bildschirm
5. Ende

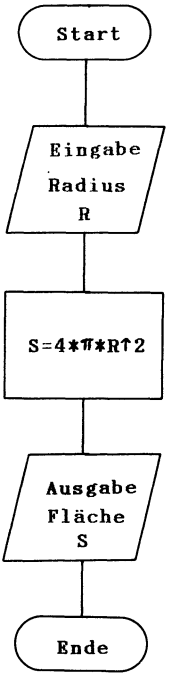
Auf der nächsten Seite sehen Sie den dazu gehörenden Datenflußplan sowie den Programmablaufplan. Dieser Programmablaufplan zählt zu den LINEAREN Programmablaufplänen, d.h. es finden keine Verzweigungen

in Form von Unterprogrammen oder Schleifen statt. Sollten Ihnen die Begriffe Unterprogramme und Schleifen noch nichts sagen, so spielt dies im Moment noch keine Rolle. Diese Begriffe werden in einem späteren Kapitel erklärt.

Datenflußplan



Programmablaufplan



3. Erstellen des Programms

```
10 INPUT"WELCHER RADIUS IN CM";R
20 LET S=4*π*R↑2
30 PRINT S
40 END
```

4. Testlauf des Programms

Eigentlich müßten wir jetzt zuerst unseren Datenflußplan und den Programmablaufplan daraufhin überprüfen, ob alles in logisch einwandfreier Form geplant wurde. Dann erst dürfte man das eigentliche Programm zu einem Testlauf mit RUN starten. Da unser Programm noch mit einem Blick überschaubar ist, können wir direkt den Befehl RUN eingeben.

5. Dokumentation

Die Dokumentation eines Programms sollte so gehalten sein, daß auch andere Programmierer sich in kürzester Zeit in das Programm einarbeiten können, um z.B. Änderungen vornehmen zu können. Bei unserem kleinen Programm genügen der Datenfluß- und der Programmablaufplan sowie die Kurzbeschreibung unter Punkt 1 (Definition des Problems).

Wie Sie gesehen haben, steht in unserem Programm jeder Befehl in einer eigenen Zeile. Das dient sehr der Übersichtlichkeit von Programmen. Vermeiden Sie es also, sogenannte MULTISTATEMENTS zu verwenden, d.h. eine ganze Zeile voll mit Basic-Befehlen zu packen. Bei größeren Programmen durchschauen Sie dann nachher Ihr eigenes "Kunstwerk" nicht mehr. Gewöhnen Sie sich es an, in ZEHNERSCHRITTEN zu programmieren, das erleichtert späteres Dazufügen von Zeilen. Sie können durchaus erst die Zeile 20 programmieren und dann die Zeile 10. Der Computer ordnet die Zeilen automatisch nach Größe der

Zeilennummern und führt sie auch in dieser Reihenfolge aus, falls nicht andere Basic-Befehle diese Folge durch Programmsprünge verändern.

Nun zu der Besprechung der in unserem Programm verwendeten Befehle INPUT, LET, PRINT und END.

2.1.1 EINGABE VON WERTEN MIT INPUT

Der Befehl INPUT wird in einem Programm zur Übergabe von Werten während des Programmablaufs benutzt. Der Benutzer kann somit direkten Einfluß auf den Programmablauf nehmen. Nach INPUT kann ein Kommentar in Anführungszeichen folgen, ähnlich wie in unserem Beispielprogramm. Dem Kommentar können eine oder mehrere Variablen folgen. Die erste Variable, die dem Kommentar folgt, wird durch ein Semikolon abgetrennt. Die Variablen untereinander müssen durch Kommata getrennt werden. Trifft das Programm auf den INPUT-Befehl, so wird der Programmablauf unterbrochen und auf dem Bildschirm erscheint der Cursor. Das Programm erwartet nun eine Eingabe über die Tastatur, die mit der RETURN-Taste abgeschlossen wird. Die folgenden Beispiele sollen das noch etwas verdeutlichen.

a) INPUT S Eingabe: 1

Hier wird der Variablen S der Wert 1 zugeordnet.

b)' INPUT"WELCHER RADIUS";S Eingabe: 3

Auf dem Bildschirm erscheint der Text "WELCHER RADIUS" und nach dem Betätigen der Tasten 3 und RETURN wird der Variablen S der Wert 3 zugeordnet.

c) INPUT A,B,C Eingabe: 4.3,.5,4

In diesem Fall werden den Variablen A,B,C nacheinander die Werte 4.3, .5 und 4 zugeordnet. Dezimalstellen werden mit dem Punkt abgetrennt. Das Komma dient immer zur Unterscheidung von mehreren Variablen.

Wichtig:
Bei INPUT werden mehrere Variablen durch KOMMATA getrennt. Zur Trennung der Dezimalstellen wird der DEZIMALPUNKT verwendet.

Der LET-Befehl weist einer Variablen einen Wert zu. Der Ausdruck, der rechts vom Gleichheitszeichen steht, wird berechnet, und die Variable auf der linken Seite des Gleichheitszeichens erhält diesen Wert. Man nennt den LET-Befehl auch "Wertzuweisung". LET wird in den meisten Fällen aber nicht benutzt, da der Rechner die Zuweisung auch so annimmt. Die folgenden Beispiele sollen dies wieder verdeutlichen.

Anmerkung: Einige Basic-Dialekte benötigen den LET-Befehl unbedingt.

a) LET A=10 oder A=10

Weist der Variablen den Wert 10 zu.

b) LET A=A+5 oder A=A+5

Zum Wert von A wird noch 5 addiert. Der neue Wert wird wieder in A gespeichert.

c) LET A=A*B-8 oder A=A*B-8

Der Wert von A wird mit dem Wert von B multipliziert. Vom Ergebnis wird 8 subtrahiert. Dieses neue Ergebnis wird wieder der Variablen A zugeordnet. Auf der rechten Seite vom Gleichheitszeichen dürfen beliebige mathematische Ausdrücke stehen. Es können also auch bis zu einem gewissen Grad mathematische Formeln Verwendung finden (siehe Beispielprogramm Oberflächenberechnung einer Kugel). Auf der linken Seite des Gleichheitszeichens darf sich immer nur EINE Variable befinden.

Wichtig:
LET kann benutzt werden, um einer Variablen einen Wert zuzuweisen. Links vom Gleichheitszeichen darf nur EINE Variable stehen. Rechts vom Gleichheitszeichen kann jeder beliebige mathematische Ausdruck stehen.

Betrachten wir uns nochmal den Ausdruck aus Beispiel b). Mathematisch ergibt dies ja keinen Sinn, denn A=A+5 ist bestimmt eine falsche Aussage. Um es zu verdeutlichen, kann man auch schreiben 4=4+5, wenn A momentan den Wert 4 hätte. Dieser Ausdruck wird aber nun in Basic nicht als Gleichung betrachtet, sondern als eine Zuweisung. Stellen Sie sich vor, man hätte mit A eine Art von Schublade beschrieben. Die Zuweisung A=A+5 bedeutet nun nichts anderes als: Nehme den Inhalt von Schublade A, lege zum Inhalt 5 dazu und lege alles in Schublade A zurück. Ist doch ganz einfach, oder?

2.1.2 AUSGABE MIT PRINT

Der PRINT-Befehl ist wohl einer der ersten Befehle, den ein Anfänger in Sachen Programmierung anwendet. Zugleich zählt er aber auch zu den vielseitigsten Befehlen des Commodore-Basic. Sie können mit ihm Texte bzw. Mitteilungen oder Werte von Variablen ausgeben lassen. Sie können beides mischen, d.h. daß die Werte der Variablen mit Text versehen ausgegeben werden können.
Weiterhin besteht die Möglichkeit, mit dem PRINT-Befehl einfache Grafiken mittels der Tastaturgrafik zu erstellen. Anhand einiger Beispiele wollen wir uns die Wirkungsweise des PRINT-Befehls verdeutlichen. Die Variablen, die verwendet werden, sollen folgende Werte haben:

```
A=10 : B=20 : C=30
```

BEISPIELE:

Befehle	Ausgabe
a) PRINT A	10
b) PRINT "A"	A
c) PRINT A*B	200
d) PRINT A,B	10 20
e) PRINT B;C	20 30
f) PRINT "B";B	B 20
g) PRINT "A HAT DEN WERT";A	A HAT DEN WERT 10

Das soll vorerst an Beispielen zur Verdeutlichung reichen. Die einzelnen Anwendungen des PRINT-Befehls werden Sie noch in den einzelnen Programmen kennenlernen. Bevor wir nun die Beispiele oben durchsprechen, will ich noch etwas zur Schreibweise in den Beispielen sagen.
Der COMMODORE 64 kennt zwei Arten der Befehlsausführung. Zum ersten kann er im sogenannten DIREKTMODUS arbeiten, was der Schreibweise im o.a. Beispiel entspricht. Geben Sie also über die Tastatur PRINT A ein und betätigen die RETURN-Taste, wird dieser Befehl sofort ausgeführt. Zum Zweiten gibt es den PROGRAMMODUS, der dadurch gekennzeichnet ist, daß vor jedem Befehl bzw. vor jeder Befehlszeile eine Zeilennummer steht. Geben Sie, z.B. über die Tastatur, die Zeichenfolge

```
10 PRINT A
```

ein, und betätigen Sie danach die Taste RETURN, so wird diese Zeile im BASIC-Speicher des Rechners abgelegt. Durch RUN und RETURN wird das Programm dann gestartet.

Nun kommen wir zu Beispiel a). Diese Schreibweise von PRINT wird benutzt, um den Wert einer Variablen ausgeben zu lassen. Es erscheint auf dem Bildschirm die 10, da wir zuvor A=10 gesetzt haben.
Bei der Ausgabe von Zahlenwerten ist darauf zu achten, daß immer vor der Zahl ein Platz für das Vorzeichen der Zahl freigehalten wird. Bei positiven Zahlen haben Sie also einen Leerplatz vor der Zahl. Dieser Leerplatz wird bei negativen Zahlen durch das Minuszeichen besetzt,

sodaß die Zahlen 10 und -10 immer die gleiche Länge bei der Ausgabe besitzen.
Da bei der Schreibweise in Beispiel a) der Variablen kein Zeichen mehr folgt, werden automatisch ein WAGENRÜCKLAUF (CARRIAGE RETURN) und ein ZEILENVORSCHUB (LINE FEED) ausgeführt. Das bedeutet, daß beim nächsten PRINT-Befehl die Ausgabe am Anfang der nächsten Zeile ausgeführt wird. Den Wagenrücklauf und den Zeilenvorschub können Sie sich am Beispiel einer Schreibmaschine verdeutlichen. Stellen Sie sich vor, daß Sie die Zahl 10 auf das Papier tippen. Danach betätigen Sie den Bügel der Schreibwalze und drücken ihn nach rechts. Dabei wird die Walze um den Zeilenvorschub vorwärts bewegt - das Papier wird also ein Stück weiter herausgedreht - und der Wagen wird bis zum Anschlag nach rechts gefahren. Somit können Sie wieder am Anfang der nächsten Zeile erneut mit dem Schreiben beginnen.
Nichts anderes wird bei einem CARRIAGE RETURN mit LINEFEED bei einem Computer ausgeführt, nur mit dem Unterschied, daß Sie keinen Bügel und keinen Wagen nach rechts bewegen müssen.

In Beispiel b) wurde das A in Anführungszeichen gesetzt. Das bewirkt, daß das Zeichen A ausgegeben wird und nicht der Wert der Variablen A. Grundsätzlich werden alle Zeichen, die in Anführungszeichen stehen, ausgegeben, es sei denn, es handelt sich um bestimmte Steuerzeichen, z.B. für die Farbgebung der Schrift.

Beispiel c) zeigt Ihnen, daß Sie mit dem PRINT-Befehl auch Berechnungen ausführen lassen können. Es wird zunächst das Produkt aus den Variablen A*B (10*20) errechnet, und dann ausgegeben. Auch hier erfolgt wieder ein Carriage-Return mit Linefeed.

Beispiel d) zeigt eine Möglichkeit auf, mehrere Variablenwerte in einer Zeile drucken zu lassen. Das Komma unterdrückt also in diesem Fall den Carriage-Return mit Linefeed. Es hat aber auf die eigentliche Ausgabeform noch eine andere Wirkung.
Der COMMODORE 64 stellt 40-Zeichen in einer Bildschirmzeile dar. Die Programmzeile kann maximal

80-Zeichen umfassen, also 2*40 Zeichen = 2 Bildschirmzeilen. Wenn Sie für die einzelnen BASIC-Befehle bei der Eingabe die Abkürzungen benutzen, werden später beim Listen solch einer Programmzeile mehr Bildschirmzeilen benutzt. Bei der Programmierung nimmt der Rechner aber nur bis zu maximal 2 Bildschirmzeilen an. Aber denken Sie daran: Vermeiden Sie die Verwendung von MULTISTATEMENTS (Vollstopfen der Programmzeile mit BASIC-Befehlen).
Diese 2 Bildschirmzeilen sind insgesamt achtmal zu je 10 Stellen unterteilt. Es ist somit eine Art von Tabulator. Wird nun das Komma zwischen zwei Variablen verwendet, so wird die zweite Variable an den Anfang des zweiten Tabulators gesetzt, also ab der 10. Stelle in der Bildschirmzeile. Werden mehrere Variablen durch das Komma getrennt, so werden sie an den entsprechenden Stellen ausgegeben.

Geben Sie nun folgendes in den Rechner ein:

```
PRINT "1","2","3","4","5","6","7","8"
```

Betätigen Sie jetzt die RETURN-Taste, so sehen Sie auf dem Bildschirm genau die Positionen der einzelnen Tabulatoren. Hätten wir die Zahlen nicht in Anführungszeichen gesetzt, so wären sie durch das Vorzeichen genau um eine Stelle nach rechts verschoben ausgegeben worden.

In Beispiel e) wird Ihnen die Wirkung des Semikolons gezeigt. Dadurch werden nicht nur Wagenrücklauf und Zeilenvorschub unterdrückt, sondern auch die Funktion des Tabulators. Die Zeichen werden also hintereinander in der Reihenfolge ausgegeben, wie sie auch im PRINT-Befehl geschrieben wurden. Dadurch wird es auch ermöglicht, direkt hinter der Wertausgabe einer Variablen einen Text mit anzugeben.

Beispiel f) zeigt die Möglichkeit, die Bezeichnung der Variablen und direkt anschließend den Wert der Variablen auszugeben. Dieses wurde ebenfalls durch das Semikolon erreicht.

In Beispiel g) wurde von der Möglichkeit Gebrauch gemacht, einen näher erläuternden Text mit dem Wert einer Variablen auszugeben. Somit hat man die Gelegenheit, die Ausgabe von Ergebnissen näher zu beschreiben und dem Anwender mitzuteilen, um welchen Wert es sich hierbei handelt.

Der END-Befehl in der letzten Programmzeile kennzeichnet das logische Ende des Programms. Dieser Befehl steht in den meisten Fällen am Programmende. Er kann jedoch auch irgendwo mitten im Programm untergebracht sein, wenn z.B. nach diesem Befehl die Unterprogrammroutinen folgen. Dazu erfahren Sie jedoch später mehr.
Haben Sie nun ein Programm geschrieben und setzen in der letzten Programmzeile nicht den END-Befehl, so ist das nicht weiter tragisch, da der Rechner im Speicher auch automatisch das Programmende kennzeichnet. Trotzdem gewöhnen Sie sich es bitte an, den END-Befehl zu benutzen, da er nun mal zu einem guten Programmierstil dazu gehört.

Nun haben wir die Befehle, die in unserem Beispielprogramm vorkamen, besprochen. Eigentlich sollten bei der Benutzung dieser Befehle keine großen Schwierigkeiten mehr auftreten. Um Programme nun auch für andere besser verständlich zu machen, schauen wir uns noch die REM-Anweisung an.

2.1.3 KOMMENTARE MIT REM

Mit REM können Sie an beliebiger Stelle im Programm eine Bemerkung unterbringen. Alles, was der Anweisung REM folgt, wird vom Rechner ignoriert, auch andere BASIC-Befehle.

Wir wollen jetzt unser Beispielprogramm weiter ausbauen und auch für andere, die es nicht geschrieben haben, verständlicher machen. Das zählt übrigens auch zur Dokumentation von Programmen. Auf der nächsten Seite folgt nun das abgeänderte Programmlisting mit anschließender Beschreibung der einzelnen Programmzeilen.

```
10 REM BERECHNUNG DER KUGELOBERFLAECHE
20 REM EINGABE RADIUS IN CM
30 INPUT"WELCHER RADIUS (IN CM)";R
40 REM BERECHNUNG OBERFLAECHE
50 LET S=4*π*R↑2
60 REM AUSGABE OBERFLAECHE IN CM↑2
70 PRINT"DIE KUGELOBERFLAECHE BETRAEGT ";S;" CM↑2"
80 END
```

Die Zeilen 10-20 dienen dazu, dem Benutzer zu sagen, was das Programm macht und wie die Eingabe zu erfolgen hat. In Zeile 30 erfolgt die Eingabe der Daten mittels INPUT, wobei hier nochmal für den Anwender ein erläuternder Kommentar mit ausgegeben wird. Dies hätte man auch dadurch erreichen können, daß man den Text in einer separaten Programmzeile mit dem PRINT-Befehl ausgegeben und den INPUT-Befehl ebenfalls alleine in eine Programmzeile geschrieben hätte. Zeile 40 weist mit dem Kommentar auf die folgende Berechnug in Zeile 50 hin. In Zeile 50 wird der Variablen S durch Berechnung des linken mathematischen Ausdrucks der Wert der Oberfläche zugeordnet. Zeile 60 verweist mit dem Kommentar auf die Ausgabe in cm^2 in Zeile 70. In Zeile 70 wird durch den PRINT-Befehl die Kombination von Text- und Wertausgabe der Variablen veranlaßt. Zeile 80 beendet schließlich das Programm mit dem END-Befehl.

2.2 VARIABLEN UND DEREN BENUTZUNG IN PROGRAMMEN

Bevor ich Ihnen nun einige Aufgaben zum Lösen gebe, müssen wir noch die verschiedenen Typen der Variablen besprechen.

Im COMMODORE-BASIC gibt es DREI verschiedene Typen von Variablen, die auf verschiedene Art gekennzeichnet werden. Der erste Variablentyp ist die INTEGER-VARIABLE bzw. GANZZAHLVARIABLE. Dieser Variablentyp kann also nur

ganze Zahlen repräsentieren. Die Kennzeichnung erfolgt durch das "%" Prozentzeichen, welches einfach an den Namen der Variablen angehängt wird (z.B. A% oder C4%). Wird diesem Variablentyp eine Zahl mit Nachkommastelle zugeordnet, so wird nur die Zahl vor dem Dezimalpunkt berücksichtigt. Die Nachkommastellen gehen dabei verloren. Eine Besonderheit ist bei diesem Variablentyp allerdings zu berücksichtigen: Diesem Variablentyp dürfen nur Werte zwischen -32768 und 32767 zugeordnet werden.

Der zweite Variablentyp wird benutzt, um Dezimalzahlen (auch Fließkommazahlen genannt) darstellen zu können. Die dabei verwendeten Variablenbezeichnungen erhalten keinen Zusatz, um diesen Typ zu kennzeichnen. Erlaubte Bezeichnungen sind z.B. A oder B2.

Der dritte Variablentyp wird zusätzlich durch das "$" Dollarzeichen gekennzeichnet. Es handelt sich hierbei um die sogenannte STRINGVARIABLE. In dieser Variablen können beliebige Zeichenfolgen abgespeichert und bei Bedarf ausgegeben werden. Allerdings dürfen nicht mehr als 255 Zeichen in der Stringvariablen verwendet werden.

Bei der Verwendung der Bezeichnungen von Variablen muß allerdings einiges berücksichtigt werden. Der Rechner erkennt eine Variable nur an den ersten beiden Zeichen. Sie dürfen zwar längere Variablennamen benutzen (z.B. WERT), aber der Computer kann diese Bezeichnung nicht von der Bezeichnung WECHSEL unterscheiden, da die ersten beiden Zeichen (WE) bei beiden Variablen gleich sind. Sie dürfen also durchaus Ihren Variablen die Namen OBERFLAECHE und FLAECHE geben. Es dürfen sich in diesen Namen aber wiederum keine BASIC-Befehle verstecken. Die Bezeichnung WAND wäre unzulässig, da sich in ihr der logische Operator AND befindet. Weiterhin dürfen zusätzlich Ziffern zur Kennzeichnung benutzt werden, allerdings mit der Einschränkung, daß diese erst an zweiter Stelle auftreten dürfen. Erlaubt sind Bezeichnungen wie A1, B9 oder ähnliche. Es ist nicht

erlaubt, die Zahl an die erste Stelle zu setzen. Z.B. ist 9B VERBOTEN!

Sie müssen außerdem darauf achten, daß Sie keine Befehle zur Bezeichnung heranziehen, die aus zwei Buchstaben bestehen, wie:

OR, FN oder IF.

Der COMMODORE 64 benutzt außerdem folgende Variablenbezeichnungen für interne Funktionen:

TI, ST

Diese dürfen Sie ebenfalls nicht in Ihren Programmen verwenden. Soweit die Einschränkungen bei den Bezeichnungen für die Variablen.

2.2.1 RECHENOPERATIONEN MIT VARIABLEN

Wollen Sie nun in Ihren Programmen mit den Variablen Berechnungen durchführen, so müssen Sie vorher die Gesetzmäßigkeiten der einzelnen Rechenoperationen kennenlernen. Sie werden sicherlich noch die alte Regel "Punktrechnung vor Strichrechnung" im Gedächtnis haben. Damit sind Sie schon einen guten Schritt weiter. Die nachfolgende Aufstellung gibt Ihnen genaueren Aufschluß.

Zeichen	Rangfolge	Bedeutung
↑	ERSTE	POTENZIEREN
*	ZWEITE	MULTIPLIKATION
/		DIVISION
+	DRITTE	ADDITION
-		SUBTRAKTION

Auch für die logischen Operatoren existiert eine solche Rangfolge. Dabei hat NOT die höchste, AND die zweithöchste und OR die niedrigste Priorität nach den mathematischen Rangfolgen.

Nun sind Sie mit Ihrem bisher erworbenen Wissen soweit, um die nachfolgenden Aufgaben zu lösen. Sie beinhalten sowohl Fragen zu bestimmten Kapiteln als auch kleinere Programmieraufgaben, die Sie bitte selbständig lösen wollen. Versuchen Sie zuerst die Aufgaben zu lösen, ohne in den entsprechenden Kapiteln nachzuschlagen. Es schadet überhaupt nichts, wenn Ihnen dabei Fehler unterlaufen, denn aus gemachten Fehlern lernt man bekanntlich am besten. Benotet werden Sie hier auch nicht. Sie können also ganz unbefangen an die Sache herangehen und dann selbst zu einem Ergebnis kommen. Sind Sie irgendwo unsicher, können Sie das entsprechende Kapitel ja nochmal durcharbeiten. Übrigens beherzigen Sie bei den Programmieraufgaben die angesprochenen 5 Stufen, aus denen sich die Programmierung eines Programms zusammensetzen sollte. Geben Sie vor jedem neuen Programm, das Sie eingeben wollen, den Befehl NEW ein, damit der BASIC-Speicher und somit das alte Programm gelöscht werden. Und nun viel Erfolg beim Lösen der Aufgaben.

AUFGABEN

1. Untersuchen Sie die folgenden Variablenbezeichnungen auf Zulässigkeit und begründen Sie Ihre Entscheidung.

a) X1	b) ERDE$	c) STAND
d) ODER%	e) GRIFF	f) NORD$
g) 4Z%	h) 25	i) FO

2. Schreiben Sie ein Programm, das vier Werte A,B,C und D einliest und die Werte A und B in einer Zeile hintereinander und die Werte C und D in der nächsten Zeile tabelliert ausgibt.

3. Schreiben Sie ein Programm, das Ihnen die Fläche eines rechtwinkligen Dreiecks in Quadratmetern berechnet und die Ausgabe mit einem entsprechenden Text versieht.

4. Schreiben Sie ein Programm, das das Idealgewicht (Körpergröße in cm minus 100 minus 10 Prozent) eines Menschen berechnet. Es soll die Eingabe der Körpergröße in cm verlangt werden und die Ausgabe des Körpergewichts in Kilogramm erscheinen.

5. Schreiben Sie ein Programm, das Ihnen die Anzahl der Liter in einem Aquarium berechnet, nachdem das Programm dazu aufgefordert hat, die Daten für Länge, Höhe und Breite in cm einzugeben.

6. Ändern Sie Aufgabe 2 so ab, daß das Programm jeden Wert mit Namen jeweils in eine neue Zeile schreibt.

1. a) zulässig
   b) zulässig
   c) STAND unzulässig. Zweimal falsch, da ST reservierte Variable und AND logischer Operator.
   d) zulässig
   e) GRIFF unzulässig, da Befehl IF enthalten.
   f) NORD$ unzulässig, da OR logischer Operator enthalten.
   g) 4Z% unzulässig, da Bezeichnung mit einer Ziffer beginnt.
   h) 25 unzulässig, siehe g).
   i) zulässig

2.
```
10 INPUT A,B,C,D
20 PRINT A;B
30 PRINT C,D
40 END
```

Ich hoffe, Sie haben hier auf die Anwendung von KOMMA und SEMIKOLON bei der Ausgabe der Daten geachtet. Zur Anwendung von INPUT wäre noch folgendes zu bemerken: Erwartet INPUT mehrere Daten und es wird nur ein Wert mit RETURN eingegeben, so erscheinen auf dem Bildschirm 2 Fragezeichen. Damit zeigt der Rechner an, daß er weitere Daten erwartet.

3.
```
10 REM EINGABE VON HOEHE UND
20 REM HYPOTENUSE C IN METERN
30 INPUT"EINGABE H,C";H,C
40 A=.5*H*C
50 PRINT"DIE FLAECHE HAT ";A;"M↑2"
60 END
```

```
4. 10 INPUT"EINGABE KOERPERGROESSE IN CM";CM
   20 REM BERECHNUNG IDEALGEWICHT
   30 IG=CM-100
   40 REM BERECHNUNG 10 PROZENT
   50 PR=IG/100*10
   60 IG=IG-PR
   70 PRINT"IHR IDEALGEWICHT IST";IG;"KG"
   80 END
```

Diese Aufgabe hätte man auch kürzer lösen können. Die Zeilen 30, 50 und 60 könnte man in einer einzigen Zeile zusammenfassen, wie folgendes Beispiel zeigt:

```
30 IG=(CM-100)-(CM-100)/100*10
```

Diese Zeile ist zunächst jedoch unübersichtlicher, da man nicht auf Anhieb erkennt, welche Berechnung dort durchgeführt wird. Sicher werden jetzt einige Leser bemerken wollen, daß diese Art der Programmierung Speicherplatz sparen hilft. Sie haben natürlich Recht, aber bekanntlich führen viele Wege nach Rom. Das soll heißen, daß man sich zu einem Kompromiß zwischen Übersichtlichkeit und Kürze des Programms entscheiden muß. Solange man nicht auf den Speicherplatz achten muß, sollte man sein Programm so schreiben, daß es auch andere ohne große Schwierigkeiten verstehen können. Das dient natürlich auch dazu, um sich selber zu einem späteren Zeitpunkt in seinem eigenen Programm zurechtfinden zu können.

```
5. 10 INPUT"HOEHE, LAENGE, TIEFE IN CM";H,L,T
   20 REM BERECHNUNG VOLUMEN
   30 V=H*L*T
   40 REM BERECHNUNG LITER
   50 V=V/1000
   60 PRINT"AQUARIUM INHALT";V;"LITER"
   70 END
```

In diesem Programm werden zunächst den Variablen für die Höhe, Länge und Tiefe, H, L und T, die Werte in cm

übergeben. Sie sehen, daß man den Variablen durchaus sinnvolle Bezeichnungen zukommen lassen kann. Dann wird in Zeile 30 das Volumen in ccm errechnet. In Zeile 50 wird das ausgerechnete Volumen noch durch 1000 dividiert und schon haben wir unseren Inhalt des Aquariums in Litern.

6.
```
10 INPUT A,B,C,D
20 PRINT"A";A
30 PRINT"B";B
40 PRINT"C";C
50 PRINT"D";D
60 END
```

Ich hoffe, daß Sie diese Aufgaben mit Bravour gemeistert haben. Sollten dennoch Schwierigkeiten bei einzelnen Aufgaben aufgetreten sein, so lesen Sie die entsprechenden Passagen noch einmal genau durch.
Im nächsten Kapitel wollen wir uns einige neue Befehle und ihre Verwendungen im Programm aneignen.

2.3 NUMERISCHE FUNKTIONEN

Bisher haben wir uns mit einfachen Wertzuweisungen von Variablen befaßt, d.h. es wurden nicht die vorhandenen mathematischen Funktionen des COMMODORE 64 benutzt, sondern es wurden nur die vier Grundrechenarten zur Berechnung herangezogen. In diesem Kapitel wollen wir uns nun mit den vorgegebenen Funktionen wie COS(X) oder SIN(X) befassen. Dazu wollen wir einen kleinen Abstecher in die Mathematik machen. Bekommen Sie jetzt keinen Schreck, denn Sie werden nicht mit Formeln überhäuft oder mit langatmigen mathematischen Beweisverfahren konfrontiert werden. Dies soll ein BASIC-Trainingsbuch bleiben und kein mathematisches Lexikon werden.
In vielen BASIC-Büchern und auch im Handbuch des COMMODORE 64 findet man immer die Angabe, daß die Werte der trigonometrischen Funktionen wie SIN(X), COS(X) oder TAN(X) im BOGENMAß anzugeben sind. Was ist nun dieses Bogenmaß?

Nun, es handelt sich hierbei auch um eine Winkelangabe, zu der der SINUS oder COSINUS berechnet werden soll. Die normale Aufteilung eines Kreises in 360 Grad dürfte jedem bekannt sein. 1 Grad ist also der 360igste Teil eines Kreises. 90 Grad stehen demnach für den Viertelkreis, 180 für den Halbkreis usw.

Beim Bogenmaß hat man nun nicht den Kreis in 360 Teile aufgeteilt, sondern hat den KREISUMFANG für die Berechnung zugrunde gelegt. Der Kreisumfang errechnet sich nach der Formel:

U=2*PI*R

Nun hat man zur Vereinfachung der Berechnung den Einheitskreis (Radius=1) herangezogen. Somit ergibt sich für den Kreisumfang dann

U=2*PI*1 oder U=2*PI

Die Bezeichnung des Bogenmaßes erfolgt nun nicht in Grad, sondern in "RAD" (RADIANT). Das würde für unser Beispiel bedeuten, daß der Kreis 360 Grad oder 2*PI (6.2831..) Rad besitzt. 90 Grad würden also im Bogenmaß 2*PI/4 oder PI/2 Rad entsprechen. Der Vorteil des Bogenmaßes liegt darin, daß man aus dem Wert des Bogenmaßes bei einem Radius von 1 direkt die Länge des Kreisbogens ermitteln kann. Die eigentliche Berechnung mit dem Bogenmaß ist am Anfang etwas gewöhnungsbedürftig, da man sich unter 90 Grad eher den Viertelkreis vorstellen kann als unter PI/2 Rad.

Dieser kleine Exkurs in die Mathematik soll zunächst genügen. Sie können sich nun unter dem Begriff des Bogenmaßes etwas vorstellen, sodaß wir nun ein paar Beispielprogramme schreiben und anwenden wollen, damit es noch deutlicher wird.

Geben Sie nun das folgende Beispielprogramm in den Rechner ein.

```
10 INPUT"EINGABE IN GRAD";GR
20 REM BERECHNUNG DES SINUS
30 SI=SIN(GR*π/180)
40 PRINT"DER SINUS VON";GR;"GRAD ";
50 PRINT"IST =";SI
60 END
```

Starten Sie es nun mit RUN und geben Sie für den Winkel den Wert 90 ein. Als Ergebnis sollten Sie 1 erhalten. Dieses Programm erwartet die Eingabe des Winkels in Grad und berechnet den dazugehörigen Sinus. Wollen Sie in Ihren eigenen Programmen also die Winkel in Grad eingeben, so müssen Sie die Umrechnung in Zeile 30 benutzen. Für die Berechnung des Cosinus´ müßten Sie in Zeile 30 nur SIN durch COS ersetzen. Die Variable SI können Sie beibehalten. Wundern Sie sich nicht, wenn Sie das Programm in der COS-Version starten und bei Eingabe von 90 den Wert 7.3145904E-10 erhalten. Dies ist der internen Rechenungenauigkeit des COMMODORE 64 zuzuschreiben. Diese Zahl ist allerdings so klein, daß man Sie getrost als Null werten kann. Um den Unterschied

zum Bogenmaß zu verdeutlichen, geben Sie das Programm in der folgenden Version ein. Zuvor jedoch geben Sie noch NEW ein und betätigen die RETURN-Taste.

```
10 INPUT"EINGABE IM BOGENMAß";BM
20 REM BERECHNUNG DES SINUS
30 SI=SIN(BM)
40 PRINT"DER SINUS VON";BM;"RAD ";
50 PRINT"IST =";SI
60 END
```

Das einzige, was sich gegenüber unserem vorigen Programm geändert hat, ist die Zeile 30. Starten Sie nun das Programm und geben Sie den Wert 1.57079633 (entspricht $\pi/2$) ein. Als Ergebnis erhalten Sie wieder einen Wert, der fast Null ist. Normalerweise ist der Sinus von $\pi/2$ (Phi Halbe) genau Null. Diese geringe Abweichung ist wieder der internen Rechenungenauigkeit des CBM 64 zuzuschreiben.

Die Anwendung der anderen Funktionen ist denkbar einfach. Wie auf den vorderen Seiten bei den Kurzbeschreibungen der BASIC-Befehle erläutert, wird diesen numerischen Funktionen immer nur ein Wert übergeben, welcher dann zur Berechnung herangezogen wird. So berechnet SQR(X) die Quadratwurzel von X, oder ATN(X) den Arcustangens von X. Die Funktionen EXP(X) und LOG(X) berechnen die X-te Potenz von e=2.71827183 bzw. den Logarithmus zur Basis von e. Die eine Funktion stellt also zur anderen die Umkehrfunktion dar. Geben Sie folgende Befehlsfolge im Direktmodus in den Rechner ein und drücken Sie RETURN:

```
PRINT EXP(1)
```

Sie erhalten als Ergebnis die Zahl 2.71828183, auch als Eulersche Zahl bekannt. Wiederholen Sie den gleichen Vorgang mit der folgenden Befehlsfolge:

```
PRINT LOG(2.71828183)
```

Nun erhalten Sie wiederum die 1. Wollen Sie den Logarithmus zur Basis 10 berechnen, so müssen Sie nur

LOG(X) durch LOG(10) dividieren. Der Logarithmus zur Basis 10 nennt sich auch "dekadischer Logarithmus" und der Logarithmus zur Basis e "natürlicher Logarithmus". Das folgende Beispielprogramm errechnet Ihnen sowohl den natürlichen als auch den dekadischen Logarithmus.

```
10 INPUT"EINGABE DER ZAHL";Z
20 REM BERECHNUNG NAT. LOGARITHMUS
30 LN=LOG(Z)
40 REM BERECHNUNG DEKA. LOGARITHMUS
50 LO=LOG(Z)/LOG(10)
60 PRINT"DER NATUERLICHE LOGARITHMUS ";
70 PRINT"VON";Z;" BETRAEGT";LN
80 PRINT
90 PRINT"DER DEKADISCHE LOGARITHMUS ";
100 PRINT"VON";Z;" BETRAEGT";LO
110 END
```

Sie sehen, es ist also relativ einfach, diese Funktionen in Programmen zu handhaben. Die einzige Schwierigkeit besteht eben darin, wie man die Werte umgerechnet bekommt.

WICHTIG:
Die trigonometrischen Funktionen erwarten die Werte im Bogenmaß. Für die Berechnung in Grad müssen diese entsprechend umgerechnet werden. Die Funktionen LOG und EXP beziehen sich auf den Exponenten bzw. die Basis "e".

Die Funktion SGN(X) ergibt das Vorzeichen von X. Das Ergebnis ist 1, wenn X positiv ist, 0, wenn X=0 und -1, wenn X negativ ist. Für X kann jede beliebige Zahl eingesetzt werden. Die gleiche einfache Anwendung finden wir bei der Funktion INT(X). Diese Funktion ist sehr nützlich, wenn es um das Runden von Zahlen geht. Mit einer entsprechenden Routine kann man auf beliebig viele

Stellen nach dem Komma runden. Das nachfolgende kleine Programm soll Ihnen das verdeutlichen.

```
10 INPUT"WIEVIELE STELLEN NACH DEM KOMMA";X%
20 INPUT"WELCHE ZAHL";Z
30 REM RUNDEN
40 Z=INT (Z * 10↑X% + .5) / 10↑X%
50 REM AUSGABE GERUNDETE ZAHL
60 PRINT Z
70 END
```

Das Programm ist recht einfach, trotzdem will ich Ihnen die wichtigsten Zeilen erklären. In Zeile 10 wird zunächst nach der Stellenzahl gefragt, auf die nach dem Dezimalpunkt gerundet werden soll. Dieser Wert wird der Variablen X% zugeordnet. Es handelt sich hierbei um eine Integervariable, da ja nur ganze Zahlen als Eingabe einen Sinn ergeben. In Zeile 20 wird nach einer beliebigen Zahl gefragt. Geben Sie hier irgendeine Dezimalzahl ein, deren Stellenzahl größer als die zu rundende Stellenzahl ist. In Zeile 30 wird schließlich die eigentliche Rundung vorgenommen. Z wird zuerst mit 10 hoch X% multipliziert. Damit werden je nach Eingabe von X% die Nachkommastellen, die gerundet werden sollen, zunächst vor den Dezimalpunkt geholt. Dann werden .5 addiert, um eine Rundung der nachfolgenden Kommastellen zu gewährleisten, da INT ja sämtliche Nachkommastellen "abtrennt" ohne Rücksicht auf den Wert der einzelnen Zahl. Von diesem Ausdruck wird nun der ganzzahlige Wert gebildet. Anschließend wird wieder durch 10 hoch X% dividiert und man erhält wieder eine Dezimalzahl, die jetzt aber nur die Stellen hinter dem Dezimalpunkt aufweist, die vorher durch die Multiplikation mit 10 hoch X% vor den Dezimalpunkt gesetzt wurden.

Starten Sie nun das Programm mit RUN und betätigen Sie die RETURN-Taste. Geben Sie dann einige Werte ein, um zu sehen, welche Ergebnisse Sie erhalten. Versuchen Sie vor allem die Zeile 40 zu verstehen, in der die Rundung der Zahl vorgenommen wurde, damit Sie später in eigenen Programmen selbst solche Routinen anwenden können.

2.3.1 FUNKTIONEN

Die DEF FN - Funktion ist eine praktische Möglichkeit, Speicherplatz einzusparen. Mit ihr können komplexere mathematische Funktionen dem Ausdruck FN zugeordnet werden. Dieser Ausdruck wird bei Bedarf aufgerufen und gleichzeitig wird ein Parameter mit übergeben, welcher dann in Abhängigkeit zur definierten Funktion berechnet wird. Das folgende Beispiel soll das wieder verdeutlichen.

```
10 REM DEFINITION FUNKTION
20 DEF FN F(X)=X↑2 + 2*X+ 4
30 REM EINGABE PARAMETER
40 INPUT"WELCHER WERT";X
50 REM AUSGABE
60 PRINT FN F(X)
70 END
```

In Zeile 20 wird zunächst die mathematische Funktion X↑2+2X+4 dem Ausdruck FN F(X) zugeordnet. Dabei bestimmen die Werte von X in FN F(X) das Ergebnis der Funktion. Werden innerhalb der Funktion noch andere Variablen benutzt, so werden diese durch X nicht beeinflußt, sondern behalten ihren augenblicklichen Wert bei. Dieser geht dann mit in die Berechnung ein. Sie haben also mit dieser Funktion die Möglichkeit, auf einfache Art und Weise mathematische Wertetabellen zu erstellen. Außerdem brauchen Sie innerhalb eines Programms nicht immer den kompletten mathematischen Ausdruck aufzurufen, sondern nur den Namen der Funktion mit dem entsprechenden Parameter.

2.3.2 ZUFALLSZAHLEN

Das Basic des COMMODORE 64 besitzt einen eingebauten Zufallszahlengenerator, der über den Befehl RND(X) aufgerufen werden kann. Diese Funktion wird benötigt, um

z.B. irgendeine Art von Simulation, in der der Zufall eine Rolle spielt, darzustellen. Man findet die Funktion auch sehr oft bei Spielen, um zufällige Ereignisse im Programm einzuleiten. Die Benutzung dieser Funktion ist denkbar einfach. Die Zuordnung A=RND(1) ergibt für A einen Wert zwischen 0.0 und 1.0. Bei negativen Werten von X wird immer die gleiche Folge von Zufallszahlen ausgegeben. Das folgende kleine Programm simuliert einen Würfel. Bei jedem Start des Programms wird zufällig eine Zahl zwischen 1 und 6 ausgegeben.

```
10 REM ERZEUGUNG ZUFALLSZAHL
20 A=INT (6 * RND(1))+1
30 PRINT A
40 END
```

Starten Sie nun das Programm zunächst mit RUN und RETURN. Führen Sie mehrere Programmstarts hintereinander aus und beobachten Sie die ausgegebenen Zahlen. Sie werden in der Reihenfolge der Ausgabe keine Regelmäßigkeit erkennen können.

Damit Zahlen zwischen 1 und 6 ausgegeben werden, wurde in der Zeile 20 eine Kombination aus RND und INT gewählt, da wir ja ganze Zahlen benötigen. Die 1 wurde noch addiert, damit keine Null vorkommen kann.
Mit dieser Art der Zufallszahlengenerierung können Sie in jedem beliebigem Intervall Zufallszahlen erzeugen lassen. Dabei steht die 6 für die obere Grenze des Intervalls und die +1 für die untere Grenze des Intervalls. Wollen Sie nun Zufallszahlen im Bereich von 100 bis 150 erzeugen, so müßte die Zeile 20 so aussehen:

```
20 A=INT (50 * RND(1))+100
```

oder in der allgemeinen Schreibweise, wobei O die obere Grenze und U die untere Grenze darstellt:

```
A=INT((O-U)*RND(1))+U
```

2.3.3 ASC(X$) UND CHR$(X)

Der COMMODORE 64 besitzt die Möglichkeit, durch den PRINT-Befehl Zahlen, Buchstaben und Grafikzeichen, wenn diese zwischen den Anführungszeichen stehen, auf dem Bildschirm ausgeben zu lassen. Weiterhin existieren unter diesen Zeichen bestimmte Steuerzeichen, die z.B. die Farbe der nachfolgenden Zeichen beeinflussen oder eine Umschaltung der Darstellung in reverser Schrift ermöglichen. Die Steuerzeichen sind meistens durch irgendwelche Grafikzeichen definiert, teilweise sogar in reverser Darstellung. Sie haben den Vorteil, daß sie sofort durch das Drücken der entsprechenden Taste beim Programmieren verfügbar sind. Sie haben jedoch auch einen großen Nachteil: Diese Zeichen sind sich teilweise sehr ähnlich, sodaß eine genaue Bestimmung der Zeichen in einem Programmlisting nicht möglich ist, was u.a. auch am verwendeten Drucker liegen kann. Für denjenigen, der das Programm geschrieben hat, ist das nicht weiter schlimm. Er kennt ja die Steuerzeichen, die er verwendet hat. Allerdings dürfte jemand, der das Programm nicht kennt und nur "abtippen" will, Schwierigkeiten bekommen. Erstens gibt es teilweise Schwierigkeiten bei der Identifizierung der Steuerzeichen und zweitens sind manche Steuerzeichen nur aufrufbar, wenn mehrere Bedingungen gleichzeitig erfüllt sind. (Anführungszeichen gesetzt; SHIFT und CTRL gleichzeitig gedrückt)

Muß man bei diesen vielen Möglichkeiten nach einem bestimmten Zeichen suchen, so ähnelt das manchmal eher einem Detektivspiel als einer Programmierung in BASIC. Dabei kann man sich und anderen in den meisten Fällen die Arbeit erheblich vereinfachen, indem man die CHR$-Codes verwendet. Es gibt z.B. den Befehl zum Löschen des Bildschirms:

```
PRINT "♥"
```

Erreicht wird dieses reverse Herz nur im Anführungszeichenmodus und bei gleichzeitigem Betätigen der Tasten SHIFT/CLR HOME. Dieses Zeichen wird nun relativ häufig benutzt, sodaß es den meisten Lesern schon bekannt sein dürfte. Aber selbst in diesem Fall kann man auch das gleiche mit einem anderen Befehl erzielen, indem man einen CHR$-Code verwendet:

PRINT CHR$(147)

Tritt diese Befehlsfolge in einem Listing auf, so dürfte es kaum Unklarheiten in Bezug auf das Eingeben dieser Befehle in den Rechner geben.
Ein anderes Beispiel sind die grafischen Steuerzeichen für den Cursor oder für die Funktionstasten. Diese Zeichen sehen sich teilweise so ähnlich, daß, wenn sie in einem Listing gleichzeitig auftauchen, kaum auseinandergehalten werden können. Da ist es weitaus einfacher und leichter, für die Funktionstasten oder den Cursor die ASCII-Werte zu benutzen.

In der Tabelle im Anhang dieses Buches oder im Handbuch zum COMMODORE 64 können Sie die entsprechenden ASCII-Werte nachschlagen. Wollen Sie z.B. den ASCII-Wert von dem Buchstaben A wissen, so geben Sie folgendes im Direktmodus in den Rechner:

PRINT ASC("A")

Betätigen Sie jetzt die RETURN-Taste, so erscheint auf dem Bildschirm der Wert 65.
CHR$ stellt nun die Umkehrung zum oberen Befehl dar. Geben Sie folgendes wieder im Direktmodus ein:

PRINT CHR$(65)

Betätigen Sie wiederum die RETURN-Taste, so erscheint auf dem Bildschirm jetzt das Zeichen A. Die Benutzung dieser Befehle ist meines Erachtens problemloser und eindeutiger als die Verwendung von Grafikzeichen. Deshalb wollen wir nach Möglichkeit auch in unseren folgenden Programmen

diese Schreibweise mit den Befehlen ASC("X") bzw. CHR$(X) beibehalten, zumindest bei den speziellen Codes wie Farbwechsel der Schrift, Umschaltung auf Kleinbuchstaben oder ähnlichen Funktionen. Die Umsetzung der Programme auf andere Rechner wird durch die Benutzung der CHR$-Codes ebenfalls erleichtert, da meistens die Bedeutung der ASCII-Werte übereinstimmt, z.B. CHR$(147) für das Löschen des Bildschirms. Der COMMODORE 64 benutzt, wie bereits erwähnt, das reverse Herz zwischen den Anführungszeichen zum Löschen des Bildschirms. Dieses Zeichen wird von anderen Rechnern schon nicht mehr für diese Funktion benutzt. Man findet dort zum Teil ein reverses "S" oder andere Zeichen. Dagegen können Sie den Befehl PRINT CHR$(147) direkt übertragen, ohne großartig nach einer bestimmten Tastenfunktion suchen zu müssen.

Damit Sie nun Ihr neuerworbenes Wissen anwenden können, will ich Ihnen zur Übung ein paar Aufgaben stellen. In diesen Aufgaben werden Sie die Befehle verwenden müssen, die auf den vorhergehenden Seiten besprochen wurden. Berücksichtigen Sie auch hier wieder die fünf Grundregeln des Programmierens. Viel Erfolg beim Lösen der Aufgaben auf der nächsten Seite.

1. Schreiben Sie ein Programm, das das Würfeln mit zwei Würfeln simuliert. Die Ergebnisse sollen getrennt in einer Zeile tabelliert ausgegeben werden. Beim Start des Programms soll der Bildschirm vorher gelöscht werden.

2. Schreiben Sie ein Programm, das Ihnen die Fläche eines beliebigen Dreiecks nach der HERONschen Formel F=SQR(S(S-A)(S-B)(S-C)) berechnet, wobei S=1/2(A+B+C) ist. Achten Sie darauf, daß Sie die Formel nicht so in Ihr Programm übernehmen können. Das Programm soll die Eingabe der Werte A,B,C verlangen. Das Ergebnis soll ebenfalls mit einem entsprechenden Text versehen werden.

3. Schreiben Sie ein Programm, welches die Eingabe eines Zeichens verlangt und anschließend den ASCII-Wert dieses Zeichens mit dem eingegebenen Zeichen in einer Zeile ausgibt.

4. Schreiben Sie ein Programm, welches die Höhe aus der Zeit des Fallens eines Körpers berechnet. Es soll die Eingabe der gemessenen Fallzeit verlangt werden. Die bremsende Wirkung des Luftwiderstandes wird nicht berücksichtigt. Die Formel lautet $S=1/2gt^2$. Der Wert der Konstanten G beträgt 9.81. Das Ergebnis soll in Metern ausgegeben werden.

5. Schreiben Sie ein Programm, das Ihnen den Benzinverbrauch pro 100 Kilometer Fahrstrecke nach folgender Formel berechnet:
   Verbrauch auf 100 = Verbrauch insgesamt / gefahrene Kilometer * 100

1.
```
10 REM LOESCHEN BILDSCHIRM
20 PRINT CHR$(147)
30 REM ERZEUGUNG ZUFALLSZAHLEN
40 W1=INT(6*RND(1))+1
50 W2=INT(6*RND(1))+1
60 REM AUSGABE ERGEBNIS
70 PRINT"WURF 1:";W1,"WURF 2:";W2
80 END
```

So ähnlich sollte Ihr Programm aussehen. Die Lösungen, die hier angeboten werden, sind natürlich nur als Lösungsvorschläge anzusehen. Wir haben ja bereits festgestellt, daß mehrere Wege nach Rom führen. Wenn Sie das Programm wiederholt mit RUN / RETURN starten, sehen Sie die Unterschiede in den ausgegebenen Zahlen.
In Zeile 20 wird der Bildschirm mit dem CHR$-Code gelöscht. Die Zeilen 40 und 50 ordnen den Variablen W1 und W2 jeweils neu erzeugte Zufallszahlen zu. Sollten Sie mit der Erzeugung der oberen und unteren Grenze Schwierigkeiten gehabt haben, so lesen Sie noch einmal im Kapitel über die Zufallszahlen nach.

2.
```
10 REM EINGABE WERTE DREIECKSSEITEN
20 INPUT"EINGABE A,B,C IN CM";A,B,C
30 REM BERECHNUNG VON S
40 S=.5*(A+B+C)
50 REM BERECHNUNG FLAECHE
60 F=SQR(S*(S-A)*(S-B)*(S-C))
70 REM AUSGABE FLAECHE
80 PRINT"DIE FLAECHE DES DREIECKS ";
90 PRINT"BETRAEGT";F;" CM↑2"
100 END
```

Bei diesem Programm mußten Sie beachten, daß zuerst S berechnet werden muß, da diese Variable bei der Berechnung der Fläche schon mit verwendet wird. Die Umsetzung der Formel in BASIC dürfte keine Schwierigkeiten bereitet haben. Achten Sie trotzdem

bewußt darauf, daß Sie in Ihren Programmen nicht in die mathematische Schreibweise der Formeln verfallen. Dies geschieht nur allzu leicht und das Programm bricht dann mit einem SYNTAX ERROR ab.

3.
```
10 INPUT"BITTE EINE TASTE DRUECKEN";A$
20 A=ASC(A$)
30 PRINT"ASCII-WERT VON";A$;"=";A
40 END
```

Sollten Sie dieses Programm bereits ausprobiert haben, so kann es sein, daß Sie versucht haben, das Komma oder nur Return einzugeben, um den ASCII-Wert dieser "Zeichen" zu erfahren. Dabei wurde dann vom Rechner die Fehlermeldung ILLEGAL QUANTITY ERROR IN 20 ausgegeben. Das ist einer der Nachteile des INPUT-Befehls, da z.B. das Komma zur Trennung von Variablen benutzt wird. Betätigen Sie nun nur die RETURN-Taste, so wird der Stringvariablen NICHTS zugeordnet, d.h. diese Variable ist leer. Da der Rechner natürlich von NICHTS den ASCII-Wert unmöglich ermitteln kann, wird die o.a. Fehlermeldung ausgegeben. Wie man dieses Problem bei der Eingabe umgeht, wird in einem späteren Kapitel erklärt (siehe GET-Befehl).

4.
```
10 G=9.81
20 INPUT"WIEVIEL SEKUNDEN";T
30 S=.5*G*T↑2
40 PRINT"DER KOERPER FIEL AUS EINER";
50 PRINT" HOEHE VON";S;" METERN"
60 END
```

Interessant ist hier die Zeile 10. Der Variablen G wird am Anfang des Programms der Wert 9.81 zugeordnet. Dieser Vorgang nennt sich VARIABLENINITIALISIERUNG. Das besagt nichts anderes, als daß man am Anfang eines Programms verschiedenen Variablen bestimmte Werte zuordnet. Das hat den Vorteil, daß im Programm selbst nur noch die Variable aufgerufen zu werden braucht und nicht die komplette Zahl, welche unter Umständen recht lang sein kann. Bei

größeren Programmen mit mehr Variablen kann das wiederum Speicherplatz sparen.

5.
```
10 INPUT"WIEVIEL LITER VERBRAUCHT";L
20 INPUT"WIEVIEL KM GEFAHREN";KM
30 V=L/KM*100
40 PRINT"VERBRAUCH AUF 100 KM";V;" LITER"
50 END
```

Dieses Programm braucht wohl nicht näher erläutert zu werden. In seiner Einfachheit erklärt es sich fast von selbst.

Haben Sie diese Aufgaben selbständig zu Ihrer eigenen Zufriedenheit gelöst, so können Sie jetzt getrost zum nächsten Kapitel übergehen. Bei Unsicherheiten schlagen Sie noch einmal in den entsprechenden Passagen nach.

2.4 TAB UND SPC

Diese beiden Befehle werden benutzt, um Daten oder Zeichen an bestimmten Positionen in Bildschirmzeilen auszugeben. Sie sind in der Anwendung sehr ähnlich, jedoch in der Wirkung unterschiedlich. Der TAB-Befehl und der Parameter in Klammern positionieren die Ausgabe immer in Bezug auf den Anfang der aktuellen Bildschirmzeile. Geben Sie folgende Befehlsfolge im Direktmodus ein:

```
PRINT TAB(15) "TEST"
```

Als Ausgabe erhalten Sie das Wort TEST ab der 15. Position in der Bildschirmzeile. Fahren Sie ruhig mit dem Cursor an den Anfang der Zeile, in der das Wort steht, und betätigen Sie 15mal die Taste, die den Cursor nach rechts bewegt. Der Cursor sollte nun unmittelbar vor dem Wort TEST stehen. Schreiben Sie nun eine neue Befehlsfolge und verwenden Sie statt TAB den SPC-Befehl. Nach dem Betätigen der RETURN-Taste erhalten Sie das gleiche Ergebnis. Bei dieser Art der Anwendung haben beide Befehle die gleiche Wirkung. Doch schon beim nächsten Beispiel werden Sie den Unterschied sehen. Löschen Sie zuerst den Bildschirm mit den Tasten SHIFT und CLR/HOME. Geben Sie danach die folgende Befehlsfolge ein:

```
PRINT TAB(5)"TEST 1" TAB(20)"TEST 2"
```

Nachdem Sie die RETURN-Taste betätigt haben wird das Wort TEST 1 ab der 5. Position und das Wort TEST 2 ab der 20. Position ausgegeben. Geben Sie nun die Befehlsfolge erneut ein und ändern Sie dabei das zweite TAB in ein SPC um. Ihre Zeile sollte dann so aussehen:

```
PRINT TAB(5)"TEST 1" SPC(20)"TEST 2"
```

Drücken Sie jetzt die RETURN-Taste, so werden Sie den Unterschied in der Ausgabe auf dem Bildschirm sehen. Das zweite Wort TEST 2 wird nicht an der 20. Position vom Anfang der Zeile gerechnet ausgegeben, sondern an der

20. Position vom letzten Zeichen des Wortes TEST 1 an gerechnet. Das bedeutet, daß der TAB-Befehl immer auf die absolute Position in der Bildschirmzeile und der SPC-Befehl immer auf die relative Position zum letzten ausgegebenen Zeichen bezogen sind. Die Werte, die beiden Befehlen übergeben werden können, dürfen nicht größer als 255 sein.

Bei der Benutzung dieser Befehle in Verbindung mit der Ausgabe auf einen Drucker ist darauf zu achten, daß der TAB-Befehl nach Möglichkeit keine Verwendung findet, da er in Verbindung mit dem PRINT#-Befehl vom Drucker nicht interpretiert oder als SPC interpretiert wird. Daher sollte der TAB-Befehl nur zusammen mit dem "normalen" PRINT-Befehl benutzt werden.

WICHTIG :
Bei TAB(X) wird immer ab der äußersten linken Position der aktuellen Bildschirmzeile gezählt. Bei SPC(X) werden quasi X Leerzeichen eingefügt und dann wird mit der Ausgabe fortgefahren.

2.5 STRINGS

Ein String bezeichnet eine Zeichenkette, die bis zu 255 beliebige Zeichen des COMMODORE 64 Zeichensatzes enthalten kann. Die Stringvariable wird durch das "$" Zeichen gekennzeichnet. A$ würde also eine reguläre Bezeichnung eines Strings darstellen. Die Zuordnung der Zeichen zu einer Stringvariablen erfolgt auf die gleiche Art und Weise wie bei den numerischen Variablen. Der einzige Unterschied besteht darin, daß die Zeichen in Anführungszeichen stehen. Eine gültige Zuordnung zeigt das folgende Beispiel:

```
A$="COMMODORE 64"
```

Versuchen Sie, einer Stringvariablen einen numerischen Wert (ohne Anführungszeichen) zuzuordnen, so erfogt die Fehlermeldung:

```
?TYPE MISMATCH ERROR
```

Die gleiche Fehlermeldung wird ausgegeben, wenn Sie versuchen, einer numerischen Variablen einen String zuzuordnen, z.B.

```
A="TEST"        (FEHLER)!
```

Beim Gebrauch der Strings läßt sich als einziger Rechenoperator das Pluszeichen (+) verwenden. Dieses Zeichen verkettet zwei Strings miteinander. Definieren wir für A$="DISKETTEN" und für B$="LAUFWERK", so ergibt die Verknüpfung mit + den String "DISKETTENLAUFWERK". Ein kleines Programm soll das verdeutlichen.

```
10 A$="DISKETTEN":B$="LAUFWERK"
20 DL$=A$+B$
30 PRINT DL$
40 END
```

In Zeile 10 werden zunächst die Stringvariablen A$ und B$ initialisiert. Zeile 20 ordnet die Verknüpfung der

Variablen A$ und B$ der Variablen DL$ zu. Zeile 30 druckt schließlich den neuen String aus.

Nun kann man Strings nicht nur miteinander verknüpfen, sondern auch auf Gleichheit der Zeichen oder auf die Anzahl der Zeichen hin vergleichen. Dazu kommen wir aber erst, wenn die Vergleichsbefehle durchgesprochen wurden (siehe IF...THEN). Auch bei diesen Vergleichen dürfen grundsätzlich nur Strings mit Strings verglichen werden. Der Vergleich zwischen einer Stringvariablen und einer numerischen Variablen ist nicht zulässig.

Das BASIC des COMMODORE 64 bietet außer der Möglichkeit des Vergleichs und der Verknüpfung noch die Möglichkeit, die Strings zu manipulieren. Solche Befehle wollen wir jetzt besprechen.

Der erste Befehl, den wir uns anschauen, ist der LEFT$ Befehl. Dieser Befehl bewirkt, daß von einem genauer bezeichneten String ein Teilstring gebildet wird. Dazu geben Sie jetzt bitte das folgende Programm ein, um das zu verdeutlichen.

```
10 A$="COMPUTER"
20 B$=LEFT$(A$,1)
30 C$=LEFT$(A$,2)
40 D$=LEFT$(A$,3)
50 E$=LEFT$(A$,4)
60 F$=LEFT$(A$,5)
70 G$=LEFT$(A$,6)
80 H$=LEFT$(A$,7)
90 I$=LEFT$(A$,8)
100 PRINT A$:PRINT B$:PRINT C$:PRINT D$
110 PRINT E$:PRINT F$:PRINT G$:PRINT H$:PRINT I$
120 END
```

Starten Sie nun das Programm mit RUN. Auf der nächsten Seite ist das Ergebnis dieses Programms abgebildet. Dieses Beispiel zeigt deutlich die Funktionsweise des LEFT$-Befehls. In Zeile 10 wird der Stringvariablen A$ die Zeichenkette COMPUTER zugeordnet. Zeile 20 bildet einen linken Teilstring von A$ mit einem Zeichen und

ordnet es der Variablen B$ zu. Zeile 30 bildet wiederum einen linken Teilstring von A$, diesmal jedoch mit zwei Zeichen. Die Zeilen 40 bis 90 sind genauso zu interpretieren. Das bedeutet, das der Befehl LEFT$(A$,X) einen linken Teilstring von A$ mit X Zeichen bildet. Die Zeilen 100 bis 110 dienen der Ausgabe der einzelnen neugebildeten Strings. Leser, die Multistatements verabscheuen, mögen mir diese Platzersparnis verzeihen. Hier nun das Ergebnis des Programms:

C
CO
COM
COMP
COMPU
COMPUT
COMPUTE
COMPUTER

Wie Sie sehen, kann man mit diesem Befehl eine Menge Spielereien betreiben. Jedoch sind natürlich auch ernsthaftere Anwendungen, vor allem in der Datenverarbeitung, vorgesehen.
Der nächste Befehl ist dem LEFT$-Befehl in seiner Wirkung sehr ähnlich. Es handelt sich dabei um den RIGHT$-Befehl. Er unterscheidet sich vom LEFT$-Befehl nur darin, daß er nicht die linken Zeichen eines Strings nimmt, sondern die rechten. Ändern Sie nun in dem Beispielprogramm auf der vorherigen Seite alle LEFT$-Befehle in RIGHT$-Befehle um, beginnend in Zeile 20 mit RIGHT$(A$,1). Starten Sie das Programm erneut mit RUN. Als Ergebnis sollten Sie den folgenden Ausdruck auf dem Bildschirm erhalten:

R
ER
TER
UTER
PUTER
MPUTER
OMPUTER
COMPUTER

Ändern Sie nun noch die Reihenfolge der Zahlen im RIGHT$-Befehl, also beginnend mit der acht und dann rückwärts zählend bis eins, so erhalten Sie genau das umgekehrte Ergebnis. Sie erhalten dann als erstes den Ausdruck COMPUTER und als letztes schließlich nur das R. Diese Beispiele sollten Ihnen nur die Funktionsweise dieser Befehle verdeutlichen. Bei der Verwendung dieser Befehle in Programmen sind Ihrer Phantasie keine Grenzen gesetzt.

Einer der interessantesten Befehle, was die Verarbeitung von Strings angeht, dürfte der Befehl MID$ sein. Mit diesem Befehl haben Sie die Möglichkeit, jedes einzelne oder auch mehrere Zeichen eines Strings auf einmal anzusprechen. Wir werden in einem späteren Kapitel sehen, wie man Laufschriften u.ä. damit erzeugen kann. Zunächst wollen wir uns anhand einfacher Beispiele die Wirkung dieses Befehls anschauen. Geben Sie hierzu das folgende Programm in Ihren Rechner ein:

```
10 A$="DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT"
20 B$=MID$(A$,1,5)
30 C$=MID$(A$,6,5)
40 D$=MID$(A$,6,11)
50 E$=MID$(A$,22,6)+MID$(A$,23,1)
60 PRINT A$
70 PRINT B$
80 PRINT C$
90 PRINT D$
100 PRINT E$
110 END
```

Starten Sie nun das Programm und sehen Sie sich das Ergebnis genau an. Mit dem MID$-Befehl haben Sie also jetzt die Möglichkeit, aus einem String ab einer bestimmten Position eine bestimmte Anzahl von Zeichen auszulesen. Diese werden dann in einem neuen String abgelegt. Die allgemeine Schreibweise des Befehls lautet:

MID$(M$,X,Y)

Dabei bedeutet M$ der Name des Strings, der benutzt

werden soll; X bezeichnet die Position, ab welchem Zeichen der Zugriff beginnen soll und Y bestimmt die Anzahl der Zeichen. Die Positionen werden immer von links nach rechts gezählt. So ordnet Zeile 20 der Variablen B$ den Teilstring DONAU zu. Es sollte ein neuer String aus A$ gebildet werden, der insgesamt fünf Zeichen enthält und der mit dem ersten Zeichen von A$ beginnt. Auf die gleiche Art wurde der String B$ gebildet. Hier wurde mit dem 6. Zeichen begonnen, so daß sich der neue String DAMPF ergab. Zeile 40 erkärt sich demnach von selbst. Interessant ist wiederum die Zeile 50. Hier wurde durch die Verknüpfung zweier Teilstrings von A$ ein neuer Begriff gebildet, der nicht direkt aus dem ursprünglichen String ablesbar ist, nämlich GESELLE. Bei der Anwendung haben Sie gesehen, daß durch den Befehl MID$ die Befehle LEFT$ und RIGHT$ ersetzt werden können. In unseren Beispielen wurden die Position und Anzahl der Zeichen durch Zahlen bezeichnet. Die Angabe durch Variablen und arithmetische Ausdrücke ist genauso erlaubt. Dieses Beispiel sollte vorerst zur Verdeutlichung der Funktionsweise des MID$ Befehls ausreichen.

Bevor wir uns den nächsten Befehl anschauen, gebe ich Ihnen eine kleine Aufgabe. Wieviele Zeichen (ohne Anführungszeichen) beinhaltet der String aus dem letzten Beispiel (DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT)?
Sie haben richtig gezählt, es sind genau 33 Zeichen.

Ich habe Ihnen diese Aufgabe natürlich nicht ohne Hintergedanken gestellt. Sie haben bestimmt schon geahnt, daß der Befehl, den wir jetzt besprechen wollen, damit zusammenhängt.

Sie können nämlich die Länge eines Strings mit dem LEN(X$)-Befehl ermitteln. Das Ergebnis ist numerisch und kann einer entsprechenden Variablen zugeordnet werden. Haben Sie nach dem Start des letzten Beispielprogrammes noch kein NEW (löscht Programm und Variable) oder CLR (setzt Variable auf Null) eingegeben, so geben Sie jetzt folgendes im Direktmodus ein:

```
PRINT LEN(A$)
```

und drücken Sie RETURN. Das Ergebnis sollte 33 sein. Sie haben soeben die Anzahl der Zeichen von A$ ermittelt. Bei der Anwendung dieses Befehls spielt es keine Rolle, aus welchen Zeichen sich der String zusammensetzt. Gezählt werden alle Zeichen, die sich im String befinden, also auch Leerzeichen. Merken Sie sich einfach, daß mit LEN die Länge eines Strings ermittelt wird.

Der VAL(X$)-Befehl befaßt sich mit der Umwandlung eines Strings X$ in einen numerischen Ausdruck. Die Zeichenkette wird also in eine Zahl umgewandelt. Beginnt der String mit einem Zeichen, das nicht in eine Zahl umgewandelt werden kann, z.B. mit einem Buchstaben, so erhält man als Ergebnis Null. Befinden sich innerhalb des Strings Buchstaben oder andere Zeichen, die nicht in eine Zahl zu übersetzen sind, so wird nur der erste Teil des Strings mit den Ziffern übersetzt. Die nachfolgenden Beispiele sollen das verdeutlichen.

BEISPIELE:

a)
```
10 A$="343.45"
20 A=VAL(A$)
30 PRINT A

Ergebnis:
343.45
```

b)
```
10 B$="D34.87F"
20 B=VAL(B$)
30 PRINT B

Ergebnis:
0
```

c)
```
10 C$="234FFC54"
20 C=VAL(C$)
30 PRINT C
```

Ergebnis:
234

d)
```
10 D$="33,21"
20 D=VAL(D$)
30 PRINT D
```

Ergebnis:
33

Geben Sie diese Beispiele ruhig in den Computer ein und probieren Sie sie nacheinander aus. Das Beispiel a) zeigt den Fall auf, bei dem der komplette String in eine Zahl übersetzt werden kann. Der String von Beispiel b) beginnt mit einem Zeichen, das nicht in eine Zahl übersetzt werden kann, und wird daher als Null interpretiert. Beispiel c) zeigt einen "gemischten" String, bei dem nur der erste Ziffernteil umgewandelt wird. Beispiel d) soll nur verdeutlichen, daß das Komma im Unterschied zum Dezimalpunkt ebenfalls nicht umgewandelt werden kann, und daß somit der restliche Ziffernteil nicht mehr berücksichtigt wird.

Der Befehl, der genau das Gegenteil von VAL$ bewirkt, also einen numerischen Ausdruck in einen String verwandelt, ist der STR$(X)-Befehl. Dabei müssen Sie beachten, daß der erzeugte String als erstes das Leerzeichen beinhaltet. Ist die Zahl positiv, so handelt es sich dabei um ein Leerzeichen. Zwei Beispiele sollen das wieder verdeutlichen.

BEISPIELE:

a)
```
10 A=1234
20 A$=STR$(A)
30 PRINT A$

Ergebnis:
 1234
```

b)
```
10 B=-1234
20 B$=STR$(B)
30 PRINT B$

Ergebnis:
-1234
```

Somit beinhalten die neu gebildeten Strings in Beispiel á) und b) jeweils fünf Zeichen. Es können sowohl die Werte von Variablen als auch Zahlen selbst in Strings umgewandelt werden. So könnte in Beispiel a) anstatt STR$(A) auch STR$(1234) stehen. Am Ergebnis würde das nichts ändern.

Einen speziellen "String" bzw. eine spezielle Stringvariable wollen wir uns in diesem Kapitel noch anschauen. Es handelt sich hierbei um eine interne Uhr im COMMODORE 64. Diese Uhr wird beim Einschalten des Rechners auf Null gesetzt und läuft dann solange wie der Rechner eingeschaltet ist. Dieser Wert wird also laufend aktualisiert und in der Variablen TI$ abgelegt. Dieser String setzt sich aus sechs Zeichen zusammen, und zwar je zwei für Stunden, Minuten und Sekunden. Diese Variable zählt nur bedingt zu den systemreservierten Variablen. Man hat nämlich die Möglichkeit, diese Variable zu beeinflussen, d.h. man kann ihr einen Wert übergeben, der dann laufend aktualisiert wird. Geben Sie nun das folgende Programm in Ihren Rechner ein:

```
10 PRINT CHR$(19);TI$
20 GOTO 10
```

Starten Sie es jetzt mit RUN und beobachten Sie die linke obere Ecke auf Ihrem Bildschirm. Dort sollten jetzt sechs Ziffern stehen, wovon die äußerst rechte laufend im Sekundentakt ihren Wert vergrößert. Daß diese sechs Ziffern immer an der gleichen Stelle erscheinen, wird durch CHR$(19) erreicht. Es handelt sich hierbei um den Tastencode der HOME-Taste. Damit wird erreicht, daß immer ab der oberen linken Bildschirmecke die Ausgabe durch PRINT geschieht und somit die alte Ausgabe überschrieben wird.
Sie haben damit eine Digitaluhr auf Ihren Bildschirm "programmiert". Unterbrechen Sie nun das Programm mit der RUN/STOP-Taste. Es erscheint die Meldung:

BREAK IN (ZEILENNUMMER)

Nun wollen wir erreichen, daß unsere Uhr ab einer bestimmten Uhrzeit an zu laufen fängt. Dazu erweitern wir unser Programm um eine Zeile:

```
10 TI$="120000"
20 PRINT CHR$(19);TI$
30 GOTO 20
```

Starten Sie dieses Programm, so werden Sie bemerken, daß die Uhr mit dem Wert 12.00 Uhr zu laufen beginnt. Damit haben Sie die Möglichkeit, die Uhr Ihres Rechners zu stellen. Achten Sie darauf, daß TI$ immer sechs Zeichen als Eingabe verlangt, da ansonsten die Fehlermeldung

?ILLEGAL QUANTITY ERROR

ausgegeben wird.

Das Programm können Sie wieder mit der RUN/STOP-Taste unterbrechen.

Der Befehl GOTO, den Sie bereits zweimal in den letzten Programmen verwendet haben, wird im nächsten Kapitel eingehend besprochen. Dieser Befehl kann den eigentlichen

Programmablauf, der ja durch die Zeilennummern bestimmt wird, manipulieren.

Bevor wir nun zum nächsten Kapitel übergehen, sollten Sie noch die Aufgaben auf der nächsten Seite lösen, damit Sie die neubesprochenen Befehle anwenden lernen. Und nun wie immer viel Erfolg beim Lösen der Aufgaben.

1. Welcher Unterschied besteht zwischen den beiden nachfolgenden Befehlsfolgen in Bezug auf die Ausgabe auf dem Bildschirm? Geben Sie sie nicht in den Rechner ein, sondern versuchen Sie sie so zu beantworten.

   ```
   PRINT SPC(5)"TEST" TAB(15)"TEST"

   PRINT TAB(5)"TEST" TAB(15)"TEST"
   ```

   a) kein Unterschied

   b) Bei der ersten Befehlsfolge werden erst fünf Leerzeichen erzeugt, dann erfolgt die Ausgabe. Nach weiteren 15 Leerzeichen erscheint die zweite Ausgabe. Bei der zweiten Befehlsfolge werden wieder erst fünf Leerzeichen erzeugt, aber schon nach 10 weiteren Leerzeichen erfolgt die zweite Ausgabe, da der zweite TAB-Befehl sich auf den Anfang der aktuellen Zeile bezieht.

   c) Bei der ersten Befehlsfolge werden erst fünf Leerzeichen erzeugt, dann erfolgt bereits nach 10 weiteren Leerzeichen die zweite Ausgabe. Bei der zweiten Befehlsfolge werden auch erst fünf Leerzeichen erzeugt, aber die zweite Ausgabe erfolgt erst nach 15 weiteren Leerzeichen.

2. Welchen Ausdruck erhält man für B$ mit folgender Befehlsfolge auf dem Bildschirm, wenn als String A$="BOHRMASCHINE" vorgegeben ist?

   ```
   B$=MID$(A$,1,1)+MID$(A$,12,1)+MID$(A$,10,2)
   ```

3. Welchen Ausdruck erhält man mit der folgenden Befehlsfolge für A$, wenn A$="ROTOR" vorgegeben ist?

```
A$=LEFT$(A$,3)+RIGHT$(A$,2)
```

4. Wie muß die Befehlsfolge aussehen, damit, bei vorgegebenem A$="SCHREIBMASCHINENKURSUS", man als Ergebnis B$="REIBEKUCHEN" bekommt?

LÖSUNGEN

1. a) ist richtig. Es existiert kein Unterschied zwischen den beiden Befehlsfolgen bei der Ausgabe auf dem Bildschirm.

2. Man erhält den Ausdruck BEIN.

3. Man erhält wieder den Ausdruck ROTOR.

4.
```
B$=MID$(A$,4,4)+MID$(A$,5,1)+MID$(A$,17,2)
   +MID$(A$,2,2)+MID$(A$,15,2)
```

   Das ist eine mögliche Lösung.

3. ERWEITERTE PROGRAMMSTRUKTUREN

Haben wir uns bisher mit den linearen Programmabläufen befaßt, so steigen wir jetzt in die Programmierung von Programmsprüngen bzw. von Programmverzweigungen ein. Lineare Programmabläufe besitzen den Nachteil, daß das Programm einmal abläuft und dann wieder neu gestartet werden muß, um ein neues Ergebnis zu erhalten. Es finden keine Verzweigungen irgendeiner Art statt. Gäbe es also keine Befehle, die solche Programmverzweigungen durchführen könnten, so wäre man bei der Programmierung so stark eingeschränkt, daß man tatsächlich nur noch sehr einfache Probleme in Programme umsetzen könnte.

3.1 UNBEDINGTE PROGRAMMSPRÜNGE

Die erste und vielleicht einfachste Art einer Programmverzweigung haben Sie bereits in dem Beispiel mit der internen Uhr des COMMODORE 64 kennengelernt. Es handelt sich dabei um den GOTO-Befehl. Dieser Befehl veranlaßt das Programm, von seinem eigentlichen Ablauf, der durch die Zeilennummern vorgegeben ist, abzuweichen. Unbedingter Programmsprung heißt er deswegen, weil er an keine Bedingung geknüpft ist, d.h. daß das Programm an dieser Stelle auf alle Fälle diesen Sprung ausführt.

Nun haben Sie in den Programmen mit der Uhr festgestellt, daß Sie diese nur mit der RUN/STOP-Taste unterbrechen konnten. Das ist wiederum der Nachteil dieser unbedingten Programmsprünge, daß man mit ihnen eigentlich nur sogenannte Endlosschleifen erzeugen kann. Wurde das Programm dann einmal gestartet, so hat man nur noch die Möglichkeit, es mit der genannten Taste zu unterbrechen. Wir wollen diesen Befehl nun an einem bekannten Beispiel anwenden. Sie erinnern sich bestimmt noch an die Aufgabe, in der das Idealgewicht einer Person berechnet werden sollte.

Stellen Sie sich nun vor, Sie geben eine Party und wollen als kleinen Gag dieses Programm vorführen. Jeder der Gäste soll sein Idealgewicht erfahren. Ohne den

GOTO-Befehl müßte das Programm bei jeder neuen Berechnung erneut gestartet werden. Daher ändern wir das Programm dahingehend ab, daß vor der letzten Programmzeile ein GOTO-Befehl eingefügt wird, der den Rechner veranlaßt, wieder an den Anfang des Programms zu springen. Unser Programm sähe dann folgendermaßen aus:

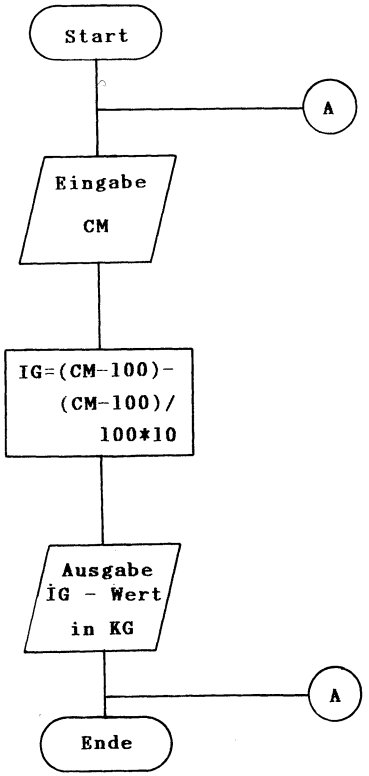
```
10 INPUT"EINGABE KOERPERGROESSE IN CM";CM
20 REM BERECHNUNG IDEALGEWICHT
30 IG=(CM-100)-(CM-100)/100*10
40 REM AUSGABE
50 PRINT"IHR IDEALGEWICHT IST";IG;"KG"
60 REM UNBEDINGTER SPRUNG MIT GOTO
70 GOTO 10
80 END
```

In diesem Programm wurde die Berechnung des Idealgewichts in einer Zeile untergebracht. Nach der Ausgabe des Ergebnisses trifft das Programm in Zeile 70 auf den GOTO-Befehl und verzweigt nach Zeile 10. Somit kann erneut ein Wert zur Berechnung übergeben werden. Wollen Sie ein Programm innerhalb von Zeile 10 abbrechen, d.h. es erwartet durch den INPUT-Befehl eine Eingabe, so müssen Sie die RUN/STOP-Taste und die RESTORE-Taste gleichzeitig drücken. Die Zeile 80 hätte man nicht einzugeben brauchen, da das Programm ja durch den GOTO-Befehl diese Zeile nie erreicht. Der Datenflußplan wird durch diesen Befehl nicht beeinträchtigt. Das untere Schaubild soll zeigen, wie er auszusehen hätte.

Datenflußplan für Berechnung Idealgewicht

Der Programmablaufplan PAP ändert sich durch diesen Befehl wie im unteren Schaubild dargestellt. Ein Symbol ist hinzugekommen. Es handelt sich dabei um den Konnektor. Er bezeichnet die Stelle, ab der das Programm weitergehen soll, wenn es den Absprungkonnektor erreicht hat. Der Absprungkonnektor steht an der Stelle, an der im Programm der GOTO-Befehl zu finden ist. Der Einsprungkonnektor steht demnach direkt nach dem Startsymbol. Beide Konnektoren wurden mit einem A gekennzeichnet, da es sich ja um ein Konnektorpaar handelt. Im folgenden nun der PAP:

PAP für Programm Idealgewicht



Das Ende-Symbol hätte hier wieder wegfallen können, der Vollständigkeit wegen wurde es dennoch angefügt. An diesem Beispiel konnte man erkennen, daß mit dem GOTO-Befehl, ohne daß vorher eine Bedingung abgefragt wird, im Grunde nur Endlosschleifen erzeugt werden können. Abhilfe schaffen hier die BEDINGTEN PROGRAMMSPRÜNGE.

3.2 BEDINGTE PROGRAMMSPRÜNGE

Die Stärke eines Computers liegt u.a. in dessen Fähigkeit, logische Entscheidungen treffen bzw. Vergleiche anstellen zu können. Er kann z.B. testen, ob eine Variable größer oder kleiner Null ist, und in Abhängigkeit davon, ob das Ergebnis WAHR oder FALSCH ist, entsprechend innerhalb des Programms verzweigen. Der IF...THEN-Befehl zählt zu diesen Befehlen, die einen Vergleich ausführen.

3.2.1 IF...THEN

Trifft der Rechner bei der Programmausführung auf einen IF...THEN-Befehl, so überprüft er die Bedingung, die dem IF folgt. Ist die Bedingung WAHR also erfüllt, so führt er die Anweisungen bzw. Befehle, die dem THEN folgen, aus. Trifft die Bedingung hinter IF nicht zu, ist sie also FALSCH, so fährt der Rechner mit der nächsten Programmzeile fort. Alle Anweisungen oder Befehle, die dem THEN folgen, werden dann ignoriert.

Dem IF können logische Operatoren, Zeichenketten, Variablen, Vergleiche und Zahlen folgen oder Kombinationen hiervon. Meistens folgt dem THEN eine Zeilennummer, zu der das Programm verzweigen soll. Möglich sind auch neue Wertzuweisungen von Variablen oder Sprünge in Unterprogramme. Dazu kommen wir aber erst in einem späteren Kapitel. Schauen wir uns zunächst ein ganz einfaches Beispiel zur Anwendung des IF...THEN-Befehls an.

```
10 INPUT"GEBEN SIE EINE ZAHL EIN";Z
20 IF Z < 0 THEN 50
30 IF Z > 0 THEN 70
40 IF Z = 0 THEN 90
50 PRINT"DIE ZAHL IST KLEINER NULL"
60 GOTO 100
70 PRINT"DIE ZAHL IST GROESSER NULL"
80 GOTO 100
90 PRINT"DIE ZAHL IST GLEICH NULL"
100 END
```

Bei diesem Programm können Sie eine beliebige Zahl eingeben und der Rechner sagt Ihnen dann, ob diese Zahl größer, kleiner oder gleich Null ist. Natürlich wissen Sie das selbst; dieses einfache Beispiel soll aber nur die Anwendung des IF...THEN-Befehls in einem Programm und die Reaktion des Rechners auf diese Anweisung verdeutlichen. Geben Sie nun eine Zahl ein, die größer als Null ist, so trifft der Rechner zunächst auf die Zeile 20. Dort wird überprüft, ob die Zahl kleiner als Null ist. Diese Bedingung wird nicht erfüllt, also fährt der Rechner mit der Ausführung in der nächsten Programmzeile fort. Dort wird überprüft, ob die eingegebene Zahl größer als Null ist. Diese Bedingung ist erfüllt und der Rechner springt entsprechend der Anweisung, die dem THEN folgt, in Zeile 70. Zeile 70 gibt die Meldung auf den Bildschirm, daß die eingegebene Zahl größer als Null ist. In Zeile 80 trifft der Rechner auf den unbedingten Sprungbefehl GOTO und springt in Zeile 100, wo das Programm beendet wird. Wollen Sie, daß das Programm kontinuierlich läuft, so brauchen Sie nur in der Zeile 100 den END-Befehl durch GOTO 10 zu ersetzen.

Nach diesem einfachen Beispielprogramm wollen wir uns nun einem schon etwas komplizierteren Programm zuwenden. Sie kennen sicherlich alle das Spielchen, wo sich jemand eine Zahl ausdenkt und ein anderer diese erraten muß. Nach jeder Frage wird nur gesagt, ob die Zahl größer, kleiner oder gleich der gedachten Zahl ist. Dieses Spiel wollen wir nun auf dem Rechner realisieren. Geben Sie dazu das folgende Programm ein:

```
10 REM ZAHLENRATEN
20 PRINT CHR$(147);CHR$(17)
30 PRINT"GEBEN SIE ZWEI ZAHLEN FUER ";CHR$(17)
40 PRINT"DIE OBERE UND UNTERE GRENZE EIN"
50 INPUT"UNTERE GRENZE";U
60 INPUT"OBERE GRENZE";O
70 Z=INT(O-U*RND(1))+U
80 INPUT"RATEN SIE";SZ
90 IF SZ < Z THEN 120
100 IF SZ > Z THEN 140
110 IF SZ = Z THEN 160
120 PRINT"DIE GEDACHTE ZAHL IST GROESSER"
130 GOTO 80
140 PRINT"DIE GEDACHTE ZAHL IST KLEINER"
150 GOTO 80
160 PRINT CHR$(147)"HURRA, SIE HABEN DIE ZAHL GEFUNDEN"
170 PRINT"WOLLEN SIE NOCHMAL JA/NEIN"
180 INPUT A$
190 IF A$="JA" THEN 20
200 END
```

Die ersten Zeilen dieses Programms brauchen wohl nicht näher erläutert zu werden. Lediglich kurz etwas zu der Zeile 20: CHR$(147) = Löschen des Bildschirms und CHR$(17) = Cursor wird eine Zeile nach unten gerückt.

In den Zeilen 30 bis 60 wird die Eingabe der Intervallgrenzen für die zu suchende Zahl verlangt. In Zeile 70 wird mittels der Zufallsfunktion RND die zu suchende Zahl über die vorher eingegebenen Intervallgrenzen bestimmt. Sollte Ihnen die Zeile 70 momentan Schwierigkeiten bereiten, so schlagen Sie nochmal die Seiten 54 f auf. Zeile 80 fordert Sie nun auf, eine Zahl einzugeben. Diese Zahl wird dann in den Zeilen 90 bis 110 mit der gebildeten Zufallszahl, die in der Variablen Z abgelegt wurde, verglichen. Je nachdem ob die Zahl größer, kleiner oder gleich der gedachten Zufallszahl ist, verzweigt der Rechner in die entsprechende Zeile und fährt dort mit dem Programm fort. Haben Sie die Zahl erraten, so springt der Rechner in die

Zeile 160. In Zeile 170 werden Sie gefragt, ob Sie das Spiel fortsetzen wollen. Geben Sie JA ein, so ist die Bedingung in Zeile 190 erfüllt und das Programm beginnt von neuem.

In den Zeilen 90 bis 110 wird der IF...THEN-Befehl also zum Vergleich zwischen der Spielerzahl (SZ) und der Zufallszahl (Z) benutzt. In Zeile 190 hingegen wird der IF...THEN-Befehl zum Vergleich einer Stringvariablen benutzt. Dabei ist zu beachten, daß, um die Bedingung WAHR werden zu lassen, beide Strings absolut gleich sein müssen. Das bedeutet, daß auch Leerzeichen mit berücksichtigt werden müsen. Sie könnten in Zeile 180 also ruhig YES eingeben, trotzdem würde das Programm beendet, da nur dann in Zeile 20 gesprungen wird, wenn Sie genau die beiden Zeichen J und A eingeben, also den String JA.

Mit dem IF...THEN-Befehl haben wir jetzt auch die Möglichkeit, gesteuerte Schleifen zu programmieren. Gesteuert heißt, daß Sie nicht willkürlich lange ablaufen, sondern an eine Bedingung geknüpft solange laufen, bis diese Bedingung erfüllt ist oder eben nicht. Dadurch lassen sich mit unserem bisherigen Wissen schon komplexere Programme "fahren". Wie eine solche gesteuerte Schleife programmiert wird, soll Ihnen das nachfolgende Programm zeigen. Wollen Sie z.B. das Einmaleins mit 3 ausgegeben haben, so programmieren Sie wie folgt:

```
10 A=3
20 PRINT A
30 A=A+3
40 IF A > 30 THEN 60
50 GOTO 20
60 END
```

In Zeile 10 wird zunächst die Variable A mit dem Wert 3 initialisiert. Zeile 20 gibt den aktuellen Wert von A auf dem Bildschirm aus. In Zeile 30 ist ein sogenannter Zähler aufgebaut, der immer zum aktuellen Inhalt der Variablen A den Wert 3 hinzuaddiert. In Zeile 40 wird überprüft, ob A schon den Wert 30 überschritten hat.

Solange A kleiner oder gleich 30 ist, fährt das Programm mit dem GOTO-Befehl in Zeile 50 fort. Damit haben wir eine Schleife erzeugt, die in unserem Falle genau 10mal durchlaufen wird. Somit haben wir jetzt auch eine Möglichkeit kennengelernt, Schleifen zu erzeugen, die nur eine bestimmte Anzahl von Durchläufen ausführen.

Auf der nächsten Seite finden Sie nun wieder einige Aufgaben, damit Sie mit den neu erlernten Befehlen vertraut werden. Wie immer viel Spaß beim Lösen der Aufgaben!

1. Schreiben Sie ein Programm, das je nach Jahreseinkommen einmal einen Steuerbetrag von 33 Prozent oder von 51 Prozent berechnet. Die Grenze soll bei einem Jahreseinkommen von 50000 DM liegen, d.h. alle Beträge, die größer als 50000 DM sind, müssen mit 51 Prozent versteuert werden. Die Ausgabe des Ergebnisses soll mit einem Begleittext geschehen.

2. Schreiben Sie ein Programm, das Ihnen die Summe der Zahlen von 1 bis 100 berechnet.

3. Schreiben Sie ein Programm, das Ihnen 6 Zufallszahlen in den Grenzen 1 bis 49 ausgibt.

4. Welche Zahlen werden durch das folgende Programm ausgegeben? Lösen Sie die Aufgabe, ohne das Programm einzugeben.

```
10 A=7
20 A=A+5:Z=Z+1
30 IF Z < 9 THEN 20
40 PRINT A,Z
50 END
```

5. Schreiben Sie ein Programm, das Ihnen aus einem beliebigen String einen beliebigen Teilstring heraussucht. Benutzen Sie zum Test den String A$="INFORMATIK" und lassen Sie den Teilstring B$="FORMAT" heraussuchen und ausgeben.

1.
```
10 REM EINGABE JAHRESEINKOMMEN
20 INPUT"JAHRESEINKOMMEN IN DM";JV
30 IF JV > 50000 THEN 70
40 REM BERECHNUNG 33 PROZENT
50 ZS=JV/100*33
60 GOTO 90
70 REM BERECHNUNG 51 PROZENT
80 ZS=JV/100*51
90 PRINT"ZU ZAHLENDER STEUERBETRAG ";
100 PRINT ZS;" DM"
110 END
```

In Zeile 20 wird nach dem Jahreseinkommen gefragt. Der eingegebene Wert wird der Variablen JV zugeordnet. In Zeile 30 wird überprüft, ob das Einkommen größer als 50000 DM ist. Trifft dies nicht zu, so werden die 33 Prozent vom Einkommen ermittelt und ausgegeben. Ist das Einkommen dagegen größer als 50000 DM, so werden in Zeile 80 die 51 Prozent berechnet und angezeigt. Diese Aufgabe dürfte eigentlich keine größeren Schwierigkeiten bereitet haben.

2. Diese Aufgabe konnte man auf mindestens zweierlei Art lösen. Zunächst die Lösung, die den Befehl IF...THEN verwendet.

```
10 REM SUMME 1 BIS 100
20 A=A+1
30 S=S+A
40 IF A < 100 THEN 20
50 PRINT"SUMME VON 1 BIS 100 =";S
60 END
```

In Zeile 20 haben wir unseren Zähler für die einzelnen Summanden von 1 bis 100. Zeile 30 berechnet die Summe der bis dahin aufgetretenen Werte von A, also 1+2+3+4

usw. Zeile 40 führt den bekannten Vergleich aus und in Zeile 50 wird schließlich die Summe S der einzelnen Summanden von 1 bis 100 ausgegeben.
Die zweite Lösung ergibt sich aus der Tatsache, daß wir es hier mit einer arithmetischen Reihe zu tun haben, d.h. die Differenz zwischen den einzelnen Gliedern ist konstant. Die Summe läßt sich nach der Formel

SN=N/2(A1+AN)

berechnen. Dabei ist "n" die Anzahl der auftretenden Glieder der Reihe, "A1" das erste Glied und "An" das letzte Glied. Demnach bietet uns die zweite Lösung sogar einen allgemeineren Lösungsweg an. Das Programm dazu könnte ungefähr so aussehen:

```
10 INPUT"ANZAHL DER GLIEDER";N
20 INPUT"ERSTES GLIED";A1
30 INPUT"LETZTES GLIED";AN
40 REM BERECHNUNG
50 SN=N/2*(A1+AN)
60 REM AUSGABE
70 PRINT"SUMME IST";SN
80 END
```

3.
```
10 REM 6 AUS 49
20 Z=Z+1
30 L=INT(49*RND(1))+1
40 IF Z > 6 THEN END
50 PRINT L;
60 GOTO 20
```

In diesem Programm wurde der END-Befehl einmal nicht an das Ende des Programms gesetzt. Es besteht also keine Notwendigkeit, den END-Befehl unbedingt in die letzte Zeile des Programms zu schreiben. Das Programm sollte ansonsten von seinem einfachen Aufbau her verstanden worden sein.

4. Bei dieser Aufgabe mußten Sie erkennen, daß jeweils

nur die letzten Werte von A und Z ausgegeben werden. Der PRINT-Befehl steht nämlich außerhalb der eigentlichen Schleife. Damit hätte Ihre Lösung lauten müssen:

```
52      9
```

Sollten Sie als zweiten Wert eine acht stehen haben, so bedenken Sie, daß das Programm solange nach Zeile 20 springt, wie Z kleiner als 9 ist. Erst wenn Z gleich 9 ist, wird die Bedingung nicht erfüllt und es erfolgt die Ausgabe in Zeile 40.

5.
```
10 REM EINGABE STRING UND TEILSTRING
20 INPUT"WELCHER STRING";A$
30 INPUT"WELCHER TEILSTRING";B$
40 I=I+1
50 C$=MID$(A$,I,LEN(B$))
60 IF C$=B$ THEN PRINT"ENTHALTEN":END
70 IF I > LEN(A$) THEN PRINT"NICHT ENTHALTEN":END
80 GOTO 40
```

Diese Aufgabe war, zugegeben, schon recht schwierig. Ihr Programm muß dem obigen nun nicht aufs Haar gleichen. Jedoch sollte es die Erzeugung des Vergleichsstrings in Zeile 50 in etwa beinhalten, da das ja die eigentliche Schwierigkeit ist. Die Aufgabenstellung bezog sich auf einen beliebigen String, d.h. unabhängig davon, nach welchen und nach wieviel Zeichen der ursprüngliche String durchsucht werden sollte. So mußte dem MID$-Befehl die Länge des zu suchenden Strings über den LEN-Befehl mitgeteilt werden. Der Zähler in Zeile 40 sorgt dafür, daß die Position von C$ immer um eine Stelle in A$ nach rechts gerückt wird. In Zeile 60 findet der Vergleich statt, ob der zu suchende String B$ mit dem momentanen String in C$ übereinstimmt. Zeile 70 fragt ab, ob die gesamte Länge von A$ schon durchsucht wurde und B$ somit nicht enthalten ist. Zur Veranschaulichung der Funktion des Programms soll das folgende Bild beitragen:

String A$="INFORMATIK" soll durchsucht werden nach
STRING B$="FORMAT".

Zeichenanzahl von B$=6
somit werden folgende Teilstrings gebildet:

1. INFORM
2. NFORMA
3. FORMAT

String 3 ist der gesuchte String.

So, das war eine harte Nuß, die Sie da zu knacken hatten. Überzeugen Sie sich aber davon, daß Sie dieses Programm bis in alle Einzelheiten verstanden haben. Sind Sie noch unsicher, so arbeiten Sie das Programm Schritt für Schritt noch einmal durch. Dann können Sie beruhigt das nächste Kapitel in Angriff nehmen.

3.2.2 FOR...TO...NEXT

Bisher haben wir eine Schleife mit dem IF...THEN-Befehl erzeugt. Das Prinzip war dabei, daß ein Zähler erzeugt wurde, dessen Wert kontinuierlich hoch- oder heruntergezählt wurde. An bestimmten Stellen im Programm wurde der Wert des Zählers überprüft und entsprechend dem Ergebnis der Prüfung (wahr oder falsch) sprang das Programm in eine andere Zeile. Die Erzeugung der Schleifen auf diese Art und Weise ist recht umständlich, zumal der Zähler und die dazugehörige Abfrage extra programmiert werden müssen. Sie ahnen sicher, daß BASIC eine komfortablere Lösung anbieten kann. Es handelt sich dabei um die FOR...NEXT-Schleifen. Zum Einstieg in diese Art der Erzeugung von Schleifen schauen wir uns das an einem Beispielprogramm an.

```
10 REM AUSGABE DER ERSTEN 10 QUADRATZAHLEN
20 PRINT CHR$(147);CHR$(17)
30 PRINT "AUSGABE DER ERSTEN 10 QUADRATZAHLEN";CHR$(17)
40 FOR I = 1 TO 10
50 PRINT "QUADRATZAHL VON";I;"=";I*I
60 NEXT I
70 PRINT "ENDE"
```

Geben Sie das Programm nun in Ihren Rechner und starten Sie es. Die Funktionsweise ist vom Prinzip her dem IF...THEN-Befehl ähnlich. Nur ist die Programmierung von Schleifen bzw. Vorgängen, die sich wiederholen, mit FOR...NEXT eleganter und spart außerdem Speicherplatz.

I wird hier als LAUFVARIABLE bezeichnet, der ein ANFANGSWERT zugeordnet wird. Dies ist in unserem Falle die 1. Der Anfangswert wird dann jeweils um 1 erhöht, bis der ENDWERT überschritten wird. Jeder Befehl, der zwischen FOR und NEXT vorkommt, wird demnach so oft wiederholt, wie die Schleife durchlaufen wird. Anfangswert und Endwert können Zahlen, Variablen und arithmetische Ausdrücke sein.

Dazu einige Beispiele:

```
10 A=10:B=20
20 FOR Z=A TO B
30 PRINT Z;
40 NEXT Z
50 END
```

In diesem Beispiel wurden zuerst die Variablen A und B initialisiert. Zeile 20 baut dann mit diesen Variablen die Schleife auf. Zeile 30 gibt solange die Werte von Z aus, bis die Laufvariable größer als 20 ist. Dieser Vorgang ist vergleichbar mit dem IF...THEN-Befehl. Dort könnte das dann so aussehen:

```
IF Z > 20 THEN 50
```

Die FOR...NEXT Schleife wird in unserem Falle solange durchlaufen, bis Z größer als 20 ist. Sie können das überprüfen, indem Sie nach Ablauf des Programms im Direktmodus den Befehl

```
PRINT Z
```

eingeben. Als Ausgabe für Z erhalten Sie dann den Wert 21! Das nächste Beispiel soll zeigen, daß auch arithmetische Ausdrücke Verwendung finden können.

```
10 A=10:B=15:C=5
20 FOR Z=A TO A+B-C
30 PRINT Z;
40 NEXT Z
50 END
```

Der Durchlauf der Schleife geschieht wie im ersten Beispiel. Der einzige Unterschied ist der, daß sich der Endwert aus dem Ausdruck A+B-C errechnet.

Will man, daß die Schleife eine andere Schrittweite als 1 annimmt, so muß man zusätzlich durch STEP die Schrittweite bestimmen. Das folgende Beispiel ergibt in Schritten von 2 die Ausgabe der geraden Zahlen zwischen 2

und 20.

```
10 REM GERADE ZAHLEN VON 2 BIS 20
20 FOR I=2 TO 20 STEP 2
30 PRINT I
40 NEXT I
50 END
```

Die Laufvariable bzw. der Anfangswert, der Endwert und die Schrittweite dürfen auch negative oder gebrochene Zahlen sein. Als Beispiel wollen wir einen Count-Down programmieren.

```
10 REM COUNT DOWN
20 FOR I=20 TO 0 STEP -1
30 PRINT I
40 NEXT I
50 END
```

Starten Sie das Programm, so läuft die Ausgabe sehr schnell vor Ihren Augen ab. Normalerweise sollte ein Count-Down ja in Sekundenschritten rückwärts zählen. Auch dafür gibt es eine Lösung. Man kann die FOR...NEXT Schleifen ineinander schachteln. Was das bedeutet, zeigt das nächste Beispiel.

```
10 REM COUNT DOWN
20 FOR I=20 TO 0 STEP -1  -------+
30 PRINT I                       +
40 FOR Z=0 TO 1000  ------+      +
50 REM ZEITSCHLEIFE       +      +
60 NEXT Z           ------+      +
70 NEXT I                 -------+
80 END
```

Geben Sie dieses Programm ein (natürlich ohne die nebenstehenden Grafikzeichen) und starten es, so werden Sie bemerken, daß fast genau im Sekundentakt zurückgezählt wird. Dafür sorgt eine sogenannte Zeitschleife in den Zeilen 40 bis 60. Solche Zeitschleifen werden sehr oft benutzt, um Textausgaben von Programmen aus eine Zeitlang zum Lesen anzuhalten.

Selbstverständlich können in diesen geschachtelten Schleifen auch andere Basic-Befehle stehen. Was geschieht nun in diesem Programm?

In Zeile 20 beginnt die erste Schleife mit I=20. Zeile 30 gibt den aktuellen Wert von I aus. In Zeile 40 beginnt nun die zweite Schleife, deren NEXT in Zeile 50 zu finden ist. Diese zweite Schleife wird erst komplett abgearbeitet, bevor die erste Schleife ihren zweiten Durchlauf startet. Die zweite Schleife wird also insgesamt genauso oft durchlaufen, wie I ausgegeben wird.

Auf einen wichtigen Umstand müssen Sie allerdings bei der Verschachtelung achten. Sie dürfen keine Schleifen kreuzen, d.h. daß die zuerst geöffnete Schleife als letzte geschlossen und die zuletzt geöffnete zuerst geschlossen werden muß. Das Programm auf der vorigen Seite zeigt die richtige Verschachtelung der Schleifen. Das folgende Beispiel soll kein Programm darstellen, sondern soll nur aufzeigen, wie die Schleifen nicht verschachtelt werden dürfen.

F A L S C H

```
10 FOR I=1 TO 20  ------+
20 PRINT I              +
30 FOR Z=1 TO 10  ---+  +
40 PRINT Z           +  +
50 NEXT I       -----+--+
60 PRINT I,Z         +
70 NEXT Z   ---------+
```

F A L S C H

Haben Sie mehrere Schleifen miteinander verschachtelt, und wollen diese auf einmal schließen, so brauchen Sie nicht für jedes FOR ein spezielles NEXT zu benutzen. Es reicht ein NEXT, dem die einzelnen Laufvariablen in der richtigen Reihenfolge angehängt werden. Die Variablen müssen durch Kommas getrennt werden. Das folgende Beispiel soll das wieder verdeutlichen.

```
10 FOR I=1 TO 10
20 FOR Z=1 TO 10
30 PRINT I;Z
40 NEXT Z,I
50 END
```

Um den Funktionsablauf verschachtelter Schleifen noch einmal zu veranschaulichen, geben Sie dieses Programm in den Rechner ein und starten Sie es. In Zeile 40 wurde nur ein NEXT benutzt, um beide Schleifen zu schließen. Auch hier muß wieder die Regel beachtet werden, daß die zuletzt geöffnete Schleife zuerst geschlossen wird. Deswegen folgt dem NEXT zuerst die Variable Z und dann erst I.

Ein Fehler, der von Anfängern oft gemacht wird, ist das Hineinspringen in eine Schleife, d.h. die Schleife wird nicht über die FOR..TO-Anweisung angesprungen, sondern irgendwo zwischen FOR und NEXT. Da eine Schleife in den meisten Fällen mehrere Programmzeilen enthält, kann es vorkommen, daß man eine Zeile innerhalb der Schleife anspringt, weil es ja gerade so gut auskommt. Den Fehler merkt man meistens erst dann, wenn das Programm zum ersten Mal gestartet wird. Das Ergebnis ist dann ein Programmabbruch mit folgender Fehlermeldung:

```
?NEXT WITHOUT FOR ERROR IN (Zeilennummer)
```

Hat man sich vorher einen ausführlichen PAP erstellt, so kommen solche Fehler in aller Regel nicht mehr vor. Weiterhin ist darauf zu achten, daß, bei Angabe eines größeren Startwertes als des Endwertes, eine negative Schrittweite mit angegeben wird. Vergessen Sie die Angabe der Schrittweite, so wird die Schleife nur einmal

durchlaufen, wie folgendes Beispiel zeigt.

```
10 FOR A=5 TO 1
20 PRINT A
30 NEXT A
40 END
```

In Zeile 10 wurde die Angabe der Schrittweite, z.B. STEP -1, vergessen. Somit erhält man als Ausgabe für A nur den Wert 5. Will man eine Schleife vorzeitig beenden, so kann man das durch das Hochsetzen der Laufvariablen. Dieses Hochsetzen kann in Abhängigkeit von bestimmten Variablen geschehen oder sonstigen Bedingungen, die innerhalb des Programms abgefragt werden können. Normalerweise werden allerdings der Anfangswert und der Endwert in Variablen abgelegt. Ändern diese Variablen ihre Werte innerhalb des Programms, so hat man schon unterschiedliche Schleifenlängen erreicht.

Im Anschluß hieran finden Sie zuerst ein Programm, welches durch Hochsetzen der Laufvariablen die Schleife vorzeitig abbricht und dann ein Programm, welches die unterschiedlichen Schleifenlängen durch Variablen bestimmt. Die Werte der Variablen werden durch die Stringfunktionen bestimmt. Solche Anwendungen findet man häufig bei Dateiverwaltungen, um z.B. nach bestimmten Zeichenkombinationen suchen zu lassen.

```
10 REM HOCHSETZEN DER LAUFVARIABLEN
20 FOR A=0 TO 20
30 PRINT A
40 IF A=12 THEN A=20
50 NEXT A
60 END
```

Das Programm ergibt in dieser Kürze selbstverständlich keinen Sinn. Es soll auch nur aufzeigen, auf welche Art und Weise man eine Schleife vorzeitig verlassen kann. Normalerweise würde die Schleife bis zur 20 hochzählen. In Zeile 40 wird aber beim Erreichen für A=12 der Wert für A auf 20 gesetzt. Dadurch werden nur die Werte bis 12 ausgegeben. Diese Methode, die Laufvariable hochzusetzen,

wird aber nur selten angewendet. Viel häufiger werden der Anfangswert und der Endwert einer Schleife in Variablen abgelegt. Dadurch lassen sich dann die Schleifen besser beeinflussen. Das folgende Beispiel soll das veranschaulichen.

```
10 INPUT"GEBEN SIE EIN WORT EIN";A$
20 FOR A=1 TO LEN(A$)
30 PRINT LEFT$(A$,A)
40 NEXT A
50 FOR A=LEN(A$) TO 1 STEP -1
60 PRINT RIGHT$(A$,A)
70 NEXT A
80 END
```

Starten Sie das Programm, geben Sie Ihren Namen ein und drücken Sie erneut die RETURN-Taste. Sie sehen, daß wir die gleiche Ausgabe erhalten, wie es bei einem Programm im Kapitel über die Stringfunktionen der Fall war. Nur haben wir hier die FOR...NEXT-Schleife benutzt und sie abhängig von der Länge des eingegebenen Strings gemacht. Das bedeutet, daß die Schleifendurchläufe durch die Stringlänge gesteuert werden. Schauen Sie sich das Beispiel nochmal genau an und versuchen Sie es bis ins Letzte nachzuvollziehen.

Wir haben nun sehr viel über die FOR...NEXT-Schleifen erfahren. Fassen wir das Wichtigste über diese Schleifen noch einmal zusammen.

1. Zu jeder FOR-Anweisung gehört genau eine NEXT-Anweisung. Eine NEXT-Anweisung kann mehrere verschachtelte Schleifen abschließen, wenn dieser NEXT-Anweisung die Laufvariablen in richtiger Reihenfolge und durch Komma getrennt folgen.

2. Es darf nie in eine Schleife hineingesprungen werden, da das Programm sonst mit einer Fehlermeldung abbricht.

3. Der Anfangswert darf bei positiver Schrittweite nicht größer als der Endwert sein, da sonst die Schleife nur einmal durchlaufen wird. Das gleiche gilt entsprechend für negative Schrittweiten.

4. Allgemein wird eine FOR...NEXT-Schleife solange durchlaufen, bis der Wert der Laufvariablen größer als der Endwert ist.

Diese Regeln beziehen sich nur auf das BASIC des COMMODORE 64. Man darf Sie nicht verallgemeinern. Es gibt bei den verschiedenen BASIC-Dialekten gerade in der Benutzung der FOR...NEXT-Schleifen einige Unterschiede.

Allgemein kann man folgende Regeln für die Benutzung von Schleifen aufstellen:

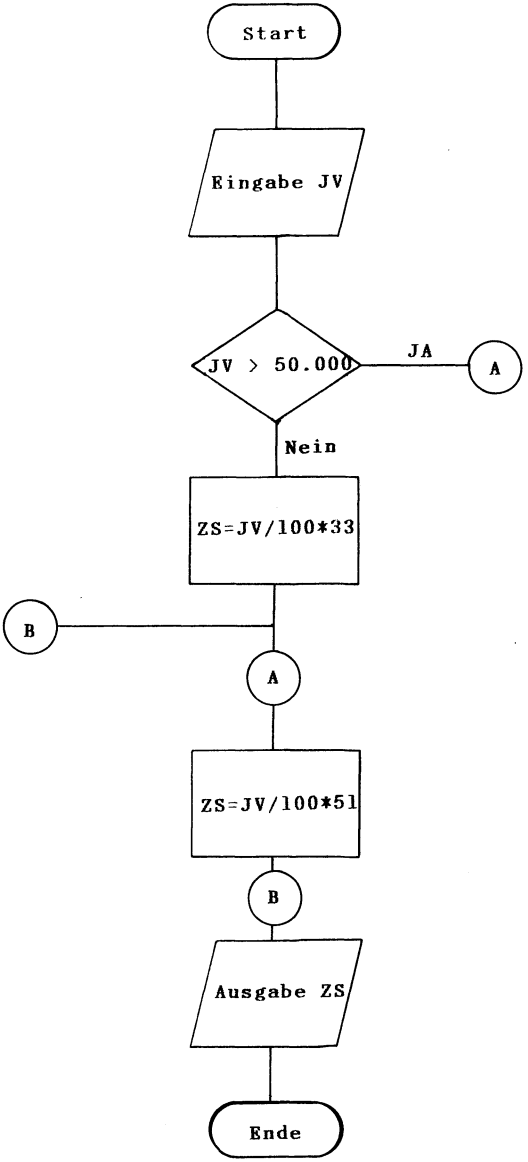
1. Steht die Anzahl der Wiederholungen der Schleifen von Anfang an fest, verwenden wir die FOR...NEXT-Schleife.

2. Ist die Anzahl der Wiederholungen der Schleifen unbekannt, so verwenden wir die Schleifen mit IF...THEN.

Auch hierzu haben wir schon eine Ausnahme im letzten Beispielprogramm kennengelernt. Diese Regeln sollen ja auch nur ein allgemeiner Anhaltspunkt sein.

Bisher haben wir nur über die Anwendung und Programmierung von bedingten und unbedingten Programmsprüngen etwas erfahren. Weiterhin wissen wir, wie man mit Programmschleifen, insbesondere mit den FOR...NEXT-Schleifen, umzugehen hat. Was noch fehlt, ist die Darstellung dieser Programmstrukturen in den Programmablaufplänen. Das Symbol, das zur Kennzeichnung einer logischen Verzweigung in PAPs gebraucht wird, ist die Raute. Wir wollen uns zuerst einen PAP für das Programm mit der Berechnung des Steuersatzes anschauen (siehe Seite 86). Auf den nächsten beiden Seiten folgen nun der Programmablaufplan sowie die dazugehörigen Erklärungen.

Programmablaufplan zum Programm "Berechnung Steuersatz"

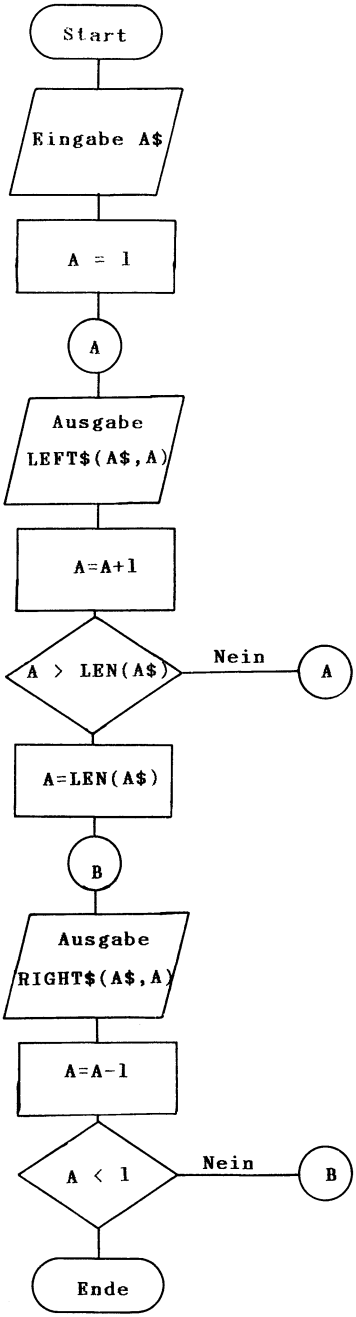


Das Startsymbol sowie das Eingabesymbol in unserem PAP dürften hinreichend bekannt sein. Die Raute ist, wie bereits erwähnt, das Symbol für eine logische Verzweigung. Sie besitzt einen sogenannten JA-Arm und einen NEIN-Arm. Trifft die Bedingung zu, so wird über den Absprungkonnektor A nach dem dazugehörigen Einsprungkonnektor A verzweigt. In unserem Beispiel hätte die Verzweigung über den JA-Arm stattgefunden. Je nach Art des Programms kann dieser Arm auch der NEIN-Arm sein. Nach dem Einsprungkonnektor A erfolgt die Berechnung des Zinssatzes von 51 Prozent mit anschließender Ausgabe des Wertes.

Trifft die Bedingung nicht zu, so werden die 33 Prozent berechnet. Nach der Berechnung kommt im PAP der Absprungkonnektor B. Er kennzeichnet hier einen unbedingten Programmsprung zum Einsprungkonnektor B. Zu beachten ist, daß der Absprungkonnektor B vor den Einsprungkonnektor A zu liegen kommt, da sonst ein logischer Fehler im PAP entstehen würde. Soweit der Programmablaufplan zu diesem Programm.

Dieser PAP hat aufgezeigt, wie der IF...THEN-Befehl in einem PAP dargestellt wird. Was wir nun noch wissen müssen, ist die Darstellungsweise einer FOR...NEXT-Schleife in einem PAP. Dazu wollen wir das Beispielprogramm von Seite 96 in einen PAP übertragen. Auf den nächsten beiden Seiten folgt wiederum der Programmablaufplan mit den dazugehörigen Erläuterungen.

Programmablaufplan zum Programm auf Seite 96



Die Symbole in diesem PAP müßten Ihnen nun von der Anwendung her geläufig sein. Im ersten Rechteck wird der Anfangswert der Schleife auf eins gesetzt. Danach erfolgt die Ausgabe eines Teilstrings mit LEFT$(A$,A). Anschließend wird der Zähler um eins erhöht. In der Raute wird abgefragt, ob der Zähler bereits größer als die Anzahl der Zeichen von A$ ist. Ist das nicht der Fall, wird über den Absprungkonnektor A zum Einsprungkonnektor A verzweigt. Die Schleife ist somit im PAP erstellt.

Wird der Zähler nun größer als LEN(A$), so tritt die zweite Schleife in Aktion. Die zweite Schleife besitzt die gleiche Anordnung der Symbole, wie die erste Schleife. Die Konnektoren erhielten allerdings andere Bezeichnungen, um Verwechslungen auszuschließen. Der Ablauf ist jedoch der gleiche wie oben bei der ersten Schleife beschrieben.
Mit diesen Informationen sollten Sie nun in der Lage sein, selbständig Programmablaufpläne für jedes Programm zu erstellen. Auch hier gilt der Spruch "Übung macht den Meister".

Damit haben wir das Thema Programmstrukturen fast durchgearbeitet. Es fehlen nur noch die berechneten Sprungbefehle. Damit befaßt sich das nächste Kapitel.

3.3 BERECHNETE SPRUNGBEFEHLE

Die berechneten Sprungbefehle haben den Vorteil, daß durch sie das eigentliche Programm flexibler ablaufen kann. Wir kennen bisher nur die Sprungbefehle, die genau in eine bestimmte Programmzeile springen. Die Zeilennummer in einem GOTO-Befehl kann also nicht beeinflußt werden, d.h. mit GOTO 100 springt das Programm grundsätzlich in die Programmzeile 100. Nun wäre es aber wünschenswert, wenn man zu Beginn eines Programms einen Wert eingeben könnte, aufgrund dessen das Programm in bestimmte Programmzeilen verzweigen kann. Sicherlich könnte man das mit einigen IF...THEN-Vergleichen erreichen, jedoch ist dazu für jeden Vergleich ein

Sprungbefehl für jede Programmzeile notwendig. Ein einfaches Beispiel soll das verdeutlichen.

```
10 REM SPRUNG IN BESTIMMTE ZEILEN
20 PRINT"GEBEN SIE EINE ZAHL ZWISCHEN";
30 PRINT CHR$(17) "1 UND 4 EIN"
40 PRINT
50 INPUT"WELCHE ZAHL" Z
60 IF Z = 1 THEN 100
70 IF Z = 2 THEN 200
80 IF Z = 3 THEN 300
90 IF Z = 4 THEN 400
100 PRINT"SPRUNG NACH ZEILE 100"
110 GOTO 410
200 PRINT"SPRUNG NACH ZEILE 200"
210 GOTO 410
300 PRINT"SPRUNG NACH ZEILE 300"
310 GOTO 410
400 PRINT"SPRUNG NACH ZEILE 400"
410 END
```

In diesem Programm wird je nach Eingabe der Zahlen 1 bis 4 in die entsprechenden Programmzeilen 100 bis 400 verzweigt. Die Programmierung dieser Abfragen mit IF...THEN ist für solche Anwendungen jedoch recht umständlich und von der Ausführung her relativ langsam. Basic bietet nun für solche Fälle eine bequemere Lösung an. Der Befehl hat folgende Schreibweise:

ON (VARIABLE) GOTO (ZEILENNUMMER)

Dieser erweiterte GOTO-Befehl mit ON veranlaßt, daß das Programm zu einer von mehreren Zeilennummern, die dem GOTO folgen, verzweigt. Der Bereich der Variablen reicht von Null bis zur Anzahl der angegebenen Zeilennummern. Besitzt die Variable keinen ganzzahligen Wert, so bleiben die Nachkommastellen unberücksichtigt. Negative Werte der Variablen verursachen die Fehlermeldung

```
?ILLEGAL QUANTITY ERROR
```

Hat die Variable einen Wert, der größer ist als die

Anzahl der zur Verfügung stehenden Zeilennummern, die dem GOTO folgen, so wird der Befehl, der dem ON...GOTO-Befehl folgt, ausgeführt. Hierzu wollen wir uns einige einfache Beispiele zur Verdeutlichung anschauen.

BEISPIELE:

a)
```
10 ON Z GOTO 100,200,250,300
20 PRINT
 .
 .
 .
```

Hat die Variable Z in diesem Beispiel den Wert 1, so springt das Programm in die Zeile 100. Durchläuft Z danach die Werte 2 bis 4, z.B. in einer Schleife, so springt das Programm nacheinander in die Zeilen 200, 250 und 300. Z bezeichnet somit die Positionen der einzelnen Zeilennummern, die dem GOTO folgen. Nimmt Z Werte an, die kleiner oder größer als die Anzahl der Zeilennummern hinter dem GOTO sind, so fährt das Programm mit dem nächsten Befehl, der dem GOTO folgt, fort. Das wäre in unserem Beipiel der PRINT-Befehl. Der ON...GOTO-Befehl wird in diesem Falle einfach überlesen.

b)
```
10 ON Z+3/4 GOTO 100,200,300
20 PRINT
 .
 .
 .
```

Sie sehen, statt einer Variablen kann auch ein arithmetischer Ausdruck Verwendung finden. Der Vorteil dieses ON...GOTO-Befehls ist der, daß durch ihn mehrere IF...THEN-Befehle ersetzt werden können. Damit spart man Programmierarbeit und auch Speicherplatz. Unser kleines Programm von vorhin hätte dann die folgende Form:

```
10 REM SPRUNG MIT ON...GOTO
20 PRINT "GEBEN SIE EINE ZAHL ZWISCHEN";
30 PRINT CHR$(17) "1 UND 4 EIN"
40 PRINT
50 INPUT "WELCHE ZAHL" Z
60 ON Z GOTO 100,200,300,400
100 PRINT "SPRUNG NACH ZEILE 100"
110 GOTO 410
200 PRINT "SPRUNG NACH ZEILE 200"
210 GOTO 410
300 PRINT "SPRUNG NACH ZEILE 300
310 GOTO 410
400 PRINT "SPRUNG NACH ZEILE 400
410 END
```

Wir haben also bei diesem kleinen Programm schon 3 Programmzeilen eingespart. Bei größeren Programmen, wo mehrere Vergleiche mit IF...THEN auftauchen können, spart man teilweise noch mehr Zeilen ein.

In diesem Programm wurde gleichzeitig eine Programmiertechnik verwendet, die sich in der Praxis, vor allem bei größeren Programmen, als sehr hilfreich erwiesen hat. Erstellen Sie zu diesem Programm einen Programmablaufplan, so werden die Sprünge in die verschiedenen Zeilen durch waagerechte Verzweigungen symbolisiert. Da man zu diesem Zeitpunkt aber noch nicht genau wissen kann, welche Zeilennummern diese Zeilen bekommen, wählt man extra große Nummern. Damit schafft man sich innerhalb des Programms Platz für andere Programmteile.

Die mit ON...GOTO angesprungenen Zeilennummern stellen innerhalb des Programms bestimmte Programmteile dar, in denen meistens spezielle Aufgaben übernommen werden. Daher ist es von Vorteil, sie durch "glatte" Zeilennummern zu kennzeichnen. Das kann z.B. in Hunderterschritten geschehen, wie in unserem Beispielprogramm. Dadurch erreicht man eine gewisse Übersichtlichkeit der einzelnen Programmabschnitte. Das Programm wird leichter lesbar.

3.3.1 BEISPIELPROGRAMM "RECHENLEHRGANG"

Wir haben zum jetzigen Zeitpunkt schon einen relativ großen Befehlsumfang von Basic kennengelernt. Das ist ein Grund, um sich an ein größeres Projekt zu wagen. Stellen Sie sich vor, Sie wollen für Ihre Kinder einen Rechenlehrgang für die vier Grundrechenarten auf dem COMMODORE 64 realisieren. Dieser Lehrgang soll folgende Eigenschaften aufweisen:

1. Auswahl einer der vier Grundrechenarten oder Programmende
2. Stellen einer Aufgabe, die in höchstens drei Versuchen gelöst werden muß.
3. Nach dem dritten Fehlversuch soll das Ergebnis angezeigt werden.
4. Eingabe der größten Zahl, mit der in den einzelnen Aufgaben gerechnet werden soll.
5. Nach jeder Aufgabe soll eine Abfrage erfolgen, ob weitere Aufgaben der gleichen Rechenart gelöst werden sollen.

Im folgenden erhalten Sie das Programmlisting dieses Rechenlehrgangs. Stören Sie sich nicht daran, wenn einzelne Zeilen nicht immer genau in Zehnerschritten voneinander getrennt sind. Da das Programm relativ lang ist, will ich Ihnen jetzt schon die Befehle zum Abspeichern des Programms nennen, obwohl wir diese noch nicht besprochen haben. Benutzen Sie die DATASETTE, so geben Sie folgenden Befehl in den Rechner, nachdem Sie das Programm eingegeben haben.

SAVE"RECHENLEHRGANG"

Danach betätigen Sie die RETURN-Taste. Es erfolgt die Meldung

PRESS RECORD & PLAY ON TAPE

Drücken Sie nun die beiden Tasten auf der DATASETTE und das Programm wird abgespeichert.

Bei Benutzung des Diskettenlaufwerks ist die Handhabung einfacher. Nachdem Sie eine formatierte Diskette in das Laufwerk gelegt haben, geben Sie in den Rechner folgenden Befehl ein:

SAVE"RECHENLEHRGANG",8

Das Programm wird nun automatisch auf die Diskette abgespeichert. Soweit die "Datensicherung" für dieses Programm. Sollten Sie es nochmal benötigen, so brauchen Sie es nur vom entsprechenden Speichermedium abzurufen. Das geschieht durch den LOAD-Befehl. Ersetzen Sie einfach an den entsprechenden Stellen SAVE durch LOAD, und das Programm wird in den Rechner geladen.

Im folgenden nun das Programmlisting für den RECHENLEHRGANG.

```
5 REM ***** MENUE *****
10 PRINT CHR$(147):F=0
20 PRINT
30 PRINT TAB(12)"RECHENLEHRGANG"
40 PRINT:PRINT
50 PRINT TAB(12)"WAEHLEN SIE:"
60 PRINT
70 PRINT TAB(12)"FUER ADDITION EINE 1"
80 PRINT
90 PRINT TAB(12)"FUER SUBTRAKTION EINE 2"
100 PRINT
110 PRINT TAB(12)"FUER DIVISION EINE 3"
120 PRINT
130 PRINT TAB(12)"FUER MULTIPLIKATION EINE 4"
133 PRINT
135 PRINT TAB(12)"FUER ENDE EINE 5"
140 PRINT
150 PRINT TAB(12);:INPUT"WELCHE ZAHL";Z
160 IF Z < 1 OR Z > 5 THEN 10
170 ON Z GOTO 200,600,1000,1300,1600
200 REM **********
210 REM  ADDITION
220 REM **********
```

```
230 PRINT CHR$(147)
240 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
250 PRINT
260 PRINT TAB(10)"FUER DIE ADDITION EIN."
270 PRINT
290 PRINT TAB(10);:INPUT"GROESSTE";GR
299 REM
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM BERECHNUNG ERGEBNIS
331 REM
340 ER=A1+A2
350 PRINT CHR$(147)
360 PRINT
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2 "= ";
380 INPUT ES
390 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:
GOTO450
400 PRINT:PRINTTAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FORI=0 TO 2000:NEXTI
460 PRINTTAB(5)"DAS ERGEBNIS LAUTET" ER
470 FORI=0 TO 3000:NEXTI
480 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
490 INPUT A$
500 IF A$="J" THEN F=0:GOTO 300
510 GOTO 10
600 REM *************
610 REM  SUBTRAKTION
620 REM *************
630 PRINT CHR$(147)
640 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
650 PRINT
660 PRINT TAB(10)"FUER DIE SUBTRAKTION EIN."
670 PRINT
690 PRINT TAB(10);:INPUT"GROESSTE";GR
```

```
699 REM
700 REM ERZEUGEN ZUFALLSZAHLEN
701 REM
710 A1=INT(GR*RND(1))+1
720 A2=INT(GR*RND(1))+1
729 REM
730 REM BERECHNUNG ERGEBNIS
731 REM
740 IF A1 < A2 THEN I = A1:A1=A2:A2=I
750 PRINT CHR$(147)
760 PRINT
770 ER=A1-A2
780 PRINT"WIEVIEL ERGIBT" A1 "-" A2 "= ";
790 INPUT ES
800 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:
GOTO860
810 PRINT:PRINTTAB(10)"FALSCH !"
820 FOR I=0 TO 2000:NEXT I
830 F=F+1
840 IF F<=2 THEN 750
850 PRINT
860 FORI=0 TO 2000:NEXTI
870 PRINTTAB(5)"DAS ERGEBNIS LAUTET" ER
880 FORI=0 TO 3000:NEXTI
890 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
900 INPUT A$
910 IF A$="J" THEN F=0:GOTO 710
920 GOTO 10
1000 REM **********
1001 REM  DIVISION
1002 REM **********
1010 PRINT CHR$(147)
1020 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
1030 PRINT
1040 PRINT TAB(10)"FUER DIE DIVISION EIN."
1050 PRINT
1070 PRINT TAB(10);:INPUT"GROESSTE";GR
1079 REM
1080 REM ERZEUGEN ZUFALLSZAHLEN
1081 REM
1090 A1=INT(GR*RND(1))+1
1100 A2=INT(GR*RND(1))+1
```

```
1109 REM
1110 REM BERECHNUNG ERGEBNIS
1111 REM
1120 ER=A1*A2
1130 PRINT CHR$(147)
1140 PRINT
1150 PRINT"WIEVIEL ERGIBT" ER "/" A1 "= ";
1160 INPUT ES
1170 IF ES=A2 THEN PRINT:PRINT TAB(10)"RICHTIG !":
F=0:GOTO 1240
1180 PRINT:PRINTTAB(10)"FALSCH !"
1190 FOR I=0 TO 2000:NEXT I
1200 F=F+1
1210 IF F<=2 THEN 1130
1220 PRINT
1230 FORI=0 TO 2000:NEXTI
1240 PRINTTAB(5)"DAS ERGEBNIS LAUTET" A2
1250 FORI=0 TO 3000:NEXTI
1260 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
1270 INPUT A$
1280 IF A$="J" THEN F=0:GOTO 1090
1290 GOTO 10
1300 REM ****************
1301 REM  MULTIPLIKATION
1302 REM ****************
1310 PRINT CHR$(147)
1320 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
1330 PRINT
1340 PRINT TAB(10)"FUER DIE MULTIPLIKATION EIN."
1350 PRINT
1370 PRINT TAB(10);:INPUT"GROESSTE";GR
1379 REM
1380 REM ERZEUGEN ZUFALLSZAHLEN
1381 REM
1390 A1=INT(GR*RND(1))+1
1400 A2=INT(GR*RND(1))+1
1409 REM
1410 REM BERECHNUNG ERGEBNIS
1411 REM
1420 ER=A1*A2
1430 PRINT CHR$(147)
1440 PRINT
```

```
1450 PRINT"WIEVIEL ERGIBT" A1 "*" A2 "= ";
1460 INPUT ES
1470 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":
F=0:GOTO 1540
1480 PRINT:PRINTTAB(10)"FALSCH !"
1490 FOR I=0 TO 2000:NEXT I
1500 F=F+1
1510 IF F<=2 THEN 1430
1520 PRINT
1530 FORI=0 TO 2000:NEXTI
1540 PRINTTAB(5)"DAS ERGEBNIS LAUTET" ER
1550 FORI=0 TO 3000:NEXTI
1560 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
1570 INPUT A$
1580 IF A$="J" THEN F=0:GOTO 1390
1590 GOTO 10
1600 PRINT CHR$(147)
1610 END
```

Wir wollen nun die wichtigsten Programmteile dieses Listings besprechen. In den Zeilen 10 bis 150 wird das sogenannte MENÜ des Programms aufgebaut. Unter einem Menü versteht man eine Auswahl von verschiedenen Programmpunkten, aus denen der Anwender sich einen Punkt auswählen kann. Jedes gute Programm sollte mindestens ein Menü vorweisen können.

Die Zeile 150 fordert nun zur Eingabe einer Zahl auf, welche jeweils einen Menüpunkt repräsentiert. In Zeile 160 wird überprüft, ob eine zulässige Zahl eingegeben wurde. Hier haben wir ein schönes Beispiel für die Anwendung eines logischen Operators. Wird entweder eine Zahl kleiner als 1 oder eine Zahl größer als 5 eingegeben, so wird nach Zeile 10 verzweigt. Es braucht nur eine dieser Bedingungen erfüllt zu sein, daher auch die Verknüpfung mit OR. Zeile 170 zeigt uns nun die Anwendung des neuen ON...GOTO-Befehls. Nimmt Z den Wert 1 an, so wird nach Zeile 200 verzweigt. Hat Z den Wert 2, so springt das Programm nach Zeile 600 usw. Die einzelnen Rechenarten werden also durch die Eingabe einer Zahl angewählt. Mit dem ON...GOTO-Befehl haben wir hier 5

IF...THEN Vergleiche eingespart.

Die Zeilen 200 bis 220 trennen den Programmteil für die Addition optisch ab. Für den Programmierer ist es zusätzlich noch eine Gedächtnisstütze. In größeren Programmen lernt man solche Abtrennungen mit REM schätzen, da durch sie das Programm übersichtlicher wird. Man braucht nicht umständlich herumzusuchen, wenn man z.B. im Programmteil "Addition" einen Fehler beseitigen möchte.

Zeile 230 löscht zunächst den Bildschirm. Die nächsten Zeilen fordern den Anwender auf, eine Zahl einzugeben, die die obere Grenze der Summanden für die Addition festlegt. Danach werden zwei Zufallszahlen A1 und A2 erzeugt, aus denen dann die Additionsaufgabe gebildet wird. In Zeile 370 wird dann die Aufgabe auf dem Bildschirm ausgegeben. Die Semikolons zwischen Text und Variablen sind nicht unbedingt notwendig, wie Sie an diesem Beispiel sehen können. In Zeile 380 wird vom Anwender das Ergebnis übernommen. Zeile 390 überprüft, ob der eingegebene Wert mit dem vorher berechneten Wert in Zeile 340 übereinstimmt. Ist das eingegebene Ergebnis falsch, wird durch die Zeile 400 erst eine Leerzeile auf dem Bildschirm erzeugt. Danach erfolgt die Ausgabe der Meldung "FALSCH !". In 410 wird eine Zeitschleife abgearbeitet, damit der Anwender die Meldung lesen kann. In Zeile 420 wurde ein Zähler gesetzt, der überprüft, wie viele Falschantworten der Anwender bei dieser Aufgabe bereits gegeben hat. Das Programm soll ja nach drei falschen Antworten das Ergebnis anzeigen. Zeile 430 überprüft, ob bereits drei falsche Antworten gegeben wurden. Ist dies nicht der Fall, verzweigt das Programm nach Zeile 350 und stellt die Aufgabe erneut. Wurden drei falsche Antworten eingegeben, so fährt das Programm mit der Ausführung in Zeile 440 fort. Wurde inzwischen das richtige Ergebnis eingegeben, so springt das Programm von Zeile 390 in die Zeile 450. Nachdem die Zeitschleife in Zeile 450 durchlaufen ist, wird in Zeile 460 das Ergebnis ausgegeben. Nach einer erneuten Zeitschleife fragt das Programm in Zeile 480 nach, ob eine weitere Aufgabe gestellt werden soll. Gibt der Anwender hier ein "J" ein,

so wird zuerst der Zähler "F" auf Null gesetzt und anschließend nach Zeile 300 verzweigt. Dort werden zwei neue Zufallszahlen gebildet, die für die neue Aufgabe benötigt werden. Betätigt der Anwender nicht die J-Taste, sondern irgendeine andere, so springt das Programm zurück nach Zeile 10, wo das Menü wieder aufgebaut wird.

Der Zähler "F" muß deshalb auf Null zurückgesetzt werden, da für die neue Aufgabe ja ebenfalls drei Versuche zur Verfügung stehen sollen. Wird das vergessen, so würde der alte Wert von "F" mitgeschleppt, und dann kann es passieren, daß schon nach zwei oder nach einer falschen Antwort das Ergebnis ausgegeben wird. Achten Sie also darauf, wenn Sie solche Zähler bei Ihren eigenen Programmen verwenden.

Die anderen Teile des Programms, "Subtraktion", "Division" und "Multiplikation", sind im Prinzip gleich aufgebaut. Sie unterscheiden sich jedoch geringfügig bei der Erzeugung der Aufgaben. Sehen wir uns zunächst die Subtraktion an.

Bei der Berechnung des Ergebnisses fällt die Zeile 740 auf. Damit wir nur positive Ergebnisse erhalten, dürfen wir nur kleinere von größeren Zahlen subtrahieren. Ist A1 größer als A2, so ist die Aufgabenstellung in Zeile 780 korrekt. Tritt jedoch genau der umgekehrte Fall ein, müssen die Werte der Variablen vertauscht werden, da sonst in Zeile 780 eine größere von einer kleineren Zahl subtrahiert wird. Genau dazu wurde die Zeile 740 eingeführt.

Ist A1 nun kleiner als A2, so wird der Wert von A1 in I zwischengespeichert. Würden wir folgende Programmierung vornehmen:

```
A1=A2 : A2=A1
```

so ginge der Wert von A1 verloren. Da zuerst A1 gleich A2 gesetzt wird, beinhaltet die Variable A1 jetzt den Wert von A2. Danach versucht man zwar, A2 gleich A1 zu setzen, aber A1 hat ja bereits den Wert von A2. Auf diese Art und

Weise schafft man es also nicht, zwei Variablen zu vertauschen. Daher muß man einen Wert zuerst "retten", d.h. in einer Variablen zwischenspeichern.

Zuerst bekommt die Variable I den Wert von A1, also I=A1. Danach wird der Variablen A1 der Wert von A2 übertragen, also A1=A2. Jetzt braucht man nur noch A2=I zu setzen, da ja I den Wert von A1 hat, und schon hat die Vertauschung stattgefunden. Diese Technik der Zwischenspeicherung ist sehr wichtig. Überzeugen Sie sich daher davon, daß Sie das Prinzip verstanden haben, um es später in eigenen Programmen anwenden zu können.

Dieser Punkt war das eigentlich Besondere im Programmteil der Subtraktion. Beim Programmteil "Division" wurde auch ein kleiner Kniff verwendet, um nur ganzzahlige Ergebnisse zu erhalten. In Zeile 1120 wird zuerst, wie bei der Multiplikation, das Ergebnis von A1 und A2 gebildet. Dann wird in der Aufgabenstellung das Ergebnis ER durch den Wert von A1 dividiert. Die Antwort kann nur eine ganzzahlige Zahl sein, da das Ergebnis ja durch zwei ganze Zahlen gebildet wurde, nämlich A1 und A2. Das wäre soweit zu diesem Programmteil zu sagen.

Der Teil der Multiplikation beinhaltet keine Besonderheiten. Er ist vom Aufbau her identisch mit dem Programmteil der Addition.

Damit haben wir die wichtigsten Eigenschaften dieses Programms besprochen und gleichzeitig eine praxisnahe Anwendung des ON...GOTO-Befehls gesehen. Bevor ich Ihnen nun wieder einige Aufgaben stelle, in denen Sie Ihr neu erworbenes Wissen selbst überprüfen können, fassen wir die wichtigsten Punkte des ON...GOTO-Befehls nochmal zusammen.

Folgendes ist bei der Anwendung des ON...GOTO-Befehls zu beachten:

1. Der Wert, der dem ON folgt - dies kann eine Zahl, eine Variable oder ein arithmetischer Ausdruck sein - bestimmt die Position der Zeilennummer in der Liste, die dem GOTO folgt. Für 1 wird die erste Zeilenummer, für 2 die zweite Zeilennummer usw. gelesen.

2. Ist dieser Wert größer oder kleiner als die Anzahl der in der Liste vorkommenden Zeilennummern, so wird der nächste Befehl, der dem GOTO folgt, ausgeführt.

3. Werte kleiner als Null oder größer als 255 bewirken einen Programmabbruch mit der Fehlermeldung: ?ILLEGAL QUANTITY ERROR IN (Zeilennummer).

4. Mit ON..GOTO können mehrere IF...THEN Vergleiche zusammengefaßt werden.

Auf der nächsten Seite habe ich nun wieder einige Aufgaben für Sie zusammengestellt, denen dann die Lösungen folgen. Berücksichtigen Sie auch diesmal wieder die auf Seite 18 f erwähnten fünf Punkte der Programmierung. Und nun wie gehabt viel Erfolg beim Lösen der Aufgaben.

1. Schreiben Sie ein Programm, welches Ihnen die "harmonische Reihe" (1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n) bis zu einem vorgegebenen Wert aufsummiert. Nach jeweils 50 Additionen soll die Anzahl der Additionen ausgegeben werden. (Also 50, 100, 150 usw.) Zum Schluß soll die Anzahl der benötigten Summanden mit ausgegeben werden.

2. Schreiben Sie ein Programm, welches Ihnen die reellen Nullstellen einer quadratischen Gleichung der Form: AX↑2+BX+C=0 berechnet. Die Lösungen der Aufgabe erhalten Sie durch folgende Formel:
   x1 = (-B + SQR(B↑2-4AC))/2A
   x2 = (-B - SQR(B↑2-4AC))/2A
   Für B↑2-4AC < 0 existieren keine reellen Lösungen. Berücksichtigen Sie das in Ihrem Programm.

3.
```
10 REM TESTAUFGABE MIT ON...GOTO
20 INPUT"GEBEN SIE EINE ZAHL EIN";Z
30 ON Z GOTO 100,150,400:PRINT CHR$(147)
40 PRINT"WERT NICHT ZULÄSSIG"
50 END
100 PRINT"ZEILE 100"
110 END
150 PRINT"ZEILE 150"
 .
 .
```
   Was geschieht in diesem Programm, wenn für Z der Wert 4 eingegeben wird? Lösen Sie die Aufgabe, ohne das Programm in den Rechner einzugeben.

1.
```
10 REM HARMONISCHE REIHE
20 PRINT CHR$(147)
30 PRINT "BIS ZU WELCHER SUMME SOLL"
40 PRINT:PRINT "ADDIERT WERDEN ?"
50 PRINT
60 INPUT S
70 Z=1
80 SH=SH + 1/Z
90 Z=Z+1
100 IF Z = 50 * INT(Z/50) THEN PRINT Z;"ADDITIONEN"
110 IF SH < S THEN 80
120 PRINT "NACH" Z "GLIEDERN IST DIE SUMME" SH
```

In den Zeilen 20 bis 60 wird der Bildschirm gelöscht und danach zur Eingabe der zu bildenden Summe aufgefordert. Zeile 70 setzt den Zähler auf 1 und in Zeile 80 wird die Summe aus den einzelnen Gliedern gebildet. Danach wird in Zeile 90 der Zähler um eins erhöht. Zeile 100 überprüft, ob der Zähler 50 oder ein Vielfaches von 50 erreicht hat. Es sollte jeweils nach 50 Gliedern eine entsprechende Ausgabe erfolgen. Mit dieser Technik können auch beliebige andere Vielfache überprüft werden. Der Wert 50 ist nur mit der zu überprüfenden Zahl auszutauschen. Hat der Zähler ein Vielfaches von 50, so wird der Befehl nach dem THEN ausgeführt. Zeile 110 prüft, ob die eingegebene Summe bereits erreicht wurde. Die Zeile 120 gibt schließlich nach Erreichen der Summe die benötigten Durchläufe sowie die gebildete Summe selbst aus.

2.
```
10 REM QUADRATISCHE GLEICHUNG
20 PRINT CHR$(147):PRINT
30 PRINT "EINGABE DER KOEFFIZIENTEN A,B,C"
40 PRINT
50 INPUT A,B,C
60 IF A=0 THEN 20:REM A MUSS <> 0 SEIN
70 D=B*B-4*A*C
80 IF D < 0 THEN 140
```

```
90 X1 =(-B+SQR(D))/(2*A)
100 X2 =(-B-SQR(D))/(2*A)
110 PRINT "LOESUNG FUER X1 =";X1
120 PRINT "LOESUNG FUER X2 =";X2
130 GOTO 150
140 PRINT "KEINE REELLEN NULLSTELLEN !!"
150 END
```

Die Umsetzung des Problems in das Programm dürfte keine Schwierigkeiten bereitet haben. Was zu beachten war, ist der Fall, für den A gleich Null wird. Es muß laut Formel mit 2*A dividiert werden. Da die Division durch Null bekanntlich nicht erlaubt ist, mußte dieser Fall am Anfang ausgeschlossen werden.

3. Ich hoffe Sie haben es richtig erkannt. Zuerst wird der Bildschirm gelöscht und dann erfolgt die Ausgabe "Wert nicht zulässig". Wichtig war hier, daß Sie erkennen mußten, daß der Befehl direkt hinter dem GOTO mit ausgeführt wird.

Nachdem Sie nun diese Aufgaben gelöst haben sowie die Lösungsvorschläge nochmal durchgearbeitet und mit Ihren verglichen haben, können Sie erst einmal eine Pause einlegen, bevor Sie zum nächsten Kapitel übergehen.

3.4 PROGRAMMABLAUFSTEUERUNG MIT GET

Bisher haben wir in unseren Programmen immer den INPUT-Befehl benutzt, um dem Programm Daten zu übergeben. Der INPUT-Befehl hat den Vorteil, daß er innerhalb eines Programms leicht zu gebrauchen ist und daß mit ihm gleichzeitig eine Textausgabe erfolgen kann. Der Nachteil dieses Befehls liegt darin, daß man durch Fehlbedienung entweder sein Menü zerstören kann oder versehentlich einen Buchstaben statt einer Zahl eingibt. Das hat dann zur Folge, daß die Meldung

?REDO FROM START

ausgegeben wird. Dadurch ist in den meisten Fällen das Menü ebenfalls hin. Weiterhin können Sie dem INPUT-Befehl kein Komma übergeben, da das Komma bei INPUT zur Unterscheidung mehrerer Variablen benötigt wird. Betätigen Sie aus Versehen die SHIFT/CLR HOME Tasten - das soll tatsächlich schon vorgekommen sein - so wird der komplette Bildschirm gelöscht. Damit soll nun nichts gegen den INPUT-Befehl gesagt sein. Für die meisten eigenen Anwendungen ist er vollkommen ausreichend. Will man jedoch seine Programme gegen eine versehentliche Fehlbedienung weitgehend absichern, so reicht der INPUT-Befehl nicht mehr aus. Bei großen Programmen findet deshalb der GET-Befehl Verwendung.

3.4.1 EINGABE VON DATEN MIT GET

GET gehört - wie übrigens INPUT auch - zu der Sorte von Befehlen, die nicht im Direktmodus benutzt werden können. Das bedeutet, daß sie nur innerhalb eines Programms eingesetzt werden können. Versuchen Sie dennoch, diese Befehle im Direktmodus einzusetzen, so gibt der Rechner die Meldung

?ILLEGAL DIRECT ERROR

aus. Wie arbeitet nun der GET-Befehl?

Dazu müssen wir zunächst wissen, daß der COMMODORE 64 einen sogenannten Tastaturpuffer besitzt. Es handelt sich dabei um einen kleinen Speicher, in dem bis zu 10 Zeichen zwischengespeichert werden können. Praktisch hat das folgende Bedeutung: Ist der Computer mit der Ausführung einer Operation beschäftigt, so können Sie während dieser Zeit maximal 10 Zeichen über die Tastatur eingeben, die der Computer dann sofort nach Beendigung der Operation benutzt. Geben Sie zur Verdeutlichung dieses Vorgangs das folgende "Programm" in den Rechner:

```
10 FOR I=0 TO 10000:NEXT I
```

Nachdem Sie diese Zeitschleife von ca. 10 Sekunden gestartet haben, betätigen Sie bitte nacheinander die Tasten T E S T. Nach Durchlaufen der Zeitschleife werden die Zeichen TEST auf dem Bildschirm angezeigt. Sie wurden also während der Operation im Tastaturpuffer zwischengespeichert, nach Ausführung von dort abgerufen und auf dem Bildschirm ausgegeben.

Der GET-Befehl überprüft nun, ob im Tastaturpuffer Zeichen vorhanden sind. Ist das der Fall, so wird durch GET das erste Zeichen ausgelesen und der Variablen, die dem GET folgt, zugeordnet. Ist der Puffer leer, so wird der Variablen der Wert Null zugeordnet. Der GET-Befehl arbeitet andauernd, d.h. daß er kurz überprüft, ob Zeichen im Tastaturpuffer vorhanden sind, und dann mit der Programmausführung fortfährt. Damit kann man nun noch keine vernünftige Datenübernahme gewährleisten. Daher findet man beim GET-Befehl meistens eine Schleife, die überprüft, ob ein Zeichen eingegeben wurde. Wurde kein Zeichen eingegeben, so wird die Zeile mit GET erneut abgearbeitet. Zu dieser Anwendung wieder ein Beispiel:

```
10 GET A$:IF A$ = "" THEN 10
20 PRINT A$
```

Geben Sie das Beispiel wieder in den Computer ein und starten Sie es. Beachten Sie, daß bei GET, im Gegensatz

zu INPUT, kein Cursor erscheint. Dieses kleine Programm wartet solange, bis Sie irgendeine Taste drücken. In Zeile 10 wird mit GET versucht, ein Zeichen aus dem Tastaturpuffer einzulesen. Der IF...THEN Vergleich überprüft, ob es sich bei A$ um einen Leerstring handelt, d.h. ob dieser String NICHTS beinhaltet. Man erreicht das durch die beiden Anführungszeichen. Achten Sie darauf, daß zwischen den Anführungszeichen nichts steht, auch kein Leerzeichen! Bei leerem Puffer kehrt das Programm wieder in die gleiche Zeile zurück. Drücken Sie jetzt eine Taste, so wird das Zeichen auf dem Bildschirm ausgegeben.

Um zu sehen, wie schnell GET arbeitet, nehmen wir eine kleine Änderung an unserem Programm vor.

```
10 GET A$:Z=Z+1:PRINT CHR$(19) Z:IF A$="" THEN 10
20 PRINT A$
```

Wir haben jetzt in Zeile 10 noch zusätzlich einen Zähler Z eingebaut, der bei jedem Schleifendurchlauf den Zähler um eins erhöht und die Anzahl der Durchläufe bis zu einem Tastendruck in der rechten oberen Bildschirmecke anzeigt. PRINT CHR$(19) hat die gleiche Funktion wie das Betätigen der HOME-Taste.

Wir sind mit diesem kleinen Programm schon in der Lage, bestimmte Tasten zu selektieren, d.h. wir können für unsere Programme nur ganz bestimmte Tasten zulassen. Ausnahmen bilden hier nur die Tasten RUN/STOP und RESTORE, da diese nicht durch CHR$-Codes abfragbar sind. Schauen wir uns dazu ein Beispiel an.

```
10 REM TASTENSELEKTIERUNG MIT GET
20 PRINT CHR$(147)
30 GET A$:IF A$ = "" THEN 30
40 IF A$=CHR$(65) THEN PRINT"TASTE A":GOTO 30
50 IF A$=CHR$(66) THEN PRINT"TASTE B":GOTO 30
60 IF A$=CHR$(69) THEN END
70 PRINT"ES SIND NUR DIE TASTEN A,B ODER"
80 PRINT"E FUER ENDE ZULAESSIG."
90 GOTO 30
```

Geben Sie dieses Programm wieder in den Rechner ein, nachdem Sie das alte Programm mit NEW gelöscht haben. Starten Sie das Programm, so werden Sie feststellen, daß außer den Tasten A,B oder E keine anderen Tasten vom Programm zugelassen werden, auch nicht SHIFT/CLR HOME.

Hätte man die Zeilen 70 und 80 weggelassen, so wäre das Programm sofort wieder nach Zeile 30 gesprungen, ohne eine Reaktion bei Betätigung einer anderen Taste zu zeigen. In der Wahl der Funktionen, die Sie den Tasten zuordnen, sind Sie vollkommen frei, d.h. Sie können jede beliebige Befehlsfolge dem THEN folgen lassen.

Mit dem folgenden kleinen Programm können Sie z.B. den ASCII-Wert eines Zeichens sowie das Zeichen selbst ausgeben lassen. Da einige Zeichen bzw. Tasten Steuerfunktionen ausführen (Farbwechsel o.ä.), wird in einer speziellen Zeile der Ursprungszustand von Farbe und Schrift wiederhergestellt. Im folgenden nun das Programm:

```
10 REM ANZEIGE DER ASCII-WERTE
20 PRINT CHR$(147)
30 GET A$:IF A$="" THEN 30
40 PRINT TAB(6) CHR$(ASC(A$)) TAB(12)ASC(A$)
50 REM ALTER ZUSTAND BILDSCHIRM
60 PRINT CHR$(154) CHR$(142)
70 GOTO 30
```

Die Zeilen 10 bis 30 dürften von der Funktion her bekannt sein. Interessant ist die Zeile 40. Hier wird auf die 6. Position der Zeile das Zeichen mit CHR$ ausgegeben. Der Wert ergibt sich aus ASC(A$). Sie sehen nur dann ein Zeichen, wenn es sich auch um darstellbare Zeichen im Normalmodus handelt. Wechseln Sie z.B. die Farbe der Schrift im Normalmodus mit CTRL und 2 in weiß, so wird auf dem Bildschirm ja auch kein Zeichen ausgegeben. Der zweite Teil der Zeile gibt schließlich den ASCII-Wert aus. Zeile 60 stellt den alten Zustand der Schriftfarbe wieder her und schaltet wieder auf den Groß/Grafikmodus um. Zeile 70 springt dann wieder zur Zeile 30.

Im nächsten Beispiel wollen wir mit GET genau vier Zeichen einlesen und dem String B$ zuordnen. Das könnte wie folgt aussehen:

```
10 REM VIER ZEICHEN MIT GET EINLESEN
20 FOR I = 1 TO 4
30 GET A$:IF A$="" THEN 30
40 B$=B$+A$
50 NEXT I
60 PRINT B$
70 END
```

In diesem Programm wird durch eine FOR...NEXT-Schleife erreicht, daß mit GET genau vier Zeichen eingelesen werden. In Zeile 30 wird überprüft, ob sich ein Zeichen im Tastaturpuffer befindet, bzw. ob eine Taste gedrückt wurde. Betätigen Sie nun eine Taste, so wird dieses Zeichen der Variablen A$ zugeordnet. Zeile 40 verkettet die eingelesenen Zeichen miteinander in der Variablen B$. Zeile 50 bildet schließlich das Ende der FOR...NEXT Schleife. Wurde die FOR...NEXT-Schleife viermal durchlaufen, so wird die Variable B$ ausgegeben.

Diese Technik kann man benutzen, um z.B. vierstellige Zahlen mit GET einzulesen. Dabei muß ja jede einzelne Ziffer eingelesen und nachträglich zu einer vierstelligen Zahl zusammengesetzt werden.

Sie haben nun einige Anwendungsbeispiele des GET-Befehls kennengelernt. Die wichtigste Eigenschaft ist wohl die der Selektierung der einzelnen Tasten. Dadurch kann man bei eigenen Programmen die Menüsteuerung fast narrensicher gestalten. Wir wollen nun in unserem Rechenlehrgang den INPUT-Befehl durch den GET-Befehl ersetzen. Auf der nächsten Seite sehen Sie, wie die ersten beiden INPUT-Befehle durch die GET-Befehle ersetzt wurden. Studieren Sie genau, welche Abfragen mit IF...THEN durchgeführt wurden, um nur bestimmte Zeichen bei der Eingabe zuzulassen. Hier hat es sich schon bezahlt gemacht, daß wir in Zehnerschritten programmiert haben. So fällt es uns nicht schwer, die zusätzlichen

Zeilen, die für den GET-Befehl notwendig sind, in das Programm einzufügen. Nachfolgend nun die ersten abgeänderten Programmzeilen bis Zeile 320.

```
.
.
.
120 PRINT
130 PRINT TAB(12)"FUER MULTIPLIKATION EINE 4"
133 PRINT
135 PRINT TAB(12)"FUER ENDE EINE 5"
140 PRINT
150 PRINT TAB(12)"WELCHE ZAHL ?"
155 GET A$:IF A$="" THEN 155
160 IF VAL(A$) < 1 OR VAL(A$) > 5 THEN 155
170 ON VAL(A$) GOTO 200,600,1000,1300,1600
200 REM **********
210 REM  ADDITION
220 REM **********
230 PRINT CHR$(147)
240 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
250 PRINT
260 PRINT TAB(10)"FUER DIE ADDITION EIN."
270 PRINT
290 PRINT TAB(10)"GROESSTE ?"
291 FOR I=1 TO 3
293 GET A$:IF A$="" THEN 293
295 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 293
297 B$=B$+A$:NEXT I
298 GR=VAL(B$)
299 REM
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
.
.
.
```

Wie Sie erkennen können, hat sich als erstes in den Zeilen 150 bis 170 etwas verändert. Im ersten Programmteil stand in Zeile 150 der INPUT-Befehl mit zusätzlicher Textausgabe. Der INPUT-Befehl wurde aus

dieser Zeile herausgenommen und gegen den PRINT-Befehl, der den Anwender zur Eingabe einer Zahl auffordert, ersetzt. In Zeile 155 wurde nun der GET-Befehl mit der bekannten Abfrage auf einen Leerstring untergebracht. Zeile 160 hat sich insofern verändert, daß wir den VAL-Befehl benutzt haben, um den String A$ in eine Zahl umzuwandeln und um ihn dadurch auf kleiner als 1 oder größer als 5 überprüfen zu können. Das gleiche Prinzip wurde in der nächsten Zeile verwendet, um das Programm durch ON...GOTO entsprechend verzweigen lassen zu können. Es wird ja mit VAL(A$) ein Wert zwischen 1 und 5 ermittelt, der jetzt genauso benutzt wird, als ob dort die Zahl selbst oder eine Variable gestanden hätte.

Das war schon das ganze Geheimnis der Umwandlung des ersten INPUT-Befehls in den GET-Befehl. Haben Sie diese ersten Änderungen vorgenommen, so starten Sie das Programm ruhig und versuchen Sie, einmal andere Zeichen oder Tasten zu betätigen als die von 1 bis 5. Probieren Sie auch, den Bildschirm mit SHIFT/CLR HOME zu löschen. Na, merken Sie den Unterschied zum INPUT-Befehl?

Einen kleinen Makel hat unsere GET Routine noch; wir sehen nicht, WO wir auf dem Bildschirm die Daten eingeben und vor allem nicht WELCHE Daten wir eingeben, da diese nicht auf dem Bildschirm angezeigt werden. Das kennen wir vom INPUT-Befehl nicht. Dort hatten wir immer unseren blinkenden Cursor, der genau unsere Position angab, und wir sahen auch, welche Daten wir bis zum Drücken der RETURN-Taste eingegeben hatten. Dies müssen wir also selber in die Hand nehmen. Wie das zu machen ist, lernen wir in einem späteren Kapitel. Vorerst reicht unser Wissen vollkommen aus, um den GET-Befehl in unseren Programmen einsetzen zu können.

3.4.2 FUNKTIONSTASTENBELEGUNG MIT GET

Einige weitere Beispiele mit GET sollen die Anwendung dieses Befehls noch mehr verdeutlichen. Der COMMODORE 64 besitzt auf der rechten Seite der Tastatur vier Tasten, die mit F1 bis F8 gekennzeichnet sind. Es handelt sich hierbei um die sogenannten Funktionstasten. Der Anfänger steht meistens ziemlich ratlos davor, wie er diese Tasten in seinen Programmen vernünftig einsetzen soll. Dabei kann man diesen Tasten nicht nur eine, sondern gleich mehrere Funktionen auferlegen. Das folgende kleine Programm soll dazu dienen, Ihnen Anregungen in dieser Richtung zu geben. Ihrer Phantasie sind da kaum Grenzen gesetzt.

```
10 REM FUNKTIONSTASTENBELEGUNG MIT GET
20 GET A$:IF A$="" THEN 20
30 REM F1
40 IF A$=CHR$(133) THEN PRINT CHR$(147)
50 REM F2
60 IF A$=CHR$(137) THEN PRINT CHR$(144);"F2"
70 REM F3
80 IF A$=CHR$(134) THEN PRINT CHR$(5);"F3"
90 REM F4
100 IF A$=CHR$(138) THEN PRINT CHR$(150):END
110 GOTO 20
```

Durch dieses Programm wurden die Funktionstasten F1 bis F4 mit bestimmten Funktionen belegt. F1 bewirkt ein komplettes Löschen des Bildschirms. Mit F2 schalten Sie die Schriftfarbe auf schwarz um und erhalten die Ausgabe "F2" ebenfalls in schwarz. Mit der F3-Taste schalten Sie zurück auf die hellblaue Schrift mit gleichzeitiger Ausgabe von "F3". Schließlich können Sie durch Betätigen der F4-Taste das Programm beenden. Die Tasten F2, F4 und F8 betätigen Sie durch gleichzeitiges Drücken der SHIFT-Taste. An diesem Beispiel können Sie sehen, daß die Belegung der Funktionstasten denkbar einfach ist, und daß Sie wirklich über die Funktionsweise der einzelnen Tasten frei verfügen können.

Mit der nächsten Anwendung haben Sie die Möglichkeit, Ihre Reaktion zu testen. Sie müssen nach Auftauchen eines Punktes möglichst schnell versuchen, irgendeine Taste zu betätigen. Ihre Reaktionszeit wird in der rechten oberen Bildschirmecke angezeigt. Wollen Sie einen erneuten Reaktionstest machen, so müssen Sie die F1-Taste drücken. Nachfolgend nun das Programm.

```
10 REM REAKTIONSTEST
20 FORI=1TO11:CU$=CU$+CHR$(17):NEXT I
30 FORI=1TO19:CU$=CU$+CHR$(29):NEXT I
40 PRINT CHR$(147)
50 FORI=0TOINT(10000*RND(1))+1:NEXTI
60 PRINT CU$;CHR$(209):TI$="000000"
70 GET A$:IF A$="" THEN 70
80 PRINT CHR$(19)TI/60;"SEC."
90 GET A$:IF A$ < > CHR$(133) THEN 90
100 GOTO 40
```

In den Zeile 20 und 30 wird mittels zweier FOR...NEXT Schleifen ein String erzeugt. Dieser String beinhaltet 11mal das Steuerzeichen, um den Cursor nach unten zu bewegen, und 19mal das Steuerzeichen, um den Cursor nach rechts zu bewegen. Dadurch wird in Zeile 60 erreicht, daß der Punkt (CHR$(209)) genau in der Mitte des Bildschirms auftaucht. In Zeile 50 wird die Verzögerungszeit bis zum Erscheinen des Punktes zufällig zwischen einer bis ca. 10 Sekunden generiert. Zeile 60 setzt dann die "Uhr" auf Null, damit man durch Auslesen von TI die Zeit bis zum Drücken einer Taste berechnen kann. Die restlichen Programmpunkte dürften zu diesem Zeitpunkt keine Schwierigkeiten mehr bereiten.

Das soll nun vorerst an Beispielen für die Verwendung des GET-Befehls reichen. Dieser Befehl wird Ihnen sowieso noch in anderen Programmen oft genug begegnen, wo Sie dann dessen Verwendung genau analysieren können. Schauen wir uns nun noch einige Befehle bzw. Funktionen an, welche in Programmen nicht so häufig Verwendung finden. Man sollte jedoch wissen, mit "wem" man es zu tun hat, wenn ein solcher Befehl in einem Programm einmal Verwendung findet.

3.5 FRE, POS, SYS, USR(X), WAIT

Diese Befehle bzw. Funktionen werden, wie bereits erwähnt, relativ selten in Basicprogrammen verwendet. Das sagt allerdings nichts über deren Wichtigkeit aus. Schauen wir uns also die Befehle der Reihe nach an.

FRE

Diese Funktion wird zur Ermittlung des freien Speicherplatzes benötigt. Die Schreibweise der Funktion sieht wie folgt aus:

FRE(X)

Wollen Sie den freien Speicherplatz Ihres Rechners wissen, so können Sie im Direktmodus folgendes eingeben:

PRINT FRE(X)

Als Ausgabe erhalten Sie einen negativen Wert. Zu diesem Wert muß noch 65536 addiert werden, um den aktuellen Speicherplatz, der zur Verfügung steht, zu erhalten. Die Variable X bei FRE wird durch diesen Vorgang nicht beeinflußt. Statt X kann auch jede andere Variable eingesetzt werden. Selbst wenn X mit einem Wert dieser Funktion übergeben wird, ändert sich dadurch nichts. Es handelt sich also bloß um eine Scheinvariable.

POS

Diese Funktion wird Ihnen innerhalb eines Programms sehr selten begegnen. Mit ihr kann man die aktuelle Cursorposition innerhalb einer Programmzeile erfahren. Damit kann sie Werte zwischen 0 und 79 annehmen. Folgende Beispiele sollen die Funktion näher erläutern. Geben Sie im Direktmodus folgendes in den Rechner ein:

```
PRINT "TEST" POS(X);"TEST A" POS(X)
```

und betätigen Sie die RETURN-Taste. Als Ausgabe erhalten Sie:

```
TEST 4 TEST A 13
```

Die Zahl vier gibt die Position des Cursors nach der ersten Ausführung des PRINT-Befehls bekannt. Dementsprechend zeigt die Zahl 13 die Position des Cursors nach Ausführung des zweiten PRINT-Befehls an. Sie können das überprüfen, indem Sie die Zeile nochmal ohne den ersten POS-Befehl eingeben. Sie werden dann feststellen, daß "TEST A" direkt hinter "TEST" ausgegeben wird, also das erste Zeichen an vierter Position in der Zeile steht. Denken Sie beim Abzählen daran, daß die Zeile mit Position Null beginnt. Mit POS können Sie also die Position in der Zeile ermitteln, an der die nächste Ausgabe mit PRINT erfolgen würde. Dies soll zur Erklärung des POS-Befehls genügen.

SYS

Mit diesem Befehl können Sie an eine Adresse im Rechner springen, an der ein eigenes Maschinenprogramm oder aber eine Routine des Rechners beginnt. Die Kontrolle des Mikroprozessors wird dann nicht mehr durch die Basicbefehle über den Interpreter gesteuert, sondern direkt durch den Maschinencode, der sich nur noch aus Zahlenwerten zusammensetzt. Sie können mit SYS im Prinzip jede Speicherstelle des Rechners ansprechen. Bei vielen Adressen "stürzt" der Rechner jedoch ab, d.h. er bleibt in seinen eigenen Routinen hängen, da nicht an der richtigen Stelle eingesprungen wurde. Dieser Befehl setzt also gewisse Kenntnisse des Betriebssystems voraus.

Eine bekannte Anwendung ist die Adresse des Kaltstarts. Der Befehl lautet:

```
SYS 64738
```

Geben Sie diesen Befehl in den Rechner, so wird der

komplette Speicherinhalt gelöscht und der Rechner wird in den Zustand zurückgesetzt, wie Sie ihn vom Einschalten her kennen. Wenden Sie diesen Befehl also mit einer gewissen Vorsicht an.

USR(X)

Der Kurzbeschreibung zu Beginn dieses Buches ist eigentlich nichts mehr hinzuzufügen. Diese Funktion arbeitet ähnlich wie SYS, nur mit dem Unterschied, daß hier ein Wert mit an das Maschinenunterprogramm übergeben werden kann. Die Benutzung dieser Funktion setzt schon fortgeschrittene Kenntnisse in Basic und in Maschinenprogrammierung voraus, sodaß sich der Anfänger in Sachen Basic damit vorerst nicht zu beschäftigen braucht.

WAIT

Durch den WAIT-Befehl können Sie ein Programm solange anhalten, bis eine Speicherstelle einen bestimmten Wert angenommen hat. Genauer gesagt, wartet das Programm mit WAIT auf ein bestimmtes Bitmuster in der Speicherstelle. Auch dieser Befehl wird relativ selten eingesetzt. Im nächsten Kapitel werden wir lernen, wie man damit auf einen Tastendruck von der Tastatur warten kann, ohne den GET-Befehl verwenden zu müssen.

Soweit die Befehle FRE, POS, SYS, USR(X) und WAIT. Im nächsten Kapitel befassen wir uns mit den beiden Befehlen PEEK und POKE. Dabei werden wir dann ein paar von den Befehlen anwenden können, die wir gerade besprochen haben.

3.6 PEEK UND POKE

PEEK

Der PEEK-Befehl wird dazu benutzt, den Wert einer Speicheradresse auszulesen. PEEK bedeutet im Englischen soviel wie "flüchiger Blick" bzw. "spähen, gucken". Das Programm "späht" in eine Speicherzelle und gibt den Wert dieser Zelle aus. Die Schreibweise des Befehls lautet wie folgt:

```
PEEK(ADRESSE)
```

Dabei kann für "Adresse" ein Wert zwischen 0 und 65535 benutzt werden. "Spähen" wir einmal in die Speicherstelle 1024. Geben Sie dazu folgendes ein:

```
PRINT PEEK(1024)
```

Es handelt sich hierbei um eine Adresse des Bildschirmspeichers. Befindet sich auf Ihrem Bildschirm in der linken oberen Position kein Zeichen, so erhalten Sie als Ausgabe den Wert 32. Sie können diese Werte auch durchaus Variablen zuordnen, wie folgendes Beispiel zeigt.

```
X=PEEK(1024)
```

Jetzt wurde der Variablen X der Wert 32 zugeordnet. Sie können das überprüfen, indem Sie eingeben:

```
PRINT X
```

Sie erhalten dann auf dem Bildschirm wieder den Wert 32. Schauen wir uns einmal die ersten 42 Werte des Basicspeichers an, der bei Adresse 2048 beginnt. Geben Sie dazu das folgende Programm in den Rechner ein:

```
10 REM 42 BASICSPEICHERSTELLEN
20 FOR I=2048 TO 2089
30 PRINT PEEK(I);
40 NEXT I
50 END
```

Starten Sie das Programm nun in der gewohnten Weise. Auf dem Bildschirm erscheint eine Zahlenfolge von 42 Zahlen. Ohne weiter in die Materie eindringen zu wollen, sei hier nur gesagt, daß Sie nun Ihr Basicprogramm so sehen, wie es der COMMODORE 64 intern abspeichert. Ist doch recht interessant, nicht wahr?

Allerdings werden Sie wahrscheinlich vorerst diesen Befehl recht selten in Ihren Programmen einsetzen. Das dürfte für den nächsten Befehl nicht zutreffen.

POKE

Dieser Befehl ist das genaue Gegenteil vom PEEK-Befehl. Mit POKE schreiben Sie einen Wert in eine Speicherstelle. Der POKE-Befehl wird von Anfängern zuerst häufig dazu benutzt, den Farbspeicher für den Bildschirmrahmen und den Bildschirmhintergrund zu verändern. Dabei ist 53280 die Speicherstelle für den Bildschirmrahmen und 53281 für den Bildschirmhintergrund. Wollen Sie die Farbe des kompletten Bildschirms in schwarz umändern, so geben Sie folgende POKE-Befehle in den Rechner ein:

POKE 53280,0:POKE 53281,0

und betätigen wieder die RETURN-Taste. Der gesamte Bildschirm wird dabei schwarz. Die Schrift besitzt immer noch die gleiche Farbe. Diese können Sie mit der CTRL- und der entsprechenden Farbtaste (1-8) verändern, oder auch mit dem POKE-Befehl. Es existiert eine Speicherstelle, in der Sie durch Hineinschreiben der Werte 0 bis 15 die Farbe der Schrift bestimmen können. Es handelt sich hierbei um die Speicherstelle 646. Führen Sie den folgenden Befehl aus:

```
POKE 646,1
```

Die Farbe des Cursors ist nun weiß, und ebenso alles, was Sie neu auf den Bildschirm schreiben. Die Farbwerte 0=schwarz, 1=weiß usw. entnehmen Sie bitte dem Handbuch zu Ihrem COMMODORE 64.

Im vorigen Kapitel wurde schon angedeutet, daß der WAIT-Befehl dazu benutzt werden kann, die Betätigung einer Taste zu registrieren. Hierzu wird die Speicherstelle 198 benötigt. In dieser Speicherstelle wird registriert, wieviele Tasten gedrückt wurden. Das folgende kleine Programm soll Ihnen eine Anwendung des WAIT-Befehls zeigen.

```
10 REM TASTATURABFRAGE MIT WAIT
20 POKE 198,0
30 WAIT 198,1
40 PRINT "TASTE GEDRUECKT"
50 END
```

Zeile 20 setzt die Adresse 198 auf Null, was dann soviel bedeutet, daß keine Taste gedrückt wurde. Das Programm wartet nun durch den WAIT-Befehl in Zeile 30 solange, bis die Speicherstelle 198 den Wert eins annimmt. Das ist genau dann der Fall, sobald eine Taste gedrückt wird. Danach fährt das Programm in seiner Ausführung fort und gibt die Meldung aus: "TASTE GEDRUECKT".

Damit haben Sie eine komfortable Alternative zur Tastaturabfrage mit GET kennengelernt, da ja der Vergleich mit IF...THEN nicht benötigt wird. Dem POKE-Befehl kann natürlich auch eine Variable folgen, deren Wert dann in die Speicherstelle geschrieben wird. Der Wert darf allerdings nicht größer als 255 sein.

Das soll zum Gebrauch der Befehle PEEK und POKE genügen. Die Anwendungsmöglichkeiten in den einzelnen Programmen dürften aus diesen Erklärungen hervorgehen. Im nächsten Kapitel befassen wir uns mit der Möglichkeit, Informationen bzw. Daten als Konstanten im Programm abzulegen, so daß sie jederzeit abrufbereit sind.

3.7 READ, DATA, UND RESTORE

Bisher haben wir nur die Möglichkeit kennengelernt, Daten von der Tastatur aus einzugeben. Dazu benutzten wir die Befehle INPUT oder GET. Die Daten wurden dann irgendwelchen Variablen zugeordnet und weiterverarbeitet.

Benötigt unser Programm aber eine große Anzahl von Daten, d.h. numerische Werte oder Zeichenketten (Strings), so ist es sehr umständlich, diese Werte bei jedem Neustart des Programms wieder eingeben zu müssen. Diesen Umstand kann man umgehen, indem man die Kombination von READ und DATA anwendet.

Die DATA-Anweisung setzt sich aus einer Liste von Daten zusammen, wobei die einzelnen Daten durch Kommata getrennt werden. Die Art der Daten, die in der DATA-Anweisung abgelegt werden, können sowohl numerischen Typs als auch Strings sein.

Mit READ können die einzelnen Daten der Reihe nach ausgelesen werden. Dabei ist darauf zu achten, daß der Variablentyp, der dem READ folgt, auch dem gelesenen Datentyp entspricht. Sie dürfen also mit einer numerischen Variablen keinen String in einer DATA-Zeile lesen.

Die DATA-Zeilen sind an keine feste Stelle innerhalb des Programms gebunden. Sie können am Anfang, in der Mitte, am Ende oder sonstwo im Programm stehen. Trifft das Programm auf einen READ-Befehl, so sucht es sich automatisch die dazu passende DATA-Zeile. Schauen wir uns dazu ein einfaches Beispiel an. Geben Sie zuerst NEW ein - diesen Befehl sollten Sie übrigens jedesmal vor einer neuen Programmeingabe ausführen - und dann das nachfolgende Programm.

```
10 READ X
20 PRINT X
30 DATA 50
40 END
```

Als Ausgabe erhalten Sie den Wert 50. In Zeile 10 wird der numerischen Variablen X mittels READ der Wert 50 zugeordnet. Trifft das Programm also auf den READ-Befehl, so sucht es die dazugehörige DATA-Zeile und liest den ersten Wert. Dieser Wert wird der Variablen, die dem READ folgt, zugeordnet. Anschließend wird in Zeile 20 der Inhalt der Variablen X ausgegeben. Die jetzt folgende Zeile 30 hat keinen Einfluß mehr auf den Programmablauf. Ändern Sie nun das Programm in der folgenden Weise ab:

```
10 READ X,Y,Z
20 PRINT X,Y,Z
30 DATA 10,20,30
40 END
```

Nachdem Sie das Programm gestartet haben, erhalten Sie die folgende Ausgabe:

10 20 30

READ hat hier zuerst den ersten Wert in der DATA-Zeile der Variablen X zugeordnet. Danach wurde der zweite Wert gelesen und der Variablen Y zugeordnet und schließlich wurde die Variable Z mit dem dritten Wert belegt. Bei jedem Lesezugriff auf die Daten in den DATA-Zeilen wird also immer der nächste Wert gelesen. Es wird dazu intern im Rechner ein sogenannter ZEIGER verwendet, der bei jedem Lesezugriff um eins weiter gerückt wird. Dieser Zeiger weist damit immer auf das nächste zu lesende Element. Beim Start eines Programms zeigt dieser Zeiger demnach auf das erste Element in der DATA-Zeile. Die nächsten Zeilen sollen dies einmal veranschaulichen. Der Zeiger soll durch dieses "↑" Zeichen dargestellt werden.

```
30 DATA 10,20,30
        ↑
```

Erfolgt jetzt vom Programm der Zugriff mit READ, so wird der Zeiger um eins erhöht und zeigt somit auf das zweite Element.

```
30 DATA 10,20,30
          ↑
```

Wird dieses Element jetzt ebenfalls gelesen, so wird der Zeiger wieder um eins erhöht. Trifft der Zeiger auf das Ende einer Liste von DATA-Anweisungen, so wird er NICHT automatisch zurück auf das erste Element gesetzt, sondern zeigt quasi hinter das letzte Element. Wird nun versucht, erneut mit READ auf die Liste zuzugreifen, gibt der Rechner die Fehlermeldung

```
?OUT OF DATA ERROR IN (Zeilennumer)
```

aus. Was ist aber nun, wenn man öfters auf diese Daten zugreifen will? Auch hierfür gibt es eine Lösung. Sie heißt:

RESTORE

Der RESTORE-Befehl setzt den Zeiger zurück auf das erste Element einer DATA-Zeile. Damit haben Sie jetzt die Möglichkeit, die Daten in den DATA-Zeilen beliebig oft auslesen zu lassen. Geben Sie zunächst das folgende Programm ein, um einmal zu sehen, was geschieht, wenn das Programm versucht, mehr Daten zu lesen als vorhanden sind.

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
40 READ D,E,F
50 PRINT D,E,F
60 END
```

Nachdem die Werte 10, 20, 30 ausgegeben wurden, erscheint die Fehlermeldung:

```
?OUT OF DATA ERROR IN 40
```

In Zeile 40 wurde nämlich versucht, ein viertes Element der DATA-Zeile zu lesen, welches ja nicht vorhanden ist. Um diesen Fehler auszuschalten, könnte man entweder

weitere drei Elemente an die DATA-Zeile anhängen, oder aber einfach den Zeiger mit RESTORE zurücksetzen. Probieren wir das einmal aus. Geben Sie dazu folgende Zeile in den Rechner ein und betätigen Sie danach die RETURN-Taste.

```
35 RESTORE
```

Geben Sie jetzt den Befehl LIST ein, dann sollte Ihr Programm wie folgt aussehen:

```
10 READ A,B,C
20 PRINT A,B,C
30 DATA 10,20,30
35 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 END
```

Wenn Sie das Programm jetzt starten, so unterbleibt die Fehlermeldung und Sie erhalten die folgende Ausgabe:

```
10          20          30
10          20          30
```

Durch den Befehl RESTORE wurde der Zeiger wieder auf das erste Datenelement gesetzt, und somit konnten den numerischen Variablen D, E, und F ebenfalls die Werte 10, 20, und 30 zugeordnet werden. Nun werden ja mit dem Befehl

```
READ A,B,C
```

drei Werte auf einmal aus der DATA-Zeile ausgelesen. Dies geschieht dadurch, daß mit einem READ gleich drei Variablen angesprochen werden. Man kann selbstverständlich die Werte auch nacheinander nur einer Variablen zuordnen, wie das folgende Beispiel zeigen wird.

```
10 FOR I=1 TO 3
20 READ X
30 PRINT X
40 NEXT I
50 DATA 10,20,30
60 END
```

In diesem Beispiel wurden die Befehle READ X und PRINT X in einer FOR...NEXT-Schleife zusammengefaßt, die insgesamt dreimal durchlaufen wird. Bei jedem Durchlauf wird X ein neuer Wert aus der DATA-Zeile zugeordnet und danach ausgegeben.

Wie bereits erwähnt, können Sie auch Strings in den DATA-Zeilen unterbringen. Normalerweise brauchen diese Strings nicht in Anführungszeichen gesetzt zu werden.

Wie überall gibt es auch hier Ausnahmen. Und zwar müssen Sie das Komma, den Doppelpunkt, das Semikolon, Leerzeichen, geshiftete Buchstaben sowie graphische Zeichen und die Cursorsteuerzeichen in Anführungszeichen setzen. Denken Sie vor allem in Ihren Programmen daran, daß kein String einer numerischen Variablen zugeordnet werden darf, da das Programm sonst mit der Fehlermeldung

```
?SYNTAX ERROR IN (Zeilennumer)
```

abbricht. Haben Sie eine längere Liste von gemischten Daten, so kann eine solch falsche Zuordnung sehr rasch geschehen. Das folgende Beispiel soll das Problem deutlich machen.

```
10 FOR I=1 TO 3
20 READ A,B,C$
30 PRINT A,B,C$
40 NEXT I
50 DATA 10,20,TEST 1,30,40,TEST 2,50,TEST 3,OK
60 END
```

Bis zum zweiten Durchlauf der FOR...NEXT Schleife arbeitet das Programm einwandfrei. Bis dahin stimmen auch die Zuordnungen der Daten zu den Variablen, die dem READ-Befehl folgen. Laut READ sollen zuerst zwei numerische Variablen und dann eine Stringvariable gelesen werden. Die Reihenfolge der Daten in Zeile 50 entspricht der Variablenordnung hinter READ aber nur bis zur siebten Position, also dem Wert 50. Danach versucht das Proramm, der numerischen Variablen B den String "TEST 3" zuzuordnen. Da dies ja, wie Sie wissen, nicht möglich ist, bricht das Programm nach zwei Durchläufen mit der Fehlermeldung

?SYNTAX ERROR IN 50

ab. Daher achten Sie bei Verwendung solcher Variablenkombinationen hinter READ darauf, daß die Datentypen in den DATA-Zeilen auch die geeigneten Variablentypen zugeordnet bekommen.

Wir haben nun eine Menge darüber erfahren, wie man Daten dem Rechner bzw. dem Programm übergeben kann. Einmal haben wir die Möglichkeit, Daten per Tastatur mit dem INPUT- oder GET-Befehl einzulesen. Die zweite Möglichkeit, die wir gerade kennengelernt haben, besteht darin, die anfallenden Daten in DATA-Zeilen abzulegen, um sie so immer verfügbar zu haben. Diese Daten werden ja bei der Speicherung des Programms auf Kassette oder Diskette mit abgespeichert! Will man dagegen mittels INPUT oder GET eingegebene Daten "retten", so muß man diese in einer separaten Datei auf Diskette oder Kassette abspeichern.

Eine sehr häufige Anwendung dieser Kombination von READ und DATA findet man, wenn mittels Basic ein Programm in Maschinensprache generiert werden soll. Das sind dann meistens Unmengen von DATA-Zeilen, die mittels einer FOR...NEXT-Schleife mit READ gelesen werden, und dann mit POKE in die entsprechenden Speicherstellen geschrieben werden. Danach kann man mit dem SYS-Befehl das Maschinenprogramm starten. Wie so etwas aussehen kann, soll das nächste Programm demonstrieren. Mit dieser

Maschinenroutine werden der Bildschirm gelöscht, die Bildschirmfarbe auf schwarz und die Schriftfarbe auf orange gesetzt. Alle diejenigen, die sich in der Maschinensprache auskennen, mögen mir dieses "primitive" Beispiel verzeihen. Aber das Programm soll ja auch nur zu Demonstrationszwecken dienen.

```
10 FORI=828 TO 844
20 READ M
30 POKEI,M
40 NEXT I
50 DATA 169,0,141,32,208,141,33,208,169,8,141,134,
2,32,68,229,96
60 END
```

Das Maschinenprogramm wird ab Speicheradresse 828 abgelegt. Das ist beim COMMODORE 64 der Kassettenpuffer. Damit diese Maschinenroutine läuft, dürfen Sie nicht die DATASETTE benutzen, da dann dieser Speicherbereich überschrieben würde.

Was macht dieses Programm nun? Zunächst wird eine FOR...NEXT-Schleife aufgebaut, die den Startwert des Kassettenpuffers benutzt. Dann werden mittels READ die Werte aus der DATA-Zeile ausgelesen und mit POKE in den Speicherbereich geschrieben. Wenn Sie das Programm gestartet haben, meldet der Rechner sich nach kurzer Zeit mit READY. Geben Sie jetzt den Befehl

SYS 828

in den Rechner und drücken Sie RETURN. Augenblicklich wird der Bildschirm gelöscht und die Bildschirmfarbe auf schwarz gesetzt. Die Farbe der Schrift wird in orange geändert. Sie können, solange der Rechner eingeschaltet ist, diese Routine im Direktmodus oder auch in Ihren Programmen mit SYS 828 aufrufen.

Wollen Sie das mit Basicbefehlen erreichen, so müßten Sie das folgende Programm verwenden:

```
10 REM BILDSCHIRM SCHWARZ
20 POKE 53280,0
30 POKE 53281,0
40 PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
50 PRINT CHR$(129):REM SCHRIFTFARBE ORANGE
```

So, das war eine Masse an neuen Informationen und Anwendungsmöglichkeiten zu den Befehlen READ, DATA und RESTORE. Versichern Sie sich, daß Sie dieses Kapitel verstanden haben. Sollte dies nicht der Fall sein, so arbeiten Sie es noch einmal durch.

Im nächsten Kapitel befassen wir uns mit der Generierung von Feldern bzw. Arrays. Auch dort werden wir die Befehle READ und DATA sinnvoll einsetzen können.

4.1 FELDER

Die Programmierung bzw. Verwaltung von Feldern, auch ARRAYS genannt, gehört mit zu den schwierigsten Problemen, vor die ein Anfänger in Sachen Basic gestellt werden kann. Je komplexer die Felder, umso schwieriger ist deren Handhabung. Selbst fortgeschrittene Programmierer haben noch Probleme mit der Verwaltung von solchen Feldern.

Nun schlagen Sie nicht gleich entmutigt das Buch zu. Man kann alles lernen, auch den Umgang mit diesen Feldern. Es ist alles eine Sache der Übung.
Wir fangen wie immer mit ganz einfachen Beispielen an.

4.1.1 EINDIMENSIONALE FELDER

Stellen Sie sich vor, Sie wollen ein Programm schreiben, das Ihnen Ihr durchschnittliches Monatsgehalt berechnet. Wir gehen dabei von 12 Monatsgehältern aus. Das erste Beispiel benutzt eine Schleife, um die Beträge für die Monatsgehälter einzulesen. Mit Ihren bisherigen Kenntnissen müßten Sie auf die folgende Art und Weise vorgehen:

```
10 REM DURCHSCHNITTLICHES MONATSGEHALT
20 REM BERECHNUNG FUER 12 MONATE
30 FOR I=1 TO 12
40 INPUT"MONATSGEHALT";M
50 S=S+M
60 NEXT I
70 D=S/12
80 D=INT(D*100)/100
90 PRINT"DURCHSCHNITTLICHES EINKOMMEN";
100 PRINT D;"DM"
110 END
```

Nachdem Sie dieses Programm in den Rechner gegeben haben, starten Sie es und geben Sie 12 Werte ein. Sie erhalten als Ausgabe das durchschnittliche Monatseinkommen auf 2 Stellen nach dem Dezimalpunkt gerundet. In diesem Programm werden die einzelnen Monatsgehälter in einer FOR...NEXT-Schleife mit INPUT eingelesen. Noch innerhalb der Schleife wird die Summe der Gehälter gebildet (Zeile 50). Zeile 70 berechnet das durchschnittliche Monatseinkommen und in Zeile 80 wird der Betrag auf 2 Stellen gerundet. Das Programm sollte in seinem Aufbau soweit verstanden worden sein.

Wir haben also jetzt das durchschnittliche Monatseinkommen berechnen lassen. Was ist aber, wenn wir nachträglich genau wissen wollen, welches Gehalt wir im Monat Mai bekommen haben? Im letzten Beispiel sind die einzelnen Monatswerte ja verloren gegangen. Nichts leichter als das, werden Sie sagen. Wir nehmen statt einer Variablen eben 12 Variablen, und ordnen je einer ein Monatsgehalt zu. Gut, ändern wir unser Programm dahingehend ab. Mit diesem Vorschlag haben Sie sich übrigens ein gutes Stück der Programmierung von Feldern genähert. Geben wir jedoch zunächst das abgeänderte Programm ein:

```
10 REM DURCHSCHNITTLICHES MONATSGEHALT
20 REM RETTEN DER EINZELNEN MONATSWERTE
30 INPUT"MONATSGEHALT 1";M1
40 INPUT"MONATSGEHALT 2";M2
50 INPUT"MONATSGEHALT 3";M3
60 INPUT"MONATSGEHALT 4";M4
70 INPUT"MONATSGEHALT 5";M5
80 INPUT"MONATSGEHALT 6";M6
90 INPUT"MONATSGEHALT 7";M7
100 INPUT"MONATSGEHALT 8";M8
110 INPUT"MONATSGEHALT 9";M9
120 INPUT"MONATSGEHALT 10";M10
130 INPUT"MONATSGEHALT 11";M11
140 INPUT"MONATSGEHALT 12";M12
150 S=M1+M2+M3+M4+M5+M6+M7+M8+M9+M10+M11+M12
```

```
160 D=S/12
170 D=INT(D*100)/100
180 PRINT"DURCHSCHNITTLICHES EINKOMMEN";
190 PRINT D;"DM"
200 END
```

Wenn Sie das Programm jetzt weiter ausbauen möchten, können Sie jederzeit auf die einzelnen Monatswerte zurückgreifen. Wollen Sie z.B. wissen, wie groß der Betrag Ihres Gehalts im Monat Mai war, so brauchen Sie nur die Variable M5 abzufragen, und schon haben Sie Ihre Antwort. Dies ist natürlich nur unter der Voraussetzung möglich, daß die laufenden Monatszahlen den Zahlen der Variablen entsprechen.

Wir haben jetzt zwar die Monatswerte auch weiterhin in unserem Programm verfügbar, jedoch haben wir uns diesen Vorteil mit neun zusätzlichen Programmzeilen erkaufen müssen. Außerdem ist, wie Sie zugeben müssen, die Benutzung von zwölf Variablen recht umständlich. Haben Sie z.B. vor, sich diese 12 Beträge wieder ausgeben zu lassen, so können Sie diesen Vorgang noch nicht einmal in einer Schleife realisieren, da es sich ja um zwölf verschiedene Variablen handelt. Sie müßten daher für jede einzelne Variable einen PRINT-Befehl benutzen, oder einem PRINT-Befehl alle zwölf Variablen folgen lassen. Das sähe dann ungefähr so aus:

```
    .
    .
    .
100 PRINT M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11,M12
    .
    .
    .
```

Das ist nun nicht gerade übersichtlich oder gar elegant programmiert. Wie schön wäre es, wenn man die Möglichkeit hätte, eine Variable mit einem laufenden Index zu versehen, etwa in der folgenden Art:

A(I)

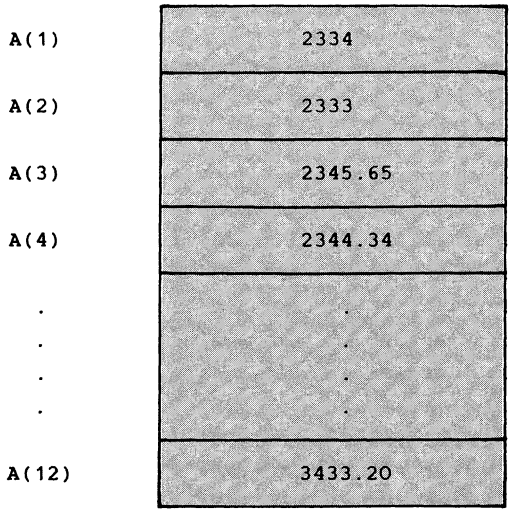
Damit wäre es möglich, die Monatswerte durch eine Schleife ausgeben zu lassen, und auch durch Angabe von I auf jeden Monatswert zugreifen zu können. Sie ahnen es sicherlich schon. Diese Möglichkeit existiert und hat den Namen FELD oder ARRAY.

Was versteht man nun genau unter dem Begriff Feld oder Array?

Wir hatten am Anfang des Buches eine Variable mit einer Schublade verglichen, in der, je nach Art der Variablen, numerische Werte oder Strings enthalten sein können. Sie können sich nun ein solches Feld als einen Schrank mit übereinanderliegenden Schubladen vorstellen. Dabei ist jede Schublade durch eine Zahl gekennzeichnet. Diese Zahl hat nichts mit dem eigentlichen Inhalt dieser Schublade zu tun. Man nennt diese Zahl auch INDEX. Dieser Index wird in Klammern gesetzt und so von der eigentlichen Variablen abgetrennt. Die Schreibweise haben wir vorhin schon kennengelernt. Trotzdem wollen wir sie noch einmal zeigen:

A(1)

Eine solche Variable bezeichnet man als FELDVARIABLE oder auch als INDIZIERTE VARIABLE, da sie ja durch den Index genauer bezeichnet wird. Der Index ist der Wert in den runden Klammern. Verwechseln Sie diese Schreibweise nicht mit der Variablen A1! Das ist ein himmelweiter Unterschied. Das folgende Bild soll nun die Struktur eines solchen Feldes verdeutlichen.



Sie sehen, daß so ein Feld große Ähnlichkeit mit einer Tabelle hat, in der die einzelnen Werte untereinander geschrieben wurden. Unser Feld hat demnach 12 einzelne "Schubladen", denen jeweils ein Wert zugeordnet wurde. Wollen wir jetzt den Inhalt des dritten Elements wissen, so brauchen wir nur den Index 3 zu verwenden. Das könnte z.B. so aussehen:

```
PRINT A(3)
```

Als Ausgabe würden wir in unserem Falle den Wert

```
2345.65
```

erhalten. Wollen wir nun solche Felder in unseren Programmen verwenden, so müssen wir dem Computer erst mitteilen, wie groß unser Feld sein soll, d.h. wie viele Elemente es enthalten soll. Dafür existiert in Basic die DIM-Anweisung. Sie hat die folgende Schreibweise:

DIM FELDNAME(ANZAHL DER FELDER)

Für unser Feld müßten wir also folgende Schreibweise verwenden:

DIM A(12)

Dabei ist A der Feldname und 12 die maximale Anzahl der Felder. Die DIM-Anweisung steht meistens am Anfang eines Programms. Wurde ein Feld einmal dimensioniert, so darf es während des gesamten Programmablaufs nicht erneut mit DIM verändert werden. Sonst gibt der Rechner die Fehlermeldung

?REDIM'D ARRAY ERROR IN (Zeilennummer)

aus. In unserem Beispiel wurde ein Feld für Gleitkommavariablen definiert. Sie können selbstverständlich auch Felder für String- bzw. Integervariablen (Ganzzahlvariablen) benutzen. Schauen wir uns dazu einige Beispiele an:

```
DIM DE$(15)
DIM GZ%(20)
DIM AB(12)
```

Es wurden hier nacheinander Felder für String-, Integer- und Gleitkommavariablen erstellt. Dabei ist darauf zu achten, daß in einem Feld für Gleitkommavariablen keine Strings abgelegt werden dürfen. Das gleiche gilt für die Integervariablen. Sie können mit einer DIM-Anweisung mehrere Felder auf einmal dimensionieren. Das sieht dann wie folgt aus:

DIM A(12),B$(16),S%(20)

Sie können natürlich auch nur Felder für Stringvariablen erstellen.

Zwei Besonderheiten müssen noch erwähnt werden.

1. Benötigen Sie nicht mehr als 11 Elemente in einem Feld, so brauchen Sie keine DIM-Anweisung zu geben. Durch Ansprechen eines Elementes, z.B. mit A(4), führt der Rechner in unserem Falle automatisch eine DIM A(10) Anweisung aus.

2. Sie werden sich gefragt haben, wieso bei DIM A(10) elf Elemente zur Verfügung stehen können. Nun, der Index beginnt bei Null und nicht erst bei eins, so daß Sie das Element A(0) noch hinzuzählen müssen.

Die Darstellung des Feldes auf Seite 145 hätte demnach noch durch das Element A(0) ergänzt werden müssen. Wir wollen jedoch dieses "Nullelement" vorerst bei unseren Betrachtungen unberücksichtigt lassen.

Unter Punkt 1 bei den Besonderheiten wurde erwähnt, daß keine Dimensionierung vorgenommen werden muß, wenn nur bis zu 11 Elemente verwendet werden. Wissen Sie allerdings genau, daß Sie nur 6 Elemente benötigen, so führen Sie ruhig die Anweisung DIM X(5) bzw. DIM X(6) aus, da dadurch wieder Speicherplatz eingespart wird.

Schauen wir uns nun das Programm mit der Berechnung des durchschnittlichen Monatseinkommens nochmal unter Verwendung eines Feldes an.

```
10 REM DURCHSCHNITTLICHES MONATSGEHALT
20 REM MIT BENUTZUNG EINES ARRAYS
30 DIM M(12)
40 FOR I=1 TO 12
50 INPUT"MONATSGEHALT";M(I)
60 S=S+M(I)
70 NEXT I
80 D=S/12
90 D=INT(D*100)/100
100 PRINT"DURCHSCHNITTLICHES EINKOMMEN";
110 PRINT D;"DM"
120 END
```

Durch die Verwendung eines Feldes haben wir nur eine Programmzeile mehr benötigt als beim ersten Beispiel. Man hätte die DIM-Anweisung auch noch in Zeile 20 unterbringen können, obwohl dann der Vergleich zwischen den beiden Programmen nicht gerecht ausgefallen wäre. So haben wir also durch EINE zusätzliche Zeile die monatlichen Gehälter weiter im Programm verfügbar. Wollen Sie z.B. vor der Ausgabe des monatlichen Durchschnitts die monatlichen Einkommen noch einmal auflisten lassen, so könnten Sie den letzten Programmteil wie folgt abändern:

```
   .
   .
90 D=INT(D*100)/100
100 FOR I=1 TO 12
110 PRINT"MONATSGEHALT"I,M(I)
120 NEXT I
130 PRINT"DURCHSCHNITTLICHES EINKOMMEN";
140 PRINT D;"DM"
150 END
```

Wir haben also durch die Verwendung eines Arrays die monatlichen Gehälter weiterhin im Programm verfügbar, ohne das eigentliche Programm komplizierter gestaltet zu haben.

Felder oder Arrays, die die folgende allgemeine Schreibweise

A(X)

haben, bezeichnet man auch als EINDIMENSIONALE FELDER, d.h. sie besitzen nur einen Index.

Der Index muß nicht unbedingt durch eine Zahl dargestellt werden. Es können auch Variablen oder arithmetische Ausdrücke Verwendung finden. Das bedeutet, daß Sie ein Feld individuell auf den jeweiligen Bedarf festlegen können. Bleiben wir bei unserem Beispiel mit den Monatsgehältern. Stellen Sie sich vor, Sie wollten das Programm so gestalten, daß es für eine verschiedene

Anzahl von Monatsgehältern einsetzbar ist. Denkbar wäre dann folgende Änderung am Anfang des Programms:

```
    .
    .
    .
30 INPUT"WIEVIELE MONATSGEHAELTER";Z
40 DIM M(Z)
    .
    .
```

Die Anzahl der Monatsgehälter bestimmt hier also die Größe des Feldes. Verwendet man diesen kleinen Kniff, kann man sein Programm also den augenblicklichen Erfordernissen genau anpassen und den Speicherbereich des Rechners optimal ausnutzen.

Versuchen Sie, ein Element eines Feldes anzusprechen, daß außerhalb des mit DIM(X) dimensionierten Feldes liegt, gibt der Rechner die Fehlermeldung

```
?BAD SUBSCRIPT ERROR
```

aus. Haben Sie z.B. ein Feld mit DIM A(15) definiert und versuchen das Element A(16) anzusprechen, so wird diese Fehlermeldung ausgegeben.

Das soll vorerst an Informationen über die "eindimensionalen Felder" ausreichen. Wir wollen nun anhand einiger Beispiele den Umgang mit diesen eindimensionalen Feldern üben und benutzen dazu die Felder X und Y$ mit jeweils 6 Elementen. Das Element mit dem Index Null wollen wir auch hier weiterhin unberücksichtigt lassen.

4.1.2 ANWENDUNGSBEISPIELE ZU EINDIMENSIONALEN FELDERN

Normalerweise können Sie davon ausgehen, daß nach der DIM-Anweisung die einzelnen Feldelemente leer sind. Innerhalb eines Programms kann es aber auch vorkommen, daß ein komplettes Feld gelöscht werden soll. Bei

numerischen Feldern können Sie das erreichen, indem Sie die Elemente mit Nullen auffüllen. Das nachfolgende Programm zeigt, wie so etwas aussehen kann.

```
10 REM LOESCHEN NUMERISCHES FELD
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=0
50 NEXT I
60 END
```

Wir sind von einem Feld mit 6 (bzw. 7) Elementen ausgegangen. Die FOR...NEXT-Schleife hat den Startwert 1 und einen Endwert, der sich aus der maximalen Anzahl der Feldelemente ergibt, also bei unserem Beispiel 6. Beim Durchlaufen dieser Schleife nimmt I nacheinander die Werte 1,2,3 bis 6 an. Dadurch werden in Zeile 40 alle Elemente des Feldes auf Null gesetzt, da der Index von X jedesmal mit erhöht wird. Ausgeschrieben sähe das dann so aus:

```
X(1)=0
X(2)=0
X(3)=0
X(4)=0
X(5)=0
X(6)=0
```

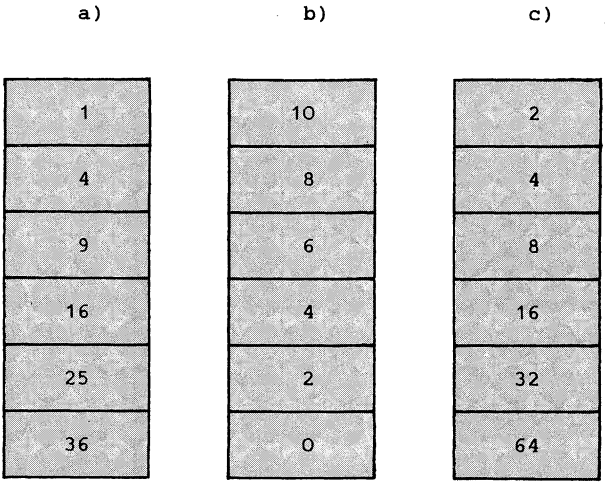
Dieses Programm dürfte das Löschen eines Feldes vom Prinzip her deutlich gemacht haben. Wollen Sie ein Feld löschen, welches Strings beinhaltet, so müssen Sie beachten, daß Sie die Elemente NICHT mit Nullen auffüllen, da ja bei einem String die Null als Zeichen interpretiert wird. Es kann ausreichen, die einzelnen Feldelemente mit einem Leerzeichen aufzufüllen. Wollen Sie in Ihrem Programm aber Strings verketten, so bekommen Sie durch das Leerzeichen immer ein Zeichen mehr in den String. Das kann je nach Programm zu Fehlern führen, z.B. dann , wenn Sie mit der LEN-Funktion arbeiten. Daher sollte man die Elemente eines Stringfeldes mit "NICHTS" auffüllen.

Das sieht dann wie folgt aus:

```
10 REM LOESCHEN STRINGFELD
20 DIM Y$(6)
30 FOR I=1 TO 6
40 Y$(I)=""
50 NEXT I
60 END
```

Der Ablauf des Programms ist der gleiche wie beim Löschen des numerischen Feldes. Es sei hier nur noch darauf hingewiesen, daß in Zeile 40 den einzelnen Elementen ein LEERSTRING zugeordnet wird. Achten Sie darauf, daß Sie kein Leerzeichen zwischen die Anführungszeichen setzen.

Kommen wir nun zu Beispielen, in denen Sie den Umgang mit dem Index in Verbindung mit FOR...Next-Schleifen üben können. Gegeben seien drei Felder mit je 6 Elementen, wobei die Elemente den folgenden Inhalt haben sollen:



Wie lassen sich nun diese drei Felder mit einer FOR...NEXT-Schleife realisieren?

ERKLÄRUNG ZU BEISPIEL A)

Nach kurzem Überlegen stellt man fest, daß die Elemente mit den Quadratzahlen des laufenden Index übereinstimmen. Das Programm dazu könnte wie folgt aussehen:

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=I*I
40 NEXT I
50 END
```

Die Funktion dieser Schleife wird am deutlichsten, wenn man sich die einzelnen Schritte aufschreibt. Für dieses Beispiel will ich Ihnen zeigen, wie so etwas aussehen kann.

FELD X(6)

*I = 1 : X(1) = 1*1 = 1*	1
*I = 2 : X(2) = 2*2 = 4*	4
*I = 3 : X(3) = 3*3 = 9*	9
*I = 4 : X(4) = 4*4 = 16*	16
*I = 5 : X(5) = 5*5 = 25*	25
*I = 6 : X(6) = 6*6 = 36*	36

Bei jedem Durchlauf der Schleife wird I um eins erhöht. Dadurch erreichen wir jedes Element des Feldes, da wir als Index I selbst benutzen. Gleichzeitig wird I mit sich selbst multipliziert und das Ergebnis dem Element zugeordnet, dessen Index gerade I ist. So wird also das Feld der Reihe nach mit den Quadratzahlen gefüllt.
Das gleiche Prinzip, nämlich die Laufvariable der Schleife zur Berechnung heranzuziehen, wurde auch in Beispiel b) verwendet.

ERKLÄRUNG ZU BEISPIEL B)

In diesem Beispiel nehmen die Werte der Feldelemente mit steigendem Index in Zweierschritten ab. Der Startwert des ersten Feldes ist die Zahl 10. Um diesen Effekt zu erzielen, können wir jedoch die FOR...NEXT Schleife nicht verändern, da über die Laufvariable ja die einzelnen Elemente angesprochen werden. Wir müssen uns also eine Zuordnungsregel ausdenken, die mit steigendem Index die Werte in den Elementen in Zweierschritten vermindert. Die Lösung zu diesem Problem könnte wie folgt aussehen:

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=12 - 2*I
40 NEXT I
50 END
```

In Zeile 30 steht unsere Zuordnungsregel. Wir haben wieder den Index zur Berechnung herangezogen. Lassen Sie I in Gedanken nacheinander die Werte 1 bis 6 annehmen, so stellen Sie fest, daß sich dabei genau unsere Zahlenfolge aus Beispiel b) ergibt. Sie können das überprüfen, indem Sie das Programm durch die folgenden Zeilen ergänzen, die dann das gesamte Feld ausgeben.

```
50 FOR I=1 TO 6
60 PRINT X(I)
70 NEXT I
80 END
```

Sollten Sie Schwierigkeiten haben, sich den Vorgang gedanklich vorstellen zu können, so verwenden Sie einfach die Methode aus Beispiel a), indem Sie den Ablauf zu Papier bringen.

Besprechen wir schließlich noch Beispiel c). Sie haben sicherlich schon erkannt, daß auch hier wieder eine Gesetzmäßigkeit dahintersteckt.

ERKLÄRUNG ZU BEISPIEL C)

Die Zahlenfolge in diesem Beispiel kam Ihnen sicherlich gleich von Anfang an sehr bekannt vor. Richtig, es handelt sich hierbei um die Potenzen von 2. Diese Zahlen sind Ihnen im Kapitel über die Zahlensysteme schon einmal begegnet. Es dürfte daher keine Schwierigkeit bereiten, hier eine entsprechende Zuordnungsregel zu finden. Wir benutzen die 2 als Konstante und die Laufvariable bzw. den Index als Potenz. Das sieht als Programm dann so aus:

```
10 DIM X(6)
20 FOR I=1 TO 6
30 X(I)=2↑I
40 NEXT I
50 END
```

Auch hier können Sie durch Anhängen der Programmzeilen aus Beispiel b) die Richtigkeit dieser Zuordnung überprüfen. Benutzen Sie auch hier ruhig ein Blatt Papier, um sich den Vorgang zu verdeutlichen. Überzeugen Sie sich davon, daß Sie das Prinzip dieser Technik verstanden haben. Treffen Sie nämlich in komplexeren Programmen auf solche Techniken bzw. Anwendungen, so werden Sie große Schwierigkeiten bekommen, den Programmablauf zu verstehen.

Bisher haben wir nur numerische Felder betrachtet. Nun wollen wir uns noch den Stringfeldern zuwenden, da bei diesen meistens keine Gesetzmäßigkeiten vorhanden sind, was die Belegung der einzelnen Elemente betrifft. Die Zuordnungen für die Stringfelder werden häufig über die Tastatur eingeleitet. Eine weitere Möglichkeit besteht

darin, über die Befehle READ und DATA ein solches Feld zu "laden".

Stringfelder werden z.B. benutzt, um Namen, Adressen oder auch Zahlenwerte, die dann allerdings als String vorliegen, im Rechner zu speichern. Insofern sind Stringfelder von der Verwendung her vielseitiger.
Lassen Sie uns zunächst ein Feld erstellen, in dem wir die Vornamen aus unserem Bekanntenkreis im Rechner abspeichern können. Das erscheint im Moment zwar sinnlos, da wir noch nicht gelernt haben, diese Daten auf ein Speichermedium abzuspeichern, ist aber für das Verständnis späterer Datenverwaltungsprogramme unerläßlich.

Haben Sie die Anzahl Ihrer Bekannten genau im Kopf? Das ist wahrscheinlich nicht der Fall. Daher wird ein solches Programm immer die Größe eines Feldes vorgeben müssen. Wir können hier also nicht die Größe des Feldes vorher abfragen und dementsprechend eine DIM-Anweisung im Programm vornehmen. Ist die Kapazität eines Feldes innerhalb des Programms ausgelastet, sollte eine entsprechende Meldung durch das Programm ausgegeben werden, damit es nicht von selbst mit einer Fehlermeldung abbricht. Unser Beispiel soll zeigen, wie ein solches Programm aufgebaut werden könnte.

```
10 REM VORNAMENLISTE
20 DIM Y$(6)
30 Z=Z+1
40 INPUT"VORNAME";Y$(Z)
50 PRINT"WEITERE EINGABEN J/N ?"
60 GET A$:IF A$ ="" THEN 60
70 IF A$ < > "J" THEN 100
80 IF Z < 6 THEN 30
90 PRINT"LISTE VOLL !"
100 END
```

Es wurde hier zur besseren Übersichtlichkeit nur ein Feld mit 6 (bzw. 7) Elementen aufgebaut (Zeile 20). Zeile 30 beinhaltet den Zähler, der bei jeder Eingabe um eins erhöht wird. Da wir nicht die genaue Anzahl der Vornamen wissen, die wir eingeben wollen, kann hier keine FOR...NEXT Schleife verwendet werden. Zeile 40 verlangt mit INPUT die Eingabe eines Vornamens und ordnet diesen dem Element mit dem Index von Z zu. In Zeile 50 wird gefragt, ob weitere Eingaben gemacht werden sollen. Die Funktion von Zeile 60 dürfte inzwischen bekannt sein. Zeile 70 vergleicht das eingegebene Zeichen mit "J". Ist das Zeichen ungleich "J", wird das Programm in Zeile 100 beendet. Sollen weitere Eingaben stattfinden, so vergleicht Zeile 80 den Zähler auf kleiner 6. Hat der Zähler bereits den Wert 6, so wird durch Zeile 90 die Meldung "LISTE VOLL" ausgegeben und das Programm wird wiederum beendet. Hätten wir diese Zählerabfrage nicht eingebaut, so würde das Programm bei einem Wert größer 6 mit der Fehlermeldung

?BAD SUBSCRIPT ERROR IN 40

abbrechen. Erreicht Z nämlich den Wert 7, so wird in Zeile 40 versucht, das Element Y$(7) anzusprechen, das laut DIM-Anweisung aber nicht existiert. Solche ungewollten Programmabbrüche sollte man möglichst vermeiden.

Das Programm gibt so natürlich noch nicht viel her. Das Prinzip sollte aber klar geworden sein. Man hätte jetzt zusätzlich noch eine Abfrage hineinbringen können, ob das gesamte Feld anschließend hätte ausgegeben werden sollen. Diese Ergänzung können Sie ja mal selbst vornehmen, indem Sie eine entsprechende IF...THEN Abfrage im Programm unterbringen. Das Feld können Sie dann, wie bereits erklärt, wieder mit einer FOR...NEXT Schleife ausgeben lassen.

Die Dimensionierung des Feldes könnte mit DIM D$(200) bei einer eigenen Dateiverwaltung vollkommen ausreichend sein. Allerdings reicht da kein eindimensionales Feld zur Erfassung der Daten aus.

Kommen wir aber zunächst noch zu einem Beispiel, wo ein Feld mit den Befehlen READ und DATA mit Daten belegt wird. Stellen Sie sich vor, Sie benötigen in einem Programm des öfteren die Wochentage. Es ist nun recht mühselig, diese Daten bei jedem Programmstart neu eingeben zu müssen. Was spricht also dagegen, diese Daten in DATA-Zeilen abzulegen und gleich nach dem Programmstart mit READ in ein Feld einlesen zu lassen? Schauen wir uns das wiederum an einem Beispielprogramm an.

```
10 REM WOCHENTAGE
20 DIM WT$(7)
30 FOR I=1 TO 7
40 READ WT$(I)
50 NEXT I
60 DATA MONTAG,DIENSTAG,MITTWOCH,DONNERSTAG,FREITAG,
SAMSTAG,SONNTAG
70 REM AUSGABE JA/NEIN
80 PRINT"AUSGABE DES FELDES J/N ?"
90 GET A$:IF A$="" THEN 90
100 IF A$ < > "J" THEN 140
110 FOR I=1 TO 7
120 PRINT WT$(I)
130 NEXT I
140 END
```

Dieses Programm ähnelt sehr dem Programm mit der Vornamenliste. Der eigentliche Unterschied besteht in der Zeile 40, wo anstatt durch INPUT die Daten den einzelnen Elementen mit READ zugewiesen werden. Die DATA-Zeile erklärt sich somit von selbst. Den Schluß des Programms bildet eine Ausgabemöglichkeit des kompletten Feldes.

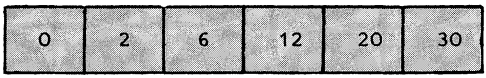
Damit haben Sie auch die Möglichkeit kennengelernt, Daten einem Feld mit den Befehlen READ und DATA zu übergeben.

Bevor wir uns nun den mehrdimensionalen Feldern zuwenden wollen, können Sie anhand der Aufgaben auf der nächsten Seite überprüfen, ob Sie die Thematik "eindimensionale Felder" verstanden haben. Beim Lösen der Aufgaben wünsche ich Ihnen wie immer viel Spaß!

1) Schreiben Sie ein Programm, das sechs Namen einliest und diese Daten in einem Feld ablegt. Weiterhin soll das Programm Ihnen den Namen ausgeben, der von der alphabetischen Reihenfolge her vorne steht. Testen Sie das Programm mit den Namen Rolf, Peter, Hans, Christel, Franz und Helmut. Denken Sie daran, daß Sie Strings untereinander auf gleich, kleiner oder größer vergleichen können. Als Ergebnis müßten Sie den Namen "Christel" erhalten.

2) Schreiben Sie ein Programm, das sechs Zufallszahlen erzeugt und diese Zahlen ebenfalls in einem Feld ablegt. Weiterhin soll die größte dieser Zahlen ausgegeben werden. Die Zufallszahlen sollen in einem Bereich zwischen 50 und 150 vorkommen.

3) Gegeben sei das folgende Feld X(6):



Schreiben Sie ein Programm und entwickeln Sie eine Zuordnungsregel, die dieses Feld erzeugt. Lassen Sie das Feld zur eigenen Überprüfung ausgeben. Stören Sie sich nicht daran, daß die einzelnen Elemente diesmal waagerecht angeordnet sind. Bei eindimensionalen Feldern spielt das keine Rolle. Damit keine Mißverständnisse entstehen, wurden die einzelnen Elemente noch beschriftet.

1.
```
10 REM NAMEN EINLESEN
20 DIM Y$(6)
30 FOR I=1 TO 6
40 INPUT"NAME";Y$(I)
50 NEXT I
60 REM 1. ALPHABETISCHER NAME
70 Y$(0)=Y$(1)
80 FOR I=2 TO 6
90 IF Y$(0) < = Y$(I) THEN 110
100 Y$(0) = Y$(I)
110 NEXT I
120 PRINT"1.NAME";Y$(0)
130 END
```

Der erste Programmteil dürfte Ihnen keine Schwierigkeiten bereitet haben, da er schon vorher in einigen Beispielen vorgestellt worden ist. Da Sie wußten, daß insgesamt 6 Namen eingelesen werden sollten, konnten Sie hier eine FOR...NEXT-Schleife verwenden. Der zweite Teil des Programms war, zugegeben, schon etwas kniffliger. Haben Sie dieses Problem ebenfalls gelöst, so dürfen Sie sich jetzt selbst auf die Schulter klopfen. Wir hatten schon in einem früheren Kapitel über die Zwischenspeicherung von Werten bzw. Daten gesprochen. Genau diese Technik mußten Sie auch hier wieder verwenden. Es sollen ja keine Daten verloren gehen. Welche Stringvariable Sie nun dafür benutzt haben, ist eigentlich nicht so wichtig. Angeboten hat sich hier allerdings das Feldelement Y$(0), da dies sowieso bisher keine Verwendung fand. In Zeile 70 wurde also der Inhalt von Element Y$(1) in Y$(0) zwischengespeichert. In Zeile 80 beginnt dann die FOR...NEXT-Schleife mit dem Startwert 2. Startwert 1 kann entfallen, da wir ja nicht das erste Element mit sich selbst zu vergleichen brauchen. In Zeile 90 werden dann die einzelnen Namen der Reihe nach miteinander verglichen. Ist der Name in Y$(0) bereits "kleiner" als der, der momentan in Y$(I) gespeichert ist, so wird nach Zeile 110 verzweigt und die Laufvariable um 1 erhöht.

Ist allerdings der Name in Y$(0) "größer" als der, der sich zur Zeit in Y$(I) befindet, so wird jetzt Y$(0) der Name von Y$(I) zugeordnet. Hat die Laufvariable den Wert 6 erreicht, so befindet sich in Y$(0) der gesuchte Name. Schließlich wird dieser durch die Zeile 110 mit einem entsprechenden Kommentar ausgegeben.
Der zweite Teil der Aufgabe war, zugegeben, nicht ganz einfach, aber ein wenig knobeln macht ja nebenbei auch Spaß, oder nicht?

Noch ein Wort zum Vergleich von Zeichenketten. Werden zwei Zeichenketten auf größer oder kleiner verglichen, so wird jeder einzelne Buchstabe der beiden Strings miteinander verglichen. Ausschlaggebend sind dabei die ASCII-Werte der einzelnen Zeichen. Damit ist auch zu erklären, daß der String "HAND" kleiner als der String "HANS" ist, da der ASCII-Wert von D = 68 und von S = 83 ist.

2.
```
10 REM ZAHLEN EINLESEN
20 DIM X(6)
30 FOR I=1 TO 6
40 X(I)=INT(100*RND(1))+50
50 NEXT I
60 REM GROESSTE ZAHL SUCHEN
70 X(0)=X(1)
80 FOR I=2 TO 6
90 IF X(0) > = X(I) THEN 110
100 X(0) = X(I)
110 NEXT I
120 PRINT"GROESSTE ZAHL ";X(0)
130 END
```

Dieses Programm hat von der Struktur her den gleichen Aufbau wie das Programm aus Aufgabe 1. Hatten Sie also die Lösung von Aufgabe 1, so hatten Sie auch gleichzeitig die Lösung der Aufgabe 2. Der Unterschied besteht lediglich in der Art des Feldes (numerisch) und der Zufallszahlengenerierung in Zeile 40. Die Vergleiche zur

Auffindung der größten Zahl basieren auf dem gleichen Prinzip wie in Aufgabe 1. In Zeile 90 wird nur auf größer/gleich untersucht, da wir ja die größte Zahl ausfindig machen wollten.

Das war soweit die Lösung der Aufgabe 2. Kommen wir nun zur Aufgabe 3, wo Sie die Zuordnungsregel für das Feld finden mußten.

```
3. 10 REM ZUORDNUNGSREGEL FUER FOLGE
   20 DIM X(6)
   30 FOR I=1 TO 6
   40 X(I)=I*I-I
   50 NEXT I
   60 REM AUSGABE FELD
   70 FOR I=1 TO 6
   80 PRINT X(I)
   90 NEXT I
   100 END
```

Die Werte in dieser Aufgabe wurden durch die Multiplikation der Laufvariablen I mit sich selbst und anschließender Subtraktion von I gebildet. Diese Lösung ist natürlich nur als Vorschlag gedacht. Sollten Sie auf andere Art und Weise zum gleichen Ergebnis gekommen sein, so ist Ihre Lösung selbstverständlich nicht falsch. Der Aufbau des Programms dürfte eigentlich einsichtig sein.

Da diese Aufgaben nicht gerade einfach waren, dürfen Sie sich bei korrekter Lösung ein wenig ausruhen, bevor Sie das nächste Kapitel über die "mehrdimensionalen Felder" in Angriff nehmen.

4.1.3 MEHRDIMENSIONALE FELDER

Bisher haben wir nur eindimensionale Felder verwendet. Wir hatten diese Felder mit einer Liste verglichen, in der die einzelnen Daten übereinander geschrieben waren. Nun bestehen diese Listen meistens nicht nur aus übereinander oder nebeneinander geschriebenen Daten, sondern bilden eine Kombination aus beiden. Sie setzen sich also aus ZEILEN und SPALTEN zusammen. Stellen Sie sich vor, Sie wollten das Programm, das die Vornamen in einem Feld abgelegt hat, in der Art erweitern, daß die Nachnamen und die Geburtsdaten ebenfalls über den gleichen Index abrufbar sind.

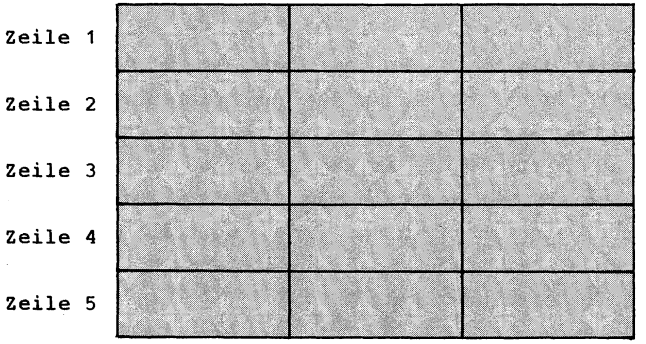
Nichts leichter als das, werden Sie sagen. Wir bilden drei Felder, in denen die Daten entsprechend abgespeichert werden können. Feld A$(X) soll die Vornamen, Feld B$(X) die Nachnamen und Feld C$(X) die Geburtsdaten enthalten. Damit haben Sie dann drei Felder mit unterschiedlichen Namen erzeugt. Den Zweck würden diese Felder unter Umständen erfüllen, jedoch ist die Verwaltung dieser drei Felder innerhalb des Programms recht umständlich. Wir sind hier wieder an einer ähnlichen Stelle angelangt wie bei der Einführung der eindimensionalen Felder.

Warum sollte es also nicht möglich sein, statt drei einzelner Felder nur ein Feld zu erstellen, das die gleichen Bedingungen erfüllt? Die Lösung unseres Problems heißt:

MEHRDIMENSIONALE FELDER

Für unsere Zwecke benötigen wir ein zweidimensionales Feld, da wir den einzelnen Vornamen in der gleichen ZEILE die dazugehörigen Daten für Nachnamen und Geburtsdatum zuordnen wollen. Unser Feld benötigt also Zeilen und Spalten. Die Struktur eines solchen Feldes soll wieder das folgende Schaubild verdeutlichen:

NAME DES FELDES = A$



Die DIM-Anweisung für dieses Feld lautet:

DIM A$(5,3)

Damit wird also ein Feld mit 5 Zeilen und 3 Spalten generiert. Führen Sie die DIM-Anweisung nicht aus und benutzen eins der Elemente aus diesem Feld, so erzeugt der Rechner automatisch ein zweidimensionales Feld der Größe (10,10). Es lohnt sich also, die DIM-Anweisung auch bei kleineren mehrdimensionalen Feldern anzuwenden, da sonst sehr viel Speicherplatz verschenkt würde. Wie wird nun ein solches Feld benutzt?

Nun, stellen Sie sich vor, Sie wollen die ersten drei Spalten der ersten Zeile dieses Feldes mit Daten auffüllen. Sie könnten dazu folgende Programmzeile verwenden:

```
    .
    .
40 INPUT"VORNAME,NAME,GEBOREN";A$(1,1),A$(1,2),A$(1,3)
    .
    .
```

Dadurch wird die Eingabe von drei Elementen verlangt (Vorname, Name, Geburtsdatum), die durch Kommata abzutrennen sind. Diese Zeile wirkt jedoch innerhalb eines Programms unübersichtlich, so daß man sich entschließen sollte, insgesamt drei INPUT-Befehle auf drei verschiedene Programmzeilen zu verteilen. Das könnte wie folgt aussehen:

```
    .
    .
40 INPUT"VORNAME";A$(1,1)
50 INPUT"NAME";A$(1,2)
60 INPUT"GEBOREN";A$(1,3)
    .
    .
```

Diese Anordnung ist weitaus übersichtlicher und hilft Eingabefehler zu vermeiden, da bei der Eingabe für jedes Element eine Bildschirmzeile zur Verfügung steht.

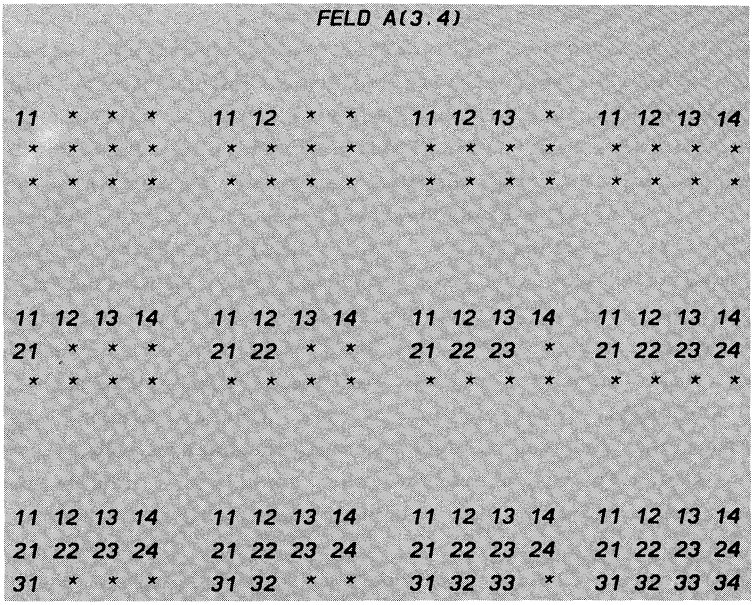
Sie werden sich bestimmt fragen, warum man die Eingabe nicht in einer Schleife realisiert, wo nacheinander der Vorname, der Name und das Geburtsdatum mit nur einem einzigen INPUT-Befehl eingelesen werden.
In unserem Beispiel benutzt jeder INPUT-Befehl einen eigenen Kommentar, der den einzugebenden Wert näher beschreibt. Daher können in diesem Fall die drei INPUT-Befehle nicht durch einen INPUT-Befehl ersetzt werden und somit kann auch keine Schleife Verwendung finden. Wir wollen allerdings genau 6mal den Vornamen, den Nachnamen und das Geburtsdatum einlesen lassen. Dafür können wir eine FOR...NEXT-Schleife verwenden, wie das folgende Beispiel zeigen soll:

```
10 REM 6 VORNAMEN, NAMEN UND
20 REM GEBURTSDATEN EINLESEN
30 DIM A$(6,3)
40 FOR I=1 TO 6
50 INPUT"VORNAME";A$(I,1)
60 INPUT"NAME";A$(I,2)
70 INPUT"GEBOREN";A$(I,3)
80 NEXT I:END
```

Solche ähnlichen Programmteile werden Sie überall bei kleineren Dateiverwaltungsprogrammen finden. Nun werden aber mehrdimensionale Felder nicht nur per Tastatur mit Daten versorgt, sondern auch durch Daten aus DATA-Zeilen, die mit dem READ-Befehl in das Feld eingelesen werden. Sie kennen diese Technik schon von den eindimensionalen Feldern. Bei Programmen dieser Art können wir verschachtelte FOR...NEXT-Schleifen verwenden. Das folgende Beispiel zeigt uns, wie ein zweidimensionales Feld der Größe (3,4) mit Daten aus DATA-Zeilen gefüllt wird.

```
10 REM FELD AUS DATAZEILE LADEN
20 DIM X(3,4)
30 FOR Z=1 TO 3
40 FOR S=1 TO 4
50 READ X(Z,S)
60 NEXT S,Z
70 DATA 11,12,13,14,21,22,23,24,31,32,33,34
80 REM AUSGABE FELD
90 PRINT"FELD ANZEIGEN (J/N) ?"
100 GET A$:IF A$="" THEN 100
110 IF A$ < > "J" THEN 150
120 FOR Z=1 TO 3
130 PRINT X(Z,1);X(Z,2);X(Z,3);X(Z,4)
140 NEXT Z
150 END
```

Wir haben hier ein Beispiel, wo durch Einsatz zweier ineinandergeschachtelter FOR...NEXT-Schleifen ein Feld mit 3 Zeilen und 4 Spalten gefüllt wird. Die innere Schleife bewirkt, daß alle Elemente einer Zeile nacheinander mit Daten gefüllt werden. Ist diese Schleife abgearbeitet, so bewirkt die äußere Schleife, daß nacheinander alle 3 Zeilen erreicht werden. Wie sich das Feld nach und nach mit Daten füllt, soll das nachstehende Bild verdeutlichen.



Wollen Sie das Feld ausgeben lassen, so brauchen Sie nur die J-Taste zu betätigen. Das Feld wird in der Form ausgegeben, wie Sie es im Bild oben sehen. Diese Ausgabe wurde mit der Zeile 130 erreicht. Es werden alle 4 Spalten einer Zeile auf einmal ausgegeben. Nur die Ausgabe aller 3 Zeilen wurde mit einer FOR...NEXT-Schleife bewirkt. Eine andere Möglichkeit, sich das Feld auf diese Art ausgeben zu lassen, soll Ihnen das nächste Beispiel aufzeigen. Dabei wurde auf die ersten Programmzeilen verzichtet.

```
    .
    .
80 REM AUSGABE FELD
90 PRINT"FELD ANZEIGEN (J/N) ?"
100 GET A$:IF A$="" THEN 100
110 IF A$ < > "J" THEN 150
120 FOR Z=1 TO 3
130 FOR S=1 TO 4
140 PRINT A(Z,S);:ZZ=ZZ+1
150 IF ZZ=4 THEN ZZ=0:PRINT:PRINT
160 NEXT S,Z
170 END
```

Ändern Sie Ihr Programm in dieser Art ab, so können Sie zwei verschachtelte FOR...NEXT-Schleifen benutzen. Dabei bewirkt Zeile 140 durch das Semikolon, daß die Werte der einzelnen Elemente hintereinander ausgegeben werden. Um nun die Struktur des Feldes auf dem Bildschirm aufzubauen, muß nach Ausgabe der vier Elemente einer Zeile ja eine neue Bildschirmzeile begonnen werden. Daher wurde ein zusätzlicher Zähler (ZZ) benötigt, der registriert, wie oft eine Ausgabe mit PRINT bereits erfolgte. Zeile 150 fragt ab, ob dieser Zähler den Wert 4 angenommen hat. Ist dies der Fall, wird der Zähler zurück auf Null gesetzt. Danach erfolgt ein zusätzlicher PRINT-Befehl, der bewirkt, daß das nächste Element am Anfang der nächsten Bildschirmzeile ausgegeben wird. Sie können die Ausgabe übersichtlicher gestalten, indem Sie zwei Print-Befehle, wie in unserem Beispiel, verwenden.

Hier noch ein kleiner Tip: Wollen Sie die Ausgabe genau verfolgen, so können Sie eine zusätzliche Zeitschleife einbauen. Dazu geben Sie einfach folgende Programmzeile ein:

```
155 FOR N=1 TO 1000:NEXT N
```

Dadurch wird nach Ausgabe eines Feldelements eine Pause von ca. 1 Sekunde eingelegt. Soweit die Erklärung zu dieser zweiten Möglichkeit der Ausgabe eines zweidimensionalen Feldes.

Kommen wir nun noch einmal auf das erste Beispiel zurück (vgl. Seite 155). Dort wurde gesagt, daß solche Programmteile bei Dateiverwaltungsprogrammen in ähnlicher Art anzutreffen sind. In diesem Beispiel wurde angenommen, daß nur die Daten von 6 Personen aufzunehmen waren. Dieser Fall ist aber nicht die Regel. Meistens steht nämlich nicht die Anzahl der Personen fest, sondern nur die Anzahl der personenbezogenen Daten, d.h. Sie wollen nur Name, Vorname und Telefonnummer aufnehmen. Wollen Sie also ein Programm schreiben, welches Ihnen Ihr Telefonregister ersetzen soll, so kennen Sie nur einen Wert des zweidimensionalen Feldes genau, nämlich die

personenbezogenen Daten. Den anderen Wert, also die Anzahl der aufzunehmenden Personen, müssen Sie vorher grob abschätzen. Nehmen wir an, daß Ihr Bekanntenkreis ca. 100 Personen umfaßt. Die DIM-Anweisung DIM X$(120,3) dürfte dann in diesem Fall vollkommen ausreichend sein. Schauen wir uns hierzu ein Beispiel an.

```
10 REM DATEN EINLESEN
20 DIM X$(120,3)
30 PRINT CHR$(147)
40 Z=Z+1
50 INPUT"VORNAME";X$(Z,1)
60 PRINT
70 INPUT"NAME";X$(Z,2)
80 PRINT
90 INPUT"TELEFONNR.";X$(Z,3)
100 PRINT
110 PRINT"WOLLEN SIE WEITERE DATEN "
120 PRINT"EINGEBEN (J/N) ?"
130 GET A$:IF A$="" THEN 130
140 IF A$ = "J" THEN 30
150 END
```

Daß dieses Problem eleganter gelöst werden kann, versteht sich von selbst. Hier geht es zunächst nur darum, daß das Prinzip verstanden wird. Dadurch, daß wir keine FOR...NEXT-Schleife verwenden, müssen wir den Zähler in Zeile 40 einfügen, um den Index bei jeder neuen Eingabe um eins zu erhöhen. Danach erfolgt über die INPUT-Befehle die Eingabe der Daten, die dann den entsprechenden Feldelementen zugeordnet werden. Sollen weitere Daten eingegeben werden, verzweigt das Programm nach Zeile 30. Ansonsten wird das Programm beendet. Hier könnte man sich bei einem kompletten Datenverwaltungsprogramm einen Sprung zum Hauptmenü vorstellen, wo der Anwender dann weitere Programmpunkte anwählen kann.

Es wurde hier in der Zeile 140 der Vergleich mit IF...THEN vorgenommen, um zu überprüfen, ob weitere Daten eingegeben werden sollen. Sie sehen damit den Unterschied zum ersten Programm, wo diese Aufgabe von einer FOR...NEXT-Schleife übernommen wurde. Zur Erinnerung: Wir

verwenden IF...THEN genau dann, wenn die Anzahl der einzugebenden Daten nicht bekannt ist.

Diese Beispiele sollten vorerst genügen, um Ihnen den Umgang mit mehrdimensionalen Feldern näher zu bringen. Der COMMODORE 64 hat theoretisch die Möglichkeit, Felder mit bis zu 255 Indizes zu erstellen. Das bedeutet, daß nicht nur zwei-, drei- oder gar vierdimensionale Felder erzeugt werden könnten, sondern Felder mit bis zu 255 Dimensionen. Allerdings nur theoretisch, denn der zur Verfügung stehende Speicherplatz ist eben begrenzt. Außerdem kann man sich ein dreidimensionales Feld noch vorstellen, z.B. als Würfel, aber bei vier- oder gar fünfdimensionalen Feldern hört das Vorstellungsvermögen schon auf.

Schauen wir uns trotzdem zum dreidimensionalen Feld noch ein Beispiel an. Wir wollen die Indizes wie folgt definieren:

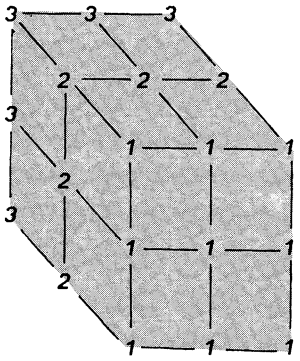
X = ZEILE

Y = SPALTE

Z = TIEFE

Es soll nun ein dreidimensionales Feld erzeugt werden, mit dem ein Würfel dargestellt wird, der eine Kantenlänge von 3 aufweist. Sie können sich das dreidimensionale Feld am besten so vorstellen, daß man, wie in unserem Beispiel, drei zweidimensionale Felder hintereinandergefügt hat. Das Schaubild auf der nächsten Seite soll dies veranschaulichen.

Mit etwas räumlichem Vorstellungsvermögen können Sie sich dabei einen Würfel mit der Kantenlänge 3 vorstellen.



Um dieses Feld zu erzeugen, muß die DIM-Anweisung wie folgt aussehen:

DIM W(X,Y,Z)

bzw. konkret auf unser Beispiel bezogen:

DIM W(3,3,3)

Damit besitzt das Feld insgesamt 27 einzelne Elemente (3*3*3=27) bzw. eigentlich 64 Elemente (4*4*4=64), wenn wir das Nullelement mit hinzurechnen.

Ihre Aufgabe besteht nun darin, ein Programm zu schreiben, das dieses Feld mit Daten auffüllt. Die Anzahl der Daten ist bekannt. Gehen Sie dabei so vor, daß zuerst die Spaltenwerte einer Zeile, danach die Zeilen selbst (also wie beim zweidimensionalen Feld) und schließlich die einzelnen "Scheiben" des Feldes aufgefüllt werden. Schreiben Sie das Programm so, daß genau das obenstehende Feld erzeugt wird. Damit sind die Wertzuweisungen der einzelnen Scheiben gemeint. Sie brauchen nicht den räumlichen Effekt bei der Ausgabe zu erzielen.

Ihr Programm sollte in etwa so aussehen:

```
10 REM 3-D-FELD
20 DIM W(3,3,3)
30 FOR Z=1 TO 3
40 FOR Y=1 TO 3
50 FOR X=1 TO 3
60 READ W(X,Y,Z)
70 NEXT X,Y,Z
80 DATA 1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,
3,3,3,3,3
90 REM AUSGABE FELD
100 FOR Z=1 TO 3
110 FOR Y=1 TO 3
120 FOR X=1 TO 3
130 PRINT W(X,Y,Z);:ZZ=ZZ+1
140 IF ZZ=3 THEN ZZ=0:PRINT
150 NEXT X,Y,Z
160 END
```

Die DIM-Anweisung in Zeile 20 sollten Sie nicht vergessen haben. Was nämlich für ein zweidimensionales Feld gilt, gilt für ein dreidimensionales Feld erst recht. Vergessen Sie die DIM-Anweisung, erzeugt der Rechner ein Feld von 10*10*10=1000 bzw. 11*11*11=1331 Elementen! Sie benötigen aber nur 27. Das wäre eine recht große Verschwendung von Speicherplatz. In den Zeilen 30 bis 50 werden drei FOR...NEXT Schleifen miteinander verknüpft. Die äußere Schleife bewirkt das Auffüllen der einzelnen Scheiben des Würfels. Dementsprechend sind die beiden anderen Schleifen zu interpretieren. In Zeile 70 schließt ein NEXT alle drei Schleifen auf einmal. Achten Sie dabei immer darauf, daß die Laufvariablen hinter dem NEXT in der richtigen Reihenfolge gesetzt werden. Die Ausgabe des Feldes sollte zwar nicht in der räumlichen Darstellung erfolgen, aber die einzelnen "Scheiben" des Würfels müßten schon erkennbar sein. Eine Ausgabe, in der alle Werte hinter- oder untereinander aufgelistet werden, wäre nicht im Sinne der Aufgabe gewesen.

Damit wollen wir das Kapitel über die mehrdimensionalen Felder abschließen. Sie sollten nun in der Lage sein, ein- oder mehrdimensionale Felder in Ihren Programmen verwalten zu können. Dabei sind die ein- bzw. zweidimensionalen Felder diejenigen, die in Programmen die meiste Anwendung finden. Das größte Anwendungsgebiet haben diese Felder wohl bei der Dateiverwaltung.

Im nächsten Kapitel geht es nun um die Benutzung von Unterprogrammen, die immer dann sinnvoll einzusetzen sind, wenn eine Folge von Anweisungen mehrmals durchgeführt werden muß.

Was ist eigentlich ein Unterprogramm? Nun, wie bereits erwähnt, haben Sie die Möglichkeit, Programmteile, die innerhalb eines Programms öfters benutzt werden, als Unterprogramm zu gestalten, welches Sie dann je nach Bedarf aufrufen können. Ein Unterprogramm ist demnach ein eigenständiger Programmteil, der meistens am Ende oder am Anfang des eigentlichen Hauptprogramms liegt. Der Aufruf geschieht durch den Befehl:

GOSUB (ZEILENNUMER)

GOSUB ist die Abkürzung für "GOTO SUBROUTINE", was soviel heißt wie "GEHE INS UNTERPROGRAMM". Die Zeilennummer bezeichnet die Stelle, an der das Unterprogramm beginnt. Trifft das Programm auf diesen Befehl, so merkt es sich die Zeilennummer, von der aus es in das Unterprogramm gesprungen ist. Es verzweigt dann zu der Zeilennummer des Unterprogramms und fährt dort mit der Programmausführung fort, bis es auf den Rücksprungbefehl

RETURN

trifft. Dann wird mit der Anweisung im Programm fortgefahren, die dem GOSUB-Befehl folgt. Trifft das Programm auf den RETURN-Befehl, ohne vorher den GOSUB-Befehl erhalten zu haben, bricht das Programm mit der Fehlermeldung

?RETURN WITHOUT GOSUB ERROR IN (Zeilennummer)

ab. Immer wenn Sie ein Unterprogramm aufrufen wollen, müssen Sie also den GOSUB-Befehl benutzen. Oft wird allerdings der Fehler gemacht, daß mit dem GOTO-Befehl einfach in ein Unterprogramm gesprungen wird. Das geschieht häufig nach Vergleichen mit IF...THEN, wie das folgende Beispiel demonstrieren soll.

```
110 IF A < 1 THEN GOTO 130   !!! F E H L E R !!!
120 GOTO 90
130 REM UNTERPROGRAMM
140 A=A+1
150 RETURN
```

In diesem Beispiel wird also von Zeile 110 mit GOTO in ein Unterprogramm gesprungen, und zwar für den Fall, daß A kleiner als 1 ist. Bei dieser fehlerhaften Anwendung des GOTO-Befehls in Verbindung mit einem Unterprogramm würde das Programm in diesem Fall mit der Fehlermeldung

```
?RETURN WITHOUT GOSUB ERROR IN 150
```

abbrechen. Die richtige Schreibweise der Zeile 110 müßte so aussehen:

```
110 IF A < 1 THEN GOSUB 130
```

Ein weiterer beliebter und zugleich schwer zu erkennender Fehler bei der Benutzung von Unterprogrammen wird im folgenden Beispiel gezeigt:

```
10 REM FEHLER IM UNTERPROGRAMM
20 PRINT
30 PRINT CHR$(18) Z
40 GOSUB 70
50 Z=Z+1:GOTO 20
60 END
70 REM UNTERPROGRAMM
80 FOR I=1 TO 25
90 PRINT I;
100 IF I >= 15 THEN 50
110 NEXT I
120 RETURN
```

Geben Sie dieses Programm ein und starten Sie es. Sie werden feststellen, daß nach sechsmaligem Durchlaufen des Unterprogramms der gesamte Programmablauf mit der Fehlermeldung

?OUT OF MEMORY ERROR IN 100

abgebrochen wird. Der Fehler liegt hier nicht im Aufruf, sondern im Verlassen des Unterprogramms. Das Programm erzeugt nach dem Start eine Leerzeile (Zeile 20). Danach wird der aktuelle Wert der Variablen Z - Z dient als Zähler für die Anzahl der Durchläufe des Unterprogramms - in reverser Schrift (CHR$(18)) ausgegeben. In Zeile 40 erfolgt der Aufruf des Unterprogramms mit GOSUB 70. Das Programm springt ins Unterprogramm nach Zeile 70 und beginnt mit der Ausführung der FOR...NEXT-Schleife. Zeile 90 gibt den Wert der Laufvariablen I aus.

In der Zeile 100 wurde dann sogleich gegen 2 Regeln verstoßen. Erstens sollte man eine FOR...NEXT-Schleife nicht mit GOTO verlassen, da es auch hier zu Problemen mit der rechnerinternen Verwaltung dieser Schleifen kommen kann. Zweitens, und das ist in diesem Fall der eigentlich schwerwiegende Fehler, darf man aus einem Unterprogramm nicht heraus, ohne den Befehl RETURN zu benutzen. In Zeile 100 wurde jedoch ein Sprung aus dem Unterprogramm nach Zeile 50 verlangt für den Fall, daß I größer oder gleich 15 ist. Statt "THEN 50" hätte dort "THEN RETURN" stehen müssen.

Innerhalb eines Unterprogramms können Sie durchaus mit dem GOTO-Befehl hin- und her springen, wie sie es schon von normalen Programmen gewohnt sind. Sie dürfen nur nicht mit GOTO das Unterprogramm verlassen. In unserem Beispiel brach das Programm bereits nach dem 6. Durchlauf des Unterprogramms ab. Haben Sie bei großen Programmen einen Fehler dieser Art eingebaut, so kann es durchaus geschehen, daß das Programm zunächst dem Anschein nach einwandfrei läuft. Plötzlich und unerwartet bekommen Sie dann die o.a. Fehlermeldung ausgeworfen. Deswegen sollten Sie beim Umgang mit Unterprogrammen darauf achten, daß Sie sich diesen Fehler auf jeden Fall ersparen. Übrigens, eine gut durchdachte vorherige Planung des Programmablaufs hilft solche Fehler zu vermeiden.

Kommen wir nun zu einer praktischen Anwendung von

Unterprogrammen. Sie erinnern sich bestimmt noch an das Programm "Rechenlehrgang". Untersuchen Sie das Programmlisting auf den Seiten 106 ff einmal auf Programmteile, die sich im Programm wiederholen. Sie werden wahrscheinlich feststellen, daß man einige Programmteile durchaus in einem Unterprogramm zusammenfassen könnte. Die Zeilen, die sich oft wiederholen, sind im folgenden noch einmal aufgeführt:

```
230 PRINT CHR$(147)
240 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
250 PRINT
260 PRINT TAB(10)"FUER DIE ADDITION EIN."
270 PRINT
290 PRINT TAB(10);:INPUT"GROESSTE";GR
299 REM
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM BERECHNUNG ERGEBNIS
331 REM
340 ER=A1+A2
350 PRINT CHR$(147)
360 PRINT
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2 "= ";
380 INPUT ES
390 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG":F=0:
GOTO450
400 PRINT:PRINTTAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FORI=0 TO 2000:NEXTI
460 PRINTTAB(5)"DAS ERGEBNIS LAUTET" ER
470 FORI=0 TO 3000:NEXTI
480 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
490 INPUT A$
500 IF A$="J" THEN F=0:GOTO 300
510 GOTO 10
```

Diese sich oft wiederholenden Programmzeilen kann man nun in bestimmten Blöcken zusammenfassen. Als erstes würden sich hier die Zeilen 230 bis 290 anbieten, da die Eingabe einer größten Zahl bei jeder Rechenart verlangt wird. Das Problem bei diesem Programmteil liegt darin, daß jede Rechenart, für die eine größte Zahl verlangt wird, ja einzeln genannt werden muß. Die Ausgabe

GEBEN SIE DIE GROESSTE ZAHL

FUER DIE ADDITION EIN.

muß also innerhalb des Unterprogramms flexibler in Bezug auf die Rechenart gestaltet werden.

Da wir im Menue über die eingegebenen Zahlen auf die einzelnen Programmteile Addition, Subtraktion usw. mit ON X GOTO zugreifen, bietet es sich an, dieses X als Index zu gebrauchen. Wozu wir das machen, werden wir gleich sehen. Wir können am Anfang des Programms ein Feld erstellen, das die Begriffe Addition, Subtraktion, Division und Multiplikation beinhaltet, und zwar genau in der Reihenfolge, in der diese Begriffe im Menü angelegt sind. Dabei können wir schon das Menü selbst mit dem Inhalt dieses Feldes ausgeben lassen. Schauen wir uns zunächst den abgeänderten Programmstart an.

```
10 REM      *****************
20 REM      * PROGRAMMSTART *
30 REM      *****************
40 POKE 53280,0:POKE 53281,0:PRINT CHR$(30)
50 DIM RA$(4),BE$(4)
60 FOR I=1 TO 4
70 READ RA$(I),BE$(I)
80 NEXT I
90 DATA ADDITION,+,SUBTRAKTION,-,DIVISION,/,
MULTIPLIKATION,*
100 GOTO 580
   .
   .
   .
   .
```

Die Zeile 40 bewirkt, daß die gesamte Bildschirmfarbe auf schwarz gesetzt wird. Weiterhin wird die Schriftfarbe in grün umgewandelt. In Zeile 50 werden zwei Felder, RA$ und BE$, generiert. Mit der FOR...NEXT-Schleife in Zeile 60 werden die beiden Felder geladen, und zwar werden Feld RA$ die Begriffe Addition, Subtraktion usw. zugeordnet und Feld BE$ die entsprechenden Zeichen der Rechenarten. Wieso das Programm ab Zeile 100 in die Zeile 580 springt, werden Sie nachher selbst erkennen können. Nachdem das Programm gestartet wurde, werden die Felder mit diesen Daten gefüllt. Damit wir sehen, wie Feld RA$ beim Aufbau des Menüs verwendet wird, schauen wir uns nun die Programmzeilen des geänderten Programms von Zeile 570 bis 790 an.

```
.
.
.
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 PRINT CHR$(147):F=0
610 PRINT
620 PRINT TAB(12)"RECHENLEHRGANG"
630 PRINT:PRINT
640 PRINT TAB(12)"WAEHLEN SIE:"
650 PRINT
660 PRINT TAB(12)"FUER "RA$(1)" EINE 1"
670 PRINT
680 PRINT TAB(12)"FUER "RA$(2)" EINE 2"
690 PRINT
700 PRINT TAB(12)"FUER "RA$(3)" EINE 3"
710 PRINT
720 PRINT TAB(12)"FUER "RA$(4)" EINE 4"
730 PRINT
740 PRINT TAB(12)"FUER ENDE EINE 5"
750 PRINT
760 PRINT TAB(12)"WELCHE ZAHL ?"
770 GET E$:IF E$="" THEN 770
780 IF VAL(E$) < 1 OR VAL(E$) > 5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
```

Im Gegensatz zum Programm auf Seite 106 ff werden die Menüpunkte Addition, Subtraktion usw. nicht einzeln erwähnt, sondern in den Zeilen 660 bis 720 aus dem Feld RA$ ausgelesen. Der Übersichtlichkeit wegen wurden hier die Feldelemente jeweils einzeln angesprochen. Man hätte dies auch durch eine FOR...NEXT-Schleife erreichen können wie das folgende Beispiel zeigen soll.

```
  .
  .
660 FOR I=1 TO 4
670 PRINT TAB(12)"FUER ";RA$(I);"EINE";I
680 PRINT
690 NEXT I
  .
  .
```

Wird nun in Zeile 770 mit GET ein Zeichen eingelesen, wird zunächst in Zeile 780 überprüft, ob es sich um ein zulässiges Zeichen handelt (zwischen 1 und 5). Ist das der Fall, so wird mit VAL(E$) der numerische Wert ermittelt und dieser der Variablen P zugeordnet. Über P wird dann in die entsprechenden Zeilen weiter verzweigt. Wurde die Addition angewählt, so hat P jetzt den Wert 1. Damit wird nach Zeile 800 verzweigt. Dort beginnt der Programmteil der Addition. Schauen wir uns auch hierfür die Programmzeilen an.

```
  .
  .
800 REM ************
810 REM * ADDITION *
820 REM ************
830 GOSUB 110
840 GOSUB 310
850 ER=A1+A2
860 GOSUB 390
870 IF A$="J" THEN 840
880 GOTO 580
  .
  .
```

In Zeile 830 wird jetzt das erste Unterprogramm aufgerufen. Das ist der Teil, in dem die höchste Zahl

eingegeben wird, mit der gerechnet werden soll. Nachfolgend wird dieses erste Unterprogramm aufgeführt.

```
  .
  .
110 REM     ******************
120 REM     * UNTERPROGRAMME *
130 REM     ******************
140 REM ************************
150 REM * EINGABE HOECHSTE ZAHL *
160 REM ************************
170 PRINT CHR$(147):A$="":B$=""
180 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 GET A$:IF A$="" THEN 240
250 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 240
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=VAL(B$)
290 RETURN
  .
  .
```

Der erste Teil von Zeile 170 dürfte inzwischen bekannt sein. Warum werden aber in der zweiten Hälfte die Variablen A$ und B$ mit NICHTS aufgefüllt? Nun, da B$ in Zeile 260 mit A$ verknüpft wird, schleppt B$ diesen Inhalt immer mit. Wird die Rechenart gewechselt und damit dieses Unterprogramm erneut aufgerufen, so würden zum alten Inhalt der Variablen die neu eingelesenen Werte hinzuaddiert. Damit hätte man dann schon eine sechsstellige Startzahl für die Berechnung der Zufallszahlen. Daher werden die beiden Variablen zu Beginn des Unterprogramms auf "Null" bzw. NICHTS gesetzt.

Die Eingabe der drei Ziffern mittels GET wurde schon auf Seite 122 ff angesprochen. In unserem jetzigen Beispiel wurden allerdings ein paar kleine Änderungen notwendig. Zum einen können hier jeweils nur Ziffern zwischen 0 und 9 eingegeben werden (Zeile 250). Dann wurde mit PRINT A$; in Zeile 260 erreicht, daß man erkennen kann, was gerade eingegeben wurde. Wollen Sie nur einen zweistelligen Wert benutzen, so müssen Sie zuerst eine Null und dann die restlichen Ziffern eingeben.

Die Zeile 200 ist nun recht interessant. Hier wird nämlich der Wert von P als Index benutzt, um den Begriff der passenden Rechenart aus dem Feld RA$ auszulesen. Sie sehen, wie sinnvoll solche indizierten Variablen eingesetzt werden können. Das setzt natürlich voraus, daß die Daten in der richtigen Reihenfolge in diesem Feld abgelegt werden. Haben wir also die Addition angewählt, hat P den Wert 1. In RA$(1) steht ja der Begriff der Addition. Aus diesem Grunde wurden auch die Zeichen für die Rechenarten mit dem gleichen Index in einem anderen Feld abgelegt. Mit diesem kleinen Trick haben wir es also geschafft, unser Unterprogramm auf jede der vier Rechenarten abzustimmen.

Die nächste Programmzeile bei der Addition (Zeile 840) ruft das Unterprogramm für die Erzeugung der Zufallszahlen auf. Das ist das kleinste und einfachste Unterprogramm. Trotzdem sollen die Zeilen der Vollständigkeit halber erwähnt werden.

```
  .
  .
300 REM *************************
310 REM * ERZEUGEN ZUFALLSZAHLEN *
320 REM *************************
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
  .
  .
```

Zu diesem Unterprogramm brauchen wohl keine näheren Erklärungen abgegeben werden.

Wichtiger ist das nächste Unterprogramm "Aufgabenstellung". Im folgenden wieder die Programmzeilen zu diesem Unterprogramm:

```
.
.
360 REM ********************
370 REM * AUFGABENSTELLUNG *
380 REM ********************
390 PRINT CHR$(147)
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=ER THEN PRINT:PRINTTAB(10)"RICHTIG !":F=0:
GOTO 500
440 PRINT:PRINTTAB(10)"FALSCH !"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
470 IF F <= 2 THEN 390
480 PRINT
490 FOR I=0 TO 2000:NEXT I
500 PRINT:PRINTTAB(5)"DAS ERGEBNIS LAUTET"ER
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINTTAB(5)"NOCH EINE AUFGABE  J/N";
550 INPUT A$
560 RETURN
.
.
```

Die Zeilen 390 und 400 bedürfen keiner Erklärung. Interessant ist in diesem Unterprogramm die Zeile 410. Hier wird die Fragestellung der Aufgaben formuliert. Zunächst werden die beiden Worte

WIEVIEL ERGIBT

ausgegeben. Danach werden nacheinander die Werte der Variablen A1, BE$(P) und A2 ausgegeben. Diesen Variablen folgt noch das Gleichheitszeichen. Die Zeile schließt dann mit dem Semikolon, damit der INPUT-Befehl aus

Programmzeile 420 in der gleichen Bildschirmzeile ausgegeben werden kann. Die allgemeine Form dieser Bildschirmausgabe könnte man so formulieren:

WIEVIEL ERGIBT A1 + (bzw.-,*,/) A2 = ?

Es wird also in Abhängigkeit der gewählten Rechenart über den Index P wieder das passende Rechenzeichen BE$(P) mit ausgegeben. Diese Programmzeile soll ja nun für das gesamte Programm bzw. für alle Rechenarten allgemeingültig sein, d.h. wir müssen gewährleisten, daß, unabhängig von der Rechenart, sich in den Variablen A1 und A2 immer die "richtigen" Werte befinden. Damit unser Unterprogramm für alle Rechenarten gültig bleibt, müssen wir also die Werte der Variablen A1 und A2 in den einzelnen Programmpunkten der Addition, Subtraktion usw. aufbereiten.

Die restlichen Programmzeilen bis Zeile 540 sollten Ihnen noch vom ersten "Rechenlehrgang" her bekannt sein. Zeile 550 übernimmt in A$ die Antwort auf die Frage aus Zeile 540. Zeile 560 beendet das Unterprogramm mit dem RETURN-Befehl. Der Inhalt der Variablen A$ wird mit in die entsprechenden Programmteile der einzelnen Rechenarten übernommen.

Damit hätten wir den Programmteil, in dem sich die Unterprogramme befinden, abgeschlossen. Sicherlich haben Sie bemerkt, daß wir die Unterprogramme an den Anfang des Programms gelegt haben. In vielen Büchern findet man den Ratschlag, die Unterprogramme an das Ende des Hauptprogramms zu legen. Diese Entscheidung wurde sicherlich im Hinblick auf eine später anzulegende Programmbibliothek getroffen. Man hat dann dort seine Unterprogramme nach Zeilennummern sortiert vorliegen, d.h. das Unterprogramm für die Rundung einer beliebigen Zahl liegt ab Zeilennummer 50000 usw. Wird nun ein neues Programm geschrieben, kann man diese Routinen über einen speziellen Befehl (MERGE) nachladen und an das Programm anhängen. Dieser MERGE-Befehl wird meistens durch eine BASIC-Spracherweiterung zur Verfügung gestellt.

Was spricht aber nun dafür, seine Unterprogramme mit niedrigeren Zeilennummern zu versehen und so an den Anfang seines Programmes zu setzen? Nun, es ist wohl ziemlich gleich, ob mit dem MERGE-Befehl ein Unterprogramm an ein Programm angehängt oder ob ein Programm an Unterprogramme gehängt wird. Damit wäre die Situation schon mal patt.

Bei einem Unterprogrammaufruf springt der Rechner jedoch erst an den Anfang des Programms und beginnt von dort aus mit der Suche nach der Zeilennummer, in der das Unterprogramm beginnt. Liegen die Unterprogramme nun am Anfang eines Programms, so wird die Ausführungszeit des gesamten Programms verkürzt.

Wir sprachen davon, daß in den einzelnen Programmteilen für die Rechenarten die Variablen A1 und A2 für das Unterprogramm "Aufgabenstellung" entsprechend aufbereitet werden müssen. Das trifft allerdings nur für die Subtraktion und die Division zu, da die Werte der Variablen aus der Addition ER=A1+A2 und aus der Multiplikation ER=A1*A2 direkt in das Unterprogramm übergeben werden können.

Bei der Subtraktion muß man darauf achten, daß A1 immer größer ist als A2, damit kein negativer Wert entsteht. Dies wird durch die Programmzeile 940 gewährleistet. Ist A1 kleiner als A2, werden die beiden Variablenwerte in der bekannten Weise vertauscht (Zwischenspeicherung).

Bei der Division wollen wir nur ganzzahlige Ergebnisse erhalten. Aus diesem Grund wird zunächst durch die Multiplikation der Variablen A1 und A2 das Ergebnis ER berechnet. Im früheren "Rechenlehrgang" konnten wir dann die Aufgabenstellung wie folgt stellen:

```
WIEVIEL ERGIBT ER / A1 = ?
```

Es wurde somit das vorher berechnete Ergebnis ER durch die Variable A1 dividiert, was zwangsläufig einen ganzzahligen Wert, nämlich den der Variablen A2, ergeben mußte. In unserem Unterprogramm können wir aber aus

Gründen der Allgemeingültigkeit diese Umstellung nicht direkt vornehmen. Das muß wieder im Programmteil "Division" erfolgen. Dazu dienen die folgenden Programmzeilen:

```
  .
  .
1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
  .
  .
```

Auch hier wird erst wieder das Ergebnis über die Multiplikation berechnet. Die Zeile 1050 bewirkt dann, daß die Werte für das Unterprogramm "Aufgabenstellung" richtig zugeordnet werden. Da wir nicht die Variablenbezeichnungen selbst bei der Ausgabe umstellen können, müssen wir die Werte der Variablen umordnen. Hier wird ebenfalls wieder die Technik des Zwischenspeicherns verwendet. Die Variable I erhält zunächst den Wert der Variablen ER. ER wird dann der Wert von A1 zugeordnet und A1 erhält schließlich den Wert von I, also von ER. Damit wurden diese beiden Variablenwerte ausgetauscht. Somit erhalten wir im Unterprogramm bei der Aufgabenstellung die richtige Zuordnung der Werte.
Im folgenden nun die Auflistung des kompletten Programms.

```
10 REM      *****************
20 REM      * PROGRAMMSTART *
30 REM      *****************
40 POKE 53280,0:POKE 53281,0:PRINT CHR$(30)
50 DIM RA$(4),BE$(4)
60 FOR I=1 TO 4
70 READ RA$(I),BE$(I)
80 NEXT I
90 DATA ADDITION,+,SUBTRAKTION,-,DIVISION,/,
MULTIPLIKATION,*
100 GOTO 580
```

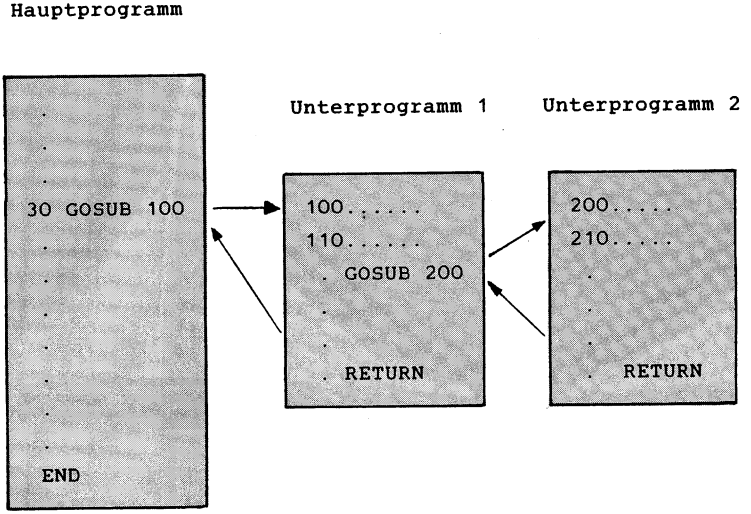
```
110 REM     ******************
120 REM     * UNTERPROGRAMME *
130 REM     ******************
140 REM *************************
150 REM * EINGABE HOECHSTE ZAHL *
160 REM *************************
170 PRINT CHR$(147):A$="":B$=""
180 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 GET A$:IF A$="" THEN 240
250 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 240
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=VAL(B$)
290 RETURN
300 REM ***************************
310 REM * ERZEUGEN ZUFALLSZAHLEN *
320 REM ***************************
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
360 REM ********************
370 REM * AUFGABENSTELLUNG *
380 REM ********************
390 PRINT CHR$(147)
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:
GOTO 500
440 PRINT:PRINT TAB(10)"FALSCH !"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
470 IF F <= 2 THEN 390
480 PRINT
490 FOR I=0 TO 2000:NEXT I
500 PRINT:PRINT TAB(5)"DAS ERGEBNIS LAUTET"ER
510 F=0
```

```
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5)"NOCH EINE AUFGABE  J/N";
550 INPUT A$
560 RETURN
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 PRINT CHR$(147):F=0
610 PRINT
620 PRINT TAB(12)"RECHENLEHRGANG"
630 PRINT:PRINT
640 PRINT TAB(12)"WAEHLEN SIE:"
650 PRINT
660 PRINT TAB(12)"FUER "RA$(1)" EINE 1"
670 PRINT
680 PRINT TAB(12)"FUER "RA$(2)" EINE 2"
690 PRINT
700 PRINT TAB(12)"FUER "RA$(3)" EINE 3"
710 PRINT
720 PRINT TAB(12)"FUER "RA$(4)" EINE 4"
730 PRINT
740 PRINT TAB(12)"FUER ENDE EINE 5"
750 PRINT
760 PRINT TAB(12)"WELCHE ZAHL ?"
770 GET E$:IF E$="" THEN 770
780 IF VAL(E$) < 1 OR VAL(E$) > 5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
800 REM ************
810 REM * ADDITION *
820 REM ************
830 GOSUB 110
840 GOSUB 310
850 ER=A1+A2
860 GOSUB 390
870 IF A$="J" THEN 840
880 GOTO 580
```

```
890 REM ***************
900 REM * SUBTRAKTION *
910 REM ***************
920 GOSUB 110
930 GOSUB 310
940 IF A1 < A2 THEN I=A1:A1=A2:A2=I
950 ER=A1-A2
960 GOSUB 390
970 IF A$="J" THEN 930
980 GOTO 580
990 REM  ************
1000 REM * DIVISION *
1010 REM ************
1020 GOSUB 110
1030 GOSUB 310
1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
1060 GOSUB 390
1070 IF A$="J" THEN 1030
1080 GOTO 580
1090 REM ******************
1100 REM * MULTIPLIKATION *
1110 REM ******************
1120 GOSUB 110
1130 GOSUB 310
1140 ER=A1*A2
1150 GOSUB 390
1160 IF A$="J" THEN 1130
1170 GOTO 580
1180 REM ********
1190 REM * ENDE *
1200 REM ********
1210 PRINT CHR$(147)
1220 END
```

Damit haben wir durch den Einsatz von drei Unterprogrammen ca. 43 Programmzeilen eingespart, und das trotz einer höheren Anzahl von REM-Zeilen! Sie sehen, daß die Verwendung von Unterprogrammen nicht nur komfortabler ist, sondern auch Speicherplatz sparen hilft. Den Sinn der Zeile 100 haben Sie jetzt bestimmt auch erkannt. Dadurch werden die Unterprogramme übersprungen und direkt ins Menü verzweigt.

Es gibt nun nicht nur die Möglichkeit, die Unterprogramme vom Hauptprogramm aus aufzurufen, sondern auch von einem Unterprogramm aus. Grafisch würde das folgendermaßen aussehen:



Das Programm trifft bei der Ausführung auf den GOSUB-Befehl in Zeile 30. Es springt dann ins Unterprogramm 1 ab Zeile 100. Im Unterprogramm 1 wird das Unterprogramm 2 mit GOSUB 200 aufgerufen. Das Unterprogramm 2 wird durchlaufen bis zum RETURN-Befehl. Danach springt das Programm zurück in das Unterprogramm 1 und fährt mit der Anweisung fort, die dem GOSUB folgt. Das Unterprogramm 1 wird ebenfalls bis zum RETURN-Befehl durchlaufen. Von da aus geht es wieder zurück ins

Hauptprogramm zur Anweisung, die dem GOSUB folgt.

Unterprogramme darf man also ähnlich "verschachteln" wie wir es von den FOR...NEXT-Schleifen her kennen.

Weiterhin haben wir bereits den Befehl ON X GOTO kennengelernt. Diese Befehlsfolge existiert auch in der Form mit GOSUB. Hier sieht die Schreibweise genauso aus, wie das folgende Beispiel zeigt:

ON P GOSUB 800,890,990

Beachten Sie nur, daß das Programm nach dem Durchlaufen der Unterprogramme an dieser Stelle mit der Programmausführung fortfährt.

Ich hoffe, daß Sie durch diese ausführliche Darstellung eines Beispiels zur Unterprogrammtechnik mit dieser ein wenig vertraut geworden sind. Damit Sie nun überprüfen können, inwieweit Sie dieses Thema verstanden haben, sollen Sie wieder eine kleine Aufgabe lösen.
Es gibt bei dem neu aufgelisteten Programm "Rechenlehrgang" eine weitere Möglichkeit, Unterprogramme einzusetzen. Ihre Aufgabe ist es nun, das Programm in dieser Form abzuändern. Sie brauchen dazu nicht das ganze Programm neu zu schreiben. Überlegen Sie vorher, welche Programmteile abgeändert werden müßten. Die neuen Unterprogramme brauchen Sie diesmal nicht an den Anfang des Hauptprogramms zu legen, da dann zuviel Schreibarbeit erforderlich wäre.
Die Lösung finden Sie auf der nächsten Seite.

```
   .
   .
780 IF VAL(E$) < 1 OR VAL(E$) > 5 THEN 770
790 P=VAL(E$)
800 IF P=5 THEN 1100
810 GOSUB 110
820 GOSUB 310
830 ON P GOSUB 880,930,990,1050
840 GOSUB 390
850 IF A$="J" THEN 820
860 GOTO 580
870 REM ************
880 REM * ADDITION *
890 REM ************
900 ER=A1+A2
910 RETURN
920 REM ***************
930 REM * SUBTRAKTION *
940 REM ***************
950 IF A1 < A2 THEN I=A1:A1=A2:A2=I
960 ER=A1-A2
970 RETURN
980 REM  ************
990 REM  * DIVISION *
1000 REM ************
1010 ER=A1*A2
1020 I=ER:ER=A1:A1=I
1030 RETURN
1040 REM ******************
1050 REM * MULTIPLIKATION *
1060 REM ******************
1070 ER=A1*A2
1080 RETURN
1090 REM ********
1100 REM * ENDE *
1110 REM ********
1120 PRINT CHR$(147)
1130 END
```

Sie werden die Lösung wahrscheinlich rasch gefunden haben. In den Programmteilen, in denen die vier Grundrechenarten ausgeführt werden, kommen insgesamt fünf Programmzeilen immer wieder vor. Das sind z.B. bei der Addition die folgenden Zeilen:

```
830 GOSUB 110
840 GOSUB 310
860 GOSUB 390
870 IF A$="J" THEN 840
880 GOTO 580
```

Der einzige Unterschied in den einzelnen Programmteilen der Rechenarten liegt also in der Berechnung selbst. Daher wurden die Zeilen 830, 840, 860, 870 und 880 hinter das Menü gesetzt. Die Abfrage von P=5 wurde separat vorgenommen, da es sich beim Programmende um kein Unterprogramm handelt. Dadurch können wir die Berechnungen für Addition, Subtraktion usw. als Unterprogramme schreiben und mit dem Befehl

ON P GOSUB

aufrufen. Auf diese Art und Weise haben wir weitere neun Programmzeilen eingespart.

Mit diesem Lösungsvorschlag wollen wir nun das Kapitel über die Unterprogramme abschließen. Die wichtigsten Dinge, die Sie bei der Anwendung von Unterprogrammen beachten müssen, seien hier noch einmal kurz zusammengefaßt:

1. Unterprogramme dienen dazu, häufig sich wiederholende Programmteile zusammenzufassen.

2. Unterprogramme werden mit GOSUB aufgerufen und mit RETURN beendet. Sie dürfen nie mit GOTO (xx) aufgerufen oder verlassen werden. INNERHALB eines Unterprogramms darf der GOTO-Befehl jedoch verwendet werden.

3. Unterprogramme dürfen geschachtelt werden in dem Sinne, daß von einem Unterprogramm (UP1) das nächste

Unterprogramm (UP2) aufgerufen wird usw. Da die Rücksprungadressen rechnerintern gespeichert werden, ist die Anzahl der geschachtelten Unterproramme jedoch begrenzt. Achten Sie auch darauf, daß jeder GOSUB-Befehl auf das entsprechende RETURN trifft.

4. Unterprogramme dürfen auch mit ON X GOSUB aufgerufen werden.

Im nächsten Kapitel wollen wir uns nun mit dem Aufbau und der Technik von Menüs beschäftigen.

4.3 MENÜTECHNIKEN

Sie wollen sicherlich, nachdem Sie in der Programmierung mit Basic fortgeschritten sind, einmal selbst größere Programme schreiben, sei es, um sich selbst irgendeine Arbeit zu erleichtern oder aber um solche Programme zum Verkauf anzubieten. Dabei ist von entscheidender Bedeutung, daß Sie solche Programme "anwenderfreundlich" gestalten. Was ist nun darunter zu verstehen?

Ein Programm soll ja nun einmal bestimmte Funktionen ausführen, d.h. bestimmte Arbeiten für Sie oder andere leisten. Dazu muß der jeweilige Anwender aber genau wissen, wie er mit dem Programm umzugehen hat, sprich welche Taste er betätigen muß, um eine bestimmte Arbeit oder Operation in Gang zu setzen oder die Antwort auf eine bestimmte Frage zu erhalten. Anwenderfreundlich heißt daher, daß das Programm von einer Person, die den eigentlichen Inhalt des Programms noch nicht kennt, trotzdem ohne großartige Erklärungen benutzt werden kann. Ohne Handbuch wird selbstverständlich kein größeres Programm auskommen. Es sollte jedoch immer gewährleistet sein, daß nicht für jede kleine Funktion erst das Handbuch zu Rate gezogen werden muß.

Bekommen Sie jetzt keinen Schrecken. Sie brauchen noch kein Handbuch zu schreiben. Vorerst reicht es vollkommen aus, wenn Sie die Prinzipien der Menütechnik beherrschen.

Der Begriff "Menü" wurde schon im Zusammenhang mit dem Programm "Rechenlehrgang" erwähnt. Hier noch einmal eine Erklärung, was man unter einem Menü versteht. Im Menü eines Programms sind einzelne Programmpunkte aufgeführt, die der Anwender durch Betätigen von Zahlen- oder Buchstabentasten anwählen kann. Sie haben bereits mit so einem Menü beim "Rechenlehrgang" gearbeitet. Die äußere Gestaltung eines solchen Menüs bleibt dem Programmierer selbst überlassen. Nach Möglichkeit sollte ein Menü aber übersichtlich, d.h. nicht zu überladen, aufgebaut sein - obwohl sich dies in Einzelfällen nicht immer realisieren läßt. Weiterhin sollte der optische Eindruck eines Menüs

ebenfalls berücksichtigt werden. Optische Trennungen durch bestimmte Zeichenfolgen sind durchaus erwünscht. Aber Vorsicht, auch hier gilt es, nicht zu übertreiben. Denken Sie grundsätzlich daran, daß beim Anwender des Programms auch der Spruch gilt: "Das Auge ißt mit!"

Wie man ein Menü aufbaut, wollen wir uns nun anhand eines konkreten Beispiels Schritt für Schritt erarbeiten. Als Beispiel wollen wir uns eine mathematische Tabelle aufbauen, die wir wie ein Nachschlagewerk benutzen können.

Zunächst müssen wir uns darüber klar werden, was dieses Programm leisten soll. Gehen wir davon aus, daß wir folgende Berechnungen benötigen:

QUADRATWURZEL
SINUS
COSINUS
NATÜRLICHER LOGARITHMUS
DEKADISCHER LOGARITHMUS

Diese fünf Berechnungen sollen also je nach Auswahl ausgeführt werden.

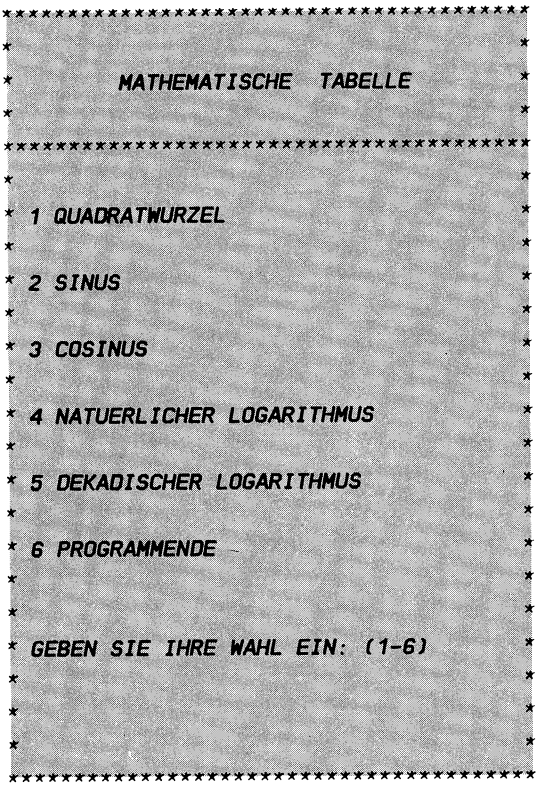
Ein Punkt fehlt noch in unserem Menü. Es handelt sich um den Punkt Programmende. Sie sollten Ihre Programme so schreiben, daß man sie auf "normalem" Wege beenden kann, und nicht dazu die RUN/STOP-Taste oder den Ein-Ausschalter betätigen muß. Damit gibt es in unserem Menü insgesamt sechs Wahlmöglichkeiten. Weiterhin benötigen wir eine Aufforderung des Programms an den Anwender, eine Zahl oder einen Buchstaben einzugeben, etwa in der Art:

```
GEBEN SIE IHRE WAHL EIN

              (1-6) ?
```

Damit wäre unser Menü von der Planung her schon fast vollendet. Nun wollen wir noch eine Überschrift und zur optischen Aufbesserung noch einen Rahmen im Menü unterbringen. Die Bildschirmfarbe soll komplett auf

schwarz gesetzt werden. Die Überschrift, so ist geplant, soll bei Ausführung des Programms bei jedem Programmteil auf dem Bildschirm vorhanden sein. Zur Erstellung der Überschrift bietet sich somit ein Unterprogramm an. Wie das Menü später auf dem Bildschirm erscheinen soll, wird im folgenden dargestellt.



In der letzten Zeile soll gleichzeitig die Eingabe der Zahl erfolgen. Für die Eingabe nehmen wir vorerst einmal den INPUT-Befehl. Später werden wir sehen, wie der GET-Befehl an dieser Stelle Verwendung finden kann. Wie das Programm für die Erzeugung dieses Menüs aussieht, wird im folgenden aufgelistet.

```
10 REM *****************
20 REM * PROGRAMMSTART *
30 REM *****************
40 POKE 53280,0:POKE 53281,0
50 PRINT CHR$(147)
60 DIM M$(6)
70 FOR I=1 TO 6
80 READ M$(I):NEXT I
90 DATA "  1 QUADRATWURZEL"
100 DATA "  2 SINUS"
110 DATA "  3 COSINUS"
120 DATA "  4 NATUERLICHER LOGARITHMUS"
130 DATA "  5 DEKADISCHER LOGARITHMUS"
140 DATA "  6 PROGRAMMENDE"
150 GOTO 330
160 REM ******************
170 REM * UNTERPROGRAMME *
180 REM ******************
190 REM
200 REM *************
210 REM * KOPFZEILE *
220 REM *************
230 PRINT CHR$(147);
240 FOR I=1 TO 40:PRINT"*";:NEXT I
250 PRINT "*                                      *";
260 PRINT "*          MATHEMATISCHE TABELLE       *";
270 PRINT "*                                      *";
280 FOR I=1 TO 40:PRINT"*";:NEXT I
290 RETURN
300 REM *********
310 REM * MENUE *
320 REM *********
330 GOSUB 230
340 FOR I=1 TO 18
350 PRINT "*                                      *";
360 NEXT I
370 FORI=1TO40:PRINT"*";:NEXTI
380 PRINT CHR$(19);
390 FOR I=1 TO 3:PRINT:NEXT I
400 FOR I=1 TO 6
410 PRINT CHR$(29);M$(I)
420 NEXT I
```

```
430 PRINT
440 PRINT CHR$(29);
450 PRINT"  GEBEN SIE IHRE WAHL EIN  (1-6)";
460 INPUT W$
 .
 .
```

Zunächst sollen die einzelnen Programmzeilen kurz erklärt werden. Die Zeilen 40 bis 50 setzen die Bildschirmfarben auf schwarz und löschen den Bildschirm. In den Zeilen 60 bis 80 wird das Feld M$ generiert und mit den Daten aus den DATA Zeilen 90 bis 140 geladen. Danach überspringt das Programm den Programmteil der Unterprogramme und fährt mit der Ausführung in Zeile 330 fort. Von dort springt das Programm in das Unterprogramm ab Zeile 230, in dem die Kopfzeile erzeugt wird. Die beiden FOR...NEXT-Schleifen in den Zeilen 240 und 280 bewirken, daß jeweils eine Linie mit Sternchen auf den Bildschirm gebracht wird. Dazwischen liegen die PRINT-Befehle für die Überschrift.

Danach wird das Unterprogramm verlassen und der Programmablauf wird in Zeile 340 fortgesetzt. In einer weiteren Schleife (340-360) wird der Rahmen des Menüs aufgebaut. In Zeile 370 wird die untere Linie des Menüs ausgegeben und Zeile 380 bringt den Cursor wieder in die linke obere Bildschirmecke. Zeile 390 gibt drei Leerzeilen aus, damit die Menüpunkte nicht direkt in der Kopfzeile erscheinen. Die Zeilen 400 bis 420 geben dann nacheinander das Feld mit den einzelnen Menüpunkten aus. In Zeile 410 wird vor jeder Ausgabe eines neuen Elements der Cursor um eine Position nach rechts gerückt, damit die Ausgabe den linken Menürahmen nicht überschreibt. Sie können ruhig einmal ausprobieren, was geschieht, wenn Sie diesen Befehl entfernen. Zeile 450 gibt schließlich noch die Aufforderung an den Anwender aus, einen Wert einzugeben. Dadurch daß der PRINT-Befehl in dieser Zeile mit einem Semikolon endet, wird die Eingabe mit INPUT direkt nach der Klammer (1-6) erwartet.

Damit hätten wir die Erstellung des Menüs abgeschlossen.

Die Aufbereitung des eingegebenen Wertes mit entsprechender Verzweigung zu anderen Programmteilen wollen wir uns hier ersparen. Das Prinzip ist das gleiche wie auch beim "Rechenlehrgang". Da wir hier allerdings mit INPUT gearbeitet haben, müßten Sie die eingegebenen Werte noch auf Zulässigkeit überprüfen, da wir ja nicht wie beim GET-Befehl bestimmte Tasten selektieren können.

Zum Unterprogramm Kopfzeile sei noch ein Hinweis erlaubt. Diesen Teil können Sie immer dann aufrufen, wenn Sie den Bildschirm neu aufbauen wollen. Würden Sie also in den Programmteil zur Wurzelberechnung verzweigen, sollte dort als erster Aufruf

GOSUB 230

stehen. Anschließend können Sie zur Eingabe des zu berechnenden Wertes auffordern. Durch solche Kopfzeilen erhalten Ihre Programme eine nicht zu unterschätzende optische Aufwertung.

Zur Übung können Sie dieses Programm ja einmal fertigstellen. Wir aber wollen uns nun etwas mehr mit dem GET-Befehl auseinandersetzen.

4.3.1 VERWENDUNG VON GET-ROUTINEN IM MENÜ

In einem früheren Kapitel wurde angesprochen, daß der Nachteil des GET-Befehls darin liegt, daß man nicht erkennen kann, was man eingegeben hat. Im "Rechenlehrgang" (neuerer Teil) wurde ein Weg gefunden, die Eingabe doch sichtbar zu machen. Man gab durch den PRINT-Befehl einfach die mit GET eingelesenen Werte von A$ aus. Durch das Semikolon wurde erreicht, daß die Ausgabe der einzeln eingegebenen Zeichen hintereinander erfolgen konnte.

Diese Eingaberoutine mit GET ist aber noch sehr primitiv. Hatte man drei Zahlen bzw. Zeichen eingegeben, wurden diese automatisch übernommen, ohne daß man eine Möglichkeit zur Korrektur hatte. Ebenfalls wurden auf

alle Fälle drei Zeichen eingelesen, sodaß man gezwungen war, für die Zahl 54 die Ziffernfolge 054 einzugeben. Wollte man eine größere Zahl als 999 eingeben, war dies ebenfalls nicht möglich. Also doch wieder zurück zum INPUT-Befehl?

Es sei hier nochmals erwähnt, daß der INPUT-Befehl normalerweise für eigene Anwendungen ausreicht. Wollen Sie aber Ihre Programme gegen Fehlbedienung absichern, kommen Sie nicht umhin, den GET-Befehl einzusetzen.

Wir wollen uns nun an die Entwicklung einer eigenen GET-Routine begeben. Diese können Sie nach Fertigstellung in Ihre eigenen Programme als Unterprogramm aufnehmen. Die erste Zeile dürfte Ihnen inzwischen bekannt sein:

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

Damit können Sie jedes Zeichen von der Tastatur aus einlesen und A$ zuordnen. Wir wollen für unseren Fall aber nur Zahlen zulassen. Daher müssen wir diese Tasten selektieren, d.h. andere Eingaben als Zahlen müssen ausgeschlossen werden. Das erreichen wir mit einer IF...THEN Abfrage:

```
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
```

Die ASCII-Werte 48 bis 57 repräsentieren die Zahlen von 0 bis 9. Ist der eingegebene ASCII-Wert also kleiner 48 oder größer 57, wird die Eingabe ignoriert und zurück in die Zeile 10 gesprungen. Wollen wir nun nur Zahlen bis zu einer bestimmten Stellenzahl zulassen, müssen wir die eingegebenen gültigen Zeichen zählen. Soll die größte Zahl also vierstellig sein, müssen wir den Zähler auf den Wert größer vier abfragen. Wir benötigen also zwei weitere Zeilen. Eine, in der der Zähler hochgezählt und eine, in der der Zähler auf den Wert vier überprüft wird.

```
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Der Zähler Z in Zeile 30 wird jetzt nur dann hochgezählt, wenn ein zulässiger Wert eingegeben wurde. Hat Z bereits den Wert 4, so wird kein weiterer Wert akzeptiert und das Programm springt wieder in die Zeile 10.

Wir müssen jetzt unserer Routine noch mitteilen, daß der eingegebene Wert übernommen werden soll. Dafür bietet sich, wie beim INPUT-Befehl, die RETURN-Taste an. Welchen ASCII-CODE besitzt die RETURN-Taste? In der Tabelle erfahren wir, daß diese Taste den ASCII-Wert 13 hat. Wir brauchen damit nur den ASC(A$) auf den Wert 13 abzufragen. Doch Vorsicht, wo bauen wir diese Abfrage jetzt ein? Da 13 kleiner als 48 ist, dürfen wir diese Abfrage nicht hinter die Zeile 20 setzen, da dann die RETURN-Taste ignoriert würde. Die Abfrage muß also eine Zeilennummer bekommen, die kleiner als 20 ist. Nehmen wir hierfür die Nummer 15. Wie soll denn die Routine nun weiter verzweigen, wenn die RETURN-Taste gedrückt wurde? Da wir das zu diesem Zeitpunkt noch nicht genau wissen, aber zugleich absehen können, daß die Routine nicht allzu groß wird, soll vorerst nach Zeile 100 verzweigt werden.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Was wir jetzt noch benötigen, ist eine Zeile, in der die eingegebenen Zeichen miteinander in einer Stringvariablen verknüpft werden. Das soll in der Zeile 50 geschehen. Weiterhin wollten wir unsere eingegebenen Zeichen ja auf dem Bildschirm erkennen können. Diese Aufgabe soll Zeile 60 übernehmen.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
```

Mit Zeile 60 werden die Zeichen ab der aktuellen Cursorposition ausgegeben und zwar hintereinander (Semikolon), d.h. an der Stelle auf dem Bildschirm, an der der letzte PRINT-Befehl ausgeführt wurde. Nachdem das Zeichen ausgegeben wurde, soll das Programm wieder in die Zeile 10 springen, also GOTO 10.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
```

Jetzt brauchen wir nur noch den erzeugten String in B$ in einen numerischen Wert umzuwandeln und diesen einer numerischen Variablen zu übergeben. Das kann nach Betätigen der RETURN-Taste geschehen. Außerdem müssen wir daran denken, den Zähler Z nach dem Drücken der RETURN-Taste wieder auf Null zu setzen. Sonst würde der Wert bis zum nächsten Aufruf der Routine mitgezogen und man hätte nicht mehr die Möglichkeit, eine vierstellige Zahl einzugeben.

Soll die Routine als Unterprogramm Verwendung finden, muß die letzte Zeile selbstverständlich ein RETURN beinhalten. Wir wollen uns zunächst die komplette Routine anschauen. Nachdem Sie sie eingegeben haben, können Sie ja ein wenig damit herumexperimentieren. Sie sollten vielleicht als letzte Zeile die Ausgabe der Variablen vorsehen, an die der numerische Wert übergeben wurde.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
100 B=VAL(B$):Z=0
110 PRINT B
120 END
```

Damit hätten wir eine GET-Routine, mit der wir bis zu vierstellige Zahlen einlesen und anzeigen lassen können. Wollen Sie noch größere Zahlen einlesen lassen, so können Sie das durch Verändern des Wertes 4 in der Zeile 40 erreichen.

Diese Routine ist somit schon um einiges komfortabler als die, die wir vom "Rechenlehrgang" her kennen. Allerdings fehlt ihr noch die Funktion, daß man bereits eingegebene Werte wieder löschen kann. Diese Funktion gehört schon zu den etwas komplizierteren Merkmalen einer selbstgebauten GET-Routine. Im folgenden soll nun eine Routine, die diese Funktion beinhaltet, aufgelistet werden.

```
10 REM GET-ROUTINE MIT LOESCHFUNKTION
20 GET A$:IF A$="" THEN 20
30 IF ASC(A$) = 13 THEN 130
40 IF ASC(A$) < > 20 THEN 70
50 IF LEN(B$) < 1 THEN 20
60 B$=LEFT$(B$,LEN(B$)-1):Z=Z-1:GOTO 110
70 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 20
80 Z=Z+1
90 IF Z > 4 THEN Z=4:GOTO 20
100 B$=B$+A$
110 PRINT A$;
120 GOTO 20
130 B=VAL(B$):Z=0
140 PRINT B
150 END
```

Wir wollen nun die Zeilen im Programm besprechen, die noch hinzugekommen sind. Zunächst wäre da die Zeile 40. In dieser Zeile wird überprüft, ob die Taste INST/DEL betätigt wurde. Der ASCII-Wert dieser Taste ist 20, wie Sie der Tabelle im Anhang entnehmen können. Wurde die Taste nicht gedrückt, so verzweigt das Programm weiter nach Zeile 70. Haben Sie die INST/DEL-Taste allerdings betätigt, so wird in Zeile 50 überprüft, ob der String B$ noch Zeichen enthält. Wird diese Abfrage versäumt, so würde der Löschversuch eines weiteren Zeichens bei einem bereits leeren String zu einem Programmabbruch mit der Fehlermeldung

```
?ILLEGAL QUANTITY ERROR IN 60
```

führen. Ist der String also leer, wird die INST/DEL-Taste ignoriert und das Programm verzweigt erneut zur Zeile 20. In der Zeile 60 findet nun der eigentliche Vorgang des Löschens statt. Mit der Befehlsfolge

```
B$=LEFT$(B$,LEN(B$)-1)
```

wird der bis dahin eingegebene String B$ um ein Zeichen verkürzt. Der Befehl LEFT$(B$,X) erzeugt bekanntlich einen String mit den X linken Zeichen von B$. In den meisten Fällen steht an der Stelle von X eine Zahl. Hier wurde jedoch anstelle von X der LEN-Befehl benutzt. Die Berechnung LEN(B$)-1 wird zuerst ausgeführt. Das bedeutet, daß ein Wert gebildet wird, der genau um den Wert eins kleiner ist als die aktuelle Länge von B$. Mit diesem neuen Wert wird nun ein Teilstring von B$ gebildet, der genau um ein Zeichen kürzer ist als der ursprüngliche String von B$. Dieser um ein Zeichen verkürzte String wird erneut B$ zugeordnet. Der Vorgang ist in etwa mit der folgenden Operation bei numerischen Variablen vergleichbar:

```
A=A-1
```

Hier wird von der Variablen A der Wert 1 subtrahiert und das Ergebnis wiederum A zugeordnet.

Mit dieser Befehlsfolge haben wir also erreicht, daß das zuletzt dem String B$ zugeordnete Zeichen wieder gelöscht wurde.

Weiterhin muß in der Zeile 60 der Zähler Z um den Wert 1 vermindert werden, da er die Anzahl der Ziffern der gesamten Zahl bestimmt. Wir wollen ja eine Zahl mit maximal vier Stellen eingeben. Daher wurde Z als Zähler benutzt, der die bereits gültigen eingegebenen Zeichen zählt. Löschen wir ein Zeichen, so muß nicht nur der String B$ um ein Zeichen verkürzt werden, sondern es muß auch vom Zähler Z der Wert 1 subtrahiert werden. Würde man vergessen, den Zähler ebenfalls zu verkleinern, so wäre eine weitere Eingabe nach vier gültigen Zeichen nicht mehr möglich. Der String B$ würde zwar jedesmal beim Betätigen der INST/DEL-Taste um ein Zeichen verkürzt, aber da der Zähler bereits den Wert vier angenommen hat, würde das Programm in Zeile 90 dann wieder nach Zeile 20 verzweigen.

Der letzte Befehl in der Zeile 60 veranlaßt das Programm, in die Zeile 110 zu springen. Dort wird der aktuelle Inhalt von A$ ausgegeben. In A$ befindet sich nach dem Betätigen der INST/DEL-Taste der CHR$(20)-Code. Dieser bewirkt beim Ausdruck mit PRINT genau das gleiche, als ob Sie die INST/DEL-Taste direkt benutzen würden. Um das zu veranschaulichen, geben Sie einmal folgende Befehlsfolge in den Rechner:

```
PRINT "COMMODORE 65";CHR$(20);"4"
```

Drücken Sie nun die RETURN-Taste, so werden Sie feststellen, daß Sie folgenden Ausdruck erhalten:

```
COMMODORE 64
```

Der CHR$(20)-Code löscht also bei der Ausgabe mit PRINT genau wie die INST/DEL-Taste ein Zeichen. Die "4" tritt sofort an die Stelle des gelöschten Zeichens. Dadurch erhalten Sie dann den oben gezeigten Ausdruck. Nichts anderes geschieht in unserer GET-Routine in der Zeile 110, wenn A$ den CHR$(20)-Code enthält. Das ist

schon alles, was zu den neu hinzugekommenen Zeilen zu sagen wäre.

Damit haben wir nun eine Get-Routine aufgebaut, die es uns erlaubt, je nach Manipulation des Wertes Z kürzere oder längere Zeichenketten einlesen zu lassen. Außerdem kann man diese Zeichen gleichzeitig anzeigen lassen sowie auch wieder löschen. Diese Routine ist dem INPUT-Befehl von der Bedienung her schon recht ähnlich geworden, nur daß wir hier bestimmen können, welche Zeichen wir zulassen wollen oder nicht.

Zum Schluß wollen wir unsere Get-Routine noch so verfeinern, daß wir auch eine Art von Cursor zur Verfügung haben. Er blinkt zwar nicht, jedoch wissen wir, wo das nächste Zeichen einzugeben ist. Als unseren eigenen "Cursor" wollen wir den schmalen Grafikstrich nehmen, der auf der Taste mit dem Klammeraffen zu finden ist. Dieses Zeichen besitzt den CHR$(164)-Code. In diesem Zusammenhang wird der CHR$(157)-Code mit verwendet, um den realen Bildschirmcursor, der bei der Verwendung von GET ja unsichtbar bleibt, eine Stelle nach links zu bewegen. Dadurch liegt der Cursor über unserem Grafikzeichen. Wird nun ein Zeichen eingegeben, wird es durch den Befehl PRINT A$; an der Stelle des aktuellen Bildschirmcursors ausgegeben. Dadurch wird das Grafikzeichen durch das eingegebene Zeichen überschrieben. Gleichzeitig wird mit der Ausgabe des neuen Zeichens unser Cursor (CHR$(164)) direkt hinter diesem Zeichen wieder mit ausgegeben (Zeile 210).

Weiterhin wollen wir die Routine dahingehend erweitern, daß Buchstaben und das Leerzeichen mit eingegeben werden können. Dazu müssen wir zwei weitere Abfragen mit IF...THEN verwenden, um u.a. auszuschließen, daß die ASCII-Werte, die zwischen den Zahlen und den Buchstaben liegen (58 bis 64), zugelassen werden. Doch schauen Sie sich zunächst das Programmlisting dieser Routine an.

```
10 REM ************************
20 REM * CURSOR POSITIONIEREN *
30 REM ************************
40 PRINT CHR$(147);
50 FOR I=1 TO 5
60 PRINT CHR$(17);CHR$(29);
70 NEXT I
80 PRINT CHR$(164);CHR$(157);
90 REM  ********************
100 REM * ZEICHEN EINLESEN *
110 REM ********************
120 GET A$:IF A$="" THEN 120
130 IF ASC(A$)=13 THEN 260
140 IF ASC(A$)=32 THEN 200
150 IF ASC(A$) < > 20 THEN 170
160 IF LEN(B$) > = 1 THEN B$=LEFT$(B$,LEN(B$)-1):GOTO 210
170 IF ASC(A$) < 48 THEN 120
180 IF ASC(A$) > 90 THEN 120
190 IF ASC(A$) > 57 AND ASC(A$) < 65 THEN 120
200 B$=B$+A$
210 PRINT A$;CHR$(164);CHR$(157);
220 GOTO 120
230 REM ******************
240 REM * AUSGABE STRING *
250 REM ******************
260 PRINT CHR$(147)
270 FOR I=1 TO 15
280 PRINT CHR$(17);CHR$(29);
290 NEXT I
300 PRINT B$
```

In den Zeilen 40 bis 70 werden zunächst der Bildschirm gelöscht und der Bildschirmcursor in die 5. Bildschirmzeile und die 5. Bildschirmspalte gebracht. Dann wird unser selbstdefinierter Cursor an dieser Stelle ausgegeben (Zeile 80) und der unsichtbare Bildschirmcursor wird um eine Stelle nach links (CHR$(157)) über unseren Cursor gesetzt. In Zeile 120 beginnt nun die eigentliche Routine. Der Inhalt der Zeile 140 ist neu hinzugekommen. Hier wird abgefragt, ob das eingegebene Zeichen ein Leerzeichen (CHR$(32)) war.

Die nächsten Zeilen sind wiederum bekannt. Zeile 180 fragt ab, ob das eingegebene Zeichen einen größeren ASCII-Wert als der Buchstabe "Z" hatte. In Zeile 190 wird der Bereich zwischen den ASCII-Werten 57 und 65 abgefragt. Es wurde dazu der logische Operator AND verwendet. AND mußte hier benutzt werden, da beide Bedingungen erfüllt sein müssen. Damit das Programm nach Zeile 120 verzweigen kann, muß der eingegebene ASCII-Wert sowohl größer 57 als auch kleiner 65 sein (das ist das Intervall zwischen dem Zahlen- und Buchstabenbereich). Hätten wir hier ein OR verwendet, so wären weder eine Zahl noch ein Buchstabe als Eingabe zugelassen worden. Die Wirkung der Zeile 210 wurde ja bereits besprochen. Die folgenden Zeilen sollen das noch einmal verdeutlichen. Die Position des unsichtbaren Bildschirmcursors soll durch den Stern "*" gekennzeichnet werden.

Wir nehmen an, daß wir den Buchstaben A eingegeben haben. Die Zeile 210 bewirkt dann folgendes:

A* :Nach Ausführung von PRINT A$;

A_* :Nach Ausführung von CHR$(164);

A* :Nach Ausführung von CHR$(157);

Ich hoffe, daß Ihnen durch dieses kleine Schaubild die Wirkung der Zeile 210 deutlich geworden ist.

Die restlichen Programmzeilen dieser Routine sollten Ihnen inzwischen bekannt sein. Damit haben wir uns eine GET-Routine erstellt, die dem INPUT-Befehl schon sehr ähnlich ist. Sie sollten nun in der Lage sein, diese Routine Ihren Programmen entsprechend anpassen zu können, d.h. daß Sie durch entsprechende IF...THEN Abfragen selbst bestimmen, welche Tasten Sie selektieren möchten. Damit steht Ihnen für Ihre Menüs oder sonstigen Eingaben innerhalb von Programmen eine recht komfortable und sichere Routine zur Verfügung.

Was jetzt noch fehlt, ist eine Routine, mit der Sie auf einfache Art und Weise den Cursor auf dem Bildschirm positionieren können, d.h. daß Sie nur durch die Angabe von Bildschirmzeilen- und Bildschirmspaltennummer die Position des Cursors bestimmen können. Vielleicht kennen einige von Ihnen die beiden Speicherstellen 211 und 214 des COMMODORE 64. Sie haben damit die Möglichkeit durch den POKE-Befehl und nach dem Aufruf einer Routine im Betriebssystem den Cursor auf dem Bildschirm zu positionieren.

Diese Art der Cursorpositionierung hat aber den Nachteil, daß Sie wieder rechnerspezifisch ist. Ich will Ihnen nun eine Möglichkeit aufzeigen, die Sie auch für andere Rechner verwenden können.

Sie basiert auf der Überlegung, daß man den Cursor ja auch mit den CHR$-Codes CHR$(17), CHR$(29), CHR$(145) und CHR$(156) steuern kann. Für uns sind hier nur die Codes CHR$(17) und CHR$(29) interessant, also die, die den Cursor nach unten und nach rechts bewegen. Mit diesen Steuercodes werden zunächst zwei Strings gebildet. Der eine String enthält 40mal das Steuerzeichen, das den Cursor nach rechts bewegt (CHR$(29)). Der zweite String enthält 25mal das Steuerzeichen, das den Cursor um eine Zeile nach unten bewegt (CHR$(17)). Diese Programmzeilen wollen wir uns nun anschauen.

```
  .
  .
90 REM CURSOR NACH UNTEN
100 FOR I=1 TO 25
110 CU$=CU$+CHR$(17)
120 NEXT I
130 REM CURSOR RECHTS
140 FOR I=1 TO 40
110 CR$=CR$+CHR$(29)
120 NEXT I
  .
  .
```

Durch diese Programmzeilen werden also unsere zwei Strings generiert. Damit haben wir schon die Hälfte der Arbeit für unsere Cusorpositionierung geschafft. Was jetzt noch kommen muß, ist die eigentliche Positionierung. Wir wollten durch die Angabe zweier Werte den Cursor an eine bestimmte Stelle des Bildschirms bringen. Die Schreibweise soll dabei die folgende sein:

S=ZEILE.SPALTE

oder speziell:

S=10.12

Die Variable S erhält demnach einen Wert zugeordnet, bei dem die Vorkommastelle die Zeile und die Nachkommastelle die Spalte angibt. Diese Variable muß jetzt noch entsprechend verarbeitet werden und dann kann der Cursor schon gesetzt werden. Die folgenden Zeilen zeigen Ihnen, wie Sie dabei vorzugehen haben.

```
.
.
300 REM CURSORPOSITIONIERUNG
310 PRINT CHR$(19);LEFT$(CU$,S)
320 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
330 RETURN
.
.
```

Das ist schon die gesamte Routine zur Cursorpositionierung. Die Zeile 310 dürfte schnell verstanden werden. Zunächst wird bei jeder Positionierung der Cursor in die linke obere Bildschirmecke gebracht. Damit hat man einen Ausgangspunkt geschaffen, an dem man sich orientieren kann. Danach wird ein Teilstring von CU$ entsprechend dem Wert von S gebildet, der den Cursor in unserem Fall 10 Zeilen nach unten bewegt. Es wird ja bei LEFT$(CU$,S) nur die Vorkommastelle von S berücksichtigt.

Nach Ausführung der Zeile 310 steht der Cursor somit schon in der richtigen Bildschirmzeile.

Zeile 320 ist etwas komplizierter im Aufbau. Beginnen wir mit der inneren Klammer. Dort wird mit S-INT(S) die Nachkommastelle von der Vorkommastelle abgetrennt. Die Nachkommastelle, in unserem Fall .12, wird nun mit 100 multipliziert. Anschließend wird noch .5 addiert. Damit wird erreicht, daß der Nachkommawert auch wirklich erhalten bleibt. Es kann nämlich schon mal vorkommen, daß man bei der Berechnung von 100*(S-INT(S)) nicht genau, wie in unserem Beispiel, 12 als Ergebnis erhält, sondern 11.99999, als ganzzahligen Wert also nur 11. Wird nun .5 addiert, so ergibt sich ein Wert von 12.49999. Da ja bei der Anwendung von LEFT$(CR$,S) von S nur der ganzzahlige Teil berücksichtigt wird, erreicht man also durch diesen kleinen Trick, daß auf jeden Fall der Wert 12 erreicht wird. Mit diesem Wert wird nun ein Teilstring mit 12 Zeichen (Steuerzeichen) von CR$ gebildet. Bei der Ausgabe mit PRINT wird der Cursor dadurch um 12 Stellen nach rechts gesetzt. Die letzte Zeile enthält ein RETURN, da die Positionierung ja als Unterprogramm aufgerufen werden soll.

Damit steht Ihnen eine Routine zur Cursorpositionierung zur Verfügung, die vom Aufbau her sehr schnell verstanden werden kann und sich dadurch auch leicht in eigenen Programmen anwenden läßt. Weiterhin dürfte diese Routine sich sehr leicht auch auf andere Rechner übertragen lassen. Hierzu brauchen Sie nur die entsprechenden Cursorsteuerzeichen einzusetzen.

Diese Routine kann man nun auch für den Aufbau von Menüs verwenden. Wir wollen nun das Programm für die "Mathematische Tabelle" dahingehend abändern, daß wir die Ausgabe der einzelnen Menüpunkte über unsere neue Cursorpositionierung vornehmen können. Dazu werden in den DATA Zeilen die Leerzeichen vor den Menüpunkten entfernt. Danach erfolgt im Programm die Erzeugung der beiden Strings für die Cursorsteuerung. Schauen Sie sich jedoch zunächst das abgeänderte Programmlisting an.

```
90 DATA "1 QUADRATWURZEL"
100 DATA "2 SINUS"
110 DATA "3 COSINUS"
120 DATA "4 NATUERLICHER LOGARITHMUS"
130 DATA "5 DEKADISCHER LOGARITHMUS"
140 DATA "6 PROGRAMMENDE"
150 REM *****************
160 REM * CURSOR RECHTS *
170 REM *****************
180 FOR I=1 TO 40
190 CR$=CR$+CHR$(29)
200 NEXT I
210 REM **********************
220 REM * CURSOR NACH UNTEN *
230 REM **********************
240 FOR I=1 TO 25
250 CU$=CU$+CHR$(17)
260 NEXT I
270 GOTO 510
280 REM ******************
290 REM * UNTERPROGRAMME *
300 REM ******************
310 REM
320 REM *************
330 REM * KOPFZEILE *
340 REM *************
350 PRINT CHR$(147);
360 FOR I=1 TO 40:PRINT"*";:NEXT I
370 PRINT "*                                      *";
380 PRINT "*          MATHEMATISCHE TABELLE       *";
390 PRINT "*                                      *";
400 FOR I=1 TO 40:PRINT "*";:NEXT I
410 RETURN
420 REM ************************
430 REM * CURSORPOSITIONIERUNG *
440 REM ************************
450 PRINT CHR$(19);LEFT$(CU$,S);
460 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
470 RETURN
480 REM *********
```

```
490 REM * MENUE *
500 REM *********
510 GOSUB 350
520 FOR I=1 TO 18
530 PRINT "*                                          *";
540 NEXT I
550 FOR I=1 TO 40:PRINT"*";:NEXT I
560 PRINT CHR$(19);
570 FOR I=1 TO 3:PRINT:NEXT I
580 FOR I=1 TO 6
590 S=4+I*2+.03
600 GOSUB 450: REM * CURSOR POSIT. *
610 PRINT M$(I)
620 NEXT I
```

Wie bereits erwähnt, wurden bei den Menüpunkten in den DATA-Zeilen die Leerzeichen entfernt. Neu hinzugekommen sind dann die Programmzeilen 150 bis 260, die vom Inhalt her schon besprochen wurden. Weiterhin wurden die Zeilen 420 bis 470 als Unterprogramm hinzugefügt. Auch diese sind vom Inhalt her bereits erläutert worden. Die folgenden Zeilen des ersten Programms zur "Mathematischen Tabelle"

```
400 FOR I=1 TO 6
410 PRINT CHR$(29);M$(I)
420 NEXT I
```

wurden nun durch die Zeilen

```
580 FOR I=1 TO 6
590 S=4+I*2+.03
600 GOSUB 450: REM * CURSOR POSIT. *
610 PRINT M$(I)
620 NEXT I
```

ersetzt. Wir wollen die Menüpunkte, beginnend mit der 6.Zeile und der 3.Spalte, ausgeben lassen. Da sich nur der Wert der Zeile verändert und der der Spalte konstant

bleibt, haben wir den Nachkommawert für S schon bestimmt, nämlich .03.

Geben Sie hier nicht .3 an, das würde bedeuten, daß die Menüpunkte erst ab der 30.Spalte ausgegeben würden!

Der Zeilenwert beginnt mit 6 und soll jeweils um zwei Werte bei jeder Ausgabe zunehmen. Hier bietet sich wieder die Laufvariable "I" zur eleganten Berechnung der Zeilennummer an. Für die ZEILENnummer erhalten wir demnach beim ersten Durchlauf

S=4+1*2

gleich 6, beim zweiten Durchlauf

S=4+2*2

gleich 8 usw. Damit hätten wir unsere neue Routine schon im Programm eingebaut.

Bei der bisherigen Menütechnik mußten wir immer irgendwelche Tasten, seien es Zahlen- oder Buchstabentasten, betätigen, um die entsprechenden Programme aufzurufen. Ich möchte Ihnen noch eine andere Art der Menütechnik zeigen, bei der Sie nur noch die Taste, die den Cursor nach unten oder nach oben bewegt, zu benutzen brauchen. Haben Sie dann den entsprechenden Menüpunkt angewählt, betätigen Sie nur die RETURN-Taste, und das Programm fährt mit dem entsprechenden Menüpunkt fort. Die Kennzeichnung des angewählten Menüpunkts soll durch die Darstellung desselben in reverser Schrift erfolgen. Wir benötigen dazu zwei Felder. Das erste Feld soll die Menüpunkte in normaler Schrift, das zweite Feld in reverser Schrift enthalten. Schauen wir uns auch dazu zunächst das veränderte Programmlisting an.

```
10 REM *****************
20 REM * PROGRAMMSTART *
30 REM *****************
40 POKE 53280,0:POKE 53281,0
50 PRINT CHR$(147)
60 DIM M$(6): REM * MENUEPUNKTE
70 DIM MR$(6): REM * MENUEPUNKTE REVERS
80 REM  **********************
90 REM  * FELDER AUFFUELLEN *
100 REM **********************
110 FOR I=1 TO 6
120 READ M$(I)
130 MR$(I)=CHR$(18)+M$(I)+CHR$(146)
140 NEXT I
150 DATA "1 QUADRATWURZEL"
160 DATA "2 SINUS"
170 DATA "3 COSINUS"
180 DATA "4 NATUERLICHER LOGARITHMUS"
190 DATA "5 DEKADISCHER LOGARITHMUS"
200 DATA "6 PROGRAMMENDE"
210 REM *****************
220 REM * CURSOR RECHTS *
230 REM *****************
240 FOR I=1 TO 40
250 CR$=CR$+CHR$(29)
260 NEXT I
270 REM **********************
280 REM * CURSOR NACH UNTEN *
290 REM **********************
300 FOR I=1 TO 25
310 CU$=CU$+CHR$(17)
320 NEXT I
330 GOTO 570
340 REM ******************
350 REM * UNTERPROGRAMME *
360 REM ******************
370 REM
380 REM *************
390 REM * KOPFZEILE *
400 REM *************
410 PRINT CHR$(147);
420 FOR I=1 TO 40:PRINT"*";:NEXT I
```

```
430 PRINT "*                                          *";
440 PRINT "*          MATHEMATISCHE TABELLE           *";
450 PRINT "*                                          *";
460 FOR I=1 TO 40:PRINT"*";:NEXT I
470 RETURN
480 REM ************************
490 REM * CURSORPOSITIONIERUNG *
500 REM ************************
510 PRINT CHR$(19);LEFT$(CU$,S);
520 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
530 RETURN
540 REM *********
550 REM * MENUE *
560 REM *********
570 GOSUB 410
580 FOR I=1 TO 18
590 PRINT "*                                          *";
600 NEXT I
610 FOR I=1 TO 40:PRINT"*";:NEXT I
620 PRINT CHR$(19);
630 FOR I=1 TO 3:PRINT:NEXT I
640 FOR I=1 TO 6
650 S=4+I*2+.03
660 GOSUB 510: REM CURSOR POSIT.
670 PRINT M$(I)
680 NEXT I
690 S=6.03:GOSUB 510: REM CURSOR POSIT.
700 PRINT MR$(1):Z=1
710 GET MP$:IF MP$="" THEN 710
720 IF ASC(MP$) < > 17 OR Z >= 6 THEN 760
730 S=4+Z*2+.03:GOSUB 510
740 PRINT M$(Z)
750 Z=Z+1:GOTO 820
760 IF ASC(MP$) < > 145 OR Z=1 THEN 800
770 S=4+Z*2+.03:GOSUB 510
780 PRINT M$(Z)
790 Z=Z-1:GOTO 820
800 IF ASC(MP$) = 13 THEN 850
810 GOTO 710
820 S=4+Z*2+.03:GOSUB 510
830 PRINT MR$(Z)
840 GOTO 710
```

```
850 ON Z GOSUB 1000,2000,3000,4000,5000,6000
    .
    .
```

In Zeile 60 und 70 werden jeweils ein Feld für die Menüpunkte in normaler Schrift M$(6) und in reverser Schrift MR$(6) dimensioniert. Die Zeilen 110 bis 140 fuellen diese beiden Felder mit den entsprechenden Menüpunkten auf. Die reversen Menüpunkte erzeugt die Zeile 130. Der CHR$(18)-Code bewirkt, daß die reverse Schrift eingeschaltet wird. Der CHR$(146)-Code schaltet wieder um auf die Normalschrift. Dadurch wird erreicht, daß die Menüpunkte im Feld MR$(6) revers abgelegt werden. Die restlichen Zeilen bis zum Aufbau des Menüs sind inzwischen bekannt. Auch die Ausgabe der Menüpunkte von Zeile 640 bis 680 dürften verstanden worden sein. Interessant wird es wieder ab Zeile 690. Dort wird der Cursor zunächst in der 6.Zeile und in der 3.Spalte positioniert. Anschließend wird in Zeile 700 der 1. reverse Menüpunkt ausgegeben und der Zähler Z auf eins gesetzt. Zeile 710 enthält die bekannte GET-Abfrage. Die Zeile 720 fragt ab, ob die betätigte Taste verschieden von der Taste war, die den Cursor nach unten bewegt oder ob Z schon größer oder gleich 6 ist. Die Abfrage auf größer oder gleich 6 geschieht deshalb, weil wir insgesamt 6 Menüpunkte haben. Beim Erreichen des letzten Menüpunktes muß ja die Taste, die den Cursor nach unten bewegt, im folgenden als CHR$(17) benannt, blockiert sein. Wurde nun die CHR$(17)-Taste betätigt, so muß zunächst der reverse Menüpunkt 1 wieder in Normalschrift erscheinen. Das bewirken die Zeilen 730 und 740. Die Zeile 750 erhöht den Zähler um den Wert 1. Anschließend wird zur Zeile 820 verzweigt, um den nächsten Menüpunkt in reverser Schrift ausgeben zu können (Zeilen 820 bis 840). Die Zeile 760 ist genauso zu interpretieren wie die Zeile 720, nur daß hier nicht die CHR$(17)-Taste abgefragt wird, sondern die CHR$(145)-Taste (Cursor nach oben). Hier darf der Zähler natürlich nicht kleiner als eins werden. Wurde nun die CHR$(145)-Taste betätigt, so wird wieder der momentan in reverser Schrift dargestellte Menüpunkt in Normalschrift ausgegeben (Zeilen 770 und 780). Danach wird der Zähler Z um eins

vermindert und anschließend zur Zeile 820 verzweigt. Dort wird dann der angesprochene Menüpunkt wieder in reverser Schrift ausgegeben. Betätigen Sie die RETURN-Taste, so wird von Zeile 720 über Zeile 760 nach Zeile 800 verzweigt. Von dort wird wiederum nach Zeile 850 verzweigt, in der dann entsprechend dem Wert von Z mit ON Z GOSUB die weiteren Unterprogramme aufgerufen werden können. Sie haben hier selbstverständlich die Möglichkeit, auch mit ON Z GOTO zu arbeiten. Ich hoffe, daß durch diese ausführliche Beschreibung dieses Prinzip der Menütechnik von Ihnen verstanden worden ist.

Damit stehen Ihnen nun alle Möglichkeiten offen, die erlernten Menütechniken anzuwenden, zu verfeinern, zu verändern oder gar eigene zu entwickeln. Ihrer Phantasie sind in dieser Hinsicht keine Grenzen gesetzt. Versuchen Sie ruhig einmal, aus dem Ansatz des Programms für die "Mathematische Tabelle" ein vollständiges Programm zu entwickeln. Wie Sie das Menü gestalten, bleibt Ihnen selbst überlassen. Diesmal werde ich Ihnen dazu keine Lösung anbieten, da so vielleicht der Anreiz noch größer wird.

Das Kapitel über die Menütechnik wäre somit abgeschlossen. In den beiden nächsten Kapiteln werde ich noch kurz auf ein Sortierverfahren und das Prinzip der Dateiverwaltung eingehen.

4.4 SORTIERVERFAHREN

Bei vielen Programmen wird es sich als nötig erweisen, die anfallenden Daten nach verschiedenen Ordnungskriterien (der Größe nach, in alphabetischer Reihenfolge usw.) zu sortieren. Es existieren hier eine Menge unterschiedlicher Verfahren, die sich untereinander in Leistung und Schwierigkeitsgrad unterscheiden. Allgemein kann man sagen, daß ein Verfahren umso schwieriger zu durchschauen ist, je schneller es die anfallenden Daten sortieren kann. Wir wollen daher hier nur ein einfaches Verfahren kennenlernen, damit Sie zumindest eine Einführung in diese Materie erhalten. Für einen Anfänger ist es eher abschreckend als anregend, die komplizierteren Verfahren kennenzulernen. Sollten Sie einmal etwas Erfahrung im Umgang mit einfachen Sortierungen gesammelt haben, so können Sie sich dann an die komplizierteren Strukturen dieser Verfahren wagen. Es gibt eine Reihe von Fachbüchern, in denen diese ausführlich beschrieben sind.

Wir wollen uns nun mit dem sogenannten BUBBLE-SORT Verfahren vertraut machen. Der Name BUBBLE-SORT rührt wohl daher, daß bei diesem Verfahren die einzelnen Elemente der Größe nach wie Blasen (engl. BUBBLE) im Wasser nach oben steigen. Als Beispiel wollen wir ein Feld mit Zufallszahlen auffüllen und dieses sortiert ausgeben lassen. Wir nehmen ein Feld mit 6 Elementen. Zunächst die Programmzeilen, die das Feld dimensionieren und mit Werten auffüllen:

```
10 REM FELD GENERIEREN
20 DIM F(6)
30 FOR I=1 TO 6
40 A=INT(50*RND(0))+1
50 F(I)=A
60 NEXT I
   .
   .
```

Das Prinzip unseres Sortierverfahrens beruht darauf, daß

immer zwei benachbarte Elemente miteinander verglichen werden. Ist ein Element größer als das andere, so findet eine Vertauschung statt. Somit werden nacheinander alle Elemente miteinander verglichen. Damit dies deutlich wird, realisieren wir das Verfahren mit IF...THEN-Abfragen. Man könnte es auch mit einer FOR...NEXT-Schleife programmieren, dabei wird allerdings das Verfahren nicht so deutlich. Haben Sie das Verfahren einmal verstanden, können Sie es durchaus mit einer FOR...NEXT-Schleife ausführen lassen. Doch nun zu den eigentlichen Programmzeilen:

```
.
.
100 REM BUBBLE-SORT
110 Z=0
120 IF F(6) > F(5) THEN 140
130 F(0)=F(6):F(6)=F(5):F(5)=F(0):Z=1
140 IF F(5) > F(4) THEN 160
150 F(0)=F(5):F(5)=F(4):F(4)=F(0):Z=1
160 IF F(4) > F(3) THEN 180
170 F(0)=F(4):F(4)=F(3):F(3)=F(0):Z=1
180 IF F(3) > F(2) THEN 200
190 F(0)=F(3):F(3)=F(2):F(2)=F(0):Z=1
200 IF F(2) > F(1) THEN 220
210 F(0)=F(2):F(2)=F(1):F(1)=F(0):Z=1
220 IF Z=1 THEN 110
230 FOR I=1 TO 6
240 PRINT F(I)
250 NEXT I
260 END
```

In Zeile 110 wird zunächst Z auf Null gesetzt. Warum das so ist, werden Sie im weiteren Verlauf erkennen können. Zeile 120 führt nun den ersten Vergleich durch. Ist der Inhalt des Elements von F(6) bereits größer als von F(5), so braucht keine Vertauschung vorgenommen zu werden und es kann direkt nach Zeile 140 verzweigt werden. Ist F(6) allerdings kleiner als F(5), so erfolgt in Zeile 130 eine Vertauschung. Das Prinzip dieser Vertauschung dürfte Ihnen schon bekannt sein. Wir verwenden hier das Element F(0) zur Zwischenspeicherung eines Variablenwertes.

Danach wird der Wert von F(5) dem Element F(6) zugeordnet. Anschließend erhält F(5) den Wert von F(0), also von F(6). Dieses Prinzip wurde auch in den anderen Programmzeilen angewendet.

Danach wird Z auf eins gesetzt, da eine Vertauschung stattgefunden hat. Wir können also anhand des Zustandes von Z erkennen, ob eine Vertauschung stattgefunden hat oder nicht. Hat Z den Wert 1, so wurde ausgetauscht, hat Z den Wert 0, so wurde nicht ausgetauscht. Dieser "Trick" wird häufig in der Programmierung angewandt, um überprüfen zu lassen, ob bestimmte Vorgänge stattgefunden haben oder nicht. Diese Variablen, wie in unserem Beispiel Z, nennt man auch FLAGS. FLAG bedeutet soviel wie Flagge oder Zeichen. Hat Z also nach einem Durchlauf immer noch den Wert Null, so wissen wir, daß keine Vertauschung stattfand und daß wir somit unser Feld sortiert vorliegen haben. Das hat den Vorteil, daß die Sortierung bereits nach einem Durchlauf abgebrochen werden kann, wenn schon zufällig die richtige Reihenfolge der Elemente vorliegt. Dieses Verfahren findet man auch unter dem Namen "BUBBLE-SORT mit Weiche". Weiche deshalb, weil eben jederzeit nach Erreichen der zufälligen Sortierung das Verfahren abgebrochen werden kann.

Die letzten Zeilen geben dann das sortierte Feld aus. Wollen Sie beobachten, wie die Sortierung im einzelnen stattfindet, so ändern Sie die letzten Programmzeilen bitte wie folgt ab:

```
  .
  .
220 FOR I=1 TO 6
230 PRINT F(I);
240 NEXT I
250 PRINT
260 IF Z=1 THEN 110
270 END
```

Damit wären wir schon am Ende der Besprechung dieses Sortierverfahrens angelangt. Beschäftigen Sie sich eingehend mit dieser Materie, so daß Sie auch später auf

kompliziertere Verfahren zurückgreifen können, womit Sie sich einen zeitlichen Vorteil verschaffen. Das BUBBLE-SORT-Verfahren ist geeignet für eine Anzahl bis zu etwa 100 Elementen.

4.5 DAS PRINZIP DER DATEIVERWALTUNG

Eine Datei (engl. FILE) setzt sich zusammen aus einer Anzahl beliebiger Daten, die auf einem Speichermedium (Kassette oder Diskette) abgespeichert sind und von dort wieder in den Computer geladen werden können. Bei der Dateiverwaltung treten immer wieder drei Begriffe auf: Datei, Datensatz und Datenfeld. Sie können sich diese Begriffe am besten mit folgendem Vergleich verdeutlichen: Unter einer Datei können Sie sich einen Karteikasten vorstellen. Dabei stellt ein Datensatz eine einzelne Karteikarte aus dem Karteikasten dar, und das Datenfeld ist ein einzelner Eintrag auf einer solchen Karteikarte.

Laut Definition ist also ein abgespeichertes Programm bereits eine Datei. Allerdings wird meistens unter einer Datei eine Sammlung von Namen, Zahlen oder anderen zusammenhängenden Daten verstanden, die zusammen auf ein Speichermedium abgespeichert wurden. Wollen Sie nun mit externen Speichergeräten arbeiten, so benötigen Sie als erstes die Befehle LOAD und SAVE. Da sich eine rationelle Dateiverwaltung nur mit dem Diskettenlaufwerk realisieren läßt, wollen wir uns hier auf den Umgang mit diesem beschränken. Um ein Programm von Diskette zu laden, geben Sie folgendes ein:

```
LOAD"PROGRAMMNAME",GA
```

GA steht hier für die Geräteadresse, welche beim Diskettenlaufwerk meistens den Wert 8 besitzt. In den Anführungszeichen dürfen maximal 16 Zeichen verwendet werden, so daß Ihre Befehlsfolge jetzt wie folgt aussehen könnte:

```
LOAD"BEISPIEL",8
```

Wollen Sie ein Programm abspeichern, so wird dafür der SAVE-Befehl verwendet. Er hat die folgende Schreibweise:

SAVE"BEISPIEL",8

Die genaue Anwendung dieser und der noch folgenden Befehle entnehmen Sie bitte dem Handbuch zum 1541 Diskettenlaufwerk. Wollen Sie sich eingehender mit der Materie des Diskettenlaufwerks und den sich daraus ergebenden phantastischen Möglichkeiten befassen, so empfehle ich Ihnen "Das große Floppy-Buch" aus dem Hause DATA BECKER.

Eine einfache Dateiverwaltung funktioniert im Prinzip etwa so: Es werden Daten in ein zuvor dimensioniertes Feld eingelesen. Die Daten dieses Feldes werden sodann komplett auf eine Diskette abgespeichert. Innerhalb des Programms hat man dann meistens die Möglichkeit, nach bestimmten Daten suchen zu lassen, diese zu verändern und wieder auf die Diskette zurückzuschreiben.

Wollen Sie Daten an das Diskettenlaufwerk übergeben, müssen Sie zuerst eine Datei eröffnen. Dies geschieht mit dem OPEN-Befehl. Die Befehlsfolge

OPEN 1,8,2,"ADRESSEN,S,W"

öffnet beim Diskettenlaufwerk (Geräteadresse 8) einen Kanal zur Datenübertragung und gleichzeitig die sequentielle Datei "ADRESSEN" zum Schreiben (W=WRITE), d.h. Sie können nun Daten zum Laufwerk übertragen. Sequentiell bedeutet, daß die Daten hintereinander abgespeichert werden. Hierzu benötigen Sie den

PRINT#

Befehl. Dem PRINT# folgt noch eine Zahl, die sich auf die zuvor mit OPEN geöffnete Datei bezieht, also in unserem Falle PRINT#1. Geben Sie nun den Befehl PRINT#1,D$ ein, so wird der Inahlt der Variablen D$ auf die Diskette geschrieben. Wollen Sie die Datenübertragung beenden, so

müssen Sie die Datei mit dem Befehl

CLOSE

wieder schließen. Auch hier wird wieder die Zahl der zuvor geöffneten Datei mit angegeben, also CLOSE 1.

Damit wissen wir nun, wie eine sequentielle Datei auf der Diskette angelegt werden kann. Wie bekommen wir aber nun unsere abgespeicherten Daten wieder in den Computer? Auch dazu muß die entsprechende Datei erst wieder geöffnet werden. Da wir diesmal die Daten aber lesen wollen, ändert sich der OPEN-Befehl wie folgt:

OPEN 1,8,2,"ADRESSEN,S,R"

Damit wird die sequentielle Datei "ADRESSEN" zum Lesen (R=READ) geöffnet. Zum Einlesen der Daten verwenden wir den

INPUT#

Befehl. Hier muß ebenfalls wieder die Zahl der zuvor geöffneten Datei mit angegeben werden. Diese Zahl nennt sich "logische Filenummer". Wollen wir in unserem Programm also Daten von der Diskette lesen, müssen wir folgende Befehlsfolge verwenden:

INPUT#1,D$

Mit diesem Befehl können Sie Daten mit einer maximalen Länge von 80 Zeichen einlesen. Mit dem GET#-Befehl wird, wie beim "normalen" GET-Befehl auch, jeweils ein Zeichen von der Diskette gelesen.

Wollen Sie bestimmte Befehle an das Diskettenlaufwerk übermitteln (formatieren, löschen von Files usw.), müssen Sie die Sekundäradresse 15 verwenden. Dabei handelt es sich um den Befehlskanal des Diskettenlaufwerks (s. Handbuch 1541, Das große Floppy-Buch). Eine solche Befehlsfolge könnte wie folgt aussehen:

OPEN 1,8,15,"S:ADRESSEN":CLOSE 1

Mit dieser Befehlsfolge wird das File "ADRESSEN" auf der Diskette gelöscht. Im nachfolgenden Programm wurde dieser Befehl verwendet, um eine Datei neu auf die Diskette abspeichern zu können. Gibt es nämlich bereits eine Datei gleichen Namens auf der Diskette, so beginnt die rote Leuchtdiode am Laufwerk zu blinken und signalisiert damit einen Fehler. Fragt man diesen Fehler über den Befehlskanal ab, so erhält man als Ausgabe die Meldung

63 FILE EXISTS

Mit diesem Wissen sollten Sie nun das nachfolgende Programm verstehen können. Sie können in einem Menü auswählen, ob Sie Daten eingeben, speichern, laden oder ausgeben lassen oder das Programm beenden wollen. Wählen Sie "Daten eingeben", haben Sie die Möglichkeit, vier Namen mit Vornamen einzugeben.

Das Programm ist bewußt einfach gehalten, um Ihnen das Prinzip einer Datenverwaltung aufzeigen zu können. Mit dem bisher erworbenen Wissen - und unter Zuhilfenahme der erwähnten Bücher - stehen Ihnen alle Wege offen, sich aus diesem Ansatz einer Dateiverwaltung Ihre eigene Dateiverwaltung zu schreiben.

```
10 DIM D$(4,2)
20 PRINT CHR$(147)
30 PRINT "WOLLEN SIE"
40 PRINT
50 PRINT "DATEN EINGEBEN ? (1)"
60 PRINT
70 PRINT "DATEN SPEICHERN ? (2)"
80 PRINT
90 PRINT "DATEN LADEN ? (3)"
100 PRINT
110 PRINT "DATEN AUSGEBEN ? (4)"
120 PRINT
130 PRINT "BEENDEN ? (5)"
140 GET A$:IF A$="" THEN 140
```

```
150 ON VAL(A$) GOTO 190,290,400,500,600
160 REM ******************
170 REM * DATEN EINGEBEN *
180 REM ******************
190 PRINT CHR$(147)
200 FOR I=1 TO 4
210 INPUT "NAME";D$(I,1)
220 INPUT "VORNAME";D$(I,2)
230 PRINT CHR$(147)
240 NEXT I
250 GOTO 20
260 REM ********************
270 REM * DATEN SPEICHERN *
280 REM ********************
290 OPEN 1,8,15,"S:ADRESSEN":CLOSE1
300 OPEN 1,8,2,"ADRESSEN,S,W"
310 FOR I=1 TO 4
320 FOR Z=1 TO 2
330 PRINT#1,D$(I,Z)
340 NEXT Z,I
350 CLOSE 1
360 GOTO 20
370 REM ***************
380 REM * DATEN LADEN *
390 REM ***************
400 OPEN 1,8,2,"ADRESSEN,S,R"
410 FOR I=1 TO 4
420 FOR Z=1 TO 2
430 INPUT#1,D$(I,Z)
440 NEXT Z,I
450 CLOSE 1
460 GOTO 20
470 REM ******************
480 REM * DATEN AUSGEBEN *
490 REM ******************
500 PRINT CHR$(147)
510 FOR I=1 TO 4
520 FOR Z=1 TO 2
530 PRINTD$(I,Z)
540 NEXT Z,I
550 FOR I=1 TO 3000:NEXT I
560 GOTO 20
```

```
570 REM ********
580 REM * ENDE *
590 REM ********
600 PRINT CHR$(147)
610 END
```

ABS

Kategorie: Numerische Funktionen

Schreibweise: ABS(X)

Funktion: Ergibt den Absolutwert einer Zahl X (nicht vorzeichenbehaftet). Der Absolutwert einer negativen Zahl ergibt sich aus der Multiplikation mit -1.

Beispiel:

```
PRINT ABS(2.2),ABS(-3.4)
```

Ausgabe: 2.2 3.4

Kategorie: logischer Operator

Schreibweise: <Ausdruck> AND <Ausdruck>

Funktion: "logisches Und" wird in der Boole´schen Algebra zur Wahrheitsprüfung zweier Ausdrücke benutzt. Das Ergebnis ist nur dann wahr, wenn beide Ausdrücke wahr sind.

Beispiel:

```
100 IF A=5 AND B=3 THEN GOTO 150
```

Verzweigt nur dann zur Programmzeile 150, wenn A den Wert 5 und B den Wert 3 besitzt.

Kategorie: String Funktionen

Schreibweise: ASC("X")

Funktion: Ergibt den ASCII Wert von X. Handelt es sich um einen String, der sich aus mehr als einem Zeichen zusammensetzt, so wird nur der ASCII-Wert des ersten Zeichens ausgegeben. Beinhaltet der String "nichts", so wird die Fehlermeldung ?ILLEGAL QUANTITY ERROR ausgegeben.

Beispiel:

```
PRINT ASC("A")
```

Ausgabe: 65

Kategorie: Numerische Funktionen

Schreibweise: ATN(X)

Funktion: Als Ergebnis dieser mathematischen Funktion erhält man den Arcustangens der Zahl X. Das Ergebnis liegt immer in dem Intervall $-\pi/2$ bis $+\pi/2$.

Beispiel:

```
PRINT ATN(π/2)
```

Ausgabe: 1.00388482

Kategorie: String Funktionen

Schreibweise: CHR$(X)

Funktion: Beim Aufruf dieser Funktion wird eine Zahl X in das entsprechende Zeichen des ASCII-Codes umgewandelt. Die Zahl muß zwischen 0 und 255 liegen.

Beispiel:

```
PRINT CHR$(65)
```

Ausgabe: A

CLOSE

Kategorie: Befehle

Schreibweise: CLOSE X

Funktion: Schließen des Files X, wobei X die logische Filenummer darstellt. Es handelt sich hierbei um die gleiche Filenummer, die vorher bei der OPEN-Anweisung zum Öffnen des Files benutzt wurde (siehe OPEN).

Beispiel:

```
CLOSE 1
```

Schließt die Datei, die zuvor mit der logischen Filenummer 1 mit OPEN geöffnet wurde.

Kategorie: Befehle

Schreibweise: CLR

Funktion: Dieser Befehl löscht alle Variablen, Felder sowie vom Benutzer definierte Funktionen. Er beeinträchtigt, im Gegensatz zu NEW, das eigentliche BASIC-Programm nicht.

Beispiel:

```
10 A=10
20 PRINT A;
30 CLR
40 PRINT A
```

Ausgabe: 10 0

Kategorie: Befehle (Ausgabe)

Schreibweise: CMD X

Funktion: Leitet Daten, die eigentlich auf dem Bildschirm ausgegeben werden, auf ein entsprechendes Peripheriegerät um, z.B. auf Drucker, Diskettenlaufwerk oder Datasette. Hierbei ist X wiederum das File, das zuvor in der OPEN Anweisung benutzt wurde, um das Peripheriegerät anzusprechen. Bei Benutzung der Befehle PRINT und LIST werden die Daten dann auf das logische File übertragen.

Beispiel:

```
10 OPEN 1,4
20 CMD 1
30 LIST
40 CLOSE 1
```

Durch dieses kleine Programm wird ein Programmlisting auf dem Drucker ausgegeben.

CONT

Kategorie: Befehle

Schreibweise: CONT

Funktion: Mit CONT können Programme fortgesetzt werden, die durch die STOP-Taste, den Befehl STOP oder den Befehl END unterbrochen wurden. Das Programm nimmt genau an der Stelle seine Funktion wieder auf, an der es unterbrochen wurde.
Dieser Befehl ist nicht anwendbar, wenn am Programm Änderungen vorgenommen wurden oder das Programm durch eine Fehlermeldung abgebrochen wurde.

COS

Kategorie: Numerische Funktionen

Schreibweise: COS(X)

Funktion: Dieser Befehl ergibt den Cosinus einer Zahl X.
X ist hier das Bogenmaß eines Winkels.

Beispiel:

```
PRINT COS(π/2)
```

Ausgabe: 0

DATA

Kategorie: Anweisung

Schreibweise: DATA X,Y,Z

Funktion: Mit dieser Anweisung kann man bei der Programmierung Informationen ablegen, die mit dem READ-Befehl ausgelesen werden können. Die Informationen können sowohl Zahlen als auch Zeichen (Strings) sein. Bestimmte Zeichen müssen in Hochkommata stehen, wenn sie Komma, Leerzeichen oder den Doppelpunkt enthalten. Die Daten werden der Reihe nach gelesen, d.h. von links nach rechts.

Beispiel:

```
100 DATA 400,410,420,"TEST"
```

Kategorie: Anweisung

Schreibweise: DEF FN F(X)=X*Y

Funktion: Mit dieser Anweisung kann vom Programmierer eine mathematische Funktion definiert werden, die später im Programm nur noch mit dem Funktionsnamen aufgerufen werden kann. Hierbei ist "F" der Funktionsname, "(X)" die Variable und "X*Y" die eigentliche Funktion. Die Funktion selber kann aus einer beliebigen mathematischen Anweisung bestehen.

Beispiel:

```
10 DEF FN F(X)=X*Y
20 Y=10
30 PRINT FN F(8)
```

Ausgabe: 80

DIM

Kategorie: Anweisung

Schreibweise: DIM A(X)

Funktion: Mit dieser Anweisung werden ein Feld (Array) oder eine Matrix (mehrdimensionales Feld) dimensioniert. (X) wird hierbei als Index bezeichnet; man spricht auch von indizierten Variablen, in diesem Falle A(X). A(X) nennt man eindimensionales Feld und A(X,Y) mehrdimensionales Feld.

Beispiel:

```
DIM A(14),B$(2),C%(5,2)
```

Dieses Beispiel generiert 2 eindimensionale Felder und ein zweidimensionales Feld.

END

Kategorie: Befehle

Schreibweise: END

Funktion: Trifft das Programm bei der Ausführung auf diesen Befehl, so wird der Programmablauf beendet und die Meldung "READY." wird ausgegeben. Diese Anweisung ist ähnlich wie die STOP-Anweisung. Nach beiden Befehlen kann das Programm mit CONT wieder fortgesetzt werden. END muß nicht unbedingt am Programmende stehen.

EXP

Kategorie: Numerische Funktionen

Schreibweise: EXP(X)

Funktion: Hiermit läßt sich die X-te Potenz der Konstanten e (2.718281823) berechnen, wobei X nicht größer sein darf als 88.0296919.

Beispiel:

```
PRINT EXP(1)
```

Ausgabe: 2.71828183

Kategorie: Numerische Funktionen

Schreibweise: FN A(X)

Funktion: Ergibt den Funktionswert einer Zahl X, die zuvor in der Funktion mit DEF FN A(X) festgelegt wurde. (s. DEF FN)

Kategorie: Befehle

Schreibweise: FOR X=1 TO 10 STEP 2

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, eine Variable (hier X) als Zähler zu benutzen. Sie müssen den Startwert der Variablen angeben (hier 1), den Endwert (hier 10) und die Schrittweite (hier 2). Geben Sie keine Schrittweite an, so wird automatisch die Schrittweite 1 angenommen. Es kann jede beliebige Gleitpunktzahl benutzt werden.

Beispiel:

```
10 FOR I=2 TO 20 STEP 2
20 PRINT I;
30 NEXT I
```

Gibt alle geradzahligen Zahlen von 2 bis 20 aus.

Kategorie: Numerische Funktionen

Schreibweise: FRE(X)

Funktion: Mit dieser Funktion erhalten Sie die Anzahl freier Bytes, die Sie noch für Ihr Programm zur Verfügung haben. X kann jeden beliebigen Wert annehmen, da es nicht zur Berechnung herangezogen wird.

Beispiel:

```
PRINT FRE(U)
```

Ausgabe: -26634 (wenn kein Programm im Rechner)

Kategorie: Befehle

Schreibweise: GET X$

Funktion: Mit diesem Befehl können Sie jedes Zeichen einzeln von der Tastatur aus einlesen. Das eingelesene Zeichen wird der Variablen, die GET folgt, hier X$, zugeordnet. Wird eine numerische Variable verwendet, z.B. X, und es wird ein anderes Zeichen als eine Zahl eingegeben, so erscheint die Fehlermeldung "SYNTAX ERROR". Daher sollten aus Gründen der Programmsicherheit die Zeichen als Stringvariablen eingelesen und nachher im Programm als Zahlenwerte aufbereitet werden.

Beispiel:

```
10 GET A$: IF A$="" THEN 10
20 PRINT A$;
30 GOTO 10
```

Dieses Programm wartet bis eine Taste gedrückt wird, und gibt das Zeichen auf dem Bildschirm aus.

Kategorie: Befehle (Eingabe)

Schreibweise: GET#1,X$

Funktion: Mit diesem Befehl werden einzelne Zeichen von einem Peripheriegerät eingelesen, welches vorher durch einen OPEN-Befehl angesprochen wurde. Hier ist 1 wieder die logische Filenummer und X$ die Variable, der das eingelesene Zeichen zugeordnet wird. Dieser Befehl entspricht im Prinzip dem GET-Befehl.

Beispiel:

```
10 OPEN 1,8,2,"TEST,S,R"
20 GET#1,A$:IF A$="" THEN 20
30 PRINT A$;
40 IF ST<>64 THEN 20
50 CLOSE 1
```

Dieses Programm liest die Datei "TEST" aus und gibt die Zeichen auf dem Bildschirm aus.

Kategorie: Befehle

Schreibweise: GOSUB XX

Funktion: Mit diesem Befehl wird innerhalb eines Programms zu einem Unterprogramm verzweigt, wobei XX hier die Zeilennummer angibt, an der das Unterprogramm beginnt. Trifft das Programm im Unterprogramm auf die RETURN-Anweisung, so springt das Programm zu der Anweisung zurück, die unmittelbar hinter dem GOSUB-Befehl steht. Achten Sie darauf, daß ein Unterprogramm nie mit einem GOTO verlassen wird, da der Rechner sonst mit der internen Verwaltung der Unterprogramme durcheinanderkommt. Es wird in einem späteren Kapitel anhand eines Beispiels gezeigt, was geschieht, wenn man sich nicht an diese Regel hält.

Beispiel:

```
.
1000 GOSUB 100
1010 PRINT
.
```

In Zeile 1000 wird in ein Unterprogramm verzweigt (Zeile 100). Danach wird mit der Programmausführung in Zeile 1010 fortgefahren.

Kategorie: Befehle

Schreibweise: GOTO XX

Funktion: Mit diesem Befehl springt das Programm an eine durch XX angegebene Zeilennummer des Programms. Dadurch kann der eigentliche Programmablauf, der durch die Zeilennummern bestimmt wird, beeinflußt werden.

Beispiel:

```
.
100 GOTO 1500
.
```

In Zeile 100 springt das Programm in Zeile 1500 und fährt dort mit der Programmausführung fort.

Kategorie: Befehle

Schreibweise: IF (Ausdruck) THEN (Ausdruck)

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, innerhalb des Programms eine Bedingung überprüfen zu lassen. Falls diese Bedingung wahr ist, wird die entsprechende Anweisung hinter THEN ausgeführt. Wird die Bedingung hinter IF als falsch gewertet, so wird alles, was dem THEN folgt, ignoriert, und das Programm fährt in der nächsten Programmzeile fort. Der Ausdruck hinter IF enthält in den meisten Fällen einen Vergleich, z.B. IF A=10 THEN, wobei der Ausdruck hinter THEN jeden beliebigen BASIC-Befehl enthalten kann.

Beispiel:

```
.
100 IF A > 100 THEN END
.
```

Nimmt A einen Wert größer als 100 an, so wird das Programm beendet.

Kategorie: Befehle

Schreibweise: INPUT X oder INPUT"Kommentar";X

Funktion: Mit diesem Befehl steht dem Anwender die Möglichkeit zur Verfügung, dem Programm Daten zu übergeben, die dann durch das Programm entsprechend verarbeitet werden können. Trifft das Programm auf diesen Befehl, so erscheinen auf dem Bildschirm ein Fragezeichen und der Cursor. Jetzt kann der Anwender Daten eingeben und durch Drücken der RETURN-Taste die Eingabe abschließen. Im Programm kann dem INPUT direkt die Variable folgen oder noch , zusätzlich durch einen Kommentar ergänzt werden, wobei zwischen Kommentar und der Variablen stets ein Semikolon stehen muß. Der INPUT-Befehl kann nur innerhalb eines Programms benutzt werden.

Beispiele:

```
10 INPUT "IHREN NAMEN BITTE";A$
20 PRINT "HALLO ";A$
```

Dieses Programm fordert Sie auf, Ihren Namen einzugeben und begrüßt Sie anschließend mit "HALLO".

Kategorie: Befehle (Eingabe)

Schreibweise: INPUT#1,X$

Funktion: INPUT# hat eine ähnliche Funktion wie GET#, nur mit dem Unterschied, daß hier nicht ein Zeichen nach dem anderen zugeordnet wird, sondern bis zu maximal 80 Zeichen auf einmal. 1 ist hier wieder die logische Filenummer des zuvor mit OPEN 1 angesprochenen Peripheriegerätes. X$ ist die Variable, der die eingelesenen Daten zugeordnet werden. Trifft INPUT# beim Lesen der Daten auf ein Semikolon, Komma, Doppelpunkt oder auf ein RETURN (CHR$(13)), so werden diese als das Ende der Variablen interpretiert.

Beispiel:

```
10 OPEN 15,8,15
20 INPUT#15,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
40 CLOSE 15
```

Dieses Programm liest eine Fehlermeldung des Diskettenlaufwerkes aus, und zeigt sie auf dem Bildschirm an.

INT

Kategorie: Numerische Funktionen

Schreibweise: INT(X)

Funktion: Ergibt den ganzzahligen Wert der Variablen X, d.h. es werden die Nachkommastellen einfach abgeschnitten, unabhängig davon, wie groß der Wert hinter der Kommastelle ist. Es findet keine Rundung im üblichen Sinne statt. Die erhaltenen Werte sind immer kleiner oder gleich X. Bei negativen Zahlen werden die Werte für X also dem Betrag nach größer; z.B. INT(-1.23) ergibt -2.

Beispiel:

```
PRINT INT(2.8),INT(2.3),INT(-3.1)
```

Ausgabe: 2 2 -4

Kategorie: String Funktionen

Schreibweise: LEFT$(X$,A)

Funktion: Gibt die linken Zeichen von X$ wieder, angefangen beim ersten linken Zeichen von X$. A bestimmt die Anzahl der Zeichen, die aus X$ gelesen werden. LEFT$(X$,4) liest also die ersten vier linken Zeichen von X$. A darf Werte zwischen 0 und 255 annehmen, sonst wird die Fehlermeldung ILLEGAL QUANTITY ERROR ausgegeben.

Beispiel:

```
10 A$="COMMODORE 64"
20 PRINT LEFT$(A$,9)
```

Ausgabe: COMMODORE

Kategorie: Numerische Funktionen

Schreibweise: LEN(X$)

Funktion: Mit dieser Funktion erhält man die Anzahl der Zeichen von X$. Es werden alle Zeichen im String berücksichtigt, also auch Leerzeichen.

Beispiel:

```
10 A$="COMMODORE 64"
20 PRINT LEN(A$)
```

Ausgabe: 12

Kategorie: Befehle

Schreibweise: LET X=5

Funktion: Der LET-Befehl dient dazu, einer Variablen einen Wert oder String zuzuweisen. Hier würde also der Variablen X der Wert 5 zugeordnet. LET wird jedoch beim Programmieren kaum verwendet, da die Zuweisung X=5 bereits genügt.

Beispiel:

```
LET A=5
```

Weist der Variablen A den Wert 5 zu.

LIST

Kategorie: Befehle

Schreibweise: LIST (Zeilennr.) - (Zeilennr.)
Funktion: Der LIST-Befehl ermöglicht es Ihnen, sich das momentane BASIC-Programm zeilenweise oder ganz zeigen zu lassen. Wird nur LIST eingegeben, so wird das komplette Programm am Bildschirm gezeigt. Geben Sie zusätzlich Zeilennummern an, so werden nur die entsprechenden Zeilen aufgelistet. In Verbindung mit dem CMD-Befehl kann die Ausgabe an ein Peripheriegerät gehen, z.B. Drucker oder Diskette. Bei Verwendung von LIST in einem Programm wird der Programmablauf nach LIST unterbrochen.

Beipiel:

```
LIST          (zeigt das komplette Programmlisting)
LIST 10       (zeigt nur Programmzeile 10)
LIST -100     (zeigt Programmlisting bis zur Zeile 100)
LIST 100-     (zeigt Programmlisting ab Zeile 100 bis
               zum Ende)
LIST 10-20    (zeigt nur die Programmzeilen von 10 bis 20)
```

Kategorie: Befehle

Schreibweise: LOAD "Programmname",GA,SA

Funktion: Mit LOAD können Programme von der DATASETTE oder dem Diskettenlaufwerk in den Programmspeicher geladen werden. Wird nur LOAD eingegeben, wird das nächste Programm von der Kassette eingelesen. Bei Angabe des Programmnamens in Anführungszeichen und der Geräteadresse GA=8 für das Diskettenlaufwerk wird das entsprechende Programm von der Diskette an die interne Speicheradresse 2048 geladen. Das ist normalerweise der BASIC-Anfang. Wird zusätzlich noch die Sekundäradresse SA=1 angegeben, wird das Programm an die Speicheradresse geladen, von der es zuvor abgespeichert wurde.

Beispiel:

```
LOAD "TEST",8
```

Das Programm TEST wird von der Diskette in den Rechner geladen.

LOG

Kategorie: Numerische Funktionen

Schreibweise: LOG(X)

Funktion: Mit dieser Funktion erhalten Sie den natürlichen Logarithmus (Basis e) von X, wobei X größer Null sein muß.

Beispiel:

```
PRINT LOG(2.71828183)
```

Ausgabe: 1

Kategorie: String Funktionen

Schreibweise: MID$(X$,A,B)

Funktion: Diese Funktion definiert einen Teilstring von X$, der B Zeichen von X$ enthält, wobei A das Zeichen bestimmt, von dem ab der Teilstring gebildet wird. Sowohl A als auch B können Werte zwischen 0 und 255 annehmen.

Beispiel:

```
10 A$="COMMODORE 64"
20 PRINT MID$(A$,2,4)
```

Ausgabe: OMMO

NEW

Kategorie: Befehle

Schreibweise: NEW

Funktion: Mit NEW werden das aktuelle Programm im Speicher sowie alle Variablen gelöscht. Grundsätzlich sollte vor jeder neuen Programmeingabe NEW eingegeben werden. Seien Sie jedoch vorsichtig mit diesem Befehl, da Ihr Programm und somit die Programmierarbeit von vielleicht mehreren Stunden verloren ist, falls es nicht vorher abgespeichert wurde.

Sie können ein Programm anstatt mit END auch mit NEW beenden.

Kategorie: Befehle

Schreibweise: NEXT X

Funktion: NEXT bezeichnet das Ende der zuvor mit "FOR X=0 to 10" geöffneten Schleife. Trifft das Programm auf einen NEXT-Befehl, so wird die dazugehörige Laufvariable, hier X, um eins erhöht, falls keine andere Schrittweite angegeben wurde. X wird dann mit dem Endwert verglichen, hier 10, und falls X > 10 ist, wird die Schleife verlassen. Es können mit NEXT mehrere Schleifen auf einmal beendet werden, dazu müssen lediglich alle Laufvariablen mit angegeben und durch Kommata abgetrennt werden, z.B. NEXT X,Y,Z.

Beispiel:

```
10 FOR X=10 TO 0 STEP -1
20 FOR I=1 TO 1000:NEXT I
30 PRINT X;
40 NEXT X
```

NOT

Kategorie: Logischer Operator

Schreibweise: NOT X

Funktion: Mit NOT können Sie das Ergebnis einer Vergleichsoperation, die die Wahrheitswerte WAHR = -1 / FALSCH = 0 enthält, umkehren, d.h. aus WAHR wird FALSCH und umgekehrt.

```
10 A=0
20 IF NOT A < 1 THEN END
30 ..
```

Im obigen Beispiel fährt das Programm mit der Ausführung in Zeile 30 fort.

Kategorie: Befehle

Schreibweise: ON X GOTO 10,20,30 / ON X GOSUB 100,200

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, in Abhängigkeit des Wertes von X nach bestimmten Programmzeilen zu verzweigen bzw. in bestimmte Unterprogramme zu springen, deren Zeilennummern hinter GOTO oder GOSUB aufgeführt sind. Hat X den Wert 1, so würde im obigen Beispiel nach Zeile 10 bzw. in das Unterprogramm ab Zeile 100 verzweigt. Bei geschickter Anwendung kann man mit dieser Befehlsfolge mehrere IF..THEN.. Befehle einsparen.

Kategorie: Befehle (Ein-/Ausgabe)

Schreibweise: OPEN X,GA,SA

Funktion: Der OPEN-Befehl ermöglicht es, Daten zwischen dem Rechner und den Peripheriegeräten auszutauschen. X bezeichnet hier die bereits bei anderen Befehlen erwähnte logische Filenummer. Dies kann eine Zahl zwischen 0 und 255 sein, wobei Zahlen über 128 spezielle Funktionen bei Peripheriegeräten auslösen können. Daher sollten Sie keine Zahlen nehmen, die größer als 128 sind. GA steht für die Geräteadresse. Diese ist in den meisten Fällen für die einzelnen Geräte vorbestimmt, z.B. Diskettenlaufwerk=8, Kassettenlaufwerk=1, Drucker=4. Danach kann in bestimmten Fällen noch die sogenannte Sekundäradresse SA folgen, die ebenfalls durch ein Komma abgetrennt wird. Die Sekundäradresse hat für die einzelnen Peripheriegeräte unterschiedliche Auswirkungen. Beim Drucker hat sie teilweise die Funktion eines Schalters, wobei bestimmte Druckmodi ein- bzw. ausgeschaltet werden. Die genauen Funktionen der Sekundäradresse können Sie den entsprechenden Handbüchern entnehmen.

Kategorie: logischer Operator

Schreibweise: (Ausdruck) OR (Ausdruck)

Funktion: Der logische Operator OR dient ebenfalls dazu, zwei oder mehrere Ausdrücke auf ihren Wahrheitswert hin zu überprüfen, ähnlich wie wir es vom AND her kennen. Der Unterschied liegt darin, daß beim OR lediglich von mehreren Ausdrücken nur einer "wahr" sein muß, um dem gesamten Ausdruck den Wert "wahr" zu geben.

Beispiel:

```
10 A=10:B=5
20 IF A=5 OR B=5 THEN END
30 ..
```

Hier wird in Zeile 20 das Programm beendet, da bereits eine Bedingung erfüllt ist. (B=5)

Kategorie: Numerische Funktionen

Schreibweise: PEEK(X)

Funktion: Ergibt eine Zahl zwischen 0 und 255, die aus einer Speicheradresse X gelesen wird. X muß im Bereich von 0 bis 65535 liegen.

Beispiel:

```
PRINT PEEK(2048)
```

Ausgabe: 0

Kategorie: Befehle

Schreibweise: POKE X,A

Funktion: Dieser Befehl ist in etwa die Umkehrung der PEEK-Funktion. Mit POKE wird ein Wert A in eine Speicherstelle X geschrieben, wobei A einem Wert zwischen 0 und 255 und X einer Speicherstelle zwischen 0 und 65535 entsprechen muß.

Beispiel:

```
POKE 53280,0
```

Setzt die Bildschirmrahmenfarbe auf schwarz.

Kategorie: Numerische Funktionen

Schreibweise: POS(A)

Funktion: Mit dieser Funktion erhalten Sie die Cursorposition in der Bildschirmzeile, an der der nächste Print-Befehl ausgeführt würde. A wird nicht zur Berechnung herangezogen bzw. durch einen Wert belegt, ähnlich wie beim FRE(A) Befehl. Statt A kann auch eine Zahl verwendet werden.

Beispiel:

```
PRINT "TEST" POS(X)
```

Ausgabe: TEST 4

Kategorie: Befehle

Schreibweise: PRINT (Variable) oder PRINT "TEXT"

Funktion: Der PRINT-Befehl ist wohl einer der vielseitigsten Befehle innerhalb des BASIC vom COMMODORE 64 überhaupt. Er wird hauptsächlich dazu benutzt, um Daten oder Mitteilungen vom Programm auf dem Bildschirm auszugeben. Folgt dem PRINT eine Variable, so wird der aktuelle Wert der Variablen ausgedruckt; steht dagegen eine Zeichenkette in Anführungszeichen, so wird die Zeichenkette genauso ausgedruckt.

Beispiel:

```
10 PRINT CHR$(147)
20 PRINT "COMMODORE 64"
```

Dieses Programm löscht den Bildschirm und gibt den Text "COMMODORE 64" aus.

Kategorie: Befehle (Ausgabe)

Schreibweise: PRINT#X,"Daten"

Funktion: Mit PRINT# werden Daten in eine Datei geschrieben, die vorher mit OPEN geöffnet wurde. X bezeichnet wieder die logische Filenummer und "Daten" steht hier für spezielle Befehle oder Variablen, die in die Datei geschrieben werden sollen.

Beispiel:

```
10 OPEN 15,8,15
20 PRINT#15,"I"
30 CLOSE 15
```

Hier wird mit PRINT# über den Befehlskanal des Diskettenlaufwerks der Befehl zum Initialisieren einer Diskette gegeben.

Kategorie: Befehle

Schreibweise: READ X

Funktion: Liest ein Element einer DATA-Zeile und ordnet es der Variablen X zu. Die zu lesenden Elemente müssen mit dem angegebenen Variablentyp übereinstimmen; in diesem Beispiel wären also für X nur numerische Werte zugelassen. Werden mehr READ-Befehle gegeben als DATA-Elemente vorhanden, so wird die Fehlermeldung "?OUT OF DATA ERROR" ausgegeben.

Beispiel:

```
10 READ A$
20 PRINT A$;
30 DATA COMPUTER
```

Kategorie: Anweisung

Schreibweise: REM TEXT

Funktion: Mit REM können innerhalb von Programmen Erklärungen zu einzelnen Programmteilen gegeben werden, ohne daß dadurch der eigentliche Ablauf des Programms beeinflußt wird. Wird die REM-Anweisung in einer Zeile benutzt, so wird alles, was der REM-Anweisung in dieser Zeile folgt, vom Programm ignoriert.

Beispiel:

```
100 REM UNTERPROGRAMME
110 PRINT CHR$(147):REM BILDSCHIRM LÖSCHEN
```

RESTORE

Kategorie: Befehle

Schreibweise: RESTORE

Funktion: Bei Benutzung des READ-Befehls wird immer ein Zeiger auf das nächste zu lesende Element in der DATA-Zeile gesetzt. Mit RESTORE wird dieser Zeiger wieder an den Anfang der DATA-Zeile gesetzt, so daß durch diesen Befehl ein Element mehrmals gelesen werden kann.

Beispiel:

```
 .
100 READ A,B,C
110 RESTORE
120 READ X,Y,Z
 .
200 DATA 45,23,6
```

Kategorie: Befehle

Schreibweise: RETURN

Funktion: Mit diesem Befehl wird ein Unterprogramm beendet. Trifft das Programm auf den RETURN-Befehl, so springt es zu der Zeile, wo der zugehörige GOSUB-Befehl stand, und fährt dort mit der Ausführung des Programms fort.

Beispiel:

```
10 PRINT CHR$(147)
 .
40 RETURN
 .
 .
100 GOSUB 10
```

Kategorie: String Funktionen

Schreibweise: RIGHT$ (X$,A)

Funktion: Ergibt einen Teilstring der rechten Seite von X$, dessen Länge durch den Wert A bestimmt wird. A kann Werte zwischen 0 und 255 annehmen. Ist der Wert größer als die ursprüngliche Stringlänge, so wird der ganze String wiedergegeben.

Beispiel:

```
10 A$="COMMODORE 64"
20 PRINT RIGHT$(A$,2)
```

Ausgabe: 64

Kategorie: Numerische Funktionen

Schreibweise: RND (X)

Funktion: RND erzeugt eine Zufallszahl zwischen 0.0 und 1.0. Der Wert von X hat bis auf das Vorzeichen keine Bedeutung. Bei positiven X-Werten wird stets die gleiche Folge von Zufallszahlen bei gegebenem Startwert X erzeugt. Unterschiedliche Zahlenfolgen erhält man durch Änderung der Startwerte.

Beispiel:

```
10 FOR I=1 TO 6
20 A=INT (6*RND(1))+1
30 PRINT A;
40 NEXT I
```

Dieses Programm erzeugt 6 Zufallszahlen im Bereich von 1 bis 6.

Kategorie: Befehle

Schreibweise: RUN (Zeilennummer)

Funktion: Mit RUN wird ein Programm gestartet. Bei Bedarf kann zusätzlich eine Zeilennummer mit angegeben werden, von der aus das Programm starten soll. Es werden automatisch alle Variablen auf Null gesetzt. Will man dies vermeiden, so kann man den GOTO-Befehl mit Angabe der Zeilennummer verwenden. Dabei bleiben die Variablenwerte erhalten.

Beispiel:

```
RUN    - (startet ein Programm)
RUN 50 - (startet ein Programm ab Zeile 50)
```

SAVE

Kategorie: Befehle

Schreibweise: SAVE "Name",GA

Funktion: SAVE speichert ein im Programmspeicher enthaltenes Programm auf Kassette oder Diskette ab. Wird SAVE ohne Angabe der Geräteadresse GA angewendet, so wird automatisch auf die Kassette abgespeichert. Durch Angabe der Geräteadresse 8 kann man das Programm auf das Diskettenlaufwerk abspeichern.

Beispiel:

```
SAVE"TEST 1",8  - (speichert ein Programm auf Diskette)
SAVE"TEST 2"    - (speichert ein Programm auf Kassette)
```

Kategorie: Numerische Funktionen

Schreibweise: SGN (X)

Funktion: Mit SGN (X) erhalten sie einen ganzzahligen Wert, welcher abhängig vom Vorzeichen von X ist. Bei X > 0 erhält man 1, bei X = 0 erhält man 0 und bei X < 0 erhält man -1.

Beispiel:

```
PRINT SGN(4),SGN(0),SGN(-3)
```

Ausgabe: 1 0 -1

SIN

Kategorie: Numerische Funktionen

Schreibweise: SIN (X)

Funktion: Mit dieser Funktion erhalten Sie den Sinus des Winkels von X, wobei X im Bogenmaß anzugeben ist.

Beispiel:

```
PRINT SIN(π/2)
```

Ausgabe: 1

Wollen Sie mit Grad anstatt mit dem Bogenmaß rechnen, so müssen Sie den Wert noch mit π/180 multiplizieren.

Beispiel für 90 Grad:

```
PRINT SIN(90*π/180)
```

Ausgabe: 1

Kategorie: String Funktionen

Schreibweise: SPC (X)

Funktion: Mit SPC hat man die Möglichkeit, X Zeichen zu überspringen, um dann an der Stelle mit PRINT eine Ausgabe auf dem Bildschirm zu erhalten. X kann hier wieder Werte zwischen 0 und 255 annehmen.

Beispiel:

```
PRINT "TEST";SPC(10);"TEST"
```

Ausgabe: TEST TEST

Kategorie: Numerische Funktionen

Schreibweise: SQR (X)

Funktion: Ergibt den Wert der Quadratwurzel von X. X darf keine negativen Werte annehmen.

Beispiel:

```
PRINT SQR(64),SQR(30.25)
```

Ausgabe: 8 5.5

STEP

Kategorie: Befehle

Schreibweise: STEP X

Funktion: STEP wird bei der Programmierung von Schleifen benutzt, um die Schrittweite für die Laufvariable anzugeben. X kann jeden beliebigen Wert außer Null annehmen. Wird STEP nicht benutzt, so ist die Schrittweite gleich 1.

Beispiel:

```
10 FOR I=1 TO 2.1 STEP .2
20 PRINT I;
30 NEXT
```

Ausgabe: 1 1.2 1.4 1.6 1.8 2

Kategorie: Befehle

Schreibweise: STOP

Funktion: STOP wird innerhalb eines Programms benutzt, um den Programmablauf an dieser Stelle zu unterbrechen. Als Meldung auf dem Bildschirm erscheint BREAK IN X, wobei X die Zeilennummer angibt, in der unterbrochen wurde. Mit CONT kann die Programmausführung wieder aufgenommen werden.

Beispiel:

```
10 PRINT CHR$(147)
20 GOSUB 1200
30 STOP
 .
```

Dieses Programm bricht in Zeile 30 mit der Meldung

BREAK IN 30

ab. Durch Eingabe von CONT kann das Programm fortgeführt werden.

STR$

Kategorie: String Funktionen

Schreibweise: STR$ (X)

Funktion: Durch STR$ wird X in einen String umgewandelt. Ist X positiv oder Null, so beginnt der String mit einem Leerzeichen.

Beispiel:

```
10 A=1234.23
20 A$=STR$(A)
30 PRINT A$;
```

Ausgabe: 1234.23

Der String A$ setzt sich in diesem Fall aus 8 Zeichen zusammen!

SYS

Kategorie: Befehle

Schreibweise: SYS X

Funktion: Mit SYS wird ein Maschinensprache-Programm aufgerufen, das an der Adresse X beginnt, wobei X Werte zwischen 0 und 65535 annehmen kann. Ein bekannter SYS-Befehl ist der System-Kaltstart mit SYS 64738.

Beispiel:

```
SYS 58692
```

Dieser SYS-Befehl löscht den Bildschirm.

Kategorie: String Funktionen

Schreibweise: TAB (X)

Funktion: Mit TAB kann der Cursor an eine durch X bestimmte Zeilenposition gebracht werden. Der Wert von X kann zwischen 0 und 255 liegen. Die Zählung beginnt an der äußersten linken Position der momentanen Zeile. TAB sollte nur in Verbindung mit dem PRINT-Befehl verwendet werden, da er zusammen mit PRINT# (z.B. von einigen Druckern) nicht interpretiert werden kann.

Beispiel:

```
PRINT "TEST";TAB(5);"TEST"
```

Ausgabe: TEST TEST

TAN

Kategorie: Numerische Funktionen

Schreibweise: TAN (X)

Funktion: Ergibt den Tangens des Winkels von X, wobei X im Bogenmaß angegeben werden muß.

Beispiel:

```
PRINT TAN(π/4)
```

Ausgabe: .999999999

Kategorie: Numerische Funktionen

Schreibweise: USR (X)

Funktion: Mit USR wird in ein Maschinenunterprogramm verzweigt, dessen Startwert zuvor in den Speicherstellen 785-786 abgelegt sein muß. Dies geschieht durch entsprechende POKE-Befehle. Der Wert von X wird an das Maschinenprogramm übergeben, welches selbst einen anderen Wert an das Basic-Programm zurückgibt. Diese Möglichkeit der Parameterübergabe existiert bei dem SYS-Befehl nicht.

Beispiel:

```
 .
 .
50 USR(5)
 .
 .
```

Übergibt in Zeile 50 den Wert 5 an ein Maschinenprogramm.

Kategorie: Numerische Funktion

Schreibweise: VAL (X$)

Funktion: VAL wandelt X$ in einen numerischen Wert um. Trifft VAL im String auf ein Zeichen, welches nicht in eine Zahl umgewandelt werden kann, z.B. A, wird nur der Teil bis zum A in einen Zahlenwert umgewandelt. Ist das erste Zeichen, außer Leer-, Plus- oder Minuszeichen, nicht in eine Zahl überführbar, so ergibt sich der Wert Null.

Beispiel:

```
10 A$="12 TV GERAETE"
20 PRINT VAL(A$)
```

Ausgabe: 12

VERIFY

Kategorie: Befehle

Schreibweise: VERIFY "Datei",GA

Funktion: Mit VERIFY kann ein gerade abgespeichertes Programm mit dem im Speicher befindlichen Programm verglichen werden; hiermit kann also überprüft werden, ob ein Programm ordnungsgemäß abgespeichert wurde. GA gibt hier wieder die Geräteadresse an. Diese kann bei Kassettenbetrieb vernachlässigt werden. Mit dem Befehl VERIFY "*",8 wird automatisch das zuletzt abgespeicherte Programm auf Diskette mit dem im Speicher befindlichen verglichen.

Beispiel:

VERIFY "TEST",8

Überprüft das Programm TEST auf Diskette mit dem Programm im Rechner. Wird kein Fehler gefunden, gibt der Rechner die Meldung

OK

aus.

WAIT

Kategorie: Befehle

Schreibweise: WAIT X,Y

Funktion: Mit WAIT wird die Programmausführung solange angehalten, bis die Speicherstelle X den Wert Y angenommen hat. Dieser Befehl wird eigentlich recht selten benutzt.

Beispiel:

```
 .
 .
100 WAIT 198,1
 .
 .
```

Das Programm wartet in Zeile 100 auf einen Tastendruck. Anschließend wird mit der Programmausführung fortgefahren.

ASCII-CODES (GROß-/GRAFIKMODUS)

Dez.	Hex.	
1	1	
2	2	
3	3	
4	4	
5	5	Schriftfarbe weiß
6	6	
7	7	
8	8	SHIFT/COMMODORE blockiert
9	9	SHIFT/COMMODORE entriegelt
10	0A	
11	0B	
12	0C	
13	0D	CARRIAGE RETURN
14	0E	Groß-/Kleinschriftmodus
15	0F	
16	10	
17	11	Cursor nach unten
18	12	Revers einschalten
19	13	Cursor HOME
20	14	Delete (ein Zeichen löschen)
21	15	
22	16	
23	17	
24	18	
25	19	
26	1A	
27	1B	
28	1C	Schriftfarbe rot
29	1D	Cursor rechts
30	1E	Schriftfarbe grün
31	1F	Schriftfarbe blau
32	20	Leerzeichen
33	21	!

Dez.	Hex.	
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J

Dez.	Hex.	
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	£
93	5D	]
94	5E	↑
95	5F	←
96	60	—
97	61	♠
98	62	|
99	63	—
100	64	—
101	65	—
102	66	—
103	67	|
104	68	|
105	69	╮
106	6A	╰
107	6B	╯
108	6C	╰
109	6D	╲
110	6E	╱
111	6F	╭
112	70	╮
113	71	●
114	72	—
115	73	●

Dez.	Hex.	
116	74	⎸
117	75	⎺
118	76	⎽
119	77	▯
120	78	⎼
121	79	⎸
122	7A	♠
123	7B	┼
124	7C	▒
125	7D	⎸
126	7E	π
127	7F	◥
128	80	
129	81	Schriftfarbe orange
130	82	
131	83	
132	84	
133	85	Funktionstaste 1
134	86	Funktionstaste 3
135	87	Funktionstaste 5
136	88	Funktionstaste 7
137	89	Funktionstaste 2
138	8A	Funktionstaste 4
139	8B	Funktionstaste 6
140	8C	Funktionstaste 8
141	8D	SHIFT + CARRIAGE RETURN
142	8E	Groß-/Grafikmodus
143	8F	
144	90	Schriftfarbe schwarz
145	91	Cursor nach oben
146	92	Revers ausschalten
147	93	SHIFT + CLR/HOME
148	94	INST-Taste
149	95	Schriftfarbe braun
150	96	Schriftfarbe hellrot
151	97	Schriftfarbe grau 1
152	98	Schriftfarbe grau 2
153	99	Schriftfarbe hellgrün
154	9A	Schriftfarbe hellblau
155	9B	Schriftfarbe grau 3
156	9C	Schriftfarbe violett

Dez.	Hex.	
157	9D	Cursor links
158	9E	Schriftfarbe gelb
159	9F	Schriftfarbe türkis
160	A0	Leerzeichen
161	A1	▮
162	A2	▬
163	A3	▔
164	A4	▁
165	A5	▏
166	A6	▒
167	A7	▕
168	A8	▩
169	A9	◤
170	AA	▕
171	AB	├
172	AC	▪
173	AD	└
174	AE	┐
175	AF	▬
176	B0	┌
177	B1	┴
178	B2	┬
179	B3	┤
180	B4	▏
181	B5	▮
182	B6	▮
183	B7	▔
184	B8	▬
185	B9	▬
186	BA	┘
187	BB	▪
188	BC	▪
189	BD	┘
190	BE	▪
191	BF	▪

192 bis 223 sind mit den Codes 96 bis 127 identisch.
224 bis 254 sind mit den Codes 160 bis 190 identisch.
255 ist mit Code 126 identisch

ASCII-CODES (GROß-/KLEINSCHRIFTMODUS)

Dez.	Hex.	
1	1	
2	2	
3	3	
4	4	
5	5	Schriftfarbe weiß
6	6	
7	7	
8	8	SHIFT/COMMODORE blockiert
9	9	SHIFT/COMMODORE entriegelt
10	0A	
11	0B	
12	0C	
13	0D	CARRIAGE RETURN
14	0E	Groß-/Kleinschriftmodus
15	0F	
16	10	
17	11	Cursor nach unten
18	12	Revers einschalten
19	13	Cursor HOME
20	14	Delete (ein Zeichen löschen)
21	15	
22	16	
23	17	
24	18	
25	19	
26	1A	
27	1B	
28	1C	Schriftfarbe rot
29	1D	Cursor rechts
30	1E	Schriftfarbe grün
31	1F	Schriftfarbe blau
32	20	Leerzeichen
33	21	!

Dez.	Hex.	
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	a
66	42	b
67	43	c
68	44	d
69	45	e
70	46	f
71	47	g
72	48	h
73	49	i
74	4A	j

Dez.	Hex.	
75	4B	k
76	4C	l
77	4D	m
78	4E	n
79	4F	o
80	50	p
81	51	q
82	52	r
83	53	s
84	54	t
85	55	u
86	56	v
87	57	w
88	58	x
89	59	y
90	5A	z
91	5B	[
92	5C	£
93	5D	]
94	5E	↑
95	5F	←
96	60	-
97	61	A
98	62	B
99	63	C
100	64	D
101	65	E
102	66	F
103	67	G
104	68	H
105	69	I
106	6A	J
107	6B	K
108	6C	L
109	6D	M
110	6E	N
111	6F	O
112	70	P
113	71	Q
114	72	R
115	73	S

Dez.	Hex.	
116	74	T
117	75	U
118	76	V
119	77	W
120	78	X
121	79	Y
122	7A	Z
123	7B	+
124	7C	▒
125	7D	|
126	7E	▚
127	7F	▒
128	80	
129	81	Schriftfarbe orange
130	82	
131	83	
132	84	
133	85	Funktionstaste 1
134	86	Funktionstaste 3
135	87	Funktionstaste 5
136	88	Funktionstaste 7
137	89	Funktionstaste 2
138	8A	Funktionstaste 4
139	8B	Funktionstaste 6
140	8C	Funktionstaste 8
141	8D	SHIFT + CARRIAGE RETURN
142	8E	Groß-/Grafikmodus
143	8F	
144	90	Schriftfarbe schwarz
145	91	Cursor nach oben
146	92	Revers ausschalten
147	93	SHIFT + CLR/HOME
148	94	INST-Taste
149	95	Schriftfarbe braun
150	96	Schriftfarbe hellrot
151	97	Schriftfarbe grau 1
152	98	Schriftfarbe grau 2
153	99	Schriftfarbe hellgrün
154	9A	Schriftfarbe hellblau
155	9B	Schriftfarbe grau 3
156	9C	Schriftfarbe violett

Dez.	Hex.	
157	9D	Cursor links
158	9E	Schriftfarbe gelb
159	9F	Schriftfarbe türkis
160	A0	Leerzeichen
161	A1	
162	A2	
163	A3	
164	A4	
165	A5	
166	A6	
167	A7	
168	A8	
169	A9	
170	AA	
171	AB	
172	AC	
173	AD	
174	AE	
175	AF	
176	B0	
177	B1	
178	B2	
179	B3	
180	B4	
181	B5	
182	B6	
183	B7	
184	B8	
185	B9	
186	BA	
187	BB	
188	BC	
189	BD	
190	BE	
191	BF	

192 bis 223 sind mit den Codes 96 bis 127 identisch.
224 bis 254 sind mit den Codes 160 bis 190 identisch.
255 ist mit Code 126 identisch

ANHANG 2

SACHREGISTER

ABS....229
Algorithmus....9
AND....42 f., 209
Arrays....141, 144 ff.
ASC....56 ff.
ASCII-Code....21, 202
ATN....51

Basic....9
Bedingte Programmsprünge....80 ff.
Berechnete Sprungbefehle....101 ff.
Bit....24
Bogenmaß....49
Bubble-Sort-Verfahren....220 ff.
Byte....24

Carriage Return....38
CHR$....56 ff.
CLOSE....225
CLR....69
CONT....237
COS....49
Cosinus....49

DATA....133 ff., 157, 166
Datenfluß....11
Datenflußplan....11 ff., 32, 78
Dateiverwaltung....155, 168, 173, 223 ff.
DEF FN....54
DIM....146 ff.
Direktmodus....37
Dokumentation....18, 33
Dualsystem....22 f.

Eindimensionale Felder.....141 ff.
END.....40 f., 87 f.
EXP.....51 f.

Felder.....141, 144
Feldvariable.....144 f.
Flag.....222
FOR..TO..NEXT.....90 ff.
FOR..NEXT-Schleifen.....90 ff.
FORTRAN.....9
FRE.....127
Funktionen.....49 ff., 54 ff.

Ganzzahlvariable (s.a. Integervariable).....41 f.
GET.....118 ff.
GET#.....225
GOSUB.....174
GOTO.....77

Hexadezimalsystem.....25 ff.
High-Byte.....22 ff.

IF..THEN.....80 ff.
Indizierte Variable (s.a. Feldvariable).....144 ff.
INPUT.....17, 34 f.,118, 200, 201
INT.....52 f., 55
Integervariable (s.a. Ganzzahlvariable).....41

Konnektor.....16, 79, 99, 101
Kreisumfang.....49 f.

LEFT$.....66 ff.
LET.....35 f.
LEN.....69 f.
Lineare Programmablaufpläne.....31 f.
Line Feed.....38
LIST.....258
LOAD.....106, 223 f.
Logarithmus.....51
LOG.....51 f.
logischer Operator.....44, 110, 209
Low-Byte.....22 ff.

Maschinensprache.....10
Mehrdimensionale Felder.....163 ff.
Menü.....110, 195 ff.
MID$.....68 ff.
Multistatements.....33, 59

NEW.....44, 69
Nibble.....27
NOT.....44
Numerische Funktionen.....49 ff.

ON GOSUB.....191
ON GOTO.....102 ff., 110, 113
OPEN.....224 f.
OR.....43 f., 110, 209

PEEK.....130
POKE.....25, 131, 139
POS.....127 f.
Positionierung d. Cursors.....210 f.
PRINT.....17, 36 ff.
Programm.....9
Programmablaufplan.....11 f., 15 f., 32, 79, 97 ff.
Programmierschablone.....11 f.

Rad.....50
READ.....133 ff., 157, 166
Realvariable.....41
Rechenlehrgang.....105 ff., 123, 177 ff., 186 ff.
REM.....40
Reservierte Variablen.....43
RESTORE.....135 ff.
RETURN.....174
RIGHT$.....67
RND.....54 f.
RUN.....17, 37

SAVE.....105, 224 ff.
Schrittweite.....91 f.
SGN.....52
SIN.....49

Sinus.....49
Sortierverfahren.....220 ff.
SPC.....63 f.
SQR.....51
STEP.....91 f.
STOP.....286
STR$.....71 f.
String.....65 ff. 133, 137 f.
Stringvariable.....42, 65 ff.
Stufen d. Programmierens.....19, 31 ff.
SYS.....128, 139

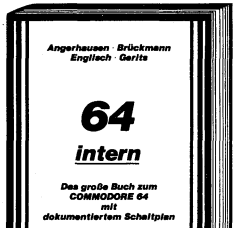
TAB.....63 f.
Tabulator.....39 f.
TAN.....49
Textvariable (Stringvariable).....42
TI.....43

Unbedingte Programmsprünge.....77 ff.
Unterprogramm.....174 ff.

VAL.....70 f.
Variablen.....34 ff., 41 ff.
Variableninitialisierung.....61 f.
Verschachtelung v. Schleifen.....92 f.

WAIT.....129

Zahlensysteme.....22
Zehnerschritte.....33
Zufallsfunktion.....54 f.
Zufallszahlen.....55
Zwischenspeicherung.....112 f., 221



Angerhausen/Brückmann/Englisch/Gerits
64 Intern
Das große Buch zum Commodore 64 mit dokumentiertem Schaltplan

Die Herausforderung für jeden ernsthaften Anwender! Alles über Technik, Betriebssystem und fortgeschrittene Programmierung des Commodore 64. Mit ausführlichem ROM-Listing, sorgfältig dokumentierten Originalschaltplänen zum Ausklappen, zahlreichen Abbildungen, Schaltbildern, Blockdiagrammen und - natürlich - mit anspruchsvollen Programmen. Mit diesem unentbehrlichen Buch lernen Sie Ihren C 64 erst richtig kennen.

352 Seiten, 2 Schaltpläne, DM 69,-
ISBN 3-89011-000-2

Brückmann/Gerits/Wiens
Das große Druckerbuch
369 Seiten, DM 49,-
ISBN 3-89011-020-7

Mit diesem Buch meistern Sie absolut jedes Drucker-Problem. Ob Sekundäradressen, Schnittstellen, Steuerzeichen, formatierte Datenausgabe oder Grafik-Hardcopy: alles hervorragend erklärt. Selbstverständlich wieder viele nützliche Programme zum Abtippen; außerdem wichtige Hilfen zur Druckeranpassung, ein Betriebssystemlisting des MPS 801 und ein eigenes Kapitel zum VC-1520. Jetzt holen Sie das Optimum aus Ihrem Drucker heraus!

Das Standardwerk zur Floppy VC 1541. Alles über Diskettenprogrammierung vom Einsteiger bis zum Profi. Neben grundlegenden Informationen zum DOS, zu den Systembefehlen und Fehlermeldungen stehen mehrere Kapitel zur praktischen Dateiverwaltung mit der Floppy. Umfangreiches, dokumentiertes DOS-Listing.Dazu eine Fundgrube verschiedenster Programme und Hilfsroutinen, die das Buch für jeden Floppy-Anwender einfach zur Pflichtlektüre machen.

Englisch/Szczepanowski
Das große Floppy-Buch
482 Seiten, DM 49,-
ISBN 3-89011-006-3

Selbsthilfe spart Zeit, Ärger und Geld - gerade Probleme wie Floppy-Justage oder Reparaturen der Platine sind mit oft einfachen Mitteln zu lösen. Anleitungen zur Behebung der meisten Störfälle, Ersatzteillisten und eine Einführung in Mechanik und Elektronik des Laufwerks. Natürlich gehören auch genaue Angaben zu Werkzeug und Arbeitsmaterial zum Buch, das in jeder Beziehung für "effektiv und preiswert" steht.

Hermann
VC-1541 Pflegen und Reparieren
ca. 200 Seiten, DM 49,-
ISBN 3-89011-179-7

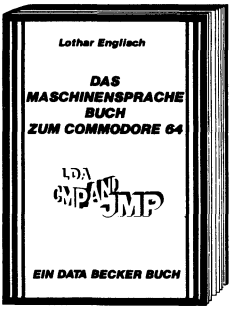
Das Superbuch, das Ihnen zeigt, was alles in Ihrem Rekorder steckt. Informiert detailliert und leichtverständlich über Datasette und Cassettenspeicherung. Mit den Spitzenprogrammen Autostart, Catalog (sucht und lädt automatisch!), Backup von und auf Floppy, Save von Speicherbereichen und einem neuen Cassetten-Betriebssystem mit dem 10-20 mal schnelleren (!) Fasttape. Außerdem weitere nützliche Hinweise (Kopfjustage, Kontroll-Lautsprecher) und Programme.

Paulissen
Das Cassettenbuch zum Commodore 64 und VC-20
190 Seiten, DM 29,-
ISBN 3-89011-030-4

Brückmann
Der Commodore 64 und der Rest der Welt
229 Seiten, DM 49,-
ISBN 3-89011-015-0

Literatur speziell für den engagierten Hobbyelektroniker vom fähigen Techniker zusammengestellt. Schwerpunkt sind ausgesuchte Ideen zu verschiedenen Einsatzmöglichkeiten des C 64: Motorsteuerung, A/D-Wandler, Spannungs- und Temperaturmessung und Lichtorgel. Dazu eine Reihe hochinteressanter Schaltungen zum Nachbau: EPROM-Programmer, Sprachsynthesizer, Frequenzzähler und noch mehr.

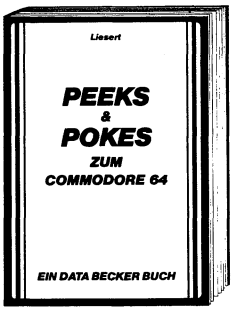
Ein Bestseller, der erfolgreich und umfassend in die Maschinensprache einführt. Sie lernen Aufbau und Arbeitsweise des 6510 Mikroprozessors kennen, erfahren Wichtiges über Eingabe und Start von Maschinenprogrammen sowie über den Umgang mit Monitor, Assembler und Disassembler. Assembler und Disassembler sind im Buch als Programme ebenso enthalten wie ein Einzelschrittsimulator. Viele ausführlich beschriebene Beispielprogramme und Routinen machen Ihnen den Einstieg leicht.



Englisch
Das Maschinensprachebuch zum Commodore 64
201 Seiten, DM 39,-
ISBN 3-89011-008-8

Sie haben den Einstieg in die Maschinensprache geschafft? Dann werden Sie jetzt zum Profi! Von der Problemanalyse bis zum Maschinenspracheprogrammierung eingeführt. Dazu wieder viele Beispielprogramme, komplette Maschinenroutinen und wichtige Tips & Tricks zur Maschinenprogrammierung und zur Arbeit mit dem Betriebssystem.

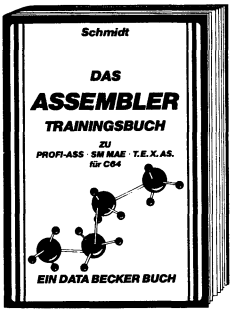
Englisch
Das Maschinensprachebuch für Fortgeschrittene zum Commodore 64
206 Seiten, DM 39,-
ISBN 3-89011-022-3



Liesert
Peeks & Pokes zum Commodore 64
177 Seiten, DM 29,-
ISBN 3-89011-032-0

Leichtverständlich wird hier der Umgang mit PEEK- und POKE-Befehlen erklärt, die vieles vereinfachen, was sonst komplizierte Maschinenroutinen nötig machen würde. Dazu nützliche POKEs und Ihre Anwendungsmöglichkeiten. Außerdem Grundlegendes zum Aufbau des C-64: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks, Charakter-Generator, Sprite-Register und vieles mehr. Mit einer ersten Einführung in die Maschinensprache und etlichen Beispielprogrammen.

Schmidt
Assembler Trainingsbuch
ca. 250 Seiten, DM 39,-
erscheint April 1985
ISBN 3-89011-071-1



Dem interessierten Anfänger werden hier die weitverbreiteten Assembler Profimat, MAE 64 und T.EX.AS. ausführlich anhand von Übungen und Beispielen erklärt und aufbauend eine konsequente Einführung in die Maschinensprache vermittelt. Gleichzeitig ein fundiertes Nachschlagewerk: Ein umfassender und übersichtlicher Anhang mit Erläuterungen aller wichtigen Begriffe sowie ein reichhaltiges Stichwortverzeichnis ergänzen dieses Trainingsbuch optimal.

Bedienerfreundlich und erfolgreich in BASIC programmieren ist kein Privileg von Fachleuten. Wie man es macht, verraten die Software-Autoren aus dem Hause DATA BEKKER: Menuesteuerung, Maskenaufbau, Parameterisierung und Dokumentation sind die Stichworte. Dazu die neue leistungsfähige Datenstruktur QUISAM mit lauffertigen Beispielprogrammen.

Einfach besser programmieren!

Angerhausen/Becker/
Gerits/Schellenberger
64 für Profis
302 Seiten, DM 49,-
ISBN 3-89011-007-X

Eine umfassende, praxisorientierte Einführung in den Komplex Dateiverwaltung, Datenbanken, Datenbanksprachen und Expertensysteme. Erklärt werden logische und physische Datenstrukturen oder sequentieller und Direktzugriff. Wer wissen will, wie ein Hashing-Algorithmus aufgebaut ist oder wie man ein komplettes Dateiverwaltungsprogramm erstellt (das im Buch als ausführliches Listing enthalten ist), der braucht dieses Superbuch.

Baloui
Alles über Datenbanken und Dateiverwaltung für den Commodore 64
222 Seiten, DM 39,-
ISBN 3-89011-054-1

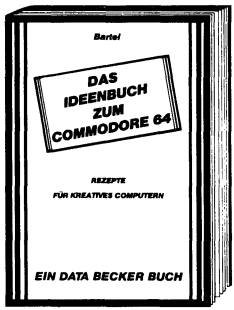
Voß
Das Schulbuch zum Commodore 64
300 Seiten, DM 49,-
ISBN 3-89011-019-3

Was liegt näher, als den Commodore 64 auch für schulische Zwecke einzusetzen? Hilfestellung in jeder Beziehung bietet dieses Schulbuch, dessen Stoff von erfahrenen Pädagogen didaktisch aufbereitet und strukturiert wurde. So lernt man nicht nur die Computeranwendung in den Fächern Mathematik, Physik, Chemie, Biologie, Fremdsprachen und Geographie, sondern es bleibt auch einiges Wissen über Elektronik und Informatik hängen.

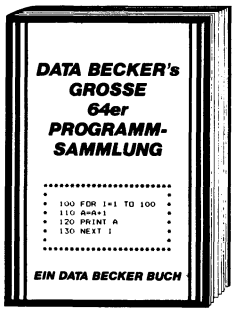
Sauer
Das Trainingsbuch zu LOGO
230 Seiten, DM 39,-
ISBN 3-89011-044-4

LOGO - eine bemerkenswerte Sprache nicht nur für Kinder sondern für viele Bereiche. Diese tiefgreifende Einführung bietet Ihnen sinnvolles Erlernen und Training der vielen Möglichkeiten, die LOGO bietet. Aus dem Inhalt: Grafikprogrammierung, Wörter- und Listenverarbeitung, Funktionsplotter, Maskengenerator, 3-D-Grafik, Prozeduren, Rekursion, Sprites und Musik, und vieles mehr.

Hier kommt das Allroundtalent des C64 voll zum Zuge, mit pfiffigen Programmen zum Nutzen und Lernen: Gedichte vom Computer, Einladung zur Party, Werbebriefe, Autokostenberechnung, Rezeptkartei, Gesundheitsarchiv, Handarbeitshilfen und noch mehr. Viele Anregungen, leichtverständlich und spannend geschrieben. Für jeden 64er-Anwender unbedingt empfehlenswert!



Bartel
Das Ideenbuch zum Commodore 64
Rezepte für kreatives Computern
243 Seiten, DM 29,-
ISBN 3-89011-027-4



DATA BECKERs große 64er Programmsammlung
252 Seiten, DM 49,-
ISBN 3-89011-014-2

Mehr als 50 interessante Anwendungsprogramme aus allen Bereichen, zusammengestellt von 50 erfolgreichen Autoren. Vielfalt und Abwechslung sind garantiert! Spiele, Graphik & Sound, Mathematik, Hilfsprogramme und auch größere Anwendungsprogramme. Dazu wertvolle Tips & Tricks zum Selbermachen. Da ist mit Sicherheit für jeden etwas dabei!

Walkowiak
Adventures - und wie man sie programmiert
225 Seiten, DM 39,-
ISBN 3-89011-043-6

Ein faszinierender Führer in die phantastische Welt der Abenteuerspiele. Hier läßt sich ein arrivierter Autor in die Karten gucken: er zeigt, wie Adventures funktionieren, wie man sie erfolgreich spielt und wie man eigene Adventures programmiert. Der Clou des Buches ist neben fertigen Adventures zum Abtippen ein kompletter ADVENTURE-GENERATOR, mit dem das Selbsterstellen packender Adventures zum Kinderspiel wird. Achtung: dieses buch macht süchtig!

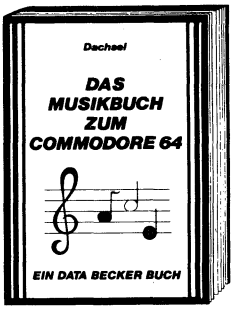
Sie wollten schcon immer mal ein Telespiel selbst programmieren? Hier ist für Sie das top-Buch, zugeschnitten auf den Commodore 64 und mit Berücksichtigung des Commodore 128! Schrittweise lernen Sie zu programmieren, wie man Pac Man durchs Labyrinth schleust oder wie Captain Future spannende Abenteuer in fremden Galaxien überlebt. Handfeste Anwendungen mit vielen Beispielen, Listings und Programmirtips. Auch mit wenig Programmir-Praxis stellen sich schnell überraschende Erfolge ein.

Linden
Superspiele - selbst gemacht
ca. 200 Seiten, DM 29,-
erscheint Mai 1985
ISBN 3-89011-087-8

Der Bestseller zur Grafikprogrammierung des C 64 vom Autor der berühmten Supergrafik. Für Einsteiger, Fortgeschrittene und Profis. Bringt alles über Sprites, High-Res-Graphik und Multicolor bis hin zu 3-D und CAD. Unzählige Superprogramme und Routinen zum Abtippen. Grafik ist zwar eine der großen Stärken des C 64, allerdings bleibt der Zugriff darauf mit BASIC gerade für den Anfänger ohne Anleitung wohl ein Wunschtraum. Mit diesem Buch nicht mehr!

Plenge
Das Grafikbuch zum Commodore 64
295 Seiten, DM 39,-
ISBN3-89011-011-8

Nutzen Sie die Klangmöglichkeiten des C 64! Neben einer kurzen Einführung in die Computermusik finden Sie hier Informationen zu Soundregistern, ADSR-Programmierung, Synchronisation und Ringmodulation. Zahlreiche Beispiele für Sound- und Songprogrammierung sowie wichtige Routinen runden den Inhalt ab. Nicht vergessen wurden auch Themen wie beispielsweise der Anschluß an eine Stereoanlage oder die Verarbeitung externer Tonsignale. Also, Komponisten, ans Werk!



Dachsel
Das Musikbuch zum Commodore 64
208 Seiten, DM 39,-
ISBN 3-89011-012-6

Szczepanowski/Plenge
Das Trainingsbuch zum Simons Basic
380 Seiten, DM 49,-
ISBN 3-89011-009-6

Wer den großen Programmierkomfort, den SIMONs BASIC bietet, voll nutzen möchte, der muß mit den einzelnen Befehlen richtig umgehen können. Da aber das nicht gerade umfangreiche Handbuch sowie auch die Problempunkte, die dieses BASIC aufweist, dem Anwender einige Stolpersteine in den Weg legen, ist das Trainingsbuch ein "Muß" für jeden, der den optimalen Weg zu ausgesprochen leistungsfähigen Programmen gehen will.

Schäfer
Das Handbuch zur DFÜ Datenfernübertragung mit dem Commodore 64
173 Seiten, DM 39,-
ISBN 3-89011-058-4

Die Datenübertagungswelle rollt! Dieses Handbuch bietet eine praxisorientierte Einführung in die Grundlagen der Datenübertragung (Akustikkoppler, DATEX-P, Mailboxen, Datenbanken). Als Clou gibt es neben einem umfanagreichen Verzeichnis von Telefon- bzw. DATEX-P-Nummern und Anschriften der Mailboxen und Datenbanken fix und fertige Mailbox- und Datenübertragungsprogramme, mit denen Sie z. B. Ihren C 64 in eine Mailbox umwandeln können.

Diese umfassende Einführung in die Textverarbeitung, speziell zugeschnitten auf das weitverbreitete Textverarbeitssystem TEXTOMAT für den C 64, behandelt grundsätzliche Probleme ebenso wie konkrete Anwendungsmöglichkeiten. Aus dem Inhalt: Vorteile einer Textverarbeitung - Laden und Installieren von TEXTOMAT - Druckeranpassungen für viele Drucker - Erstellen von Listen Tabellen, Formularen, Statistiken etc. - Kombination mit DATAMAT, Serienbriefe.

Froitzheim
Das Trainingsbuch zu TEXTOMAT
200 Seiten, DM 39,-
ISBN 3-89011-031-2

Schmidt
Das Trainingsbuch zu DATAMAT
317 Seiten, DM 39,-
ISBN 3-89011-035-5

Sicheres Beherrschen und sinnvoller Einsatz des Dateiverwaltungsprogramms DATAMAT ist Ziel dieses Trainingsbuches. So werden Themen wie z. B. Maske herstellen, Datei einrichten und pflegen, suchen, ändern, löschen, Etikettendruck und Datenübergabe an andere Programme genau beschrieben; im zweiten Teil des Buches sind darüber hinaus praktische Beispiele angeführt, zu denen Literatur- und Schallplattendatei ebenso gehören wie Firmenadressen oder Lagerverwaltung.

DATA BECKER präsentiert ein neues Textverarbeitungsprogramm der Superlative:

TEXTOMAT PLUS

Das alles kann TEXTOMAT:

Diskettenprogramm - durchgehend menuegesteuert - deutscher Zeichensatz auch auf COMMODORE-Druckern - Rechenfunktionen für alle Grundrechenarten - 24.000 Zeichen pro Text im Speicher - beliebig lange Texte durch Verknüpfung - wahlweise 40 oder 80 Zeilen pro Zeile durch horizontales Scrolling des Bildschirms - läuft mit 1 oder 2 Floppies - frei programmierbare Steuerzeichen - Formulareinstellung für Randeinstellung usw. - komplette Bausteinverarbeitung - Blockoperationen, Suchen und Ersetzen - Serienbriefe mit DATAMAT - formatierte Ausgabe auf Bildschirm - an fast jeden Drucker anpaßbar - ausführliches deutsches Handbuch mit Übungslektionen.

TEXTOMAT DM 99,-*

Und das kann TEXTOMAT PLUS zusätzlich:

+ Anzahl der Zeichen pro Zeile frei zwischen 40 und 240 einstellbar - neues Formatieren des Textes bei jedem Einlesen in den Speicher, so daß es keine Rolle spielt, mit welcher Einstellung der Text geschrieben wurde.
+ 8 frei definierbare Floskeltasten zum Schreiben von Wörtern oder Sätzen auf Tastendruck.
+ Wordwrap zieht jedes Wort, das nicht mehr in eine Zeile paßt, sofort in die nächste Zeile.
+ Frei einstellbarer Tabulator

+ Alle einmal definierten Tabulatorpositionen und Floskeltasten, die Formateinstellungen usw. können natürlich im Formular auf Diskette gespeichert und beliebig oft aufgerufen werden.
+ Von Ihnen eingegebene Trennvorschläge werden bei der Formatierung automatisch ausgeführt, so daß lange Wörter nicht mehr große Löcher im Text verursachen.

+ Formatierte Ausgabe auf Bildschirm mit der Anzeige von Überschriften, Seitenumbruch, Seitennummern usw. ermöglichen es, sich ein genaues Bild vom Aussehen des Textes zu machen, ohne auch nur ein Blatt Papier zu verschwenden.
+ Anzeige wahlweise im 40-Zeichenmodus oder über die integrierte softwaremäßige 80-Zeichenkarte möglich.
+ Senden und Empfangen von Texten über Akustikkoppler - dabei können auch Texte von anderen Quellen außer TEXTOMAT PLUS empfangen werden. Eine frei editierbare Konvertierungstabelle verhindert Schwierigkeiten mit den ASCII-Codes anderer Computer.
+ Beliebiger Zeichensatz sowohl für Drucker als auch für Bildschirm erstellbar. Sei es griechisch oder seien es nur ein paar mathematische Sonderzeichen - jedes Zeichen auf dem Bildschirm kann in einer maximalen Matrix von 16x16 Punkten auf den COMMODORE-Druckern MPS 801, 802, 803 und den EPSON-Druckern RX80 bzw. FX80 mit DATA BECKER-Interface ausgedruckt werden. Durch den Ausdruck im Grafikmodus ist es jetzt auch möglich, Proportionalschrift auf allen diesen Druckern (auch den COMMODORE-Druckern!) zu erstellen.
+ Unterstützung des frei definierbaren Zeichensatzes des EPSON-FX 80 in allen Belangen.
+ Mischen von Text und Grafik mit den oben genannten Druckertypen. Jede normal gespeicherte Grafik wie z.B. von SUPERGRAPHIK, KALKUMAT oder KOALA-PAD kann auch ausschnittweise in den Text integriert werden.
+ Druckausgabe auch auf Floppy, so daß der Text in eine Datei geschrieben wird. Damit ist es z.B. möglich, eine Fotosatzmaschine anzusteuern.
+ Wahlweise menuegesteuerte Bedienung des Programms oder schnelle Direktanwahl der Befehle über Buchstaben für den geübten Anwender.
+ Sehr umfangreiches, reich illustriertes Handbuch, in dem alle Funktionen ausführlich beschrieben sind.

TEXTOMAT PLUS DM 248,-* * unverbindliche Preisempfehlung. Alle Programme auf Diskette für VC-1541.

KALKUMAT
DM 198,-*

Das Software-Paket KALKUMAT setzt neue Standards für Kalkulations-Programme auf dem C64. Bewährte Leistungsmerkmale solcher Programme wurden erhalten, neue hinzugefügt und das Ganze mit einer ungewöhnlich komfortablen Bedienerführung versehen.
KALKUMAT bietet Ihnen dank des integrierten Graphik-Pakets Kalkugraph optimales Arbeiten mit Zahlen, übersichtlichen Tabellen, Graphiken und Statistiken.

KALKUMAT in Stichworten:
Tabellen bis zu 255 Zeilen in 63 Spalten erstellbar - jede Spaltenbreite frei wählbar - bequeme Eingabe aller Werte über Edit-Zeile (wie in BASIC-Programmen) - alle Optionen über Menues erreichbar - automatisches Zentrieren von Spaltenüberschriften - Titelfeldfunktion zum komfortablen Kommentieren - Fenstertechnik - Replizieren und Kopieren von Formeln - Einfügen und Löschen von Zeilen und Spalten - in Zweifelsfällen einer von mehr als 90 Hilfsbildschirmen aufrufbar - alle Texte in deutscher Sprache - wahlweise amerikanischer oder deutscher Zeichensatz (mit Umlauten) - vielfältige graphische Darstellungsmöglichkeiten von Werten: Kuchengraphik, Kurvenzüge, Minimum-Maximum-Graphik, Säulendiagramme - Kuchen- und Balkengraphik auch dreidimensional möglich - Beschriftung der Graphiken in verschiedenen Größen horizontal und vertikal möglich - 8 verschiedene Graphiken können in Fenstern gleichzeitig dargestellt werden - alle Graphiken als Hardcopy auf den COMMODORE-Druckern 1525, 801, 802, 803 und EPSON-Druckern mit DATA BECKER-Interface - Handbuch mit ausführlichem Anwender- und Übungsteil.

FAKTUMAT
DM 148,-*

Wirkliche Arbeits- und Zeitersparnis beim Schreiben von Rechnungen bringt FAKTUMAT.
FAKTUMAT bietet Ihnen:
Sofortfakturierung mit integrierter Lagerbuchführung - individuelle Anpassung von Steuersätzen, Maßeinheiten und Firmendaten - Kunden- und Artikelstamm voll pflegbar - schneller Zugriff auf Kunden - und Artikeldaten über freidefinierbaren, 6-stelligen Schlüssel - automatische Fortschreibung von Artikel- und Kundendaten, individuell nutzbar.

FAKTUMAT in Stichworten:
voll menuegesteuert - läuft mit einer oder zwei Floppies - Diskettenwechsel (eine Floppy) nur beim Wechsel von Hauptmenue und Unterprogramm - arbeitet mit 1525, 1526, MPS 801, 802, 803 und EPSON Drucker mit DATA BECKER-Interface - voll parameterisiert: Firmenkopf, Mehrwertsteuer und Rabattsätze, Größe der Dateien beliebig wählbar - 5 Zeilen für Firmenkopf je 30 Zeichen (erste Zeile erscheint auf der Rechnung in Breitschrift) - 4 Mehrwertsteuersätze; während der Rechnungsschreibung können also Artikel mit unterschiedlichem Mehrwertsteuersatz verrechnet werden - 10 Rabattsätze (Rabattsatz 1 vorbelegt mit 0%); bei der Rechnungsschreibung kann jedem Artikel ein Rabattsatz zugewiesen werden - maximal 1900 Artikel bei 50 Kunden oder 950 Kunden bei 100 Artikeln, max. Artikel = (1000 - Kunden)*2; max. Kunden = (2000 - Artikel)/2 - manuelle Eingabe von Artikeln und/oder Kunden während der Rechnungsschreibung; d.h. es können mehr Artikel verrechnet werden als überhaupt in die Datei passen (bei Verzicht auf Lagerbuchführung) bzw. es können Rechnungen an Kunden geschrieben werden, die nicht erfaßt wurden - integrierte Lagerbuchführung mit Ausgabe einer Inventurliste - Rechnungsbeträge und Datum werden in der Kundendatei festgehalten - Druck von: Rechnung (mit Abbuchen aus Lager), Rechnung (ohne Abbuchen aus Lager), Lieferschein - detailliertes deutsches Handbuch mit Übungs- und Anwendungsteil - deutsche Bedienerführung innerhalb des Programms.

* unverbindliche Preisempfehlung. Alle Programme auf Diskette für VC-1541.

DAS STEHT DRIN:

Das BASIC-TRAININGSBUCH zum Commodore 64 ist eine ausführliche, didaktisch gut geschriebene Einführung in das CBM BASIC V2. Von den BASIC-Befehlen über die Problemanalyse bis zum fertigen Algorithmus lernt man schnell und sicher das Programmieren. Übungsaufgaben helfen, das Gelernte zu vertiefen. Gleichzeitig erhält der BASIC-Programmierer ein praxisbezogenes Nachschlagewerk.

Aus dem Inhalt:

- Grundlagen des Programmierens
- Datenfluß- und Programmablaufpläne
- Das Dualsystem
- Bit & Byte
- ASCII-Codes
- Programmablaufpläne
- Komplexere BASIC-Programme
- Unterprogramme und Menuetechniken
- Cursorpositionierung
- Mehrdimensionale Felder
- Ausführliche Liste der BASIC-Befehle mit vielen Beispielen
- Sortierverfahren

UND GESCHRIEBEN HAT DIESES BUCH:

Frank Kampow (27) ist Programmierer in der DATA BECKER Softwareabteilung. Seine vielfältigen Erfahrungen als Seminarleiter zu verschiedensten EDV-Themen einerseits, die jahrelange Programmierpraxis andererseits, machen dieses Buch zu einer nützlichen Hilfe für jeden 64er Anwender.

ISBN 3-89011-023-1