KAMPOW

DAS

BASIC-TRAININGSBUCH

ZUM

CPC 464

EIN DATA BECKER BUCH

ISBN 3-89011-038-X

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, technische Angaben und Programme in diesem Buch wurden von den 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 sind die Autoren jederzeit dankbar.

INHALTSVERZEICHNIS

3. KAPITEL

4. KAPITEL

5. KAPITEL

ANHANG

Dieses Buch ist, wie der Titel schon angibt, ein Trainingsbuch. Es wendet sich an all diejenigen, die sich - vor kurzem vielleicht - einen Schneider CPC 464 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. Daher sollte auch Ihr CPC 464 beim Arbeiten mit diesem Buch vor Ihnen stehen, damit Sie die Beispiele direkt eingeben können.

Der 1.Teil dieses Buches 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.

Der 5.Teil gibt dann schließlich noch eine kurze Einweisung in die Musik- und Grafikbefehle des CPC.

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 CPC 464 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, so daß das BASIC des CPC 464 nicht direkt

auf andere Microcomputer 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 ENTER-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 $V=4PIr^3/3$, 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*PI*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.

Programmierschablone

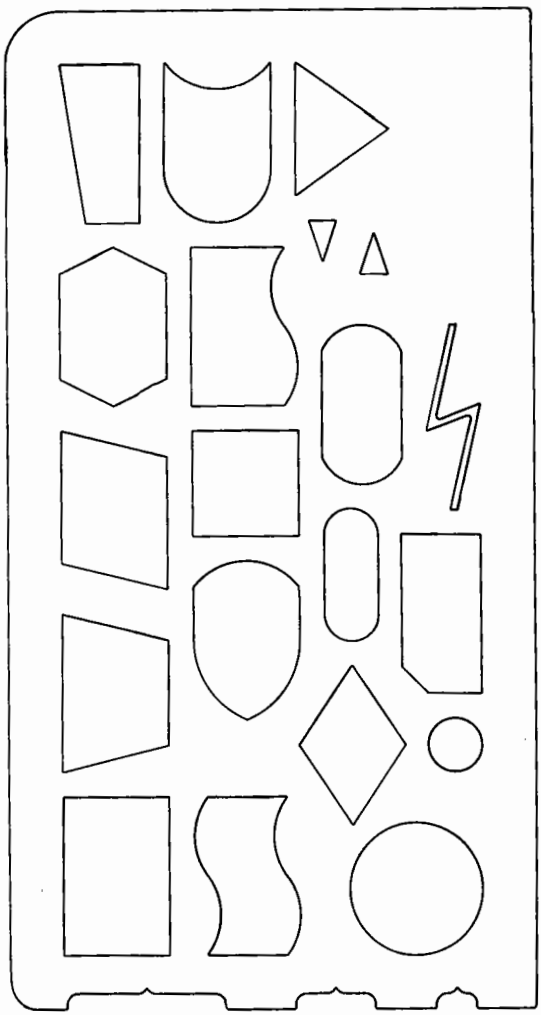


Bild 1

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.

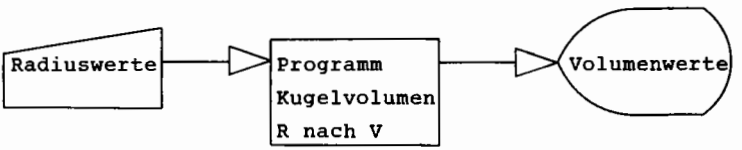


Bild 2

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.

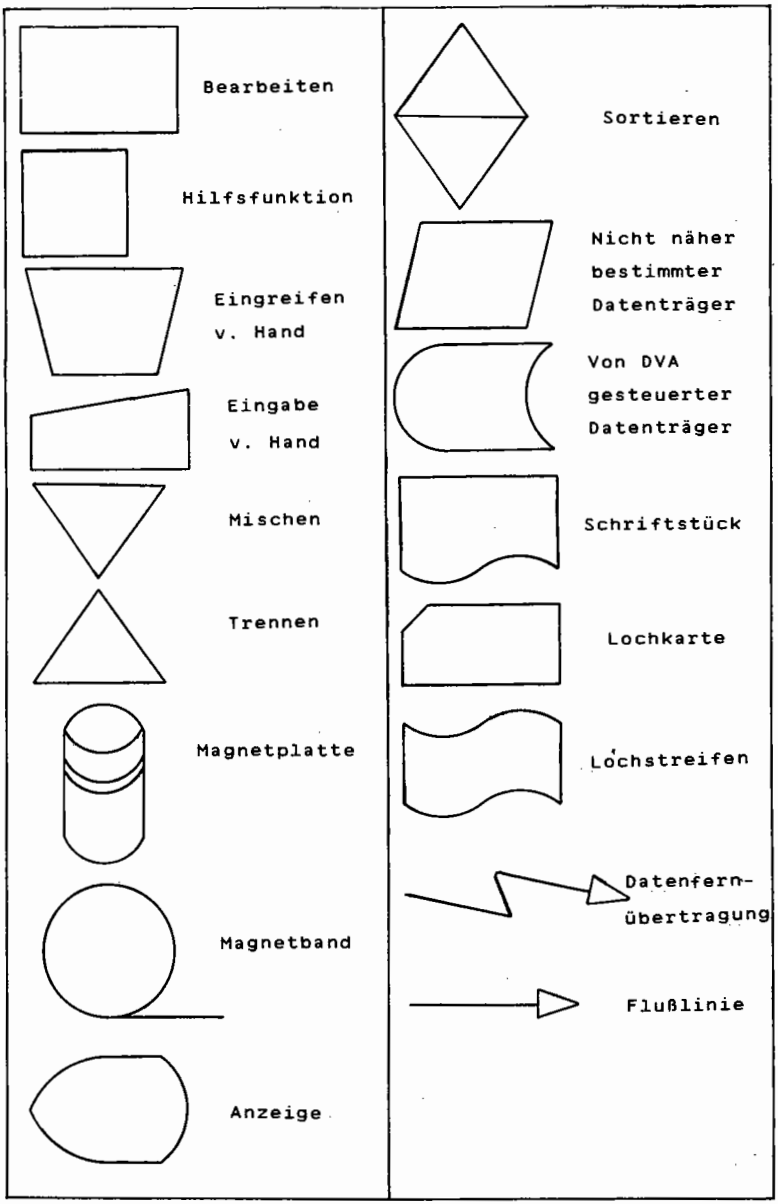


Bild 3

Bevor Sie nun die nächsten Abschnitte lesen, versuchen Sie 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.

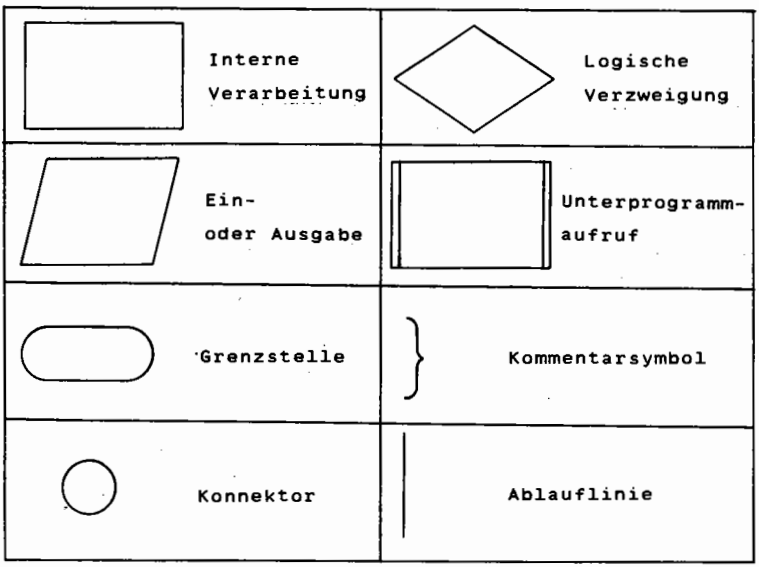


Bild 4

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.

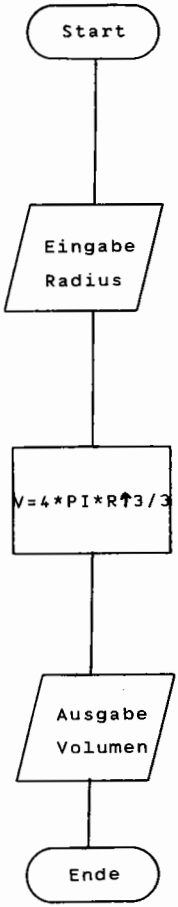


Bild 5

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 weiterführende Fachliteratur verwiesen.

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

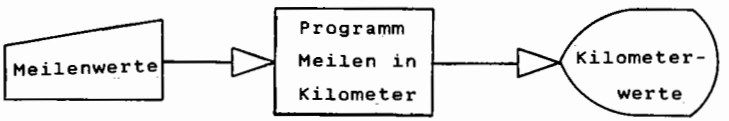
1. Definition des Problems (Erarbeiten der Problemstellung, Problemanalyse)

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

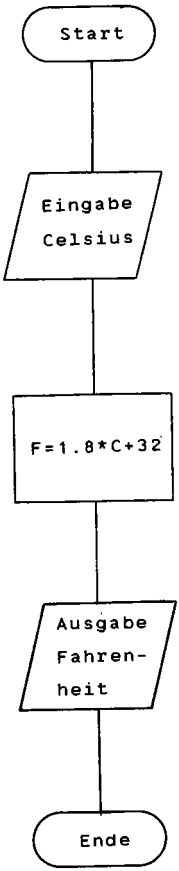


Bild 7

1.3 ASCII-CODES

Der CPC 464 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 0-31 für bestimmte Steuerzeichen, die Zahlen 32-90 für Großbuchstaben und die Zahlen 91-127 für Kleinbuchstaben und einige andere Zeichen verwendet. Der CPC 464 ASCII-Code entspricht dem Standard-ASCII-Code. Die Zahlenwerte von 128-255 stellen bestimmte Grafikzeichen des CPC 464 dar. Genaueres entnehmen Sie bitte den Tabellen im Handbuch zum CPC 464.

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 (BInary DigiT). 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 CPC 464 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 CPC 464 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 Basic-Befehle des CPC 464 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 CPC 464 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:

$S = 4PIr^2$

2. Entwurf des Algorithmus zur Lösung

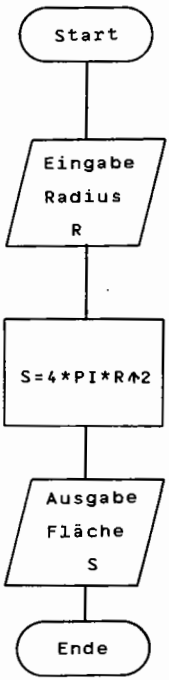
1. Start
2. Eingabe von r
3. Berechnung von $S = 4PIr^2$
4. Ausgabe von S auf Bildschirm
5. Ende

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*PI*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 Kommas 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 ENTER-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 ENTER 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.

Wird allerdings ein Komma zwischen dem Fragetext und der ersten Variablen benutzt, so wird bei der Ausgabe das Fragezeichen unterdrückt.

Wichtig:
Bei INPUT werden mehrere Variablen durch KOMMAS getrennt. Zur Trennung der Dezimal-stellen wird der DEZIMALPUNKT verwendet.

Eine weitere Form des INPUT-Befehls ist der Befehl

LINE INPUT.

Er unterscheidet sich vom INPUT-Befehl eigentlich nur darin, daß man der folgenden Variablen auch das Komma übergeben kann. Weiterhin kann dem LINE INPUT nur eine Variable zugeordnet werden, wogegen dem INPUT-Befehl eine Variablenliste folgen kann. Man könnte im obigen Beispiel ebenfalls schreiben:

```
LINE INPUT"Welcher Radius";S
```

Wichtig:
Dem LINE INPUT-Befehl kann nur eine Variable zugeordnet werden. Einer Stringvariablen kann das KOMMA mit übergeben werden. (s.S.49)

2.1.2 WERTZUWEISUNG MIT LET

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?

2.1.3 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 Schneider CPC 464-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 Grafiksymbole 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 möglichen 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 CPC 464 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 ENTER-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 ENTER, so wird diese Zeile im BASIC-Speicher des Rechners abgelegt. Durch RUN und ENTER 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 "Klingel".(s.S. 71f)

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 CPC 464 stellt im MODE 1 40-Zeichen in einer Bildschirmzeile dar. Eine Programmzeile kann maximal 255-Zeichen umfassen, also ca. 6 1/4 Bildschirmzeilen im MODE 1. Benutzen Sie allerdings z.B. das Fragezeichen '?' als Abkürzung für den PRINT-Befehl in einer Programmzeile und nutzen Sie in dieser Programmzeile die 255 Zeichen voll aus, so wird beim Listen dieser Programmzeile der PRINT-Befehl ausgeschrieben. Für den Platz, der hierfür jetzt mehr benötigt wird, werden überzählige Zeichen am Ende der Programmzeile entsprechend abgeschnitten. Das ist eine Besonderheit, die zu beachten ist. Außerdem denken Sie daran: Vermeiden Sie die Verwendung von MULTISTATEMENTS (Vollstopfen der Programmzeilen mit BASIC-Befehlen).
Jede Bildschirmzeile des CPC 464 ist nach dem Einschalten in Zonen zu je 13 Zeichen 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 14. 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 ENTER-Taste, so sehen Sie auf dem Bildschirm genau die Positionen der einzelnen Tabulatoren in den Bildschirmzeilen. 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.

Besprechen wir nun zusätzlich noch den PRINT USING-Befehl, da er sehr eng mit dem PRINT-Befehl verbunden ist.

2.1.3.1 PRINT USING

Der PRINT USING-Befehl stellt eine modifizierte Form des PRINT-Befehls dar. Diesen Befehl werden Sie hauptsächlich schätzen lernen, wenn Sie Werte ´formatiert´, d.h. in einer bestimmten Form, ausgeben lassen wollen. Dafür stehen Ihnen im SCHNEIDER-Basic folgende Zeichen zur Verfügung:

Für numerische Ausgaben:

#	bestimmt Anzahl der Ausgabestellen.
.	bestimmt Position des Dezimalpunkts.
+	wird bei positiven Zahlen als Vorzeichen mit ausgegeben.
-	wird bei negativen Zahlen am Ende ausgegeben.
**	statt mit Leerzeichen, wird mit * aufgefüllt.
$$	als erstes Zeichen wird $ ausgegeben.
**$	kombinierter Einsatz von $ und *.
,	jede dritte Stelle vor dem Dezimalpunkt wird durch Komma abgetrennt.
↑↑↑↑	Werte werden in Exponentialschreibweise ausgegeben.

Für Textausgaben:

!	es wird nur das erste Zeichen des Textes bzw. der Stringvariablen ausgegeben.
\ \	es werden so viele Zeichen des Textes ausgegeben, wie Leerzeichen plus der beiden Schrägstriche angegeben werden.
&	es wird der komplette Text ausgegeben.

Lassen Sie uns nun den Gebrauch des PRINT USING-Befehls anhand einiger Beispiele üben. Geben Sie zuerst folgendes in den Rechner ein:

```
a  = 12345.678            (ENTER-Taste betätigen)
b  = 34.3455              (ENTER)
c  = -128                 (ENTER)
d$ = "Schneider CPC 464"  (ENTER)
```

Geben Sie nun die folgenden PRINT USING-Befehlsfolgen in den Rechner ein und betätigen Sie jedesmal danach die ENTER-Taste.

```
print using "#####.##";a
```

Als Ausgabe erhalten Sie den auf 2 Nachkommastellen gerundeten Wert:

```
 12345.68
```

```
print using "#####.##";b
```

Ausgabe:

```
    34.35
```

Beachten Sie, daß der Dezimalpunkt bei beiden Werten genau übereinanderliegt. Dies wäre bei der Anwendung des PRINT-Befehls nicht ohne weiteres möglich gewesen. Wird das angegebene Format vom Wert der Zahl überschritten, so wird vor der Zahl ein Prozentzeichen ausgegeben. Das angegebene Format kann hierbei nicht berücksichtigt werden.

```
print using "####.##";a
```

Ausgabe:

```
 %12345.68
```

Wollen Sie positive bzw. negative Ergebnisse besonders hervorheben, so dienen dazu die folgenden Kombinationen:

```
 print using "+#####.##";a
```

Ausgabe:

```
 +12345.68
```

```
 print using "#####.##+";a
```

Ausgabe:

```
 12345.68+
```

```
 print using "#####.##-";c
```

Ausgabe:

```
   128.00-
```

Hier wird die negative Kennzeichnung hinter der Zahl ausgegeben. Normalerweise wird sie der Zahl vorangestellt.

Das nächste Beispiel zeigt Ihnen, wie Sie die Ausgabe der Zahlen im angegebenen Format mit ´Sternchen´ auffüllen können. Diese Form kennen Sie vielleicht von Überweisungsträgern her.

```
print using "**######.##";a
```

Ausgabe:

```
**12345.68
```

```
print using "**######.##";b
```

Ausgabe:

```
*****34.35
```

Die nächste Form der Ausgabe setzt der Zahl das Dollarzeichen ´$´ voran.

```
print using "$$######.##";a
```

Ausgabe:

```
 $12345.68
```

Selbstverständlich können Sie das Dollarzeichen '$' und die 'Sternchen' miteinander kombinieren, wie es das folgende Beispiel zeigt.

```
print using "**$#####.##";b
```

Ausgabe:

```
*****$34.35
```

Außerdem können Sie bei PRINT USING jede dritte Stelle vor dem Dezimalpunkt durch ein Komma optisch hervorheben. Zusätzliche Bezeichnungen wie 'DM' sind ebenfalls möglich.

```
print using "**###,###.## DM";a
```

Ausgabe:

```
**12,345.68 DM
```

Wollen Sie die Zahlenwerte in der Exponentialschreibweise ausgeben lassen, so dient dazu die folgende Form von PRINT USING.

```
print using "##.##↑↑↑↑";a
```

Ausgabe:

```
 1.23E+04
```

Damit hätten wir die Erläuterungen zur numerischen Ausgabeform von PRINT USING abgeschlossen. Schauen wir uns jetzt noch die Möglichkeiten für die verschiedenen Textausgaben an.

```
print using "!";d$
```

Ausgabe:

```
S
```

Bei Benutzung des Ausrufezeichens wird also nur das erste Zeichen der Textvariablen ´d$´ ausgegeben. Wir erinnern uns, daß wir ´d$´ die Zeichenkette "Schneider CPC 464" übergeben hatten.
Die nächste Form erlaubt es uns, eine beliebige Anzahl von Zeichen einer Zeichenkette ausgeben zu lassen.

```
print using "\       \";d$
```

Ausgabe:

```
 Schneide
```

Oben wurden zwischen den Anführungszeichen insgesamt 8 Zeichen eingegeben (einschließlich der beiden Schrägstriche). Somit werden auch von der Textvariablen

´d$´ nur die ersten 8 Zeichen ausgegeben. Diese Anwendung von PRINT USING ist verwandt mit dem LEFT$-Befehl, der zu einem späteren Zeitpunkt besprochen wird.(s.S. 82 f.)
Zum Schluß wollen wir noch die Möglichkeit besprechen, wie man sich eine Textvariable komplett ausgeben lassen kann.

```
print using "&";d$
```

Ausgabe:

```
Schneider CPC 464
```

Dies könnte man auch durch

PRINT d$

erreichen. Es kann jedoch durchaus vorkommen, daß man von dieser Möglichkeit innerhalb des PRINT USING-Befehls einmal Gebrauch machen muß.

Der Vollständigkeit halber sei hier noch der Befehl

WRITE

erwähnt. Es handelt sich hierbei ebenfalls um einen nahen Verwandten des PRINT-Befehls. Er unterscheidet sich jedoch etwas in der Ausgabe vom PRINT-Befehl. Geben Sie z.B. die folgende Befehlsfolge in den Rechner

WRITE "Schneider"

und betätigen die ENTER-Taste, so wird

"Schneider"

auf dem Bildschirm ausgegeben. Die Anführungszeichen werden also im Gegensatz zum PRINT-Befehl mit angezeigt.

Folgen Zahlen oder Variablen durch Kommas getrennt dem WRITE-Befehl, so hat das Komma keine Tabulatorfunktion, sondern wird mit angezeigt.

```
WRITE 2,3,4
```

Ausgabe:

```
2,3,4
```

Damit hätten wir die Befehle, die in unserem Beispielprogramm vorkamen, sowie zwei nähere ´Verwandte´ von ihnen, besprochen. Eigentlich sollten bei der Benutzung dieser Befehle keine größeren Schwierigkeiten mehr auftreten.
Um Programme nun auch für andere besser verständlich zu machen, schauen wir uns noch die REM-Anweisung an.

2.1.4 KOMMENTARE MIT REM

Mit REM können Sie an beliebiger Stelle im Programm eine Bemerkung (engl. REMark) 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. Im Anschluß sehen Sie 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)";RADIUS
40 REM BERECHNUNG OBERFLAECHE
50 LET OFL=4*PI*RADIUS↑2
60 REM AUSGABE OBERFLAECHE IN CM↑2
70 PRINT"DIE KUGELOBERFLAECHE BETRAEGT ";OFL;" 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 cm2 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 SCHNEIDER-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 (REAL-VARIABLE) wird benutzt, um Dezimalzahlen (auch Fließkommazahlen genannt) darstellen zu können. Die dabei verwendeten Variablenbezeichnungen benötigen normalerweise keinen Zusatz, um diesen Typ zu kennzeichnen. Bei Bedarf ist der Zusatz für diesen Variablentyp das Ausrufezeichen '!'. Erlaubte Bezeichnungen sind z.B. A oder B2. Nach dem Einschalten des Rechners werden übrigens alle numerischen Variablen zuerst als REAL-Variablen betrachtet.

Der dritte Variablentyp wird zusätzlich durch das "$" Dollarzeichen gekennzeichnet. Es handelt sich hierbei um die sogenannte STRINGVARIABLE oder auch TEXTVARIABLE. 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 CPC 464 läßt Variablenbezeichnungen bis zu 40 Zeichen zu. Sie dürfen also durchaus einer Variablen die Bezeichnung ´DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT´ geben. Viele Microcomputer können solche langen Variablennamen nicht unterscheiden. Durch diese Eigenschaft des SCHNEIDER-BASIC werden Programme leichter verständlich und lesbar, da den Variablen sinnvolle Bezeichnungen gegeben werden können wie z.B. ´Zahlung´, ´Wechsel´ oder ´Name´. Auch dürfen sich in den Variablen Basic-Schlüsselwörter befinden. Die Bezeichnung WAND wäre zulässig, obwohl 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, BETRAG9 oder ähnliche. Es ist nicht erlaubt, die Zahl an die erste Stelle zu setzen. Z.B. ist ´9BETRAG´ NICHT ZULÄSSIG!

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

OR, FN oder ABS.

Eine Liste der reservierten Wörter finden Sie im Anhang dieses Buches sowie im Handbuch des CPC 464. 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 einen 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) Datum$ c) TAG
   d) ODER% e) IF f) TIME$
   g) 4Zahl% h) 255 i) MONTAG

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) TAG unzulässig, da SCHNEIDER-BASIC Befehl.
   d) zulässig
   e) IF unzulässig, siehe c).
   f) TIME$ unzulässig, da SCHNEIDER-BASIC Variable.
   g) 4Zahl% unzulässig, da Bezeichnung mit einer Ziffer beginnt.
   h) 255 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 ENTER eingegeben, so erscheint auf dem Bildschirm die Meldung ´Redo from start´. 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";Hoehe,C
40 A=.5*Hoehe*C
50 PRINT"DIE FLAECHE HAT ";A;"M↑2"
60 END
```

4.
```
10 INPUT"EINGABE KOERPERGROESSE IN CM";CM
20 REM BERECHNUNG IDEALGEWICHT
30 IDG=CM-100
40 REM BERECHNUNG 10 PROZENT
50 PR=IDG/100*10
60 IDG=IDG-PR
70 PRINT"IHR IDEALGEWICHT IST";IDG;"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 IDG=(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";Hoe,Lae,Tie
20 REM BERECHNUNG VOLUMEN
30 Vol=Hoe*Lae*Tie
40 REM BERECHNUNG LITER
50 Vol=Vol/1000
60 PRINT"AQUARIUM INHALT";Vol;"LITER"
70 END
```

In diesem Programm werden zunächst den Variablen für die Höhe, Länge und Tiefe, Hoe, Lae und Tie, 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 den 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 CPC 464 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 CPC 464 findet man die Angabe, daß die Werte der trigonometrischen Funktionen wie SIN(X), COS(X) oder TAN(X) standardmäßig 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, so daß 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 CLS:DEG
20 INPUT"EINGABE IN GRAD";GR
30 REM BERECHNUNG DES SINUS
40 SI=SIN(GR)
50 PRINT"DER SINUS VON";GR;"GRAD ";
60 PRINT"IST =";SI
70 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 Umschaltung in Zeile 10 'DEG' auf das Winkelgradmaß benutzen. In Zeile 10 wurde noch ein zusätzlicher Befehl 'CLS' (engl. CLear Screen) benutzt. Dieser Befehl löscht den gesamten Bildschirm. Sie sollten sich angewöhnen, diesen Befehl jedem Ihrer Programme voranzustellen, damit spätere Ausgaben auf dem Bildschirm nicht durch alte Mitteilungen gestört werden. Für die Berechnung des Cosinus' müßten Sie in Zeile 40 nur SIN durch COS ersetzen. Die Variable SI können Sie beibehalten. 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 ENTER-Taste.

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

Das einzige, was sich gegenüber unserem vorigen Programm geändert hat, ist die Zeile 10. Eigentlich ist die Anweisung 'RAD' in Zeile 10 unnötig, da nach Eingabe von 'NEW' der CPC 464 automatisch wieder auf das Bogenmaß umschaltet. Sie sei hier trotzdem zur Übung mit angegeben. Starten Sie nun das Programm und geben Sie den Wert 1.57079633 (entspricht PI/2 oder dem Viertelkreis) ein. Als Ergebnis erhalten Sie wieder den Wert Null.

Die Anwendung der anderen Funktionen ist genauso einfach. Diesen numerischen Funktionen wird 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 ENTER:

```
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 LOG10(X) ersetzen. 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 CLS
20 INPUT"EINGABE DER ZAHL";Z
30 REM BERECHNUNG NAT. LOGARITHMUS
40 LN=LOG(Z)
50 REM BERECHNUNG DEKA. LOGARITHMUS
60 LO=LOG10(Z)
70 PRINT"DER NATUERLICHE LOGARITHMUS ";
80 PRINT"VON";Z;" BETRAEGT";LN
90 PRINT
100 PRINT"DER DEKADISCHE LOGARITHMUS ";
110 PRINT"VON";Z;" BETRAEGT";LO
120 END
```

Sie sehen, es ist also relativ einfach, diese Funktionen in Programmen zu handhaben. Die einzige Schwierigkeit besteht eben in der unterschiedlichen Handhabung zwischen Bogenmaß und Winkelgradmaß.

WICHTIG:
Die trigonometrischen Funktionen erwarten die Werte im Bogenmaß. Für die Berechnung in Grad muß erst mit 'DEG' in das Winkelgradmaß umgeschaltet 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 kann man auch zum Runden von Zahlen benutzen. Mit einer entsprechenden Routine kann man auf beliebig viele Stellen nach dem Komma runden. Das nachfolgende kleine Programm soll Ihnen das verdeutlichen.

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

Das Programm ist recht einfach, trotzdem will ich Ihnen die wichtigsten Zeilen erklären. In Zeile 20 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 30 wird nach einer beliebigen Zahl gefragt. Geben Sie hier irgendeine Dezimalzahl ein, deren Stellenzahl größer als die zu rundende Stellenzahl ist.

In Zeile 50 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 ENTER-Taste. Geben Sie dann einige Werte ein, um zu sehen, welche Ergebnisse Sie erhalten. Versuchen Sie vor allem die Zeile 50 zu verstehen, in der die Rundung der Zahl vorgenommen wurde. Dadurch können Sie sich dann besser vorstellen, was Ihr CPC 464 alles leisten muß, wenn Sie den ROUND-Befehl anwenden.

Mit ´ROUND´ können Sie auf komfortable Weise Zahlen bzw. Variablenwerte runden. Die Schreibweise des Befehls sieht wie folgt aus:

```
ROUND (X,Y)
```

Dabei wird in ´X´ die zu rundende Zahl übergeben und in ´Y´ die Anzahl der Stellen, auf die gerundet werden soll.

```
PRINT ROUND(3.12354,3)
```

Ausgabe:

```
 3.124
```

So einfach können Sie im SCHNEIDER-BASIC Zahlen runden. Das obige Programmbeispiel sollten Sie dennoch nicht

vergessen. Es war ein gutes Beispiel dafür, wie man sich nützliche Routinen selber entwickeln kann. Denn nicht alle Computer besitzen ein so leistungsfähiges Basic wie der CPC 464.

Ein weiterer Befehl zum Runden von Zahlen sei hier noch kurz erwähnt. Der Befehl

CINT

rundet eine Zahl bzw. Variable und wandelt sie gleichzeitig in das Integerformat um. Daher liegt der Zahlenbereich für diesen Befehl auch im Intervall von -32768 bis +32767.

Der Befehl

CREAL

bewirkt allerdings keine Rundung, sondern nur, daß eine Variable in das Realformat, also mit Dezimalpunkt, umgewandelt wird.

Mit der Funktion MOD (modulo) können Sie den Restwert einer Division berechnen lassen. Geben Sie

PRINT 32 MOD 7

ein, so erhalten Sie als Ausgabe den Wert 4.

2.3.1 FUNKTIONEN MIT DEF FN

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 CPC 464 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 ENTER. 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 (150 * 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
```

Wollen Sie dem Zufallszahlengenerator einen neuen Startwert übergeben, so steht Ihnen im Schneider-Basic der Befehl

```
RANDOMIZE
```

zur Verfügung. Die Anwendung ist denkbar einfach, wie das folgende Beispiel zeigt.

```
RANDOMIZE 45
```

Geben Sie diesen Befehl ein, so wurde dem Zufallszahlengenerator der neue Startwert 45 zugeordnet. Jetzt können Sie mit

```
A=RND(4)
```

der Variablen ´A´ eine neue Zufallszahl zuordnen.

2.3.3 WEITERE BEFEHLE FÜR DEN UMGANG MIT VARIABLEN

Das umfangreiche Basic des CPC 464 besitzt eine Menge an Befehlen, mit denen man Variablen bzw. Variablenformate bestimmen, umwandeln oder löschen kann. Diese Befehle sollen im folgenden nun kurz besprochen werden.

Sie erinnern sich bestimmt noch an das Kapitel über die Zahlensysteme. Der CPC 464 bietet dem Benutzer zwei Befehle, die es ihm ermöglichen, dezimale Zahlen entweder

in das duale oder in das hexadezimale Zahlensystem umzuwandeln.

Der Befehl

BIN$(X,Y)

wandelt Dezimalzahlen in Dualzahlen um. Dabei steht 'X' für die Zahl, die umgerechnet werden soll, und 'Y' bestimmt das Format, in der die Zahl ausgegeben werden soll.

PRINT BIN$(24,8)

Ausgabe:

00011000

Der Befehl

HEX$(X)

wandelt Dezimalzahlen in Hexadezimalzahlen um. 'X' steht hier für die umzurechnende Zahl.

PRINT HEX$(60)

Ausgabe:

3C

Wie bereits erwähnt, existieren im Schneider-Basic drei verschiedene Variablentypen (Integer, String und Real). Wollen Sie nun innerhalb eines Programms bestimmten Variablentypen bestimmte Variablenbezeichnungen zukommen lassen, so stehen Ihnen hierfür die folgenden Befehle zur Verfügung:

DEFINT

DEFSTR

DEFREAL

Geben Sie z.B.

DEFINT A-B

in den Rechner, so haben Sie damit alle Variablen, die mit ´A´ oder mit ´B´ beginnen, als INTEGERVARIABLE festgelegt. Sie brauchen dann innerhalb des Programms nicht jedesmal ein Prozentzeichen an den Variablennamen anzuhängen. Analog dazu sind die anderen beiden Befehle zu gebrauchen.
Doch Vorsicht, diese Art der Variablenfestlegung erfordert ein hohes Maß an Übersicht bei der Programmierung. Vergessen Sie z.B., daß Sie die Variablen A bis B als Integervariable definiert haben, und ordnen nur einer Variablen ´A´ innerhalb eines Programms einen String zu, so bricht das Programm nach dem Start mit der Fehlermeldung

Type mismatch in (Zeilennummer)

ab.

Wollen Sie bei der Werteausgabe nur den Zahlenanteil vor dem Dezimalpunkt ausgeben lassen, so können Sie dazu den Befehl

FIX

benutzen. FIX arbeitet ähnlich wie der INT-Befehl, nur mit dem Unterschied, daß FIX grundsätzlich den Vorkommateil abtrennt, ohne eine Rundung vorzunehmen. Während INT im negativen Bereich zur nächst kleineren Zahl rundet, z.B.

```
PRINT INT(-3.55)
```

Ausgabe:

```
-4
```

würde die Ausgabe mit FIX wie folgt aussehen:

```
PRINT FIX(-3.55)
```

Ausgabe:

```
-3
```

Zwei weitere leistungsfähige Befehle dienen zum Feststellen der größten bzw. der kleinsten Zahl innerhalb einer Liste von Variablen. Der Befehl

```
MAX
```

gibt die größte Zahl aus einer Liste von Zahlen aus. Diese Liste kann auch Variablenbezeichnungen beinhalten.

```
PRINT MAX(23,4,65,67,123,344,33)
```

Ausgabe:

```
344
```

Analog dazu findet der Befehl

```
MIN
```

die kleinste Zahl einer Variablenliste.

```
Z=-56

PRINT MIN(3,4,1,Z,-3,-25)
```

Ausgabe:

```
-56
```

Damit hätten wir soweit die wichtigsten Befehle besprochen, die man für den Umgang mit Variablen benötigt.

2.3.4 ASC(X$) UND CHR$(X)

Der CPC 464 besitzt die Möglichkeit, durch den PRINT-Befehl Zahlen, Buchstaben und teilweise Grafikzeichen 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. 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 es nur "abtippen" will, Schwierigkeiten bekommen. Erstens gibt es teilweise Schwierigkeiten bei der Identifizierung der Steuerzeichen und zweitens sind manche Steuerzeichen nicht durch alle Drucker ausdruckbar, was wiederum zu Mißverständnissen beim Lesen solcher Programmlistings führt.

Muß man dann noch auf der Tastatur nach einem bestimmten Grafikzeichen 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 Erzeugen des Pieptons (Klingel):

```
PRINT "Ω"
```

Erreicht wird dieses Klingelzeichen nur durch gleichzeitiges Betätigen der Tasten CTRL und 'G'. Solche Grafikzeichen, die bestimmte Funktionen aufrufen, werden nun gerne an Stelle der weiter unten aufgeführten Schreibweise benutzt. Diese Sonderfunktion läßt sich jedoch auch mit einer anderen Befehlsfolge erzielen, indem wir einen CHR$-Code verwenden:

```
PRINT CHR$(7)
```

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 ENTER-Taste (Pfeile). 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 Cursorsteuerung oder andere Tasten die ASCII-Werte zu benutzen.

In der Tabelle im Anhang dieses Buches oder im Handbuch zum CPC 464 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 ENTER-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 ENTER-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$(X) beibehalten, zumindest bei den speziellen Codes für die Steuerzeichen wie das der ´Klingel´. Die Umsetzung der Programme auf andere Rechner wird durch die Benutzung der CHR$-Codes ebenfalls erleichtert, da meistens die Bedeutung der ASCII-Werte übereinstimmt, z.B. CHR$(7) für die ´Klingel´ (engl. bell). Der CPC 464 benutzt, wie bereits erwähnt, ein spezielles Grafikzeichen, das einer Klingel ähnlich sieht, um einen einfachen Piepston zu erzeugen. Dieses Zeichen wird auf anderen Rechnern schon gänzlich anders dargestellt. Teilweise besitzen einige Rechner für solche Funktionen überhaupt keine speziellen Grafikzeichen. Dagegen können Sie den Befehl PRINT CHR$(7) direkt übertragen, ohne großartig nach einer bestimmten Tastenfunktion oder einem Grafikzeichen 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 CLS
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 / ENTER starten, sehen Sie die Unterschiede in den ausgegebenen Zahlen.
In Zeile 20 wird der Bildschirm 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 ENTER einzugeben, um den ASCII-Wert dieser "Zeichen" zu erfahren. Dabei wurde dann vom Rechner die Fehlermeldung REDO FROM START bzw. IMPROPER ARGUMENT 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 ENTER-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 INKEY-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";Lit
20 INPUT"WIEVIEL KM GEFAHREN";Km
30 Ver=Lit/Km*100
40 PRINT"VERBRAUCH AUF 100 KM";Ver;" 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, SPC UND ZONE

Diese drei 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 an 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 14mal die Taste, die den Cursor nach rechts bewegt. Der Cursor sollte nun genau über dem Buchstaben ´T´ stehen. Schreiben Sie nun eine neue Befehlsfolge und verwenden Sie statt TAB den SPC-Befehl. Nach dem Betätigen der ENTER-Taste erhalten Sie fast das gleiche Ergebnis. Bei dieser Art der Anwendung trat der Unterschied zwischen den beiden Befehlen nicht so deutlich hervor. Doch schon beim nächsten Beispiel werden Sie einen sichtbaren Unterschied bemerken. Löschen Sie zuerst den Bildschirm mit dem Befehl ´CLS´. Geben Sie danach die folgende Befehlsfolge ein:

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

Nachdem Sie die ENTER-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 ENTER-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.

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.

Wie Sie bereits wissen, wird der Bildschirm des CPC 464 nach dem Einschalten in Zonen unterteilt, die jeweils 13 Zeichen breit sind. Wollen Sie nun diesen Wert verändern, so steht Ihnen dazu der Befehl

ZONE

zur Verfügung. Geben Sie z.B.

```
ZONE 10
```

ein und betätigen die ENTER-Taste, so wird der Tabulator auf 10 Zeichen festgesetzt.
Damit Sie den Unterschied zur Standardeinstellung sehen, können Sie jetzt das Beispiel ausprobieren, welches im Kapitel über den PRINT-Befehl aufgezeigt wurde.

2.5 STRINGS

Ein String bezeichnet eine Zeichenkette, die bis zu 255 beliebige Zeichen des CPC 464 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$="Schneider CPC 464"
```

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

```
Type mismatch
```

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 CPC 464 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. Weiterhin können Sie nicht nur Zeichen innerhalb eines Strings auslesen lassen, sondern auch ganz bestimmte Zeichen abändern bzw. neu zuordnen. Schreiben Sie z.B.

```
MID$(A$,7,1)="u"
```

so ändern Sie das siebte Zeichen in ein "u" um. Es heißt dann nicht mehr ´Donaudampf...´ sondern ´Donaudumpf...´. 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 CLEAR (setzt Variable auf Null) eingegeben, so geben Sie jetzt folgendes im Direktmodus ein:

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

und drücken Sie ENTER. 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 (auch Kommas), die nicht in eine Zahl zu übersetzen sind, so wird nur der erste Teil des Strings mit den Ziffern berücksichtigt. Die nachfolgenden Beispiele sollen Ihnen 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(X$) bewirkt, also einen numerischen Ausdruck in einen String verwandelt, ist der STR$(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 a) 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.

Ein weiterer nützlicher Befehl beim Umgang mit Strings, ist der Befehl

INSTR

mit dem Sie einen String nach einer beliebigen Zeichenfolge durchsuchen lassen können. Die Schreibweise des Befehls sieht wie folgt aus:

```
INSTR (X,A$,B$)
```

Dabei steht ´X´ für die Position, ab der der String ´A$´ durchsucht werden soll. ´X´ muß hier immer größer als Null sein. ´B$´ steht für die Zeichenfolge, nach der gesucht werden soll. Als Ergebnis erhalten Sie die Positionsnummer der Zeichenfolge im String. Ist die Zeichenfolge nicht enthalten, so wird das Ergebnis ´Null´. Das nachfolgende kleine Programm soll die Funktion etwas verdeutlichen.

```
10 A$="Donaudampfschiffahrtsgesellschaft"
20 B$="schiff"
30 Z=INSTR(1,A$,B$)
40 PRINT Z
50 END
```

Starten Sie dieses Programm mit RUN, so erhalten Sie als Ergebnis für Z den Wert 11. Berücksichtigen Sie bei der Anwendung dieses Befehls, daß der String nach genau den gleichen Zeichen durchsucht wird. Das bedeutet, daß er die Zeichenfolge "Schiff" nicht gefunden hätte, da diese mit einem großen ´S´ beginnt.

WICHTIG:
Bei Anwedung von INSTR(X,A$,B$) muß ´X´ immer größer Null sein. Ansonsten wird die Fehlermeldung ´IMPROPER ARGUMENT IN (Zeilennummer) ausgegeben. Die Zeichenfolge, nach der gesucht wird, muß auf das Zeichen genau in dem String enthalten sein. Beachten Sie auch Groß- und Kleinbuchstaben.

Zwei weitere Befehle, die ebenfalls den Umgang mit Strings etwas erleichtern, sind die Befehle

UPPER$

und

LOWER$.

Die Wirkungsweise dieser Befehle ist rasch erklärt. Der Befehl UPPER$ wandelt einen String komplett in Großbuchstaben um, und der LOWER$-Befehl wandelt einen String in Kleinbuchstaben um. Ein einfaches Beispiel soll hier zur Verdeutlichung genügen.

```
A$="CPC 464"          (ENTER)
PRINT LOWER$(A$)      (ENTER)
```

Ausgabe:

cpc 464

In einem späteren Kapitel werden wir uns mit der optischen Aufbereitung von Programmen befassen. Ein Befehl, der Sie dabei unterstützt, ist der

STRING$

Befehl. Mit diesem Befehl haben Sie die Möglichkeit, einen String zu erzeugen, der sich aus lauter gleichen Zeichen zusammensetzt. Geben Sie z.B. die folgende Befehlsfolge in den Recher,

```
A$=STRING$(40,"*")    (ENTER)
PRINT A$              (ENTER)
```

so erhalten Sie als Ausgabe eine Zeile mit 40 Sternchen. Damit haben Sie die Möglichkeit, auf einfache Art und Weise Bildschirmmasken zu erstellen. Doch dazu erfahren Sie in einem späteren Kapitel mehr.

Der Befehl

SPACE$

ist in der Ausführung ähnlich dem STRING$-Befehl. Er kann zu verschiedenen Anwendungen herangezogen werden. Mit

```
A$=SPACE$(40)
```

wird der String A$ mit 40 Leerzeichen (Blanks) aufgefüllt. Der Befehl kann allerdings auch direkt zur Positionierung einer Ausgabe benutzt werden, wie das folgende Beispiel zeigt:

```
PRINT SPACE$(10);"CPC 464"
```

Ausgabe:

```
          CPC 464
```

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)"TEST1" TAB(15)"TEST2"

   PRINT TAB(5)"TEST1" TAB(15)"TEST2"
   ```

   a) Einziger Unterschied liegt darin, daß bei der ersten Befehlsfolge der String "TEST1" ein Zeichen weiter rechts ausgegeben wird.

   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.

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.

2.6 EDITIEREN VON PROGRAMMEN

Bevor wir nun dazu übergehen, größere Programme zu entwickeln, wollen wir uns kurz mit den Befehlen bzw. Anweisungen befassen, die uns das Programmieren etwas erleichtern. Der Begriff ´EDITIEREN´ umfaßt eigentlich alles, was mit der Veränderung eines Programms zu tun hat. Sei es nun das Löschen oder Hinzufügen von Programmzeilen oder das Beseitigen von ´Syntax-Fehlern´. Der Umgang mit dem ´Cursor´, und wie man damit Programmzeilen kopiert, wird in diesem Buch als bekannt vorausgesetzt. Sollten Sie trotzdem noch Schwierigkeiten damit haben, so lesen Sie bitte im Handbuch des CPC 464 das entsprechende Kapitel nach. Das Buch "CPC 464 FÜR EINSTEIGER" behandelt u.a. dieses Thema sehr ausführlich und verdeutlicht die Benutzung des ´Cursors´ anhand von Bildern.

Es wurde bereits erwähnt, daß Sie die Zeilennumerierung in ZEHNERSCHRITTEN vornehmen sollen. Ein Befehl, der Sie dabei unterstützt, ist der

AUTO

Befehl. Geben Sie AUTO in den Rechner und betätigen die ENTER-Taste, so erscheinen auf dem Bildschirm die Zahl 10 und der Cursor. Damit erwartet der Rechner Ihre Eingabe für die Programmzeile 10. Befindet sich bereits ein Programm im Speicher, so werden alle Zeilennummern in diesem Programm, die ein Vielfaches von 10 bilden, also 10, 20, 30 usw., mit einem Stern "*" hinter der Zeilennummer gekennzeichnet.

20*

Drücken Sie jetzt die ENTER-Taste, so wird diese Programmzeile unverändert übernommen. Sobald Sie jedoch

in diese Zeile hineinschreiben, geht der alte Inhalt automatisch verloren. Mit der ´ESC´-Taste können Sie die automatische Zeilennumerierung wieder abschalten.

Schreiben Sie ein Programm, ohne die automatische Zeilennumerierung zu benutzen, so benötigen Sie früher oder später den

RENUM

Befehl. Bei der Entwicklung eines Programms werden immer wieder Zeilen eingefügt werden müssen. Damit könnte die Numerierung dann nach kurzer Zeit so aussehen:

10
12
15
19
20
21

Geben Sie jetzt im Direktmodus RENUM ein, so werden alle Programmzeilen in Zehnerschritten neu durchnumeriert, so daß Sie im obigen Beispiel jetzt die Zeilennummern von 10 bis 60 hätten. Die Leistungsfähigkeit dieses Befehls zeichnet sich besonders dadurch aus, daß auch Programmsprünge mit den Befehlen ´GOTO´, ´GOSUB´ usw., mit berücksichtigt werden.
Sie können ebenfalls nur einen bestimmten Abschnitt des Programms neu durchnumerieren, indem Sie

RENUM X,Y,Z

eingeben. Dabei steht ´X´ für die Zeilennummer, mit der die neue Numerierung beginnen soll, und ´Y´ für die Zeilennummer, ab der numeriert werden soll. Mit ´Z´ können Sie noch die Schrittweite bestimmen. Der Befehl

RENUM 200,100,5

würde das Programm ab Zeilennummer 100 in Fünferschritten, beginnend mit der neuen Zeilennummer 200, durchnumerieren.

Es kann vorkommen, daß Sie aus einem Programm nur bestimmte Zeilen löschen wollen. Hierfür geben Sie einfach

DELETE 100-200

in den Rechner, und schon werden alle Zeilennummern von 100 bis 200 gelöscht. Sie können diesen Befehl aber auch variieren, indem Sie eingeben:

DELETE -200

oder

DELETE 200-

Sie können also auch bis zu einer bestimmten Zeilennummer oder ab einer bestimmten Zeilennummer Programmzeilen löschen.

Zwei weitere Befehle stehen Ihnen zum Löschen von Variablen zur Verfügung. Der Befehl

CLEAR

löscht alle Variablen und Arrays (Felder). Sollte Ihnen der Begriff ´Array´ noch nicht geläufig sein, so ist das momentan nicht weiter von Bedeutung. In einem späteren Kapitel wird dieser Begriff ausführlich erklärt.
Sie können also mit ´CLEAR´ ab einer bestimmten Stelle im Programm alle Variablen zurücksetzen. Wollen Sie nur Felder bzw. ein Feld löschen, so können Sie dazu den

ERASE

Befehl verwenden.

Mit

ERASE A

wird das vorher mit DIM A(20) erzeugte Feld gelöscht. Damit wird gleichzeitig neuer Speicherplatz geschaffen.

Wollen Sie ein Programm einem Testlauf unterziehen und es an einer bestimmten Programmzeile unterbrechen lassen, um z.B. zu überprüfen, ob es bis dahin fehlerfrei läuft, so können Sie an dieser Stelle den

STOP

Befehl einsetzen. Trifft das Programm auf diesen Befehl, bricht es mit der Meldung

Break in (Zeilennummer)

ab. Mit dem Befehl

CONT

können Sie das Programm an der Stelle wieder fortfahren lassen, an der es mit dem STOP-Befehl unterbrochen wurde.

Bei der Erstellung Ihrer Programme wird es nicht ausbleiben, daß Ihnen Fehler unterlaufen. Dieses können sowohl Fehler logischer Art als auch syntaktischer Art sein. Haben Sie nun in einer Programmzeile einen Fehler entdeckt und wollen Sie diese Zeile nicht mit der Kopiermethode editieren, so können Sie mit dem Befehl

EDIT (Zeilennummer)

diese Zeile auf einfache Art verändern. Es wird die angegebene Zeile auf dem Bildschirm angezeigt und Sie können sofort mit der Editierung beginnen, als hätten Sie diese Zeile gerade erst geschrieben.

Manchmal ist es ganz nützlich, wenn man den Programmablauf anhand der Zeilennummern verfolgen kann, um z.B. Vergleiche zu seinem PAP anstellen zu können. Nach der Eingabe des Befehls

TRON

(engl.=TRace ON) wird jede Zeilennummer beim Programmablauf vorher in eckigen Klammern ausgegeben. Der Befehl

TROFF

(engl.=TRace OFF) schaltet diese Hilfe wieder aus.

Der Befehl

NEW

löscht das momentan im Speicher befindliche Programm. Sie haben diesen Befehl ja schon bei Ihren Aufgaben eingesetzt. Er wurde hier nur der Vollständigkeit halber noch einmal erwähnt. Es sei hier ebenfalls noch der berühmte DREI-FINGER-GRIFF erwähnt. Gemeint ist hier die Tastenkombination SHIFT, CTRL und ESC. Bei Anwendung dieser Kombination wird der CPC 464 in den Einschaltzustand zurückversetzt. Dabei gehen alle vorher eingegebenen Informationen verloren.

Mit dem Befehl

LIST

können Sie sich das Programm auf dem Bildschirm anzeigen lassen. Diesen Befehl können Sie ebenfalls auf die gleiche Art variieren wie den DELETE-Befehl.

Zwei Befehle, die mit der eigentlichen Editierung von Programmen nichts zu tun haben, mit denen Sie aber den Programmspeicherplatz abfragen bzw. beeinflußen können,

sollen hier noch erwähnt werden.

Geben Sie

PRINT HIMEM

in den Rechner, so erhalten Sie die folgende Ausgabe:

43903

Dieser Befehl gibt Ihnen die höchste vom Basicspeicherplatz belegte Adresse aus. Die oben aufgeführte Zahl repräsentiert den Standardwert. Sie können mit dem Befehl

MEMORY (Adresse)

die Speicherobergrenze herabsetzen (siehe dazu auch Handbuch). Der dadurch nicht belegte Speicherplatz kann z.B. für eigene Maschinenroutinen Platz bieten.

Damit hätten wir das Kapitel über die Einführung in das Programmieren mit Basic abgeschlossen. Im nächsten Kapitel werden wir uns mit den erweiterten Programmstrukturen sowie mit der Programmierung von Schleifen befassen.

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 wollen wir uns an einem kleinen Beispielprogramm verdeutlichen. 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.

Für unser Beispiel benutzen wir eine interessante Variable des CPC 464. Es handelt sich hierbei um die Variable 'TIME'. In dieser Variablen wird die Zeit, die seit dem Einschalten des Rechners vergangen ist, abgelegt. Die Zeit wird in Einheiten zu 1/300 Sekunden angegeben. Mit dem Befehl

```
PRINT INT(TIME/300)
```

können Sie also genau feststellen, wie viele Sekunden Ihr Rechner eingeschaltet ist. Kommen wir nun zu unserem

Beispielprogramm. Geben Sie als erstes den Befehl 'NEW' ein. Jetzt geben Sie das nachfolgende Programm in den Rechner.

```
10 CLS
20 PRINT CHR$(30);INT(TIME/300)
30 GOTO 20
40 END
```

Starten Sie dieses Programm, so werden laufend in der oberen linken Ecke des Bildschirms die Anzahl der Sekunden angezeigt, die seit dem Einschalten des Rechners vergangen sind. Der CHR$(30)-Code bewirkt, daß der Cursor immer wieder in die linke obere Ecke des Bildschirms gesetzt wird. Bei manchen Computern existiert dafür die sogenannte HOME-Taste.

Nun werden Sie in diesem Programm mit der Uhr feststellen, daß Sie es nur durch zweimaliges Betätigen der ESC-Taste unterbrechen können. 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 CLS:INPUT"EINGABE KOERPERGROESSE IN CM";CM
```

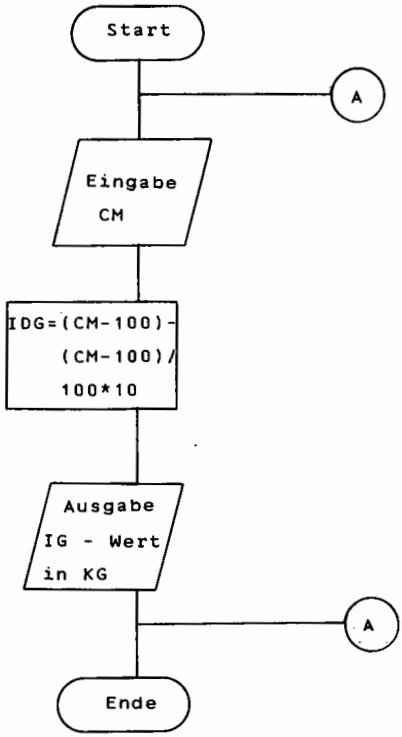
```
20 REM BERECHNUNG IDEALGEWICHT
30 IDG=(CM-100)-(CM-100)/100*10
40 REM AUSGABE
50 PRINT"IHR IDEALGEWICHT IST";IDG;"KG"
60 REM UNBEDINGTER SPRUNG MIT GOTO
70 GOTO 10
80 END
```

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 dazu wiederum die ESC-Taste betätigen. 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...ELSE

Trifft der Rechner bei der Programmausführung auf einen IF..THEN..ELSE-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 oder den Befehlen, die dem ´ELSE´ folgen. Alle Anweisungen oder Befehle, die dem THEN folgen, werden dann ignoriert. ´ELSE´ ist nur ein Zusatz, d.h. er muß nicht unbedingt dem IF...THEN folgen.

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 CLS:PRINT
30 PRINT "Geben Sie zwei Zahlen fuer ";CHR$(10)
40 PRINT"die obere und untere Grenze ein.";chr$(10)
50 INPUT"Untere Grenze";UG
60 INPUT"Obere Grenze";OG
70 Z=INT(OG*RND(1))+UG
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 CLS:PRINT "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 30: CHR$(10) = 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 65 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 Zahl, die der Spieler eingibt (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 müssen. 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.

Das obere Beispiel bietet sich gut dazu an, den Zusatz ´ELSE´ zu gebrauchen. Wir sparen dadurch die Programmzeile 50 ein. Im folgenden nun das abgeänderte Programm mit ´ELSE´.

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

Wenn Sie dieses Programm starten, so werden Sie feststellen, daß Sie das gleiche Ergebnis erhalten wie beim ersten Beispiel. Da also in den ersten 10 Vergleichen ´A´ nicht größer als 30 ist, die Bedingung also nicht erfüllt ist, wird der Befehl hinter dem ´ELSE´

ausgeführt. Das Programm verzweigt nach Zeile 20. Sobald 'A' den Wert 33 erhält, wird das Programm in Zeile 60 beendet. Die Zeilennummer 60 wurde hier bewußt beibehalten, um den Fortfall der Programmzeile 50 zu verdeutlichen. Damit steht Ihnen nun auch hier ein Befehl zur Verfügung, mit dem Sie Ihre Programme eleganter gestalten können. Außerdem sparen Sie so ganz nebenbei noch Programmzeilen ein.

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. Die besondere Schwierigkeit dieser Aufgabe soll darin bestehen, daß Sie den INSTR-Befehl nicht benutzen dürfen. Sie sollen also nebenbei eine Routine in Basic schreiben, die den INSTR-Befehl ersetzt.

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. Sie mußten den INSTR-Befehl simulieren. 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 CLS:PRINT
30 PRINT"Ausgabe der ersten 10 Quadratzahlen";CHR$(10)
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 (INKREMENT genannt), 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 (INKREMENT) 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:

Unexpected NEXT 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 nicht 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 auch keine Ausgabe, da der PRINT-Befehl innerhalb der Schleife liegt.
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 ENTER-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 überhaupt nicht durchlaufen wird. Das gleiche gilt 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 CPC 464. Man darf Sie nicht verallgemeinern. Es gibt bei den verschiedenen BASIC-Dialekten gerade in der Benutzung der FOR...NEXT-Schleifen einige Unterschiede.

3.2.3 WHILE...WEND

Mit der Befehlskombination ´WHILE...WEND´ steht Ihnen eine weitere Möglichkeit zur Verfügung, Schleifen innerhalb eines Programms aufzubauen. Diese Form der Schleifensteuerung ist jedoch gegenüber den FOR...NEXT-Schleifen flexibler zu gestalten. Sie benötigen bei WHILE...WEND kein fest vorgegebenes Inkrement, also keine feste Schrittweite.

Der Beginn der Schleife wird mit ´WHILE´ und das Ende der Schleife mit ´WEND´ gebildet. Der logische Ausdruck, der dem WHILE folgt, wird bei jedem Schleifendurchlauf überprüft. Solange der Ausdruck ´WAHR´ ist, wird die

Schleife bis WEND durchlaufen. Trifft die Bedingung hinter WHILE nicht mehr zu, wird mit der Ausführung des Programms nach dem WEND fortgefahren. Sie können mit dieser Befehlskombination eine Schleife rein zufällig abbrechen lassen, wie das folgende kleine Beispiel zeigt:

```
10 WHILE A < 100
20 A=INT(101*RND(1))
30 Z=Z+1
40 WEND
50 PRINT A,Z
60 END
```

Hier wird die WHILE...WEND-Schleife solange durchlaufen, wie ´A´ kleiner als 100 ist. Die Variable ´A´ bekommt ihre Werte durch die RND-Funktion zugewiesen. Erhält ´A´ nun den Wert 100, fährt das Programm in Zeile 50 fort. Dort werden ´A´ und der Zähler ´Z´ ausgegeben. Durch den Zähler erfahren Sie, wie oft die Schleife durchlaufen wurde. Ich hoffe, daß Ihnen durch dieses kleine Beispiel die Wirkungsweise der WHILE...WEND-Schleife klarer geworden ist.

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

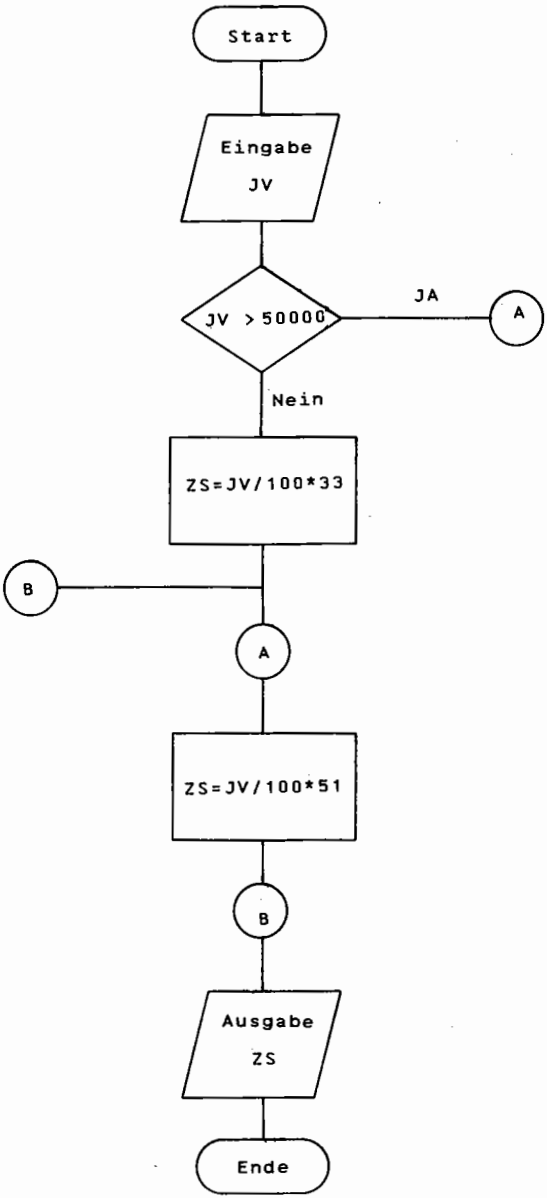
1. Steht die Anzahl der Wiederholungen der Schleifen von Anfang an fest, verwenden wir die FOR...NEXT-Schleife.

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

Auch hierzu haben wir schon eine Ausnahme im vorletzten 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 112). Auf den nächsten beiden Seiten folgen nun der Programmablaufplan sowie die dazugehörigen Erklärungen.

Programmablaufplan zum Programm "Berechnung Steuersatz"

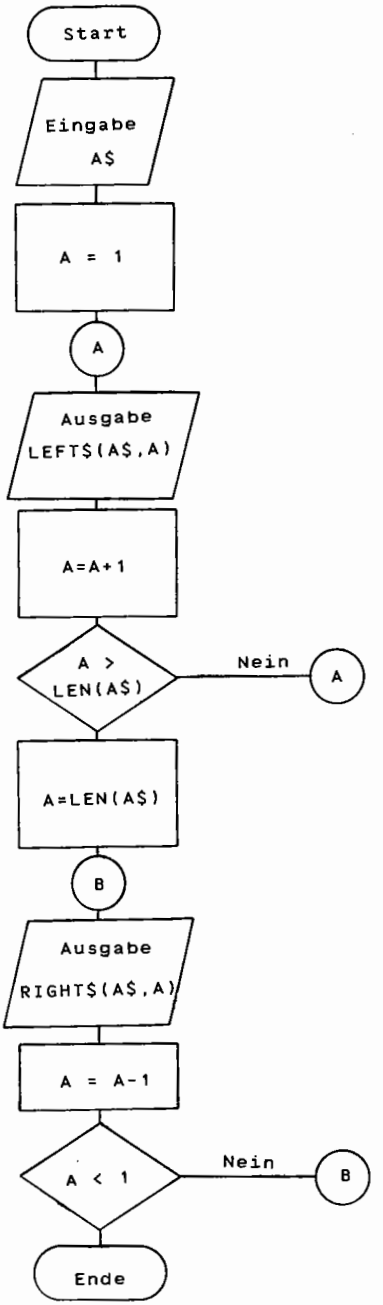


Das Startsymbol sowie das Eingabesymbol in unserem PAP dürften hinreichend bekannt sein. Die Raute ist, wie bereits erwähnt, das Symbol für eine logische Verzweigung. Sie besitzt einen sogenannten JA-Arm und einen NEIN-Arm. Trifft die Bedingung zu, so wird über den Absprungkonnektor A nach dem dazugehörigen 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 122 in einen PAP übertragen. Auf den nächsten beiden Seiten folgt wiederum der Programmablaufplan mit den dazugehörigen Erläuterungen.

Programmablaufplan zum Programm auf Seite 122



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 Einsprung-konnektor 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$(10) "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

Improper Argument in (Zeilennummer)

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 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$(10) "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 CPC 464 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. Legen Sie zuvor eine Kassette guter Qualität in das Kassettenlaufwerk und geben Sie folgenden Befehl in den Rechner, nachdem Sie das Programm eingegeben haben.

```
SAVE"RECHENLEHRGANG"
```

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

```
Press REC and PLAY then any key:
```

Drücken Sie nun die beiden Tasten am Kassettenlaufwerk herunter und anschließend zur Bestätigung eine weitere Taste auf der Tastatur. Das Programm wird danach 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. Sie können das Programm auch mit ´RUN"Rechenlehrgang"´ laden. Das Programm wird dann nach dem Laden automatisch gestartet.

Im folgenden nun das Programmlisting für den RECHENLEHRGANG.

```
5 REM ***** MENUE *****
10 CLS: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 CLS
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 CLS
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:PRINT TAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5) "Das Ergebnis lautet";ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(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 CLS
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 CLS
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:
GOTO 860
810 PRINT:PRINT TAB(10)"FALSCH !"
820 FOR I=0 TO 2000:NEXT I
830 F=F+1
840 IF F<=2 THEN 750
850 PRINT
860 FOR I=0 TO 2000:NEXT I
870 PRINT TAB(5)"Das Ergebnis lautet";ER
880 FOR I=0 TO 3000:NEXT I
890 PRINT TAB(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 CLS
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 CLS
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:PRINT TAB(10)"FALSCH !"
1190 FOR I=0 TO 2000:NEXT I
1200 F=F+1
1210 IF F<=2 THEN 1130
1220 PRINT
1230 FOR I=0 TO 2000:NEXT I
1240 PRINT TAB(5)"Das Ergebnis lautet";A2
1250 FOR I=0 TO 3000:NEXT I
1260 PRINT TAB(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 CLS
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 CLS
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:PRINT TAB(10)"FALSCH !"
1490 FOR I=0 TO 2000:NEXT I
1500 F=F+1
1510 IF F<=2 THEN 1430
```

```
1520 PRINT
1530 FOR I=0 TO 2000:NEXT I
1540 PRINT TAB(5)"Das Ergebnis lautet";ER
1550 FOR I=0 TO 3000:NEXT I
1560 PRINT TAB(5)"Noch eine Aufgabe  J/N";
1570 INPUT A$
1580 IF A$="J" THEN F=0:GOTO 1390
1590 GOTO 10
1600 CLS
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  < F E H L E R ! >
```

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, wollen wir noch den ON..ERROR..GOTO-Befehl besprechen, der in der Funktion sehr dem ON...GOTO-Befehl ähnelt. Vorher 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: Improper argument in (Zeilennummer).

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

3.3.2 PROGRAMMSPRÜNGE MIT ON..ERROR..GOTO

Diese spezielle Art des ON...GOTO-Befehls wird benutzt, um innerhalb eines Programms auftretende Fehler selber per Programm zu verwalten. Dazu wird meistens der ON..ERROR..GOTO-Befehl am Anfang eines Programms untergebracht. Der Rechner ´merkt´ sich diese Stelle, an der der Befehl zum ersten Mal auftritt. Alle Fehlermeldungen, die nach diesem Befehl auftreten, veranlassen das Programm, zu der Zeilennummer zu springen, die dem ON..ERROR..GOTO-Befehl folgt. Ab dieser Zeilennummer kann dann ein kleines Programm stehen, das auf den aufgetretenen Fehler entsprechend reagiert. Woher wissen wir aber, welcher Fehler in welcher Zeilennummer entstanden ist?

Auch hierfür gibt es im Basic des CPC 464 zwei Variablen, die daraufhin abgefragt werden können.

Es handelt sich dabei um

ERR

und

ERL.

Geben Sie

PRINT ERR

in den Rechner, so erhalten Sie die Fehlernummer des Fehlers. Im Handbuch können Sie dann unter der Fehlernummer nachschauen, um welche Art von Fehler es sich dabei handelt (Handbuch Anhang VIII S.1 ff.). In Ihrem Programm können Sie dann entsprechend dem Wert von ´ERR´ eine eigene Fehlermeldung ausgeben, ohne daß das Programm ungewollt abbricht. Mit der Variablen ´ERL´ können Sie zusätzlich noch die Zeilennummer abfragen, in der der Fehler auftrat.

Diese Art der eigenen Fehlerbehandlung hat den Vorteil, daß innerhalb des Programms erst bestimmte Schritte unternommen werden können, bevor man das Programm dann endgültig beendet. Somit können, z.B. in der Dateiverwaltung, beim Autreten eines Fehlers erst alle Dateien geschlossen werden (kein Datenverlust), bevor das Programm dann die eigene Fehlermeldung ausgibt und dann beendet wird.

Manchmal jedoch ist es gar nicht notwendig, das Programm zu beenden. Es reicht dann vollkommen aus, eine Fehlermeldung für den Anwender auszugeben und dann mit dem Programm fortzufahren. Für solch einen Fall existiert der Befehl

RESUME

Mit

RESUME (Zeilennummer)

bestimmen Sie, in welcher Zeile das Programm nach der Fehlermeldung fortfahren soll. Ein einfaches Beispiel soll dies wiederum verdeutlichen. Geben Sie das nachfolgende Programm in den Rechner und starten Sie es.

```
10 ON ERROR GOTO 1000
20 FOR I=-1 TO 3
30 ON I GOTO 200,300
40 NEXT I
100 END
200 CLS
210 PRINT "Zeile 210"
220 FOR A=1 TO 2000:NEXT
230 GOTO 40
300 CLS
310 PRINT "Zeile 310"
320 FOR A=1 TO 2000:NEXT
330 GOTO 40
1000 REM Fehlerbehandlung
1010 CLS
1020 PRINT "Fehler";ERR;"in Zeile";ERL
1030 FOR A=1 TO 2000:NEXT
1030 RESUME 40
```

In diesem Programm steckt ein Fehler in Zeile 30. Die Laufvariable ´I´ hat als ersten Wert -1. Nun soll in Zeile 30 über ´I´ in bestimmte Zeilen gesprungen werden. Wie wir aber wissen, funktioniert dies nicht mit negativen Werten. Daher springt dieses Programm beim ersten Durchlauf in die Fehlerroutine ab Zeile 1000 und gibt dort eine entsprechende Meldung aus. Jetzt veranlaßt der RESUME-Befehl, daß das Programm in Zeile 40 mit der Programmausführung fortfährt. Es werden nun noch die beiden Zeilen 200 und 300 abgearbeitet und dann wird das Programm in Zeile 100 ordnungsgemäß beendet. Wie Sie an

diesem Beispiel gesehen haben, ist der Gebrauch dieser Befehle doch recht einfach. Sie können in Ihren Programmen selbstverständlich den Fehler noch genauer beschreiben, indem Sie entsprechend dem Wert von ´ERR´ eine Meldung ausgeben lassen.

Sie haben natürlich auch die Möglichkeit, ein Programm mit der ´Originalfehlermeldung´ des CPC 464 zu beenden. Der Befehl

ERROR X

gibt Ihnen entsprechend dem Wert von ´X´ die dazugehörige Fehlermeldung aus und bricht das Programm ab.

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 17 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. 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:CLS
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 CLS
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. Danach wird 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 CLS: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. Zu beachten 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 INKEY$

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. Sie müßten dann den LINE INPUT-Befehl als Ersatz nehmen, wobei aber wieder nur einer Variablen ein Wert übergeben werden kann. 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 INKEY$- bzw. INKEY-Befehl Verwendung.

3.4.1 EINGABE VON DATEN MIT INKEY$

Der INKEY$-Befehl wird, wie der INPUT-Befehl übrigens auch, meistens nur im Programmodus benutzt.

Wie arbeitet nun der INKEY$-Befehl?

Dazu müssen wir zunächst wissen, daß der CPC 464 einen sogenannten Tastaturpuffer besitzt. Es handelt sich dabei

um einen kleinen Speicher, in dem bis zu 20 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 20 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 INKEY$-Befehl überprüft nun, ob im Tastaturpuffer Zeichen vorhanden sind. Ist das der Fall, so wird durch INKEY$ das erste Zeichen ausgelesen und der Variablen vor dem INKEY$ zugeordnet. Ist der Puffer leer, so wird der Variablen der Wert Null zugeordnet. Der INKEY$-Befehl arbeitet andauernd, d.h. daß er kurz überprüft, ob Zeichen im Tastaturpuffer vorhanden sind, und dann mit der Programmausführung fortfährt. Damit kann man nun noch keine vernünftige Datenübernahme gewährleisten. Daher findet man beim INKEY$-Befehl meistens eine Schleife, die überprüft, ob ein Zeichen eingegeben wurde. Wurde kein Zeichen eingegeben, so wird die Zeile mit INKEY$ erneut abgearbeitet. Zu dieser Anwendung wieder ein Beispiel:

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

Geben Sie das Beispiel wieder in den Computer ein und starten Sie es. Beachten Sie, daß bei INKEY$, im Gegensatz zu INPUT, kein Cursor erscheint. Dieses kleine Programm wartet solange, bis Sie irgendeine Taste

drücken. In Zeile 10 wird mit INKEY$ 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 INKEY$ arbeitet, nehmen wir eine kleine Änderung an unserem Programm vor.

```
10 CLS
20 A$=INKEY$:Z=Z+1:PRINT CHR$(30) Z:IF A$="" THEN 20
30 PRINT A$
```

Wir haben jetzt in Zeile 20 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$(30) setzt den Cursor jedesmal in die linke obere Bildschirmecke.

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. Eine Ausnahme bildet hier nur die Taste ´ESC´. Schauen wir uns dazu ein Beispiel an.

```
10 REM TASTENSELEKTIERUNG MIT INKEY$
20 CLS
30 A$=INKEY$:IF A$ = "" THEN 30
40 IF A$=CHR$(97) THEN PRINT"Taste a":GOTO 30
50 IF A$=CHR$(98) THEN PRINT"Taste b":GOTO 30
60 IF A$=CHR$(101) 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, außer der ESC-Taste, da sie nicht speziell abgefragt werden kann.
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.), rate ich Ihnen, nachdem Sie das Programm ausprobiert haben, den Rechner in den Einschaltzustand zurückzusetzen (CTRL, SHIFT, ESC). Im folgenden nun das Programm:

```
10 REM ANZEIGE DER ASCII-WERTE
20 CLS
30 A$=INKEY$:IF A$="" THEN 30
40 PRINT TAB(6) CHR$(ASC(A$)) TAB(12)ASC(A$)
50 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. Der zweite Teil der Zeile gibt schließlich den ASCII-Wert aus. Zeile 50 springt dann wieder zur Zeile 30.

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

```
10 REM VIER ZEICHEN MIT INKEY$ EINLESEN
20 FOR I = 1 TO 4
30 A$=INKEY$: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 INKEY$ 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 INKEY$ einzulesen. Dabei muß ja jede einzelne Ziffer eingelesen und nachträglich zu einer vierstelligen Zahl zusammengesetzt werden.

Sie haben nun einige Anwendungsbeispiele des INKEY$-Befehls kennengelernt. Die wichtigste Eigenschaft ist wohl die der Selektierung der einzelnen Tasten. Dadurch kann man bei eigenen Programmen die Menüsteuerung fast narrensicher gestalten. Wir wollen nun in unserem Rechenlehrgang den INPUT-Befehl durch den INKEY$-Befehl ersetzen. Auf der nächsten Seite sehen Sie, wie die ersten beiden INPUT-Befehle durch die INKEY$-Befehle ersetzt wurden. Studieren Sie genau, welche Abfragen mit IF...THEN durchgeführt wurden, um nur bestimmte Zeichen bei der Eingabe zuzulassen. Hier hat es sich schon bezahlt gemacht, daß wir in Zehnerschritten programmiert haben. So fällt es uns nicht schwer, die zusätzlichen Zeilen, die für den INKEY$-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 A$=INKEY$: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 CLS
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 A$=INKEY$: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 INKEY$-Befehl mit der bekannten Abfrage auf einen Leerstring untergebracht. Zeile 160 hat sich insofern verändert, daß wir den VAL-Befehl benutzt haben, um den String A$ 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 INKEY$-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. Sie werden sehen, daß das Programm nur die Tasten 1 bis 5 zuläßt.

Einen kleinen Makel hat unsere INKEY$ 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 Cursor, der genau unsere Position angab, und wir sahen auch, welche Daten wir bis zum Drücken der ENTER-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 INKEY$-Befehl in unseren Programmen einsetzen zu können.

3.4.2 TASTATURABFRAGE MIT INKEY

Auf der Seite 152 haben wir bereits gesehen, wie mit dem INKEY$-Befehl nur bestimmte Tasten innerhalb eines Programms zugelassen werden können. Dort wurden zur Abfrage der Tasten die CHR$-Codes verwendet.
Der CPC 464 besitzt aber nun noch einen eigenen Tastencode für die Tastatur (s. Anhang). Dieser Code kann mit dem INKEY-Befehl abgefragt werden. Dieser Zahlencode ist NICHT identisch mit dem ASCII-Code. Der Vorteil des INKEY-Befehls gegenüber dem INKEY$-Befehl liegt darin, daß mit dem INKEY-Befehl alle Tastenkombinationen mit SHIFT und CTRL ebenfalls erfaßt werden. Die Schreibweise (Syntax) des Befehls sieht wie folgt aus:

```
INKEY(X)
```

´X´ steht hier stellvertretend für den Zahlencode einer Taste, z.B. 52 für die Taste ´G´ (s. Anhang). Das folgende kleine Programm können Sie durch Betätigen der Taste ´G´ beenden.

```
10 REM Tastaturabfrage mit INKEY
20 CLS:IF INKEY(52)=0 THEN 30 ELSE 20
30 PRINT "G"
40 END
```

Haben Sie das Programm gestartet und betätigen die entsprechende Taste, erhalten Sie den Buchstaben ´G´ ausgegeben und das Programm wird beendet. Der Rechner meldet sich wieder mit ´Ready´. Doch halt, wieso wurde denn jetzt noch ein ´G´ ausgegeben? Die Antwort auf diese Frage ist recht einfach. Der Tastaturpuffer wird mit INKEY nicht abgefragt, so daß die gerade betätigte Taste im Tastaturpuffer gespeichert und nach dem Programmende ausgegeben wird. Dies ist ein wichtiger Unterschied zum INKEY$-Befehl.

Im Programm oben wurde der Wert von INKEY(52) direkt

abgefragt. Sie können diesen Wert jedoch auch einer Variablen zuordnen und dann die Variable auf diesen Wert hin überprüfen. Das könnte dann ungefähr so aussehen:

```
10 CLS
20 A=INKEY(52):IF A=0 THEN 30 ELSE 20
30 PRINT "G"
40 END
```

Wie Sie gesehen haben, wurde INKEY(52) auf den Wert 'Null' überprüft. Wird also eine Taste gedrückt, in diesem Fall die Taste 'G', erhält INKEY den Wert 'Null' zurück. Bei nicht gedrückter Taste ist dieser Wert -1. Wollen Sie eine Tastenkombination mit SHIFT abfragen, so müssen Sie den Wert 32 verwenden. Dieser Wert beträgt bei CTRL 128 und bei der Kombination SHIFT und CTRL 160. Das untere Diagramm soll das noch einmal verdeutlichen.

KOMBINATION	*WERT ABFRAGEN AUF*
KEINE TASTE	*-1*
TASTE GEDRÜCKT	*0*
SHIFT + TASTE	*32*
CTRL + TASTE	*128*
CTRL + SHIFT + TASTE	*160*

Bevor wir nun zu einer weiteren Anwendung mit INKEY$ bzw. INKEY kommen, muß noch ein weiterer Befehl besprochen werden, der im nächsten Programm verwendet werden soll. Es handelt sich hierbei um den

```
LOCATE X,Y
```

Befehl. Er bewirkt eine Positionierung des Cursors durch die Angabe von Spalten- und Zeilennummer (X,Y). Schauen Sie sich nun zunächst das folgende Programm an.

```
10 REM Reaktionstest
20 CLS
30 FOR I=0 TO INT(10000*RND(1))+1:NEXT
40 LOCATE 21,13:PRINT CHR$(224)
50 Start=INT(TIME/3)
60 A$=INKEY$:Zeit = ((TIME/3)-Start):IF A$="" THEN 60
70 LOCATE 1,24
80 PRINT USING"Sie haben###.## Sekunden
benoetigt.";Zeit/100
90 FOR I=0 TO 4000:NEXT
100 CLS
110 LOCATE 5,5
120 PRINT "Wollen Sie nochmal (j/n) ?"
130 A$=INKEY$:IF A$="j" THEN 20
140 IF A$="n" THEN 150 ELSE 130
150 END
```

In der Zeile 30 wird mittels einer FOR...NEXT-Schleife eine zufällige Zeitverzögerung generiert. In Zeile 40 wird die Cursorposition mit dem LOCATE-Befehl auf die 21. Spalte und die 13. Zeile gesetzt. Dadurch erscheint nach dem Ablauf der Zeitschleife in Zeile 30 das ´Gesicht´ (CHR$(224)) ziemlich genau in der Mitte des Bildschirms. Zeile 50 übernimmt dann den aktuellen Wert der Variablen ´TIME´ in die Variable ´Start´. In Zeile 60 wird mit INKEY$ abgefragt, ob schon eine Taste gedrückt wurde. Danach wird die Variable ´TIME´ durch 3 dividiert. Damit erhält man die hundertstel Sekunden Einheit. Von diesem Wert wird die Variable ´Start´ subtrahiert, da hier die Zeit abgelegt wurde, die seit dem Einschalten des Rechners vergangen ist. Dieser neue Wert wird der Variablen ´Zeit´ zugeordnet. Jetzt wird durch den Inhalt von A$ überprüft, ob eine Taste betätigt wurde. Zeile 70 positioniert dann wiederum den Cursor und in Zeile 80 wird die benötigte Zeit mit einem entsprechenden Text versehen ausgegeben. In Zeile 90 befindet sich wieder eine Zeitschleife zum Lesen der Meldung. Die restlichen Programmzeilen dürften in ihrer Bedeutung verstanden worden sein.

Das soll nun vorerst an Beispielen für die Verwendung des INKEY- bzw. INKEY$-Befehls reichen. Dieser Befehl wird Ihnen sowieso noch in anderen Programmen oft genug begegnen, wo Sie dann dessen Verwendung genau analysieren können.

Wir haben nun gelernt, wie wir mit INKEY innerhalb unserer Programme bestimmten Tasten bestimmte Funktionen zuordnen können.
Mit dem

```
KEY
```

Befehl können Sie ebenfalls einer oder mehreren Tasten eine bestimmte Funktion übertragen. Sie können so z.B. der Zahlentaste 1 auf dem Zehnerblock den Befehl ´CLS´ zuordnen.

```
KEY 1,"CLS"+CHR$(13)
```

Der CHR$(13)-Code (Carriage Return) bewirkt, daß beim Betätigen der Taste 1 der Befehl sofort ausgeführt wird. Wäre dieser Code nicht angegeben worden, so müßte man zusätzlich noch die ENTER-Taste betätigen, damit der Befehl zur Ausführung gelangt.
Ein Befehl, der in diesem Zusammenhang oft mit benutzt wird, ist der

```
KEY DEF A,B,X,Y,Z
```

Befehl. Mit diesem Befehl haben Sie die Möglichkeit, einer Taste ein anderes Zeichen zuzuordnen. Dabei bedeutet ´A´ die Tastennummer, ´B´ Wiederholungsfunktion (0=AUS/1=AN), ´X´ das normale Zeichen, ´Y´ das Zeichen mit SHIFT und ´Z´ das Zeichen mit CTRL. Der folgende Befehl ordnet der Taste ´G´ das umgekehrte Fragezeichen zu.

```
KEY DEF 52,1,174,174,174
```

Betätigen Sie jetzt nacheinander die Taste ´G´ in Kombination mit SHIFT und CTRL, so erscheint auf dem Bildschirm das umgekehrte Fragezeichen.

Mit diesen Befehlen können Sie also die Tastaturbelegung ändern. Mit den beiden nächsten Befehlen ändern Sie die Zeichen selber. Der Befehl

```
SYMBOL AFTER X
```

gibt mit ´X´ den ASCII-Wert des Zeichens an, ab dem die Änderung stattfinden soll. Der Standardwert beträgt 240. Geben Sie also

```
SYMBOL AFTER 32
```

ein, so können Sie alle Zeichen ab dem ASCII-Wert 32 selbst definieren. Wollen Sie also anfangen, einen dänischen Zeichensatz zu generieren, gebe ich Ihnen hier schon mal die Werte für das dänische ´A´ mit dem Kreis.

```
SYMBOL AFTER 65
SYMBOL 65,24,36,24,60,102,126,102,102
```

Wenn Sie danach SHIFT A drücken, werden Sie den Unterschied bemerken.

Schauen wir uns nun im nächsten Kapitel noch einige Befehle bzw. Funktionen an, welche in Programmen nicht so häufig verwendet werden. Man sollte jedoch wissen, mit "wem" man es zu tun hat, wenn ein solcher Befehl in einem Programm einmal Verwendung findet.

3.5 CALL, DI, FRE, POS, UNT, WAIT UND ANDERE

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.

ABS

Diese Funktion ergibt den Absolutwert einer Zahl X (nicht vorzeichenbehaftet). Der Absolutwert einer negativen Zahl ergibt sich mit der Multiplikation mit -1.

```
PRINT ABS(-464)
```

Ausgabe:

```
464
```

CALL (X)

Mit diesem Befehl können Sie an eine Adresse im Rechner springen, die außerhalb des Basicbereichs liegt. Bei der Benutzung dieses Befehls ist eine gewisse Vorsicht geboten, da er beim Anwender Kenntnisse des Betriebssystems voraussetzt. Bei der Programmierung in Basic werden Sie diesen Befehl vorerst nicht benötigen.

DI

Der Befehl DI (engl. = Disable Interrupt) verhindert alle Unterbrechungen (Interrupts), die z.B. durch die Befehle ´EVERY´ oder ´AFTER´ hervorgerufen werden. Ausgenommen ist hiervon die ESC-Taste. Die Befehle ´EVERY´ und ´AFTER´ werden in einem späteren Kapitel besprochen.

EI

Der Befehl EI (engl. = Enable Interrupt) hebt die Wirkung des Befehls DI wieder auf. Nach Ausführung dieses Befehls sind alle Unterbrechungen wieder zulässig.

FRE

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

FRE(X)

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

PRINT FRE(X)

Als Ausgabe erhalten Sie den zur Verfügung stehenden freien Basicspeicherplatz. 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.

INP

Mit dieser Funktion ermitteln Sie den Wert, der an der angegebenen Speicheradresse der Eingabeschnittstelle anliegt. Die Anwendung der Funktion geschieht wie folgt:

PRINT INP(&FF00)

JOY

Diese Funktion unterstützt Sie bei der Programmierung eigener Spiele. Das folgende kleine Programm gibt Ihnen die Werte der einzelnen Stellungen des Joysticks an, so daß Sie die Werte in eigenen Programmen abfragen und verwenden können.

```
10 CLS
20 PRINT CHR$(30);JOY(0)
30 GOTO 20
```

OUT

Diese Funktion schickt einen Wert zwischen 0 und 255 an eine angegebene Adresse, z.B.:

```
OUT &FF0A,205
```

PEEK

Mit dieser Funktion können Sie sich den Inhalt einer Speicheradresse des CPC 464 ausgeben lassen, z.B.:

```
PRINT PEEK(368)
```

POKE

Mit dem POKE-Befehl können Sie einen Wert in eine bestimmte Speicherstelle schreiben, z.B.:

```
POKE 366,10
```

POS

Diese Funktion wird Ihnen innerhalb eines Programms

selten begegnen. Mit ihr kann man die aktuelle Cursorposition innerhalb einer Bildschirmzeile erfahren. Folgende Beispiele sollen die Funktion näher erläutern. Geben Sie im Direktmodus folgendes in den Rechner ein:

```
PRINT "TEST" POS(#0);"TEST A" POS(#0)
```

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

```
TEST 5 TEST A 14
```

Die Zahl fünf gibt die Position des Cursors nach der ersten Ausführung des PRINT-Befehls bekannt. Dementsprechend zeigt die Zahl 14 die Position des Cursors nach Ausführung des zweiten PRINT-Befehls an. Beim POS-Befehl muß das Ein-/Ausgabegerät immer mit angegeben werden. In diesem Fall handelte es sich um den Bildschirm (#0).

VPOS

Diese Funktion ähnelt der von POS. Hierbei erhalten Sie jedoch den Wert der vertikalen Position des Cursors. Auch hier muß wieder das Ein-/Ausgabegerät mit angegeben werden (PRINT VPOS (#0)).

SPEED KEY

Mit diesem Befehl können Sie die Verzögerung und die Wiederholungsgeschwindigkeit der Tasten beeinflussen. Die Zeiteinheiten betragen dabei 0.02 Sekunden. Geben Sie also den Befehl

```
SPEED KEY 10,5
```

ein, so wird 0.2 Sekunden nach Betätigung einer Taste diese alle 0.1 Sekunden wiederholt.

SPEED WRITE

Mit diesem Befehl bestimmen Sie die Schreibgeschwindigkeit, mit der auf der Kassette gespeichert werden soll. Der Standardwert beträgt 1000 Baud (Baud = Bits pro Sekunde) und entspricht ´SPEED WRITE 0´. ´SPEED WRITE 1´ erhöht den Wert auf 2000 Baud.

UNT

Diese Funktion rechnet die Werte im Bereich von 0 bis 65535 in Werte des Bereichs von -32768 bis +32767 um, z.B.:

```
PRINT UNT(50000)
```

WAIT

Dieser Befehl wartet, bis eine Ein-/Ausgabeadresse einen bestimmten Wert zwischen 0 und 255 angenommen hat.

WIDTH

Dieser Befehl begrenzt die Anzahl der zu druckenden Zeichen in einer Zeile bei der Druckausgabe, z.B.:

```
WIDTH 90
```

Das soll soweit zur Erklärung dieser Befehle genügen. In den meisten Ihrer selbstgeschriebenen Programme werden diese Befehle sehr selten auftauchen. Wollen Sie sich dennoch eingehend mit der Funktionsweise des einen oder

anderen Befehls vertraut machen, so sei hier auf die entsprechende weiterführende Literatur verwiesen.

Im nächsten Kapitel befassen wir uns mit der Möglichkeit, Informationen bzw. Daten als Konstanten in Programmen abzulegen, so daß sie jederzeit abrufbereit sind.

3.6 READ, DATA, UND RESTORE

Bisher haben wir nur die Möglichkeit kennengelernt, Daten von der Tastatur aus einzugeben. Dazu benutzten wir die Befehle INPUT oder INKEY$. 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 Kommas 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

```
DATA exhausted 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:

```
DATA exhausted 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 ENTER-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. 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 und Leerzeichen 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 INKEY$-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 INKEY$ 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 CALL-Befehl das Maschinenprogramm starten.

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 DU=S/12
80 DU=INT(DU*100)/100
90 PRINT"Durchschnittliches Einkommen";
100 PRINT DU;"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 DU=S/12
170 DU=INT(DU*100)/100
180 PRINT"Durchschnittliches Einkommen";
190 PRINT DU;"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.

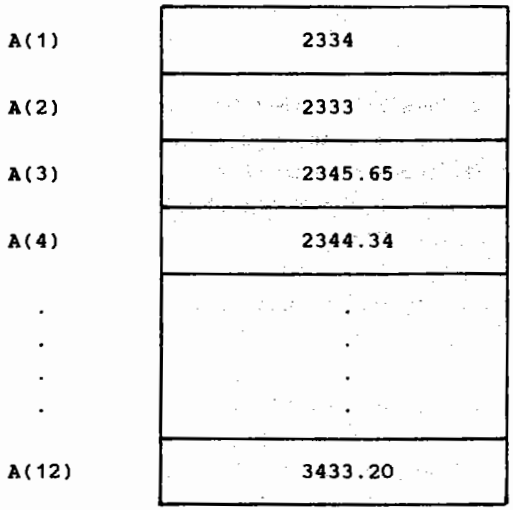
Was versteht man nun genau unter dem Begriff Feld oder Array?

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

Array already dimensioned 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 178 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 DU=S/12
90 DU=INT(DU*100)/100
100 PRINT"Durchschnittliches Einkommen";
110 PRINT DU;"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 DU=INT(DU*100)/100
100 FOR I=1 TO 12
110 PRINT"MONATSGEHALT"I,M(I)
120 NEXT I
130 PRINT"Durchschnittliches Einkommen";
140 PRINT DU;"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, das außerhalb des mit DIM(X) dimensionierten Feldes liegt, gibt der Rechner die Fehlermeldung

Subscript out of range

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.

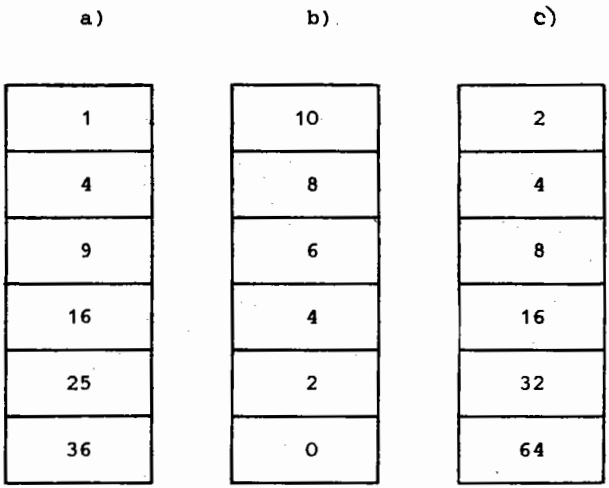
Das sieht dann wie folgt aus:

```
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.

Beachten Sie, daß Sie mit dieser Methode nur die Inhalte der Felder löschen, d.h. die Dimensionierung bleibt erhalten. Verwenden Sie dagegen den ERASE-Befehl, so wird das komplette Feld gelöscht, d.h. die Dimensionierung wird mit gelöscht.

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 A$=INKEY$: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

Subscript out of range 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 WOTA$(7)
30 FOR I=1 TO 7
40 READ WOTA$(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 A$=INKEY$:IF A$="" THEN 90
100 IF A$ < > "j" THEN 140
110 FOR I=1 TO 7
120 PRINT WOTA$(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):

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)
0	2	6	12	20	30

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$

	Spalte 1	Spalte 2	Spalte 3
Zeile 1			
Zeile 2			
Zeile 3			
Zeile 4			
Zeile 5			

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 Kommas 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 DATA-Zeile 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 A$=INKEY$: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.

FELD A(3,4)

```
11  *  *  *    11 12  *  *    11 12 13  *    11 12 13 14
 *  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *
 *  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *

11 12 13 14    11 12 13 14    11 12 13 14    11 12 13 14
21  *  *  *    21 22  *  *    21 22 23  *    21 22 23 24
 *  *  *  *     *  *  *  *     *  *  *  *     *  *  *  *

11 12 13 14    11 12 13 14    11 12 13 14    11 12 13 14
21 22 23 24    21 22 23 24    21 22 23 24    21 22 23 24
31  *  *  *    31 32  *  *    31 32 33  *    31 32 33 34
```

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 A$=INKEY$: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 189). 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$(120,3)
30 CLS
40 Z=Z+1
50 INPUT"Vorname";X$(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 A$=INKEY$: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 CPC 464 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 folgende Schaubild 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. Versuchen Sie die Aufgabe zu lösen, ohne direkt auf untenstehende Lösung zu schauen.

LÖSUNG

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 bzw. 64. 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

Unexpected RETURN 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

Unexpected RETURN 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 CLS:PRINT
30 PRINT CHR$(24);Z;CHR$(24)
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 siebzehnmaligem Durchlaufen

des Unterprogramms der gesamte Programmablauf mit der Fehlermeldung

Memory full in 100

abgebrochen wird. Der Fehler liegt hier nicht im Aufruf, sondern im Verlassen des Unterprogramms. Das Programm löscht nach dem Start den Bildschirm und erzeugt 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$(24)) 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 17. 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 135 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 CLS
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 CLS
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:PRINT TAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
```

```
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5)"Das Ergebnis lautet";ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(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 Menü 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 REM
```

```
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
  .
  .
  .
  .
```

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 CLS: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 E$=INKEY$: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 135 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 INKEY$ 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 CLS:A$="":B$=""
180 PRINT TAB(10)"Geben Sie die groesste Zahl "
190 PRINT
200 PRINT TAB(10)"fuer die "RA$(P)" ein."
210 PRINT
220 PRINT TAB(10)"groesste ? ";
230 FOR I=1 TO 3
240 A$=INKEY$:IF A$="" 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 INKEY$ wurde schon auf Seite 154 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 CLS
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
```

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 den Befehl ´MERGE´ nachladen und an das Programm anhängen.

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 Variablen-bezeichnungen 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 REM ************************
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 CLS:A$="":B$=""
180 PRINT TAB(10)"Geben Sie die groesste Zahl "
190 PRINT
200 PRINT TAB(10)"fuer die "RA$(P)" ein."
210 PRINT
220 PRINT TAB(10)"groesste ? ";
230 FOR I=1 TO 3
240 A$=INKEY$:IF A$="" THEN 240
250 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 240
```

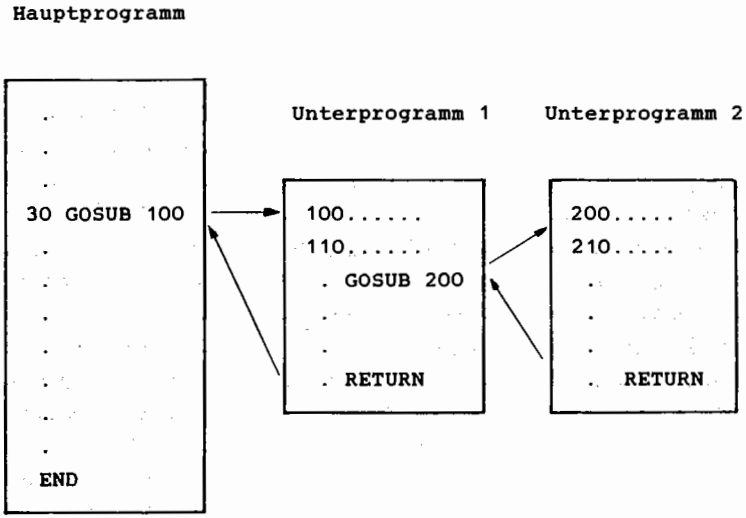
```
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=VAL(B$)
290 RETURN
300 REM ***************************
310 REM * Erzeugen Zufallszahlen *
320 REM ***************************
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
360 REM ********************
370 REM * Aufgabenstellung *
380 REM ********************
390 CLS
400 PRINT
410 PRINT"Wieviel ergibt";A1;BE$(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 CLS: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 E$=INKEY$: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 CLS
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. Versuchen Sie auch hier, die Aufgabe selbständig zu lösen.
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 CLS
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.

Nach diesem Lösungsvorschlag wollen wir nun die wichtigsten Dinge, die Sie bei der Anwendung von Unterprogrammen beachten müssen, noch einmal kurz zusammenfassen:

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.

Damit hätten wir die Unterprogrammtechnik eingehend besprochen. Es existieren jedoch noch drei weitere Befehle, die ebenfalls bei der Benutzung von Unterprogrammen eine Rolle spielen. Diese Befehle unterscheiden sich von den GOSUB- und ON..X..GOSUB-Befehlen darin, daß sie nur dann in ein Unterprogramm verzweigen, wenn eine bestimmte Taste gedrückt wurde. Die o.a. Regeln gelten daher auch für diese Befehle.

Im folgenden sollen diese Befehle kurz erklärt werden. Der Befehl

ON BREAK GOSUB

verzweigt nur dann in ein Unterprogramm, wenn vorher zweimal die ESC-Taste gedrückt wurde. Sie können also in bestimmten Programmteilen diese Taste außer Funktion setzen, d.h. man kann das Programm mit dieser Taste nicht mehr unterbrechen. Das folgende Programm soll Ihnen die Funktionsweise dieses Befehls verdeutlichen.

```
10 CLS
20 ON BREAK GOSUB 100
30 LOCATE 3,5
40 PRINT "Druecken Sie zweimal die ESC-Taste."
50 FOR I=1 TO 100:NEXT I
```

```
60 LOCATE 3,5
70 PRINT CHR$(24);"Druecken Sie zweimal die
ESC-Taste.";CHR$(24)
80 FOR I=1 TO 100:NEXT I
90 GOTO 30
100 CLS
110 LOCATE 2,12
120 PRINT "Die ESC-Taste wurde zweimal betaetigt."
130 FOR I=1 TO 1000:NEXT I
140 CLS:RETURN
```

Starten Sie das Programm in der o.a. Form, so können Sie es nur beenden, indem Sie den Rechner in den Einschaltzustand zurücksetzen (CTRL, SHIFT u. ESC). Daher existiert noch ein Befehl, der den ON BREAK GOSUB-Befehl ergänzt. Es handelt sich um den

ON BREAK STOP

Befehl. Verändern Sie nun das o.a. Programm wie folgt:

```
140 ON BREAK STOP
150 CLS:RETURN
```

Wenn Sie das Programm jetzt starten, so wird beim ersten Doppel-ESC das Unterprogramm ausgeführt. Mit dem zweiten Doppel-ESC können Sie das Programm wieder unterbrechen.

Zum Schluß sei hier noch der Befehl

ON SQ GOSUB X

erwähnt. Er wird hauptsächlich bei der Programmierung von Sound benutzt. Das Programm verzweigt in Abhängigkeit des Wertes von SQ in die angegebenen Zeilennummern, die dem GOSUB folgen.

4.2.1 UNTERPROGRAMME MIT AFTER UND EVERY

Der CPC 464 bietet noch zwei weitere Befehle zum Aufruf von Unterprogrammen an. Der Unterschied zu den bisher erwähnten Befehlen liegt darin, daß die nun folgenden Befehle in Abhängigkeit eines bestimmten Zeitfaktors ein Unterprogramm aufrufen können. Mit dem Befehl

AFTER X,Y GOSUB (Zeilennummer)

können nach einer gewissen Zeitspanne Unterprogramme ausgeführt werden. Hierbei bestimmt ´X´ die Zeitspanne, nach deren Ablauf das Unterprogramm ausgeführt werden soll. Die Zeit ist in Einheiten zu 1/50 Sekunden unterteilt. Mit ´Y´ wird einer der vier zur Verfügung stehenden Zeitgeber ausgewählt. Der Wert muß zwischen 0 und 3 liegen, wobei der Standardwert hier 0 ist. Ist die vorgegebene Zeit abgelaufen, wird in das entsprechende Unterprogramm verzweigt. Dieses Unterprogramm wird ganz normal mit einem RETURN beendet. Das Programm kann dann mit der Ausführung des Hauptprogramms an der Stelle fortfahren, an der es zuvor in das Unterprogramm gesprungen ist. Das folgende Programm gibt nach einer Zeitspanne von 20 Sekunden eine entsprechende Meldung aus. Zur Überprüfung der Zeitspanne wird in der oberen linken Bildschirmecke die laufende Zeit in Sekunden angezeigt.

```
10 ZEIT=INT(TIME/300)
20 CLS
30 AFTER 1000 GOSUB 70
40 ZEIT1=INT(TIME/300)-ZEIT
50 PRINT CHR$(30);ZEIT1;"SEC"
60 GOTO 40
70 REM Unterprogramm
80 CLS
90 LOCATE 5,14
100 PRINT "Es sind ";ZEIT1;" Sek. vergangen."
```

```
110 RETURN
```

Sie können das Programm danach mit der ESC-Taste unterbrechen. Wollen Sie, daß das Programm alle 20 Sekunden solch eine Meldung ausgibt, so benötigen Sie dazu den

```
EVERY X,Y GOSUB (Zeilennummer)
```

Befehl. Mit dem EVERY-Befehl haben Sie die Möglichkeit, in regelmäßigen Zeitintervallen ein Unterprogramm ausführen zu lassen. Wir ändern das Programmbeispiel mit ´AFTER´ wie folgt ab:

```
30 EVERY 1000 GOSUB 70
```

Starten Sie das Programm jetzt erneut, so wird alle 20 Sekunden eine entsprechende Meldung angezeigt. In diesem Zusammenhang soll noch auf die Funktion

```
REMAIN
```

verwiesen werden. Mit

```
PRINT REMAIN (0)
```

erfahren Sie die noch verbleibende Zeit des Zeitgebers ´0´. Sie können hier die Werte von 0 bis 3 einsetzen. Andere Werte haben die Fehlermeldung

```
Improper argument
```

zur Folge.

Damit wollen wir das Kapitel über die Unterprogramme abschließen. Im nächsten Kapitel werden wir uns 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 ESC-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 Bildschirmfarben sollen komplett beibehalten 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.

```
****************************************
*                                      *
*        MATHEMATISCHE  TABELLE        *
*                                      *
****************************************
*                                      *
* 1 QUADRATWURZEL                      *
*                                      *
* 2 SINUS                              *
*                                      *
* 3 COSINUS                            *
*                                      *
* 4 NATUERLICHER LOGARITHMUS           *
*                                      *
* 5 DEKADISCHER LOGARITHMUS            *
*                                      *
* 6 PROGRAMMENDE                       *
*                                      *
*                                      *
* GEBEN SIE IHRE WAHL EIN: (1-6)       *
*                                      *
*                                      *
*                                      *
****************************************
```

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 INKEY$-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 REM
50 CLS
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 CLS
240 PRINT STRING$(40,"*");
250 PRINT"*                                      *";
260 PRINT"*         Mathematische Tabelle        *";
270 PRINT"*                                      *";
280 PRINT STRING$(40,"*");
290 RETURN
```

```
300 REM *********
310 REM * MENUE *
320 REM *********
330 GOSUB 230
340 FOR I=1 TO 19
350 PRINT"*                                        *";
360 NEXT I
370 PRINT STRING$(40,"*");
380 PRINT CHR$(30);
390 FOR I=1 TO 5:PRINT:NEXT I
400 FOR I=1 TO 6
410 PRINT CHR$(10);CHR$(9);M$(I)
420 NEXT I
430 PRINT
440 PRINT CHR$(10);CHR$(9);
450 PRINT"  Geben Sie Ihre Wahl ein  (1-6)";
460 INPUT W$
 .
 .
```

Zunächst sollen die einzelnen Programmzeilen kurz erklärt werden. Die Zeile 50 löscht 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 STRING$-Befehle 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 fünf 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 und nach unten gerückt, damit die Ausgabe den linken Menürahmen nicht überschreibt und jeweils eine Leerzeile zwischen den einzelnen Menüpunkten erzeugt wird. 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 INKEY$-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 INKEY$-Befehl auseinandersetzen.

4.3.1 VERWENDUNG VON EIGENEN INKEY$-ROUTINEN IM MENÜ

In einem früheren Kapitel wurde angesprochen, daß der Nachteil des INKEY$-Befehls darin liegt, daß man nicht erkennen kann, was man eingegeben hat. Im "Rechenlehrgang" (neuerer Teil) wurde ein Weg gefunden, die Eingabe doch sichtbar zu machen. Man gab durch den PRINT-Befehl einfach die mit INKEY$ eingelesenen Werte von A$ aus. Durch das Semikolon wurde erreicht, daß die Ausgabe der einzeln eingegebenen Zeichen hintereinander erfolgen konnte.

Diese Eingaberoutine mit INKEY$ 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 INKEY$-Befehl einzusetzen.

Wir wollen uns nun an die Entwicklung einer eigenen INKEY$-Routine begeben. Diese können Sie nach Fertigstellung in Ihre eigenen Programme als Unterprogramm aufnehmen. Die erste Zeile dürfte Ihnen inzwischen bekannt sein:

```
10 A$=INKEY$: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 A$=INKEY$: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 A$=INKEY$: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 ENTER-Taste an. Welchen ASCII-CODE besitzt die ENTER-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 ENTER-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 ENTER-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 A$=INKEY$: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 A$=INKEY$: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 A$=INKEY$: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 ENTER-Taste geschehen. Außerdem müssen wir daran denken, den Zähler Z nach dem Drücken der ENTER-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 A$=INKEY$: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 INKEY$-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 INKEY$-Routine. Im folgenden soll nun eine Routine, die diese Funktion beinhaltet, aufgelistet werden.

```
10 REM INKEY$-Routine mit Loeschfunktion
20 A$=INKEY$:IF A$="" THEN 20
30 IF ASC(A$) = 13 THEN 130
40 IF ASC(A$) < > 127 THEN 70
50 IF LEN(B$) < 1 THEN 20
55 A$=CHR$(8)+CHR$(16)
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 DEL-Taste betätigt wurde. Der ASCII-Wert dieser Taste ist 127. Wurde die Taste nicht gedrückt, so verzweigt das Programm weiter nach Zeile 70. Haben Sie die 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

```
Improper argument in 60
```

führen. Ist der String also leer, wird die 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 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 DEL-Taste der CHR$(127)-Code. Dieser bewirkt beim Ausdruck mit PRINT ein Grafikzeichen. Wir müssen also eine Lösung finden, durch die beim Ausdruck der Cursor um eine Stelle zurückgesetzt und das Zeichen an dieser Stelle gelöscht wird. Um das zu veranschaulichen, geben Sie einmal folgende Befehlsfolge in den Rechner:

```
PRINT "Schneider CPC 465";CHR$(8);CHR$(16);"4"
```

Drücken Sie nun die ENTER-Taste, so werden Sie feststellen, daß Sie folgenden Ausdruck erhalten:

```
Schneider CPC 464
```

Der CHR$(8)-Code setzt den Cursor um eine Stelle zurück und der CHR$(16)-Code löscht das Zeichen unter der aktuellen Cursorposition. Die "4" tritt sofort an die Stelle des gelöschten Zeichens. Dadurch erhalten Sie dann den oben gezeigten Ausdruck. Nichts anderes geschieht in unserer INKEY$-Routine in der Zeile 110, wenn A$ den CHR$(8)-Code und den CHR$(16)-Code aus Zeile 55 enthält. Das ist schon alles, was zu den neu hinzugekommenen Zeilen zu sagen wäre.

Damit haben wir nun eine INKEY$-Routine aufgebaut, die es uns erlaubt, je nach Manipulation des Wertes Z kürzere oder längere Zeichenketten einlesen zu lassen. Außerdem kann man diese Zeichen sowohl anzeigen lassen als 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 INKEY$-Routine noch so verfeinern, daß wir auch eine Art von Cursor zur Verfügung haben. Als unseren eigenen "Cursor" wollen wir den schmalen Grafikstrich nehmen, der auf der Taste mit der Null liegt. Dieses Zeichen besitzt den CHR$(95)-Code. In diesem Zusammenhang wird wieder der CHR$(8)-Code verwendet, um den realen Bildschirmcursor, der bei der Verwendung von INKEY$ 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$; an der Stelle des aktuellen Bildschirmcursors ausgegeben. Dadurch wird das Grafikzeichen durch das eingegebene Zeichen überschrieben. Gleichzeitig wird mit der Ausgabe des neuen Zeichens unser Cursor (CHR$(95)) 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 CLS
50 LOCATE 5,5
60 PRINT CHR$(95);CHR$(8);
70 REM  ********************
80 REM * Zeichen einlesen *
90 REM ********************
100 A$=INKEY$:IF A$="" THEN 100
110 IF ASC(A$)=13 THEN 240
```

```
120 IF ASC(A$)=32 THEN 180
130 IF ASC(A$) < > 127 THEN 150
140 IF LEN(B$) > = 1 THEN B$=LEFT$(B$,LEN(B$)-1)
:A$=CHR$(16)+CHR$(8)+CHR$(16):GOTO 190
150 IF ASC(A$) < 48 THEN 100
160 IF ASC(A$) > 90 THEN 100
170 IF ASC(A$) > 57 AND ASC(A$) < 65 THEN 100
180 B$=B$+A$
190 PRINT A$;CHR$(95);CHR$(8);
200 GOTO 100
210 REM ******************
220 REM * Ausgabe String *
230 REM ******************
240 CLS
250 LOCATE 10,15
260 PRINT B$
```

In den Zeilen 40 und 50 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 60) und der unsichtbare Bildschirmcursor wird um eine Stelle nach links (CHR$(8)) über unseren Cursor gesetzt. In Zeile 100 beginnt nun die eigentliche Routine. Der Inhalt der Zeile 120 ist neu hinzugekommen. Hier wird abgefragt, ob das eingegebene Zeichen ein Leerzeichen (CHR$(32)) war. Die nächsten Zeilen sind wiederum bekannt. Zeile 160 fragt ab, ob das eingegebene Zeichen einen größeren ASCII-Wert als der Buchstabe "Z" hatte. In Zeile 170 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 100 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 190 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 190 bewirkt dann folgendes:

A* :Nach Ausführung von PRINT A$;

A_* :Nach Ausführung von CHR$(95);

A* :Nach Ausführung von CHR$(8);

Ich hoffe, daß Ihnen durch dieses kleine Schaubild die Wirkung der Zeile 190 deutlich geworden ist.

Die restlichen Programmzeilen dieser Routine sollten Ihnen inzwischen bekannt sein. Damit haben wir uns eine INKEY$-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.

Sie haben jetzt gesehen, wie man solche eigenen Routinen entwickeln kann. Ich hatte bereits in einem früheren Kapitel erwähnt, daß Sie ruhig selbst solche Routinen entwickeln sollen, die in Ihren Programmen dann bestimmte Aufgaben ausführen können. Dabei lernt man sehr gut das Basic eines Computers kennen. Versuchen Sie doch mal, den LOCATE-Befehl innerhalb eines Basicprogramms zu simulieren.

4.3.2 WINDOW-TECHNIKEN

Der CPC 464 besitzt die Möglichkeit, bis zu acht verschiedene ´Bildschirmfenster´ (WINDOWs) zu definieren. Jedes dieser Fenster kann separat für bestimmte Ausgaben angesprochen werden. Damit hat man zusätzliche Möglichkeiten, die Menüs für Programme optisch zu verbessern. So können Sie z.B. Meldungen eines Programms, die den Benutzer zu bestimmten Handlungen aufrufen (Kassette einlegen! o.ä.), in einem extra definierten Fenster ausgeben lassen, ohne das eigentliche Hauptmenü zu überschreiben.

Man könnte z.B. die letzte Bildschirmzeile als ein Fenster definieren. In diesem Fenster werden dann alle Meldungen des Programms ausgegeben. Der Befehl, mit dem man die Fenster definiert, lautet:

```
WINDOW #X,A,B,C,D
```

´X´ bezeichnet hier die Nummer des Fensters. Die Parameter ´A´ und ´B´ bestimmen die linke und rechte Spalte und die Parameter ´C´ und ´D´ die obere und untere Zeile des Fensters.

Wir wollen nun ein Fenster definieren, das nur die untere Bildschirmzeile umfaßt. Dazu geben Sie folgendes in Ihren Rechner:

```
WINDOW #1,1,40,25,25
```

Betätigen Sie jetzt die ENTER-Taste, so geschieht optisch zunächst nichts. Wenn Sie aber folgenden Befehl eingeben,

```
PRINT $1," Das ist die letzte Bildschirmzeile."
```

so wird diese Ausgabe auf das Fenster 1 geleitet. Sie können einfach dem PRINT das Ziffernkreuz und eine Zahl folgen lassen, um die Ausgabe auf ein von Ihnen

definiertes Fenster umzuleiten. Geben Sie jetzt

```
CLS #1
```

ein, so wird nur der Inhalt von Fenster 1 gelöscht. Mit dem Befehl

```
PAPER
```

können Sie die Hintergrundfarbe des Bildschirms verändern. Geben Sie nun die folgenden Befehle in den Rechner:

```
PAPER #1,3:CLS #1
```

Sie sehen, daß die letzte Bildschirmzeile mit der Farbe rot ausgefüllt wurde. Damit kann man die Bildschirmfenster innerhalb eines Menüs optisch hervorheben. Wenn Sie jetzt erneut den PRINT #1-Befehl ausführen, erscheint die Ausgabe in gelber Schrift auf rotem Hintergrund.

Der Bildschirm selbst ist nach dem Einschalten als Fenster 0 anzusehen. Wird der Befehl 'CLS' ausgeführt, wird der gesamte Bildschirm gelöscht, auch der Inhalt von Fenster 1, da es ja innerhalb von Fenster 0 liegt. Wir können das vermeiden, indem wir das Fenster 0 um eine Zeile verkürzen. Mit dem Befehl

```
WINDOW #0,1,40,1,24
```

erhält das Fenster 0 nur noch 24 Bildschirmzeilen zugewiesen. Geben Sie jetzt den Befehl 'CLS' ein, so wird eine Meldung in Fenster 1 nicht überschrieben.

Das Standard-Ausgabefenster ist das Fenster 0. Wollen Sie z.B. Fenster 0 auf das Fenster 1 legen und umgekehrt, so können Sie den Befehl

```
WINDOW SWAP X,Y
```

verwenden. 'X' und 'Y' stehen hier für die Nummern der Fenster, die ausgetauscht werden sollen. Schreiben Sie z.B.

```
WINDOW SWAP 0,1
```

so liegt das Fenster 0 nun in der letzten Bildschirmzeile und das Fenster 1 benutzt jetzt den restlichen Bildschirmbereich.

Mit diesem Wissen können Sie nun die Menüs in den bereits erstellten Programmen nach Ihrem Geschmack ausbauen. Achten Sie aber auch hier darauf, daß Sie den Bildschirmaufbau nicht überladen.

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 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 Menütechniken wäre somit abgeschlossen. In den beiden nächsten Kapiteln werde ich noch kurz auf ein Sortierverfahren und das Prinzip der Dateiverwaltung eingehen.

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. Eine rationelle bzw. schnelle Dateiverwaltung läßt sich eigentlich nur mit einem Diskettenlaufwerk realisieren, da der Zugriff auf die einzelnen Daten wesentlich schneller ausgeführt werden kann. Wir wollen uns hier jedoch auf das Kassettenlaufwerk beschränken, da es wohl das erste externe Speichermedium ist, mit dem Sie in Kontakt treten

werden. Um ein Programm von Kassette in den CPC 464 zu laden, geben Sie folgendes ein:

LOAD "Programmname"

In den Anführungszeichen dürfen maximal 16 Zeichen verwendet werden, so daß Ihre Befehlsfolge jetzt wie folgt aussehen könnte:

LOAD "BEISPIEL"

Der CPC 464 gibt danach die folgende Meldung aus:

Press PLAY then any key:

Sie betätigen dann die PLAY-Taste am Kassettenlaufwerk und danach eine Taste auf der Tastatur. Danach wird das Programm in den Rechner geladen.

Wollen Sie ein Programm abspeichern, so wird dafür der SAVE-Befehl verwendet. Er hat die folgende Schreibweise:

SAVE "BEISPIEL"

Der Rechner gibt danach die Meldung

Press REC and PLAY then any key:

Wenn Sie diese Tasten gedrückt haben, betätigen Sie wiederum zur Bestätigung eine Taste auf der Tastatur. Der Rechner beginnt danach sofort mit der Abspeicherung des Programms oder der Daten.

VORSICHT! Der Rechner merkt nicht, wenn er ein bereits abgespeichertes Programm überschreibt. Vergewissern Sie sich vorher, daß sich auf diesem Teil der Kassette noch keine Daten befinden. Mit dem Befehl

CAT

können Sie sich den Inhalt einer Programmkassette auf dem Bildschirm anzeigen lassen. Anhand des Laufwerkzählers können Sie festhalten, welche Bereiche der Kassette noch nicht beschrieben sind.

Der SAVE-Befehl stellt Ihnen verschiedene Möglichkeiten zur Verfügung, ein Programm abzuspeichern. Der Dateityp folgt dem Dateinamen und wird durch ein Komma getrennt.

```
SAVE "Name",A
```

Mit diesem Befehl speichern Sie ein Programm als ASCII-Datei ab. Wollen Sie ein Programm schützen, so daß es nach dem Laden nicht mehr zu listen ist, müssen Sie das ´A´ gegen ein ´P´ ersetzen. Verwenden Sie ein ´B´ als Kennzeichen, wird das Programm als Binärdatei abgespeichert. Die genaue Anwendung dieser und der noch folgenden Befehle werden im Buch "CPC 464 für Einsteiger" ausführlich besprochen.

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 ein Speichermedium (Kassette oder Diskette) abgespeichert. Innerhalb des Programms hat man dann meistens die Möglichkeit, nach bestimmten Daten suchen zu lassen, diese zu verändern und wieder auf die Kassette oder Diskette zurückzuschreiben.

Wollen Sie Daten an das Kassettenlaufwerk übergeben, müssen Sie zuerst eine Datei eröffnen. Dies geschieht mit dem OPENOUT-Befehl. Die Befehlsfolge

```
OPENOUT "ADRESSEN"
```

öffnet beim Kassettenlaufwerk eine Ausgabedatei, d.h. Sie können nun Daten zum Laufwerk übertragen. Geben Sie vor dem Dateinamen ein Ausrufezeichen ein, so werden die sonst üblichen Meldungen unterdrückt. Sie können nun Daten zum Laufwerk übertragen. Hierzu benötigen Sie noch

den

PRINT#9

Befehl. Dem PRINT# folgt die Zahl 9, die sich auf das Kassettenlaufwerk bezieht. Geben Sie nun den Befehl PRINT#9,D$ ein, so wird der Inhalt der Variablen D$ auf die Kassette geschrieben. Wollen Sie die Datenübertragung beenden, so müssen Sie die Datei mit dem Befehl

CLOSEOUT

wieder schließen.

Damit wissen wir nun, wie eine Datei auf der Kassette 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 OPENOUT-Befehl in:

OPENIN "ADRESSEN"

Damit wird die Datei "ADRESSEN" zum Lesen geöffnet. Zum Einlesen der Daten verwenden wir den

INPUT#9

Befehl. Hier muß ebenfalls wieder die Zahl des angesprochenen Gerätes (9 = Kassettenlaufwerk) mit angegeben werden. Wollen wir in unserem Programm also Daten von der Kassette lesen, müssen wir folgende Befehlsfolge verwenden:

INPUT#9,D$

Mit diesem Befehl können Sie die zuvor abgespeicherten Daten wieder in den Rechner einlesen und einer Variablen zuordnen. Mit dem Befehl

CLOSEIN

wird die zuvor zum Lesen geöffnete Datei wieder geschlossen.

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 individuelle Dateiverwaltung zu schreiben.

```
10 DIM D$(4,2)
20 CLS
30 PRINT"Wollen Sie"
40 PRINT
50 PRINT"Daten eingeben ? (1)"
60 PRINT
70 PRINT"Daten speichern ? (2)"
80 PRINT
90 PRINT"Daten laden ? (3)"
100 PRINT
110 PRINT"Daten ausgeben ? (4)"
120 PRINT
130 PRINT"Beenden ? (5)"
140 A$=INKEY$:IF A$="" THEN 140
150 ON VAL(A$) GOTO 190,290,400,500,600
160 REM ******************
170 REM * Daten eingeben *
180 REM ******************
```

```
190 CLS
200 FOR I=1 TO 4
210 INPUT "Name";D$(I,1)
220 INPUT"Vorname";D$(I,2)
230 CLS
240 NEXT I
250 GOTO 20
260 REM ********************
270 REM * Daten speichern *
280 REM ********************
290 REM
300 OPENOUT "Adressen"
310 FOR I=1 TO 4
320 FOR Z=1 TO 2
330 PRINT#9,D$(I,Z)
340 NEXT Z,I
350 CLOSEOUT
360 GOTO 20
370 REM ***************
380 REM * Daten laden *
390 REM ***************
400 OPENIN "ADRESSEN"
410 FOR I=1 TO 4
420 FOR Z=1 TO 2
430 INPUT#9,D$(I,Z)
440 NEXT Z,I
450 CLOSEIN
460 GOTO 20
470 REM ********************
480 REM * Daten ausgeben *
490 REM ********************
500 CLS
510 FOR I=1 TO 4
520 FOR Z=1 TO 2
530 PRINT D$(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 CLS
610 END
```

Zum Schluß dieses Kapitels seien noch kurz die restlichen vier Befehle erwähnt, die Ihnen ebenfalls den Umgang mit Dateien und dem Kassettenlaufwerk erleichtern werden. Der Befehl

CHAIN

lädt ein Programm in den Rechner, wobei das augenblicklich im Speicher befindliche Programm gelöscht wird. Mit den Befehlen

CHAIN MERGE

und

MERGE

haben Sie die Möglichkeit, Programme miteinander zu verknüpfen. Sie müssen aber darauf achten, daß die Zeilennummern der Programme unterschiedlich sind. Das Programm im Speicher muß also andere Zeilennummern aufweisen als das Programm, das Sie nachladen wollen. Da Ihnen der CPC 464 diese Befehle zur Verfügung stellt, lohnt es sich, eine Programmbibliothek aufzubauen. Sie schreiben bestimmte Programmteile einmal und speichern diese dann ab. Benötigen Sie dann bei der Erstellung eines Programms einen dieser Teile, so laden Sie ihn einfach unter Benutzung der o.a. Befehle in das bestehende Programm. Komfortabler kann man sich das Programmieren schon kaum noch vorstellen.

Mit der Funktion

EOF

können Sie innerhalb eines Programms feststellen, ob das Ende einer Datei (EOF = End Of File) schon erreicht wurde. Bei Dateiende wird der Wert -1 zurückgegeben, ansonsten der Wert 0.

Damit schließen wir das Kapitel über die Dateiverwaltung ab. Im letzten Kapitel sollen noch kurz die Befehle für den Grafik- und Musikbereich des CPC 464 besprochen werden.

5. MUSIK UND GRAFIK

In diesem Kapitel sollen die Befehle des Schneider-Basic, die sich mit den Bereichen Musik und Grafik befassen, kurz erklärt werden. Da zu diesem Thema bereits spezielle Literatur vorhanden ist, und eine ausführliche Beschreibung den Rahmen dieses Buches sprengen würde, wird sich nur auf das Wesentliche dieser Befehle beschränkt.

5.1 MUSIK

Im folgenden sollen nun zuerst die einzelnen Befehle für den Bereich 'Musik' erklärt werden. Der wichtigste Befehl in dieser Hinsicht ist der

SOUND

Befehl. Dem SOUND-Befehl können bis zu 7 Parameter übergeben werden, so daß die Schreibweise wie folgt aussieht:

```
SOUND A,B,C,D,E,F,G
```

Dabei bedeuten die Parameter im einzelnen

A - Kanalstatus (Wertebereich 1-255)
B - Periode, beeinflußt Frequenz (Werte 0-4095)
    Frequenz = 125000/Periode
C - Tondauer (Wertebereich -32768 bis +32767)
    Die Einheit beträgt hier 1/100 Sekunde
D - Lautstärke (0 bis 15)
E - Hüllkurve-Lautstärke (0 bis 15)
F - Hüllkurve-Ton (0 bis 15)
G - Geräuschcharakter (0 bis 15)

```
SOUND 1,284,100,12,0,0,0
```

Dieser Befehl erzeugt einen Ton auf Kanal 1 mit der Frequenz 440 Hertz, einer Dauer von einer Sekunde und der Lautstärke 12. Geben Sie dem letzten Parameter einen Wert zwischen 1 und 15, so können Sie diesen Ton noch mit einem Rauschen überlagern.
Die Parameter ´E´ und ´F´ beziehen sich auf Hüllkurven, die noch zu definieren sind. Der Standardwert ist hier jeweils 0. Der Befehl zur Definition der Hüllkurve für die Lautstärke lautet:

```
ENV
```

Mit diesem Befehl bestimmen Sie z.B., in welchem Bereich die Lautstärke an- und abschwellen soll. Der Befehl

```
ENT
```

erlaubt Ihnen, den Ton selbst zu beeinflussen. ENT erzeugt kleine Schwankungen in der Frequenz eines Tons.
Eine Funktion, die schon bei den Unterprogrammen erwähnt wurde, steht im direkten Zusammenhang mit den Kanälen für den Sound. Mit

```
PRINT SQ(X)
```

können Sie den Kanalstatus abfragen. ´X´ darf die Werte 1, 2 oder 4 annehmen. Entsprechend dem Wert von SQ kann dann in Unterprogramme verzweigt werden. Wird der Befehl

```
RELEASE X
```

benutzt, so werden die Haltepunkte, die durch den SOUND-Befehl bestimmt werden können, in den Kanälen aufgehoben. ´X´ kann hier Werte zwischen 0 und 7 annehmen.

Das soll uns an Informationen zum Kapitel Musik genügen. Weiterführende Informationen über die vielfältigen

Möglichkeiten der Musikprogrammierung des CPC 464 finden Sie in Ihrem Handbuch oder in anderen DATA BECKER Büchern über den CPC 464.

5.2 GRAFIK

Der CPC 464 kann in drei verschiedene Grafikarten umschalten. Nach dem Einschalten des CPC befindet sich dieser in MODE 1, was einer grafischen Auflösung von 320 x 200 Bildpunkten entspricht. In diesem Modus sind vier Farben verfügbar. Die Anzahl der Zeichen pro Bildschirmzeile beträgt 40. Mit dem Befehl

```
MODE X
```

kann zwischen den einzelnen Grafikarten hin- und hergeschaltet werden. Die einzelnen Modi sind

```
MODE 0

MODE 1

MODE 2
```

In MODE 0 verringert sich die grafische Auflösung auf 160 x 200 Punkte. In diesem Modus sind auch nur 20 Zeichen pro Zeile darstellbar. Dafür stehen dem Anwender 16 Farben zur Verfügung. Insgesamt bietet der CPC 464 eine Auswahl von 27 verschiedenen Farben an. MODE 2 besitzt die höchste grafische Auflösung von 640 x 200 Punkten. Hier können Sie nur auf zwei Farben zurückgreifen. Dafür können Sie 80 Zeichen in einer Zeile darstellen.

Die folgende Tabelle soll das noch einmal verdeutlichen.

	AUFLÖSUNG	FARBEN	ZEICHEN / ZEILE
MODE 0	160 * 200	16	20
MODE 1	320 * 200	4	40
MODE 2	640 * 200	2	80

Kommen wir nun zu den Befehlen, die die Programmierung der Grafik unterstützen. Der Befehl

BORDER X

verändert die Farbe des Bildschirmrandes. 'X' kann Werte zwischen 0 und 31 annehmen. Jedoch sind nur Werte zwischen 0 und 26 sinnvoll, da diese die entsprechenden Farben in der Tabelle des Handbuchs repräsentieren. Sie können hier auch zwei Parameter angeben, was zur Folge hat, daß zwischen diesen beiden Farben hin- und hergeschaltet wird. Die Zeitdauer zwischen den Farbwechseln kann mit dem Befehl

SPEED INK X,Y

bestimmt werden. Die Zeiteinheiten werden in 0.02 Sekunden gemessen.

Mit dem Befehl

CLG (Farbstiftnr.)

können Sie den Grafikschirm mit der Farbe ausfüllen, die dem angegebenen Farbstift zugeordnet ist. Geben Sie nach

dem Einschalten

```
CLG 3
```

ein, erhält der gesamte Grafikschirm die Farbe ´rot´. Geben Sie keine Farbstiftnummer an, so wird der Farbstift verwendet, der beim letzten CLG-Befehl benutzt wurde.

Der Grafikschirm ist in 640 x 400 Positionen unterteilt. Wollen wir nun einen einzelnen Punkt auf diesem Schirm ansprechen, so benötigen wir dazu die X- und die Y-Koordinate. Die X-Werte liegen zwischen 0 und 639 und die Y-Werte zwischen 0 und 399. Der Nullpunkt (0,0) des Bildschirms liegt in der linken unteren Ecke. Die Werte nehmen also von links nach rechts und von unten nach oben zu. Das ist bei der Angabe der Parameter zu berücksichtigen.
Auf dem Grafikschirm existiert ebenfalls ein Cursor, der sogenannte Grafikcursor. Dieser ist jedoch nicht sichtbar. Mit dem Befehl

```
DRAW X,Y
```

wird eine Linie von der aktuellen Cursorposition zur angegebenen X,Y-Position gezeichnet. Geben Sie also

```
DRAW 100,100
```

ein, wird vom Punkt 0,0 zum Punkt 100,100 eine Linie gezeichnet. Wiederholen Sie diesen Befehl, so nehmen Sie keine Veänderung war. Das wäre beim folgenden Befehl nicht geschehen. Mit

```
DRAWR X,Y
```

wird immer zur angegebenen ´relativen´ X,Y-Position gezeichnet. Wiederholen Sie diesen Befehl, wird erneut ab der Cursorposition eine Linie gezeichnet. Nach der zweiten Ausführung dieses Befehls wäre die Position des Grafikcursors also 200,200.

In der Anwendung ähnlich sind die zwei Befehle

PLOT X,Y

und

PLOTR X,Y

Der Unterschied zu ´DRAW´ liegt darin, daß diese Befehle keine Linien zeichnen, sondern an der angegebenen X,Y-Position einen Punkt setzen. Wollten Sie also mit diesen Befehlen eine Linie ziehen, müßten Sie dieses in einer Schleife realisieren. Das folgende Beispiel ´plottet´ eine Linie zur Position 100,100.

```
MOVE 0,0:FOR I=1 TO 100:PLOT I,I:NEXT
```

In diesem Beispiel wurde der Befehl MOVE verwendet. Dieser Befehl bewegt den Grafikcursor unsichtbar an die bezeichnete X,Y-Position. Auch hier existiert der Befehl ´MOVER X,Y´, der sich auf die relativen X,Y-Positionen bezieht.

Mit den Befehlen

XPOS

und

YPOS

haben Sie die Möglichkeit, jederzeit die Position des Grafikcursors abzufragen. Sie können diese Werte mit PRINT ausgeben lassen oder einer Variablen zuordnen.

Mit dem Befehl

ORIGIN X,Y

haben Sie die Möglichkeit, den Koordinatenursprung zu verlegen. Geben Sie

```
ORIGIN 320,200
```

ein, so verlegen Sie den Ursprung in die Mitte des Bildschirms.

Wenn Sie Ihre Grafik mit Text versehen wollen, so können Sie mit dem Befehl

```
TAG
```

die Ausgabe mit PRINT auf die Position des Grafikcursors umleiten. Der Befehl

```
TAGOFF
```

schaltet die Ausgabe wieder auf den normalen Textcursor um.

Wie bereits erwähnt, stehen Ihnen insgesamt 27 Farben zur Verfügung. Je nach Modus können Sie davon 16, 4 oder 2 Farben benutzen. Mit dem Befehl

```
INK
```

bestimmen Sie, welche Farben benutzt werden sollen. Z.B. ordnet

```
INK 1,3
```

die Farbe ´rot´ dem Register 1 zu. Wählen Sie jetzt mit

```
PEN 1
```

den Farbstift Nummer 1 aus, ändern Sie die Schriftfarbe in rot. Mit PEN bestimmen Sie also, aus welchem der 16 Register die Farbe für die Schrift genommen werden soll.

Auf die gleiche Art und Weise bestimmen Sie die Farbe des Hintergrundes mit dem

PAPER

Befehl.

Wollen Sie nun wissen, mit welchem Farbstift an der Position 250,150 gezeichnet wurde, geben Sie einfach

PRINT TEST(250,150)

ein. Sie erhalten als Ausgabe die verwendete Farbstiftnummer. Der Befehl

TESTR (X,Y)

überprüft die Farbstiftnummer an der relativen X,Y-Position. Steht also der Grafikcursor auf Position 100,240 und geben Sie

PRINT TESTR(50,30)

ein, wird die Position 150,270 überprüft.

ASCII-CODES

Dez.	Hex.	
0	0	keine Wirkung
1	1	Ausgabe eines Symbols. Wird durch zweiten Parameter bestimmt.
2	2	Textcursor ausschalten.
3	3	Textcursor einschalten.
4	4	Durch zweite Parameterangabe gleiche Funktion wie MODE.
5	5	Ausgabe eines Symbols auf die Position des Grafikcursors.
6	6	Textbildschirm aktiv
7	7	Klingel
8	8	Cursor eine Stelle zurück.
9	9	Cursor eine Stelle vor.
10	0A	Cursor eine Zeile nach unten.
11	0B	Cursor eine Zeile nach oben.
12	0C	wie CLS
13	0D	CARRIAGE RETURN (ENTER)
14	0E	Durch Angabe eines zweiten Parameter gleiche Funktion wie PAPER.
15	0F	Durch Angabe eines zweiten Parameters gleiche Funktion wie PEN.
16	10	Zeichen unter Cursor löschen.
17	11	Zeile bis Cursor löschen.
18	12	Zeile ab Cursor löschen.
19	13	Bildschirm bis Cursor löschen.
20	14	Bildschirm ab Cursor löschen.
21	15	Textbildschirm abschalten.
22	16	Transparent ein/aus (1/0).
23	17	Farbstfitmodus für Grafik setzen.
24	18	Revers ein/aus.

Dez.	Hex.	
25	19	Durch nachfolgende 9 Parameter gleiche Funktion wie SYMBOL.
26	1A	Gleiche Funktion wie WINDOW durch Angabe entsprechender Parameter.
27	1B	Keine Wirkung.
28	1C	Gleiche Funktion wie INK.
29	1D	Gleiche Funktion wie BORDER.
30	1E	Cursor in die linke obere Bildschirmecke setzen.
31	1F	Gleiche Funktion wie LOCATE.
32	20	Leerzeichen
33	21	!
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	;

Dez.	Hex.	
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
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

Dez.	Hex.	
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
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	▒

RESERVIERTE WÖRTER

ABS, AFTER, AND, ASC, ATN, AUTO

BIN$, BORDER

CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN, CLOSEOUT, CLS, CONT, COS, CREAL

DATA DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE, DI, DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR, EVERY, EXP

FIX, FN, FOR, FRE

GOSUB, GOTO

HEX$, HIMEM

IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT

JOY

KEY

LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWER$

MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT

ON, ON BREAK, ON ERROR GOTO, ON SQ, OPENIN, OPENOUT, OR,

ORIGIN, OUT

PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT

RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN

SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR, STEP, STOP, STR$, STRING$, SWAP, SYMBOL

TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON

UNT, UPPER$, USING

VAL, VPOS

WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE

XOR, XPOS

YPOS

ZONE

TASTATUR-CODE

Taste	Code
LEERTASTE	47
ESC	66
TAB	68
CAPS LOCK	70
SHIFT	21
DEL	79
ENTER	18
CTRL	23
CLR	16
↑	24
1	64
2	65
3	57
4	56
5	49
6	48
7	41
8	40
9	33
0	32
-	25
A	69
B	54
C	62
D	61
E	58
F	53
G	52
H	44
I	35
J	45
K	37

Taste	Code
L	36
M	38
N	46
O	34
P	27
Q	67
R	50
S	60
T	51
U	42
V	55
W	59
X	63
Y	43
Z	71
'	39
.	31
/	30
\	22
:	29
;	28
[	17
]	19
@	26

ANHANG 4

SACHREGISTER