Kampow

COMMODORE 128

Das große BASIC-Buch

EIN DATA BECKER BUCH

Kampow

COMMODORE 128

Das große BASIC-Buch

EIN DATA BECKER BUCH

ISBN 3-89011-114-9

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

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

Inhaltsverzeichnis

Einleitung **11**

1. Grundlagen des Programmierens **17**
1.1 Algorithmus und Programm 17
1.2 Die Computersprache BASIC 17
1.3 Datenfluß- und Programmablaufpläne 20
1.3.1 Datenflußpläne 22
1.3.2 Programmablaufpläne 25
1.4 ASCII-Codes 31
1.5 Zahlensysteme 32
1.5.1 Das Dualsystem 33
1.5.2 Bit und Byte 34
1.5.3 Das Hexadezimalsystem 36
1.6 Die logischen Operatoren 39
1.6.1 NOT 41
1.6.2 AND 42
1.6.3 OR 43
1.6.4 XOR 45

Aufgaben 47

2. Einführung in das Programmieren mit BASIC **50**
2.1 Das erste BASIC-Programm 50
2.1.1 Eingabe von Werten mit INPUT 54
2.1.2 Wertzuweisung mit LET 55
2.1.3 Ausgabe mit PRINT 56
2.1.3.1 PRINT USING 61
2.1.3.2 PUDEF beeinflußt PRINT USING 67
2.1.4 Kommentare mit REM 69
2.2 Variablen und deren Verwendung 70
2.2.1 Rechenoperationen mit Variablen 72

Aufgaben 74

2.3 Numerische Funktionen 75
2.3.1 Funktionen mit DEF FN 80
2.3.2 Zufallszahlen 81

2.3.3 Noch mehr Befehle für Variablen 82
2.3.4 ASC(X$) und CHR$(X) 85

Aufgaben 89

2.4 TAB(und SPC(..... 90
2.5 Strings 91
2.5.1 LEFT$ 93
2.5.2 RIGHT$ 94
2.5.3 MID$ 95
2.5.4 LEN(X$) 97
2.5.5 VAL(X$) 97
2.5.6 STR$(X) 99
2.5.7 INSTR 100
2.5.8 TI$ 101

Aufgaben 103

2.6 Editieren von Programmen 105

3. Erweiterte Programmstrukturen 112
3.1 Unbedingte Programmsprünge 112
3.2 Bedingte Programmsprünge 116
3.2.1 IF...THEN...ELSE 116
3.2.2 BEGIN...BEND 121

Aufgaben 123

3.2.3 FOR...TO...NEXT 124
3.2.4 Schleifen mit DO...LOOP 132
3.2.4.1 DO...LOOP mit UNTIL und WHILE 133
3.3 Berechnete Sprungbefehle 139
3.3.1 Beispielprogramm "Rechenlehrgang" 143
3.3.2 Programmsprünge mit TRAP 153

Aufgaben 158

3.4 Programmablaufsteuerung mit GET 159
3.4.1 Eingabe von Daten mit GET 159
3.4.2 Tastaturabfrage mit GETKEY 163
3.4.3 Die Belegung der Funktionstasten 167
3.4.4 Funktionstastenabfrage mit GET 168
3.5 FRE, POS, SYS, USR(X), WAIT 171
3.6 PEEK und POKE 174
3.7 READ, DATA und RESTORE 177

4. Komplexere BASIC-Anwendungen 188
4.1 Felder 188
4.1.1 Eindimensionale Felder 188
4.1.2 Beispiele zu eindimensionalen Feldern 197

Aufgaben 205

4.1.3 Mehrdimensionale Felder 206
4.2 Unterprogramme 216
4.3 Menütechniken 238
4.3.1 Verwendung von GET-Routinen im Menü 244
4.3.2 Cursorpositionierung mit CHAR 254
4.3.3 Cursorsteuerung mit CHR$-Codes 255
4.4 WINDOW-Techniken 265
4.5 Sortierverfahren 269

5. Das Prinzip der Dateiverwaltung 274
5.1 Allgemeines zur Datenspeicherung 274
5.2 Verschiedene Dateitypen 275
5.3 Die Datei 276
5.4 Relative Dateiverwaltung 282

6. Musik und Grafik 286
6.1 Musik 286
6.2 Grafik 291
6.2.1 Analoguhr 295

7. BASIC Intern 300
7.1 Der Monitor 300
7.2 Der Variablenzeiger 303

8. Utilities **306**
8.1 Hardcopy Text 306
8.2 Binärumwandlung 307
8.3 Ausgabe mit führenden Nullen 307
8.4 Oszillierendes Flag 308
8.5 Programmlisting auf Diskette 308
8.6 Auslesen einer sequentiellen Datei 308

9. Lösungen **310**

Anhang

A Befehlsübersicht 328
B Reservierte Wörter 426
C Die Tokentabelle 430
D Dezimal-, Hexadezimal- und Binärtabelle 433
E Fehlermeldungen 437
F Die Zeichensätze 439
G Stichwortregister 452

Einleitung

Dieses Buch wendet sich an all diejenigen, die sich - vor kurzem vielleicht - einen Commodore 128 angeschaft 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 Commodore 128 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 bekommen Sie die wichtigsten theoretischen und praktischen Grundlagen der Datenverarbeitung vermittelt.

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 oder einen ähnlichen 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 finden Sie in einem extra Kapitel am Ende des Buches. Sie sind eingehend erklärt. Arbeiten Sie die Lösungen zu den Aufgaben zu-

erst durch, da bei den Lösungsvorschlägen Überleitungen zum jeweils nächsten Kapitel vorhanden sind.

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 nicht nur lesen, sondern auch den Umgang mit BASIC lernen.

Der 5. Teil führt Sie kurz in das Prinzip der Dateiverwaltung ein.

Der 6. Teil erklärt anhand von Beispielen (z.B. einer Analoguhr) einige Grafik- und Musikbefehle des Commodore 128.

Der 7. Teil befaßt sich mit BASIC-Intern. Hier lernen Sie, wie BASIC-Befehle im Speicher abgelegt werden, wie Variablen gespeichert werden usw. In diesem Kapitel werden die Grundlagen vermittelt, um z.B. den Einstieg in Bücher wie *"128 Intern"* zu erleichtern.

Der 8. Teil stellt Ihnen einige nützliche BASIC-Routinen (Utilities) zur Verfügung, die Sie in Ihren eigenen Programmen verwenden können.

Der 9. Teil schließlich ist der Lösungsteil, in dem die Lösungen ausführlich besprochen werden.

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".

Anmerkung:

Aus drucktechnischen Gründen wird in diesem Buch das "π"-Zeichen so dargestellt:

π = PI

Finden Sie dieses Zeichen also in den Programmen vor, so müssen Sie jeweils die entsprechende Taste betätigen.

1

GRUNDLAGEN DES PROGRAMMIERENS

1. Grundlagen des Programmierens

1.1 Algorithmus und Programm

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**.

1.2 Die Computersprache BASIC

Die Sprache ist im Falle des Commodore 128 BASIC. BASIC wurde im Jahre 1961 am Darthmouth College in New Hampshire (USA) entwickelt und setzt sich aus den Anfangsbuchstaben von

*B*eginner's *A*ll purpose *S*ymbolic *I*nstruction *C*ode

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 Commodore 128 nicht direkt auf andere Computer anwendbar ist. So besitzt z.B. der Commodore 64 weniger leistungsfähige Befehle als der Commodore 128. Auf dem Commodore 128 erstellte Programme müssen also im Hinblick auf die BASIC-Versionen anderer Computer entsprechend angepaßt werden. Das BASIC 7.0 des Commodore 128 beinhaltet sowohl die Befehle des BASIC 2.0 (CBM 64) als auch die Befehle des BASIC 4.0 (8000er Serie etc.) plus einer Menge an zusätzlichen Befehlen, die die Programmierung der Grafik und des Sounds erleichtern.

Der Computer versteht nun allerdings die einzelnen BASIC-Befehle nicht direkt. Diese müssen erst in einen entsprechenden Code, die sogenannte Maschinensprache, übersetzt werden, mit dem der Computer dann arbeiten kann. Diese Übersetzung der BASIC-Befehle übernimmt der **BASIC-Interpreter** im Computer. Geben Sie nun einen BASIC-Befehl über die Tastatur in den Computer ein und drücken die RETURN-Taste, so wird dieser Befehl erst über den **Interpreter** geleitet, dort in den computereigenen Code umgewandelt und dann erst ausgeführt.

Zusammenfassend kann man also sagen, daß man unter Programmieren die Übersetzung eines **Algorithmus** in eine Programmiersprache, in unserem Falle BASIC, versteht.

Nun wird aber von Anfängern, jedoch auch von vielen Fortgeschrittenen, meistens in der folgenden Art und Weise vorgegangen:
Herr Müller möchte sich zum Beispiel bei vorgegebenem Radius den Rauminhalt einer Kugel für 20 verschiedene Radien berechnen lassen. Die Formel wird ruckzuck aus der Formelsammlung abgelesen, demnach ist das Volumen einer Kugel $V=4PIr^3/3$ (PI=PI-Zeichen auf der Tastatur), und in den Com-

puter 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 fragen: "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 der einzelnen Programmschritte unterlaufen uns immer wieder Fehler. Der Weg von der Aufgabenstellung bis zum fertigen Programm ist also 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.

1.3 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

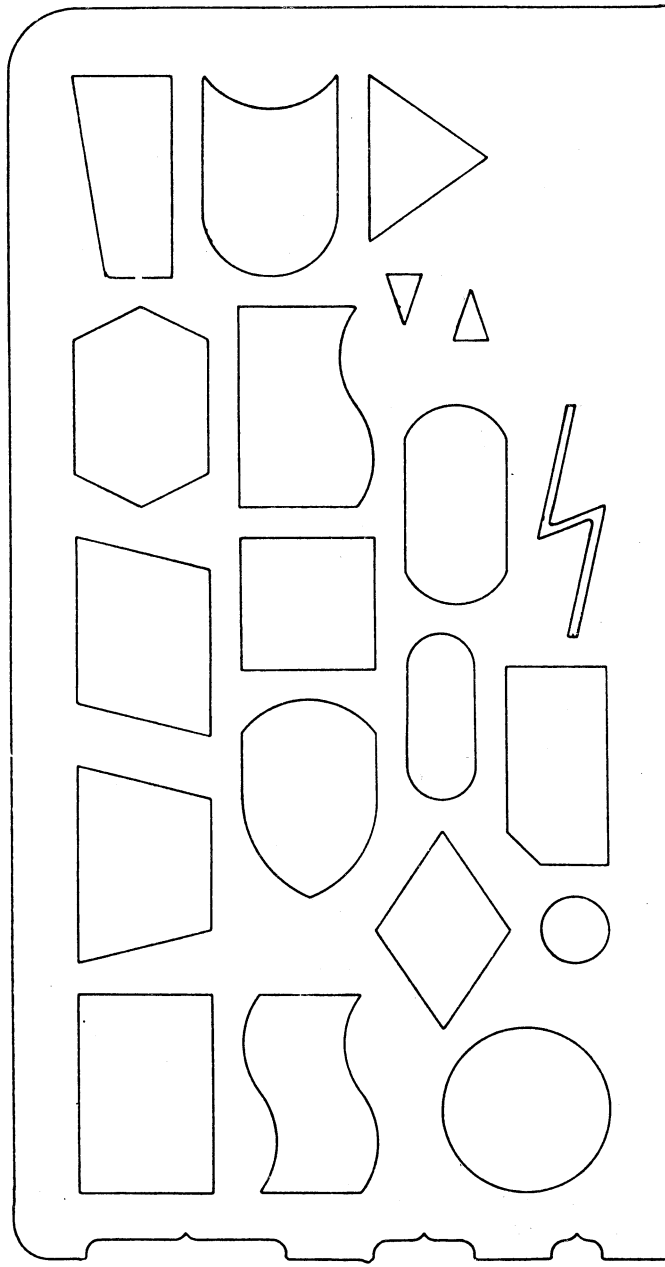


Bild 1

1.3.1 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.

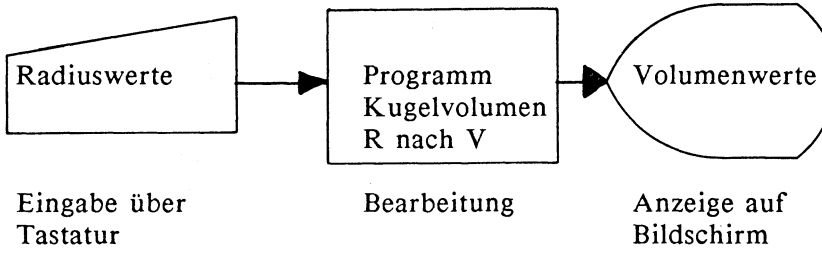


Bild 2

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 selbst. Haben Sie sich also rechtzeitig an die Erstellung solcher Datenflußpläne gewöhnt, so fällt es Ihnen umso leichter, sie auf größere Programme anzuwenden.

Die Bedeutung der einzelnen Symbole für die Datenflußpläne entnehmen Sie bitte dem Bild 3.

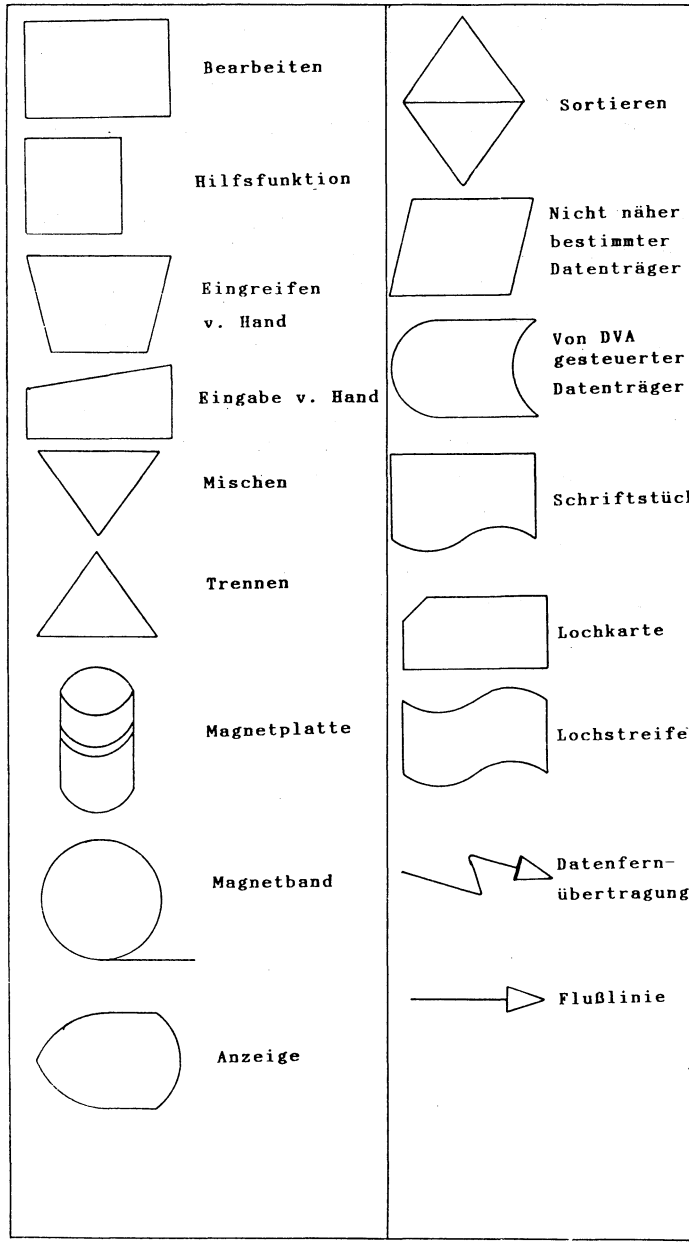

Bearbeiten
Hilfsfunktion
Eingreifen v. Hand
Eingabe v. Hand
Mischen
Trennen
Magnetplatte
Magnetband
Anzeige
Sortieren
Nicht näher bestimmter Datenträger
Von DVA gesteuerter Datenträger
Schriftstück
Lochkarte
Lochstreifen
Datenfern-übertragung
Flußlinie

Bild 3

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

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

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

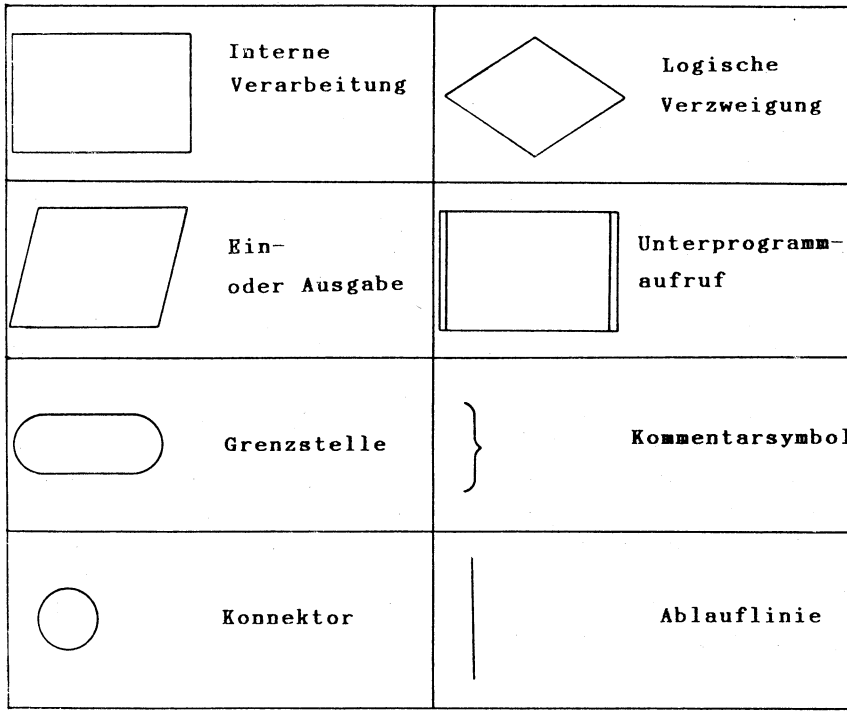


Bild 4

1.3.2 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 Programmierscha-

blone 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.

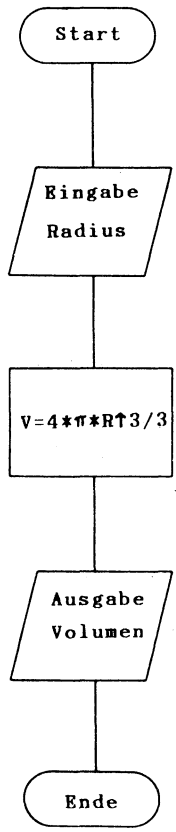


Bild 5

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 kleinen Beispielprogramm wurde hier keine FOR-NEXT-Schleife (vgl. hierzu entsprechendes Kapitel) 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 PAPs. Fällt alles zu Ihrer Zufriedenheit aus, können Sie daran gehen, das Programm mit RUN zu starten.

Versuchen Sie jetzt einmal selbst 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. Soll jedoch nach einiger Zeit einmal eine Programmänderung vorgenommen werden, kann es passieren, daß ein Programmierer sein eigenes Programm nicht mehr versteht (dies ist tatsächlich schon vorgekommen). Das liegt ganz einfach daran, daß sich kaum jemand an Kleinigkeiten erinnern kann, die er vielleicht vor einem Jahr in seinem Programm untergebracht hat. Deshalb sollte man es sich angewöhnen, zu seinem Programm eine Dokumentation zu erstellen. Diese sollte so gehalten sein, daß man das Programm auch noch nach mehreren Monaten versteht.

Soweit die Kapitel zu Datenfluß- und Programmablaufplänen. Wollen Sie sich mehr mit dieser Materie befassen, so sei hier auf die entsprechende Fachliteratur verwiesen.

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

Lösungsvorschlag

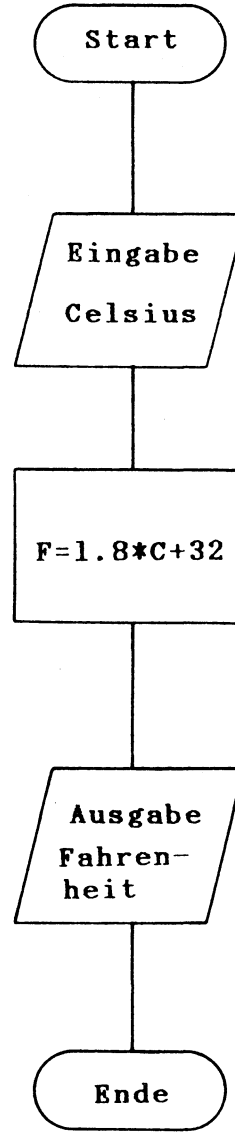


Bild 7

1.4 ASCII-Codes

Der COMMODORE 128 kann, wie schon erwähnt, 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 "*A*merican *S*tandard *C*ode for *I*nformation *I*nterchange", was soviel heißt wie "*Amerikanischer Standardcode für den Informationsaustausch*". Er wurde entwickelt, um einen Datenaustausch auch zwischen verschiedenen Informationsträgern zu gewährleisten, d.h. daß z.B. das Zeichen "A" im **ASCII-Code** immer den **Wert 65** hat. Wird nun diese Zahl an einen Computer oder Drucker gesendet, der ebenfalls mit dem ASCII-Code arbeitet, wird dieser Wert immer als das Zeichen "A" interpretiert. Dabei hat die Entfernung zwischen Sender und Empfänger keinerlei Bedeutung. Ob Sie nun über die Tastatur Zeichen in den Computer eingeben - diese werden ja ebenfalls über eine Leitung an den Rechner weitergeleitet - oder ob Sie über ein Telefonmodem Ihre Daten, z.B. nach Amerika, übertragen; sobald der Empfänger den Wert 65 erhält, wird dieser in ein "A" übersetzt. Der Standard-ASCII-Code benutzt die Werte von 0 bis 127.

Die meisten Computerhersteller haben sich allerdings für einen erweiterten ASCII-Code entschlossen, um auch Zeichen nach eigenem Belieben darstellen zu können. Dieser Code wird auch ASCII-Code genannt, obwohl er mit dem Standard-ASCII-Code nicht in allen Werten übereinstimmt.

Beim Standard-ASCII-Code werden die Zahlen 32-90 für Großbuchstaben und die Zahlen 91-127 für Kleinbuchstaben und einige andere Zeichen verwendet. Der ASCII-Code des Commodore 128 ist nur im Bereich zwischen den Zahlen 32-91 und noch mit wenigen anderen Zahlen mit dem Standard-ASCII-Code identisch. Genaueres entnehmen Sie bitte den Tabellen im Anhang.

1.5 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 **Ziffern 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 eine Zahl eines beliebigen anderen Zahlensystems umwandeln. So können wir 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.

$$(5678_{10})$$

1.5.1 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:

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

oder

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

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.

1.5.2 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** (*B*inary Dig*IT*). Ein Bit kann **zwei Zustände** oder Werte haben:

0 oder 1.

Man spricht auch von einem **gesetzten Bit** beim **Wert** von **1**, oder von einem **nicht gesetzten Bit** beim **Wert** von **0**. Der

Commodore 128 besitzt einen *8-Bit Prozessor*, d.h. daß er in einer Speicherstelle maximal einen dezimalen Wert von 255 ablegen kann. Sie sehen jetzt, warum im obigen Beispiel mit 8 Ziffern gearbeitet wurde. Jede Ziffer entspricht einem Bit. Alle **acht Bits** zusammengefaßt nennt man **Byte**. Sind nun alle acht Bits "gesetzt", so erhält man einen dezimalen Wert von 255. Das sind insgesamt 256 mögliche Werte, nämlich 0-255. Der Commodore 128 kann aber insgesamt 65535 Speicherstellen adressieren (ansprechen). 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** oder auch Sedezimalsystem.

1.5.3 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 9 darstellen sollen, unterscheiden zu können, bedient man sich der Buchstaben **A-F**. Damit sieht dann die dezimale Ziffernfolge

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 usw.

in hexadezimaler Schreibweise wie folgt aus:

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 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**.

$$2E0C_{16}$$

$$= 2*16^3 + 14*16^2 + 0*16^1 + 12*16^0$$

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

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:

$$0ABC_{16}$$

$$= 0*16^3 + 10*16^2 + 11*16^1 + 12*16^0$$

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

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 0ABC.

Der erste Teil (die ersten zwei Zeichen) ist unser High-Byte. Dieses darf ja maximal den Wert 255 annehmen, welches in hexadezimaler Schreibweise FF entspricht.

0A 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 **Nybble** oder **Nibble** (beide Schreibweisen sind gebräuchlich). Einige Personen (z.B. Lehrer) hören aber lieber den Begriff **Tetrade**.

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:

$C001_{16}$

1.6 Die logischen Operatoren

Die **logischen Operatoren**, oder auch **Boolesche Operatoren** nach *G. Boole*[1] benannt (*sprich: Bulsche Operatoren*), werden Ihnen in fast jedem Programm einmal begegnen. Mit diesen Operatoren werden Vergleiche und Bitmanipulationen erst möglich. Das BASIC 7.0 des Commodore 128 stellt Ihnen die **drei Grundverknüpfungen**

NOT, AND und OR

zur Verfügung. Diese drei Operatoren reichen aus, um auch die kompliziertesten logischen Verknüpfungen zu realisieren. So ist auch die Funktion XOR nur eine Kombination aus diesen drei

1 George Boole engl. Mathematiker - geb. Lincoln 12.11.1815, gest. Cork (Irland) 8.12.1864 - G.Boole gilt als der bedeutendste Urheber der mathematischen Logik.

Operatoren, wie wir später noch sehen werden. In der Digitaltechnik finden wir diese drei Operatoren in den verschiedensten Kombinationen in den ICs wieder (z.B. NAND-, NOR- und EXOR-Gatter).

Wie wir bereits wissen, kann der Computer nur zwei Zustände unterscheiden, nämlich **An** und **Aus**. Dadurch bedingt kennt der Computer auch nur eine **zweiwertige Aussagenlogik**. Er kann nur entscheiden ob eine Aussage **wahr** oder **falsch** ist. Eine Aussage wäre z.B.:

2 < 3 (*zwei kleiner drei*)

Bei dieser Aussage handelt es sich um eine **wahre Aussage**. Der Computer teilt uns diese Entscheidung allerdings nicht durch die Ausgabe "Wahr" oder "Falsch" mit, sondern er zeigt uns dies durch entsprechende Zahlenwerte an. Ist eine Aussage **'Wahr'**, wie im obigen Beispiel, so gibt uns der Computer einen von **Null verschiedenen Wert** aus. Geben Sie folgende Befehlsfolge in den Rechner:

PRINT 2 < 3 *(RETURN)*

Ausgabe:

-1

Der Wert ist von **Null verschieden**, also handelt es sich für den Computer um eine **wahre** Aussage. In den meisten Fällen wird eine wahre Aussage als Ergebnis den Wert -1 erhalten. Erzeugen wir nun eine **falsche** Aussage:

PRINT 3 < 2

Ausgabe:

0

Der Wert ist **gleich Null**. Damit zeigt der Computer an, daß es sich um eine **falsche** Aussage handelt. Eine **falsche Aussage** hat grundsätzlich den Wert und **nur den Wert Null** als Ergebnis.

Die drei logischen Operatoren verknüpfen nun zwei Werte miteinander, indem sie diese bitweise betrachten. Besprechen wir die Operatoren nun im einzelnen.

1.6.1 NOT

Der Operator NOT hat zur Folge, daß aus einer wahren Aussage eine falsche und aus einer falschen eine wahre Antwort wird. Die folgende Übersicht soll das verdeutlichen.

Operator	Wert 1	Wert 2	Ergebnis
NOT	-1	-	0
	0	-	-1

Beispiel:

```
PRINT NOT 0
```

Ausgabe:

```
-1
```

```
PRINT NOT -1
```

Ausgabe:

```
0
```

1.6.2 AND

Der Operator AND hat als Ergebnis nur dann eine wahre Aussage, wenn beide Bedingungen wahr sind.

Operator	Wert 1	Wert 2	Ergebnis
AND	0	0	0
	0	-1	0
	-1	0	0
	-1	-1	-1

Beispiel:

```
PRINT 0 AND 0,0 AND 1,1 AND 0,1 AND 1
```

Ausgabe:

```
0    0    0    1
```

Ein weiteres Beispiel soll die Funktion von AND verdeutlichen.

Beispiel:

```
PRINT 23 AND 12
```

Ausgabe:

```
4
```

Um dieses Ergebnis verständlich zu machen, schauen wir uns das Bitmuster der Werte 12 und 23 an.

Das Bitmuster von 23 ist gleich

00010111.

Das Bitmuster von 12 ist gleich

00001100.

Diese zwei Bitmuster werden nun mit AND verküpft.

```
  00010111
           > AND
  00001100

= 00000100 = 4
```

1.6.3 OR

Der Operator OR erzeugt eine wahre Aussage, sobald eine der beiden Aussagen wahr ist.

Operator	Wert 1	Wert 2	Ergebnis
OR	0	0	0
	0	-1	-1
	-1	0	-1
	-1	-1	-1

Beispiel:

PRINT 0 OR 0,0 OR 1,1 OR 0,1 OR 1

Ausgabe:

0 1 1 1

Ein weiteres Beispiel soll die Funktion von OR verdeutlichen.

Beispiel:

PRINT 23 OR 12

Ausgabe:

31

Um dieses Ergebnis verständlich zu machen, schauen wir uns wiederum das Bitmuster der Werte 12 und 23 an.

Das Bitmuster von 23 ist gleich

00010111.

Das Bitmuster von 12 ist gleich

00001100.

Diese zwei Bitmuster werden nun mit OR verküpft.

```
  00010111
           > OR
  00001100

= 00011111 = 31
```

Genau wie die Rechenarten besitzen auch die logischen Operatoren eine Priorität. Dabei hat **NOT** die **stärkste**, **AND** die **zweitstärkste** und **OR** die **schwächste Priorität**. Das bedeutet konkret, daß z.B. zuerst eine Negation ausgeführt wird, bevor eine Verknüpfung mit AND oder OR ausgeführt wird. Selbstverständlich kann durch Klammerung der logischen Ausdrücke diese Reihenfolge verändert werden.

Zum Schluß des Kapitels über die logischen Operatoren wollen wir noch die Funktion **XOR** (e**X**klusives **OR**) besprechen, da diese in ihrer Wirkungsweise unmittelbar mit diesem Kapitel zu tun hat.

1.6.4 XOR

Wie bereits erwähnt setzt sich diese Funktion aus einer Kombination der drei Operatoren zusammen. Betrachten wir jedoch zunächst die Funktion von XOR.

Im täglichen Sprachgebrauch verwenden wir meistens diese *exklusive ODER (ausschließendes ODER)*. Wenn z.B. ein Freund zum anderen sagt: "Ich komme mit dem Fahrrad oder ich komme mit dem Auto.", so schließen sich beide Möglichkeiten gegenseitig aus, da er ja nicht gleichzeitig mit dem Auto und mit dem Fahrrad fahren kann. Entweder fährt er mit dem Auto, dann kommt er nicht mit dem Rad, oder er fährt mit dem Rad und kommt nicht mit dem Auto. Somit erhält man bei der **XOR-Funktion** nur dann eine **wahre Aussage**, wenn die **beiden** zu verküpfenden **Aussagen** einen **verschiedenen Wahrheitsgehalt** besitzen.

Daß die XOR-Funktion in der beschriebenen Art und Weise arbeitet, können Sie überprüfen, indem Sie die folgende Zeile in den Computer eingeben:

```
PRINT XOR(0,0),XOR(0,1),XOR(1,0),XOR(1,1)
```

Ausgabe:

```
 0   1   1   0
```

Damit sieht die Tabelle für die XOR-Funktion wie folgt aus:

Operator	Wert 1	Wert 2	Ergebnis
XOR	0	0	0
	0	-1	-1
	-1	0	-1
	-1	-1	0

Nun will ich Ihnen noch, wie bereits versprochen, die Boolsche Operation für die XOR-Funktion verraten. Die Funktion XOR setzt sich aus den drei Grundoperatoren wie folgt zusammen:

Q = (X AND NOT Y) OR (NOT X AND Y)

Dabei ist Q jeweils das Ergebnis der Operation, wenn X und Y nacheinander die Werte 0 und 1 annehmen.

So, nun habe ich vorerst genug von mir gegeben. Es wird Zeit, daß Sie etwas zur Übung tun. 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 finden Sie wie bereits erwähnt im Anhang des Buches.

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) F0CA b) 1268
   c) 35A0 d) 0255
   e) F000 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

2

EINFÜHRUNG IN DAS PROGRAMMIEREN MIT BASIC

2. Einführung in das Programmieren mit BASIC

In diesem Kapitel soll die Verwendung zunächst einfacher, später komplexerer BASIC-Befehle anhand einfacher BASIC-Programme gelernt werden. Das erste Programm soll genau nach den fünf aufgestellten Grundregeln erstellt werden. Danach wollen wir uns hauptsächlich mit dem dritten Punkt befassen, nämlich mit dem Umsetzen des Algorithmus' in BASIC.

2.1 Das erste BASIC-Programm

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

1. Definition des Problems

Sein Commodore 128 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

Im folgenden 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

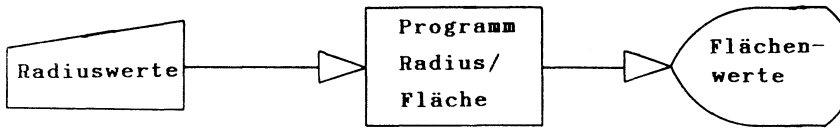


Bild 8

Programmablaufplan

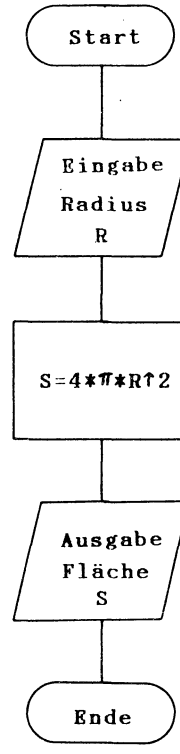


Bild 9

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 in den Rechner eingeben und dann die Zeile 10. Der Computer ordnet die Zeilen automatisch nach Größe der Zeilennummern und führt sie auch in dieser Reihenfolge aus, falls nicht andere BASIC-Befehle diese Folge durch Programmsprünge verändern.

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

2.1.1 Eingabe von Werten mit INPUT

Der Befehl INPUT wird in einem Programm zur Übergabe von Werten während des Programmablaufs benutzt. Der Benutzer kann somit direkten Einfluß auf den Programmablauf nehmen. Nach INPUT kann ein Kommentar in Anführungszeichen folgen, ähnlich wie in unserem Beispielprogramm. Dem Kommentar können eine oder mehrere Variablen folgen. Die erste Variable, die dem Kommentar folgt, wird durch ein Semikolon abgetrennt. Die Variablen untereinander müssen durch Kommata getrennt werden. Trifft das Programm auf den INPUT-Befehl, so wird der Programmablauf unterbrochen und auf dem Bildschirm erscheint der Cursor. Das Programm erwartet nun eine Eingabe über die Tastatur, die mit der RETURN-Taste bzw. 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 RETURN wird der Variablen S der Wert 3 zugeordnet.

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

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

Wichtig: Bei INPUT werden mehrere Variablen durch **Kommata** getrennt. Zur Trennung der Dezimalstellen wird der **Dezimalpunkt** verwendet.

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 Commodore 128 die Zuweisung auch so annimmt[1]. Die folgenden Beispiele sollen dies wieder verdeutlichen.

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.

1 Anmerkung: Einige BASIC-Dialekte benötigen den LET-Befehl unbedingt.

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 noch einmal 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 wieder 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 BASIC 7.0 des Commodore 128. Sie können mit diesem Befehl Texte bzw. Mitteilungen oder Werte von Variablen ausgeben lassen. Sie können beides kombinieren, d.h. daß 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 IST GLEICH";A	A IST GLEICH 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 oben aufgeführten Beispiele durchsprechen, will ich noch etwas zur Schreibweise in den Beispielen sagen.

Der Commodore 128 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 Sie danach die RETURN- oder 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 RETURN- oder ENTER-Taste, so wird diese Zeile im BASIC-Speicher des Rechners abgelegt. Durch die Eingabe des Befehls **RUN** und durch das Betätigen der RETURN-Taste 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 Zahl 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, so daß 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 hat zur Folge, daß beim nächsten PRINT-Befehl die Ausgabe am Anfang der nächsten Zeile ausgeführt wird. Die Begriffe Wagenrücklauf und Zeilenvorschub wollen wir uns 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 innerhalb der Anführungszeichen stehen, ausgegeben, es sei denn, es handelt sich um bestimmte Steuerzeichen, z.B. für das Klingelzeichen. Dieses **Klingelzeichen** hat den **CHR$-Code 7** (s.a. Kap. ASC(X$) und CHR$(X)). Verwenden Sie also innerhalb der Anführungszeichen die Tastenkombination **CONTROL G** (Control Taste und G Taste gleichzeitig betätigen), so erscheint ein inverses G. Dieses Zeichen wird jedoch nicht ausgegeben, sondern Sie werden einen Piepston vernehmen.

Die Befehlsfolgen

PRINT "<*CONTROL G*>"

und

PRINT CHR$(7)

erzeugen also jeweils einen Piepston.

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 **beeinflußt** die eigentliche **Ausgabeform** aber noch auf eine andere Art und Weise.

Der Commodore 128 kann 40 oder 80 Zeichen pro Zeile darstellen, je nachdem ob Sie ein normales TV-Gerät oder einen RGB-Monitor angeschlossen haben. Sollten Sie in der glücklichen Lage sein, beide Geräte an Ihren Computer anschließen zu können, haben Sie die Möglichkeit, mit der 40/80 DISPLAY-Taste zwischen den beiden Sichtgeräten hin- und herzuschalten.

Bei jeder Darstellung, ob 40 oder 80 Zeichen, ist jede Bildschirmzeile nochmals in Bereiche zu je 10 Zeichen unterteilt (4*10 Zeichen bei 40-Zeichendarstellung bzw. 8*10 Zeichen 80-Zeichendarstellung), sogenannte Tabulatoren. Wird nun das Komma zwischen zwei Variablen verwendet, so wird die zweite Variable an den Anfang des zweiten Tabulators ausgegeben, also ab der 10. Stelle in der Bildschirmzeile. Werden mehrere Variablen durch das Komma getrennt, so werden sie an den entsprechenden Stellen ausgegeben.

Geben Sie nun folgendes in den Rechner ein:

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

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

In Beispiel e) wird Ihnen die Wirkung des **Semikolons** gezeigt. Dadurch werden nicht nur der **Wagenrücklauf** und der **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 beschreibenden 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 Computer **im Speicher** *(nicht im Programmlisting selbst)* 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 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 Zahlenwerte oder Variablen bzw. Zeichenketten (Strings) 'formatiert', d.h. in einer bestimmten Form, ausgeben lassen wollen. Dafür stehen im BASIC 7.0 der PRINT USING-Anweisung folgende Zeichen zur Verfügung:

#	Nummernkreuz bestimmt die auszugebende Anzahl der Zeichen.
+	wird bei positiven Zahlen mit angegeben.
-	wird bei negativen Zahlen mit ausgegeben. Plus- oder Minuszeichen können nur getrennt verwendet werden.
.	kennzeichnet die Position des Dezimalpunktes.

$ wird benutzt, um z.B. Währungen zu kennzeichnen.

↑↑↑↑ Zahlen werden in wissenschaftlicher Notation ausgegeben. (z.B. 1.23 E+02)

= String (Zeichenkette) wird innerhalb des Formatfeldes zentriert ausgegeben.

> String wird innerhalb des Formatfeldes rechtsbündig ausgegeben.

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

```
A   = 12345.678          <RETURN-Taste betätigen>
B   = 34.3455            <RETURN>
C   = -128               <RETURN>
D$  = "CBM 128"          <RETURN>
```

Geben Sie nun die folgenden PRINT USING-Befehlsfolgen in den Computer ein und betätigen Sie jedesmal danach die RETURN-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 an der gleichen Bildschirmposition ausgegeben wird. Dies wäre bei der Anwendung des PRINT-Befehls nicht ohne weiteres möglich gewesen.

Wird das angegebene Format vom Wert der Zahl überschritten, so werden statt des Zahlenwerts nur Sternchen bzw. die Multiplikationszeichen ausgegeben. So würde die folgende PRINT USING-Anweisung

```
PRINT USING "####.##";A
```

diese *Ausgabe* verursachen:

```
*******
```

Durch diese Ausgabe gibt Ihnen der Commodore 128 also zu verstehen, daß der Zahlenwert nicht in das von Ihnen gewählte Formatfeld der PRINT USING-Anweisung paßt.

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.

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

```
PRINT USING "#$#####.##";A
```

Ausgabe:

```
$12345.68
```

```
PRINT USING "#$#####.##";B
```

Ausgabe:

```
$34.35
```

Sie werden sicherlich bemerkt haben, daß dem Dollarzeichen noch ein Nummernzeichen '#' vorangestellt wurde. Dieses zusätzliche Nummernzeichen bewirkt, daß das Dollarzeichen immer direkt vor der Zahl ausgegeben wird. Lassen Sie das Nummernzeichen fortfallen, so wird das Dollarzeichen linksbündig ausgegeben.

```
PRINT USING "$#####.##";B
```

Ausgabe:

```
$  34.35
```

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

```
PRINT USING "##,###.## DM";A
```

Ausgabe:

```
12,345.68 DM
```

Wie diese Ausgabe der europäischen Schreibweise angepaßt werden kann (Komma und Dezimalpunkt vertauschen), können Sie bei der nachfolgenden Beschreibung der PUDEF-Anweisung nachlesen.

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

```
PRINT USING "#.##↑↑↑↑";A
```

Ausgabe:

```
1.23E+04
```

Wie viele Stellen vor bzw. nach dem Dezimalpunkt erscheinen sollen, können Sie selbstverständlich individuell bestimmen, wie das folgende Beispiel zeigt.

```
PRINT USING "###.##↑↑↑↑";A
```

Ausgabe:

```
123.46E+02
```

Zwei weitere Steuerzeichen sind noch relevant, und zwar das

> Zeichen

und das

= Zeichen.

Diese zwei Zeichen finden hauptsächlich bei der Ausgabe von Strings bzw. Text eine Verwendung, können aber selbstverständlich auch bei numerischen Ausgaben benutzt werden.

```
PRINT USING "=##########";D$
```

Ausgabe:

```
   CBM 128
```

Der Inhalt der Variablen D$ wurde durch diese Anweisung innerhalb des Formatfeldes **zentriert** ausgegeben. Das "=" Zeichen zentriert also die Ausgaben von Stringvariablen oder numerischen Variablen.

Das > Zeichen wird innerhalb der PRINT USING-Anweisung benutzt, um Ausgaben **rechtsbündig** im Formatfeld anzuzeigen.

```
PRINT USING ">##########";D$
```

Ausgabe:

```
   CBM 128
```

Damit hätten wir die Erläuterungen der Steuerzeichen der PRINT USING-Anweisung abgeschlossen. Der Commodore 128 bietet allerdings über eine zusätzliche Anweisung die Möglich-

keit, die PRINT USING-Anweisung noch weiter zu beeinflussen. Diese Anweisung nennt sich **PUDEF.**

2.1.3.2 PUDEF beeinflußt PRINT USING

Wie bereits vorhin angesprochen, gibt es die Möglichkeit, über die PUDEF-Anweisung die Form der Ausgabe von PRINT USING zu beeinflussen. Der **PUDEF-Anweisung** folgt eine Zeichenkette oder Stringvariable die sich aus **maximal vier Zeichen** zusammensetzen darf.

Die folgende Schreibweise stellt die Standardeinstellung der PRINT USING-Anweisung dar.

PUDEF " ,.$"

Das **erste Zeichen** ist das **Leerzeichen,** das **zweite** das **Komma,** das **dritte** der **Dezimalpunkt** und schließlich das **vierte** das **Dollarzeichen.**

In dieser Reihenfolge sind auch die Veränderungen für die einzelnen Zeichen vorzunehmen. Wollen Sie also das vierte Zeichen ändern, so müssen Sie die ersten drei Zeichen ebenfalls mit eingeben, auch wenn diese sich nicht ändern. Diese vier Zeichen können Sie auch in einer Stringvariablen ablegen, wie das folgende Beispiel zeigt:

```
A$=" ,.$"
PUDEF A$
```

Diese Form der Anweisung PUDEF ist ebenfalls zulässig.

Nun wollen wir anhand einiger Beispiele die Funktion von PUDEF verdeutlichen.
Für die Ausgabe von Zahlenwerten soll die europäische Schreibweise gewählt werden, d.h. der Dezimalpunkt muß als Komma erscheinen. Weiterhin sollen die Tausenderstellen optisch

durch einen Punkt getrennt werden. Dazu muß das Komma der PRINT USING-Anweisung als Punkt definiert werden.

Die PUDEF-Anweisung sieht demnach wie folgt aus:

```
PUDEF " ,"
```

Die folgende PRINT USING-Anweisung

```
PRINT USING "###,###.## DM";A
```

erzeugt dann die folgende *Ausgabe*:

```
12.345,68 DM
```

Durch die PUDEF-Anweisung wurde also das zweite und dritte Zeichen (Komma und Dezimalpunkt) der PRINT USING-Anweisung verändert (Komma = Dezimalpunkt und Dezimalpunkt = Komma). Das erste Zeichen (Leerzeichen) wurde nicht verändert, mußte aber dennoch mit angegeben werden.

Jetzt soll noch das Leerzeichen durch das Multiplikationszeichen (Sternchen) ersetzt werden. Dadurch erreichen wir eine Ausgabe, wie man sie von Überweisungsträgern her kennt. Die dazu nötige PUDEF-Anweisung sieht wie folgt aus:

```
PUDEF "*"
```

Geben Sie nun wieder die gleiche PRINT USING-Anweisung wie oben ein.

```
PRINT USING "###,###.## DM";A
```

Sie erhalten dann die folgende *Ausgabe*:

```
*12.345,68 DM
```

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 - aber auch für sich selbst - 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 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)";R
40 REM BERECHNUNG OBERFLAECHE
50 LET S=4*PI*R↑2
60 REM AUSGABE OBERFLAECHE IN CM↑2
70 PRINT"DIE KUGELOBERFLAECHE BETRAEGT ";S;"CM↑2"
80 END
```

Die Zeilen 10-20 dienen dazu, dem Benutzer zu sagen, was das Programm macht und wie die Eingabe zu erfolgen hat. In Zeile 30 erfolgt die Eingabe der Daten mittels INPUT, wobei hier nochmal für den Anwender ein erläuternder Kommentar mit ausgegeben wird. Dies hätte man auch dadurch erreichen können, daß man den Text in einer separaten Programmzeile mit

dem PRINT-Befehl ausgegeben und den INPUT-Befehl ebenfalls alleine in eine Programmzeile geschrieben hätte. Zeile 40 weist mit dem Kommentar auf die folgende Berechnug in Zeile 50 hin. In Zeile 50 wird der Variablen S durch Berechnung des rechten mathematischen Ausdrucks der Wert der Oberfläche zugeordnet. Zeile 60 verweist mit dem Kommentar auf die Ausgabe in CM^2 in Zeile 70. In Zeile 70 wird durch den PRINT-Befehl die Kombination von Text- und Wertausgabe der Variablen veranlaßt. Zeile 80 beendet schließlich das Programm mit dem END-Befehl.

2.2 Variablen und deren Verwendung

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

Im Commodore-BASIC 7.0 gibt es **drei** verschiedene **Typen** von **Variablen**, die auf drei verschiedene Arten 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, die *Real-Variable* wird benutzt, um Dezimalzahlen darstellen zu können. Die dabei verwendeten Variablenbezeichnungen erhalten keinen Zusatz, um diesen Typ zu kennzeichnen. Erlaubte Bezeichnungen sind z.B. A oder B2.

Der dritte Variablentyp wird zusätzlich durch das "$" Dollarzeichen gekennzeichnet. Es handelt sich hierbei um die sogenannte *Stringvariable*. In dieser Variablen können beliebige Zeichenfolgen abgespeichert und bei Bedarf ausgegeben werden.

Allerdings dürfen nicht mehr als 255 Zeichen in der Stringvariablen verwendet werden, da sonst die Fehlermeldung

?STRING TOO LONG ERROR

ausgegeben wird.

Bei der Verwendung der Bezeichnungen von Variablen muß allerdings einiges berücksichtigt werden. Der Rechner erkennt eine Variable nur an den ersten beiden Zeichen. Sie dürfen zwar längere Variablennamen benutzen (z.B. WERT), aber der Computer kann diese Bezeichnung nicht von der Bezeichnung WECHSEL unterscheiden, da die **ersten zwei Zeichen** (WE) bei beiden Variablen gleich sind. Sie dürfen also durchaus Ihren Variablen die Namen *OBERFLAECHE* und *FLAECHE* geben. Es dürfen sich in diesen Namen aber wiederum keine BASIC-Befehle verstecken. Die **Bezeichnung WAND** wäre **unzulässig**, da sich in ihr der logische Operator AND befindet.

Weiterhin dürfen zusätzlich Ziffern zur Kennzeichnung benutzt werden, allerdings mit der Einschränkung, daß diese erst an zweiter Stelle auftreten dürfen. Erlaubt sind Bezeichnungen wie A1, B9 oder ähnliche. Es ist nicht erlaubt, die Zahl an die erste Stelle zu setzen.

Z.B. ist 9B **nicht zulässig!**

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

OR, FN oder IF.

Der Commodore 128 benutzt außerdem folgende Variablenbezeichnungen für interne Funktionen:

TI, TI$, ST, ER, ERR$

Diese dürfen Sie ebenfalls nicht in Ihren Programmen verwenden. Soweit die Einschränkungen bei den Bezeichnungen für die Variablen (siehe auch Anhang 2. Reservierte Wörter).

2.2.1 Rechenoperationen mit Variablen

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

Zeichen	Rangfolge	Bedeutung
↑	Erste	Potenzieren
*	Zweite	Multiplikation
/		Division
+	Dritte	Addition
-		Subtraktion

Auch für die logischen Operatoren existiert ja eine solche Rangfolge, wie wir bereits gesehen haben.

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. Im nächsten Kapitel wollen wir uns dann einige neue Befehle und ihre Verwendungen im Programm aneignen. Und nun viel Erfolg beim Lösen der Aufgaben.

Aufgaben

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

a)	X1	b)	ERDE$	c)	STAND
d)	ODER%	e)	GRIFF	f)	NORD$
g)	4Z%	h)	25	i)	F0

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.

2.3 Numerische Funktionen

Bisher haben wir uns mit einfachen Wertzuweisungen von Variablen befaßt, d.h. es wurden nicht die vorhandenen mathematischen Funktionen des Commodore 128 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-Buch bleiben und kein mathematisches Lexikon werden.

In vielen BASIC-Büchern und auch im Handbuch des Commodore 128 findet man immer die Angabe, daß die Werte der trigonometrischen Funktionen wie SIN(X), COS(X) oder TAN(X) im **Bogenmaß** anzugeben sind. Was ist nun dieses Bogenmaß?

Nun, es handelt sich hierbei auch um eine Winkelangabe, zu der der **Sinus** oder **Cosinus** berechnet werden soll. Die normale Aufteilung eines Kreises in 360 Grad dürfte jedem bekannt sein. 1 Grad ist also der 360ste Teil eines Kreises. 90 Grad stehen demnach für den Viertelkreis, 180 Grad 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 INPUT"EINGABE IN GRAD";GR
20 REM BERECHNUNG DES SINUS
30 SI=SIN(GR*PI/180)
40 PRINT"DER SINUS VON";GR;"GRAD ";
50 PRINT"IST =";SI
60 END
```

Starten Sie es jetzt mit RUN und geben Sie für den **Winkel** den **Wert 90** ein. Als **Ergebnis** sollten Sie **1** erhalten. Dieses Programm erwartet die Eingabe des Winkels in Grad und berechnet den dazugehörigen Sinus. Wollen Sie in Ihren eigenen Programmen also die Winkel in Grad eingeben, so müssen Sie die Umrechnung in Zeile 30 benutzen. Für die Berechnung des Cosinus' müßten Sie in Zeile 30 nur SIN durch COS ersetzen. Die Variable SI können Sie beibehalten.
Wundern Sie sich nicht, wenn Sie das Programm in der COS-Version starten und bei Eingabe von 90 den Wert 7.3145904E-10 erhalten. Dies ist der internen Rechenungenauigkeit des Commodore 128 zuzuschreiben. Diese Zahl ist allerdings so klein, daß man Sie getrost als Null werten kann. Um den Unterschied zum Bogenmaß zu verdeutlichen, geben Sie das

Programm in der folgenden Version ein. Zuvor jedoch geben Sie noch NEW ein und betätigen die RETURN-Taste.

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

Das einzige, was sich gegenüber unserem vorigen Programm geändert hat, ist die Zeile 30. Starten Sie nun das Programm und geben Sie den Wert 1.57079633 (entspricht PI/2) ein. Als Ergebnis erhalten Sie wieder einen Wert, der fast Null ist. Normalerweise ist der Sinus von PI/2 (Pi Halbe) genau Null. Diese geringe Abweichung ist wieder der internen Rechenungenauigkeit des CBM 128 zuzuschreiben.

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

PRINT EXP(1)

Sie erhalten als *Ergebnis* die Zahl

2.71828183,

auch als *Eulersche Zahl* bekannt. Wiederholen Sie den gleichen Vorgang mit der folgenden Befehlsfolge:

PRINT LOG(2.71828183)

Nun erhalten Sie wiederum die 1. Wollen Sie den Logarithmus zur *Basis 10* berechnen, so müssen Sie nur **LOG(X) durch LOG(10) dividieren.** Der Logarithmus zur *Basis 10* nennt sich auch "*dekadischer Logarithmus*" und der Logarithmus zur *Basis e* "*natürlicher Logarithmus*".

Das folgende Beispielprogramm errechnet Ihnen sowohl den natürlichen als auch den dekadischen Logarithmus.

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

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

Wichtig: Die trigonometrischen Funktionen erwarten die Werte im Bogenmaß. Für die Berechnung in Grad müssen diese entsprechend umgerechnet werden.

Die Funktionen LOG und EXP beziehen sich auf den Exponenten bzw. die Basis "e".

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

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

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

durch die Multiplikation mit 10 hoch X% vor den Dezimalpunkt gesetzt wurden.

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

2.3.1 Funktionen 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 Commodore 128 besitzt einen eingebauten Zufallszahlengenerator, der über die Funktion 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 im Bereich zwischen 0.0 und 1.0 (*Null und Eins ausgeschlossen*). Bei negativen Werten von X wird immer die gleiche Folge von Zufallszahlen ausgegeben. Das folgende kleine Programm simuliert einen Würfel. Bei jedem Start des Programms wird zufällig eine Zahl zwischen 1 und 6 ausgegeben.

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

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

Damit Zahlen zwischen 1 und 6 ausgegeben werden, wurde in der Zeile 20 eine **Kombination** aus **RND** und **INT** gewählt, da wir ja ganze Zahlen benötigen. Die 1 wurde noch addiert, damit

keine Null vorkommen kann (untere Grenze) und der maximale Wert von 6 erreicht werden kann.

Mit dieser Art der Zufallszahlengenerierung können Sie in jedem beliebigem Intervall Zufallszahlen erzeugen lassen. Dabei steht die 6 für die obere Grenze des Intervalls und die +1 für die untere Grenze des Intervalls. Wollen Sie nun Zufallszahlen im Bereich von 100 bis 150 erzeugen, so müßte die Zeile 20 so aussehen:

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

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

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

Bei den einfacheren Beispielen erkennt man diese allgemeine Formel nicht immer auf Anhieb. Die folgende Zeile

```
20 A=INT(6*RND(1))+1
```

muß korrekt eigentlich so lauten:

```
20 A=INT((6+1-1)*RND(1))+1
```

Sie sehen, daß bei einer unteren Grenze von 1, sich der Ausdruck im ersten Teil der Formel vereinfacht.

2.3.3 Noch mehr Befehle für Variablen

Das umfangreiche BASIC 7.0 des Commodore 128 besitzt eine Menge an Befehlen, mit denen Variablen bzw. Variablenformate beeinflußt werden können. Zwei weitere Befehle bzw. Funktionen sollen im folgenden nun kurz besprochen werden.

Sie erinnern sich bestimmt noch an das Kapitel über die Zahlensysteme. Der Commodore 128 bietet dem Benutzer zwei Funktionen, die es ihm ermöglichen, dezimale Zahlen in das

hexadezimale oder hexadezimale Zahlen in das dezimale Zahlensystem umzuwandeln.

Die Funktion

HEX$(X)

wandelt Dezimalzahlen in Hexadezimalzahlen um. 'X' steht hier für die umzurechnende Zahl. Dabei darf *X Werte im Bereich zwischen 0 (Null) und 65535* annehmen. Jeder andere Wert von X hat die Fehlermeldung

?ILLEGAL QUANTITY ERROR

zur Folge.

Beispiel:

```
PRINT HEX$(60)
```

Ausgabe:

```
003C
```

Sie sehen, daß die hexadezimalen Zahlen immer vierstellig ausgegeben werden. Damit können Sie als größte Hexadezimalzahl FFFF oder dezimal eben 65535 erhalten. Statt einer Zahl können Sie natürlich auch eine Variable einsetzen.

Beispiel:

```
A=1024:PRINT HEX$(A)          <RETURN>
```

Ausgabe:

```
0400
```

Die Funktion

DEC("X")

wandelt Hexadezimalzahlen in Dezimalzahlen um. 'X' steht hier wiederum für die umzurechnende Zahl. Dabei müssen die *X-Werte im Bereich zwischen 0000 (Null) und FFFF* in Anführungszeichen oder in Form einer Stringvariablen der Funktion übergeben werden. Jeder andere Wert von X hat wieder die Fehlermeldung

?ILLEGAL QUANTITY ERROR

zur Folge.

Beispiel:

```
PRINT DEC("C200")
```

Ausgabe:

```
49664
```

Das nächste Beispiel verwendet eine Stringvariable, die zuvor entsprechend definiert wurde.

Beispiel:

```
A$="ABCD":PRINT DEC (A$)          <RETURN>
```

Ausgabe:

```
43981
```

Verwenden Sie eine Stringvariable, so müssen Sie darauf achten, daß diese Variable nicht mehr als vier Zeichen beinhaltet (Leerzeichen ausgenommen). Weiterhin dürfen nur die Zeichen 0 bis 9 und A bis F eingesetzt werden.

Damit hätten wir soweit die wichtigsten Funktionen, die die Umwandlung von Variablen in die zwei Zahlensysteme ermöglichen, besprochen.

2.3.4 ASC(X$) und CHR$(X)

Der Commodore 128 besitzt die Möglichkeit, durch den PRINT-Befehl Zahlen, Buchstaben und Grafikzeichen, wenn diese zwischen den Anführungszeichen (*Anführungszeichenmodus* oder auch *Quotemode* genannt) stehen, auf dem Bildschirm ausgeben zu lassen. Weiterhin existieren unter diesen Zeichen bestimmte Steuerzeichen, die z.B. die Farbe der nachfolgenden Zeichen beeinflussen oder eine Umschaltung der Darstellung in reverser Schrift ermöglichen. Die Steuerzeichen sind meistens durch irgendwelche Grafikzeichen definiert, teilweise sogar in reverser Darstellung. Sie haben den Vorteil, daß sie sofort durch das Drücken der entsprechenden Taste beim Programmieren verfügbar sind. Sie haben jedoch auch einen großen Nachteil:
Diese Zeichen sind sich teilweise sehr ähnlich, so daß 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, spielt das weiter keine Rolle. Er kennt ja die Steuerzeichen, die er verwendet hat. Allerdings dürfte jemand, der das Programm nicht kennt und nur "abtippen" will, Schwierigkeiten bekommen. Erstens gibt es zum Teil Probleme bei der Identifizierung der Steuerzeichen und zweitens sind manche Steuerzeichen nur aufrufbar, wenn mehrere Bedingungen gleichzeitig erfüllt sind. (Anführungszeichen gesetzt; SHIFT und CTRL gleichzeitig gedrückt).

Muß man bei diesen vielen Möglichkeiten nach einem bestimmten Zeichen suchen, so ähnelt das manchmal eher einem Detektivspiel als einer Programmierung in BASIC. Dabei kann man

sich und anderen in den meisten Fällen die Arbeit erheblich vereinfachen, indem man die CHR$-Codes verwendet. Es gibt z.B. folgende PRINT-Anweisung, um den Bildschirm zu löschen (die SCNCLR Anweisung lassen wir bei diesem Beispiel unberücksichtigt):

PRINT "♥"

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

PRINT CHR$(147)

Tritt diese Befehlsfolge in einem Listing auf, so dürfte es kaum Unklarheiten in Bezug auf das Eingeben dieser Befehle in den Rechner geben.

Ein anderes Beispiel sind die grafischen Steuerzeichen für den Cursor oder für die Funktionstasten. Diese Zeichen sehen sich teilweise so ähnlich, daß, wenn sie in einem Listing gleichzeitig auftauchen, kaum auseinandergehalten werden können. Da ist es weitaus einfacher und leichter, für die Funktionstasten oder den Cursor die ASCII-Werte zu benutzen.

Im Handbuch zum Commodore 128 können Sie die entsprechenden ASCII-Werte nachschlagen. Wollen Sie z.B. den ASCII-Wert des Buchstabens A wissen, so geben Sie folgendes im Direktmodus in den Rechner:

PRINT ASC("A")

Betätigen Sie jetzt die RETURN-Taste, so erscheint auf dem Bildschirm der Wert 65.

Die Funktion CHR$ stellt nun die Umkehrung zum oberen Befehl dar. Geben Sie folgendes wieder im Direktmodus ein:

PRINT CHR$(65)

Betätigen Sie wiederum die RETURN-Taste, so erscheint auf dem Bildschirm jetzt das Zeichen A. Die Benutzung dieser Befehle ist meines Erachtens problemloser und eindeutiger als die Verwendung von Grafikzeichen. Deshalb wollen wir nach Möglichkeit auch in unseren folgenden Programmen diese Schreibweise mit den Befehlen *ASC("X")* bzw. *CHR$(X)* beibehalten, zumindest bei den speziellen Codes wie Farbwechsel der Schrift, Umschaltung auf Kleinbuchstaben oder ähnlichen Funktionen. Die Umsetzung der Programme auf andere Rechner wird durch die Benutzung der CHR$-Codes ebenfalls erleichtert, da meistens die Bedeutung dieser speziellen ASCII-Werte bei den Commodorerechnern übereinstimmt, z.B. CHR$(147) für das Löschen des Bildschirms.

Der Commodore 128 benutzt, wie bereits erwähnt, das reverse Herz zwischen den Anführungszeichen zum Löschen des Bildschirms. Dieses Zeichen wird von anderen Rechnern schon nicht mehr für diese Funktion benutzt. Man findet dort zum Teil ein reverses "S" oder andere Zeichen. Dagegen können Sie den Befehl *PRINT CHR$(147)* direkt übertragen, ohne großartig nach einer bestimmten Tastenfunktion suchen zu müssen.

So können Sie auch innerhalb eines Programms die Funktionen aufrufen, die in Verbindung mit der ESC-Taste auf der Tastatur ein- bzw. ausgeschaltet werden können. Durch Drücken der ESC-Taste und anschließender Betätigung der E-Taste kann das Blinken des Cursors ausgeschaltet werden. Die gleiche Wirkung hat die folgende CHR$-Code Sequenz:

PRINT CHR$(27);CHR$(69)

oder

PRINT CHR$(27);"E"

Damit Sie nun Ihr neuerworbenes Wissen anwenden können, will ich Ihnen zur Übung ein paar Aufgaben stellen. Lösen Sie diese Aufgaben und vergleichen Sie dann Ihre Ergebnisse mit den Lösungsvorschlägen und Erklärungen am Ende dieses Buches. Danach können Sie dann das nächste Kapitel durcharbeiten.
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 folgenden Aufgaben.

Aufgaben

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

2.4 TAB(und SPC(

Diese beiden Funktionen werden benutzt, um Daten oder Zeichen an bestimmten Positionen in Bildschirmzeilen auszugeben. Vielleicht haben Sie sich über die Kapitelüberschrift gewundert, aber die Klammern gehören bei beiden Funktionen mit dazu. Sie dürfen also auf keinen Fall ein Leerzeichen zwischen der Funktion und der ersten Klammer unterbringen. Dies müssen Sie bei der Anwendung der zwei Funktionen unbedingt beachten. **TAB(** und **SPC(** sind in der Anwendung sehr ähnlich, jedoch in der Wirkung unterschiedlich. Die TAB-Funktion und der Parameter in Klammern positionieren die Ausgabe immer in Bezug auf den Anfang der aktuellen Bildschirmzeile. Geben Sie folgende Befehlsfolge im Direktmodus ein:

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

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

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

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

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

Drücken Sie jetzt die RETURN-Taste, so werden Sie den Unterschied in der Ausgabe auf dem Bildschirm sehen. Das zweite Wort TEST 2 wird nicht an der 20. Position vom Anfang der Zeile gerechnet ausgegeben, sondern an der 20. Position vom letzten Zeichen des Wortes TEST 1 an gerechnet. Das bedeutet, daß die *TAB(-Funktion* immer auf die **absolute** Position in der Bildschirmzeile und die *SPC(-Funktion* immer auf die **relative** Position zum letzten ausgegebenen Zeichen bezogen sind. Die Werte, die beiden Funktionen übergeben werden können, dürfen nicht größer als 255 sein. Bei der Benutzung dieser Funktionen in Verbindung mit der Ausgabe auf einen Drucker ist darauf zu achten, daß die TAB(-Funktion nach Möglichkeit keine Verwendung findet, da sie in Verbindung mit dem PRINT#-Befehl vom Drucker nicht interpretiert oder als SPC(interpretiert wird. Daher sollte die TAB(-Funktion nur zusammen mit dem "normalen" PRINT-Befehl benutzt werden.

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

2.5 Strings

Ein String bezeichnet eine Zeichenkette, die bis zu 255 beliebige Zeichen des Commodore 128 Zeichensatzes enthalten kann. Die *Stringvariable* wird durch das *"$"* Zeichen gekennzeichnet. A$ würde also eine reguläre Bezeichnung eines Strings darstellen. Die Zuordnung der Zeichen zu einer Stringvariablen erfolgt auf die gleiche Art und Weise wie bei den numerischen Variablen. Der einzige Unterschied besteht darin, daß die Zeichen in Anführungszeichen stehen. Eine gültige Zuordnung zeigt das folgende Beispiel:

```
A$="Commodore 128"
```

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

?TYPE MISMATCH ERROR

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

A="TEST" *(FEHLER)!*

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

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

In Zeile 10 werden zunächst die Stringvariablen A$ und B$ initialisiert. Zeile 20 ordnet die Verknüpfung der Variablen A$ und B$ der Variablen DL$ zu. Zeile 30 druckt schließlich den neuen String aus.

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

2.5.1 LEFT$

Das BASIC des Commodore 128 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. Das Ergebnis dieses Programms sehen Sie unten 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$, diesmal jedoch mit zwei Zeichen. Die Zeilen 40 bis 90 sind genauso zu interpretieren. Das bedeutet, daß der Befehl **LEFT$(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.

2.5.2 RIGHT$

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$-Befehl nur darin, daß er nicht die linken Zeichen eines Strings nimmt, sondern die rechten. Ändern Sie nun in dem vorigen Beispielprogramm alle LEFT$-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.

2.5.3 MID$

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 mit MID$ im BASIC 7.0 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 "DONAUD*A*MPF..." sondern "DONAUD*U*MPF...". Dieses Beispiel sollte vorerst zur Verdeutlichung der Funktionsweise des MID$ Befehls ausreichen.

2.5.4 LEN(X$)

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

PRINT LEN(A$)

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

2.5.5 VAL(X$)

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

wird nur der erste Teil des Strings mit den Ziffern übersetzt. Die nachfolgenden Beispiele sollen das verdeutlichen.

Beispiele:

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

Ergebnis: 343.45

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

Ergebnis: 0

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

Ergebnis: 234

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

Ergebnis: 33

Geben Sie diese Beispiele ruhig in den Computer ein und probieren Sie sie nacheinander aus. Das Beispiel a) zeigt den Fall auf, bei dem der komplette String in eine Zahl übersetzt werden kann. Der String von Beispiel b) beginnt mit einem Zeichen, das nicht in eine Zahl übersetzt werden kann, und wird daher als Null interpretiert. Beispiel c) zeigt einen "gemischten" String, bei

dem nur der erste Ziffernteil umgewandelt wird. Beispiel d) soll nur verdeutlichen, daß das Komma im Unterschied zum Dezimalpunkt ebenfalls nicht umgewandelt werden kann, und daß somit der restliche Ziffernteil nicht mehr berücksichtigt wird.

2.5.6 STR$(X)

Die Funktion, die genau das **Gegenteil von VAL$** bewirkt, also einen numerischen Ausdruck in einen String verwandelt, ist die **STR$(X)-Funktion**. Dabei müssen Sie beachten, daß der erzeugte String als erstes das Vorzeichen 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.

2.5.7 INSTR

Eine weitere nützliche Funktion für den Umgang mit Strings bietet uns das BASIC 7.0 mit

INSTR

mit der Sie einen String nach einer beliebigen Zeichenfolge durchsuchen lassen können. Die Schreibweise der Funktion sieht wie folgt aus:

INSTR(A$,B$,X)

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 gesuchte Zeichenfolge nicht oder ab einer bestimmten Position nicht mehr enthalten, so wird das Ergebnis Null. Das nachfolgende kleine Programm soll die Funktion etwas verdeutlichen.

```
10 A$="DONAUDAMPFSCHIFFAHRTSGESELLSCHAFT"
20 B$="SCHIFF"
30 Z=INSTR(A$,B$)
40 PRINT Z
50 END
```

Starten Sie das Programm mit RUN, so erhalten Sie als Ergebnis für Z den Wert 11. Berücksichtigen Sie bei der Anwendung dieser Funktion, daß der String nach genau den gleichen Zeichen durchsucht wird. Das bedeutet, daß er die Zeichenfolge "*Schiff*" nicht gefunden hätte, da diese sowohl Klein- als auch Großbuchstaben enthält.

Wichtig: Bei Anwendung von INSTR(A$,B$,X) muß 'X' immer größer Null sein. Ansonsten wird die Fehlermeldung

?ILLEGAL QUANTITY ERROR

ausgegeben.

Die Zeichenfolge, nach der gesucht wird, muß auf das Zeichen genau im String enthalten sein. Beachten Sie hierbei besonders den Unterschied zwischen Groß- und Kleinbuchstaben.

2.5.8 TI$

Einen speziellen "String" bzw. eine spezielle Stringvariable wollen wir uns in diesem Kapitel noch anschauen. Es handelt sich hierbei um eine interne Uhr im Commodore 128. Diese Uhr wird beim Einschalten des Rechners auf Null gesetzt und läuft dann solange wie der Rechner eingeschaltet ist. Dieser Wert wird also laufend aktualisiert und in der **Variablen TI$** abgelegt. Dieser String setzt sich aus sechs Zeichen zusammen, und zwar *je zwei für Stunden, Minuten und Sekunden.* Diese Variable zählt nur bedingt zu den **systemreservierten Variablen.** Man hat nämlich die Möglichkeit, diese Variable zu beeinflussen, d.h. man kann ihr einen Wert übergeben, der dann laufend aktualisiert wird. Geben Sie nun das folgende Programm in Ihren Rechner ein:

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

Starten Sie es jetzt mit RUN und beobachten Sie die linke obere Ecke auf Ihrem Bildschirm. Dort sollten jetzt sechs Ziffern stehen, wovon die äußerst rechte laufend im Sekundentakt ihren Wert vergrößert. Daß diese sechs Ziffern immer an der gleichen Stelle erscheinen, wird durch CHR$(19) erreicht. Es handelt sich hierbei um den Tastencode der HOME-Taste. Damit wird erreicht, daß immer ab der oberen linken Bildschirmecke die Aus-

gabe durch PRINT geschieht und somit die alte Ausgabe überschrieben wird.

Sie haben damit eine Digitaluhr auf Ihren Bildschirm "programmiert". Unterbrechen Sie nun das Programm mit der RUN/STOP-Taste. Es erscheint die Meldung:

BREAK IN (Zeilennummer)

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

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

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

?ILLEGAL QUANTITY ERROR

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

Der Befehl GOTO, den Sie bereits zweimal in den letzten Programmen verwendet haben, wird im nächsten Kapitel eingehend besprochen. Dieser Befehl kann den eigentlichen Programmablauf, der ja durch die Zeilennummern bestimmt wird, manipulieren. Bevor wir nun zum nächsten Kapitel übergehen, sollten Sie noch die folgenden Aufgaben lösen, damit Sie die neubesprochenen Befehle anwenden lernen. Und nun wie immer viel Erfolg beim Lösen der Aufgaben.

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 die Frage so zu beantworten.

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

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

   a) kein Unterschied

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

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

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

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

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

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

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

2.6 Editieren von Programmen

Bevor wir nun dazu übergehen, größere Programme zu entwickeln, wollen wir uns kurz mit den Befehlen 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 einzelne Zeichen mit der INST/DEL-Taste löscht oder einfügt, wird hier allerdings als bekannt vorausgesetzt. Sollten Sie trotzdem noch Schwierigkeiten damit haben, so lesen Sie bitte im Handbuch des Commodore 128 das entsprechende Kapitel nach.

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

AUTO

Befehl. Geben Sie z.B. **AUTO 10** in den Rechner und betätigen Sie die RETURN-Taste, so haben Sie damit die automatische Zeilennumerierung in Zehnerschritten eingeschaltet. Sie können nun mit einer beliebigen Zeilennummer mit der Programmierung beginnen. Nachdem dann diese Programmzeile mit RETURN in den Programmspeicher des Rechners übernommen wurde, wird die nächste Zeilennummer im Zehnerabstand automatisch vom Rechner vorgegeben. Haben Sie also mit Zeile 100 angefangen, wird als nächstes die Zeilennummer 110 ausgegeben. Beginnen Sie dagegen mit Zeilennumer 65, so ist 75 die nächste Zeilennummer. Der Wert, der dem Befehl AUTO mit übergeben wird, gibt damit den Abstand zu den einzelnen Programmzeilen an.

Wollen Sie die automatische Zeilennumerierung wieder aufheben, so geben Sie nur den Befehl AUTO ohne einen entsprechenden Parameter ein.

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

RENUMBER

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 RENUMBER 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 d.h. neu berechnet werden.

Sie können ebenfalls nur einen bestimmten Abschnitt des Programms neu durchnumerieren, indem Sie

RENUMBER X,Y,Z

eingeben. Dabei steht 'X' für die Zeilennummer, mit der die neue Numerierung beginnen soll, 'Y' gibt die Schrittweite an (z.B. 10), und mit 'Z' bestimmen Sie die alte Startzeile, ab der das Programm neu numeriert werden soll. Der Befehl

RENUMBER 200,5,100

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

Nun kann es auch 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. Verwenden Sie diesen Befehl jedoch vorsichtig, denn es erfolgt keine Sicherheitsabfrage. Betätigen Sie also die RETURN-Taste, so sind die angegebenen Programmzeilen unwiderruflich verloren.

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 dann an der Stelle wieder fortfahren lassen, an der es mit dem STOP-Befehl unterbrochen wurde und zwar mit den aktuellen Variableninhalten. Der Befehl RUN setzt dagegen alle Variablen wieder auf Null, auch wenn Sie RUN mit einer Zeilennummer verwenden. Dies ist ein wichtiger Unterschied zum CONT-Befehl. Allerdings können Sie CONT nicht mehr anwenden, wenn das Programm mit einer Fehlermeldung ausgestiegen ist.

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.=*TR*ace *ON*) wird jede Zeilennummer beim Programmablauf vorher in eckigen Klammern ausgegeben. Die Ausgabe der Zeilennummern auf dem Bildschirm geschieht relativ schnell, so daß es unter Umständen günstiger sein kann, diese Ausgabe auf den Drucker umzuleiten. Man kann dann nachher das Ergebnis in Ruhe auf dem Papier analysieren. Der Befehl

TROFF

(engl.=*TR*ace *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.

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. Ein **wichtiger Unterschied** zum BASIC 2.0 soll nicht unerwähnt bleiben. Im BASIC 7.0 können Sie den LIST-Befehl auch innerhalb eines Programms einsetzen, ohne daß das Programm nach Ausführung von LIST abbricht. Dies ist z.B. bei Demonstrationsprogrammen wünschenswert. Sie haben damit später die Möglichkeit, Grafiken auf dem 40-Zeichenschirm anzuzeigen und gleichzeitig den entsprechenden Programmcode auf dem 80 Zeichenschirm einzublenden.

Bei der Erstellung Ihrer Programme wird es nicht ausbleiben, daß Ihnen Fehler unterlaufen. Dieses können sowohl Fehler logischer als auch syntaktischer Art sein. Bricht Ihr Programm nun mit einer Fehlermeldung ab, so wird durch die Eingabe von

HELP

die Programmzeile, in der der Fehler auftrat, angezeigt. Diese Zeile wird auf dem 40-Zeichenschirm revers und auf dem 80 Zeichenschirm unterstrichen dargestellt.
Mit diesen Editierbefehlen steht Ihnen somit ein komfortables Hilfsmittel für die Erstellung und Korrektur Ihrer Programme zur Verfügung.

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

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 in BASIC so stark eingeschränkt, daß man tatsächlich nur noch sehr einfache Probleme in Programme umsetzen könnte.

3.1 Unbedingte Programmsprünge

Die erste und vielleicht einfachste Art einer Programmverzweigung haben Sie bereits in dem Beispiel mit der internen Uhr des Commodore 128 kennengelernt. Es handelt sich dabei um den GOTO-Befehl. Dieser Befehl veranlaßt das Programm, von seinem eigentlichen Ablauf, der durch die Zeilennummern vorgegeben ist, abzuweichen. **Unbedingter Programmsprung** heißt er deswegen, weil er **an keine Bedingung geknüpft** ist, d.h. daß das Programm an dieser Stelle auf alle Fälle diesen Sprung ausführt.

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

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

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

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

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

Datenflußplan für Berechnung Idealgewicht

Bild 10

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, an der das Programm fortfahren 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

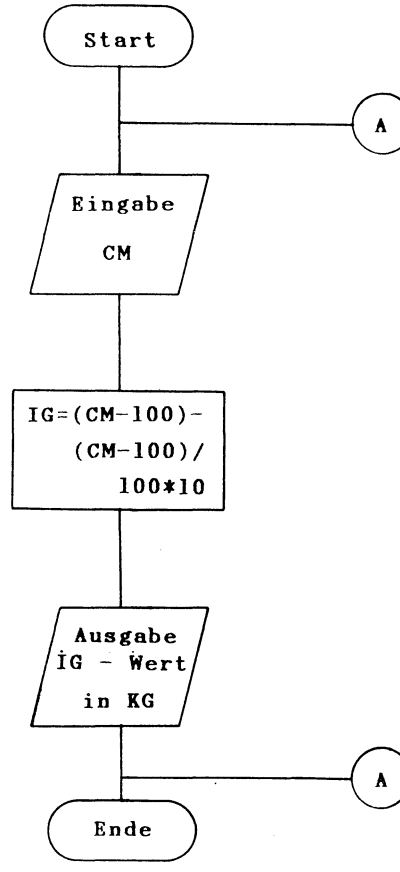


Bild 11

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...ELSE**-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 SCNCLR:PRINT
30 PRINT"GEBEN SIE ZWEI ZAHLEN FUER ";CHR$(17)
40 PRINT"DIE OBERE UND UNTERE GRENZE EIN"
50 INPUT"UNTERE GRENZE";U
60 INPUT"OBERE GRENZE";O
70 Z=INT((O+1)*RND(1))+U
80 INPUT"RATEN SIE";SZ
90 IF SZ < Z THEN 120
100 IF SZ > Z THEN 140
110 IF SZ = Z THEN 160
120 PRINT"DIE GEDACHTE ZAHL IST GROESSER"
130 GOTO 80
140 PRINT"DIE GEDACHTE ZAHL IST KLEINER"
150 GOTO 80
160 SCNCLR:PRINT "HURRA, SIE HABEN DIE ZAHL GEFUNDEN"
170 PRINT"WOLLEN SIE NOCHMAL JA/NEIN"
180 INPUT A$
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$(17) = Cursor wird eine Zeile nach unten gerückt

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

Programm fort. Haben Sie die Zahl erraten, so springt der Rechner in die Zeile 160. In Zeile 170 werden Sie gefragt, ob Sie das Spiel fortsetzen wollen. Geben Sie *JA* ein, so ist die *Bedingung* in Zeile 190 *erfüllt* und das *Programm beginnt von neuem.*

In den Zeilen 90 bis 110 wird der IF...THEN-Befehl also zum Vergleich zwischen der Spielerzahl (SZ) und der Zufallszahl (Z) benutzt. In Zeile 190 hingegen wird der IF...THEN-Befehl zum Vergleich einer Stringvariablen benutzt. Dabei ist zu beachten, daß, um die Bedingung **'wahr'** werden zu lassen, beide Strings absolut gleich sein müssen. Das bedeutet, daß auch Leerzeichen mit berücksichtigt werden müssen. Sie könnten in Zeile 180 also ruhig YES eingeben, trotzdem würde das Programm beendet, da nur dann in Zeile 20 verzweigt 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 Prorgamm 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 (*ELSE muß durch einen Doppelpunkt abgetrennt werden*). Das Programm verzweigt dann 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 und damit Speicherplatz ein.

3.2.2 BEGIN...BEND

Das BASIC 7.0 wurde um mehrere sehr nützliche Eigenschaften erweitert. Eine dieser Eigenschaften ist die **Befehlskombination BEGIN...BEND**. Durch die Befehle BEGIN...BEND besitzt der Programmierer nun die Möglichkeit, einen Block von Anweisungen und Befehlen zu kennzeichnen, der von der IF...THEN-Anweisung so behandelt wird, als würden diese Anweisungen alle in einer Zeile stehen. Die **Beschränkung** von IF...THEN, alles in **einer Zeile** unterzubringen, wird durch BEGIN...BEND also **aufgehoben**.

Diese Möglichkeit, alle Anweisungen einer IF...THEN-Abfrage innerhalb eines Blocks unterzubringen, erlaubt eine, bis dahin in BASIC nicht mögliche, *strukturierte Programmierung*. Das Programmlisting bzw. das Programm wird durch solche Blöcke **leichter 'lesbar'** und damit übersichtlicher, d.h. es können sich nicht so leicht logische Fehler einschleichen.
Die DO...LOOP-Anweisung unterstützt diese strukturierte Programmierung ebenfalls, wie wir später noch sehen werden.

Wir wollen nun als Beispiel zu BEGIN...BEND das Programm "Zahlenraten" entsprechend abändern. Es folgt zunächst das geänderte Programmlisting mit BEGIN...BEND:

```
10 REM ZAHLENRATEN
20 SCNCLR:PRINT
30 PRINT"GEBEN SIE ZWEI ZAHLEN FUER ";CHR$(17)
40 PRINT"DIE OBERE UND UNTERE GRENZE EIN"
50 INPUT"UNTERE GRENZE";U
60 INPUT"OBERE GRENZE";O
70 Z=INT((O+1)*RND(1))+U
80 INPUT"RATEN SIE";SZ
90 IF SZ <> Z THEN BEGIN
100 :  IF SZ > Z THEN PRINT "DIE GEDACHTE ZAHL IST KLEINER."
110 :  IF SZ < Z THEN PRINT "DIE GEDACHTE ZAHL IST GROESSER."
150 BEND:GOTO 80
160 SCNCLR: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 Programmzeilen des Programms wurden nicht verändert. Die erste Änderung hat die Zeile 90 erfahren. Hier wurde die IF...THEN-Anweisung entsprechend für BEGIN geändert. Der Block von BEGIN...BEND beginnt in Zeile 90 und endet in Zeile 150 mit BEND. Ist SZ nun ungleich Z, so werden die Anweisungen und Programmzeilen, die dem BEGIN folgen, ausgeführt. Wird der Befehl BEND erreicht, so wird die Anweisung ausgeführt, die dem BEND folgt, also GOTO 80. Ist die eingegebene Zahl SZ nun gleich Z, wird die nächste Programmzeile, die dem BEND folgt, ausgeführt. Der komplette Block BEGIN...BEND wird dabei übersprungen.

Sie können ja einmal ausprobieren, was geschieht, wenn Sie den Befehl GOTO 80 nicht direkt dem BEND folgen lassen, sondern ihn in einer separaten Programmzeile *(z.B. Zeile 155)* unterbringen. Merken Sie sich das Ergebnis gut. Sie könnten sonst später in Ihren eigenen Programmen böse Überraschungen erleben.

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!

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 die INSTR-Funktion nicht benutzen dürfen. Sie sollen also nebenbei eine Routine in BASIC schreiben, die die INSTR-Funktion ersetzt.

3.2.3 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 zunächst ein Beispielprogramm an.

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

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

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

Dazu einige Beispiele:

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

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

IF Z > 20 THEN 50

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

PRINT Z

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

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

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

Will man, daß die Schleife eine andere Schrittweite (*auch Inkrement genannt*) 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. Anstelle

der Zeitschleifen mit **FOR...NEXT** gibt es im BASIC 7.0 eine elegantere Möglichkeit. Mit dem Befehl

SLEEP

können Sie Ihren Commodore 128 für eine bestimmte Dauer in den "*Ruhestand*" schicken. So bewirkt **SLEEP 10**, daß der Rechner für 10 Sekunden mit der Fortführung des Programms wartet. Sie können SLEEP einen Höchstwert von 65535 zuordnen und damit Ihren 128er für gut 18 Stunden schlafen legen, falls Sie es wünschen. Soweit die Erläuterungen zum SLEEP-Befehl.

Die Zeitschleife in unserem Programm sollte nur die Verschachtelung von FOR..NEXT-Schleifen verdeutlichen. 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.

```
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  ---------+
```

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 Kommata getrennt werden. Das folgende Beispiel soll das wieder verdeutlichen.

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

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

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

Ergebnis ist dann ein Programmabbruch mit folgender Fehlermeldung:

?NEXT WITHOUT FOR ERROR IN *(Zeilennummer)*

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

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

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

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

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

Das Programm ergibt in dieser Kürze selbstverständlich keinen Sinn. Es soll auch nur aufzeigen, auf welche Art und Weise man eine Schleife vorzeitig verlassen kann. Normalerweise würde die Schleife bis zur 20 hochzählen. In Zeile 40 wird aber beim Erreichen für A=12 der Wert für A auf 20 gesetzt. Dadurch werden nur die Werte bis 12 ausgegeben. Diese Methode, die Laufvariable hochzusetzen, wird aber nur selten angewendet. Viel häufiger werden der Anfangswert und der Endwert einer Schleife in Variablen abgelegt. Dadurch lassen sich dann die Schleifen besser beeinflussen. Das folgende Beispiel soll das veranschaulichen.

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

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

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

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

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

3. Der Anfangswert darf bei positiver Schrittweite nicht größer als der Endwert sein, da sonst die Schleife nur einmal durchlaufen wird. Das gleiche gilt 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 7.0 des Commodore 128. Man darf Sie nicht verallgemeinern. Es gibt bei den verschiedenen BASIC-Dialekten gerade in der Benutzung der FOR...NEXT-Schleifen einige Unterschiede.

Befassen wir uns nun mit einer etwas anderen Art von Schleife. Es handelt sich hierbei um die bereits erwähnte DO...LOOP-Schleife.

3.2.4 Schleifen mit DO...LOOP

Mit der Kombination DO...LOOP 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 DO...LOOP **kein fest vorgegebenes Inkrement**, also keine feste Schrittweite.

Im einfachsten Fall wird der Beginn der Schleife mit **'DO'** und das Ende der Schleife mit **'LOOP'** gebildet. Alles, was innerhalb dieser Anweisungen steht, wird permanent ausgeführt. Auf diese Weise könnten wir aber nur wieder eine Endlosschleife mit DO...LOOP realisieren, wenn es da nicht den **Zusatz EXIT** gäbe. Mit EXIT haben Sie die Möglichkeit, die DO...LOOP-Anweisung zu verlassen. Trifft das Programm innerhalb einer DO...LOOP-Anweisung auf ein EXIT, so wird der nächste Befehl, der dem LOOP folgt, ausgeführt. Weiter oben wurde bereits erwähnt, daß mit der DO...LOOP-Anweisung Schleifen flexibler gestaltet werden können. Was das heißt, soll das nächste Beispiel verdeutlichen.

```
10 DO
20 :  A=INT(101)*RND(1))
30 :  Z=Z+1
40 :  IF A=100 THEN EXIT
50 LOOP
60 PRINT A,Z
70 END
```

Hier wird DO...LOOP solange durchlaufen, bis A zufällig den Wert 100 erreicht. Wird die Aussage in Zeile 40 wahr, dann wird der nächste Befehl, der dem LOOP folgt, ausgeführt. In unserem Fall ist das die Programmzeile 60. Dort werden dann A und der Zähler Z ausgegeben. Durch den Zähler erfahren Sie, wie oft die Schleife durchlaufen wurde. Die Zeilen 20 bis 40 beginnen mit einem Doppelpunkt, um die anschließenden Anweisungen

einrücken zu können. Dadurch erreicht man auch optisch eine Blockstruktur dieser Schleife.

Die Leistungsfähigkeit dieser Anweisung ist damit jedoch noch nicht erreicht. Zwei zusätzliche Anweisungen erlauben eine weitere Optimierung der Programmstruktur.

3.2.4.1 DO...LOOP mit UNTIL und WHILE

Mit der zusätzlichen Anweisung UNTIL können Sie die DO...LOOP-Schleife **solange** ausführen lassen, **bis** eine **Bedingung** bzw. ein logischer **Ausdruck wahr** wird. Das folgende Beispiel soll das wieder verdeutlichen.

```
10 DO UNTIL A=100
20 :  A=INT(101*RND(1))
30 :  Z=Z+1
50 LOOP
60 PRINT A,Z
70 END
```

Sie sehen, daß die Anweisung UNTIL direkt dem DO folgt. Damit ersetzt es die Programmzeile 40. Somit wird die Schleife jetzt **solange** durchlaufen, **bis** A den Wert 100 erreicht. Sie können die Anweisung UNTIL aber auch dem LOOP folgen lassen, so daß Sie auch wie folgt programmieren können:

50 LOOP UNTIL A=100

Das Ergebnis wird davon nicht beeinträchtigt.

Mit der Anweisung **WHILE** bleibt die DO...LOOP-Schleife **solange** wirksam, wie eine **Bedingung** bzw. ein **logischer Ausdruck wahr** ist. Unser kleines Beispielprogramm müßte mit **WHILE** demnach so aussehen:

```
10 DO WHILE A<100
20 :  A=INT(101*RND(1))
30 :  Z=Z+1
50 LOOP
60 PRINT A,Z
70 END
```

Jetzt wird die DO...LOOP-Schleife **solange** durchlaufen, **wie** A kleiner als 100 ist. Um mit **WHILE** das gleiche Ergebnis zu erhalten, mußte also in Zeile 10 die Abfrage von gleich in kleiner geändert werden.

Ich hoffe, daß Ihnen durch dieses kleine Beispiel die Wirkungsweise der DO...LOOP-Schleife in Verbindung mit den Anweisungen EXIT, UNTIL und WHILE klarer geworden ist.

Allgemein kann man also 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 DO...LOOP.

Auch hierzu haben wir schon eine Ausnahme im letzten Beispielprogramm des Kapitels 3.2.3 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. Auf den nächsten beiden Seiten folgen nun der Programmablaufplan sowie die dazugehörigen Erklärungen.

Programmablaufplan zum Programm "Berechnung Steuersatz"

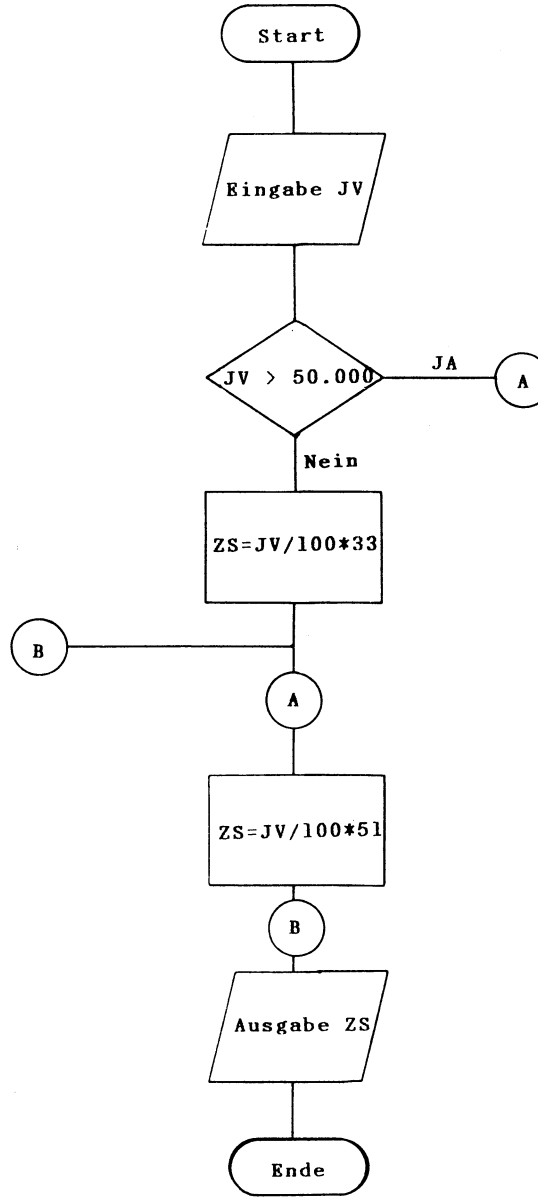


Bild 12

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, welches die Eingabe eines Wortes verlangt, in einen PAP übertragen. Auf den nächsten beiden Seiten folgt wiederum der Programmablaufplan mit den dazugehörigen Erläuterungen.

Programmablaufplan zum Programm auf Seite 130

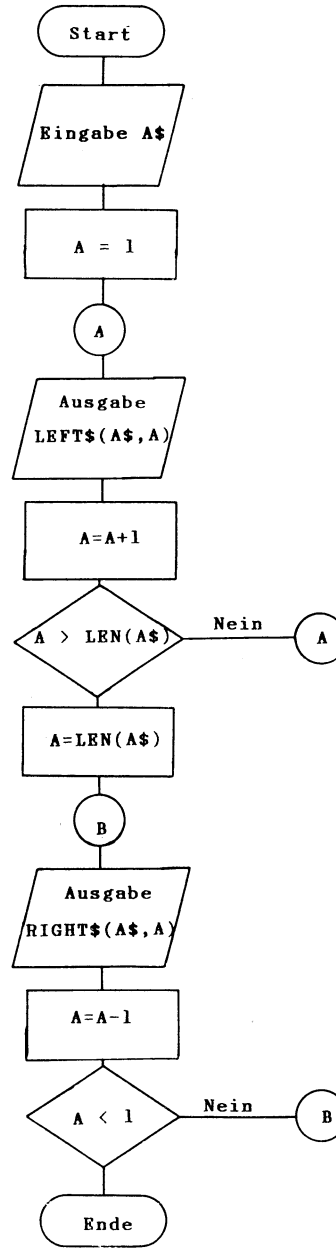


Bild 13

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

Wird der Zähler nun größer als LEN(A$), so tritt die zweite Schleife in Aktion. Die zweite Schleife besitzt die gleiche Anordnung der Symbole wie die erste Schleife. Die Konnektoren erhielten allerdings andere Bezeichnungen, um Verwechslungen auszuschließen. Der Ablauf ist jedoch der gleiche wie oben bei der ersten Schleife beschrieben.

Mit diesen Informationen sollten Sie nun in der Lage sein, selbständig Programmablaufpläne für jedes Programm zu erstellen. Es gilt wieder 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 es in bestimmte Programmzeilen verzweigen kann. Sicherlich könnte man das mit einigen IF...THEN-Vergleichen erreichen, jedoch ist dazu für jeden Vergleich ein Sprungbefehl für jede

Programmzeile notwendig. Ein einfaches Beispiel soll das verdeutlichen.

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

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

ON *(Variable)* **GOTO** *(Zeilennummer)*

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

?ILLEGAL QUANTITY ERROR

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

Beispiele:

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

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

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

Sie sehen, statt einer Variablen kann auch ein arithmetischer Ausdruck Verwendung finden. Der Vorteil dieses ON...GOTO-Befehls ist der, daß durch ihn mehrere IF...THEN-Befehle ersetzt werden können. Damit spart man Programmierarbeit und

auch Speicherplatz. Unser kleines Programm von vorhin hätte dann die folgende Form:

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

Wir haben also bei diesem kleinen Programm schon 3 Programmzeilen eingespart. Bei größeren Programmen, in denen 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 bestimme Programmteile dar, in denen meistens spezielle Aufgaben übernommen werden. Daher ist es von Vorteil, sie durch "glatte" Zeilennummern zu kennzeichnen.

Das kann z.B. in Hunderterschritten geschehen, wie in unserem Beispielprogramm. Dadurch erreicht man eine gewisse Übersichtlichkeit der einzelnen Programmabschnitte. Das Programm wird leichter lesbar.

3.3.1 Beispielprogramm "Rechenlehrgang"

Wir haben zum jetzigen Zeitpunkt schon einen relativ großen Befehlsumfang von BASIC kennengelernt. Das ist ein Grund, um sich an ein größeres Projekt zu wagen. Stellen Sie sich vor, Sie wollen für Ihre Kinder einen Rechenlehrgang für die vier Grundrechenarten auf dem Commodore 128 realisieren. Dieser Lehrgang soll folgende Eigenschaften aufweisen:

1. Auswahl einer der vier Grundrechenarten oder Programmende.

2. Stellen einer Aufgabe, die in höchstens drei Versuchen gelöst werden muß.

3. Nach dem dritten Fehlversuch soll das Ergebnis angezeigt werden.

4. Eingabe der größten Zahl, mit der in den einzelnen Aufgaben gerechnet werden soll.

5. Nach jeder Aufgabe soll eine Abfrage erfolgen, ob weitere Aufgaben der gleichen Rechenart gelöst werden sollen.

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

so geben Sie folgenden Befehl in den Rechner, nachdem Sie das Programm eingegeben haben.

SAVE"RECHENLEHRGANG"

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

PRESS RECORD & PLAY ON TAPE

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

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

DSAVE"RECHENLEHRGANG"

oder

SAVE"RECHENLEHRGANG",8

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

Im folgenden nun das Programmlisting für den **Rechenlehrgang**.

```
5 REM ***** MENUE *****
1Ø SCNCLR:F=Ø
2Ø PRINT
3Ø PRINT TAB(12)"RECHENLEHRGANG"
4Ø PRINT:PRINT
5Ø PRINT TAB(12)"WAEHLEN SIE:"
6Ø PRINT
7Ø PRINT TAB(12)"FUER ADDITION EINE 1"
8Ø PRINT
9Ø PRINT TAB(12)"FUER SUBTRAKTION  EINE 2"
1ØØ PRINT
11Ø PRINT TAB(12)"FUER DIVISION EINE  3"
12Ø PRINT
13Ø PRINT TAB(12)"FUER MULTIPLIKATION EINE 4"
133 PRINT
135 PRINT TAB(12)"FUER ENDE EINE 5"
14Ø PRINT
15Ø PRINT TAB(12);:INPUT"WELCHE ZAHL";Z
16Ø IF Z < 1 OR Z > 5 THEN 1Ø
17Ø ON Z GOTO 2ØØ,6ØØ,1ØØØ,13ØØ,16ØØ
2ØØ REM **********
21Ø REM  ADDITION
22Ø REM **********
23Ø SCNCLR
24Ø PRINT TAB(1Ø)"GEBEN SIE DIE GROESSTE ZAHL"
25Ø PRINT
26Ø PRINT TAB(1Ø)"FUER DIE ADDITION EIN."
27Ø PRINT
29Ø PRINT TAB(1Ø);:INPUT"GROESSTE";GR
299 REM
3ØØ REM ERZEUGEN ZUFALLSZAHLEN
3Ø1 REM
31Ø A1=INT(GR*RND(1))+1
32Ø A2=INT(GR*RND(1))+1
329 REM
33Ø REM BERECHNUNG ERGEBNIS
331 REM
34Ø EG=A1+A2
35Ø SCNCLR
36Ø PRINT
```

```
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2  "= ";
380 INPUT ES
390 IF ES=EG THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0: GOTO450
400 PRINT:PRINT TAB(10)"FALSCH !"
410 SLEEP 2
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 SLEEP 2
460 PRINT TAB(5)"DAS ERGEBNIS LAUTET" EG
470 SLEEP 3
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 SCNCLR
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 SCNCLR
760 PRINT
770 EG=A1-A2
780 PRINT"WIEVIEL ERGIBT" A1 "-" A2 "= ";
790 INPUT ES
800 IF ES=EG THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:GOTO860
810 PRINT:PRINT TAB(10)"FALSCH !"
```

```
82Ø SLEEP 2
83Ø F=F+1
84Ø IF F<=2 THEN 75Ø
85Ø PRINT
86Ø SLEEP 2
87Ø PRINT TAB(5)"DAS ERGEBNIS LAUTET";EG
88Ø SLEEP 3
89Ø PRINT TAB(5)"NOCH EINE AUFGABE J/N";
9ØØ INPUT A$
91Ø IF A$="J" THEN F=Ø:GOTO 71Ø
92Ø GOTO 1Ø
1ØØØ REM **********
1ØØ1 REM  DIVISION
1ØØ2 REM **********
1Ø1Ø SCNCLR
1Ø2Ø PRINT TAB(1Ø)"GEBEN SIE DIE GROESSTE ZAHL "
1Ø3Ø PRINT
1Ø4Ø PRINT TAB(1Ø)"FUER DIE DIVISION EIN."
1Ø5Ø PRINT
1Ø7Ø PRINT TAB(1Ø);:INPUT"GROESSTE";GR
1Ø79 REM
1Ø8Ø REM ERZEUGEN ZUFALLSZAHLEN
1Ø81 REM
1Ø9Ø A1=INT(GR*RND(1))+1
11ØØ A2=INT(GR*RND(1))+1
11Ø9 REM
111Ø REM BERECHNUNG ERGEBNIS
1111 REM
112Ø EG=A1*A2
113Ø SCNCLR
114Ø PRINT
115Ø PRINT"WIEVIEL ERGIBT" EG "/" A1 "= ";
116Ø INPUT ES
117Ø IF ES=A2 THEN PRINT:PRINT TAB(1Ø)"RICHTIG !":F=Ø:GOTO 124Ø
118Ø PRINT:PRINT TAB(1Ø)"FALSCH !"
119Ø SLEEP 2
12ØØ F=F+1
121Ø IF F<=2 THEN 113Ø
122Ø PRINT
123Ø SLEEP 2
```

```
1240 PRINT TAB(5)"DAS ERGEBNIS LAUTET" A2
1250 SLEEP 3
1260 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
1270 INPUT A$
1280 IF A$="J" THEN F=0:GOTO 1090
1290 GOTO 10
1300 REM ****************
1301 REM  MULTIPLIKATION
1302 REM ****************
1310 SCNCLR
1320 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
1330 PRINT
1340 PRINT TAB(10)"FUER DIE MULTIPLIKATION EIN."
1350 PRINT
1370 PRINT TAB(10);:INPUT"GROESSTE";GR
1379 REM
1380 REM ERZEUGEN ZUFALLSZAHLEN
1381 REM
1390 A1=INT(GR*RND(1))+1
1400 A2=INT(GR*RND(1))+1
1409 REM
1410 REM BERECHNUNG ERGEBNIS
1411 REM
1420 EG=A1*A2
1430 SCNCLR
1440 PRINT
1450 PRINT"WIEVIEL ERGIBT" A1 "*" A2 "= ";
1460 INPUT ES
1470 IF ES=EG THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:GOTO 1540
1480 PRINT:PRINT TAB(10)"FALSCH !"
1490 SLEEP 2
1500 F=F+1
1510 IF F<=2 THEN 1430
1520 PRINT
1530 SLEEP 2
1540 PRINT TAB(5)"DAS ERGEBNIS LAUTET" EG
1550 SLEEP 3
1560 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
1570 INPUT A$
1580 IF A$="J" THEN F=0:GOTO 1390
```

```
159Ø GOTO 1Ø
16ØØ SCNCLR
161Ø 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 Zeile 410 wird eine Zeitschleife abgearbeitet, damit der Anwender die Meldung lesen kann. Man hätte hier auch den Befehl **SLEEP 2** verwenden können, jedoch wollen wir vorerst zur Übung nach Möglichkeit mit FOR...NEXT-Schleifen arbeiten.
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 Zwischen-speicherung* 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 EG 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. Soviel wäre 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 die **TRAP-Funktion** besprechen. Diese ähnelt in gewisser Weise dem ON...GOTO-Befehl. Zunächst seien die wichtigsten Grundregeln bei der Verwendung des ON...GOTO-Befehls jedoch noch einmal zusammengefaßt:

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

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

3. Werte kleiner als Null oder größer als 255 bewirken einen Programmabbruch mit der Fehlermeldung:

   ?ILLEGAL QUANTITY ERROR IN (Zeilennummer).

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

3.3.2 Programmsprünge mit TRAP

Diese spezielle Art des ON...GOTO-Befehls wird eingesetzt, um innerhalb eines Programms auftretende Fehler selbst per Programm zu behandeln bzw. zu verwalten. Dazu wird meistens die TRAP-Funktion am Anfang eines Programms untergebracht. Der Rechner 'merkt' sich die Stelle, an der der Befehl zum ersten Mal auftritt. Alle Fehler bzw. Fehlermeldungen, die nach diesem Befehl auftreten, veranlassen das Programm, zu der Zeilennummer zu springen, die der TRAP-Funktion folgt. Sie können hier allerdings immer nur eine Zeilennummer angeben und nicht mehrere, wie Sie es vom ON...GOTO-Befehl her kennen. Ab dieser Zeilennummer kann dann ein kleines Programm (*Fehlerroutine*) 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 7.0 des Commodore 128 zwei Variablen (sogenannte Systemvariablen), die daraufhin abgefragt werden können. Es handelt sich dabei um die Variablen

ER

und

EL.

Geben Sie z.B. statt *PRINT*

PRILT

in den Rechner und betätigen die RETURN-Taste, so erhalten Sie die Fehlermeldung

?SYNTAX ERROR.

Geben Sie jetzt

PRINT ER

ein, so wird die Fehlernummer des Fehlers angezeigt. In unserem Falle wäre das die Fehlernummer 11. Im Anhang finden Sie sämtliche Fehlernummern sowie die entsprechenden Fehlerbeschreibungen aufgelistet. In Ihren Programmen können Sie dann gemäß dem Wert von **'ER'** eine eigene Fehlermeldung ausgeben, ohne daß das Programm ungewollt abbricht. Mit der Variablen **'EL'** haben Sie zusätzlich noch die Möglichkeit, die Zeilennummer abzufragen, 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 Auftreten eines Fehlers erst alle Dateien geschlossen werden (kein Datenverlust), bevor das Programm dann die eigene Fehlermeldung ausgibt und danach beendet wird.

Oft ist es jedoch gar nicht notwendig, das Programm zu beenden. Es reicht dann volkommen aus, eine Fehlermeldung für den Anwender auszugeben und dann mit dem Programm fortzufahren. Für solch eine Fall existiert die Anweisung

RESUME.

Mit

RESUME *(Zeilennummer)*

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

```
10 TRAP 1000
20 FOR I=-1 TO 3
30 ON I GOTO 200,300
40 NEXT I
100 END
200 SCNCLR
210 PRINT "ZEILE 210"
220 SLEEP 3
230 GOTO 40
300 SCNCLR
310 PRINT "ZEILE 310"
320 SLEEP 3
330 GOTO 40
1000 REM FEHLERBEHANDLUNG
1010 SCNCLR
1020 PRINT "FEHLER";ER;" IN ZEILE";EL
1030 SLEEP 3
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' mit ON...GOTO in bestimmte Zeilen verzweigt 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 die **RESUME**-Anweisung, 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 Funktionen doch recht einfach. Sie können in Ihren Programmen selbstverständlich den Fehler noch genauer beschreiben, indem Sie entsprechend dem Wert von 'ER' zusätzlich einen beschreibenden Text mit ausgeben lassen.

Sie haben aber auch die Möglichkeit, innerhalb eines Programms die '*Originalfehlermeldung*' des Commodore 128 ausgeben zu lassen. Die Funktion

ERR$(X)

gibt Ihnen gemäß dem Wert von 'X' die dazugehörige Originalmeldung aus. Da Ihnen insgesamt 41 Meldungen zur Verfügung stehen, müssen die Werte von X im Bereich zwischen 1 und 41 liegen. Das folgende kleine Programm gibt Ihnen alle zur Verfügung stehenden Fehlerstrings aus:

```
10 FOR I=1 TO 41
20 PRINT ERR$(I)
30 NEXT I
40 END
```

Im vorletzen Programm können Sie ja einmal die Zeile 1020 gegen die nachfolgende Zeile austauschen.

```
1020 PRINT "?";ERR$(ER);" ERROR IN ZEILE";EL
```

Wirkt doch täuschend echt, oder?

Auf der nächsten Seite habe ich nun wieder einige Aufgaben für Sie zusammengestellt, deren Lösungen Sie dann am Ende des Buches finden können. Berücksichtigen Sie auch diesmal wieder die am Anfang des Buches aufgeführten fünf Punkte der Programmierung. Und nun wie gehabt viel Erfolg beim Lösen der Aufgaben!

Aufgaben

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

2. Schreiben Sie ein Programm, welches Ihnen die reellen Nullstellen einer quadratischen Gleichung der Form:

   $AX^2+BX+C=0$

   berechnet. Die Lösungen der Aufgabe erhalten Sie durch folgende Formel:

   x1 = (-B + SQR(B2-4AC))/2A
   x2 = (-B - SQR(B2-4AC))/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:SCNCLR
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.

3.4 Programmablaufsteuerung mit GET

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

?REDO FROM START

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

3.4.1 Eingabe von Daten mit GET

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

?ILLEGAL DIRECT ERROR

aus. Wie arbeitet nun der GET-Befehl?

Dazu müssen wir zunächst wissen, daß der Commodore 128 einen sogenannten *Tastaturpuffer* besitzt. Es handelt sich dabei um einen kleinen Speicher, in dem bis zu **10 Zeichen** zwischengespeichert werden können. Praktisch hat das folgende Bedeutung:
Ist der Computer mit der Ausführung einer Operation beschäftigt, so können Sie während dieser Zeit maximal 10 Zeichen über die Tastatur eingeben, die der Computer dann sofort nach Beendigung der Operation benutzt. Geben Sie zur Verdeutlichung dieses Vorgangs das folgende "Programm" in den Rechner:

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

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

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

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

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

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

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

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

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

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

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

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

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

```
10 REM ANZEIGE DER ASCII-WERTE
20 SCNCLR
30 GET A$:IF A$="" THEN 30
40 PRINT TAB(6)A$ TAB(12)ASC(A$)
50 REM ALTER ZUSTAND BILDSCHIRM
60 PRINT CHR$(153) CHR$(142)
70 GOTO 30
```

Die Zeilen 10 bis 30 dürften von der Funktion her bekannt sein. Interessant ist die Zeile 40. Hier wird auf die 6. Position der Zeile das Zeichen selbst mit A$ ausgegeben. Sie sehen nur dann ein Zeichen, wenn es sich auch um darstellbare Zeichen im Normalmodus handelt. Wechseln Sie z.B. die Farbe der Schrift im Normalmodus mit CTRL und 2 in weiß, so wird auf dem Bildschirm ja auch kein Zeichen ausgegeben. Der zweite Teil der Zeile gibt schließlich den ASCII-Wert von A$ aus. Zeile 60 stellt den alten Zustand der Schriftfarbe *(CHR$(153))* wieder her und schaltet wieder auf den Groß/Grafikmodus *(CHR$(142))* um. Zeile 70 springt dann wieder zur Zeile 30.

Sie werden sich vielleicht gefragt haben, ob es im BASIC 7.0 nicht möglich ist, bei GET die zusätzliche Abfrage auf das Leerzeichen zu umgehen?

Bei GET gibt es leider keine Möglichkeit, diese Abfrage zu ersetzen. Allerdings bietet sich mit der Funktion GETKEY eine komfortablere Lösung an.

3.4.2 Tastaturabfrage mit GETKEY

GETKEY arbeitet **ähnlich** wie **GET**, jedoch wartet **GETKEY** selbständig solange, **bis** eine **Taste gedrückt** wird. Damit fällt die zusätzliche Prüfung, die bei GET unbedingt notwendig ist, fort. Weiterhin haben Sie mit GETKEY die Möglichkeit, mehrere einzelne Zeichen auf einmal einzulesen. Das sieht dann wie folgt aus:

```
10 GETKEY A$,B$
20 PRINT A$,B$
```

Dieses Programm wartet in Zeile 10 auf das Betätigen zweier Tasten und ordnet den Variablen A$ und B$ die entsprechenden Zeichen zu. Zeile 20 zeigt diese Zeichen schließlich auf dem Bildschirm an.

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

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

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

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

Sie haben nun einige Anwendungsbeispiele mit GET bzw. GETKEY kennengelernt. Die wichtigste Eigenschaft ist wohl die der Selektierung der einzelnen Tasten. Dadurch kann man bei eigenen Programmen die Menüsteuerung fast narrensicher gestalten. Wir wollen nun in unserem Rechenlehrgang den INPUT-Befehl durch den GET-Befehl ersetzen. Auf der nächsten Seite sehen Sie, wie die ersten beiden INPUT-Befehle durch die GET-Befehle ersetzt wurden. Selbstverständlich können Sie in den entsprechenden Programmzeilen statt GET auch GETKEY verwenden. Studieren Sie genau, welche Abfragen mit IF...THEN durchgeführt wurden, um nur bestimmte

Zeichen bei der Eingabe zuzulassen. Hier hat es sich schon bezahlt gemacht, daß wir in Zehnerschritten programmiert haben. So fällt es uns nicht schwer, die zusätzlichen Zeilen, die für den GET-Befehl notwendig sind, in das Programm einzufügen. Nachfolgend nun die ersten abgeänderten Programmzeilen bis Zeile 320.

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

```
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
 .
 .
 .
```

Wie Sie erkennen können, hat sich als erstes in den Zeilen 150 bis 170 etwas verändert. Im ersten Programmteil stand in Zeile 150 der INPUT-Befehl mit zusätzlicher Textausgabe. Der INPUT-Befehl wurde aus dieser Zeile herausgenommen und gegen den PRINT-Befehl, der den Anwender zur Eingabe einer Zahl auffordert, ersetzt. In Zeile 155 wurde nun der GET-Befehl mit der bekannten Abfrage auf einen Leerstring untergebracht. Zeile 160 hat sich insofern verändert, daß wir die VAL-Funktion benutzt haben, um den String A$ in eine Zahl umzuwandeln und um ihn dadurch auf kleiner als 1 oder größer als 5 überprüfen zu können. Das gleiche Prinzip wurde in der nächsten Zeile verwendet, um das Programm durch ON...GOTO entsprechend verzweigen lassen zu können. Es wird ja mit VAL(A$) ein Wert zwischen 1 und 5 ermittelt, der jetzt genauso benutzt wird, als ob dort die Zahl selbst oder eine Variable gestanden hätte.

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

Einen kleinen Makel hat unsere GET-Routine noch; wir sehen nicht, **wo** wir auf dem Bildschirm die **Daten** eingeben und vor allem nicht **welche Daten** wir eingeben, da diese nicht auf dem Bildschirm angezeigt werden. Dieses Problem trat beim INPUT-Befehl nicht auf. Dort hatten wir immer unseren blinkenden Cursor, der genau unsere Position angab, und wir sahen auch, welche Daten wir bis zum Drücken der RETURN-Taste einge-

geben hatten. Dies müssen wir also selber in die Hand nehmen. Wie das zu machen ist, soll in einem späteren Kapitel gezeigt werden.

3.4.3 Die Belegung der Funktionstasten

Der Commodore 128 besitzt, wie auch sein kleiner Bruder, der Commodore 64, vier Funktionstasten mit den Bezeichnungen **F1** bis **F8**. Im Gegensatz zum Commodore 64 sind diese Funktionstasten aber bereits mit Befehlen belegt. Geben Sie den Befehl

KEY

in den Computer, so erhalten Sie die folgende Aufstellung:

```
KEY 1,"GRAPHIC"
KEY 2,"DLOAD"+CHR$(13)
KEY 3,"DIRECTORY"+CHR$(13)
KEY 4,"SCNCLR"+CHR$(13)
KEY 5,"DSAVE"+CHR$(13)
KEY 6,"RUN"+CHR$(13)
KEY 7,"LIST"+CHR$(13)
KEY 8,"MONITOR"+CHR$(13)
```

Das sind also die voreingestellten Befehle für die Funktionstasten. Was aber, werden Sie fragen, hat der CHR$(13)-Code für eine Bedeutung?

Nun, das ist relativ schnell erklärt. CHR$(13) ist nichts anderes als der Code für die RETURN-Taste. Wird dieser Code nun zusätzlich an einen Befehl angehängt, so wird dieser Befehl sofort ausgeführt. Betätigen Sie also die Funktiontaste F7, so wird augenblicklich Ihr Programm aufgelistet.

Sie können aber den Funktionstasten auch andere Befehle oder Funktionen zuordnen, indem Sie KEY gefolgt von der Nummer der Funktionstaste und einem Befehl in den Rechner eingeben. Das folgende Beispiel definiert für die Funktionstaste F6 die

ESC-Sequenz zum Umschalten zwischen dem 40-Zeichen- und dem 80-Zeichenmonitor.

```
KEY 6,CHR$(27)+"X"+CHR$(13)
```

Haben Sie noch BASIC-Programme vom Commodore 64, in denen die speziellen Codes der 64er-Funktionstasten benutzt werden, so können Sie durch das nachfolgende Programm die Funktionstasten des Commodore 128 mit diesen Codes belegen.

```
10 REM 64ER FUNKTIONSTASTEN AUF CBM 128
20 FOR I=1 TO 8
30 KEY I,CHR$(132+I)
40 NEXT I
50 END
```

Nachdem Sie dieses Programm gestartet haben, besitzen die Funktionstasten des Commodore 128 die gleichen Codes wie die Funktionstasten des Commodore 64. Ihre BASIC-Programme reagieren jetzt in Bezug auf die Betätigung der Funktionstasten wie auf dem Commodore 64. Behalten Sie diese Tastenbelegung für das nächste Kapitel bei.

3.4.4 Funktionstastenabfrage mit GET

Einige weitere Beispiele mit GET sollen die Anwendung dieses Befehls noch mehr verdeutlichen. Der Commodore 128 besitzt, wie bereits im vorigen Kapitel angesprochen wurde, rechts oben auf der Tastatur die vier Funktionstasten F1 bis F8. Der Anfänger steht meistens ziemlich ratlos davor, wie er diese Tasten in seinen Programmen vernünftig einsetzen soll. Dabei kann man diesen Tasten nicht nur eine, sondern gleich mehrere Funktionen auferlegen. Das folgende kleine Programm soll dazu dienen, Ihnen Anregungen in dieser Richtung zu geben. Ihrer Phantasie sind da kaum Grenzen gesetzt.

```
10 REM FUNKTIONSTASTENBELEGUNG MIT GET
20 GET A$:IF A$="" THEN 20
30 REM F1
40 IF A$=CHR$(133) THEN SCNCLR
50 REM F2
60 IF A$=CHR$(137) THEN PRINT CHR$(144);"F2"
70 REM F3
80 IF A$=CHR$(134) THEN PRINT CHR$(5);"F3"
90 REM F4
100 IF A$=CHR$(138) THEN PRINT CHR$(153):END
110 GOTO 20
```

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

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

```
10 REM REAKTIONSTEST
20 FOR I=1 TO 11:CU$=CU$+CHR$(17):NEXT I
30 FOR I=1 TO 19:CU$=CU$+CHR$(29):NEXT I
40 SCNCLR
50 FOR I=0 TO INT(10000*RND(1))+1:NEXT I
60 PRINT CHR$(7);CU$;CHR$(209):TI$="000000"
70 GETKEY A$
80 PRINT CHR$(19)TI/60;"SEC."
90 GET A$:IF A$ < > CHR$(133) THEN 90
100 GOTO 40
```

In den Zeile 20 und 30 wird mittels zweier FOR...NEXT-Schleifen ein String erzeugt. Dieser String beinhaltet 11mal das Steuerzeichen, um den Cursor nach unten zu bewegen, und 19mal das Steuerzeichen, um den Cursor nach rechts zu bewegen. Dadurch wird in Zeile 60 erreicht, daß der Punkt (CHR$(209)) genau in der Mitte des Bildschirms auftaucht. Durch die Ausgabe von CHR$(7) wird zusätzlich eine akustische Meldung erreicht. In Zeile 50 wird die Verzögerungszeit bis zum Erscheinen des Punktes zufällig zwischen einer bis ca. 10 Sekunden generiert. Zeile 60 setzt dann die "Uhr" auf Null, damit man durch Auslesen von TI die Zeit bis zum Drücken einer Taste berechnen kann. Die restlichen Programmpunkte dürften zu diesem Zeitpunkt keine Schwierigkeiten mehr bereiten. Das soll nun vorerst an Beispielen für die Verwendung von GET bzw. GETKEY reichen. Diese Funktionen werden Ihnen sowieso noch in anderen Programmen oft genug begegnen, wo Sie dann deren Verwendung genau analysieren können. Schauen wir uns nun noch einige Befehle bzw. Funktionen an, welche in Programmen nicht so häufig Verwendung finden. Sie sollten jedoch wissen, mit "*wem*" Sie es zu tun haben, wenn Ihnen ein solcher Befehl in einem Programm einmal begegnet.

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

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

FRE

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

FRE(X)

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

PRINT FRE(X)

Befindet sich kein Programm im Programmspeicher, so erhalten Sie als Augabe den Wert 58109. X darf nur die Werte 0 oder 1 annehmen, da hierdurch direkt die beiden Speicherbänke über den verbleibenden Restspeicher abgefragt werden. Mit FRE(0) fragen Sie BANK 0 (Programmspeicher) und mit FRE(1) BANK 1 (Variablenspeicher) ab.

POS

Diese Funktion wird Ihnen innerhalb eines Programms sehr selten begegnen. Mit ihr kann man die aktuelle Cursorposition innerhalb einer Programmzeile erfahren. Damit kann sie Werte zwischen 0 und 79 annehmen. Folgende Beispiele sollen die

Funktion näher erläutern. Geben Sie im Direktmodus folgendes in den Rechner ein:

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

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

TEST 4 TEST A 13

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

SYS

Mit dieser Anweisung können Sie an eine Adresse im Rechner springen, an der ein eigenes Maschinenprogramm oder aber eine Routine des Rechners beginnt. Die Kontrolle des Mikroprozessors wird dann nicht mehr durch die BASIC-Befehle über den Interpreter gesteuert, sondern direkt durch den Maschinencode, der sich nur noch aus Zahlenwerten zusammensetzt. Sie können mit SYS im Prinzip jede Speicherstelle des Rechners ansprechen. Bei vielen Adressen "stürzt" der Rechner jedoch ab, d.h. er bleibt in seinen eigenen Routinen hängen, da nicht an der richtigen Stelle eingesprungen wurde. Diese Anweisung setzt also gewisse Kenntnisse des Betriebssystems voraus. Sie können bis zu vier Parameter für Akku, X-, Y- und Statusregister mit übergeben.

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

SYS 4*4096

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

USR(X)

Diese Funktion arbeitet ähnlich wie SYS, nur mit dem Unterschied, daß hier nur ein Wert mit an das Maschinenunterprogramm übergeben werden kann. Die Benutzung dieser Funktion setzt schon fortgeschrittene Kenntnisse in BASIC und in Maschinenprogrammierung voraus, so daß sich der Anfänger in Sachen BASIC damit vorerst nicht zu beschäftigen braucht.

WAIT

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

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

3.6 PEEK und POKE

PEEK

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

PEEK(*Adresse*)

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

```
PRINT PEEK(1024)
```

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

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

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

```
PRINT X
```

Sie erhalten dann auf dem Bildschirm wieder den Wert 32. Schauen wir uns einmal die ersten 78 Werte des BASIC-Speichers an, der bei Adresse 7168 beginnt, sofern noch keine hochauflösende Grafik benutzt wurde. Haben Sie nach dem Einschalten schon eine Grafikseite angewählt, verschiebt sich der BASIC-Anfang automatisch um 9 KByte nach oben zur Adresse 16384. Geben Sie nun das folgende Programm in den Rechner ein:

```
10 REM 78 BASICSPEICHERSTELLEN
20 FOR I=7168 TO 7246
30 PRINT PEEK(I);
40 NEXT I
50 END
```

Starten Sie das Programm nun in der gewohnten Weise. Auf dem Bildschirm erscheint eine Zahlenfolge von 78 Zahlen. Ohne weiter in die Materie eindringen zu wollen, sei hier nur gesagt, daß Sie nun Ihr BASIC-Programm so sehen, wie es der Commodore 128 intern abspeichert. Ist doch recht interessant, nicht wahr?

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

POKE

Dieser Befehl ist das genaue **Gegenteil** vom PEEK-Befehl. Mit POKE schreiben Sie einen Wert in eine Speicherstelle. Der POKE-Befehl wurde beim Commodore 64 z.B. dazu benutzt, um den Farbspeicher für den Bildschirmrahmen und den Bildschirmhintergrund zu verändern. Dabei ist 53280 die Speicherstelle für den Bildschirmrahmen und 53281 für den Bildschirmhintergrund. Diese Speicherstellen stimmen mit denen beim Commodore 128 überein. Sie müssen vorher allerdings mit

BANK 15

die richtige Speicherkonfiguration einstellen, da sich der POKE-Befehl sonst auf die Bank für den Programmspeicher bezieht. Wollen Sie die Farbe des kompletten Bildschirms in schwarz umändern, so geben Sie folgende POKE-Befehle in den Rechner ein:

POKE 53280,0:POKE 53281,0

und betätigen Sie wieder die RETURN-Taste. Der gesamte Bildschirm wird dabei schwarz. Die Schrift besitzt immer noch die gleiche Farbe. Diese können Sie mit der CTRL- und der entsprechenden Farbtaste (1-8) verändern, oder auch mit dem POKE-Befehl. Sie können jetzt wieder durch Eingabe von BANK 0 die ursprüngliche Konfiguration einstellen.

Es existiert eine Speicherstelle, in der Sie durch Hineinschreiben der Werte 0 bis 15 die Farbe der Schrift bestimmen können. Es handelt sich hierbei um die Speicherstelle 241. Führen Sie den folgenden Befehl aus:

POKE 241,1

Die Farbe des Cursors ist nun weiß, und ebenso alles, was Sie neu auf den Bildschirm schreiben. Die Farbwerte 0=schwarz, 1=weiß usw. entnehmen Sie bitte dem Handbuch zu Ihrem Commodore 128.

Im vorigen Kapitel wurde schon angedeutet, daß der WAIT-Befehl dazu benutzt werden kann, die Betätigung einer Taste zu registrieren. Hierzu wird die Speicherstelle 208 benötigt. In dieser Speicherstelle wird registriert, wie viele Tasten gedrückt wurden. Das folgende kleine Programm soll Ihnen eine Anwendung des WAIT-Befehls zeigen.

```
10 REM TASTATURABFRAGE MIT WAIT
20 POKE 208,0
30 WAIT 208,1
40 PRINT "TASTE GEDRUECKT"
50 END
```

Zeile 20 setzt die Adresse 208 auf Null, was dann soviel bedeutet, daß keine Taste gedrückt wurde. Das Programm wartet nun durch den WAIT-Befehl in Zeile 30 solange, bis die Speicherstelle 208 den Wert eins annimmt. Das ist genau dann der Fall,

sobald eine Taste gedrückt wird. Danach fährt das Programm in seiner Ausführung fort und gibt die Meldung aus:

"TASTE GEDRUECKT".

Damit haben Sie eine Alternative zur Tastaturabfrage mit GET oder GETKEY kennengelernt. Dem POKE-Befehl kann natürlich auch eine Variable folgen, deren Wert dann in die Speicherstelle geschrieben wird. Der Wert darf allerdings nicht größer als 255 sein.

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

3.7 READ, DATA und RESTORE

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

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

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

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

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

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

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

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

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

```
10          20          30
```

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

```
30 DATA 10,20,30
        ↑
```

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

```
30 DATA 10,20,30
          ↑
```

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

?OUT OF DATA ERROR IN (*Zeilennumer*)

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

RESTORE

Der **RESTORE-Befehl setzt** den Zeiger **zurück** auf das erste Element einer DATA-Zeile. Damit haben Sie jetzt die Möglichkeit, die Daten in den DATA-Zeilen **beliebig oft auslesen** zu lassen. Geben Sie zunächst das folgende Programm ein, um ein-

mal zu sehen, was geschieht, wenn das Programm versucht, mehr Daten zu lesen als vorhanden sind.

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

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

?OUT OF DATA ERROR IN 40

In Zeile 40 wurde nämlich versucht, ein viertes Element der DATA-Zeile zu lesen, welches ja nicht vorhanden ist. Um diesen Fehler auszuschalten, könnte man entweder weitere drei Elemente an die DATA-Zeile anhängen, oder aber einfach den Zeiger mit RESTORE zurücksetzen. Probieren wir das einmal aus. Geben Sie dazu folgende Zeile in den Rechner ein und betätigen Sie danach die RETURN-Taste.

```
35 RESTORE
```

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

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

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

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

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

READ A,B,C

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

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

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

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

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

?TYPE MISMATCH 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 Programm, 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

?TYPE MISMATCH ERROR IN 20

ab. Daher achten Sie bei Verwendung solcher Variablen-kombinationen hinter READ darauf, daß die Datentypen in den

DATA-Zeilen auch den geeigneten Variablentypen zugeordnet werden.

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 INPUT, GET oder GETKEY einzulesen. Die zweite Möglichkeit, die wir gerade kennengelernt haben, besteht darin, die anfallenden Daten in DATA-Zeilen abzulegen, um sie so immer verfügbar zu haben. Diese Daten werden ja bei der Speicherung des Programms auf Kassette oder Diskette mit abgespeichert! Will man dagegen mittels INPUT oder GET eingegebene Daten "*retten*", so muß man diese in einer **separaten Datei** auf **Diskette** oder **Kassette** abspeichern.

Eine sehr häufige Anwendung dieser Kombination von READ und DATA findet man, wenn mittels BASIC ein Programm in Maschinensprache generiert werden soll. Das sind dann meistens Unmengen von DATA-Zeilen, die mittels einer FOR...NEXT-Schleife mit READ gelesen werden, und dann mit POKE in die entsprechenden Speicherstellen geschrieben werden. Danach kann man mit dem SYS-Befehl das Maschinenprogramm starten. Wie so etwas aussehen kann, soll das nächste Programm demonstrieren. Mit dieser Maschinenroutine werden der Bildschirm gelöscht, die Bildschirmfarbe auf schwarz und die Schriftfarbe auf orange gesetzt. Alle diejenigen, die sich in der Maschinensprache auskennen, mögen mir dieses sehr einfache Beispiel verzeihen. Aber das Programm soll ja auch nur zu Demonstrationszwecken dienen.

```
10 FORI=8000 TO 8016
20 READ M
30 POKE I,M
40 NEXT I
50 DATA 169,0,141,32,208,141,33,208,169,8,141,241,0,32,66,193,96
60 END
```

Das Maschinenprogramm wird ab Speicheradresse 8000 abgelegt. Das ist beim Commodore 128 mitten im BASIC-Speicher. Sie können solche Maschinenroutinen im Prinzip an jeden freien Speicherplatz des Rechners legen. Sie sollten allerdings darauf achten, daß Sie nicht Ihr eigenes BASIC-Programm im Speicher überschreiben. Damit diese Routine richtig läuft, müssen Sie vorher wieder mit BANK 15 die entsprechende Speicherkonfiguration auswählen.

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

READY.

Geben Sie jetzt den Befehl

SYS 8000

ein und drücken Sie RETURN. Augenblicklich wird der Bildschirm gelöscht und die Bildschirmfarbe auf schwarz gesetzt. Die Farbe der Schrift wird in orange geändert. Sie können, solange der Rechner eingeschaltet ist, diese Routine im Direktmodus oder auch in Ihren Programmen mit SYS 8000 aufrufen.

Wollen Sie das mit BASIC-Befehlen erreichen, so müßten Sie das folgende Programm verwenden:

```
10 BANK 15
20 REM BILDSCHIRM SCHWARZ
30 POKE 53280,0
40 POKE 53281,0
50 SCNCLR:REM BILDSCHIRM LOESCHEN
60 PRINT CHR$(129):REM SCHRIFTFARBE ORANGE
```

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

Im nächsten Kapitel werden Sie lernen, wie man auch für umfangreichere Probleme Programme in BASIC schreibt. Bei solchen komplexeren Programmen spielt die **Generierung von Feldern** bzw. **Arrays** eine große Rolle. Sie werden sehen, daß Sie die Befehle **READ** und **DATA** auch hierbei sinnvoll einsetzen können.

4

KOMPLEXERE

BASIC-ANWENDUNGEN

4. Komplexere BASIC-Anwendungen

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. Im ersten Beispiel wird eine Schleife benutzt, um die Beträge für die Monatsgehälter einzulesen. Mit Ihren bisherigen Kenntnissen müßten Sie auf die folgende Art und Weise vorgehen:

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

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

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

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

```
180 PRINT"DURCHSCHNITTLICHES EINKOMMEN";
190 PRINT D;"DM"
200 END
```

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

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

A(1)	2334
A(2)	2333
A(3)	2345.65
A(4)	2344.34
.	.
.	.
.	.
.	.
A(12)	3433.20

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

PRINT A(3)

Als Ausgabe würden wir in unserem Falle den Wert

2345.65

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

DIM Feldname*(Anzahl der Felder)*

Für unser Feld müßten wir also folgende Schreibweise verwenden:

DIM A(12)

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

?REDIM'D ARRAY ERROR IN *(Zeilennummer)*

aus. In unserem Beispiel wurde ein Feld für Gleitkommavariablen definiert. Sie können sebstverständlich 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 grafische Darstellung unseres Feldes 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 unter Verwendung eines Feldes an.

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

Durch die Verwendung eines Feldes haben wir nur eine Programmzeile mehr benötigt als beim ersten Beispiel. Man hätte die DIM-Anweisung auch noch in Zeile 20 unterbringen können, obwohl dann der Vergleich zwischen den beiden Programmen nicht gerecht ausgefallen wäre. So haben wir also

durch **eine** zusätzliche **Zeile** die monatlichen Gehälter weiter im Programm verfügbar. Wollen Sie z.B. vor der Ausgabe des monatlichen Durchschnitts die monatlichen Einkommen noch einmal auflisten lassen, so könnten Sie den letzten Programmteil wie folgt abändern:

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

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

A(X)

haben, bezeichnet man auch als **eindimensionale Felder**, d.h. sie **besitzen nur einen Index**. Der Index muß nicht unbedingt durch eine Zahl dargestellt werden. Es können auch Variablen oder arithmetische Ausdrücke Verwendung finden. Das bedeutet, daß Sie ein Feld individuell auf den jeweiligen Bedarf festlegen können. Bleiben wir bei unserem Beispiel mit den Monatsgehältern. Stellen Sie sich vor, Sie wollten das Programm so gestalten, daß es für eine verschiedene Anzahl von Monatsgehältern einsetzbar ist. Denkbar wäre dann folgende Änderung am Anfang des Programms:

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

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

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

?BAD SUBSCRIPT ERROR IN *(Zeilennummer)*

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 Beispiele zu eindimensionalen Feldern

Normalerweise können Sie davon ausgehen, daß nach der DIM-Anweisung die einzelnen Feldelemente leer sind. Innerhalb eines Programms kann es aber auch vorkommen, daß ein komplettes Feld gelöscht werden soll. Bei numerischen Feldern können Sie das erreichen, indem Sie die Elemente mit Nullen auffüllen. Das nachfolgende Programm zeigt, wie so etwas aussehen kann.

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

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

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

Dieses Programm dürfte das Löschen eines Feldes vom Prinzip her deutlich gemacht haben. Wollen Sie ein **Feld löschen**, welches **Strings beinhaltet**, so müssen Sie beachten, daß Sie die Elemente *nicht mit Nullen auffüllen*, da ja bei einem String die *Null als Zeichen interpretiert* wird. Es **kann ausreichen**, die einzelnen Feldelemente mit einem Leerzeichen aufzufüllen.

Wollen Sie in Ihrem Programm aber Strings verketten, so bekommen Sie durch das Leerzeichen immer ein Zeichen mehr in den String. Das kann je nach Programm zu Fehlern führen, z.B. dann, wenn Sie mit der **LEN-Funktion** arbeiten. Daher sollte man die Elemente eines Stringfeldes mit "**Nichts**" auffüllen.

Das sieht dann wie folgt aus:

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

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

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

a)

1
4
9
16
25
36

b)

10
8
6
4
2
0

c)

2
4
8
16
32
64

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 GETKEY A$
70 IF A$ < > "J" THEN 100
80 IF Z < 6 THEN 30
90 PRINT"LISTE VOLL !"
100 END
```

Es wurde hier zur besseren Übersichtlichkeit nur ein Feld mit 6 (bzw. 7) Elementen aufgebaut (Zeile 20). Zeile 30 beinhaltet den Zähler, der bei jeder Eingabe um eins erhöht wird. Da wir nicht die genaue Anzahl der Vornamen wissen, die wir eingeben wollen, kann hier keine FOR...NEXT Schleife verwendet werden. Zeile 40 verlangt mit INPUT die Eingabe eines Vornamens und ordnet diesen dem Element mit dem Index von Z zu. In Zeile 50 wird gefragt, ob weitere Eingaben gemacht werden sollen. Die Funktion von Zeile 60 dürfte inzwischen bekannt sein. Zeile 70 vergleicht das eingegebene Zeichen mit "J". Ist das Zeichen ungleich "J", wird das Programm in Zeile

100 beendet. Sollen weitere Eingaben stattfinden, so vergleicht Zeile 80 den Zähler auf **kleiner 6**. Hat der Zähler bereits den Wert 6, so wird durch Zeile 90 die Meldung **"LISTE VOLL"** ausgegeben und das Programm wird wiederum beendet. Hätten wir diese Zählerabfrage nicht eingebaut, so würde das Programm bei einem Wert größer 6 mit der Fehlermeldung

?BAD SUBSCRIPT ERROR IN 40

abbrechen. Erreicht Z nämlich den Wert 7, so wird in Zeile 40 versucht, das Element Y$(7) anzusprechen, das laut DIM-Anweisung aber nicht existiert. Solche ungewollten Programmabbrüche sollte man möglichst vermeiden. Mit TRAP und einer entsprechenden Fehlerroutine hätte man übrigens den gleichen Effekt erzielen können.

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 ausgegeben werden soll. Diese Ergänzung können Sie ja mal selbst vornehmen, indem Sie eine entsprechende IF...THEN-Abfrage im Programm unterbringen. Das Feld können Sie dann, wie bereits erklärt, wieder mit einer FOR...NEXT-Schleife ausgeben lassen.

Die **Dimensionierung** des Feldes könnte **mit DIM D$(200)** bei einer **eigenen Dateiverwaltung vollkommen ausreichend** sein. Allerdings reicht da kein eindimensionales Feld zur Erfassung der Daten aus. Kommen wir aber zunächst noch zu einem Beispiel, wo ein **Feld mit** den Befehlen **READ** und **DATA** mit Daten **belegt wird**. Stellen Sie sich vor, Sie benötigen in einem Programm des öfteren die Wochentage. Es ist nun recht mühselig, diese Daten bei jedem Programmstart neu eingeben zu müssen. Was spricht also dagegen, diese Daten in DATA-Zeilen abzulegen und gleich nach dem Programmstart mit READ in ein Feld einlesen zu lassen? Schauen wir uns das wiederum an einem Beispielprogramm an.

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

Dieses Programm ähnelt sehr dem Programm mit der Vornamenliste. Der eigentliche Unterschied ist in der Zeile 40 zu finden, 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!

Aufgaben

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.

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 bei einem ähnlichen Problem 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 *(eigentlich 6 Zeilen und 4 Spalten; wir wollten das Nullelement aber vorerst unberücksichtigt lassen).* Führen Sie die DIM-Anweisung nicht aus und benutzen eins der Elemente aus diesem Feld, so erzeugt der Rechner automatisch ein zweidimensionales Feld der Größe (10,10). Es lohnt sich also, die DIM-Anweisung auch bei kleineren mehrdimensionalen Feldern anzuwenden, da sonst sehr viel Speicherplatz verschenkt würde. Wie wird nun ein solches Feld benutzt?

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

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

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

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

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

Sie werden sich bestimmt fragen, warum man die Eingabe nicht in einer Schleife realisiert, wo nacheinander der Vorname, der Name und das Geburtsdatum mit nur einem einzigen INPUT-Befehl eingelesen werden.

In unserem Beispiel benutzt jeder INPUT-Befehl einen eigenen Kommentar, der den einzugebenden Wert näher beschreibt. Daher können in diesem Fall die drei INPUT-Befehle nicht durch einen INPUT-Befehl ersetzt werden und somit kann auch keine Schleife Verwendung finden. Wir wollen allerdings genau 6mal den Vornamen, den Nachnamen und das Geburtsdatum

einlesen lassen. Dafür können wir eine FOR...NEXT-Schleife verwenden, wie das folgende Beispiel zeigen soll:

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

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

```
10 REM FELD AUS DATAZEILE LADEN
20 DIM X(3,4)
30 FOR Z=1 TO 3
40 FOR S=1 TO 4
50 READ X(Z,S)
60 NEXT S,Z
70 DATA 11,12,13,14,21,22,23,24,31,32,33,34
80 REM AUSGABE FELD
90 PRINT"FELD ANZEIGEN (J/N) ?"
100 GETKEY A$
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 GETKEY A$
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 SLEEP 1

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. 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$(120,3)
30 SCNCLR
40 Z=Z+1
50 INPUT"VORNAME";X$(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 GETKEY A$
140 IF A$ = "J" THEN 30
150 END
```

Dieses Problem kann allerdings auch eleganter gelöst werden. Hier geht es jedoch nur darum, daß das Prinzip verstanden wird. Dadurch, daß wir keine FOR...NEXT-Schleife verwenden, müssen wir den **Zähler in Zeile 40** einfügen, um den **Index** bei jeder **neuen Eingabe** um **eins** zu **erhöhen**. Danach erfolgt über die INPUT-Befehle die Eingabe der Daten, die dann den entsprechenden Feldelementen zugeordnet werden. Sollen weitere

Daten eingegeben werden, verzweigt das Programm nach Zeile 30. Ansonsten wird das Programm beendet. Hier könnte man sich bei einem kompletten Datenverwaltungsprogramm einen Sprung zum Hauptmenü vorstellen, wo der Anwender dann weitere Programmpunkte anwählen kann.

Es wurde hier in der Zeile 140 der Vergleich mit IF...THEN vorgenommen, um zu überprüfen, ob weitere Daten eingegeben werden sollen. Sie sehen damit den Unterschied zum ersten Programm, wo diese Aufgabe von einer FOR...NEXT-Schleife übernommen wurde. *Zur Erinnerung: Wir verwenden IF...THEN genau dann, wenn die Anzahl der einzugebenden Daten nicht bekannt ist.*
Diese Beispiele sollten vorerst genügen, um Ihnen den Umgang mit mehrdimensionalen Feldern näher zu bringen. Der Commodore 128 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, was aber nicht heißen soll, daß diese Felder nicht doch bei bestimmten Problemstellungen gebraucht werden.

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

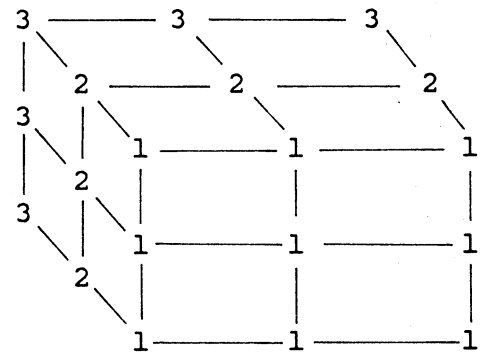
X = ZEILE

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 zweidimen-

sionale 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. Die Lösung folgt dieses Mal ausnahmsweise gleich im Anschluß.

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 brauchen aber nur 27. Das wäre eine recht große Verschwendung von Speicherplatz. In den Zeilen 30 bis 50 werden drei FOR...NEXT-Schleifen miteinander verknüpft. Die äußere Schleife bewirkt das Auffüllen der einzelnen Scheiben des Würfels. Dementsprechend sind die beiden anderen Schleifen zu interpretieren. In Zeile 70 schließt ein NEXT alle drei Schleifen auf einmal. Achten Sie dabei immer darauf, daß die **Laufvariablen hinter** dem **NEXT** in der **richtigen Reihenfolge gesetzt** werden. Die Ausgabe des Feldes sollte zwar nicht in der räumlichen Darstellung erfolgen, aber

die einzelnen "Scheiben" des Würfels müßten schon erkennbar sein. Eine Ausgabe, in der alle Werte hinter- oder untereinander aufgelistet werden, wäre nicht im Sinne der Aufgabe gewesen.

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

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

4.2 Unterprogramme

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 "*GO*to *SUB*routine", was soviel heißt wie **"Gehe ins Unterprogramm"**. Die Zeilennummer bezeichnet die Stelle, an der das Unterprogramm beginnt. Trifft das Programm auf diesen Befehl, so merkt es sich die Zeilennummer, von der aus es in das Unterprogramm gesprungen ist.

Es verzweigt dann zu der Zeilennummer des Unterprogramms und fährt dort mit der Programmausführung fort, bis es auf den Rücksprungbefehl

RETURN

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

?RETURN WITHOUT GOSUB ERROR IN *(Zeilennummer)*

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

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

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

?RETURN WITHOUT GOSUB ERROR IN 150

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

```
11Ø IF A < 1 THEN GOSUB 13Ø          < RICHTIG
```

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

```
1Ø REM FEHLER IM UNTERPROGRAMM
2Ø PRINT
3Ø PRINT CHR$(18);Z
4Ø GOSUB 7Ø
5Ø Z=Z+1:GOTO 2Ø
6Ø END
7Ø REM UNTERPROGRAMM
8Ø FOR I=1 TO 25
9Ø PRINT I;
1ØØ IF I >= 15 THEN 5Ø
11Ø NEXT I
12Ø RETURN
```

Geben Sie dieses Programm ein und starten Sie es. Sie werden feststellen, daß nach **21-** bzw. **22maligem Durchlaufen des Unterprogramms** der gesamte **Programmablauf** mit der **Fehlermeldung**

?OUT OF MEMORY ERROR IN 100

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

In der Zeile 100 wurde dann sogleich **gegen 2 Regeln verstoßen. Erstens** sollte man eine *FOR...NEXT-Schleife nicht mit GOTO verlassen*, da es auch hier zu Problemen mit der rechnerinternen Verwaltung dieser Schleifen kommen kann. **Zweitens**, und das ist in diesem Fall der eigentlich schwerwiegende Fehler, **darf** man ein **Unterprogramm nicht verlassen, 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 herspringen, 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 22. 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 dieses Programmlisting 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 SCNCLR
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 EG=A1+A2
350 SCNCLR
360 PRINT
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2 "= ";
380 INPUT ES
390 IF ES=EG THEN PRINT:PRINT TAB(10)"RICHTIG":F=0: GOTO 450
400 PRINT:PRINTTAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5)"DAS ERGEBNIS LAUTET" EG
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 COLOR 0,1:COLOR 4,1:COLOR 5,6
50 DIM RA$(4),BE$(4)
60 FOR I=1 TO 4
70 READ RA$(I),BE$(I)
80 NEXT I
90 DATA ADDITION,+,SUBTRAKTION,-,DIVISION,/, MULTIPLIKATION,*
100 GOTO 580
 .
 .
 .
```

Die Zeile 40 bewirkt durch die COLOR-Anweisung, daß die gesamte Bildschirmfarbe auf schwarz gesetzt wird *(die Anweisung COLOR wird noch ausführlicher besprochen)*. Weiterhin wird die Schriftfarbe in grün umgewandelt. In Zeile 50 werden zwei Felder, RA$ und BE$, generiert. Mit der FOR...NEXT-Schleife in Zeile 60 werden die beiden Felder geladen, und zwar

werden Feld RA$ die Begriffe Addition, Subtraktion usw. zugeordnet und Feld BE$ die entsprechenden Zeichen der Rechenarten. Wieso das Programm ab Zeile 100 in die Zeile 580 springt, werden Sie nachher selbst erkennen können. Nachdem das Programm gestartet wurde, werden die Felder mit diesen Daten gefüllt. Damit wir sehen, wie Feld RA$ beim Aufbau des Menüs verwendet wird, schauen wir uns nun die Programmzeilen des geänderten Programms von Zeile 570 bis 790 an.

```
.
.
.
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 SCNCLR: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 GETKEY E$
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 zur vorherigen Programmversion 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. Dies zeigt das folgende Beispiel.

```
.
.
.
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 GETKEY 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 EG=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 SCNCLR:A$="":B$=""
180 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL"
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 GETKEY A$
250 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 240
260 B$=B$+A$:PRINT A$;
```

```
270 NEXT I
280 GR=VAL(B$)
290 RETURN
 .
 .
 .
```

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

Die Eingabe der drei Ziffern mittels GET wurde schon angesprochen *(s. Kapitel: Eingabe v. Daten mit GET)*. 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 SCNCLR
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=EG 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"EG
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 einen speziellen Befehl *(MERGE)* nachladen und an das Programm anhängen. Dieser MERGE-Befehl wird meistens durch eine BASIC-Spracherweiterung zur Verfügung gestellt.

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

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

bemerkbar. Haben die Programme jedoch einen gewissen Umfang erreicht, so können es doch schon einige Sekunden sein, die ein Programm schneller abläuft.

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 EG=A1+A2** und aus der **Multiplikation EG=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 EG berechnet. Im früheren "*Rechenlehrgang*" konnten wir dann die Aufgabenstellung wie folgt stellen:

WIEVIEL ERGIBT EG / A1 = ?

Es wurde somit das vorher berechnete Ergebnis EG 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 EG=A1*A2
1050 I=EG:EG=A1:A1=I
.
.
.
```

Auch hier wird erst wieder das Ergebnis über die Multiplikation berechnet. Die Zeile 1050 bewirkt dann, daß die Werte für das **Unterprogramm** "*Aufgabenstellung*" richtig zugeordnet werden. Da wir nicht die Variablenbezeichnungen selbst bei der Ausgabe umstellen können, müssen wir die Werte der Variablen umordnen. Hier wird ebenfalls wieder die *Technik des Zwischenspeicherns* verwendet. Die Variable I erhält zunächst den Wert der Variablen EG. EG wird dann der Wert von A1 zugeordnet und A1 erhält schließlich den Wert von I, also von EG. 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 COLOR 0,1:COLOR 4,1:COLOR 5,6
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 SCNCLR:A$="":B$=""
180 PRINT AB(10)"GEBEN SIE DIE GROESSTE ZAHL "
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 GETKEY A$
250 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 240
260 B$=B$+A$:PRINT A$;
270 NEXT I
280 GR=VAL(B$)
290 RETURN
300 REM ***************************
310 REM * ERZEUGEN ZUFALLSZAHLEN *
320 REM ***************************
330 A1=INT(GR*RND(1))+1
340 A2=INT(GR*RND(1))+1
350 RETURN
360 REM ********************
370 REM * AUFGABENSTELLUNG *
380 REM ********************
390 SCNCLR
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=EG 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"EG
```

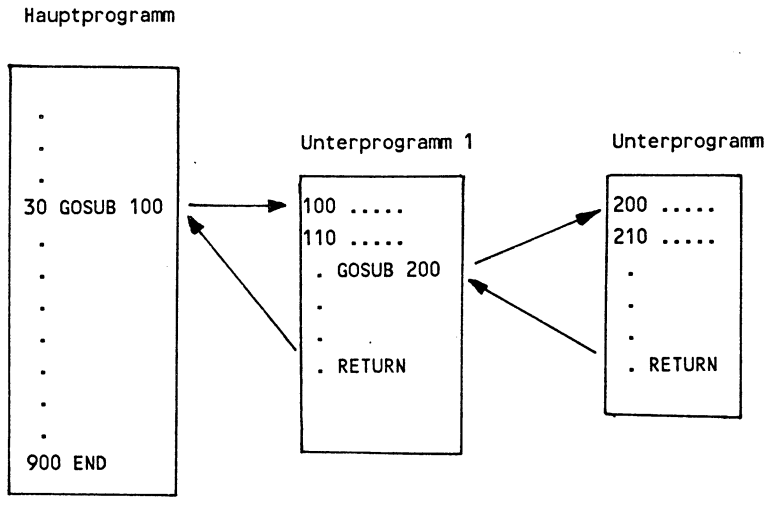
```
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
550 INPUT A$
560 RETURN
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 SCNCLR:F=0
610 PRINT
620 PRINT TAB(12)"RECHENLEHRGANG"
630 PRINT:PRINT
640 PRINT TAB(12)"WAEHLEN SIE:"
650 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 GETKEY E$
780 IF VAL(E$) < 1 OR VAL(E$) > 5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
800 REM ************
810 REM * ADDITION *
820 REM ************
830 GOSUB 110
840 GOSUB 310
850 EG=A1+A2
860 GOSUB 390
870 IF A$="J" THEN 840
880 GOTO 580
```

```
89Ø REM ***************
9ØØ REM * SUBTRAKTION *
91Ø REM ***************
92Ø GOSUB 11Ø
93Ø GOSUB 31Ø
94Ø IF A1 < A2 THEN I=A1:A1=A2:A2=I
95Ø EG=A1-A2
96Ø GOSUB 39Ø
97Ø IF A$="J" THEN 93Ø
98Ø GOTO 58Ø
99Ø REM  ************
1ØØØ REM * DIVISION *
1Ø1Ø REM ************
1Ø2Ø GOSUB 11Ø
1Ø3Ø GOSUB 31Ø
1Ø4Ø EG=A1*A2
1Ø5Ø I=EG:EG=A1:A1=I
1Ø6Ø GOSUB 39Ø
1Ø7Ø IF A$="J" THEN 1Ø3Ø
1Ø8Ø GOTO 58Ø
1Ø9Ø REM ******************
11ØØ REM * MULTIPLIKATION *
111Ø REM ******************
112Ø GOSUB 11Ø
113Ø GOSUB 31Ø
114Ø EG=A1*A2
115Ø GOSUB 39Ø
116Ø IF A$="J" THEN 113Ø
117Ø GOTO 58Ø
118Ø REM ********
119Ø REM * ENDE *
12ØØ REM ********
121Ø SCNCLR
122Ø 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 im folgenden Beispiel:

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. Auch diese Lösung finden Sie im Anschluß an diese Aufgabenstellung, da es sich anbietet, die Thematik sofort zu besprechen.

Lösung

```
.
.
.
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 EG=A1+A2
910 RETURN
920 REM ***************
930 REM * SUBTRAKTION *
940 REM ***************
950 IF A1 < A2 THEN I=A1:A1=A2:A2=I
960 EG=A1-A2
970 RETURN
980 REM  ************
990 REM  * DIVISION *
1000 REM ************
1010 EG=A1*A2
1020 I=EG:EG=A1:A1=I
1030 RETURN
1040 REM ******************
1050 REM * MULTIPLIKATION *
1060 REM ******************
1070 EG=A1*A2
1080 RETURN
1090 REM ********
1100 REM * ENDE *
1110 REM ********
1120 SCNCLR
1130 END
```

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

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

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

ON P GOSUB

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

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

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

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

3. Unterprogramme dürfen geschachtelt werden in dem Sinne, daß von einem Unterprogramm (UP1) das nächste Unterprogramm (UP2) aufgerufen wird usw. Da die Rücksprungadressen rechnerintern gespeichert werden, ist die Anzahl der geschachtelten Unterprogramme jedoch begrenzt. Achten Sie auch darauf, daß jeder GOSUB-Befehl auf das entsprechende RETURN trifft.

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

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

4.3 Menütechniken

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

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

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

Der Begriff "Menü" wurde schon im Zusammenhang mit dem Programm "Rechenlehrgang" erwähnt. Hier noch einmal eine Erklärung, was man unter einem Menü versteht. Im Menü eines Programms sind einzelne Programmpunkte aufgeführt, die der Anwender durch Betätigen von Zahlen- oder Buchstabentasten anwählen kann. Sie haben bereits mit so einem Menü beim "*Rechenlehrgang*" gearbeitet. Die äußere Gestaltung eines solchen Menüs bleibt dem Programmierer selbst überlassen. Nach Möglichkeit sollte ein Menü aber übersichtlich, d.h. nicht zu überladen, aufgebaut sein - obwohl sich dies in Einzelfällen nicht immer realisieren läßt. Weiterhin sollte der optische Eindruck eines Menüs ebenfalls berücksichtigt werden. Optische Trennungen durch bestimmte Zeichenfolgen sind durchaus erwünscht. Aber Vorsicht, auch hier gilt es, nicht zu übertreiben. Denken Sie grundsätzlich daran, daß beim Anwender des Programms auch der Spruch gilt: "Das Auge ißt mit!"

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

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

Quadratwurzel
Sinus
Cosinus
Natürlicher Logarithmus
Dekadischer Logarithmus

Diese fünf Berechnungen sollen also je nach Auswahl ausgeführt werden.

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

GEBEN SIE IHRE WAHL EIN (1-6) ?

Damit wäre unser Menü von der Planung her schon fast vollendet. Nun wollen wir noch eine Überschrift und zur optischen Aufbesserung noch einen Rahmen im Menü unterbringen. Die Bildschirmfarbe soll komplett auf schwarz gesetzt werden. Die Überschrift, so ist geplant, soll bei Ausführung des Programms bei jedem Programmteil auf dem Bildschirm vorhanden sein. Zur **Erstellung der Überschrift bietet sich** somit ein **Unterprogramm an.** Wie das Menü später auf dem Bildschirm erscheinen soll, wird im folgenden dargestellt.

```
****************************************
*                                      *
*        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 GET-Befehl an dieser Stelle Verwendung finden kann. Wie das Programm für die Erzeugung dieses Menüs aussieht, wird im folgenden aufgelistet.

```
10 REM *****************
20 REM * PROGRAMMSTART *
30 REM *****************
40 COLOR 0,1:COLOR 4,1
50 SCNCLR
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 SCNCLR
240 FOR I=1 TO 40:PRINT"*";:NEXT I
250 PRINT"*                                      *";
260 PRINT"*          MATHEMATISCHE TABELLE       *";
270 PRINT"*                                      *";
280 FOR I=1 TO 40:PRINT"*";:NEXT I
290 RETURN
```

```
300 REM *********
310 REM * MENUE *
320 REM *********
330 GOSUB 230
340 FOR I=1 TO 18
350 PRINT"*                                          *";
360 NEXT I
370 FOR I=1 TO 40:PRINT"*";:NEXT I
380 PRINT CHR$(19);
390 FOR I=1 TO 3:PRINT:NEXT I
400 FOR I=1 TO 6
410 PRINT CHR$(29);M$(I)
420 NEXT I
430 PRINT
440 PRINT CHR$(29);
450 PRINT "GEBEN SIE IHRE WAHL EIN  (1-6)";
460 INPUT W$
 .
 .
 .
```

Zunächst sollen die einzelnen Programmzeilen kurz erklärt werden. Die Zeilen 40 bis 50 setzen die Bildschirmfarben auf schwarz und löschen den Bildschirm. In den Zeilen 60 bis 80 wird das Feld M$ generiert und mit den Daten aus den DATA-Zeilen 90 bis 140 geladen. Danach überspringt das Programm den Programmteil der Unterprogramme und fährt mit der Ausführung in Zeile 330 fort. Von dort springt das Programm in das Unterprogramm ab Zeile 230, in dem die Kopfzeile erzeugt wird. Die beiden FOR...NEXT-Schleifen in den Zeilen 240 und 280 bewirken, daß jeweils eine Linie mit Sternchen auf dem Bildschirm erscheint. Dazwischen liegen die PRINT-Befehle für die Überschrift.

Danach wird das Unterprogramm verlassen und der Programmablauf wird in Zeile 340 fortgesetzt. In einer weiteren Schleife *(340-360)* wird der Rahmen des Menüs aufgebaut. In Zeile 370 wird die untere Linie des Menüs ausgegeben, und Zeile 380 bringt den Cursor wieder in die linke obere Bildschirmecke. Zeile 390 gibt drei Leerzeilen aus, damit die Menüpunkte nicht

direkt in der Kopfzeile erscheinen. Die Zeilen 400 bis 420 geben dann nacheinander das Feld mit den einzelnen Menüpunkten aus. In Zeile 410 wird vor jeder Ausgabe eines neuen Elements der Cursor um eine Position nach rechts gerückt, damit die Ausgabe den linken Menürahmen nicht überschreibt. Sie können ruhig einmal ausprobieren, was geschieht, wenn Sie diesen Befehl entfernen. Zeile 450 gibt schließlich noch die Aufforderung an den Anwender aus, einen Wert einzugeben. Dadurch, daß der PRINT-Befehl in dieser Zeile mit einem Semikolon endet, wird die Eingabe mit INPUT direkt nach der Klammer (1-6) erwartet.

Damit hätten wir die Erstellung des Menüs abgeschlossen. Die Aufbereitung des eingegebenen Wertes mit entsprechender Verzweigung zu anderen Programmteilen wollen wir uns hier ersparen. Das Prinzip ist das gleiche wie auch beim "*Rechenlehrgang*". Da wir hier allerdings mit INPUT gearbeitet haben, müßten Sie die eingegebenen Werte noch auf Zulässigkeit überprüfen, da wir ja nicht wie mit GET bzw. mit GETKEY bestimmte Tasten selektieren können.

Zum Unterprogramm *Kopfzeile* sei noch ein Hinweis gegeben. 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 den Anweisungen GET bzw. GETKEY auseinandersetzen.

4.3.1 Verwendung von GET-Routinen im Menü

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

Diese Eingaberoutine mit GET ist aber noch sehr primitiv. Hatte man drei Zahlen bzw. Zeichen eingegeben, wurden diese automatisch übernommen, ohne daß man eine Möglichkeit zur Korrektur hatte. Ebenfalls wurden auf alle Fälle drei Zeichen eingelesen, so daß man gezwungen war, für die Zahl 54 die **Zifernfolge 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, die GET-Anweisung einzusetzen.

Wir wollen uns nun an die Entwicklung einer eigenen GET-Routine begeben. Diese können Sie nach Fertigstellung in Ihre eigenen Programme als Unterprogramm aufnehmen. Sie können selbstverständlich auch GETKEY anstelle von GET einsetzen. Die erste Zeile hierbei dürfte Ihnen inzwischen bekannt sein:

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

oder

```
10 GETKEY A$
```

Damit können Sie jedes Zeichen von der Tastatur aus einlesen und A$ zuordnen. Wir wollen für unseren Fall aber nur Zahlen zulassen. Daher müssen wir diese Tasten selektieren, d.h. andere

Eingaben als Zahlen müssen ausgeschlossen werden. Das erreichen wir mit einer IF...THEN-Abfrage:

```
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
```

Die **ASCII-Werte 48 bis 57** repräsentieren die Zahlen von **0 bis 9**. Ist der eingegebene ASCII-Wert also kleiner 48 oder größer 57, wird die Eingabe ignoriert und zurück in die Zeile 10 gesprungen. Wollen wir nun nur Zahlen bis zu einer bestimmten Stellenzahl zulassen, müssen wir die eingegebenen gültigen Zeichen zählen. Soll die größte Zahl also vierstellig sein, müssen wir den Zähler auf den Wert größer vier abfragen. Wir benötigen also zwei weitere Zeilen. Eine, in der der Zähler hochgezählt und eine, in der der Zähler auf den Wert vier überprüft wird.

```
10 GET A$:IF A$="" THEN 10
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Der Zähler Z in Zeile 30 wird jetzt nur dann hochgezählt, wenn ein zulässiger Wert eingegeben wurde. Hat Z bereits den Wert 4, so wird kein weiterer Wert akzeptiert und das Programm springt wieder in die Zeile 10.

Wir müssen jetzt unserer Routine noch mitteilen, daß der eingegebene **Wert übernommen** werden soll. Dafür bietet sich, wie beim INPUT-Befehl, die **RETURN-Taste** an. Welchen **ASCII-Code** besitzt die **RETURN-Taste**? In der Tabelle erfahren wir, daß diese Taste den **ASCII-Wert 13** hat. Wir brauchen damit nur den ASC(A$) auf den Wert 13 abzufragen. *Doch Vorsicht, wo bauen wir diese Abfrage jetzt ein?* Da **13 kleiner als 48** ist, dürfen wir diese Abfrage **nicht hinter** die **Zeile 20** setzen, da dann die RETURN-Taste ignoriert würde. Die Abfrage muß also eine **Zeilennummer** bekommen, die **kleiner** als **20** ist. Nehmen wir hierfür die Nummer 15.

Wie soll denn die Routine nun weiter verzweigen, wenn die RETURN-Taste gedrückt wurde? Da wir das zu diesem Zeitpunkt noch nicht genau wissen, aber zugleich absehen können, daß die Routine nicht allzu groß wird, soll vorerst nach Zeile 100 verzweigt werden.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

Was wir jetzt noch benötigen, ist eine Zeile, in der die eingegebenen Zeichen miteinander in einer Stringvariablen verknüpft werden. Das soll in der Zeile 50 geschehen. Weiterhin wollten wir unsere eingegebenen Zeichen ja auf dem Bildschirm erkennen können. Diese Aufgabe soll Zeile 60 übernehmen.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
```

Mit Zeile 60 werden die Zeichen ab der aktuellen Cursorposition ausgegeben und zwar hintereinander (Semikolon), d.h. an der Stelle auf dem Bildschirm, an der der letzte PRINT-Befehl ausgeführt wurde. Nachdem das Zeichen ausgegeben wurde, soll das Programm wieder in die Zeile 10 springen, also **GOTO 10**.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
```

Jetzt brauchen wir nur noch den erzeugten **String** in **B$** in einen **numerischen Wert umzuwandeln** und diesen einer **numerischen Variablen** zu **übergeben**. Das kann nach Betätigen der RETURN-Taste geschehen. Außerdem müssen wir daran denken, den **Zähler Z** nach dem Drücken der RETURN-Taste **wieder auf Null** zu setzen. Sonst würde der Wert bis zum nächsten Aufruf der Routine mitgezogen und man hätte nicht mehr die Möglichkeit, eine vierstellige Zahl einzugeben.

Soll die Routine als **Unterprogramm** Verwendung finden, muß die **letzte Zeile** selbstverständlich ein **RETURN** beinhalten. Wir wollen uns zunächst die komplette Routine anschauen. Nachdem Sie sie eingegeben haben, können Sie ja ein wenig damit herumexperimentieren. Sie sollten vielleicht als letzte Zeile die Ausgabe der Variablen vorsehen, an die der numerische Wert übergeben wurde.

```
10 GET A$:IF A$="" THEN 10
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
50 B$=B$+A$
60 PRINT A$;
70 GOTO 10
100 B=VAL(B$):Z=0
110 PRINT B
120 END
```

Damit hätten wir eine **GET-Routine**, mit der wir **bis zu vierstellige Zahlen einlesen** und **anzeigen** lassen können. Wollen Sie noch größere Zahlen einlesen lassen, so können Sie das durch Verändern des Wertes 4 in der Zeile 40 erreichen.

Diese Routine ist somit schon um einiges komfortabler als die, die wir vom *"Rechenlehrgang"* her kennen. Allerdings fehlt ihr noch die **Funktion**, daß man **bereits eingegebene Werte** wieder **löschen** kann. Diese Funktion gehört schon zu den etwas komplizierteren Merkmalen einer selbstgebauten GET-Routine. Im

folgenden soll nun eine Routine, die diese Funktion beinhaltet, aufgelistet werden.

```
10 REM GET-ROUTINE MIT LOESCHFUNKTION
20 GET A$:IF A$="" THEN 20
30 IF ASC(A$) = 13 THEN 130
40 IF ASC(A$) < > 20 THEN 70
50 IF LEN(B$) < 1 THEN 20
60 B$=LEFT$(B$,LEN(B$)-1):Z=Z-1:GOTO 110
70 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 20
80 Z=Z+1
90 IF Z > 4 THEN Z=4:GOTO 20
100 B$=B$+A$
110 PRINT A$;
120 GOTO 20
130 B=VAL(B$):Z=0
140 PRINT B
150 END
```

Wir wollen nun die Zeilen im Programm besprechen, die noch hinzugekommen sind. Zunächst wäre da die **Zeile 40**. In dieser Zeile wird **überprüft**, ob die **Taste INST/DEL** betätigt wurde. Der ASCII-Wert dieser Taste ist 20. Wurde die Taste nicht gedrückt, so verzweigt das Programm weiter nach Zeile 70. Haben Sie die INST/DEL-Taste allerdings betätigt, so wird in Zeile 50 überprüft, ob der String B$ noch Zeichen enthält. Wird diese **Abfrage versäumt**, so würde der **Löschversuch** eines weiteren Zeichens **bei** einem bereits **leeren String** zu einem Programmabbruch mit der Fehlermeldung

?ILLEGAL QUANTITY ERROR IN 60

führen. Ist der **String** also **leer**, wird die **INST/DEL-Taste ignoriert** und das Programm verzweigt erneut zur Zeile 20. In der Zeile 60 findet nun der eigentliche Vorgang des Löschens statt. Mit der Befehlsfolge

B$=LEFT$(B$,LEN(B$)-1)

wird der bis dahin eingegebene String B$ um ein Zeichen verkürzt. Der Befehl LEFT$(B$,X) erzeugt bekanntlich einen String mit den **X linken** Zeichen von B$. In den meisten Fällen steht an der Stelle von X eine Zahl. Hier wurde jedoch anstelle von X der LEN-Befehl benutzt. Die **Berechnung LEN(B$)-1** wird zuerst ausgeführt. Das bedeutet, daß ein Wert gebildet wird, der genau um den **Wert eins kleiner** ist **als** die **aktuelle Länge** von **B$**. Mit diesem neuen Wert wird nun ein Teilstring von B$ gebildet, der genau um ein Zeichen kürzer ist als der ursprüngliche String von B$. Dieser um ein Zeichen verkürzte String wird erneut B$ zugeordnet. Der Vorgang ist in etwa mit der folgenden Operation bei numerischen Variablen vergleichbar:

A=A-1

Hier wird von der Variablen A der Wert 1 subtrahiert und das Ergebnis wiederum A zugeordnet.

Mit dieser Befehlsfolge haben wir also erreicht, daß das zuletzt dem String B$ zugeordnete Zeichen wieder gelöscht wurde.

Weiterhin muß in der Zeile 60 der **Zähler Z** um den **Wert 1 vermindert werden**, da er die Anzahl der Ziffern der gesamten Zahl bestimmt. Wir wollen ja eine Zahl mit maximal vier Stellen eingeben. Daher wurde Z als Zähler benutzt, der die bereits gültigen eingegebenen Zeichen zählt. **Löschen** wir **ein Zeichen**, so muß **nicht nur** der **String B$** um ein Zeichen **verkürzt werden, sondern** es muß **auch** vom **Zähler Z** der Wert 1 subtrahiert werden. Würde man vergessen, den Zähler ebenfalls zu verkleinern, so wäre eine weitere Eingabe nach vier gültigen Zeichen nicht mehr möglich. Der String B$ würde zwar jedesmal beim Betätigen der INST/DEL-Taste um ein Zeichen verkürzt, aber da der Zähler bereits den Wert vier angenommen hat, würde das Programm in Zeile 90 dann wieder nach Zeile 20 verzweigen.

Der letzte Befehl in der Zeile 60 veranlaßt das Programm, in die Zeile 110 zu springen. Dort wird der aktuelle Inhalt von A$ ausgegeben. In A$ befindet sich nach dem Betätigen der INST/DEL-Taste der CHR$(20)-Code. Dieser bewirkt beim

Ausdruck mit PRINT genau das gleiche, als ob Sie die INST/DEL-Taste direkt benutzen würden. Um das zu veranschaulichen, geben Sie einmal folgende Befehlsfolge in den Rechner:

```
PRINT "COMMODORE 129";CHR$(20);"8"
```

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

```
COMMODORE 128
```

Der CHR$(20)-Code löscht also bei der Ausgabe mit PRINT genau wie die INST/DEL-Taste ein Zeichen. Die "8" tritt sofort an die Stelle des gelöschten Zeichens. Dadurch erhalten Sie dann den oben gezeigten Ausdruck. Nichts anderes geschieht in unserer GET-Routine in der Zeile 110, wenn A$ den CHR$(20)-Code enthält. Das ist schon alles, was zu den neu hinzugekommenen Zeilen zu sagen wäre.

Damit haben wir nun eine GET-Routine aufgebaut, die es uns erlaubt, je nach Manipulation des Wertes Z kürzere oder längere Zeichenketten einlesen zu lassen. Außerdem kann man diese Zeichen gleichzeitig anzeigen lassen sowie auch wieder löschen. Diese **Routine** ist dem **INPUT-Befehl** von der Bedienung her schon recht **ähnlich** geworden, nur daß wir hier bestimmen können, welche Zeichen wir zulassen wollen oder nicht.

Zum Schluß wollen wir unsere GET-Routine noch so verfeinern, daß wir auch eine Art von **Cursor** zur Verfügung haben. Er blinkt zwar nicht, jedoch wissen wir, wo das nächste Zeichen einzugeben ist. Als unseren eigenen **"Cursor"** wollen wir den **schmalen Grafikstrich** nehmen, der auf der Taste mit dem Klammeraffen zu finden ist. Dieses Zeichen besitzt den **CHR$(164)-Code**. In diesem Zusammenhang wird der **CHR$(157)-Code** mit verwendet, um den realen Bildschirmcursor, der bei der Verwendung von GET ja unsichtbar bleibt, **eine Stelle nach links** zu bewegen. Dadurch liegt der Cursor über unserem Grafikzeichen. Wird nun ein Zeichen eingegeben, wird es durch den Befehl PRINT A$; an der Stelle des aktuellen

Bildschirmcursors ausgegeben. Dadurch wird das Grafikzeichen durch das eingegebene Zeichen überschrieben. Gleichzeitig wird mit der Ausgabe des neuen Zeichens unser **Cursor (CHR$(164))** direkt hinter diesem Zeichen wieder mit ausgegeben *(Zeile 210)*.

Weiterhin wollen wir die Routine dahingehend erweitern, daß Buchstaben und das Leerzeichen mit eingegeben werden können. Dazu müssen wir zwei weitere Abfragen mit IF...THEN verwenden, um u.a. auszuschließen, daß die **ASCII-Werte**, die **zwischen** den **Zahlen** und den **Buchstaben** liegen **(58 bis 64)**, zugelassen werden. Doch schauen Sie sich zunächst das Programmlisting dieser Routine an.

```
10 REM ************************
20 REM * CURSOR POSITIONIEREN *
30 REM ************************
40 SCNCLR
50 FOR I=1 TO 5
60 PRINT CHR$(17);CHR$(29);
70 NEXT I
80 PRINT CHR$(164);CHR$(157);
90 REM ********************
100 REM * ZEICHEN EINLESEN *
110 REM ********************
120 GET A$:IF A$="" THEN 120
130 IF ASC(A$)=13 THEN 260
140 IF ASC(A$)=32 THEN 200
150 IF ASC(A$) < > 20 THEN 170
160 IF LEN(B$) > = 1 THEN B$=LEFT$(B$,LEN(B$)-1):GOTO 210
170 IF ASC(A$) < 48 THEN 120
180 IF ASC(A$) > 90 THEN 120
190 IF ASC(A$) > 57 AND ASC(A$) < 65 THEN 120
200 B$=B$+A$
210 PRINT A$;CHR$(164);CHR$(157);
220 GOTO 120
```

```
230 REM ******************
240 REM * AUSGABE STRING *
250 REM ******************
260 SCNCLR
270 FOR I=1 TO 15
280 PRINT CHR$(17);CHR$(29);
290 NEXT I
300 PRINT B$
```

In den Zeilen 40 bis 70 werden zunächst der Bildschirm gelöscht und der Bildschirmcursor in die 5. Bildschirmzeile und die 5. Bildschirmspalte gebracht. Dann wird unser selbstdefinierter Cursor an dieser Stelle ausgegeben (Zeile 80) und der unsichtbare Bildschirmcursor wird um eine Stelle nach links (CHR$(157)) über unseren Cursor gesetzt. In Zeile 120 beginnt nun die eigentliche Routine. Der Inhalt der Zeile 140 ist neu hinzugekommen. Hier wird abgefragt, ob das eingegebene Zeichen ein Leerzeichen *(CHR$(32))* war. Die nächsten Zeilen sind wiederum bekannt. Zeile 180 fragt ab, ob das eingegebene Zeichen einen größeren ASCII-Wert als der Buchstabe "Z" hatte. In Zeile 190 wird der Bereich zwischen den ASCII-Werten 57 und 65 abgefragt. Es wurde dazu der logische Operator AND verwendet. AND mußte hier benutzt werden, da beide Bedingungen erfüllt sein müssen. Damit das Programm nach Zeile 120 verzweigen kann, muß der eingegebene ASCII-Wert sowohl größer 57 als auch kleiner 65 sein *(das ist das Intervall zwischen dem Zahlen- und Buchstabenbereich)*. Hätten wir hier ein OR verwendet, so wären weder eine Zahl noch ein Buchstabe als Eingabe zugelassen worden. Die Wirkung der Zeile 210 wurde ja bereits besprochen. Die folgenden Zeilen sollen das noch einmal verdeutlichen. Die Position des unsichtbaren Bildschirmcursors soll durch den Stern "*" gekennzeichnet werden.

Wir nehmen an, daß wir den Buchstaben A eingegeben haben. Die Zeile 210 bewirkt dann folgendes:

A* Nach Ausführung von PRINT A$;

A_* Nach Ausführung von CHR$(164);

A* Nach Ausführung von CHR$(157);

Ich hoffe, daß Ihnen durch dieses kleine Schaubild die Wirkung der Zeile 210 deutlich geworden ist.

Die restlichen Programmzeilen dieser Routine sollten Ihnen inzwischen bekannt sein.

Wir haben uns nun eine GET-Routine erstellt, die dem INPUT-Befehl schon sehr ähnlich ist. Sie sollten nun in der Lage sein, diese Routine Ihren Programmen entsprechend anpassen zu können, d.h. daß Sie durch entsprechende IF...THEN-Abfragen selbst bestimmen, welche Tasten Sie selektieren möchten. Damit steht Ihnen für Ihre Menüs oder sonstigen Eingaben innerhalb von Programmen eine recht komfortable und sichere Routine zur Verfügung.

Was jetzt noch fehlt, ist eine Routine, mit der Sie auf einfache Art und Weise den Cursor auf dem Bildschirm positionieren können, d.h. daß Sie nur durch die Angabe von Bildschirmzeilen- und Bildschirmspaltennummer die Position des Cursors bestimmen können. Sie haben zwar mit **CHAR** die Möglichkeit, den Cursor zu positionieren, jedoch hat diese Art der Cursorpositionierung aber den Nachteil, daß Sie wieder rechnerspezifisch ist.

Besprechen wir trotzdem zunächst die Möglichkeit, mit *CHAR* den Cursor zu positionieren.

4.3.2 Cursorpositionierung mit CHAR

Die Anweisung CHAR ist eigentlich für die einfache Beschriftung von Grafiken vorgesehen. So können im Grafikmodus ohne große Schwierigkeiten Balkendiagramme oder sonstige grafische Darstellungen mit einem entsprechenden Text versehen werden. Jedoch können selbst im hochauflösenden Grafikmodus für die Positionierung nur die Spaltenwerte bis 39 und die Zeilenwerte bis 24 verwendet werden, also die Unterteilung für den 40-Zeichenschirm. Eine auf den Punkt genaue Positionierung ist mit CHAR nicht möglich.

Nun kennen einige BASIC-Dialekte die Anweisung *LOCATE* für die Textausgabe im Textmodus. Im BASIC 7.0 ist hierfür ebenfalls die Anweisung **CHAR** zuständig. LOCATE hat im BASIC 7.0 nur die Aufgabe, den unsichtbaren Grafikcursor zu positionieren. Wir haben damit die Möglichkeit, mit **CHAR** im normalen Textmodus unsere Texte an beliebiger Stelle ausgeben zu lassen.

Die Schreibweise von CHAR lautet:

CHAR F,X,Y,T,I

Dabei müssen die Parameter X und Y mit angegeben werden. Die restlichen Parameter können wahlweise mit angegeben werden. Die Parameter bedeuten dabei im einzelnen:

F	Farbspeicher (0 bis 3, im Textmodus nicht relevant)
X	Spaltennummer (0 bis 39)
Y	Zeilennummer (0 bis 24)
T	auszugebender Text
I	0=normale Darstellung / 1=inverse Darstellung

Werden bei der Anwendung von CHAR nur die Parameter X und Y mit angegeben, so wird der Cursor an der entsprechenden Stelle auf dem Bildschirm positioniert. Eine anschließende Aus-

gabe mit PRINT erfolgt an dieser Stelle. Das folgende Beispiel soll das verdeutlichen.

```
10 SCNCLR
20 CHAR ,19,12
30 PRINT "AUSGABE MIT CHAR!"
40 END
```

Selbstverständlich hätte die Textausgabe auch sofort mit CHAR ausgeführt werden können. Wichtiger ist jedoch, wie Sie nach dem Starten des Programms feststellen können, daß *innerhalb eines Programms nach CHAR kein CARRIAGE RETURN erfolgt!*

Wir haben damit eine komfortable Möglichkeit kennengelernt, Textausgaben an beliebigen Stellen des Bildschirms ausgeben zu lassen.

4.3.3 Cursorsteuerung mit CHR$-Codes

Ich will Ihnen nun eine Möglichkeit aufzeigen, die Sie auch für andere Rechner verwenden können, sofern diese über entsprechende Cursorsteuerzeichen verfügen.

Sie basiert auf der Überlegung, daß man den Cursor ja auch mit den CHR$-Codes CHR$(17), CHR$(29), CHR$(145) und CHR$(156) steuern kann. Für uns sind hier nur **die Codes CHR$(17) und CHR$(29) interessant**, also die, die den *Cursor nach unten* und *nach rechts* bewegen. Mit diesen Steuercodes werden zunächst zwei Strings gebildet. Der eine String enthält 40mal das Steuerzeichen, das den Cursor nach rechts bewegt *(CHR$(29))*. Der zweite String enthält 25mal das Steuerzeichen, das den Cursor um eine Zeile nach unten bewegt *(CHR$(17))*. Diese Programmzeilen wollen wir uns nun anschauen.

```
.
.
.
90 REM CURSOR NACH UNTEN
100 FOR I=1 TO 25
110 CU$=CU$+CHR$(17)
120 NEXT I
130 REM CURSOR RECHTS
140 FOR I=1 TO 40
110 CR$=CR$+CHR$(29)
120 NEXT I
.
.
.
```

Durch diese Programmzeilen werden also unsere zwei Strings generiert. Damit haben wir schon die Hälfte der Arbeit für unsere Cusorpositionierung geschafft. Was jetzt noch kommen muß, ist die eigentliche Positionierung. Wir wollten durch die Angabe zweier Werte den Cursor an eine bestimmte Stelle des Bildschirms bringen. Die Schreibweise soll dabei die folgende sein:

S=ZEILE.SPALTE

oder speziell:

S=10.12

Die Variable S erhält demnach einen Wert zugeordnet, bei dem die Vorkommastelle die Zeile und die Nachkommastelle die Spalte angibt. Diese Variable muß jetzt noch entsprechend verarbeitet werden, und dann kann der Cursor schon gesetzt werden. Die folgenden Zeilen zeigen Ihnen, wie Sie dabei vorzugehen haben.

```
.
.
.
300 REM CURSORPOSITIONIERUNG
310 PRINT CHR$(19);LEFT$(CU$,S)
320 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
330 RETURN
.
.
.
```

Das ist schon die gesamte Routine zur Cursorpositionierung. Die Zeile 310 dürfte schnell verstanden werden. Zunächst wird bei jeder Positionierung der Cursor in die linke obere Bildschirmecke gebracht. Damit hat man einen Ausgangspunkt geschaffen, an dem man sich orientieren kann. Danach wird ein **Teilstring von CU$** entsprechend dem **Wert** von **S** gebildet, der den Cursor in unserem Fall 10 Zeilen nach unten bewegt. Es wird ja bei **LEFT$(CU$,S)** nur die **Vorkommastelle** von **S berücksichtigt**. Nach Ausführung der Zeile 310 steht der Cursor somit schon in der richtigen Bildschirmzeile.

Zeile 320 ist etwas komplizierter im Aufbau. Beginnen wir mit der inneren Klammer. Dort wird mit **S-INT(S)** die **Nachkommastelle von** der **Vorkommastelle abgetrennt**. Die Nachkommastelle, in unserem Fall **.12**, wird nun mit **100 multipliziert**. Anschließend wird noch .5 addiert. Damit wird erreicht, daß der Nachkommawert auch wirklich erhalten bleibt. Es kann nämlich schon mal vorkommen, daß man bei der Berechnung von **100*(S-INT(S))** nicht genau, wie in unserem Beispiel, **12** als Ergebnis erhält, sondern **11.99999**, als **ganzzahligen Wert** also nur **11**. Wird nun **.5 addiert**, so ergibt sich ein **Wert** von **12.49999**. Da ja bei der Anwendung von LEFT$(CR$,S) von S nur der ganzzahlige Teil berücksichtigt wird, erreicht man also durch diesen kleinen Trick, daß auf jeden Fall der Wert 12 erreicht wird. Mit diesem Wert wird nun ein Teilstring mit 12 Zeichen *(Steuerzeichen)* von CR$ gebildet. Bei der Ausgabe mit PRINT wird der Cursor dadurch um 12 Stellen nach rechts gesetzt. Die letzte Zeile enthält ein RETURN, da die Positionierung ja als Unterprogramm aufgerufen werden soll.

Damit steht Ihnen eine *Routine zur Cursorpositionierung* zur Verfügung, die vom Aufbau her sehr schnell verstanden werden kann und sich dadurch auch leicht in eigenen Programmen anwenden läßt. *Weiterhin dürfte diese Routine sich sehr leicht auch auf andere Rechner übertragen lassen.* Hierzu brauchen Sie nur die entsprechenden Cursorsteuerzeichen einzusetzen.

Diese Routine kann man nun auch für den Aufbau von Menüs verwenden. Wir wollen nun das Programm für die *"Mathematische Tabelle"* dahingehend abändern, daß wir die Ausgabe der einzelnen Menüpunkte über unsere neue Cursorpositionierung vornehmen können. Dazu werden in den DATA-Zeilen die Leerzeichen vor den Menüpunkten entfernt. Danach erfolgt im Programm die Erzeugung der beiden Strings für die Cursorsteuerung. Schauen Sie sich jedoch zunächst das abgeänderte Programmlisting an.

```
.
.
.
90 DATA "1 QUADRATWURZEL"
100 DATA "2 SINUS"
110 DATA "3 COSINUS"
120 DATA "4 NATUERLICHER LOGARITHMUS"
130 DATA "5 DEKADISCHER LOGARITHMUS"
140 DATA "6 PROGRAMMENDE"
150 REM *****************
160 REM * CURSOR RECHTS *
170 REM *****************
180 FOR I=1 TO 40
190 CR$=CR$+CHR$(29)
200 NEXT I
210 REM **********************
220 REM * CURSOR NACH UNTEN *
230 REM **********************
240 FOR I=1 TO 25
250 CU$=CU$+CHR$(17)
260 NEXT I
270 GOTO 510
```

```
280 REM ******************
290 REM * UNTERPROGRAMME *
300 REM ******************
310 REM
320 REM *************
330 REM * KOPFZEILE *
340 REM *************
350 SCNCLR
360 FOR I=1 TO 40:PRINT"*";:NEXT I
370 PRINT"*                                          *";
380 PRINT"*         MATHEMATISCHE TABELLE            *";
390 PRINT"*                                          *";
400 FOR I=1 TO 40:PRINT"*";:NEXT I
410 RETURN
420 REM ************************
430 REM * CURSORPOSITIONIERUNG *
440 REM ************************
450 PRINT CHR$(19);LEFT$(CU$,S);
460 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
470 RETURN
480 REM *********
490 REM * MENUE *
500 REM *********
510 GOSUB 350
520 FOR I=1 TO 18
530 PRINT"*                                      *";
540 NEXT I
550 FOR I=1 TO 40:PRINT"*";:NEXT I
560 PRINT CHR$(19);
570 FOR I=1 TO 3:PRINT:NEXT I
580 FOR I=1 TO 6
590 S=4+I*2+.03
600 GOSUB 450: REM * CURSOR POSIT. *
610 PRINT M$(I)
620 NEXT I
 .
 .
 .
```

Wie bereits erwähnt, wurden bei den Menüpunkten in den DATA-Zeilen die Leerzeichen entfernt. Neu hinzugekommen sind dann die Programmzeilen 150 bis 260, die vom Inhalt her schon besprochen wurden. Weiterhin wurden die Zeilen 420 bis 470 als Unterprogramm hinzugefügt. Auch diese sind vom Inhalt her bereits erläutert worden. Die folgenden Zeilen des ersten Programms zur *"Mathematischen Tabelle"*

```
400 FOR I=1 TO 6
410 PRINT CHR$(29);M$(I)
420 NEXT I
```

wurden nun durch die Zeilen

```
580 FOR I=1 TO 6
590 S=4+I*2+.03
600 GOSUB 450: REM * CURSOR POSIT. *
610 PRINT M$(I)
620 NEXT I
```

ersetzt. Wir wollen die Menüpunkte, beginnend mit der 6. Zeile und der 3. Spalte, ausgeben lassen. Da sich nur der Wert der Zeile verändert und der der Spalte konstant bleibt, haben wir den Nachkommawert für S schon bestimmt, nämlich .03.

Geben Sie hier **nicht .3** an, das würde bedeuten, daß die Menüpunkte erst ab der **30. Spalte** ausgegeben würden!

Der Zeilenwert beginnt mit 6 und soll jeweils um zwei Werte bei jeder Ausgabe zunehmen. Hier bietet sich wieder die *Laufvariable "I"* zur eleganten Berechnung der Zeilennummer an. Für die **Zeilen**nummer erhalten wir demnach beim ersten Durchlauf

S=4+1*2

gleich **6**, beim zweiten Durchlauf

S=4+2*2

gleich **8** usw. Damit hätten wir unsere neue Routine schon im Programm eingebaut.

Bei der bisherigen Menütechnik mußten wir immer irgendwelche Tasten, seien es Zahlen- oder Buchstabentasten, betätigen, um die entsprechenden Programme aufzurufen. Ich möchte Ihnen noch eine andere Art der Menütechnik zeigen, bei der Sie nur noch die **Taste**, die den *Cursor nach unten* oder *nach oben* bewegt, zu benutzen brauchen. Haben Sie dann den entsprechenden Menüpunkt angewählt, betätigen Sie nur die **RETURN-Taste**, und das Programm fährt mit dem entsprechenden Menüpunkt fort. Die **Kennzeichnung** des **angewählten Menüpunkts** soll durch die Darstellung desselben in **reverser Schrift** erfolgen. Wir benötigen dazu zwei Felder. Das erste Feld soll die Menüpunkte in normaler Schrift, das zweite Feld in reverser Schrift enthalten. Schauen wir uns auch dazu zunächst das veränderte Programmlisting an.

```
10 REM *****************
20 REM * PROGRAMMSTART *
30 REM *****************
40 COLOR 0,1:COLOR 4,1
50 SCNCLR
60 DIM M$(6): REM * MENUEPUNKTE
70 DIM MR$(6): REM * MENUEPUNKTE REVERS
80 REM  **********************
90 REM  * FELDER AUFFUELLEN *
100 REM **********************
110 FOR I=1 TO 6
120 READ M$(I)
130 MR$(I)=CHR$(18)+M$(I)+CHR$(146)
140 NEXT I
150 DATA "1 QUADRATWURZEL"
160 DATA "2 SINUS"
170 DATA "3 COSINUS"
180 DATA "4 NATUERLICHER LOGARITHMUS"
190 DATA "5 DEKADISCHER LOGARITHMUS"
200 DATA "6 PROGRAMMENDE"
```

```
210 REM *****************
220 REM * CURSOR RECHTS *
230 REM *****************
240 FOR I=1 TO 40
250 CR$=CR$+CHR$(29)
260 NEXT I
270 REM **********************
280 REM * CURSOR NACH UNTEN *
290 REM **********************
300 FOR I=1 TO 25
310 CU$=CU$+CHR$(17)
320 NEXT I 330 GOTO 570
340 REM ******************
350 REM * UNTERPROGRAMME *
360 REM ******************
370 REM
380 REM *************
390 REM * KOPFZEILE *
400 REM *************
410 SCNCLR
420 FOR I=1 TO 40:PRINT"*";:NEXT I
430 PRINT"*                                        *";
440 PRINT"*         MATHEMATISCHE TABELLE          *";
450 PRINT"*                                        *";
460 FOR I=1 TO 40:PRINT"*";:NEXT I
470 RETURN
480 REM ************************
490 REM * CURSORPOSITIONIERUNG *
500 REM ************************
510 PRINT CHR$(19);LEFT$(CU$,S);
520 PRINT LEFT$(CR$,100*(S-INT(S))+.5);
530 RETURN
540 REM *********
550 REM * MENUE *
560 REM *********
570 GOSUB 410
580 FOR I=1 TO 18
590 PRINT"*                                         *";
600 NEXT I
610 FOR I=1 TO 40:PRINT"*";:NEXT I
```

```
620 PRINT CHR$(19);
630 FOR I=1 TO 3:PRINT:NEXT I
640 FOR I=1 TO 6
650 S=4+I*2+.03
660 GOSUB 510: REM CURSOR POSIT.
670 PRINT M$(I)
680 NEXT I
690 S=6.03:GOSUB 510: REM CURSOR POSIT.
700 PRINT MR$(1):Z=1
710 GET MP$:IF MP$="" THEN 710
720 IF ASC(MP$) < > 17 OR Z >= 6 THEN 760
730 S=4+Z*2+.03:GOSUB 510
740 PRINT M$(Z)
750 Z=Z+1:GOTO 820
760 IF ASC(MP$) < > 145 OR Z=1 THEN 800
770 S=4+Z*2+.03:GOSUB 510
780 PRINT M$(Z)
790 Z=Z-1:GOTO 820
800 IF ASC(MP$) = 13 THEN 850
810 GOTO 710
820 S=4+Z*2+.03:GOSUB 510
830 PRINT MR$(Z)
840 GOTO 710
850 ON Z GOSUB 1000,2000,3000,4000,5000,6000
  .
  .
  .
```

In den Zeilen 60 und 70 werden jeweils ein Feld für die Menüpunkte in **normaler Schrift** M$(6) und in **reverser Schrift** MR$(6) dimensioniert. Die Zeilen 110 bis 140 füllen diese beiden Felder mit den entsprechenden Menüpunkten auf. Die reversen Menüpunkte erzeugt die Zeile 130. Der **CHR$(18)-Code** bewirkt, daß die **reverse Schrift** eingeschaltet wird. Der **CHR$(146)-Code** schaltet wieder um auf die **Normalschrift**. Dadurch wird erreicht, daß die Menüpunkte im Feld MR$(6) revers abgelegt werden. Die restlichen Zeilen bis zum Aufbau des Menüs sind inzwischen bekannt. Auch die Zeilen 640 bis 680, in denen die Ausgabe der Menüpunkte programmiert wurde, dürften verstanden worden sein. Interessant wird es

wieder ab Zeile 690. Dort wird der Cursor zunächst in der 6. Zeile und in der 3. Spalte positioniert. Anschließend wird in Zeile 700 der 1. reverse Menüpunkt ausgegeben und der Zähler Z auf eins gesetzt. Zeile 710 enthält die bekannte GET-Abfrage. Die Zeile 720 fragt ab, ob die betätigte Taste verschieden von der Taste war, die den Cursor nach unten bewegt oder ob Z schon größer oder gleich 6 ist. Die Abfrage auf größer oder gleich 6 geschieht deshalb, weil wir insgesamt 6 Menüpunkte haben. Beim Erreichen des letzten Menüpunktes muß ja die Taste, die den Cursor nach unten bewegt, im folgenden als CHR$(17) benannt, blockiert sein. Wurde nun die CHR$(17)-Taste betätigt, so muß zunächst der reverse Menüpunkt 1 wieder in Normalschrift erscheinen. Das bewirken die Zeilen 730 und 740. Die Zeile 750 erhöht den Zähler um den Wert 1. Anschließend wird zur Zeile 820 verzweigt, um den nächsten Menüpunkt in reverser Schrift ausgeben zu können *(Zeilen 820 bis 840)*. Die Zeile 760 ist genauso zu interpretieren wie die Zeile 720, nur daß hier nicht die CHR$(17)-Taste abgefragt wird, sondern die CHR$(145)-Taste *(Cursor nach oben)*. Hier darf der Zähler natürlich nicht kleiner als eins werden. Wurde nun die CHR$(145)-Taste betätigt, so wird wieder der momentan in reverser Schrift dargestellte Menüpunkt in Normalschrift ausgegeben *(Zeilen 770 und 780)*. Danach wird der Zähler Z um eins vermindert und anschließend zur Zeile 820 verzweigt. Dort wird dann der angesprochene Menüpunkt wieder in reverser Schrift ausgegeben. Betätigen Sie die RETURN-Taste, so wird von Zeile 720 über Zeile 760 nach Zeile 800 verzweigt. Von dort wird wiederum nach Zeile 850 verzweigt, in der dann entsprechend dem Wert von Z mit **ON Z GOSUB** die weiteren Unterprogramme aufgerufen werden können. Sie haben hier selbstverständlich die Möglichkeit, auch mit ON Z GOTO zu arbeiten.

Ich hoffe, daß durch diese ausführliche Beschreibung dieses Prinzip der Menütechnik von Ihnen verstanden worden ist.

4.4 WINDOW-Techniken

Der Commodore 128 besitzt die Möglichkeit, über den Befehl

WINDOW

Bildschirmfenster zu definieren. Diese Fenster können für bestimmte Aufgaben eingerichtet werden, um so dem Anwender mitzuteilen, daß er z.B. eine Datendiskette in das Diskettenlaufwerk legen soll. Wurde ein solches Fenster einmal definiert, so werden alle Bildschirmausgaben in diesem Fenster angezeigt. Weiterhin finden auch Eingaben, z.B. mit INPUT, in diesem Fenster statt.

Es können mehrere Fenster nacheinander eingerichtet werden. Die Aus- und Eingaben finden jedoch immer in dem Fenster statt, das zuletzt generiert wurde.

Die Schreibweise des WINDOW-Befehls sieht wie folgt aus:

WINDOW Sl,Zl,Sr,Zr,C

Die Parameter haben dabei folgende Bedeutung:

Sl	linke obere Spalte
Zl	linke obere Zeile
Sr	rechte untere Spalte
Zl	rechte untere Zeile
C	1=SCNCLR / 0=kein SCNCLR

Die Maximalwerte der Parameter für Zeilen und Spalten werden vom jeweiligen Bildschirm bestimmt. Beim 40-Zeichenbildschirm liegt der Spaltenbereich zwischen 0 und 39 und der Zeilenbereich zwischen 0 und 24. Der 80-Zeichenschirm hat die Spalten-parameter 0 bis 79 und die Zeilenparameter 0 bis 24.

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

```
10 WINDOW 0,24,39,24
20 PRINT " MELDEZEILE !";
```

Starten Sie dieses Programm, so werden Sie von der Meldung wahrscheinlich nicht viel mitbekommen. Das einzige, was Sie sehen, ist der Cursor, der nun in der unteren linken Bildschirmecke steht. Was ist geschehen?

Das Programm ist soweit in Ordnung. Definieren wir aber ein Fenster mit nur einer Zeile, so wird die anschließende Meldung

READY.

ebenfalls in dieser Zeile ausgegeben, nachdem das Programm beendet wurde. Danach erfolgt noch ein Zeilenvorschub, und deshalb sehen wir jetzt nur noch den Cursor.

Wir müssen also, nachdem die Meldung ausgegeben wurde, dieses Fenster wieder verlassen. Dazu gibt es innerhalb eines Programms zwei Möglichkeiten.

Die erste Möglichkeit besteht darin, daß wir ein neues Fenster definieren, und zwar mit den Maximalwerten des aktuellen Bildschirms. Das wäre beim 40-Zeichenschirm der folgende Befehl:

WINDOW 0,0,39,24

Damit hätten wir wieder den kompletten Bildschirm zu unserer Verfügung.

Bei der zweiten Möglichkeit finden einmal wieder die CHR$-Codes Verwendung. Man kann ein **Fenster im Direktmodus verlassen**, indem man **zweimal die CLR/HOME-Taste betätigt**. Diese Taste hat jedoch auch einen CHR$-Code, so daß wir damit eine Möglichkeit haben, ein Fenster ebenfalls im Programmodus zu verlassen, ohne den WINDOW-Befehl zu

benutzen. Diese zweite Möglichkeit bietet sich immer dann an, wenn Sie Ihre Ausgaben wieder auf dem normalen Bildschirm anzeigen lassen wollen.

Haben Sie dagegen die Absicht, in ein weiteres Fenster umzuschalten, so müssen Sie dazu wieder den WINDOW-Befehl benutzen.

Wir wollen jetzt unser Beispiel um eine Programmzeile erweitern.

```
10 WINDOW 0,24,39,24
20 PRINT " MELDEZEILE !";
30 PRINT CHR$(19);CHR$(19)
```

Zeile 30 gibt zweimal den CHR$(19)-Code *(CHR$-Code der CLR/HOME-Taste)* aus. Damit wird wieder der normale Bildschirm angewählt.

Starten Sie jetzt dieses Programm, so wird in der unteren Bildschirmzeile die Meldung ausgegeben, und anschließend meldet sich der Rechner wieder in der linken oberen Bildschirmecke mit **READY**.

Leider gibt es beim Commodore 128 keine Möglichkeit, die Fenster auf dem 40-Zeichenschirm optisch besonders hervorzuheben. Für dieses Problem bietet das folgende Programm eine Lösung.

```
10 SCNCLR
20 REM RW$=40MAL CHR$(192)
30 RW$=CHR$(192)+CHR$(192)+CHR$(192)+CHR$(192)+CHR$(192)
40 REM BL$=40MAL CHR$(32)
50 BL$=CHR$(32)+CHR$(32)+CHR$(32)+CHR$(32)+CHR$(32)
60 FOR I=1 TO 3
70 RW$=RW$+RW$:BL$=BL$+BL$
80 NEXT I
90 RS$=CHR$(221)
100 REM PARAMETER FUER FENSTER
110 SL=4:ZL=8:SR=14:ZR=18
```

```
12Ø REM RAHMEN ZEICHNEN
13Ø CHAR ,SL-1,ZL-1,CHR$(176)+LEFT$(RW$,SR-SL+1)+CHR$(174)
14Ø FOR I=ZL TO ZR
15Ø CHAR ,SL-1,ZL+Z,RS$+LEFT$(BL$,SR-SL+1)+RS$
16Ø Z=Z+1
17Ø NEXT
18Ø CHAR ,SL-1,ZR+1,CHR$(173)+LEFT$(RW$,SR-SL+1)+CHR$(189)
19Ø WINDOW SL,ZL,SR,ZR
2ØØ END
```

Dieses Programm zeichnet einen Rahmen um jedes beliebige Bildschirmfenster. Sie dürfen allerdings nicht die Maximalwerte für die Fenster benutzen. Jede andere Definition ist zulässig. Sie können sich das Programm jedoch entsprechend ausbauen.

Die Stringvariablen in den ersten Zeilen wurden über die CHR$-Codes erzeugt, weil diese Methode keine Mißverständnisse aufkommen läßt. Sie können selbstverständlich auch direkt die entsprechenden Grafikzeichen einsetzen.

Der Rahmen des Fensters wird mit der Anweisung CHAR zuerst um das Fenster ausgegeben. Danach erst wird das Fenster generiert *(Zeile 190)*.

Dieses Programm läßt sich auch gut als Unterprogramm gestalten. Die vorgegebenen Parameter in Zeile 110 können dann mit an das Unterprogramm übergeben werden *(damit entfällt dann die Zeile 110)*. Sie **initialisieren** daher in einem Programm die Parameter **SL**, **ZL**, **SR** und **ZR** und springen dann in diese Unterroutine, die den Rahmen zeichnet und das Fenster einrichtet.

In Verbindung mit dem Befehl WINDOW darf die Anweisung

RWINDOW (X)

nicht unerwähnt bleiben. Mit **RWINDOW** können Sie die Zeilen- und Spaltenzahl sowie den Zeichenmodus *(40- oder 80-Zeichen)* bestimmen. Dabei darf X die folgenden Werte annehmen:

0 ergibt Anzahl der Zeilen des Fensters
1 ergibt Anzahl der Spalten des Fensters
2 ergibt 40 oder 80, je nach Art des Zeichenmodus'

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.

4.5 Sortierverfahren

Bei vielen Programmen wird es sich als nötig erweisen, die anfallenden Daten nach verschiedenen Ordnungskriterien *(der Größe nach, in alphabetischer Reihenfolge usw.)* zu sortieren. Es existieren hier eine **Menge unterschiedlicher Verfahren**, die sich untereinander in **Leistung** und **Schwierigkeitsgrad unterscheiden**. Allgemein kann man sagen, daß ein Verfahren umso schwieriger zu durchschauen ist, je schneller es die anfallenden Daten sortieren kann. Wir wollen daher hier nur ein einfaches Verfahren kennenlernen, damit Sie zumindest eine Einführung in diese Materie erhalten. Für einen Anfänger ist es eher abschreckend als anregend, die komplizierteren Verfahren kennenzulernen. Sollten Sie einmal etwas Erfahrung im Umgang mit einfachen Sortierungen gesammelt haben, so können Sie sich dann an die komplizierteren Strukturen dieser Verfahren wagen. Es gibt eine Reihe von Fachbüchern, in denen diese ausführlich beschrieben sind.

Wir wollen uns nun mit dem sogenannten *Bubble-Sort*-Verfahren vertraut machen. Der Name *Bubble-Sort* rührt wohl daher, daß bei diesem Verfahren die **einzelnen Elemente der Größe** nach wie **Blasen** *(engl. Bubble)* im Wasser **nach oben steigen**. Als Beispiel wollen wir ein Feld mit Zufallszahlen auffüllen und dieses sortiert ausgeben lassen. Wir nehmen ein Feld mit 6 Elementen. Zunächst die Programmzeilen, die das Feld dimensionieren und mit Werten auffüllen:

```
10 REM FELD GENERIEREN
20 DIM F(6)
30 FOR I=1 TO 6
40 A=INT(50*RND(0))+1
50 F(I)=A
60 NEXT I
 .
 .
 .
```

Das Prinzip unseres Sortierverfahrens beruht darauf, daß immer **zwei benachbarte Elemente** miteinander **verglichen** werden. Ist ein Element größer als das andere, so findet eine Vertauschung statt. Somit werden nacheinander alle Elemente miteinander verglichen. Damit dies deutlich wird, realisieren wir das Verfahren mit IF...THEN-Abfragen. Man könnte es auch mit einer FOR...NEXT-Schleife programmieren, dabei wird allerdings das Verfahren nicht so deutlich. Haben Sie das Verfahren einmal verstanden, können Sie es durchaus mit einer FOR...NEXT-Schleife ausführen lassen. Doch nun zu den eigentlichen Programmzeilen:

```
.
.
.
100 REM BUBBLE-SORT
110 Z=0
120 IF F(6) > F(5) THEN 140
130 F(0)=F(6):F(6)=F(5):F(5)=F(0):Z=1
140 IF F(5) > F(4) THEN 160
150 F(0)=F(5):F(5)=F(4):F(4)=F(0):Z=1
160 IF F(4) > F(3) THEN 180
170 F(0)=F(4):F(4)=F(3):F(3)=F(0):Z=1
180 IF F(3) > F(2) THEN 200
190 F(0)=F(3):F(3)=F(2):F(2)=F(0):Z=1
200 IF F(2) > F(1) THEN 220
210 F(0)=F(2):F(2)=F(1):F(1)=F(0):Z=1
220 IF Z=1 THEN 110
230 FOR I=1 TO 6
240 PRINT F(I)
250 NEXT I
260 END
```

In **Zeile 110** wird zunächst **Z auf Null** gesetzt. Warum das so ist, werden Sie im weiteren Verlauf erkennen können. Zeile 120 führt nun den ersten Vergleich durch. Ist der Inhalt des Elements von F(6) bereits größer als von F(5), so braucht keine Vertauschung vorgenommen zu werden und es kann direkt nach Zeile 140 verzweigt werden. Ist F(6) allerdings kleiner als F(5), so erfolgt in Zeile 130 eine Vertauschung. Das Prinzip dieser Vertauschung dürfte Ihnen schon bekannt sein. Wir verwenden hier das Element F(0) zur Zwischenspeicherung eines Variablenwertes. Danach wird der Wert von F(5) dem Element F(6) zugeordnet. Anschließend erhält F(5) den Wert von F(0), also von F(6). Dieses Prinzip wurde auch in den anderen Programmzeilen angewendet.

Danach wird Z auf eins gesetzt, da eine Vertauschung stattgefunden hat. Wir können also anhand des Zustandes von Z erkennen, ob eine Vertauschung stattgefunden hat oder nicht. Hat Z den Wert 1, so wurde ausgetauscht, hat Z den Wert 0, so wurde nicht ausgetauscht. Dieser "Trick" wird häufig in der

Programmierung angewandt, um überprüfen zu lassen, ob bestimmte Vorgänge stattgefunden haben oder nicht. Diese Variablen, wie in unserem Beispiel Z, nennt man auch *Flags*. Flag bedeutet soviel wie Flagge oder Zeichen. Hat Z also nach einem Durchlauf immer noch den Wert Null, so wissen wir, daß keine Vertauschung stattfand und daß wir somit unser Feld sortiert vorliegen haben. Das hat den Vorteil, daß die Sortierung bereits nach einem Durchlauf abgebrochen werden kann, wenn schon zufällig die richtige Reihenfolge der Elemente vorliegt. Dieses Verfahren findet man auch unter dem Namen *"Bubble-Sort mit Weiche"*. Weiche deshalb, weil eben jederzeit nach Erreichen der zufälligen Sortierung das Verfahren abgebrochen werden kann. Die letzten Zeilen geben dann das sortierte Feld aus. Wollen Sie beobachten, wie die Sortierung im einzelnen stattfindet, so ändern Sie die letzten Programmzeilen bitte wie folgt ab:

```
.
.
.
220 FOR I=1 TO 6
230 PRINT F(I);
240 NEXT I
250 PRINT
260 IF Z=1 THEN 110
270 END
```

Damit wären wir schon am Ende der Besprechung dieses Sortierverfahrens angelangt. Beschäftigen Sie sich eingehend mit dieser Materie, so daß Sie auch später auf kompliziertere Verfahren zurückgreifen können, womit Sie sich einen zeitlichen Vorteil verschaffen. Das *Bubble-Sort*-Verfahren ist geeignet für eine Anzahl bis zu etwa 100 Elementen, die dann in einer noch akzeptablen Zeitspanne sortiert werden.

5

DAS PRINZIP

DER DATEIVERWALTUNG

5. Das Prinzip der Dateiverwaltung

5.1 Allgemeines zur Datenspeicherung

In diesem Kapitel werden Sie die Grundlagen der Dateiverwaltung kennenlernen. Es werden die wichtigsten Befehle bzw. Anweisungen anhand von Beispielen erklärt. Eine komplette Auflistung der Befehle und Funktionen mit Beispielen zur Dateiverwaltung finden Sie im Anhang dieses Buches.

Bevor wir uns nun näher mit der Dateiverwaltung beschäftigen, seien hier einige Informationen zum Umgang mit der Diskette gegeben. Verwenden Sie eine neue Diskette zum ersten Mal, so muß sie vorher **formatiert** werden. Beim Formatieren werden bereits einige Daten auf der Diskette vom Betriebssystem des Laufwerks untergebracht. Diese Daten sind für das Laufwerk unbedingt erforderlich, damit es später Ihre abgespeicherten Daten wiederfinden kann. Zum Formatieren der Diskette dient der folgende Befehl:

```
HEADER"DATENDISK",I85
```

Dabei ist "**DATENDISK**" der **Name der Diskette** und "**85**" die **ID-Nummer** *(Identifikationsmerkmal)* der Diskette. Diese Angabe wird unbedingt benötigt. Der Name darf sich aus maximal 16 Zeichen zusammensetzen. Haben Sie diesen Befehl eingegeben und die RETURN-Taste betätigt, so gibt der Computer noch eine Sicherheitsabfrage

```
ARE YOU SURE?
```

(Sind Sie sicher?) aus. Betätigen Sie jetzt die Y-Taste für "JA", so wird mit der Formatierung begonnen. Dieser Vorgang dauert ca. 80 Sekunden. Formatieren Sie eine *alte Datendiskette*, sind alle auf dieser Diskette abgespeicherten Daten verloren. Vergewissern Sie sich also vorher, welche Diskette Sie formatieren wollen.

Die Diskette besitzt ein Inhaltsverzeichnis, in dem alle Programme und Dateien aufgeführt sind. Dieses Inhaltsverzeichnis wird ebenfalls schon bei der Formatierung mit angelegt. Natürlich beinhaltet es noch keine Einträge. Sie können sich mit dem Befehl

DIRECTORY

oder dem Befehl

CATALOG

das Inhaltsverzeichnis einer Diskette anzeigen lassen. Das Inhaltsverzeichnis einer neu formatierten Diskette könnte z.B. so aussehen:

```
Ø "DATENDISK       " 85 2A
664 BLOCKS FREE.
```

Dieses Inhaltsverzeichnis erhalten Sie auf einem 1541-Laufwerk. Verwenden Sie dagegen ein 1571-Laufwerk, so stehen Ihnen statt 664 Blöcke genau 1328 Blöcke zur Verfügung *(falls die Diskette auf diesem Laufwerk formatiert wurde)*, also genau die doppelte Anzahl.

5.2 Verschiedene Dateitypen

Speichern Sie ein Programm auf der Diskette ab, wird der Programmname in das Inhaltsverzeichnis übernommen. Zusätzlich wird noch ein aus **drei Zeichen** bestehendes **Kürzel** mit in dem Inhaltsverzeichnis abgelegt. Dieses Kürzel wird zur näheren Bestimmung der Art einer Datei verwendet. Es existieren vier Kürzel:

PRG = Programmdatei
SEQ = sequentielle Datei
USR = USER Datei (Anwender Datei)
REL = relative Datei

Mit **PRG** wird eine normale **Programmdatei** gekennzeichnet, die Sie mit **DSAVE"Programmname"** abgespeichert haben. Eine **sequentielle Datei**, in der Sie z.B. Daten abspeichern, erhält das Kürzel **SEQ**. Das Kürzel **USR** kennzeichnet eine Datei, die vom Anwender für verschiedene Zwecke genutzt werden kann. In ihr können sowohl normale Daten wie Namen usw. abgespeichert werden als auch Daten für Sprites, die dann im Binärformat vorliegen.

Eine besondere Art der Datei wird durch das Kürzel **REL** gekennzeichnet. Die **relative Datei** unterscheidet sich gegenüber der sequentiellen Datei wohl hauptsächlich durch die **enorme Zeitersparnis** beim Datenzugriff. Wollen Sie bei der sequentiellen Datei z.B. den 54. Datensatz lesen, so müssen Sie zuerst die ersten 53 Datensätze überlesen, um dann auf den 54. Datensatz zugreifen zu können. Bei der **relativen Datei** können Sie **direkt auf** den **54. Datensatz zugreifen**, ohne die ersten 53 Sätze lesen zu müssen. Daß dabei sehr viel Zeit eingespart wird, versteht sich fast von selbst. Dieser Zeitunterschied macht die relative Datei u.a. so beliebt.

5.3 Die Datei

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 Speicher-

geräten arbeiten, so benötigen Sie als erstes die Befehle **DLOAD** und **DSAVE**. Da sich eine rationelle Dateiverwaltung nur mit dem Diskettenlaufwerk realisieren läßt, wollen wir uns hier auf den Umgang mit diesem beschränken. Um ein Programm von Diskette zu laden, geben Sie folgendes ein:

DLOAD"Programmname"

oder

LOAD"Programmname",GA

GA steht beim zweiten Beispiel für die Geräteadresse, welche beim Diskettenlaufwerk meistens den Wert 8 besitzt. Verwenden Sie **DLOAD**, so brauchen Sie die **Geräteadresse nicht** mit anzugeben. Innerhalb der Anführungszeichen dürfen **maximal 16 Zeichen** verwendet werden, so daß Ihre Befehlsfolge jetzt wie folgt aussehen könnte:

DLOAD"BEISPIEL"

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

DSAVE"BEISPIEL"

Die genaue Anwendung dieser und der noch folgenden Befehle entnehmen Sie bitte dem Handbuch zum Commodore 128 bzw. der Befehlsübersicht in diesem Buch. Denjenigen, die sich eingehender mit der Materie des Diskettenlaufwerks und den sich daraus ergebenden phantastischen Möglichkeiten befassen wollen, steht ein sehr umfangreiches Literaturangebot zur Verfügung.

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 einer Diskette abgespeichert. Innerhalb des Programms hat man dann meistens die Möglichkeit, nach bestimmten Daten suchen zu lassen, diese zu verändern und wieder auf die Diskette zurück-

zuschreiben. Wollen Sie Daten an das Diskettenlaufwerk übergeben, müssen Sie zuerst eine **Datei eröffnen.** Dies geschieht mit der **DOPEN**-Anweisung. Die Befehlsfolge

DOPEN#1,"ADRESSEN",W

öffnet beim Diskettenlaufwerk einen Kanal zur Datenübertragung und gleichzeitig die **sequentielle Datei** "ADRESSEN" zum *Schreiben (W=WRITE)*, d.h. Sie können nun Daten zum Laufwerk übertragen. **Sequentiell** bedeutet, daß die Daten **hintereinander abgespeichert** werden. Hierzu benötigen Sie den

PRINT#

Befehl. Dem PRINT# folgt noch eine Zahl *(logische Filenummer)*, die sich auf die zuvor mit OPEN geöffnete Datei bezieht, also in unserem Falle PRINT#1. Geben Sie nun den Befehl **PRINT#1,D$** ein, so wird der Inhalt der Variablen D$ auf die Diskette geschrieben. Wollen Sie die Datenübertragung beenden, so müssen Sie die Datei mit dem Befehl

DCLOSE

wieder schließen. Auch hier wird wieder die Zahl der zuvor geöffneten Datei mit angegeben, also **DCLOSE#1**.

Damit wissen wir nun, wie eine sequentielle Datei auf der Diskette angelegt werden kann. Wie bekommen wir aber nun unsere abgespeicherten Daten wieder in den Computer? Auch dazu muß die entsprechende Datei erst wieder geöffnet werden. Da wir diesmal die Daten aber lesen wollen, ändert sich der DOPEN-Befehl wie folgt:

DOPEN#1,"ADRESSEN"

Damit wird die sequentielle Datei **"ADRESSEN"** zum Lesen geöffnet. In diesem Fall benötigen wir kein zusätzliches Kürzel. Zum **Einlesen der Daten** verwenden wir den **INPUT#** Befehl. Hier muß ebenfalls wieder die Zahl der zuvor geöffneten Datei mit angegeben werden. Diese Zahl nennt sich, wie bereits

erwähnt, "**logische Filenummer**". Wollen wir in unserem Programm also Daten von der Diskette lesen, müssen wir folgende Befehlsfolge verwenden:

INPUT#1,D$

Mit diesem Befehl können Sie **Daten** mit einer **maximalen Länge von 80 Zeichen** einlesen. Mit dem **GET#**-Befehl wird, wie beim "normalen" GET-Befehl auch, jeweils ein Zeichen von der Diskette gelesen.

Ein weiterer Befehl ermöglicht es Ihnen, eine Datei auf der Diskette zu löschen. Dieser Befehl lautet:

SCRATCH *"Dateiname"*

Nachdem Sie diesen Befehl eingegeben haben, erscheint die Sicherheitsabfrage

ARE YOU SURE?

(Sind Sie sicher?). Betätigen Sie jetzt die Y-Taste für "JA", wird die Datei aus dem Inhaltsverzeichnis gelöscht.

SCRATCH "ADRESSEN"

Mit dieser Befehlsfolge wird das File "**ADRESSEN**" auf der Diskette gelöscht. Im nachfolgenden Programm wurde dieser Befehl verwendet, um eine Datei neu auf die Diskette abspeichern zu können. Existiert nämlich bereits eine Datei gleichen Namens auf der Diskette, so beginnt die rote Leuchtdiode am Laufwerk zu blinken und signalisiert damit einen Fehler. Fragt man diesen Fehler über die **Variable DS$** ab, so erhält man als Ausgabe die Meldung

63 FILE EXISTS

Mit diesem Wissen sollten Sie nun das nachfolgende Programm verstehen können. Sie können in einem Menü auswählen, ob Sie Daten eingeben, speichern, laden oder ausgeben lassen oder das

Programm beenden wollen. Wählen Sie **"Daten eingeben"**, haben Sie die Möglichkeit, vier Namen mit Vornamen einzugeben.

Das Programm ist bewußt einfach gehalten, um Ihnen das Prinzip einer sequentiellen Datenverwaltung aufzeigen zu können. Mit dem bisher erworbenen Wissen stehen Ihnen alle Wege offen, sich aus diesem Ansatz einer Dateiverwaltung Ihre eigene Dateiverwaltung zu schreiben.

```
10 DIM D$(4,2)
20 SCNCLR
30 PRINT"WOLLEN SIE"
40 PRINT
50 PRINT"DATEN EINGEBEN ? (1)"
60 PRINT
70 PRINT"DATEN SPEICHERN ? (2)"
80 PRINT
90 PRINT"DATEN LADEN ? (3)"
100 PRINT
110 PRINT"DATEN AUSGEBEN ? (4)"
120 PRINT
130 PRINT"BEENDEN ? (5)"
140 GETKEY A$
150 ON VAL(A$) GOTO 190,290,400,500,600
160 REM ******************
170 REM * DATEN EINGEBEN *
180 REM ******************
190 SCNCLR
200 FOR I=1 TO 4
210 INPUT"NAME";D$(I,1)
220 INPUT"VORNAME";D$(I,2)
230 SCNCLR
240 NEXT I
250 GOTO 20
260 REM ********************
270 REM * DATEN SPEICHERN *
280 REM ********************
290 SCRATCH"ADRESSEN"
300 DOPEN#1,"ADRESSEN",W
310 FOR I=1 TO 4
```

```
320 FOR Z=1 TO 2
330 PRINT#1,D$(I,Z)
340 NEXT Z,I
350 DCLOSE#1
360 GOTO 20
370 REM ***************
380 REM * DATEN LADEN *
390 REM ***************
400 DOPEN#1,"ADRESSEN"
410 FOR I=1 TO 4
420 FOR Z=1 TO 2
430 INPUT#1,D$(I,Z)
440 NEXT Z,I
450 DCLOSE#1
460 GOTO 20
470 REM ******************
480 REM * DATEN AUSGEBEN *
490 REM ******************
500 SCNCLR
510 FOR I=1 TO 4
520 FOR Z=1 TO 2
530 PRINT D$(I,Z)
540 NEXT Z,I
550 SLEEP 3
560 GOTO 20
570 REM ********
580 REM * ENDE *
590 REM ********
600 SCNCLR
610 END
```

Wie bereits erwähnt, gibt es noch eine weitere Art der Dateiverwaltung, die relative Dateiverwaltung.

5.4 Relative Dateiverwaltung

Die relative Dateiverwaltung hat u.a. den Vorteil, daß nicht permanent jeder Datensatz einer Datei im Speicher des Computers vorliegen muß. Wollen wir z.B. einen bestimmten Datensatz ändern, so lesen wir diesen Datensatz ein, ändern ihn und speichern ihn wieder ab.

Zunächst muß die **relative Datei** aber auf der Diskette **angelegt werden**. Anders als bei der sequentiellen Dateiverwaltung wird bei der **relativen Dateiverwaltung vorher Speicherplatz** auf der **Diskette reserviert**. In einer relativen Datei darf ein Datensatz **maximal 254 Zeichen** umfassen. Das bedeutet, daß alle Datenfelder eines Datensatzes nicht mehr als 254 Zeichen beinhalten können. Diese Anzahl ist aber für die meisten Anwendungen vollkommen ausreichend. Um aber keinen Speicherplatz zu verschenken, müssen wir uns vorher überlegen, wie viele Zeichen unser Datensatz ungefähr beinhalten soll.

Sagen wir, wir wollen in einem Datensatz maximal 100 Zeichen abspeichern. Die entsprechende Anweisung zur Erzeugung der relativen Datei sieht dann wie folgt aus:

```
DOPEN#1,"REL-ADRESSEN",L100
```

Mit dieser Anweisung wird dem Betriebssystem der Floppy **lediglich mitgeteilt**, daß ein Datensatz maximal 100 Zeichen umfaßt. Bislang wurde aber **noch kein Speicherplatz** auf der Diskette für unsere Datensätze **reserviert**. Hierzu müssen wir entscheiden, wieviel Datensätze wir anlegen wollen.

Wir entscheiden uns für 200 Datensätze. Wie aber teilen wir jetzt dem Diskettenlaufwerk mit, daß es für 200 Datensätze Speicherplatz reservieren soll?

Nun, das BASIC 7.0 kennt eine Anweisung, die speziell bei der relativen Dateiverwaltung eine Verwendung findet. Es handelt sich hierbei um die

RECORD

Anweisung. Die allgemeine Schreibweise sieht wie folgt aus:

RECORD#lfn,dnr,bnr

Dabei bedeuten **lfn** die logische Filenummer der mit **DOPEN#** geöffneten Datei, **dnr** die Datensatznummer und **bnr** die Bytenummer des Datensatzes. Mit dieser Anweisung kann man also den Datensatzzeiger sofort auf jeden beliebigen Datensatz der relativen Datei setzen.

Diese RECORD-Anweisung benutzen wir nun, um den Datensatzzeiger auf den 200. Datensatz zu setzen, obwohl dieser noch gar nicht existiert! Das Betriebssystem erkennt nun, daß dieser Datensatz noch nicht existiert und erzeugt eine Fehlermeldung,

50, RECORD NOT PRESENT,00,00

was wir am Blinken der roten Leuchtdiode erkennen können *(grüne Leuchtdiode bei der 1571)*. Diese Fehlermeldung können wir dieses Mal ruhig ignorieren. Dann schreiben wir in diesen Datensatz den CHR$(255)-Code. Dabei wird dann automatisch Speicherplatz für sämtliche 200 Datensätze reserviert. Das folgende Beispiel legt eine relative Datei mit 200 Datensätzen bei einer Datensatzlänge von 100 Zeichen an.

```
10 REM ANLEGEN RELATIVE DATEI
20 DOPEN#1,"ADRESSEN-REL",L100
30 RECORD#1,200
40 PRINT#1,CHR$(255)
50 DCLOSE#1
60 END
```

Nach Beendigung dieses Programms werden Sie feststellen, daß die rote bzw. grüne Leuchtdiode an Ihrem Laufwerk blinkt. Geben Sie jetzt

PRINT DS$

ein, erhalten Sie als Ausgabe die o. a. Fehlermeldung. Mit der Variablen **DS$** und **DS** können Sie jederzeit den Status des Diskettenlaufwerks erfahren.

Schauen Sie sich jetzt mit CATALOG oder DIRECTORY das Inhaltsverzeichnis der Diskette an, werden Sie feststellen, daß Ihre relative Datei 80 Blöcke belegt.

Ein Block auf der Diskette beinhaltet 256 Bytes. Davon müssen zwei Bytes als Zeiger auf den nächsten Block abgezogen werden. Die relative Datei benötigt 200 * 100 = 20000 Bytes an Speicherplatz für die Datensätze. Hinzu kommt noch ein Block, der sogenannte *Sidesectorblock*, in der die Tabelle für die einzelnen Datensätze abgespeichert ist. 20000 dividiert durch 254 ergibt ungefähr 78.7. Da im Inhaltsverzeichnis keine Halbblöcke belegt werden können, werden 79 plus 1 Block, also 80 Blöcke belegt.

Durch dieses kleine Rechenbeispiel können Sie selbst anhand der freien Blöcke feststellen, wie groß Ihre relative Datei werden darf.

Das wäre soweit das Wichtigste zur relativen Dateiverwaltung. Sie sind jetzt in der Lage, das Programm der sequentiellen Dateiverwaltung in eine relative Dateiverwaltung abzuändern.

Damit soll das Kapitel über die Dateiverwaltung abgeschlossen sein. Im nächsten Kapitel besprechen wir noch kurz die wichtigsten Befehle zur Musik- und Grafikprogrammierung.

6

MUSIK

UND GRAFIK

6. Musik und Grafik

In diesem Kapitel sollen die wichtigsten Befehle des BASIC 7.0, 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 bzw. Anweisungen beschränkt.

6.1 Musik

Das BASIC 7.0 stellt dem Anwender insgesamt 6 Befehle bzw. Anweisungen für die Programmierung von Musik zur Verfügung. Der Commodore 128 besitzt den gleichen SID *(Sound Interface Device)* wie der Commodore 64. Diejenigen Leser unter Ihnen, die bereits dem SID beim Commodore 64 mit POKE- und PEEK-Befehlen Töne entlockt haben, werden diese Befehle besonders zu schätzen wissen.

Zunächst sollen einige generelle Begriffe zum Sound bzw. zur Musik geklärt werden. Einfache musikalische Klänge setzen sich aus drei Hauptcharakteristika zusammen:

Frequenz *(Tonhöhe)*
Amplitude *(Lautstärke)*
Dauer

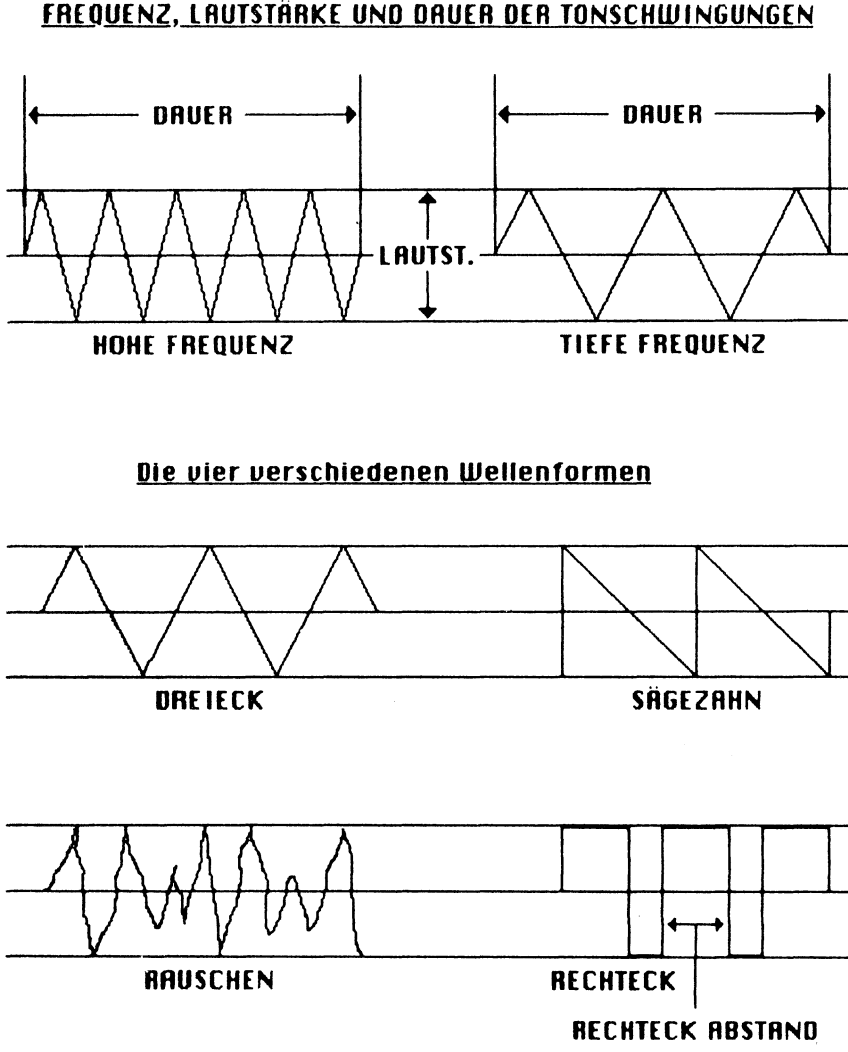

FREQUENZ, LAUTSTÄRKE UND DAUER DER TONSCHWINGUNGEN
DAUER
DAUER
LAUTST.
HOHE FREQUENZ
TIEFE FREQUENZ
Die vier verschiedenen Wellenformen
DREIECK
SÄGEZAHN
RAUSCHEN
RECHTECK
RECHTECK ABSTAND

Bild 14

Weiterhin bestimmt die Wellenform, ob ein Ton z.B. *weicher* oder eher *blechern* klingt. Der SID stellt insgesamt vier Wellenformen zur Verfügung:

Dreieck
Sägezahn
Rechteck
Rauschen

Ein weiterer Bestandteil, der den Verlauf eines Tones bestimmt, ist die Hüllkurve. In ihr werden die **Anklingphase, Haltephase** und **Ausklingphase** eines Tones festgelegt *(Attack, Decay und Release).*

Diese Begriffe sollen vorerst ausreichen, um die nachfolgenden Befehle verstehen zu können. Wollen Sie sich eingehender mit der Programmierung des SID auseinandersetzen, so finden Sie in dem Buch *"128 Intern"* dazu ausführliche Informationen.

Die einfachste Art Musik zu erzeugen bietet wohl der

PLAY

Befehl. Sie übergeben einfach dem PLAY-Befehl die Noten in Anführungszeichen, z.B.

```
PLAY "C E G C"
```

und schon vernehmen Sie aus dem Lautsprecher die entsprechenden Töne. Der PLAY-Befehl hat die folgende allgemeine Schreibweise:

```
PLAY "Vn,On,Tn,Un,Xn,NOTEN"
```

Dabei haben die einzelnen Parameter folgende Bedeutung:

Vn	Stimme (n=1-3)
On	Oktave (n=0-6)
Tn	Hüllkurve (n=0-9)
Un	Lautstärke (n=0-15)
Xn	Filter (0=aus/1=ein)

Ändern Sie das o.a. Beispiel wie folgt ab:

PLAY "O4 C E G O5 C"

Wie Sie hören, können wir mit dem **Parameter "O"** *(Buchstabe O, nicht NULL)* bestimmen, in welcher Oktave die Melodie gespielt werden soll. Der Commodore 128 kann Töne über insgesamt 7 Oktaven wiedergeben (0-6).

Weiterhin können Sie mit dem Parameter "T" die Hüllkurve bestimmen. Es stehen Ihnen insgesamt 10 vorgegebene Hüllkurven zur Verfügung:

0	**Klavier**
1	**Akkordeon**
2	**Zirkusorgel**
3	**Schlagzeug**
4	**Flöte**
5	**Gitarre**
6	**Cembalo**
7	**Orgel**
8	**Trompete**
9	**Xylophon**

Wollen Sie unseren kleinen Akkord also auf einer Gitarre erklingen lassen, so geben Sie folgendes ein:

PLAY "T5 O4 C E G O5 C"

Diese vorgegebenen Hüllkurven können Sie mit der Anweisung

ENVELOPE nr,a,d,s,r,w,p

verändern. Die Parameter bedeuten dabei im einzelnen:

nr	Nummer der Hüllkurve (0-9)
a	Attack (0-15)
d	Decay (0-15)
s	Sustain (0-15)
r	Release (0-15)
w	Wellenform (0-4) 0 = Dreieck 1 = Sägezahn 2 = Rechteck 3 = Rauschen 4 = Ringmodulation
p	Pulsverhältnis für w=2 (0-4096)

Mit dieser Anweisung können Sie die einzelnen Hüllkurven bzw. Instrumente nach Ihrem eigenen Geschmack verändern.

Ein weiterer Befehl, der zum einfachen Erzeugen von Klängen benutzt werden kann, ist der

SOUND s,f,d (,r,m,sc,w,p)

Befehl. Die Parameter bedeuten dabei im einzelnen:

s	Stimme (1-3)
f	Frequenz (0-65535)
d	Dauer in 1/60tel Sekunde (0-32767)
r	Frequenz 0 = anschwellend, 1 = abschwellend, 2 = oszillierend
m	Minimal- bzw. Maximalwert der Frequenz für Parameter r (0-65535)
sc	Schrittweite für r (0-65535)
w	Wellenform (0-3)
p	Pulsbreite (Rechteckabstand)

Die folgende Befehlsfolge könnte z.B. eine Sirene darstellen.

SOUND 1,4000,250,2,1000,100

Die Anweisungen **TEMPO** und **VOL** bestimmen die Geschwindigkeit und Lautstärke, mit der die Noten gespielt werden. Weiterhin stellt Ihnen die Anweisung **FILTER** drei verschiedene Filtertypen zur Verfügung, mit der die einzelnen Töne noch weiter beeinflußt werden können. Im Anhang sind die Befehle und Anweisungen zur Musikprogrammierung noch einmal mit den dazugehörigen Parametern aufgeführt. Am besten lernen Sie den Umgang mit diesen Befehlen, wenn Sie sie öfters anwenden und variieren.

Das soll uns an Informationen zum Kapitel Musik genügen. Weiterführende Informationen über die vielfältigen Möglichkeiten der Musikprogrammierung des Commodore 128 finden Sie in Ihrem Handbuch oder in Büchern über den Commodore 128, die sich speziell mit diesem Thema auseinandersetzen.

6.2 Grafik

Dieses Kapitel soll Sie nicht in grundlegende Techniken der Grafikprogrammierung auf dem Commodore 128 einführen. Es soll Ihnen jedoch einen Überblick über die wichtigsten Grafikbefehle geben, so daß Sie erkennen können, welche Möglichkeiten der Commodore 128 besitzt, um Grafiken zu erstellen.

Der Commodore 128 wurde mit sehr leistungsfähigen Grafikbefehlen ausgestattet, die es dem Anwender erlauben, auf einfache Art und Weise Grafiken zu erzeugen. Sämtliche zur Verfügung stehenden Grafikbefehle beziehen sich immer auf den 40-Zeichenbildschirm. Damit steht Ihnen, wie beim Commodore 64, eine Auflösung von 320 mal 200 Punkten zur Verfügung.

Sie können zwar über die Anweisung **GRAPHIC 5** auf den 80-Zeichenbildschirm umschalten, jedoch beziehen sich die

Anweisungen **BOX** oder **CIRCLE** nach wie vor grundsätzlich nur auf den 40-Zeichenbildschirm.

Bisher konnten eigentlich nur Insider oder besser INTERNisten, die jede Speicherstelle und jedes Bit mit Vornamen kannten, die Grafikmöglichkeiten des Commodore 64 voll ausnutzen. Es blieb daher dem einfachen Anwender, versagt mit der Grafik seine Erfahrungen zu machen. In Zukunft können Sie nun Befehlszeilen wie

POKE 53270,(PEEK 53270) OR 16

vergessen, um den Multicolormodus einzuschalten. Sie geben dazu einfach **GRAPHIC 3** ein und schon ist der Multicolormodus eingeschaltet.

Mit der Anweisung

GRAPHIC

können wir also eine der sechs Grafikbetriebsarten anwählen. Die allgemeine Schreibweise der Anweisung lautet:

GRAPHIC m (,c,s)

Die Parameter in Klammern können mit angegeben werden, müssen es aber nicht. Der zweite Parameter **'c'** bestimmt, ob beim Umschalten automatisch ein **SCNCLR** *(Löschen des Bildschirms)* ausgeführt wird oder nicht. Hat **c** den Wert **1**, so wird ein **SCNCLR** ausgeführt. Für **c=0** erfolgt **kein SCNCLR.**

Der Parameter **'s'** bestimmt für den kombinierten Modus von Text und Grafik *(GRAPHIC 2 oder GRAPHIC 4)*, in welcher Zeile der Textbildschirm beginnen soll. Der Vorgabewert liegt bei Zeile 19. Der Parameter **m** wählt eine der sechs Grafikarten an.

0 40-Zeichen Textmodus
1 Grafikmodus
2 kombinierter Grafik- Textmodus
3 Multicolor Grafikmodus
4 kombinierter Multicolor Grafik- Textmodus
5 80-Zeichen Textmodus

Eine **Besonderheit** sei hier noch erwähnt. Sie können die Grafikbefehle **erst dann anwenden**, wenn Sie bereits einmal mit **GRAPHIC** eine Grafikseite angewählt haben. Geben Sie nämlich direkt nach dem Einschalten des Rechners einen Grafikbefehl ein *(BOX oder auch SCNCLR 1!)*, erhalten Sie sofort die Fehlermeldung

?NO GRAPHICS AREA ERROR

Das liegt daran, daß der Commodore 128 erst einmal Speicherplatz für die Grafik schaffen muß. Ist dieser Speicherplatz noch nicht reserviert, und Sie versuchen einen wie auch immer gestalteten Grafikbefehl anzuwenden, erhalten Sie grundsätzlich diese Fehlermeldung. Sie müssen also zuerst mit *GRAPHIC* eine Grafikseite anwählen, bevor Sie die Grafikbefehle anwenden können.

Denken Sie also in Ihren Grafikprogrammen daran:

Nicht so:

10 SCNCLR 1:GRAPHIC 1

sondern so:

10 GRAPHIC 1:SCNCLR 1

Wir wollen nun anhand eines Beispiels einige Grafikbefehle kennenlernen. Unser Ziel soll eine **Analoguhr** sein, die, gesteuert durch die Variable **TI$**, uns Stunden und Minuten anzeigt. Dazu besprechen wir zunächst die Anweisung

DRAW fs,x1,y1 TO x2,y2 TO x3,y3...

Mit **DRAW** können beliebige Figuren durch Angabe von Start- und Endpunkten gezeichnet werden. Dabei kann es sich um Punkte, Linien oder beliebige Umrisse handeln. Mit fs wird der entsprechende Farbspeicher angewählt, und die Parameter x,y bestimmen jeweils Start- und Endpunkt. Es können für den Paramter fs die Werte 0-1 im hochauflösenden Grafikmodus verwendet werden und die Werte 0-3 im Multicolor Grafikmodus.

Im hochauflösenden Modus besitzt die Grafikseite die horizontalen Koordinaten von 0 bis 319 und die vertikalen Koordinaten von 0 bis 199, so daß das folgende Programm einen Punkt genau in die Mitte des Bildschirms plaziert

```
10 GRAPHIC 1:SCNCLR
20 DRAW ,160,100
30 GETKEY A$
40 GRAPHIC 0
50 END
```

Eine der wohl vielseitigsten Anweisungen ist die CIRCLE-Anweisung. Mit dieser Anweisung kann man nicht nur einfache Kreise zeichnen, sondern auch Ellipsen und Vielecke jeder Art. Hier ist wirklich Ihre Phantasie gefordert.

Die CIRCLE-Anweisung hat die folgende allgemeine Schreibweise:

CIRCLE fs,x,y,xr,yr,sw,ew,w,i

Die Parameter haben dabei die folgende Bedeutung:

fs	Farbspeicher
x,y	Koordinaten des Kreismittelpunkts
xr	Radius in X-Richtung
yr	Radius in Y-Richtung (Vorgabewert ist xr)
sw	Anfangswinkel für den Kreis
ew	Endwinkel für den Kreis
w	Winkel für Rotation
i	Winkel für die zu zeichnenden Kreissegmente (Vorgabewert = 2)

Über die CIRCLE-Anweisung werden bei der Analoguhr die beiden Zeiger generiert. Im folgenden nun das Programm für die Analoguhr.

6.2.1 Analoguhr

```
10 REM ANALOGUHR
20 GRAPHIC 1:SCNCLR 1
30 DRAW ,162,100
40 FOR I=1 TO 12
50 READ X,Y
60 CHAR ,X,Y,STR$(I)
70 NEXT I
80 FOR I=100 TO 96 STEP -1
90 CIRCLE ,162,100,I,,,,,1
100 NEXT I
110 DATA 25,3,29,7,30,12,29,17,26,21,19,23,12,21,9,17,8,12,9,7,12,3,18,1
200 REM
210 REM ZEIGER
220 REM
230 DO
240 :  CHAR ,1,1,TI$
250 :  H=VAL(LEFT$(TI$,2))
260 :  M=VAL(MID$(TI$,3,2))
270 :  X= 20*SIN(M*6*PI/180)
280 :  Y=-20*COS(M*6*PI/180)
290 :  Y1=-10*COS((H*30+M/2)*PI/180)
```

```
300 :  X1= 10*SIN((H*30+M/2)*PI/180)
310 LOOP WHILE MA=M
320 CIRCLE 0,162+XA,100+YA,8,55,,,MA*6,120
330 CIRCLE 0,162+SX,100+SY,8,45,,,HA*30+MA/2,120
340 CIRCLE 1,162+X1,100+Y1,8,45,,,H*30+M/2,120
350 CIRCLE 1,162+X,100+Y,8,55,,,M*6,120
360 XA=X:YA=Y:MA=M:HA=H
370 SX=X1:SY=Y1
380 GOTO 230
400 END
```

Das wäre soweit das Programmlisting zur Analoguhr. Denken Sie beim Eingeben des Programms daran, daß *PI* der *PI-Taste* auf Ihrer Tastatur entspricht.

Besprechen wir nun noch kurz das Programm:
Zeile 20 wählt den entsprechenden Grafikmodus an und löscht den Bildschirm. Zeile 30 zeichnet einen Punkt, der als Achse für die Zeiger gedacht ist. In den Zeilen 40 bis 70 werden die Positionen der Ziffern für das Ziffernblatt aus der DATA-Zeile ausgelesen und mit CHAR ausgegeben. Die Zeilen 80 bis 100 zeichnen mit der CIRCLE-Anweisung die äußere Umrandung des Ziffernblatts. In der Zeile 240 wird laufend die Zeit in digitaler Form auf dem Bildschirm angezeigt. Weiterhin werden in dieser Schleife die Werte für Stunden und Minuten sowie die Werte für die Positionen der Zeiger berechnet. Nach jeweils einer Minute wird die Schleife verlassen *(MA ist dann nicht mehr gleich M)*, und die alten Zeiger werden in den Zeilen 320 und 330 gelöscht. Die Zeilen 340 und 350 zeichnen dann die neuen Zeiger. Die Werte der momentanen Zeiger müssen zwischengespeichert werden, um später die Zeiger löschen zu können. Weiterhin muß MA wieder mit M gleichgesetzt werden. Dies wird durch die Zeilen 360 bis 370 erreicht. Schließlich springt das Programm in Zeile 380 zurück in die Warteschleife ab Zeile 230.

Wollen Sie das Programm mit der aktuellen Uhrzeit laufen lassen, so müssen Sie die Variable TI$ vorher neu definieren. Sie können das Programm auch dahingehend erweitern, daß zu Beginn des Programms nach der aktuellen Uhrzeit gefragt wird.

Das waren soweit die Erläuterungen zu diesem Programm. Damit wäre unser kleiner Einstieg in die Grafikprogrammierung abgeschlossen. Sämtliche Grafikbefehle finden Sie in der Befehlsübersicht im Anhang noch einmal komplett mit Beispielen aufgeführt.

7

BASIC Intern

7. BASIC Intern

Dieses Kapitel soll keinesfalls das Buch *"128 Intern"* ersetzen. Sie sollen jedoch einen kleinen Einstieg vermittelt bekommen, damit Ihnen der Übergang zu diesem Buch erleichtert wird.

7.1 Der Monitor

Der Commodore 128 besitzt einen eingebauten *Monitor*. Damit ist nun nicht das gleichnamige Datensichtgerät gemeint, sondern eine Einrichtung, mit der Sie sich den Speicherinhalt des Commodore 128 ansehen bzw. ausgeben lassen können. So ist es z.B. möglich, sich das Betriebssystem des Commodore 128 in **disassemblierter** Form anzeigen zu lassen. Diese disassemblierte Form wird auch **ROM-Listing** geannt.

Im Betriebssystem sind verschiedene Routinen in Maschinensprache untergebracht, die z.B. dafür verantwortlich sind, daß Ihre Programme sicher auf der Diskette abgespeichert werden. Ein Teil des ROM-Listings sieht z.B. so aus:

```
4BCC:  Ø1 18     ORA ($18,X)
4BCE:  DØ 26     BNE $4BF6
4BDØ:  24 7F     BIT $7F
4BD2:  1Ø ØD     BPL $4BE1
4BD4:  2Ø 34 4B  JSR $4B34
4BD7:  A5 3B     LDA $3B
4BD9:  A4 3C     LDY $3C
4BDB:  8D ØØ 12  STA $12ØØ
4BDE:  8C Ø1 12  STY $12Ø1
```

Die hexadezimalen Zahlen in der **linken Spalte bezeichnen** die **Speicherstellen**, an denen diese Informationen stehen. Die **mittlere Spalte** beinhaltet den **reinen Maschinencode** und die **rechte Spalte** den **Assemblercode**. Dieser Teil des ROM-Listings repräsentiert übrigens den ersten Teil des BASIC-Befehls END.

Trifft der Interpreter in Ihrem Programm also auf ein END, so wird diese Routine im Betriebssystem ausgeführt. Besprechen wir zunächst die wichtigsten Befehle des Monitors.

Der Befehl

MONITOR

schaltet den Monitor ein. Nach der Eingabe von **MONITOR** sehen Sie auf dem Rechner die folgende Ausgabe:

```
MONITOR
    PC   SR AC XR YR SP
; FBØØØ  ØØ ØØ ØØ ØØ F8
```

Ohne nun tiefer in die Maschinensprache eindringen zu wollen, sei hier nur gesagt, daß dort die verschiedenen Prozessorregister mit Inhalt angezeigt werden. Der blinkende Cursor erwartet nun Ihre Eingabe. Geben Sie jetzt

D F4BCC

ein, so erhalten Sie den Teil des o.a. ROM-Listings. Der Buchstabe **D** steht demnach für **disassemblieren**. Alle möglichen Befehle des Monitors wollen wir jetzt nicht besprechen. Es soll sich auf die beschränkt werden, die wir in diesem Kapitel unbedingt benötigen. Der Befehl **M** gibt uns den Speicher in hexadezimaler Form aus, wobei in der rechten Spalte druckbare ASCII-Zeichen mit angezeigt werden. Geben Sie jetzt

M F4BCC

ein, werden Sie den Unterschied zum **D-Befehl** erkennen.

Wir wollen nun ein kleines BASIC-Programm schreiben und uns dieses dann mit dem Monitor anschauen. Zunächst **verlassen** wird den **Monitor** durch die **Eingabe von X.**

Geben Sie jetzt NEW ein und dann nachfolgend das kleine Programm:

```
10 PRINT "DIESEN TEXT KANN MAN IM MONITOR LESEN."
20 END
```

Geben Sie jetzt wieder den Befehl **MONITOR** ein. Um nun zu sehen, was der Computer aus unserem Programm gemacht hat, geben Sie entweder

M 01C00

oder

M 04000

ein. Die erste Adresse kennzeichnet den BASIC-Start, falls noch keine Grafikseite angewählt wurde. Haben Sie bereits die Grafik benutzt, werden Sie Ihr Programm an der zweiten Adresse vorfinden. Wie bereits erwähnt, muß der Commodore 128 Speicherplatz für die Grafik reservieren. Dies erreicht er dadurch, daß er den BASIC-Start von $1C00 nach $4000 verschiebt *(das sind genau 9 KByte)*.

Wir gehen im weiteren Verlauf davon aus, daß das Programm ab Adresse $4000 vorliegt. Nach der Eingabe von **M 04000** werden die Speicherstellen von Adresse $4000 bis $405F angezeigt. Interessant sind für uns die Speicherstellen von $4000 bis $403F. Was Sie dort sehen, ist das BASIC-Programm, wie es der Commodore 128 in seinem Speicher ablegt.

Das einzige, was man sofort erkennen kann, ist unser Text, der in den Anführungszeichen steht. Wir können aber weder den Befehl **PRINT** noch den Befehl **END** direkt ablesen. Im ersten Kapitel des Buches wurde erwähnt, daß die BASIC-Befehle erst über den Interpreter geleitet werden und dort in einen computereigenen Code übersetzt werden. Diese Codes der BASIC-Befehle nennen sich **Token**. Das **Token** von **PRINT** ist z.B. der **Wert 153** bzw. **hexadezimal 99**. Eine komplette Tabelle aller Tokenwerte finden Sie im Anhang des Buches. Diesen hexa-

dezimalen Wert 99 erkennen wir in der ersten Zeile bei Adresse $0405. Dort steht also unser PRINT-Befehl, wie ihn der Computer interpretiert. Zwei Stellen vorher sehen wir die **Zeilennummer** als **0A** oder **dezimal 10**. Davor weist, aufgeschlüsselt in Low- und High-Byte *(30 40)*, der Zeiger auf die nächste Programmzeile. Bewegen Sie jetzt den Cursor auf das erste Zeichen der Speicherstelle $0405. Ändern Sie den Wert von $99 in $8F und betätigen Sie die RETURN-Taste. Verlassen Sie danach den Monitor mit X und schauen Sie sich Ihr Programm an.

Aus dem **PRINT** wurde ein **REM**! Sie haben das Token $99 für PRINT durch das Token $8F für REM ersetzt. Diesen Vorgang können Sie auch per Programm mit dem Befehl POKE ausführen lassen, so daß Sie damit Programme schreiben können, die sich innerhalb des Programmablaufs selbst verändern! Damit ergeben sich ungeahnte Möglichkeiten. Sie können ja einmal experimentieren...

7.2 Der Variablenzeiger

Eine weitere nützliche Funktion bietet die

POINTER *(Variable)*

Funktion. Mit ihr können Sie feststellen, an welcher Stelle im Speicher der Zeiger in der Bank 1 für die übergebene Variable zu finden ist *(pointer (engl.) = Zeiger)*. Geben Sie NEW ein und definieren Sie anschließend die Variable **A$="TEST"**. Geben Sie danach

PRINT POINTER(A$)

ein. Sie erhalten den Wert **1026**, was hexadezimal **$0402** entspricht. Wählen Sie jetzt wieder den Monitor an und geben Sie folgenden Befehl ein:

M 10400

Dort sehen Sie in der ersten Zeile in Adresse $0402 den Wert 4. Das ist die Länge der Stringvariablen A$ *(TEST=4 Zeichen)*. Die folgenden zwei Bytes *(Low, High)* weisen auf die Speicherstelle hin, in der der Inhalt der Variablen A$ zu finden ist, also in unserem Fall **FEFA**. Geben Sie jetzt den Befehl

M 1FEFA

ein, so erkennen Sie in den ersten vier Bytes den **Inhalt** der **Variablen A$** *(TEST)*.

Ich hoffe, Ihnen mit dieser kurzen Einführung zumindest einen Einblick in die internen Zusammenhänge der BASIC-Programmierung gegeben zu haben.

8

UTILITIES

8. Utilities

In diesem Kapitel finden Sie einige nützliche Routinen, die Sie in Ihren Programmen mit verwenden können.

8.1 Hardcopy Text

Das erste Programm erstellt eine Hardcopy des Textbildschirms auf dem Drucker. Diese Routine können Sie als Unterprogramm in Ihren Programmen verwenden und so z.B. Bildschirmmasken einer selbstgeschriebenen Dateiverwaltung oder Textverarbeitung ausgeben lassen. Die inversen Zeichen werden auf dem Drucker als normale Zeichen dargestellt.

```
10 REM HARDCOPY TEXT
20 OPEN 1,4:CMD1
30 FOR I=0 TO 999
40 A=PEEK(1024+I):Z=Z+1
45 IF Z=41 THEN Z=1 : PRINT#1,CHR$(13);
50 IF A > 127 THEN A=A-128
60 IF A < 32 THEN A=A OR 64 : GOTO 100
70 IF A > 31 AND A < 64 THEN 100
80 IF A > 63 AND A < 96 THEN A=A OR 32: GOTO 100
90 IF A > 95 AND A < 128 THEN A=A AND 31 OR 160
100 PRINT#1,CHR$(A);
110 NEXT
120 PRINT#1:CLOSE1
130 END
```

Wollen Sie dieses Programm als Unterprogramm verwenden, so muß in der Zeile 130 das **END** gegen ein **RETURN** ausgetauscht werden.

8.2 Binärumwandlung

Leider besitzt der Commodore 128 keine Funktion, um Zahlen in das binäre oder duale Zahlensystem umzuwandeln. Der nachfolgenden Routine wird die umzurechnende Zahl in der **Variablen A** übergeben. Die **Routine übergibt** die **Dualzahl** dann **in** der **Stringvariablen BI$**.

```
100 REM UNTERPROGRAMM BINAERUMWANDLUNG
110 BI$=""
120 FOR D=7 TO 0 STEP -1
130 BI(D)=INT(A/2↑D)
140 A=A-BI(D)*2↑D
150 BI$=BI$+RIGHT$(STR$(BI(D)),1)
160 NEXT
170 RETURN
```

8.3 Ausgabe mit führenden Nullen

Vielleicht stehen Sie auch einmal vor dem Problem, Ihr Programm, welches in BASIC 7.0 geschrieben ist, einem anderen Rechner anzupassen, der nicht über die PRINT USING-Anweisung verfügt. Die nachfolgende Routine zeigt Ihnen eine Lösung, wie Variablen formatiert mit führenden Nullen ausgegeben werden können.

```
100 PU$=RIGHT$("0000"+RIGHT$(STR$(I),LEN(STR$(I))-1),4)
```

Das Format wird durch die vier Nullen bestimmt. Benötigen Sie ein größeres Format, so brauchen Sie nur die Anzahl der Nullen entsprechend zu vergrößern. In der Variablen I wird der zu formatierende Wert abgelegt.

8.4 Oszillierendes Flag

Oft ist es wünschenswert, daß ein Flag zwischen zwei Werten hin- und herspringt. Also hat das Flag einmal den Wert Null und einmal den Wert 1. Genauso oft sieht man in Programmen dann Lösungen mit IF...THEN und GOTO. Dabei gibt es eine einfache Lösung für dieses Problem.

```
100 Z=1-Z
```

Bei jedem Durchlauf dieser Programmzeile wechselt Z entweder von 0 nach 1 oder von 1 nach 0.

8.5 Programmlisting auf Diskette

Mit den folgenden Anweisungen können Sie ein Programmlisting auf der Diskette abspeichern.

```
DOPEN#1,"Dateiname",W:CMD1:LIST:DCLOSE#1
```

8.6 Auslesen einer sequentiellen Datei

Das nachfolgende Programm liest eine sequentielle Datei von der Diskette und gibt sie auf dem Bildschirm aus.

```
10 DOPEN#1,"DATEINAME"
20 DO WHILE ST <> 64
30 :  GET#1,A$:IF A$="" THEN 30
40 :  PRINT A$;
50 LOOP
60 DCLOSE#1
70 END
```

9

LÖSUNGEN

9. Lösungen

Lösungen zu Seite 47

Aufgabe 1

a)	6C	b)	92
c)	BA	d)	F0
e)	C	f)	C9

Aufgabe 2

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

Aufgabe 3

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

Aufgabe 4

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

Lösungen zu Seite 74

Aufgabe 1

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

Aufgabe 2

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

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

Aufgabe 3

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

Aufgabe 4

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

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

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

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

Aufgabe 5

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

In diesem Programm werden zunächst den Variablen für die Höhe, Länge und Tiefe, H, L und T, die Werte in cm übergeben. Sie sehen, daß man den Variablen durchaus sinnvolle Bezeichnungen zukommen lassen kann. Dann wird in Zeile 30 das Volumen in ccm errechnet. In Zeile 50 wird das ausgerechnete Volumen noch durch 1000 dividiert und schon haben wir unseren Inhalt des Aquariums in Litern.

Aufgabe 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.

Lösungen zu Seite 89

Aufgabe 1

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

So ähnlich sollte Ihr Programm aussehen. Die Lösungen, die hier angeboten werden, sind natürlich nur als Lösungsvorschläge anzusehen. Wir haben ja bereits festgestellt, daß mehrere Wege nach Rom führen. Wenn Sie das Programm wiederholt mit RUN / RETURN starten, sehen Sie die Unterschiede in den ausgegebenen Zahlen. In Zeile 20 wird der Bildschirm mit dem CHR$-Code gelöscht. Sollten Sie hier bereits die SCNCLR Anweisung benutzt haben, so ist das selbstverständlich auch korrekt. Es sollten in diesem Beispiel allerdings auch der Umgang mit den CHR$-Codes geübt werden. 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.

Aufgabe 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.

Aufgabe 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, daß Komma oder nur Return einzugeben, um den ASCII-Wert dieser "Zeichen" zu erfahren. Dabei wurde dann vom Rechner die Fehlermeldung ILLEGAL QUANTITY ERROR IN 20 ausgegeben. Das ist einer der Nachteile des INPUT-Befehls, da z.B. das Komma zur Trennung von Variablen benutzt wird. Betätigen Sie nun nur die RETURN-Taste, so wird der Stringvariablen '*Nichts*' zugeordnet, d.h. diese Variable ist leer. Da der Rechner natürlich von '*Nichts*' den ASCII-Wert unmöglich ermitteln kann, wird die o.a.

Fehler-meldung ausgegeben. Wie man dieses Problem bei der Eingabe umgeht, wird in einem späteren Kapitel erklärt (siehe GET-Befehl).

Aufgabe 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.

Aufgabe 5

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

Dieses Programm braucht wohl nicht näher erläutert zu werden. In seiner Einfachheit erklärt es sich fast von selbst.
Haben Sie diese Aufgaben selbständig zu Ihrer eigenen Zufriedenheit gelöst, so können Sie jetzt getrost zum nächsten Kapitel übergehen. Bei Unsicherheiten schlagen Sie noch einmal in den entsprechenden Passagen nach.

Lösungen zu Seite 103

Aufgabe 1

a) ist richtig.

Aufgabe 2

Man erhält den Ausdruck BEIN.

Aufgabe 3

Man erhält wieder den Ausdruck ROTOR.

Aufgabe 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.

Lösungen zu Seite 123

Aufgabe 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.

Aufgabe 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
```

Aufgabe 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.

Aufgabe 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.

Aufgabe 5

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

Diese Aufgabe war, zugegeben, schon recht schwierig. Ihr Programm muß dem obigen nun nicht aufs Haar gleichen. Jedoch sollte es die Erzeugung des Vergleichsstrings in Zeile 50 in etwa beinhalten, da das ja die eigentliche Schwierigkeit ist. Die Aufgabenstellung bezog sich auf einen beliebigen String, d.h. unabhängig davon, nach welchen und nach wieviel Zeichen der ursprüngliche String durchsucht werden sollte. So mußte der MID$-Funktion die Länge des zu suchenden Strings über die LEN-Funktion mitgeteilt werden. Der Zähler in Zeile 40 sorgt dafür, daß die Position von C$ 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.

Lösungen zu Seite 158

Aufgabe 1

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

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

Aufgabe 2

```
10 REM QUADRATISCHE GLEICHUNG
20 SCNCLR
30 PRINT"EINGABE DER KOEFFIZIENTEN A,B,C"
40 PRINT
50 INPUT A,B,C
60 IF A=0 THEN 20:REM A MUSS <> 0 SEIN
70 D=B*B-4*A*C
80 IF D < 0 THEN 140
90 X1 =(-B+SQR(D))/(2*A)
100 X2 =(-B-SQR(D))/(2*A)
110 PRINT"LOESUNG FUER X1 =";X1
120 PRINT"LOESUNG FUER X2 =";X2
130 GOTO 150
140 PRINT"KEINE REELLEN NULLSTELLEN !!"
150 END
```

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

Aufgabe 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.

Lösungen zu Seite 205

Aufgabe 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 ver-

zweigt 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.

Aufgabe 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.

Aufgabe 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.

ANHANG

Anhang A: Befehlsübersicht

BASIC 2.0

ABS

Kategorie: Numerische Funktionen

Schreibweise: ABS(X)

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

AND

Kategorie: logischer Operator

Schreibweise: <Ausdruck> AND <Ausdruck>

Funktion: "logisches Und" wird in der Boole'schen Algebra zur Wahrheitsprüfung zweier Ausdrücke benutzt. Das Ergebnis ist nur dann wahr, wenn beide Ausdrücke wahr sind.

ASC

Kategorie: String Funktionen

Schreibweise: ASC("X")

Funktion: Ergibt den ASCII Wert von X. Handelt es sich um einen String, der sich aus mehr als einem Zeichen zusammensetzt, so wird nur der ASCII-Wert des ersten Zeichens ausgegeben. Beinhaltet der String "nichts", so wird die Fehlermeldung ?ILLEGAL QUANTITY ERROR ausgegeben.

ATN

Kategorie: Numerische Funktionen

Schreibweise: ATN(X)

Funktion: Als Ergebnis dieser mathematischen Funktion erhält man den Arcustangens der Zahl X. Das Ergebnis liegt immer in dem Intervall -Pi/2 bis +Pi/2.

CHR$

Kategorie: String Funktionen

Schreibweise: CHR$(X)

Funktion: Beim Aufruf dieser Funktion wird eine Zahl X in das entsprechende Zeichen des ASCII-Codes umgewandelt. Die Zahl muß zwischen 0 und 255 liegen.

CLOSE

Kategorie: Befehle

Schreibweise: CLOSE X

Funktion: Schließen des Files X, wobei X die logische Filenummer darstellt. Es handelt sich hierbei um die gleiche Filenummer, die vorher bei der OPEN-Anweisung zum Öffnen des Files benutzt wurde (siehe OPEN).

CLR

Kategorie: Befehle

Schreibweise: CLR

Funktion: Dieser Befehl löscht alle Variablen, Felder sowie vom Benutzer definierte Funktionen. Er beeinträchtigt, im Gegensatz zu NEW, das eigentliche BASIC-Programm nicht.

CMD

Kategorie: Befehle (Ausgabe)

Schreibweise: CMD X

Funktion: Leitet Daten, die eigentlich auf dem Bildschirm ausgegeben werden, auf ein entsprechendes Peripheriegerät um, z.B. auf Drucker, Diskettenlaufwerk oder Datasette. Hierbei ist X wiederum das File, das zuvor in der OPEN Anweisung benutzt wurde, um das Peripheriegerät anzusprechen. Bei Benutzung der Befehle PRINT und LIST werden die Daten dann auf das logische File übertragen.

CONT

Kategorie: Befehle

Schreibweise: CONT

Funktion: Mit CONT können Programme fortgesetzt werden, die durch die STOP-Taste, den Befehl STOP oder den Befehl END unterbrochen wurden. Das Programm nimmt genau an der Stelle seine Funktion wieder auf, an der es unterbrochen wurde. Dieser Befehl ist nicht anwendbar, wenn am Programm Änderungen vorgenommen wurden oder das Programm durch eine Fehlermeldung abgebrochen wurde.

COS

Kategorie: Numerische Funktionen

Schreibweise: COS(X)

Funktion: Dieser Befehl ergibt den Cosinus einer Zahl X. X ist hier das Bogenmaß eines Winkels.

DATA

Kategorie: Anweisung

Schreibweise: DATA X,Y,Z

Funktion: Mit dieser Anweisung kann man bei der Programmierung Informationen ablegen, die mit dem READ-Befehl ausgelesen werden können. Die Informationen können sowohl Zahlen als auch Zeichen (Strings) sein. Bestimmte Zeichen müssen in Hochkommata stehen, wenn sie Komma, Leerzeichen oder den Doppelpunkt enthalten. Die Daten werden der Reihe nach gelesen, d.h. von links nach rechts.

DEF FN

Kategorie: Anweisung

Schreibweise: DEF FN F(X)=X*Y

Funktion: Mit dieser Anweisung kann vom Programmierer eine mathematische Funktion definiert werden, die später im Programm nur noch mit dem Funktionsnamen aufgerufen werden kann. Hierbei ist "F" der Funktionsname, "(X)" die Variable und "X*Y" die eigentliche Funktion. Die Funktion selber kann aus einer beliebigen mathematischen Anweisung bestehen.

DIM

Kategorie: Anweisung

Schreibweise: DIM A(X)

Funktion: Mit dieser Anweisung werden ein Feld (Array) oder eine Matrix (mehrdimensionales Feld) dimensioniert. (X) wird hierbei als Index bezeichnet; man spricht auch von indizierten Variablen, in diesem Falle A(X). A(X) nennt man eindimensionales Feld und A(X,Y) mehrdimensionales Feld.

END

Kategorie: Befehle

Schreibweise: END

Funktion: Trifft das Programm bei der Ausführung auf diesen Befehl, so wird der Programmablauf beendet und die Meldung "READY." wird ausgegeben. Diese Anweisung ist ähnlich wie die STOP-Anweisung. Nach beiden Befehlen kann das Programm mit CONT wieder fortgesetzt werden.

EXP

Kategorie: Numerische Funktionen

Schreibweise: EXP(X)

Funktion: Hiermit läßt sich die X-te Potenz der Konstanten e (2.718281823) berechnen, wobei X nicht größer sein darf als 88.0296919.

FN

Kategorie: Numerische Funktionen

Schreibweise: FN A(X)

Funktion: Ergibt den Funktionswert einer Zahl X, die zuvor in der Funktion mit DEF FN A(X) festgelegt wurde.

FOR...TO...(STEP)

Kategorie: Befehle

Schreibweise: FOR X=1 TO 10 STEP 2

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, eine Variable (hier X) als Zähler zu benutzen. Sie müssen den Startwert der Variablen angeben (hier 1), den Endwert (hier 10) und die Schrittweite (hier 2). Geben Sie keine Schrittweite an, so wird automatisch die Schrittweite 1 angenommen. Es kann jede beliebige Gleitpunktzahl benutzt werden.

FRE

Kategorie: Numerische Funktionen

Schreibweise: FRE(X)

Funktion: Mit dieser Funktion erhalten Sie die Anzahl freier Bytes, die Sie noch für Ihr Programm zur Verfügung haben. X kann jeden beliebigen Wert annehmen, da es nicht zur Berechnung herangezogen wird.

GET

Kategorie: Befehle

Schreibweise: GET X$

Funktion: Mit diesem Befehl können Sie jedes Zeichen einzeln von der Tastatur aus einlesen. Das eingelesene Zeichen wird der Variablen, die GET folgt, hier X$, zugeordnet. Wird eine numerische Variable verwendet, z.B. X, und es wird ein anderes Zeichen als eine Zahl eingegeben, so erscheint die Fehlermeldung "SYNTAX ERROR". Daher sollten aus Gründen der Programmsicherheit die Zeichen als Stringvariablen eingelesen und nachher im Programm als Zahlenwerte aufbereitet werden.

GET#

Kategorie: Befehle (Eingabe)

Schreibweise: GET#1,X$

Funktion: Mit diesem Befehl werden einzelne Zeichen von einem Peripheriegerät eingelesen, welches vorher durch einen OPEN-Befehl angesprochen wurde. Hier ist 1 wieder die logische Filenummer und X$ die Variable, der das eingelesene Zeichen zugeordnet wird. Dieser Befehl entspricht im Prinzip dem GET-Befehl.

GOSUB

Kategorie: Befehle

Schreibweise: GOSUB XX

Funktion: Mit diesem Befehl wird innerhalb eines Programms zu einem Unterprogramm verzweigt, wobei XX hier die Zeilennummer angibt, an der das Unterprogramm beginnt. Trifft das Programm im Unterprogramm auf die RETURN-Anweisung, so

springt das Programm zu der Anweisung zurück, die unmittelbar hinter dem GOSUB-Befehl steht. Achten Sie darauf, daß ein Unterprogramm nie mit einem GOTO verlassen wird, da der Rechner sonst mit der internen Verwaltung der Unterprogramme durcheinanderkommt. Es wird in einem späteren Kapitel anhand eines Beispiels gezeigt, was geschieht, wenn man sich nicht an diese Regel hält.

GOTO

Kategorie: Befehle

Schreibweise: GOTO XX

Funktion: Mit diesem Befehl springt das Programm an eine durch XX angegebene Zeilennummer des Programms. Dadurch kann der eigentliche Programmablauf, der durch die Zeilennummern bestimmt wird, beeinflußt werden.

IF...THEN

Kategorie: Befehle

Schreibweise: IF (Ausdruck) THEN (Ausdruck)

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, innerhalb des Programms eine Bedingung überprüfen zu lassen. Falls diese Bedingung wahr ist, wird die entsprechende Anweisung hinter THEN ausgeführt. Wird die Bedingung hinter IF als falsch gewertet, so wird alles, was dem THEN folgt, ignoriert, und das Programm fährt in der nächsten Programmzeile fort. Der Ausdruck hinter IF enthält in den meisten Fällen einen Vergleich, z.B. IF A=10 THEN, wobei der Ausdruck hinter THEN jeden beliebigen BASIC-Befehl enthalten kann.

INPUT

Kategorie: Befehle

Schreibweise: INPUT X oder INPUT"Kommentar";X

Funktion: Mit diesem Befehl steht dem Anwender die Möglichkeit zur Verfügung, dem Programm Daten zu übergeben, die dann durch das Programm entsprechend verarbeitet werden können. Trifft das Programm auf diesen Befehl, so erscheinen auf dem Bildschirm ein Fragezeichen und der Cursor. Jetzt kann der Anwender Daten eingeben und durch Drücken der RETURN-Taste die Eingabe abschließen. Im Programm kann dem INPUT direkt die Variable folgen oder noch zusätzlich durch einen Kommentar ergänzt werden, wobei zwischen Kommentar und der Variablen stets ein Semikolon stehen muß. Der INPUT-Befehl kann nur innerhalb eines Programms benutzt werden.

INPUT#

Kategorie: Befehle (Eingabe)

Schreibweise: INPUT#1,X$

Funktion: INPUT# hat eine ähnliche Funktion wie GET#, nur mit dem Unterschied, daß hier nicht ein Zeichen nach dem anderen zugeordnet wird, sondern bis zu maximal 80 Zeichen auf einmal. 1 bezeichnet hier wieder die 'logische Filenummer' des zuvor mit OPEN 1 angesprochenen Peripheriegerätes. X$ ist die Variable, der die eingelesenen Daten zugeordnet werden. Trifft INPUT# beim Lesen der Daten auf ein Semikolon, Komma, Doppelpunkt oder auf ein Return (CHR$(13)), so werden diese als das Ende der Variablen interpretiert.

INT

Kategorie: Numerische Funktionen

Schreibweise: INT(X)

Funktion: Ergibt den ganzzahligen Wert der Variablen X, d.h. es werden die Nachkommastellen einfache abgeschnitten, unabhängig davon, wie groß der Wert hinter der Kommastelle ist. Es findet keine Rundung im üblichen Sinne statt. Die erhaltenen Werte sind immer kleiner oder gleich X. Bei negativen Zahlen werden die Werte für X also dem Betrag nach größer; z.B. INT(-1.23) ergibt -2.

LEFT$

Kategorie: String Funktionen

Schreibweise: LEFT$(X$,A)

Funktion: Gibt die linken Zeichen von X$ wieder, angefangen beim ersten linken Zeichen von X$. A bestimmt die Anzahl der Zeichen, die aus X$ gelesen werden. LEFT$(X$,4) liest also die ersten vier linken Zeichen von X$. A darf Werte zwischen 0 und 255 annehmen, sonst wird die Fehlermeldung ILLEGAL QUANTITY ERROR ausgegeben.

LEN

Kategorie: Numerische Funktionen

Schreibweise: LEN(X$)

Funktion: Mit dieser Funktion erhält man die Anzahl der Zeichen von X$. Es werden alle Zeichen im String berücksichtigt, also auch Leerzeichen.

LET

Kategorie: Befehle

Schreibweise: LET X=5

Funktion: Der LET-Befehl dient dazu, einer Variablen einen Wert oder String zuzuweisen. Hier würde also der Variablen X der Wert 5 zugeordnet. LET wird jedoch beim Programmieren kaum verwendet, da die Zuweisung X=5 bereits genügt.

LIST

Kategorie: Befehle

Schreibweise: LIST (Zeilennr.) - (Zeilennr.)

Funktion: Der LIST-Befehl ermöglicht es Ihnen, sich das momentane BASIC-Programm zeilenweise oder ganz zeigen zu lassen. Wird nur LIST eingegeben, so wird das komplette Programm am Bildschirm gezeigt. Geben Sie zusätzlich Zeilennummern an, so werden nur die entsprechenden Zeilen aufgelistet. In Verbindung mit dem CMD-Befehl kann die Ausgabe an ein Peripheriegerät gehen, z.B. Drucker oder Diskette. Bei Verwendung von LIST in einem Programm wird der Programmablauf nach LIST unterbrochen.

`LIST`	zeigt das komplette Programmlisting.
`LIST 10`	zeigt nur Programmzeile 10.
`LIST -100`	zeigt Programmlisting bis zur Zeile 100.
`LIST 100-`	zeigt Programmlisting ab Zeile 100 bis zum Ende.
`LIST 10-20`	zeigt nur die Programmzeilen von 10 bis 20.

Im C-128 Modus wird der Programmablauf nach LIST nicht unterbrochen!

LOAD

Kategorie: Befehle

Schreibweise: LOAD "Programmname",GA,SA

Funktion: Mit LOAD können Programme von der DATASETTE oder dem Diskettenlaufwerk in den Programmspeicher geladen werden. Wird nur LOAD eingegeben, wird das nächste Programm von der Kassette eingelesen. Bei Angabe des Programmnamens in Anführungszeichen und der Geräteadresse GA=8 für das Diskettenlaufwerk wird das entsprechende Programm von der Diskette an die interne Speicheradresse 2048 geladen. Das ist normalerweise der BASIC-Anfang. Wird zusätzlich noch die Sekundäradresse SA=1 angegeben, wird das Programm an die Speicheradresse geladen, von der es zuvor abgespeichert wurde.

LOG

Kategorie: Numerische Funktionen

Schreibweise: LOG(X)

Funktion: Mit dieser Funktion erhalten Sie den natürlichen Logarithmus (Basis e) von X, wobei X größer Null sein muß.

MID$

Kategorie: String Funktionen

Schreibweise: MID$(X$,A,B)

Funktion: Diese Funktion definiert einen Teilstring von X$, der B Zeichen von X$ enthält, wobei A das Zeichen bestimmt, von dem ab der Teilstring gebildet wird. Sowohl A als auch B können Werte zwischen 0 und 255 annehmen.

NEW

Kategorie: Befehle

Schreibweise: NEW

Funktion: Mit NEW werden das aktuelle Programm im Speicher sowie alle Variablen gelöscht. Grundsätzlich sollte vor jeder neuen Programmeingabe NEW eingegeben werden. Seien Sie jedoch vorsichtig mit diesem Befehl, da Ihr Programm und somit die Programmierarbeit von vielleicht mehreren Stunden verloren ist, falls es nicht vorher abgespeichert wurde.

NEXT

Kategorie: Befehle

Schreibweise: NEXT X

Funktion: NEXT bezeichnet das Ende der zuvor mit "FOR X=0 TO 10" geöffneten Schleife. Trifft das Programm auf einen NEXT-Befehl, so wird die dazugehörige Laufvariable, hier X, um eins erhöht, falls keine andere Schrittweite angegeben wurde. X wird dann mit dem Endwert verglichen, hier 10, und falls X > 10 ist, wird die Schleife verlassen. Es können mit NEXT mehrere Schleifen auf einmal beendet werden, dazu

müssen lediglich alle Laufvariablen mit angegeben und durch Kommata abgetrennt werden, z.B. NEXT X,Y,Z.

NOT

Kategorie: Logischer Operator

Schreibweise: NOT X

Funktion: Mit NOT können Sie das Ergebnis einer Vergleichsoperation, die die Wahrheitswerte Wahr = -1 / Falsch = 0 enthält, umkehren, d.h. aus WAHR wird FALSCH und umgekehrt.

ON

Kategorie: Befehle

Schreibweise: ON X GOTO 10,20,30 / ON X GOSUB 100,200

Funktion: Mit diesem Befehl haben Sie die Möglichkeit, in Abhängigkeit des Wertes von X nach bestimmten Programmzeilen zu verzweigen bzw. in bestimmte Unterprogramme zu springen, deren Zeilennummern hinter GOTO oder GOSUB aufgeführt sind. Hat X den Wert 1, so würde im obigen Beispiel nach Zeile 10 bzw. in das Unterprogramm ab Zeile 100 verzweigt. Bei geschickter Anwendung kann man mit dieser Befehlsfolge mehrere IF...THEN Befehle einsparen.

OPEN

Kategorie: Befehle (Ein-/Ausgabe)

Schreibweise: OPEN X,GA,SA

Funktion: Der OPEN-Befehl ermöglicht es, Daten zwischen dem Rechner und den Peripheriegeräten auszutauschen. X bezeichnet hier die bereits bei anderen Befehlen erwähnte logische Filenummer. Dies kann eine Zahl zwischen 0 und 255 sein, wobei Zahlen über 128 spezielle Funktionen bei Peripheriegeräten auslösen können. Daher sollten Sie keine Zahlen nehmen, die größer als 128 sind. GA steht für die Geräteadresse. Diese ist in den meisten Fällen für die einzelnen Geräte vorbestimmt, z.B. Diskettenlaufwerk=8, Kassettenlaufwerk=1, Drucker=4. Danach kann in bestimmten Fällen noch die sogenannte Sekundäradresse SA folgen, die ebenfalls durch ein Komma abgetrennt wird. Die Sekundäradresse hat für die einzelnen Peripheriegeräte unterschiedliche Auswirkungen. Beim Drucker hat sie teilweise die Funktion eines Schalters, wobei bestimmte Druckmodi ein- bzw. ausgeschaltet werden. Die genauen Funktionen der Sekundäradresse können Sie den entsprechenden Handbüchern entnehmen.

OR

Kategorie: logischer Operator

Schreibweise: (Ausdruck) OR (Ausdruck)

Funktion: Der logische Operator OR dient ebenfalls dazu, zwei oder mehrere Ausdrücke auf ihren Wahrheitswert hin zu überprüfen, ähnlich wie wir es vom AND her kennen. Der Unterschied liegt darin, daß beim OR lediglich von mehreren Ausdrücken nur einer "wahr" sein muß, um dem gesamten Ausdruck den Wert "wahr" zu geben.

PEEK

Kategorie: Numerische Funktionen

Schreibweise: PEEK(X)

Funktion: Ergibt eine Zahl zwischen 0 und 255, die aus einer Speicheradresse X gelesen wird. X muß im Bereich von 0 bis 65535 liegen.

POKE

Kategorie: Befehle

Schreibweise: POKE X,A

Funktion: Dieser Befehl ist in etwa die Umkehrung der PEEK-Funktion. Mit POKE wird ein Wert A in eine Speicherstelle X geschrieben, wobei A einem Wert zwischen 0 und 255 und X einer Speicherstelle zwischen 0 und 65535 entsprechen muß.

POS

Kategorie: Numerische Funktionen

Schreibweise: POS(A)

Funktion: Mit dieser Funktion erhalten Sie die Cursorposition in der Bildschirmzeile, an der der nächste Print-Befehl ausgeführt würde. A wird nicht zur Berechnung herangezogen bzw. durch einen Wert belegt, ähnlich wie beim FRE(A) Befehl. Statt A kann auch eine Zahl verwendet werden.

PRINT

Kategorie: Befehle

Schreibweise: PRINT (Variable) oder PRINT "TEXT"

Funktion: Der PRINT-Befehl ist wohl einer der vielseitigsten Befehle innerhalb des BASIC vom Commodore 64 überhaupt. Er wird hauptsächlich dazu benutzt, um Daten oder Mitteilungen vom Programm auf dem Bildschirm auszugeben. Folgt dem PRINT eine Variable, so wird der aktuelle Wert der Variablen ausgedruckt; steht dagegen eine Zeichenkette in Anführungszeichen, so wird die Zeichenkette genauso ausgedruckt.

PRINT#

Kategorie: Befehle (Ausgabe)

Schreibweise: PRINT#X,"Daten"

Funktion: Mit PRINT# werden Daten in eine Datei geschrieben, die vorher mit OPEN geöffnet wurde. X bezeichnet wieder die logische Filenummer und "Daten" steht hier für spezielle Befehle oder Variablen, die in die Datei geschrieben werden sollen.

READ

Kategorie: Befehle

Schreibweise: READ X

Funktion: Liest ein Element einer DATA-Zeile und ordnet es der Variablen X zu. Die zu lesenden Elemente müssen mit dem angegebenen Variablentyp übereinstimmen; in diesem Beispiel wären also für X nur numerische Werte zugelassen. Werden mehr READ-Befehle gegeben als DATA-Elemente vorhanden, so wird die Fehlermeldung "?OUT OF DATA ERROR" ausgegeben.

REM

Kategorie: Anweisungen

Schreibweise: REM TEXT

Funktion: Mit REM können innerhalb von Programmen Erklärungen zu einzelnen Programmteilen gegeben werden, ohne daß dadurch der eigentliche Ablauf des Programms beeinflußt wird. Wird die REM-Anweisung in einer Zeile benutzt, so wird alles, was der REM-Anweisung in dieser Zeile folgt, vom Programm ignoriert.

RESTORE

Kategorie: Befehle

Schreibweise: RESTORE

Funktion: Bei Benutzung des READ-Befehls wird immer ein Zeiger auf das nächste zu lesende Element in der DATA-Zeile gesetzt. Mit RESTORE wird dieser Zeiger wieder an den Anfang der DATA-Zeile gesetzt, so daß durch diesen Befehl ein Element mehrmals gelesen werden kann.

RETURN

Kategorie: Befehle

Schreibweise: RETURN

Funktion: Mit diesem Befehl wird ein Unterprogramm beendet. Trifft das Programm auf den RETURN-Befehl, so springt es zu der Zeile, wo der zugehörige GOSUB-Befehl stand, und fährt dort mit der Ausführung des Programms fort.

RIGHT$

Kategorie: String Funktionen

Schreibweise: RIGHT$ (X$,A)

Funktion: Ergibt einen Teilstring der rechten Seite von X$, dessen Länge durch den Wert A bestimmt wird. A kann Werte zwischen 0 und 255 annehmen. Ist der Wert größer als die ursprüngliche Stringlänge, so wird der ganze String wiedergegeben.

RND

Kategorie: Numerische Funktionen

Schreibweise: RND (X)

Funktion: RND erzeugt eine Zufallszahl zwischen 0.0 und 1.0. Der Wert von X hat bis auf das Vorzeichen keine Bedeutung. Bei positiven X-Werten wird stets die gleiche Folge von Zufallszahlen bei gegebenem Startwert X erzeugt. Unterschiedliche Zahlenfolgen erhält man durch Änderung der Startwerte.

RUN

Kategorie: Befehle

Schreibweise: RUN (Zeilennummer)

Funktion: Mit RUN wird ein Programm gestartet. Bei Bedarf kann zusätzlich eine Zeilennummer mit angegeben werden, von der aus das Programm starten soll. Es werden automatisch alle Variablen auf Null gesetzt. Will man dies vermeiden, so kann man den GOTO-Befehl mit Angabe der Zeilennummer verwenden. Dabei bleiben die Variablenwerte erhalten.

SAVE

Kategorie: Befehle

Schreibweise: SAVE "Name",GA

Funktion: SAVE speichert ein im Programmspeicher enthaltenes Programm auf Kassette oder Diskette ab. Wird SAVE ohne Angabe der Geräteadresse GA angewendet, so wird automatisch auf die Kassette abgespeichert. Durch Angabe der Geräteadresse 8 kann man das Programm auf das Diskettenlaufwerk abspeichern.

SGN

Kategorie: Numerische Funktionen

Schreibweise: SGN (X)

Funktion: Mit SGN (X) erhalten sie einen ganzzahligen Wert, welcher abhängig vom Vorzeichen von X ist. Bei X > 0 erhält man 1, bei X = 0 erhält man 0 und bei X < 0 erhält man -1.

SIN

Kategorie: Numerische Funktionen

Schreibweise: SIN (X)

Funktion: Mit dieser Funktion erhalten Sie den Sinus des Winkels von X, wobei X im Bogenmaß anzugeben ist.

SPC

Kategorie: String Funktionen

Schreibweise: SPC (X)

Funktion: Mit SPC hat man die Möglichkeit, X Zeichen zu überspringen, um dann an der Stelle mit PRINT eine Ausgabe auf dem Bildschirm zu erhalten. X kann hier wieder Werte zwischen 0 und 255 annehmen.

SQR

Kategorie: Numerische Funktionen

Schreibweise: SQR (X)

Funktion: Ergibt den Wert der Quadratwurzel von X. X darf keine negativen Werte annehmen.

STEP

Kategorie: Befehle

Schreibweise: STEP X

Funktion: STEP wird bei der Programmierung von Schleifen benutzt, um die Schrittweite für die Laufvariable anzugeben. X kann jeden beliebigen Wert außer Null annehmen. Wird STEP nicht benutzt, so ist die Schrittweite gleich 1.

STOP

Kategorie: Befehle

Schreibweise: STOP

Funktion: STOP wird innerhalb eines Programms benutzt, um den Programmablauf an dieser Stelle zu unterbrechen. Als Meldung auf dem Bildschirm erscheint BREAK IN X, wobei X die Zeilennummer angibt, in der unterbrochen wurde. Mit CONT kann die Programmausführung wieder aufgenommen werden.

STR$

Kategorie: String Funktionen

Schreibweise: STR$ (X)

Funktion: Durch STR$ wird X in einen String umgewandelt. Ist X positiv oder Null, so beginnt der String mit einem Leerzeichen.

SYS

Kategorie: Befehle

Schreibweise: SYS X

Funktion: Mit SYS wird ein Maschinensprache-Programm aufgerufen, das an der Adresse X beginnt, wobei X Werte zwischen 0 und 65535 annehmen kann. Ein bekannter SYS Befehl ist der System-Kaltstart mit SYS 64738 (C-64 Modus).

TAB

Kategorie: String Funktionen

Schreibweise: TAB (X)

Funktion: Mit TAB kann der Cursor an eine durch X bestimmte Zeilenposition gebracht werden. Der Wert von X kann zwischen 0 und 255 liegen. Die Zählung beginnt an der äußersten linken Position der momentanen Zeile. TAB sollte nur in Verbindung mit dem PRINT-Befehl verwendet werden, da er zusammen mit PRINT# nicht interpretiert werden kann.

TAN

Kategorie: Numerische Funktionen

Schreibweise: TAN (X)

Funktion: Ergibt den Tangens des Winkels von X, wobei X im Bogenmaß angegeben werden muß.

USR

Kategorie: Numerische Funktionen

Schreibweise: USR (X)

Funktion: Mit USR wird in ein Maschinenunterprogramm verzweigt, dessen Startwert zuvor in den Speicherstellen 785-786 abgelegt sein muß. Dies geschieht durch entsprechende POKE-Befehle. Der Wert von X wird an das Maschinenprogramm übergeben, welches selbst einen anderen Wert an das BASIC-Programm zurückgibt. Diese Möglichkeit der Parameterübergabe existiert bei dem SYS-Befehl nicht.

VAL

Kategorie: Numerische Funktionen

Schreibweise: VAL (X$)

Funktion: VAL wandelt X$ in einen numerischen Wert um. Trifft VAL im String auf ein Zeichen, welches nicht in eine Zahl umgewandelt werden kann, z.B. A, wird nur der Teil bis zum A in einen Zahlenwert umgewandelt. Ist das erste Zeichen, außer Leer-, Plus- oder Minuszeichen, nicht in eine Zahl überführbar, so ergibt sich der Wert Null.

VERIFY

Kategorie: Befehle

Schreibweise: VERIFY "Datei",GA

Funktion: Mit VERIFY kann ein gerade abgespeichertes Programm mit dem im Speicher befindlichen Programm verglichen werden; hiermit kann also überprüft werden, ob ein Programm ordnungsgemäß abgespeichert wurde. GA gibt hier wieder die Geräteadresse an. Diese kann bei Kassettenbetrieb vernachlässigt werden. Mit dem Befehl VERIFY "*",8 wird automatisch das zuletzt abgespeicherte Programm auf Diskette mit dem im Speicher befindlichen verglichen.

WAIT

Kategorie: Befehle

Schreibweise: WAIT X,Y

Funktion: Mit WAIT wird die Programmausführung solange angehalten, bis die Speicherstelle X den Wert Y (Bitmuster) angenommen hat.

BASIC 7.0

AUTO

Kategorie: Befehle

Schreibweise: AUTO (Schrittweite)

Funktion: Dieser Befehl schaltet die automatische Zeilennumerierung ein. Sobald eine Programmzeile mit RETURN in den Speicher übernommen wird, gibt der Rechner die nächste Zeilennummer aus. Nachdem Sie z.B. AUTO 10 eingegeben haben, müssen Sie die erste Zeilennummer selbst vorgeben. Nachdem dann diese Programmzeile mit Return in den Programmspeicher des Rechners übernommen wurde, wird die nächste Zeilennummer im Zehnerabstand automatisch vom Rechner vorgegeben. Haben Sie also mit Zeile 20 angefangen, wird als nächstes Zeilennummer 30 ausgegeben. Beginnen Sie dagegen mit Zeilennummer 25, so ist 35 die nächste Zeilennummer. Die Schrittweite gibt also nur den Abstand zu den einzelnen Zeilennummern an.

Beispiel:

`AUTO 10`	automatische Zeilennumerierung Zehnerschritten.
`AUTO 100`	automatische Zeilennumerierung in Hunderterschritten.
`AUTO`	schaltet die automatische Zeilennumerierung wieder aus.

BANK

Kategorie: Anweisung

Schreibweise: BANK (Nr.)

Funktion: Mit dem Befehl BANK (Nummer) können Sie eine der 64 K Bänke auswählen, auf die sich nachfolgende POKE- , PEEK- oder WAIT-Anweisungen beziehen sollen.

Beispiel:

BANK 0

Wählt für nachfolgende POKE-, PEEK- oder WAIT-Anweisungen die erste Bank aus.

```
10 BANK 0:POKE 8000,255
20 BANK 1:POKE 8000,100
30 BANK 0:PRINT PEEK(8000)
40 BANK 1:PRINT PEEK(8000)
```

Starten Sie dieses kleine Programm, so werden Sie die Wirkungsweise der BANK-Anweisung sehen.
Viele von Ihnen kennen vielleicht den POKE-Befehl des Commodore 64, um die Farbe des Bildschirmrahmens zu ändern (z.B. POKE 53280,0 = Rahmen schwarz). Dieser Befehl wirkt beim C-128 nur dann (die Adresse ist beim C-64 die gleiche), wenn Sie vorher mit BANK 4 die richtige Bank bzw. Speicherkonfiguration ausgewählt haben.

BEGIN...BEND

Kategorie: Befehle

Schreibweise: IF...THEN...BEGIN (Anweisungen)... BEND:ELSE.

Funktion: BEGIN/BEND stellt eine leistungsstarke Erweiterung zur IF...THEN Anweisung dar. Bei der Programmierung mit IF...THEN war man bisher gezwungen, diese Anweisung bzw. die Anweisungen, die dem THEN folgten, in einer Programmzeile unterzubringen. Diese Einschränkung existiert durch BEGIN/BEND nicht mehr.

BEGIN/BEND kennzeichnet einen Block von Anweisungen, der von einer IF...THEN Anweisung so behandelt wird, als würden diese Anweisungen alle in einer Zeile stehen. Damit stellt Ihnen das BASIC 7.0 eine Möglichkeit zur Verfügung, ganze Programmteile innerhalb einer IF...THEN Anweisung ablaufen zu lassen und nicht wie bisher beim Commodore 64 nur einen oder wenige Befehle. Größere Programme werden durch diese Möglichkeit besser strukturiert und damit auch übersichtlicher.

Selbstverständlich ist es möglich, innerhalb des BEGIN...BEND Blocks weitere IF...THEN Anweisungen unterzubringen, d.h. Schachtelungen vorzunehmen. Dies wird auch in dem nun folgenden Beispiel deutlich.

Beispiel:

```
10 A=INT(50*RND(1))+1
20 PRINT "SIE MUESSEN EINE ZAHL ZWISCHEN"
30 PRINT "1 UND 50 ERRATEN."
40 INPUT"IHRE ZAHL";R
50 IF R<>A THEN BEGIN :
60 :  IF R<A THEN PRINT "ZU KLEIN"
70 :  IF R>A THEN PRINT "ZU GROSS"
80 BEND:GOTO 40:ELSE PRINT "RICHTIG!":END
```

Nachdem Sie dieses Programm in den Rechner eingegeben haben, werden Sie aufgefordert, eine Zahl zwischen 1 und 50 zu erraten. Jede Eingabe von Ihnen wird im Rechner sofort daraufhin überprüft, ob die eingegebene Zahl größer oder kleiner als die gesuchte ist. Das Ergebnis wird vom Rechner ausgegeben (ZU KLEIN bzw. ZU GROSS oder RICHTIG). Die Abfrage und die Antwort des Rechners ist in den Programmzeilen 50 bis 80 programmiert. Hier wurden, wie bereits erwähnt, weitere IF...THEN Anweisungen im BEGIN...BEND Block untergebracht. Wie Sie an diesem kleinen Beispiel schon erkennen können, eröffnen sich mit BEGIN...BEND ganz neue Möglichkeiten der Programmierung. Hier noch ein Beispiel zur Verdeutlichung:

```
10 INPUT "WIE OFT";A
20 IF A<100 THEN BEGIN:
30 : PRINT "DIE GEWAEHLTE ANZAHL WAR ";A
40 : FOR Z=1 TO A
50 : PRINT "DEMONSTRATION VON  BEGIN...BEND"
60 : NEXT Z
70 : PRINT "HIER IST SCHLUSS!"
80 BEND:ELSE PRINT "ANZAHL ZU GROSS!":END
```

BOOT

Kategorie: Befehle

Schreibweise: BOOT "Dateiname"(,Ddn,Ugn)

Funktion: Dieser Befehl wird benutzt, um Binärfiles zu laden und auszuführen. Selbstverständlich muß es sich dabei um für den Rechner interpretierbare Binärfiles handeln.

Beispiel:

```
BOOT "MASCH III"
```

Lädt das Binärfile 'MASCH III' und beginnt an der Startadresse des Files mit der Ausführung. Mit BOOT können also auch z.B. Programme geladen und ausgeführt werden, die mit BSAVE abgespeichert wurden. Sie werden durch BOOT automatisch an die Adresse geladen, von der sie mit BSAVE abgespeichert wurden.

Wird BOOT ohne Angabe eines Programmnamens eingegeben, so prüft das System, ob auf der ersten Spur der eingelegten Diskette eine bestimmte Bytefolge vorhanden ist. Ist dies der Fall (z.B. bei einer CP/M-Diskette), wird automatisch CP/M geladen und damit das System neu initialisiert.

DEC

Kategorie: Funktionen

Schreibweise: DEC (Hex.-Ausdruck)

Funktion: Diese Funktion wandelt eine hexadezimale Zahl in ihr dezimales Äquivalent um. Damit entfällt die manchmal lästige Umrechnerei von hexadezimal nach dezimal, was eine willkommene Bereicherung darstellt, will man z.B. eine Adresse aus einem ROM-Listing schnell in eine dezimal Zahl umrechnen.

Beispiel:

```
PRINT DEC("0AFF")
```

Ausgabe: 2815

Das folgende kleine Programm stellt die hexadezimalen Zahlen 00 bis FF ihren entsprechenden dezimalen Zahlen 0 bis 255 gegenüber.

```
10 FOR Z=48 TO 70
20 IF Z=58 THEN Z=65
30 FOR I=48 TO 70
40 IF I=58 THEN I=65
50 Z$=CHR$(Z)+CHR$(I)
60 GOSUB 100
70 NEXT I:NEXT Z
90 END
100 PRINT "$";Z$;" =";DEC(Z$):RETURN
```

DELETE

Kategorie: Befehle

Schreibweise: DELETE (Zeilennr.)-(Zeilennr.)

Funktion: Dieser Befehl löscht eine oder mehrere BASIC-Programmzeilen. Besonders nützlich ist dieser Befehl, wenn Sie aus einem bereits existierendem Programm nur bestimmte Zeilen übernehmen wollen, z.B. die der Eingaberoutine, um diese dann in einem neuen Programm zu verwenden. Wollten Sie diese Technik bein C-64 anwenden, so waren Sie gezwungen, falls Sie keine entsprechenden Programmiertools hatten, jede Zeilennummer von Hand zu löschen. Diese Arbeit gehört jetzt der Vergangenheit an.
Die Syntaxregeln für diesen Befehl sind ähnlich dem LIST-Befehl.

Beispiel:

`DELETE 10`	löscht Zeilennummer 10.
`DELETE -100`	löscht alles bis Zeile 100.
`DELETE 200-250`	löscht von Zeile von 200 bis 250.
`DELETE 500-`	löscht ab Zeile 500 alle Programmzeilen.

DO/LOOP/WHILE/UNTIL/EXIT

Kategorie: Anweisungen

Schreibweise: DO:(UNTIL Anweisg. / WHILE Anweisg.: Anweisg.:EXIT) LOOP (UNTIL Anweisg. / WHILE Anweisg.)

Funktion: Mit DO...LOOP stellt Ihnen das BASIC 7.0 eine Anweisung für den Aufbau von Programmschleifen zur Verfügung, die man bisher nur von strukturierten Sprachen her kannte. Im einfachsten Fall stellt die Anweisung DO den Anfang und die Anweisung LOOP das Ende der Schleife dar. Dem DO bzw. LOOP können noch die zusätzlichen Anweisungen UNTIL oder WHILE folgen. Innerhalb der DO...LOOP-Schleife kann zusätzlich die Anweisung EXIT benutzt werden. Trifft das Programm auf ein EXIT, wird der nächste Befehl der dem LOOP folgt ausgeführt (ähnlich wie bei FOR...NEXT).

Beispiel:

```
10 DO: PRINT"COMMODORE 128"
20 Z=Z+1
30 LOOP UNTIL Z=25
40 END
```

In diesem Beispiel wird die Schleife so oft durchlaufen, 'BIS' Z den Wert 25 erreicht hat (Zeile 30). UNTIL wird also benutzt, um einen Programmteil abzuarbeiten, bis ein bestimmtes Ereignis eintritt bzw. bis eine bestimmte Bedingung erfüllt ist.

```
10 DO: PRINT"COMMODORE 128"
20 Z=Z+1
30 LOOP WHILE Z < 25
40 END
```

Dieses Beispiel zeigt eine Anwendung von WHILE. Die Schleife wird abgearbeitet, solange Z < 25 ist (Zeile 30). WHILE wird benutzt, um einen Programmteil abzuarbeiten, während eine

bestimmte Bedingung erfüllt ist. Ist die Bedingung nicht mehr erfüllt (Z größer 25), wird das Programm beendet.

```
10 DO
20 PRINT "WELCHEN COMPUTER BEVORZUGEN SIE";
30 INPUT A$
40 LOOP UNTIL A$="COMMODORE 128"
50 PRINT "SIE HABEN EINE GUTE WAHL GETROFFEN."
60 END
```

In diesem Fall wird die DO...LOOP-Schleife solange durchlaufen, bis auf die Frage nach dem bevorzugten Computer mit COMMODORE 128 geantwortet wird. Man hätte mit EXIT auch wie folgt programmieren können:

```
35 IF A$="COMMODORE 128" THEN EXIT
40 LOOP
```

ERR$

Kategorie: Funktionen

Schreibweise: ERR$(X)

Funktion: Diese Funktion gibt einen Stringausdruck zurück, der einen Fehler näher beschreibt. Somit kann in Verbindung von TRAP, RESUME und den Systemvariablen ER und EL eine genaue Fehlermeldung ausgegeben werden, ohne daß das Programm abgebrochen wird.

Beispiel:

```
10 FOR I=1 TO 41
20 PRINT ERR$(I)
30 NEXT I
40 END
```

Dieses Programm gibt Ihnen alle zur Verfügung stehenden Fehlerstrings aus.

Wollen Sie also in Ihren Fehlerroutinen die original Systemfehlermeldungen verwenden, so stellt Ihnen die ERR$ Funktion diese zur Verfügung. Damit ersparen Sie sich die Textausgabe mit dem PRINT Befehl.

FAST / SLOW

Kategorie: Befehle

Schreibweise: FAST

Funktion: Der Befehl FAST gibt Ihnen von BASIC aus die Möglichkeit, den C-128 mit 2 MHz zu fahren, d.h. mit der doppelten Geschwindigkeit. Dies ist vor allem bei langwierigen Berechnungen von Vorteil. Da aber in dieser Betriebsart der VIC mit dem Bildschirmaufbau (40 Zeichen) nicht mehr mitkommt, wird der 40 Zeichenschirm gleich ausgeblendet, d.h. nach Eingabe dieses Befehls sehen Sie auf dem 40 Zeichen-Bildschirm nichts mehr. Die 80 Zeichendarstellung bleibt davon unberührt. Durch Eingabe von SLOW können Sie den C-128 wieder auf 1 MHz umschalten. Gleichzeitig wird auch wieder der 40 Zeichenschirm aktiviert. Sie können diese Befehle z.B. dazu benutzen, um auf dem 40 Zeichenschirm unsichtbar im FAST-Modus eine Grafik aufbauen zu lassen. Gleichzeitig geben Sie auf dem 80 Zeichenschirm eine Meldung wie "Bitte warten." aus. Danach schalten Sie mit SLOW wieder in den Normalmodus. Mit dem folgenden Beispielprogramm können Sie den Geschwindigkeitsunterschied feststellen.

Beispiel:

```
10 REM 1 MHZ
20 FOR I=1 TO 5000:NEXT I
30 GETKEY A$
40 FAST : REM 2 MHZ
50 FOR I=1 TO 5000:NEXT I:SLOW:END
```

FETCH

Kategorie: Anweisungen

Schreibweise: FETCH a,b,c,d

Funktion: Mit FETCH können komplette Speicherbereiche aus den Speicherbänken 1 bis 15 in den BASIC-Arbeitsspeicher (Bank 1) geladen werden. Dadurch ist es z.B. möglich, Variableninhalte in eine andere Bank wegzuschreiben, also zwischenzuspeichern, um diese bei Bedarf wieder in den Arbeitsspeicher zurückzuholen (s.a. STASH).

Die Parameter a,b,c geben die Anzahl sowie Start- und Zieladresse an. Parameter d bestimmt die Bank, aus der gelesen werden soll. Für a, b, c gilt der Maximalwert von 65535 und für d dürfen Werte zwischen 1 und 15 benutzt werden.

Beispiel:

```
FETCH 144,24576,16831,15
```

GETKEY

Kategorie: Anweisungen

Schreibweise: GETKEY A$

Funktion: GETKEY arbeitet ähnlich wie GET, nur mit dem Unterschied, daß GETKEY auf einen Tastendruck wartet, während GET sofort weiter im Programm fortfährt.

Beispiel:

Bei GET mußten Sie bisher in der folgenden Art und Weise verfahren:

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

Dies ist bei GETKEY nicht mehr notwendig. GETKEY wartet bis eine Taste gedrückt wird und ordnet diesen Wert der Variablen zu. Die Zeile ändert sich demnach wie folgt:

```
10 GETKEY A$
```

Wollen Sie in Ihrem Programm warten bis die Taste "A" gedrückt wird, dann könnte das ungefähr so aussehen:

```
.
.
.
70 GETKEY A$
80 IF A$ <> "A" THEN 70
.
.
.
```

GO 64

Kategorie: Anweisungen

Schreibweise: GO 64

Funktion: Mit dieser Anweisung können Sie vom C-128 Modus in den C-64 Modus umschalten. Nachdem Sie 'GO 64' eingegeben haben, werden Sie gefragt, ob Sie auch bestimmt den C-128 Modus verlassen wollen:

ARE YOU SURE?

(Sind Sie sicher?) Geben Sie jetzt 'Y' ein, so sehen Sie nach einem kurzen Augenblick die Einschaltmeldung des CBM 64. Ihnen steht nun ein original C-64 zur Verfügung, auf dem die gesamte 64er Software einsetzbar ist. Wollen Sie vom C-64 Modus wieder zurück in den C-128 Modus, so gelingt Ihnen das nur durch das Betätigen des Resettasters oder aber durch die weniger elegante Art des Aus- und wieder Einschaltens des Rechners.

HELP

Kategorie: Befehle

Schreibweise: HELP

Funktion: Nach dem Auftreten einer Fehlermeldung (SYNTAX ERROR o.ä.) kann dieser Befehl eingegeben werden. Die Programmzeile, in der der Fehler auftrat, wird angezeigt, und die vermutliche Fehlerstelle erscheint im 40 Zeichenmodus revers. Im 80 Zeichenmodus wird die Stelle unterstrichen dargestellt.

Beispiel:

```
10 SCNCLR
20 GOSUP 100 REM INITROUTINEN
30 ...
 .
 .
 .
```

Wird dieses Programm gestartet, so bricht es mit der Fehlermeldung ?SYNTAX ERROR IN 20 ab, da ein Schreibfehler beim GOSUB vorliegt. Geben Sie nun

HELP

ein, so wird die fehlerhafte Zeile angezeigt:

```
20 GOSUP 100 REM INITROUTINEN
```

HEX$

Kategorie: Funktionen

Schreibweise: HEX$ (X)

Funktion: Diese Funktion wandelt eine dezimale Zahl in eine hexadezimale Zahl um. X darf dabei im Bereich von 0 bis 65535 liegen. (0 <= X <= 65535)

Beispiel:

```
PRINT HEX$(49152)
```

Ausgabe: C000

In Verbindung mit der DEC-Funktion, haben Sie jetzt endlich auch die Möglichkeit, innerhalb des BASIC 7.0 dezimale Werte in hexadezimale umzurechnen und umgekehrt. Damit können Sie nun auf einfache Art und Weise mal schnell einen Maschinencode z.B. A9 oder eine vielleicht nicht so bekannte Speicheradresse z.B. FF53 in eine dezimale Zahl umwandeln, um diese in Ihrem Programm einzusetzen.

Beispiel:

```
10 OPEN 1,4:CMD1
20 FOR I=0 TO 255
30 PRINT#1,I;"= ";HEX$(I)
40 NEXT I:CLOSE 1
```

IF/THEN/ELSE

Kategorie: Anweisungen

Schreibweise: IF ... THEN ... : ELSE

Funktion: Die Wirkungsweise von IF...THEN wurde bereits im Kapitel des BASIC 2.0 erklärt. Im BASIC 7.0 erhielt IF...THEN noch die Ergänzung 'ELSE'. Dadurch wird es möglich, noch in der gleichen Programmzeile auf eine 'nicht wahre' Bedingung zu reagieren.

Beispiel:

```
11Ø IF A$="J" THEN A=5 : ELSE GOSUB 1ØØØ
```

Hat in diesem Beispiel die Variable A$ nicht 'J' als Inhalt wird ein Unterprogramm in Zeile 1000 ausgeführt.

Wichtig: Das ELSE benötigt einen Doppelpunkt nach der THEN-Anweisung.

Beispiel:

```
1Ø GETKEY A$
2Ø IF ASC(A$) > 47 AND ASC(A$) < 58 THEN 3Ø : ELSE IF ASC(A$)=69 THEN 3Ø
: ELSE 1Ø
3Ø PRINT A$;
```

Dieses Beispiel läßt für die Eingabe nur 0 bis 9 und E zu.

INSTR

Kategorie: Funktionen

Schreibweise: INSTR (str1,str2(,stnr))

Funktion: Diese Funktion gibt die Position an, ab der String 2 (str2) in String 1 (str1) zu finden ist. Ist String 2 nicht enthalten erhält man den Wert 0. Zusätzlich kann noch die Position mit angegeben werden, ab der String 1 nach String 2 durchsucht werden soll (stnr).

Beispiel:

```
10 A$="COMMODORE 128":B$="128":C$="DORE"
20 P1=INSTR(A$,B$)
30 P2=INSTR(A$,C$,3)
40 PRINT P1,P2
```

Ausgabe: 11 6

Zeile 20 durchsucht den String A$ ab dem ersten Zeichen nach B$. Das Ergebnis wird der Variablen P1 zugeordnet. P1 erhält den Wert 11, da der gesuchte String B$ ab dem elften Zeichen in A$ zu finden ist. Zeile 30 durchsucht den String A$ ab dem dritten Zeichen auf C$. Das Ergebnis lautet 6, da C$ ab dem 6.Zeichen in A$ zu finden ist.

JOY

Kategorie: Funktionen

Schreibweise: JOY (X)

Funktion: Mit JOY können die Joystickports abgefragt werden. Für X=1 wird der Wert für Port 1 und für X=2 der Wert für Port 2 wiedergegeben.

Die Werte haben dabei die folgende Bedeutung:

0	keine Funktion
1	Oben
2	Oben/Rechts
3	Rechts
4	Unten/Rechts
5	Unten
6	Unten/Links
7	Links
8	Oben/links
128	Feuertaste

Werte über 128 besagen, daß die Feuertaste gleichzeitig mit einer der anderen Funktionen betätigt wurde. Mit dieser Funktion ersparen Sie sich also die umständliche Handhabung der Joystickports mit PEEK- und POKE-Befehlen, wie Sie es vielleicht vom C-64 her kennen. Damit ist es auch dem Einsteiger möglich diese Ports abzufragen, ohne gleich "intern" werden zu müssen.

KEY

Kategorie: Befehle

Schreibweise: KEY (KEY nr,"Zuweisung")

Funktion: Dieser Befehl besitzt zwei Funktionen. Geben Sie nur 'KEY' ein, so werden die Belegungen der acht Funktionstasten (F1-F8) aufgelistet. Geben Sie 'KEY' gefolgt von einer Nummer ein, so können Sie für diese Funktionstaste eine neue Belegung definieren.

Beispiel:

KEY listet alle Funktionstastenbelegungen auf:

```
KEY 1,"GRAPHIC"
KEY 2,"DLOAD"+CHR$(13)
KEY 3,"DIRECTORY"+CHR$(13)
KEY 4,"SCNCLR"+CHR$(13)
KEY 5,"DSAVE"+CHR$(13)
KEY 6,"RUN"+CHR$(13)
KEY 7,"LIST"+CHR$(13)
KEY 8,"MONITOR"+CHR$(13)
```

```
KEY 6,CHR$(27)+"X"+CHR$(13)
```

Definiert für die Funktionstaste 6 eine ESC X Sequenz.

MID$

Kategorie: Funktionen

Schreibweise: MID$(A$,X,Y)

Funktion: MID$ wurde im BASIC 7.0 um eine sinnvolle Anwendung erweitert. Man hat die Möglichkeit, über MID$ innerhalb eines Strings einzelne Zeichen neu zuzuordnen. Sonst sind alle Funktionen zum BASIC 2.0 erhalten geblieben.

Beispiel:

```
10 A$="COMMODORE 64 "
20 MID$(A$,11,3)="128"
30 PRINT A$
40 END
```

Ausgabe: COMMODORE 128

Die Zuordnung im obigen Beispiel war nur deshalb möglich, da A$ bereits eine Zeichenlänge von 13 Zeichen hatte. Hätte in Zeile 10 das Leerzeichen hinter der Ziffer 4 gefehlt, so hätte diese Zuordnung nicht stattfinden können, da der neue String mehr Zeichen beinhaltet hätte, als der ursprüngliche. Diese Eigenart muß bei der Zuordnung mit MID$ unbedingt berücksichtigt werden.

MONITOR

Kategorie: Befehle

Schreibweise: MONITOR

Funktion: Dieser Befehl schaltet den eingebauten Maschinensprachemonitor ein. Damit können Sie auf einfache Weise Maschinenprogramme erstellen. Ihnen stehen Befehle wie A(ssemble), D(isassemble), C(ompare), F(ill), H(unt), M(emory), R(egister), T(ransfer) usw. zur Verfügung. Durch Eingabe von 'X' kehren Sie zurück zum BASIC 7.0.

Im Kapitel 7 wurde der Monitor bereits besprochen. Aus Gründen der Vollständigkeit werden die restlichen Befehle hier aufgeführt.

G	ausführen des Programms
L	laden einer Datei in den Speicher
S	speichern eines Bereichs auf Peripheriegerät
V	VERIFY von Daten
X	Monitor verlassen
>	Speicher ändern
;	Prozessorregister ändern
@	Statusmeldung Diskettenlaufwerk

PEN

Kategorie: Funktionen

Schreibweise: PEN (X)

Funktion: Mit dieser Funktion kann die aktuelle Position des Lightpen ermittelt werden. PEN(0) gibt die X-Position und PEN(1) die Y-Position des Lightpens wieder.

Der Lightpen oder auch Lichtgriffel ist ein Hilfsmittel, mit dem der Anwender Zeichnungen fertigen oder auch Menüpunkte anwählen kann. Dazu braucht er nur den Lightpen auf die entsprechende Position auf dem Bildschirm zu richten und dann den Taster zu betätigen. In diesem Moment stehen dann, abhängig von der Position des Lightpen, bestimmte Werte am entsprechenden Port zur Verfügung.

Die Werte müssen im Programm entsprechend aufbereitet werden, da es sich hier nicht um skalierte Werte wie z.B. bei der Grafik handelt (z.B. 0-319). Der Wert für X wird sich etwa im Bereich von 60 bis 320 bewegen und der Y-Wert im Bereich von 50 bis 250.

POINTER

Kategorie: Funktionen

Schreibweise: POINTER (Variable)

Funktion: Mit der Pointerfunktion haben Sie die Möglichkeit, die jeweilige Speicheradresse eines bestimmten Variablenzeigers ausfindig zu machen. Sobald Sie eine Variable definieren - dabei ist es gleich ob Sie das im Direktmodus oder im Programmodus ausführen - wird für diese Variable ein Zeiger angelegt, der wiederum auf einen Speicherbereich zeigt, in dem der Inhalt dieser Variablen abgelegt ist. Die Pointerfunktion teilt Ihnen also die Speicheradresse dieses Variablenzeigers mit. Durch Auslesen dieses Speicherbereichs können Sie dann die Variablenlänge, z.B.

bei einer Stringvariablen, und die Speicheradresse des Variableninhalts ausfindig machen.

Beispiel:

Sie definieren A$="TEST".

Geben Sie jetzt

```
PRINT POINTER(A$)
```

in den Rechner ein, so erhalten Sie z.B. die Ausgabe

```
1034
```

POT

Kategorie: Funktionen

Schreibweise: POT (X)

Funktion: POT(X) ermittelt die Werte für die Paddles, wobei für X die Werte 1 bis 4 zulässig sind.

X=1	Position von Paddle 1
X=2	Position von Paddle 2
X=3	Position von Paddle 3
X=4	Position von Paddle 4

Das folgende kleine Programm steuert Sprite 1 in Abhängigkeit von der Stellung des Paddle 1 horizontal über den Bildschirm. Solche Routinen werden Sie später auch in Ihren selbstgeschriebenen Spielprogrammen verwenden können.

Beispiel:

```
10 SPRITE 1,1,1 : REM SPRITE 1 EINSCHALTEN
20 PA=POT(1)    : REM LESEN PADDLE 1
30 MOVSPR 1,PA,90
40 GOTO 20
```

PRINT USING

Kategorie: Anweisungen

Schreibweise: PRINT USING "Format";A

Funktion: Die PRINT USING Anweisung dient zur formatierten Ausgabe von String- bzw. Zahlenvariablen. Das Format der Ausgabe wird in Anführungszeichen angegeben, denen dann die auszugebende Variable bzw. die auszugebenden Variablen folgen. Zur Definition des Formates werden folgende Zeichen benutzt:

\# Nummernkreuz bestimmt die auszugebende Anzahl der Zeichen.

\+ wird bei positiven Zahlen mit ausgegeben.

\- wird bei negativen Zahlen mit ausgegeben. Plus- oder Minuszeichen können nur getrennt verwendet werden.

. kennzeichnet die Position des Dezimalpunktes.

$ wird benutzt, um z.B. Währungen zu kennzeichnen.

↑↑↑↑ Zahlen werden in wissenschaftlicher Notation ausgegeben. (z.B. 1.23 E+02)

= String wird innerhalb des Formatfeldes zentriert ausgegeben.

> String wird innerhalb des Formatfeldes rechtsbündig ausgegeben.

Beispiel:

A soll nacheinander folgende Werte annehmen:

```
A=34.57 : A=1234 : A=5421.236 : A=54632

PRINT USING "####.##";A
```

Ausgabe:
```
  34.57
1234.00
5421.24
*******
```

Ist eine Zahl größer als das angegebene Format - in diesem Beispiel 5 statt 4 Ziffern vor dem Dezimalpunkt -, so wird das Formatfeld mit Sternchen aufgefüllt.

Die gleichen Werte sollen jetzt noch einmal in der wissenschaftlichen Notation ausgegeben werden.

```
PRINT USING "#.##↑↑↑↑";A
```

Ausgabe:
```
3.46E+01
1.23E+03
5.42E+03
5.46E+04
```

PUDEF

Kategorie: Anweisungen

Schreibweise: PUDEF "(1-4 Zeichen)"

Funktion: Mit PUDEF können die Zeichen in der PRINT USING Anweisung nachträglich verändert werden. Die erste Position, in dem maximal vier Zeichen umfassenden String, definiert das Leerzeichen. Die zweite Position definiert das

Komma, die dritte den Dezimalpunkt und die vierte Position das Dollarzeichen.

Beispiel:

Normalerweise wird zum Auffüllen des Formats das Leerzeichen ausgegeben. Hier soll jetzt der Stern * verwendet werden (wird z.B. bei Überweisungsträgern angewendet). Ferner sollen die Tausenderstellen durch einen Punkt gekennzeichnet werden und der Dezimalpunkt durch ein Komma.

```
10 PUDEF "*., "
20 A=4325.674
30 PRINT USING "###,###.##",A
```

Ausgabe: **4.325,67

RCLR

Kategorie: Funktionen

Schreibweise: RCLR (X)

Funktion: Ermittelt die aktuellen Werte der Farbspeicher. RCLR gibt einen Wert zwischen 1 und 16 zurück, entsprechend den Farbwerten (1=schwarz usw.).

Folgende Werte für X sind zulässig:

0	40-Zeichen Hintergrundfarbe
1	Grafik Zeichenfarbe
2	Grafik Multicolor Zeichenfarbe 1
3	Grafik Multicolor Zeichenfarbe 2
4	40-Zeichen Rahmenfarbe
5	40- oder 80-Zeichen Schriftfarbe
6	80-Zeichen Hintergrundfarbe

Diese Funktion dient also zur Überprüfung der eingestellten Farbwerte in den verschiedenen Text- bzw. Grafikmodi. Sie brauchen sich damit keine Speicherstellen zu merken, um evt. mit PEEK nachzuschauen welche Farbe Sie denn z.B. im Grafikmodus 1 verwendet haben. Unabhängig vom jeweiligen Modus können Sie jederzeit die restlichen Modi mit RCLR(X) überprüfen.

RDOT

Kategorie: Funktionen

Schreibweise: RDOT (X)

Funktion: Diese Funktion ermittelt die Position des unsichtbaren Grafikcursors sowie den aktuellen Farbspeicher an der Stelle des Cursors.

`RDOT (Ø)` ergibt die X-Position des Grafikcursors.

`RDOT (1)` ergibt die Y-Position des Grafikcursors.

`RDOT (2)` ermittelt den aktuellen Farbspeicher.
(0 Hintergrundfarbe=kein Punkt gesetzt)
(1 Vordergrundfarbe=Punkt gesetzt)

Beispiel:

```
1Ø GRAPHIC 1,1 : REM GRAFIKMODUS 1 UND SCNCLR
2Ø GRAPHIC Ø
3Ø FOR I=Ø TO 2
4Ø PRINT RDOT(I),
5Ø NEXT I
6Ø GRAPHIC 1:LOCATE 15Ø,12Ø:GRAPHIC Ø
7Ø FOR I=Ø TO 2
8Ø PRINT RDOT(I),
9Ø NEXT I
```

RENUMBER

Kategorie: Befehle

Schreibweise: RENUMBER (neue Startzeile, Inkrement, alte Startzeile)

Funktion: Mit den angegebenen Parametern werden die Programmzeilen neu durchnumeriert. Das Inkrement bestimmt den Abstand zwischen den Programmzeilen. Der Vorgabewert ist 10.

Beispiel:

```
RENUMBER
```

numeriert ein Programm neu ab der ersten Zeile, beginnend mit 10 in Zehnerschritten.

```
RENUMBER 1000,20,1
```

1 bedeutet die ursprüngliche Startzeile des Programms. Das Programm beginnt jetzt mit Zeile 1000. Der Abstand der Programmzeilen beträgt 20.

Wollen Sie ein Programm ab Zeile 120 in 10 Zehnerschritten neu numerieren, so müssen Sie folgende Befehlszeile verwenden:

```
RENUMBER 120,10,120
```

RESUME

Kategorie: Anweisungen

Schreibweise: RESUME (Zeilennr. / NEXT)

Funktion: RESUME wird in Verbindung mit TRAP benutzt, um das Programm fortführen zu können. Wird RESUME ohne Zeilennummer oder NEXT benutzt, so versucht das Programm erneut in der Zeile fortzufahren, in der der ERROR verursacht wurde. Das kann u.U. zu einer Endlosschleife führen.

RESUME Zeilennr. gibt die Zeilennummer an, in der das Programm fortfahren soll, nachdem ein ERROR aufgetreten ist.

Mit RESUME NEXT fährt das Programm mit dem Befehl fort, der der Stelle folgt, die den ERROR ausgelöst hat.

Beispiel:

```
10 TRAP 1000
20 PRINT "IN DER ZEILE 30 IST EIN SYNTAX ERROR."
30 PRINT:PRIEMT
40 PRINT "DURCH RESUME NEXT WURDE IN ZEILE 40 ";
50 PRINT "DAS PROGRAMM FORTGEFÜHRT."
60 END
1000 RESUME NEXT
```

RGR

Kategorie: Funktionen

Schreibweise: RGR (A)

Funktion: Diese Funktion gibt den aktuellen Grafikmodus wieder, d.h. es werden Werte zwischen 0 und 5 zurückgegeben.

Beispiel:

```
PRINT RGR(A)
```

Ausgabe: 5

Grafikmodus 5 = 80-Zeichen-Textmodus

Mit dieser Funktion besteht die Möglichkeit, innerhalb eines Programms festzustellen, welcher aktuelle Grafikmodus angewählt ist. Dies ist z.B. sehr nützlich für Programme, die zwischen der 80-Zeichen- und der 40-Zeichendarstellung hin- und herschalten. Damit kann vom Programm überprüft werden, welcher Bildschirm gerade benutzt wird. Der Anwender braucht also dann dem Programm nicht extra mitzuteilen, in welchem Modus zur Zeit gearbeitet wird.

Anmerkung: Bei der Variablen A von RGR(A) handelt es sich um eine sogenannte Scheinvariable, d.h. der aktuelle Inhalt der Variablen A wird nicht beeinflußt.

RREG

Kategorie: Anweisungen

Schreibweise: RREG (a,x,y,s)

Funktion: Nach einem Aufruf mit SYS können durch diese Anweisung den Variablen A, X, Y und S die Inhalte des Akkus, des X-Registers, des Y-Registers und des Status-Registers übergeben werden.

Geben Sie z.B. SYS 828 ein, dann springt das System automatisch in den Monitor und meldet ein BREAK. Der Monitor zeigt Ihnen nun die einzelnen Inhalte der Register in hexadezimaler Schreibweise.

```
SR AC XR YR
33 FF ØØ ØØ
```

Verlassen Sie nun den Monitor durch Eingabe von X.
Jetzt geben Sie RREG A,X,Y,S ein.

Danach können Sie sich durch PRINT die Werte in dezimaler Form ansehen.

```
PRINT A,X,Y,S
```

Ausgabe: 255 0 0 51

RWINDOW

Kategorie: Anweisungen

Schreibweise: RWINDOW (X)

Funktion: Durch Eingabe von RWINDOW können die Spalten- und Zeilenzahl eines Fensters sowie der Zeichenmodus (40 oder 80) bestimmt werden. X hat dabei folgende Bedeutung:

0	ergibt Anzahl der Zeilen des Fensters.
1	ergibt Anzahl der Spalten des Fensters.
2	ergibt 40 oder 80, je nach Art des Zeichenmodus'.

Beispiel:

```
PRINT RWINDOW (0)
```

Ausgabe: 24 (24 Zeilen)

Durch diese Anweisung kann die Größe eines Fensters ermittelt werden, welches zuvor mit dem WINDOW-Befehl definiert wurde. Das Programm bzw. der Programmierer kann also dadurch dementsprechend seine Bildschirmausgaben für ein Fenster formatieren bzw. anpassen.

SLEEP

Kategorie: Befehle

Schreibweise: SLEEP x

Funktion: Mit SLEEP ist es möglich durch Angabe einer Zahl x (1<= x <=65536) eine einfache Zeitschleife aufzubauen. X gibt dabei die Dauer in Sekunden an.

Beispiel:

```
SLEEP 10
```

Führt eine Zeitschleife von 10 Sekunden Dauer aus.

In den BASIC-Versionen 2.0 bzw. 4.0 mußte man, um eine Verzögerung innerhalb eines Programms zu erreichen (z.B. um den Anwendern genügend Zeit zu geben, eine Textausgabe zu lesen), eine FOR...NEXT-Schleife aufbauen.

```
11Ø FOR I=Ø TO 1ØØØØ:NEXT I : REM ZEITSCHLEIFE 1Ø SEC.
```

Mit dem SLEEP-Befehl wird hier eine komfortablere Lösung angeboten.

STASH

Kategorie: Anweisungen

Schreibweise: STASH a,b,c,d

Funktion: Mit STASH können komplette Speicherbereiche aus dem BASIC-Arbeitsspeicher (Bank 1) in externen Speicherbänken abgespeichert werden. Dadurch ist es z.B. möglich, Variableninhalte in eine andere Bank wegzuschreiben, also zwischenzuspeichern, um diese bei Bedarf wieder in den Arbeitsspeicher zurückzuholen (s.a FETCH). Es kann mit STASH natürlich nur in freie RAM-Bereiche geschrieben bzw.

gespeichert werden. Bei der Verwendung des STASH-Befehls müssen Sie natürlich aufpassen, daß Sie nicht versehentlich den Variablen- oder sogar Programmbereich überschreiben.

Die Parameter a, b, c geben die Anzahl sowie Start- und Zieladresse an. Parameter d bestimmt die Bank, in die geschrieben werden soll. Für a,b,c gilt der Maximalwert von 65535 und für d dürfen Werte zwischen 1 und 15 benutzt werden.

Beispiel:

```
STASH 1000,16831,16831,4
```

Diese Befehlszeile schreibt 1000 Bytes von Adresse 16831 in Bank 1 nach Bank 4 an die gleiche Speicheradresse.

SWAP

Kategorie: Anweisungen

Schreibweise: SWAP a,b,c,d

Funktion: Mit SWAP können komplette Speicherbereiche zwischen dem BASIC-Arbeitsspeicher (Bank 1) und anderen Speicherbänken ausgetauscht werden. Dadurch ist es z.B. möglich, Variableninhalte zwischen zwei Bänken auszutauschen, die vielleicht vorher mit STASH in eine andere Bank zwischengespeichert wurden. Auch bei SWAP gilt natürlich, daß dieser Austausch nur in freien RAM-Bereichen stattfinden kann. Bei der Verwendung des SWAP-Befehls müssen Sie ebenfalls aufpassen, daß Sie nicht versehentlich den Variablen- oder sogar Programmbereich überschreiben.

Die Parameter a, b, c geben die Anzahl sowie Start- und Zieladresse an. Parameter d bestimmt die Bank, in der die Werte ausgetauscht werden sollen. Für a,b,c gilt der Maximalwert von 65535 und für d dürfen Werte zwischen 1 und 15 benutzt werden.

Beispiel:

```
SWAP 1000,16831,16831,4
```

Diese Befehlszeile tauscht 1000 Bytes ab Adresse 16831 in Bank 1 gegen 1000 Bytes in Bank 4 an der gleichen Speicheradresse aus.

SYS

Kategorie: Anweisungen

Schreibweise: SYS Adresse (,A,X,Y,S)

Funktion: Im BASIC 7.0 können mit SYS noch zusätzlich die Parameter für den Akku 'A', das X-Register 'X', das Y-Register 'Y' und das Statusregister 'S' übergeben werden.

Beispiel:

Geben Sie das folgende kleine Maschinenprogramm mit dem Monitor ab Adresse $04000 ein.

```
STA $D020
STX $D021
RTS
```

Verlassen Sie den Monitor und geben in den Rechner

```
BANK 4
```

ein. Mit dem folgenden SYS-Aufruf können Sie jetzt die Farbe für Bildschirmrahmen und Hintergrund setzen.

```
SYS 4*4096,0,1 (Rahmen schwarz, Hintergrund weiß)
```

TRAP

Kategorie: Anweisungen

Schreibweise: TRAP (Zeilennr.)

Funktion: Wird dieser Befehl zu Beginn eines Programms eingegeben, so kann auf jede Fehlermeldung, mit Ausnahme von "UNDEF'D STATEMENT ERROR", innerhalb des Programms reagiert werden. Tritt ein Fehler auf, so springt das Programm in die Zeilennummer, die bei TRAP angegeben wurde (s.RESUME). Die Zeilennummer, in der der Fehler auftrat, kann in der Systemvariablen 'EL' abgefragt werden; ebenso kann die Fehlermeldung in 'ERR$(ER)' ausgelesen werden.
Wurden Ihre Programme bisher durch nicht eingeschaltete Peripheriegeräte unsanft aus ihrem Lauf gerissen (DEVICE NOT PRESENT), so können Sie das jetzt durch eine entsprechende Fehlerroutine mit TRAP vermeiden.

Beispiel:

```
10 TRAP 100
20 DOPEN#1,"ADRESSEN"
30 INPUT#1,A$
40 DCLOSE#1
50 PRINT A$;
60 END
100 IF ER=5 AND EL=30 THEN PRINT "FLOPPY EINSCHALTEN UND TASTE
DRUECKEN."
110 GETKEY B$:RESUME
```

TRON

Kategorie: Befehle

Schreibweise: TRON

Funktion: Dieser Befehl schaltet den TRACE-Modus (TRaceON) ein. Dadurch wird beim Programmablauf die Zeilennummer angezeigt, die gerade vom Programm abgearbeitet wird.

TROFF

Kategorie: Anweisungen

Schreibweise: TROFF

Funktion: TROFF schaltet den TRACE-Modus wieder aus (TRaceOFF). Das Programm wird wieder normal ausgeführt, ohne die Zeilennummern anzuzeigen.

Mit diesen beiden Befehlen hat der Anwender die Möglichkeit, sein Programm im Ablauf genau zu verfolgen. Dies ist sehr wünschenswert, wenn z.B. nach Programmablauffehlern gefahndet werden muß. Man erkennt genau, welche Zeilennummer zu welchem Zeitpunkt abgearbeitet wird, und kann damit die Fehlerquelle schon sehr genau bestimmen.

WINDOW

Kategorie: Befehle

Schreibweise: WINDOW los,lor,urs,urr (,c)

Funktion: Dieser Befehl generiert ein Fenster auf dem aktuellen Bildschirm. Die Maximalwerte der Parameter werden vom jeweiligen Bildschirm bestimmt (40 oder 80 Zeichen).

los	linke obere Spalte
lor	linke obere Reihe
urs	untere rechte Spalte
urr	untere rechte Reihe

c	kann zusätzlich mit angegeben werden.
c	1 automatisches 'clear screen' im Fenster.

Beispiel:

```
10 WINDOW 30,15,39,24,1
```

Diese Programmzeile definiert ein Fenster im rechten unteren Bildschirmbereich. Solche Fenster werden oft benutzt, um Untermenüs in Programmen darzustellen.

Anmerkung: Die Definition des Fensters kann auch mit ESC T und ESC B über die Position des Cursors erfolgen.

XOR

Kategorie: Funktionen (logischer Operator)

Schreibweise: XOR (X,Y)

Funktion: Die Parameter X und Y werden durch diesen logischen Operator 'Exklusiv Oder' verknüpft. Es dürfen Werte von 0 bis 65535 verwendet werden.

Beispiel:

```
PRINT XOR(0,0),XOR(0,1),XOR(1,0),XOR(1,1)
```

Ausgabe: 0 1 1 0

```
PRINT XOR(23,56)
```

Ausgabe: 47

```
23 entspricht dual  Ø1Ø111
56 entspricht dual  111ØØØ   >  XOR = 1Ø1111 = 47
```

Dateiverwaltungsbefehle in BASIC 7.0

APPEND

Kategorie: Befehle

Schreibweise: APPEND#lfn,"Dateiname"(,Ddn on Ugn)

Funktion: Dieser Befehl öffnet das File "Dateiname" mit der logischen Filenummer 'lfn' und setzt den Datenzeiger an das Ende der Datei. Damit wird das Anhängen von Daten an eine sequentielle Datei ermöglicht. Bei Doppellaufwerken können zusätzlich noch die Parameter Ddn=Drivenummer und Ugn=Gerätenummer mit angegeben werden.

Beispiel:

```
10 DOPEN#1,"TEST 1",D0,W
20 FOR I=33 TO 65:PRINT#1,CHR$(I);NEXT I:DCLOSE#1
40 DOPEN#1,"TEST 1"
50 GET#1,A$:IF A$="" THEN 50
60 PRINT A$;IF ST<>64 THEN 50:ELSE DCLOSE#1:END
100 APPEND#1,"TEST 1",D0 ON U8
110 FOR I=66 TO 90
120 PRINT#1,CHR$(I);
130 NEXT I:DCLOSE#1:END
```

Dieses Programm schreibt in die Datei 'TEST 1' die CHR$-Codes von 33 bis 65. Danach wird die Datei ausgelesen und die Zeichen auf dem Bildschirm ausgegeben. Sie können nun mit 'RUN 100' die Datei um die CHR$-Codes 66-90 erweitern und danach mit 'RUN 60' erneut ausgeben lassen.

BACKUP

Kategorie: Befehle

Schreibweise: BACKUP Ddn TO Ddn(,ON Ugn)

Funktion: Dieser Befehl kopiert den Inhalt einer kompletten Diskette auf eine andere Diskette. Der BACKUP-Befehl funktioniert jedoch nur in Verbindung mit Doppellaufwerken.

Beispiel:

```
BACKUP D0 TO D1, ON U8
```

Dieses Beispiel kopiert den Inhalt der Diskette von Drive D0 auf die Diskette in Drive D1. Das Doppellaufwerk hat hierbei die Geräteadresse 8.
Der BACKUP-Befehl erstellt eine Kopie der Originaldiskette, d.h. daß evt. vorhandene Daten auf der Zieldiskette verloren gehen. Beim BACKUP-Vorgang wird die Zieldiskette gleichzeitig mit formatiert. Bereits existierende Dateien werden überschrieben.
Dieser Befehl sollte also nur dazu benutzt werden, um z.B. eine Sicherheitskopie einer Datendiskette zu erstellen. Er ist nicht anzuwenden, wenn nur wenige Dateien übertragen werden sollen. Hierzu dient der COPY-Befehl.

BLOAD

Kategorie: Befehle

Schreibweise: BLOAD"Dateiname"((ON),Bbnr,Psa)

Funktion: Dieser Befehl lädt ein Binärfile an eine durch 'Psa' näher bestimmte Startadresse. Mit 'Bbnr' wird die Bank bestimmt, in welche das Binärfile geladen werden soll.

Beispiel:

```
BLOAD "BPROG" ON B1,P4000
```

Lädt das Binärfile BPROG nach Bank 1 an die Startadresse 4000. Folgende Syntax ist ebenfalls zulässig:

```
BLOAD "BPROG" ,B1,P4000
```

Die folgende Befehlszeile lädt ein Binärfile, welches Spritedaten enthält, an die entsprechende Speicheradresse.

```
BLOAD "SPRITE" ,B0,P3584
```

BSAVE

Kategorie: Befehle

Schreibweise: BSAVE "Dateiname"(,Bbnr,Psta TO Pea)

Funktion: Speichert die Datei 'Dateiname' als Binärfile ab. Dabei bedeuten 'Bbnr'=Banknummer, Psta=Startadresse und Pea=Endadresse.

Beispiel:

```
BSAVE "MASCH I",B1,P 16384 TO P 17408
```

Speichert den Bereich von Adresse 16384 bis 17408 in Bank Nr. 1 unter dem Namen 'MASCH I' auf Diskette als Binärfile ab.

Sie haben durch diesen Befehl also die Möglichkeit, mit dem Monitor erstellte Maschinenprogramme abzuspeichern, um diese dann z.B. mit BOOT wieder zu laden und direkt ausführen zu lassen. Sie können aber auch die Daten eines Sprites mit BSAVE abspeichern, um sie zu einem späteren Zeitpunkt mit BLOAD wieder verfügbar zu machen.

Die folgende Befehlszeile gibt Ihnen ein Beispiel, wie Spritedaten in ein Binärfile abgespeichert werden können.

```
BSAVE "SPRITES" ,B0,P3584 TO P4096
```

CATALOG

Kategorie: Befehle

Schreibweise: CATALOG (Ddn,Ugn,Platzhalter)

Funktion: Gibt das Inhaltsverzeichnis einer Diskette auf dem Bildschirm aus. Wird bei Doppellaufwerken nur 'CATALOG' angegeben, so wird automatisch das Inhaltsverzeichnis von beiden Laufwerken angezeigt. Mit dem Parameter 'Platzhalter' können die auszugebenden Filenamen näher bestimmt werden.

Beispiel:

```
CATALOG D0,U8,"??S?H"
```

Zeigt alle Fileeinträge an, deren Namen aus fünf Buchstaben bestehen und die an dritter Stelle ein 'S' und an letzter Stelle ein 'H' aufweisen.

CATALOG "TES*" - zeigt alle Fileeinträge an, die mit 'TES' beginnen. Durch diesen Befehl können Sie sich das Inhaltsverzeichnis einer Diskette ausgeben lassen, ohne daß das im Speicher befindliche Programm überschrieben wird. Das war bisher durch die Befehlsfolge

```
LOAD"$",8
```

nicht möglich, da dabei grundsätzlich das Programm überschrieben wurde.

COLLECT

Kategorie: Befehle

Schreibweise: COLLECT (Ddn ON Ugn)

Funktion: Dieser Befehl gibt alle als belegt gekennzeichneten Blöcke der Diskette, die nicht einer Datei zuzuordnen sind, wieder frei. Weiterhin werden nicht ordnungsgemäß geschlossene Dateien aus dem Inhaltsverzeichnis entfernt. COLLECT hat also die gleiche Funktion wie der VALIDATE-Befehl des BASIC 2.0.

Beispiel:

```
COLLECT D0 on U8
```

oder

```
COLLECT D0,U9
```

oder

```
COLLECT
```

Auf diese einfache Art können Sie jetzt Ihre Datendiskette aufräumen. Sie brauchen dazu kein Programm der folgenden Art mehr zu schreiben:

```
10 OPEN 15,8,15:PRINT#15,"V":CLOSE 15
```

CONCAT

Kategorie: Befehle

Schreibweise: CONCAT (Ddn,) "Quelldatei" TO (Ddn,) "Zieldatei" (ON Ugn)

Funktion: Dieser Befehl verkettet sequentielle Files, indem die Daten der 'Quelldatei' an die 'Zieldatei' angehängt werden.

Beispiel:

```
CONCAT D0,"DATEN1" TO D0,"DATEN2" ON U8
```

Hängt an die Datei 'DATEN2' die Daten der Datei 'DATEN1' an.

```
10 DOPEN#1,"DATEN1,W"
20 PRINT#1,"0123456789"
30 DCLOSE#1
40 DOPEN#1,"DATEN2,W"
50 PRINT#1,"987654321"
60 DCLOSE#1
70 CONCAT "DATEN1" TO "DATEN2"
```

Die Datei 'DATEN2' beinhaltet jetzt die Daten

'9876543210123456789'

COPY

Kategorie: Befehle

Schreibweise: COPY (Ddn,) "Quelldatei" TO (Ddn,) "Neuedatei" (ON Ugn)

Funktion: Mit diesem Befehl können Dateien von einem Laufwerk auf das andere kopiert werden. Ebenfalls kann auf derselben Diskette eine Kopie einer Datei unter einem anderen Dateinamen erstellt werden.

Beispiel:

```
COPY D0,"CON" TO D0,"CONBACKUP" ON U8
```

erstellt auf derselben Diskette eine Kopie der Datei 'CON' mit dem neuen Namen 'CONBACKUP'.

```
COPY D0,"CON" TO D1,"TEST"
```

kopiert die Datei 'CON' von Laufwerk 1 unter Umbennung in 'TEST' nach Laufwerk 2. (Doppellaufwerk)

Den COPY-Befehl können Sie einsetzen, wenn nur eine oder aber wenige Dateien zu übertragen sind. Die Dateien der Zieldiskette bleiben erhalten und werden nicht überschrieben. Es muß aber darauf geachtet werden, daß eine zu übertragende Datei nicht bereits unter dem gleichen Namen auf der Zieldiskette vorhanden ist.

DCLEAR

Kategorie: Anweisungen

Schreibweise: DCLEAR (Ddn ON Ugn)

Funktion: Mit DCLEAR werden alle geöffneten Kanäle zum Diskettenlaufwerk zurückgesetzt.

Vorsicht! Erfolgt ein DCLEAR, bevor eine Datei mit DCLOSE# ordnungsgemäß geschlossen wurde, so bleibt die Datei zum Schreiben geöffnet! Diese Datei kann dann auf normalem Wege nicht mehr gelesen oder beschrieben werden.

Beispiel:

```
10 DOPEN#1,"ADRESSEN",W
20 PRINT#1,"ADRESSE 1"
30 PRINT#1,"ADRESSE 2"
40 DCLOSE#1
50 DCLEAR
```

Dieses Beispiel zeigt die richtige Vorgehensweise. Zuerst wird die geöffnete Datei mit DCLOSE geschlossen und erst danach erfolgt das DCLEAR.

DCLOSE

Kategorie: Anweisungen

Schreibweise: DCLOSE (#lfn) (ON Ugn)

Funktion: Mit DCLOSE werden eine Datei oder alle momentan geöffneten Dateien geschlossen. (#lfn) bezeichnet hier wieder die logische Filenummer und (Ugn) die Gerätenummer.

Beispiel:

```
DCLOSE#1
```

Schließt die Datei, die zuvor mit DOPEN#1 geöffnet wurde.

DCLOSE - schließt alle geöffneten Dateien.

```
10 DOPEN#1,"ADRESSEN",W
20 DOPEN#2,"ADRESSEN-GE"
30 PRINT#1,"ADRESSE 1"
40 PRINT#1,"ADRESSE 2"
50 INPUT#2,AD$
60 DCLOSE#1:DCLOSE#2
70 DCLEAR
```

In diesem Beispiel hätte in Zeile 60 nur ein DCLOSE die gleiche Wirkung erzielt.

DIRECTORY

Kategorie: Befehle

Schreibweise: DIRECTORY (Ddn, Ugn,"Dateiname")

Funktion: Zeigt das Inhaltsverzeichnis einer Diskette auf dem Bildschirm an. Dieser Befehl gleicht dem 'CATALOG' Befehl. Beim 'Dateinamen' können ebenfalls die bekannten Platzhalter (?, *) verwendet werden.

Beispiel:

`DIRECTORY`	zeigt das komplette Inhaltsverzeichnis.
`DIRECTORY DØ,"?S*"`	zeigt alle Einträge an, die an zweiter Stelle ein 'S' aufweisen.
`DIRECTORY D1,"TEXT*"`	zeigt alle Einträge an, die mit 'TEXT' beginnen.

Durch diesen Befehl können Sie sich ebenfalls das Inhaltsverzeichnis einer Diskette ausgeben lassen, ohne daß das im Speicher befindliche Programm überschrieben wird. Bisher waren Sie gezwungen durch die Befehlsfolge

```
LOAD"$",8
```

sich den Inhalt der Diskette anzuschauen. Dabei wurde aber jedesmal das Programm überschrieben.

DLOAD

Kategorie: Befehle

Schreibweise: DLOAD"Dateiname" (,Ddn,Ugn)

Funktion: Dieser Befehl lädt ein Programm von Diskette in den Speicher des Rechners. DLOAD kann auch innerhalb eines Programms verwendet werden.

Beispiel:

```
DLOAD "GRAFIK"
```

lädt das Programm 'GRAFIK' in den Rechner.

oder

```
DLOAD (A$)
```

lädt das Programm in den Rechner, dessen Name in A$ steht.

```
10 INPUT "PROGRAMMNAME";A$
20 DLOAD (A$)
```

DOPEN

Kategorie: Anweisungen

Schreibweise: DOPEN#lfn,"Dateiname"(,Ddn,Ugn)(,Lx/W)

Funktion: Eröffnet eine Datei zum Lesen oder Schreiben. Soll eine relative Datei generiert werden, wird die Angabe L(Recordlänge) benutzt. Für jede andere Datei, die zum Schreiben geöffnet werden soll, tritt an Stelle von 'L' der Parameter 'W'. (lfn=logische Filenummer, Ddn=Laufwerksnummer und Ugn=Gerätenummer) Wird eine Datei zum Lesen geöffnet, so entfällt der Parameter 'W'.

Beispiel:

```
DOPEN#1,"ADRESSEN",W
```

öffnet die Datei 'ADRESSEN' zum Schreiben.

```
DOPEN#1,"TEST",L15,D1,U9
```

öffnet auf Gerät 9, Laufwerk 1 eine relative Datei mit der Recordlänge 15 zum Schreiben.

```
DOPEN#1,"ADRESSEN"
```

öffnet die Datei 'ADRESSEN' zum Lesen.

DSAVE

Kategorie: Befehle

Schreibweise: DSAVE "Dateiname" (,Ddn,Ugn)

Funktion: Dieser Befehl speichert ein Programm auf der Diskette ab. Den Befehl 'SAVE' sollten Sie benutzen, um Programme auf Kassette abzuspeichern.

Beispiel:

`DSAVE "LAGER"`	speichert das Programm 'LAGER' auf Diskette.
`DSAVE "LAGER",D1,U9`	speichert das Programm 'LAGER' auf Diskette (Laufwerk 1 und Gerät 9.)

'LAGER' kann hier ebenfalls einer Stringvariablen übergeben werden, so daß die folgende Syntax auch zulässig ist:

```
A$="LAGER"
DSAVE (A$)
```

Mit DSAVE ersparen Sie sich damit die zusätzliche Angabe des Laufwerks (SAVE"TEST",8).

DVERIFY

Kategorie: Befehle

Schreibweise: DVERIFY (Ddn,Ugn)

Funktion: DVERIFY vergleicht das Programm im Speicher mit dem Programm auf Diskette. Wurde das Programm korrekt abgespeichert, wird die Meldung

OK

ausgegeben.

Stimmt das Programm auf der Diskette nicht hundertprozentig mit dem im Speicher befindlichen überein, so wird die Meldung

VERIFY ERROR

ausgegeben.

Die Programme, die auf dem C-64 abgespeichert wurden, lassen sich - sofern es sich um reine BASIC-Programme handelt - ohne weiteres in den C-128 laden. Geben Sie nun den Befehl DVERIFY ein, so bekommen Sie selbstverständlich einen Error angezeigt, da z.B. schon die abgespeicherte Startadresse nicht mit der Startadresse des C-128 übereinstimmt. Wollen Sie dies vermeiden, so müssen Sie das Programm noch einmal im Commodore 128 Modus abspeichern. Danach erhalten Sie auch mit DVERIFY ein OK.

HEADER

Kategorie: Befehle

Schreibweise: HEADER "Diskettenname"(,Iid,Ddn,Ugn)

Funktion: Durch diesen Befehl wird eine Diskette neu formatiert. Alle Daten, die sich evt. noch auf dieser Diskette befinden, gehen durch diesen Vorgang verloren. Iid steht für eine aus zwei Zeichen bestehende ID-Nummer.
Wurde eine Diskette bereits formatiert, so kann man die ID-Kennung auslassen. Die Diskette erhält dann nur einen neuen Namen. Der Diskettenname darf aus bis zu 16 Zeichen bestehen.

Beispiel:

```
HEADER "DATENDISK",I85,D0
```

Die Diskette erhält den Namen 'Datendisk' und die ID=85. Dieser Vorgang dauert ca. 80 Sekunden.

```
HEADER "DATENDISK 1",D0
```

Die Diskette erhält nur den neuen Namen 'Datendisk 1'. (schnelle Formatierung). Dieser Vorgang ist nur dann möglich, wenn die Diskette mindestens einmal vorher mit der ID-Kennung formatiert wurde. Die Formatierung ohne Angabe der ID dauert ca. 2 Sekunden.

RECORD

Kategorie: Anweisungen

Schreibweise: RECORD#lfn,dnr (,bnr)

Funktion: Setzt den Zeiger einer relativen Datei auf jeden beliebigen Datensatz (dnr=Datensatznummer) und jedes beliebige Byte eines Datensatzes (bnr=Bytenummer).

Beispiel:

```
10 DOPEN#1,"ADRESSEN"
20 RECORD#1,5,10
30 INPUT#1,A$:DCLOSE#1
40 PRINT A$;:END
```

Dieses Programm öffnet die relative Datei 'Adressen' zum Lesen. Danach wird mit RECORD der Dateizeiger auf den fünften Datensatz und das 10. Byte gesetzt. Anschließend wird mit INPUT# der entsprechende Datensatz gelesen und ausgegeben.
Der RECORD-Befehl erleichtert das Einlesen von Daten aus einer relativen Datei ungemein. Sie brauchen als zusätzlichen Parameter eigentlich nur noch die Recordnummer anzugeben. Wollten Sie dagegen beim C-64 diesen Zeiger auf einen bestimmten Record setzen, war dazu eine Menge Vorarbeit zu leisten. Eine typische Anweisung zur Positionierung sah z.B. so aus:

```
PRINT#1,"P"+CHR$(2)+CHR$(10)+CHR$(1)+CHR$(5)
```

RENAME

Kategorie: Befehle

Schreibweise: RENAME "alter Name" TO "neuer Name" (,Ddn,Ugn)

Funktion: Mit diesem Befehl kann eine Datei umbenannt werden.

Beispiel:

```
RENAME "ADRESSEN" TO "ANSCHRIFTEN",D0
```

Die Datei 'Adressen' erhält den neuen Namen 'Anschriften'. Hierbei muß beachtet werden, daß eine Datei keinen Namen erhält, der bereits von einer anderen Datei benutzt wird.

Dieser Befehl erspart Ihnen ebenfalls das sonst notwendige Öffnen des Befehlskanals, über den dann der entsprechende Befehl zur Umbenennung gelangt, wie es das folgende Beispiel noch einmal zeigt.

```
10 OPEN 1,8,15,"R:ANSCHRIFT=ADRESSEN":CLOSE 1
```

Gibt der Datei ADRESSEN den neuen Namen ANSCHRIFTEN.

SCRATCH

Kategorie: Befehle

Schreibweise: SCRATCH "Dateiname" (,Ddn,Ugn)

Funktion: Dieser Befehl löscht eine Datei im Inhaltsverzeichnis der Diskette. Vorher fragt der Rechner

```
ARE YOU SURE?
```

(Sind Sie sicher?). Geben Sie jetzt 'Y' ein, so wird die angegebene Datei gelöscht.

Beispiel:

```
SCRATCH "ADRESSEN"
ARE YOU SURE?Y
```

Löscht die Datei 'Adressen'.

Vorsicht! Verwenden Sie beim SCRATCH-Befehl nur den Stern, so werden alle auf der Diskette befindlichen Dateieinträge gelöscht.

```
SCRATCH "*"
```

löscht alle Dateieinträge!

Befehle für Sprites und Shapes

Der wesentliche Unterschied zwischen Sprites und Shapes besteht darin, daß Sprites vom VIC Chip gesondert behandelt werden. Dadurch können Sprites schneller über den Bildschirm bewegt werden als Shapes. Ferner ist es möglich bei den Sprites festzustellen, ob eine Kollision mit dem Hintergrund oder einem Sprite stattgefunden hat.

BUMP

Kategorie: Funktionen

Schreibweise: BUMP (X)

Funktion: Mit BUMP kann bestimmt werden, welche Spritekollisionen seit der letzten Abfrage stattgefunden haben. Dabei ergibt BUMP(1) die Information darüber, welche Sprites miteinander kollidiert sind; mit BUMP(2) kann festgestellt werden, welche Sprites mit dem Hintergrund kollidiert sind.

Bei den Werten, die BUMP liefert, müssen die Bitpositionen ausgewertet werden. Erhält man also für BUMP(1) den Wert 129, so hat eine Kollision zwischen Sprite 1 und Sprite 8 stattgefunden.

Das gleiche gilt für BUMP(2).

Erhält man den Wert 15, so hatten die Sprites 1, 2 und 3 eine Kollision mit einem Hintergrundzeichen.

COLLISION

Kategorie: Anweisungen

Schreibweise: COLLISION A(,zn)

Funktion: Mit dieser Anweisung kann man das Programm entsprechend auf eine Sprite-Sprite- oder Sprite-Hintergrundkollision reagieren lassen. Ferner besteht ebenfalls die Möglichkeit abzufragen, ob ein Lightpen aktiv ist. Findet nun eine solche Kollision statt, so springt das Programm nach der Anweisung COLLISION in ein Unterprogramm. Dieses Unterprogramm muß wie jedes andere Unterprogramm mit einem RETURN enden. Die Zeilennummer des Unterprogramms wird durch den Parameter 'zn' bestimmt. Der Parameter A kann folgende Werte annehmen:

1 für Sprite/Sprite Kollision
2 für Sprite/Hintergrund Kollision
3 für Lightpen

Beispiel:

```
90  ...
100 COLLISION 1,1000
 .
 .
 .
1000 SOUND 3,965,60
1010 RETURN
```

Findet in diesem Beispiel eine Sprite/Sprite Kollision statt, so verzweigt das Programm in Zeile 100 in das Unterprogramm ab Zeile 1000, erzeugt dort ein entsprechendes Geräusch und kehrt aus dem Unterprogramm wieder zurück.

MOVSPR

Kategorie: Anweisungen

Schreibweise: MOVSPR s,x,y oder MOVSPR s,w#v

Funktion: Mit MOVSPR kann zum einen ein Sprite durch Angabe der X- und Y-Position an einen bestimmten Punkt des Bildschirms bewegt werden. Zum zweiten kann eine kontinuierliche Bewegung unter Angabe des Winkels und der Geschwindigkeit eines Sprites definiert werden.

Beispiel:

```
MOVSPR 1,120,90
```

positioniert Sprite 1 nach x=120 und y=90.

```
MOVSPR 3,90#5
```

definiert für Sprite 3 eine kontinuierliche Bewegung mit der Geschwindigkeit 5 unter einem Winkel von 90 Grad, also quer über den Bildschirm.

Dieser Befehl gibt Ihnen die Möglichkeit, auf einfache Weise Sprites auf dem Bildschirm zu positionieren oder Sprites unter Angabe des Winkels eine kontinuierliche Geschwindigkeit zu geben. Dies ist im C-64 Modus nur über mehrere POKE-Befehle zu erreichen.

RSPCOLOR

Kategorie: Funktionen

Schreibweise: RSPCOLOR (X)

Funktion: Diese Funktion überprüft, welche Multicolor-Werte bei den Sprites zuletzt gültig waren. Sie erhalten mit dieser Funktion die Multicolorwerte zurück, die über den SPRCOLOR-Befehl definiert wurden. RSPCOLOR(1) gibt Ihnen den ersten Parameter und RSPCOLOR (2) gibt Ihnen den zweiten Parameter an, der über SPRCOLOR a,b gesetzt wurde.

Beispiel:

```
PRINT RSPCOLOR(1)
```

Ausgabe: 2

Definieren Sie also vorher die Multicolor-Werte der Sprites mit SPRCOLOR 2,12 können Sie diese später in Ihrem Programm mit

```
PRINT RSPCOLOR(1),RSPCOLOR(2)
```

ausgeben lassen oder durch die folgende Programmzeile den Variablen A% und B% zuordnen:

```
100 A%=RSPCOLOR(1):B%=RSPCOLOR(2)
```

RSPPOS

Kategorie: Funktionen

Schreibweise: RSPPOS (Spritenr.,X)

Funktion: Mit dieser Funktion kann man von einem bestimmten Sprite die X-Position, Y-Position oder die Geschwindigkeit abfragen. Dabei kann X folgende Werte annehmen:

0	aktuelle X-Position
1	aktuelle Y-Position
2	Geschwindigkeit (0-15)

Beispiel:

```
PRINT RSPPOS (1,0)
```

bestimmt von Sprite 1 die X-Position.

```
PRINT RSPPOS (3,1)
```

bestimmt von Sprite 3 die Y-Position.

```
PRINT RSPPOS (2,2)
```

bestimmt von Sprite 2 die Geschwindigkeit, welche mit MOVSPR definiert wurde.

RSPRITE

Kategorie: Funktionen

Schreibweise: RSPRITE (A,X)

Funktion: Mit RSPRITE lassen sich verschiedene Parameter eines Sprites bestimmen bzw. abfragen. Für A muß die Nummer des Sprites angegeben werden und für X einer der nachfolgenden Werte:

```
Ø    Sprite ein- oder ausgeschaltet
1    Spritefarbe
2    Hintergrundpriorität (ja=1/nein=Ø)
3    vergrößert in X-Richtung (ja=1/nein=Ø)
4    vergrößert in Y-Richtung (ja=1/nein=Ø)
5    Sprite in Multicolor (ja=1/nein=Ø)
```

Beispiel:

```
PRINT RSPRITE (2,1)
```

Ausgabe: 1 (Sprite 2 hat die Farbe 1=schwarz)

```
PRINT RSPRITE (5,5)
```

Ausgabe: 0 (Sprite 5 Multicolor nicht eingeschaltet)

SPRCOLOR

Kategorie: Befehle

Schreibweise: SPRCOLOR a,b

Funktion: Dieser Befehl setzt die Multicolorwerte für alle Sprites. Der Parameter A bestimmt den Multicolorwert 1 und B den Multicolorwert 2.

Der Wert von A kann mit RSPCOLOR(1) und der Wert von B mit RSPCOLOR(2) wieder abgefragt werden (s.a. RSPCOLOR).

Beispiel:

```
10 SPRITE 1,1,,,,,1
20 SPRCOLOR 3,1
30 MOVSPR 1,145#8
```

Dieses Programm schaltet Sprite Nummer 1 ein und setzt es in den Multicolormodus. Zeile 20 definiert für den Multicolorwert 1 die Farbe Rot und für den zweiten Multicolorwert die Farbe Schwarz. Zeile 30 schließlich setzt Sprite 1 unter einem Winkel von 145 Grad mit der Geschwindigkeitsstufe 8 in Bewegung.

SPRITE

Kategorie: Anweisungen

Schreibweise: SPRITE nr,a,f,p,x,y,m

Funktion: Diese Anweisung bestimmt die Parameter eines Sprites. nr' steht für die Nummer des Sprites. a entscheidet, ob der Sprite eingeschaltet (a=1) oder ausgeschaltet (a=0) ist. f bestimmt die Farbe (1-16) und p entscheidet über die Hintergrundpriorität. x,y bestimmen, ob der Sprite in X- oder Y-Richtung vergrößert dargestellt wird. Der Parameter m entscheidet schließlich über den Multicolormodus.

nr	Spritenummer
a	ein- (a=1) oder ausgeschaltet (a=0)
f	Farbe (1-16)
p	Vordergrund- (p=0) oder Hintergrundpriorität (p=1)
x	vergrößert in X-Richtung (x=1) / normal (x=0)
y	vergrößert in Y-Richtung (y=1) / normal (y=0)
m	Multicolor ein (m=1) / Multicolor aus (m=0)

Beispiel:

```
100 SPRITE 1,1,1,0,1,1,0
110 SPRITE 2,1,3,1
120 SPRITE 4,0
```

Dieses Beispiel schaltet die Sprites 1 und 2 ein sowie Sprite 4 aus und setzt die genannten Parameter der einzelnen Sprites.

SPRSAV

Kategorie: Anweisungen

Schreibweise: SPRSAV A$,1

Funktion: Speichert eine Stringvariable in einen Spritespeicherbereich. Dieser Befehl kommt hauptsächlich zum Einsatz, wenn vorher durch SSHAPE einer Stringvariablen Grafikinformationen übergeben wurden. Möchte man diese Grafik jetzt als Sprite verwenden, so besteht die Möglichkeit, mit SPRSAV diese Information in den entsprechenden Speicherbereichen abzulegen.

Andererseits haben Sie mit SPRSAV 1,A$ die Möglichkeit, die Spriteinformation - hier von Sprite 1 - in der Variablen A$ abzulegen und damit A$ als Shape zu behandeln.

Beispiel:

```
SPRSAV X$,1
```

Der Inhalt der Stringvariablen X$ wird im Speicherbereich von Sprite 1 abgelegt.

```
SPRSAV 1,A$
```

Die Spriteinformation von Sprite 1 wird in der Variablen A$ abgespeichert.

SSHAPE/GSHAPE

Kategorie: Anweisungen

Schreibweise: SSHAPE A$,x1,y1,x2,y2

Funktion: SSHAPE ordnet eine Teilgrafik einer Stringvariablen zu. Da eine Stringvariable nur 255 Zeichen beinhalten kann, ist die Größe der Grafik beschränkt. Beinhalten die Eckkoordinaten einen zu großen Grafikbereich, so wird die Fehlermeldung 'STRING TOO LONG ERROR' ausgegeben.
GSHAPE kann die einer Variablen zugeordnete Grafikinformation wieder an jeder beliebigen Stelle des Grafikbildschirms ausgeben.

Beispiel:

```
SSHAPE SP$,10,10,30,30
```

ordnet der Variablen SP$ den entsprechenden Grafikbereich zu.

```
GSHAPE SP$,100,120
```

gibt die Grafikinformation von SP$ an der Stelle 100,120 auf dem Grafikbildschirm wieder aus.

```
5Ø GRAPHIC1,1:FOR I=Ø TO 32Ø STEP 2:GSHAPE SP$,I,1ØØ:NEXT
```

Diese Programmzeile bewegt den Shape SP$ quer über den Grafikbildschirm.

SPRDEF

Kategorie: Befehle

Schreibweise: SPRDEF

Funktion: Der SPRDEF Befehl schaltet um in den Spriteeditor. Dadurch steht dem Anwender ein Mittel zur Verfügung, mit dem er auf komfortable Weise Sprites erstellen kann.
In einem Feld von 24 mal 21 Zeichen kann der Anwender mit dem Cursor die Form der Sprites bestimmen. Die Daten der Sprites können sofort weiterverarbeitet werden, sei es auf Diskette oder sofort innerhalb eines Programms. Der Spriteeditor stellt die folgenden Funktionen zur Verfügung:

CLR Taste	löscht kompletten Editierbereich
M Taste	schaltet Multicolor ein/aus
CTRL 1-8	bestimmt die Farben 1-8
C= 1-8	bestimmt die Farben 9-16
Taste 1	löscht bereits gesetzte Punkte
Taste 2	setzt Punkte
Taste 3	Funktion wie Taste 1 nur für Multicolor
Taste 4	Funktion wie Taste 2 nur für Multicolor
A Taste	schaltet automatische Cursorsteuerung an/aus
X Taste	vergrößert Sprite in X-Richtung
Y Taste	vergrößert Sprite in Y-Richtung
C Taste	kopiert z.B. Sprite 1 nach Sprite 2

Die Steuerung des Cursors (+) erfolgt über die CRSR-Tasten. Mit SHIFT/RETURN werden die Spriteinformationen in dem entsprechenden Speicherbereich für die Sprites gespeichert.

Grafikbefehle

BOX

Kategorie: Anweisungen

Schreibweise: BOX (fs,x1,y1,x2,y2,w,a)

Funktion: Zeichnet ein beliebiges Rechteck an der angegebenen Position auf den Bildschirm. Dabei bedeuten die folgenden Parameter im einzelnen:

fs	Farbspeicher (0-3)
x1,y1	obere linke Ecke
x2,y2	untere rechte Ecke
w	Winkelangabe, um die das Rechteck gedreht werden soll.
a	Rechteck ausfüllen (0=nein/1=ja)

Beispiel:

```
10 SCNCLR
20 GRAPHIC1:Z=5
30 FOR I=0 TO 361 STEP 5
40 Z=Z+1
50 BOX 1,160+Z,100+Z,160-Z,100-Z,I
60 NEXT
70 END
```

CHAR

Kategorie: Anweisungen

Schreibweise: CHAR (fs,x,y,st,fl)

Funktion: Mit CHAR kann sowohl im Textmodus als auch im Grafikmodus ein Textstring unter Angabe von Zeilen- und Spaltennummer ausgegeben werden. Dadurch wird die Beschriftung von Grafiken erheblich vereinfacht. Die Parameter bedeuten im einzelnen:

fs	Farbspeicher (0-3)
x	Spaltennummer
y	Zeilennummer
st	auszugebender Text
fl	0=normale Darstellung/1=reverse Darstellung

Beispiel:

```
10 CHAR 1,18,12,"TESTAUSGABE"
20 SLEEP 1
30 CHAR 1,18,12,"TESTAUSGABE",1
40 SLEEP 1
50 GOTO 10
```

Diese Anweisung erleichtert die Beschriftung erstellter Grafiken ungemein. Diejenigen von Ihnen, die sich schon einmal an der Beschriftung von Grafiken auf dem C-64 versucht haben, werden diesen Befehl besonders zu schätzen wissen.

CIRCLE

Kategorie: Anweisungen

Schreibweise: CIRCLE (fs,x,y,xr,yr,sw,ew,w,i)

Funktion: Die CIRCLE-Anweisung ist wohl eine der vielseitigsten Grafikanweisungen im BASIC 7.0. Der Anwender kann mit ihr Kreise, Kreisbögen, Ellipsen, Quadrate usw. an beliebigen Stellen auf dem Bildschirm erstellen. Die Parameter haben dabei die folgende Bedeutung:

fs	Farbspeicher (0-3)
x,y	Koordinaten Kreismittelpunkt
xr	Radius in X-Richtung
yr	Radius in Y-Richtung (Vorgabewert ist xr)
sw	Anfangswinkel für Kreis
ew	Endwinkel für Kreis
w	Winkel für Rotation
i	Winkel für die zu zeichnenden Kreissegmente (Vorgabewert = 2)

Beispiel:

```
10 SCNCLR:GRAPHIC1:Z=5
20 FOR I=1 TO 361 STEP 5:Z=Z+1
30 CIRCLE ,160+Z,100+Z,160-Z,100-Z,I
40 NEXT
```

COLOR

Kategorie: Anweisungen

Schreibweise: COLOR fs,fw

Funktion: COLOR ordnet einem der sieben Farbspeicher fs einen mit fw bestimmten Farbwert zu. 'fw' kann die Werte von 1 bis 16 annehmen.

Beispiel:

```
COLOR 0,1
```

setzt den Hintergrund im 40-Zeichenmodus auf schwarz.

```
COLOR 6,8
```

setzt den Hintergrund im 80-Zeichenmodus auf gelb.

```
COLOR 1,2
```

setzt im Grafikmodus die Vordergrundfarbe auf weiß.
Die ersten Parameter fs haben hierbei die folgende Bedeutung:

0	Hintergrund 40 Zeichenmodus
1	Vordergrund Grafikmodus
2	Multicolor 1 Grafikmodus
3	Multicolor 2 Grafikmodus
4	Bildschirmrahmen 40 Zeichenmodus
5	Zeichenfarbe 40- oder 80-Zeichenmodus
6	Hintergrund 80 Zeichenmodus

DRAW

Kategorie: Anweisungen

Schreibweise: DRAW fs,x1,y1 TO x2,y2 TO x3,y3 ...

Funktion: Mit DRAW können beliebige Figuren durch Angabe von Start- und Endpunkten gezeichnet werden. Dabei kann es sich um Punkte, Linien oder beliebige Umrisse handeln. Mit fs wird wieder der entsprechende Farbspeicher angewählt, und die Parameter x,y bestimmen jeweils Start- und Endpunkt.

Beispiel:

```
DRAW 1,10,20 TO 300,20 TO 150,190 TO 10,20
```

Zeichnet ein Dreieck.

```
10 SCNCLR:GRAPHIC1
20 X=160:Y=100:R=50
30 FOR A=1 TO 360
40 DRAW 1,X+R*COS(A),Y+R*SIN(A)
50 NEXT A
60 END
```

Auf diese Art können Sie einen Kreis zeichnen lassen, ohne die CIRCLE Anweisung zu benutzen.

GRAPHIC

Kategorie: Anweisungen

Schreibweise: GRAPHIC m(,c,s)

Funktion: Mit GRAPHIC wird eine der sechs Grafikbetriebsarten angewählt. Der zweite Parameter 'c' bestimmt, ob beim Umschalten automatisch ein SCNCLR ausgeführt wird oder nicht. Hat c den Wert 1, so wird ein SCNCLR ausgeführt. Für c=0 erfolgt kein SCNCLR.

Der Parameter 's' bestimmt für den kombinierten Modus von Text und Grafik (GRAPHIC 2 oder GRAPHIC 4), in welcher Zeile der Textbildschirm beginnen soll. Der Vorgabewert liegt hier bei Zeile 19.

Im folgenden nun die sechs Grafikbetriebsarten:

0	40-Zeichen Textmodus
1	Grafikmodus
2	kombinierter Grafik- Textmodus
3	Multicolor Grafikmodus
4	kombinierter Multicolor Grafik- Textmodus
5	80-Zeichen Textmodus

Beispiel:

```
GRAPHIC 2,1,15
```

Diese Anweisung schaltet um in den kombinierten Grafik-Textmodus und löscht automatisch die Grafikseite. Der Textbildschirm beginnt ab Zeile 15.

LOCATE

Kategorie: Anweisungen

Schreibweise: LOCATE x,y

Funktion: Durch die Anweisung LOCATE kann der unsichtbare Grafikcursor an jede beliebige Stelle des Grafikbildschirms bewegt werden.

Beispiel:

```
LOCATE 160,100
```

Positioniert den unsichtbaren Grafikcursor etwa in die Mitte des Grafikbildschirms.

Mit der Funktion RDOT können Sie jederzeit die Position des Grafikcursors erfahren. Würden Sie also nach der oben gegebenen Anweisung folgende Befehlszeile in den Rechner eingeben

```
PRINT RDOT(0),RDOT(1)
```

so erhielten Sie als Ausgabe wieder die Werte 160 und 100. Mit diesen beiden Anweisungen haben Sie also jederzeit die vollständige Kontrolle über den nicht sichtbaren Grafikcursor.

PAINT

Kategorie: Anweisungen

Schreibweise: PAINT fs,x,y,m

Funktion: PAINT füllt eine Fläche, z.B. einen Kreis, mit einer Farbe aus. Die Koordinaten x,y geben den Punkt an, ab dem PAINT mit dem Ausfüllen beginnen soll. Ist die Fläche nicht vollständig umrahmt, besitzt sie also keine eindeutige Grenze, so wird der komplette Grafikbildschirm ausgefüllt.

Durch den Parameter m kann zusätzlich bestimmt werden, aus welchem Farbspeicher sich PAINT den Farbwert holt (Vorder- oder Hintergrundfarbe).

Beispiel:

```
10 SCNCLR:GRAPHIC 1
20 BOX,40,40,100,100
30 PAINT,50,50
```

In diesem Beispiel würde das entstandene Quadrat vollständig ausgefüllt werden. Hätten Sie im obigen Beispiel statt der BOX Anweisung die folgende CIRCLE Anweisung benutzt, um einen dreiviertel Kreis zu zeichnen

```
CIRCLE ,160,100,40,,0,270
```

und hätten anschließend die Anweisung PAINT,160,100 benutzt, so wäre die komplette Grafikseite ausgefüllt worden, da der Kreis keine geschlossene Grenze aufweist.

SCALE

Kategorie: Anweisungen

Schreibweise: SCALE n

Funktion: SCALE n wechselt zwischen zwei Skalierungen hin und her. Mit SCALE 1 hat der Grafikbildschirm horizontal und vertikal die Koordinaten von 0 bis 1023. Bevor Sie nun einen falschen Schluß ziehen, nein, Ihnen steht natürlich jetzt keine Auflösung von 1024 mal 1024 Punkten zur Verfügung. Es sind nach wie vor 320 mal 200 Punkte, nur daß die waagrechten Punkte nun eine Einheit von 3.2 und die senkrechten Punkte eine Einheit von 5.12 Punkten aufweisen. Es wurde also lediglich der Maßstab verändert.
Es existieren relativ viele Grafikprogramme, die für Computer geschrieben wurden, die eine entsprechende Skalierung des Grafikbildschirms von 1024 mal 1024 besitzen. Damit Sie auch diese Programme ohne große Schwierigkeiten auf dem C-128 angepaßt bekommen, wurde diese Anweisung implementiert. Es müssen somit nur die entsprechenden Grafikbefehle angepaßt werden. Durch Eingabe von SCALE 0 wird wieder auf die normale Skalierung von 319 mal 199 bzw. 159 mal 199 umgeschaltet.

Beispiel:

```
10 GRAPHIC 1,1
20 SCALE 0
30 CIRCLE ,160,100,60
40 SCALE 1
50 CIRCLE ,160,100,60
```

SCNCLR

Kategorie: Anweisungen

Schreibweise: SCNCLR n

Funktion: Mit SCNCLR wird die aktuelle Bildschirmseite gelöscht. Unter Angabe des Parameters 'n' wirkt SCNCLR auf die entsprechende Grafikseite (n=0 für 40-Zeichen-Textmodus, n=1 für Grafikmodus usw.)

WIDTH

Kategorie: Befehle

Schreibweise: WIDTH n

Funktion: Mit WIDTH kann die Zeichendichte der Grafik bestimmt werden. Für WIDTH 1 wird mit einfacher Dichte gezeichnet und für WIDTH 2 wird mit doppelter Dichte gezeichnet.

Beispiel:

```
10 SCNCLR:GRAPHIC1
20 BOX,20,20,60,60
30 WIDTH 2
40 BOX,60,60,110,110
50 END
```

Soundbefehle

ENVELOPE

Kategorie: Anweisungen

Schreibweise: ENVELOPE nr,a,d,s,r,w,p

Funktion: Mit diesem Befehl wählen Sie eine der vorgegebenen Hüllkurven des Commodore 128 aus. Diese Hüllkurve können Sie dann durch Angabe der einzelnen Parameter beliebig verändern. Ihnen stehen insgesamt 10 Hüllkurven zur Verfügung (0-9), die bereits bestimmte Musikinstrumente repräsentieren. Durch Veränderung der Parameter können diese Instrumente verfremdet werden, oder Sie können völlig neue Klangformen kreieren.

Die Parameter haben folgende Bedeutung:

nr	Nummer der Hüllkurve (0-9)
a	Attack (0-15)
d	Decay (0-15)
s	Sustain (0-15)
r	Release (0-15)
w	Wellenform (Wave)
	0 Dreieckschwingung
	1 Sägezahnschwingung
	2 Rechteckschwingung
	3 Rauschen
	4 Ringmodulation
p	Pulsverhältnis für w=2 (0-4096)

FILTER

Kategorie: Anweisungen

Schreibweise: FILTER fr,tp,bp,hp,re

Funktion: Mit der FILTER Anweisung können Sie drei verschiedene Filterarten ein- oder ausschalten (Hoch-, Tief- und Bandpaß). Dadurch können die einzelnen Töne noch weiter verfeinert bzw. beeinflußt werden.

Die Parameter bedeuten dabei im einzelnen:

fr	Frequenzgrenze, ab der der Filter wirksam wird.
tp	Tiefpaßfilter (0=aus/1=ein)
bp	Bandpaßfilter (0=aus/1=ein)
hp	Hochpaßfilter (0=aus/1=ein)
re	Resonanz (0-15)

Die Resonanz bestimmt, inwieweit ein Ton hart oder weich klingt.

PLAY

Kategorie: Befehle

Schreibweise: PLAY "Vn,On,Tn,Un,Xn,noten"

Funktion: Mit dem PLAY-Befehl können auf einfache Art und Weise Töne bzw. Melodien gespielt werden (Vergleichbar mit dem PRINT-Befehl bei der Textausgabe). Der Stringausdruck setzt sich aus den Kontrollcodes (Kürzeln) für den Synthesizer sowie aus den Kürzeln für die Musiknoten zusammen. Wie für alle Stringausdrücke gilt auch hier die 255 Zeichen Beschränkung.

Die Parameter haben die folgende Bedeutung:

Vn	Stimme (n=1-3)
On	Oktave (n=0-6)
Tn	Hüllkurve (n=0-9)
Un	Lautstärke (n=0-15)
Xn	Filter (0=aus/1=ein)

Noten = C,D,E,F,G,A,B

Die Noten können noch mit zusätzlichen Kennungen versehen werden, die folgende Bedeutung haben:

#	Halbton nach oben
$	Halbton nach unten
W	ganze Note
H	Halbe Note
Q	Viertel Note
I	Achtel Note
S	Sechzehntel Note
.	punktierte Note
R	spielt Rest einer Note
M	wartet, bis alle Stimmen zu Ende gespielt wurden

SOUND

Kategorie: Befehle

Schreibweise: SOUND s,f,d (,r,m,sc,w,p)

Funktion: Der SOUND-Befehl stellt eine einfache Möglichkeit dar, Musik bzw. Geräusche zu erzeugen. Es stehen einem zwar nicht die vielfältigen Möglichkeiten zur Verfügung, die man durch die Kombinationen von ENVELOPE, FILTER usw. bekommt, dafür braucht man sich aber auch nicht den Kopf über die vielen verschiedenen Parameter und deren Zusammenhang zu zerbrechen, die für diese Befehle notwendig sind.

Die Parameter des SOUND-Befehls haben die folgende Bedeutung:

s	Stimme (1-3)
f	Frequenz (0-65535)
d	Dauer in 1/60tel Sekunde (0-32767)
r	Gibt an, ob die Frequenz hoch- (=0) oder runtergezählt (=1) wird. (2=schwingt zwischen zwei Frequenzwerten hin und her.)
m	Gibt den Minimalwert der Frequenz für Parameter r an. (0-65535)
sc	Bestimmt die Schrittweite für r (0-32676)
w	Wellenform (0-3)
p	Pulsbreite (Rechteckabstand)

TEMPO

Kategorie: Anweisungen

Schreibweise: TEMPO x

Funktion: TEMPO bestimmt die Geschwindigkeit, mit der die Noten gespielt werden sollen. Der Wert für x bewegt sich im Bereich von 0 bis 255. Der Vorgabewert ist 8. Mit der folgenden Formel kann die Dauer bestimmt werden:

D=19.22/x Sekunden

Je größer x also wird, desto schneller werden die Noten gespielt.

VOL

Kategorie: Anweisungen

Schreibweise: VOL x

Funktion: Durch die Anweisung VOL wird die Lautstärke bestimmt, mit der die Noten gespielt werden sollen. Die Werte für X liegen im Bereich von 0 bis 15, wobei 15 die größte Lautstärke darstellt.

Anhang B: Reservierte Wörter

A

ABS, AND, APPEND, ASC, ATN, AUTO

B

BACKUP, BANK, BEGIN, BEND, BLOAD, BOOT, BOX, BSAVE, BUMP

C

CATALOG, CHAR, CHR$, CIRCLE, CLOSE, CLR, CMD, COLLECT, COLLISION, COLOR, CONCAT, CONT, COPY

D

DATA, DCLEAR, DCLOSE, DEC, DEF, DELETE, DIM, DIRECTORY, DLOAD, DO, DOPEN, DRAW, DS, DSAVE, DS$, DVERIFY

E

EL, ELSE, END, ENVELOPE, ER, ERR$, EXIT, EXP

F

FAST, FETCH, FILTER, FN, FOR, FRE

G

GET, GETKEY, GO64, GOSUB, GOTO, GRAPHIC, GSHAPE

H

HEADER, HELP, HEX$

I

IF, INSTR, INPUT, INPUT#, INT

J

JOY

K

KEY

L

LEFT$, LEN, LET, LIST, LOAD, LOCATE, LOG, LOOP

M

MID$, MONITOR, MOVSPR

N

NEW, NEXT, NOT

O

OFF, ON, OPEN, OR

P

PAINT, PEEK, PEN, PLAY, POINTER, POKE, POS, POT, PRINT, PRINT#, PUDEF

Q

QUIT

R

RCLR, RDOT, READ, RECORD, REM, RENAME, RENUMBER, RESTORE, RESUME, RETURN, RGR, RIGHT$, RND, RREG, RSPCOLOR, RSPPOS, RSPRITE, RUN, RWINDOW

S

SAVE, SCALE, SCNCLR, SCRATCH, SGN, SIN, SLEEP, SLOW, SOUND, SPC, SPCOLOR, SPRDEF, SPRITE, SPRSAV, SQR, SSHAPE, ST, STASH, STEP, STOP, SWAP, SYS

T

TAB, TAN, TEMPO, THEN, TI, TI$, TO, TRAP, TRON, TROFF

U

UNTIL, USING, USR

V

VAL, VERIFY, VOL

W

WAIT, WHILE, WIDTH, WINDOW

X

XOR

Anhang C: Die Tokentabelle

Befehl	Token	Befehl	Token
END	$80	FOR	$81
NEXT	$82	DATA	$83
INPUT#	$84	INPUT	$85
DIM	$86	READ	$87
LET	$88	GOTO	$89
RUN	$8A	IF	$8B
RESTORE	$8C	GOSUB	$8D
RETURN	$8E	REM	$8F
STOP	$90	ON	$91
WAIT	$92	LOAD	$93
SAVE	$94	VERIFY	$95
DEF	$96	POKE	$97
PRINT#	$98	PRINT	$99
CONT	$9A	LIST	$9B
CLR	$9C	CMD	$9D
SYS	$9E	OPEN	$9F
CLOSE	$A0	GET	$A1
NEW	$A2	TAB(	$A3
TO	$A4	FN	$A5
SPC(	$A6	THEN	$A7
NOT	$A8	STEP	$A9
+	$AA	-	$AB
*	$AC	/	$AD
↑	$AE	AND	$AF
OR	$B0	>	$B1
=	$B2	<	$B3
SGN	$B4	INT	$B5
ABS	$B6	USR	$B7
FRE	$B8	POS	$B9
SQR	$BA	RND	$BB
LOG	$BC	EXP	$BD
COS	$BE	SIN	$BF
TAN	$C0	ATN	$C1
PEEK	$C2	LEN	$C3
STR$	$C4	VAL	$C5

Befehl	Token	Befehl	Token
ASC	$C6	CHR$	$C7
LEFT$	$C8	RIGHT$	$C9
MID$	$CA	GO	$CB
RGR	$CC	RCLR	$CD
POT	$CE $02	BUMP	$CE $03
PEN	$CE $04	RSPPOS	$CE $05
RSPRITE	$CE $06	RSPCOLOR	$CE $07
XOR	$CE $08	RWINDOW	$CE $09
POINTER	$CE $0A	JOY	$CF
RDOT	$D0	DEC	$D1
HEX$	$D2	ERR$	$D3
INSTR	$D4	ELSE	$D5
RESUME	$D6	TRAP	$D7
TRON	$D8	TROFF	$D9
SOUND	$DA	VOL	$DB
AUTO	$DC	PUDEF	$DD
GRAPHIC	$DE	PAINT	$DF
CHAR	$E0	BOX	$E1
CIRCLE	$E2	GSHAPE	$E3
SSHAPE	$E4	DRAW	$E5
LOCATE	$E6	COLOR	$E7
SCNCLR	$E8	SCALE	$E9
HELP	$EA	DO	$EB
LOOP	$EC	EXIT	$ED
DIRECTORY	$EE	DSAVE	$EF
DLOAD	$F0	HEADER	$$F1
SCRATCH	$F2	COLLECT	$F3
COPY	$F4	RENAME	$F5
BACKUP	$F6	DELETE	$F7
RENUMBER	$F8	KEY	$F9
MONITOR	$FA	USING	$FB
UNTIL	$FC	WHILE	$FD
BANK	$FE $02	FILTER	$FE $03
PLAY	$FE $04	TEMPO	$FE $05
MOVSPR	$FE $06	SPRITE	$FE $07
SPRCOLOR	$FE $08	RREG	$FE $09
ENVELOPE	$FE $0A	SLEEP	$FE $0B
CATALOG	$FE $0C	DOPEN	$FE $0D

Befehl	Token	Befehl	Token
APPEND	$FE $0E	DCLOSE	$FE $0F
BSAVE	$FE $10	BLOAD	$FE $11
RECORD	$FE $12	CONCAT	$FE $13
DVERIFY	$FE $14	DCLEAR	$FE $15
SPRSAV	$FE $16	COLLISION	$FE $17
BEGIN	$FE $18	BEND	$FE $19
WINDOW	$FE $1A	BOOT	$FE $1B
WIDTH	$FE $1C	SPRDEF	$FE $1D
QUIT	$FE $1E	STASH	$FE $1F
FETCH	$FE $21	SWAP	$FE $23
OFF	$FE $24	FAST	$FE $25
SLOW	$FE $26		

Anhang D: Dezimal-, Hexadezimal- und Binärtabelle

Dez.	Hex.	Binär	Dez.	Hex.	Binär
#000	$00	%00000000	#001	$01	%00000001
#002	$02	%00000010	#003	$03	%00000011
#004	$04	%00000100	#005	$05	%00000101
#006	$00	%00000110	#007	$07	%00000111
#008	$08	%00001000	#009	$09	%00001001
#010	$0A	%00001010	#011	$0B	%00001011
#012	$0C	%00001100	#013	$0D	%00001101
#014	$0E	%00001110	#015	$0F	%00001111
#016	$10	%00010000	#017	$11	%00010001
#018	$12	%00010010	#019	$13	%00010011
#020	$14	%00010100	#021	$15	%00010101
#022	$16	%00010110	#023	$17	%00010111
#024	$18	%00011000	#025	$19	%00011001
#026	$1A	%00011010	#027	$1B	%00011011
#028	$1C	%00011100	#029	$1D	%00011101
#030	$1E	%00011110	#031	$1F	%00011111
#032	$20	%00100000	#033	$21	%00100001
#034	$22	%00100010	#035	$23	%00100011
#036	$24	%00100100	#037	$25	%00100101
#038	$26	%00100110	#039	$27	%00100111
#040	$28	%00101000	#041	$29	%00101001
#042	$2A	%00101010	#043	$2B	%00101011
#044	$2C	%00101100	#045	$2D	%00101101
#046	$2E	%00101110	#047	$2F	%00101111
#048	$30	%00110000	#049	$31	%00110001
#050	$32	%00110010	#051	$33	%00110011
#052	$34	%00110100	#053	$35	%00110101
#054	$36	%00110110	#055	$37	%00110111
#056	$38	%00111000	#057	$39	%00111001
#058	$3A	%00111010	#059	$3B	%00111011
#060	$3C	%00111100	#061	$3D	%00111101
#062	$3E	%00111110	#063	$3F	%00111111
#064	$40	%01000000	#065	$41	%01000001
#066	$42	%01000010	#067	$43	%01000011
#068	$44	%01000100	#069	$45	%01000101
#070	$46	%01000110	#071	$47	%01000111

Dez.	Hex.	Binär	Dez.	Hex.	Binär
#072	$48	%01001000	#073	$49	%01001001
#074	$4A	%01001010	#075	$4B	%01001011
#076	$4C	%01001100	#077	$4D	%01001101
#078	$4E	%01001110	#079	$4F	%01001111
#080	$50	%01010000	#081	$51	%01010001
#082	$52	%01010010	#083	$53	%01010011
#084	$54	%01010100	#085	$55	%01010101
#086	$56	%01010110	#087	$57	%01010111
#088	$58	%01011000	#089	$59	%01011001
#090	$5A	%01011010	#091	$5B	%01011011
#092	$5C	%01011100	#093	$5D	%01011101
#094	$5E	%01011110	#095	$5F	%01011111
#096	$60	%01100000	#097	$61	%01100001
#098	$62	%01100010	#099	$63	%01100011
#100	$64	%01100100	#101	$65	%01100101
#102	$66	%01100110	#103	$67	%01100111
#104	$68	%01101000	#105	$69	%01101001
#106	$6A	%01101010	#107	$6B	%01101011
#108	$6C	%01101100	#109	$6D	%01101101
#110	$6E	%01101110	#111	$6F	%01101111
#112	$70	%01110000	#113	$71	%01110001
#114	$72	%01110010	#115	$73	%01110011
#116	$74	%01110100	#117	$75	%01110101
#118	$76	%01110110	#119	$77	%01110111
#120	$78	%01111000	#121	$79	%01111001
#122	$7A	%01111010	#123	$7B	%01111011
#124	$7C	%01111100	#125	$7D	%01111101
#126	$7E	%01111110	#127	$7F	%01111111
#128	$80	%10000000	#129	$81	%10000001
#130	$82	%10000010	#131	$83	%10000011
#132	$84	%10000100	#133	$85	%10000101
#134	$86	%10000110	#135	$87	%10000111
#136	$88	%10001000	#137	$89	%10001001
#138	$8A	%10001010	#139	$8B	%10001011
#140	$8C	%10001100	#141	$8D	%10001101
#142	$8E	%10001110	#143	$8F	%10001111
#144	$90	%10010000	#145	$91	%10010001
#146	$92	%10010010	#147	$93	%10010011

Dez.	Hex.	Binär	Dez.	Hex.	Binär
#148	$94	%10010100	#149	$95	%10010101
#150	$96	%10010110	#151	$97	%10010111
#152	$98	%10011000	#153	$99	%10011001
#154	$9A	%10011010	#155	$9B	%10011011
#156	$9C	%10011100	#157	$9D	%10011101
#158	$9E	%10011110	#159	$9F	%10011111
#160	$A0	%10100000	#161	$A1	%10100001
#162	$A2	%10100010	#163	$A3	%10100011
#164	$A4	%10100100	#165	$A5	%10100101
#166	$A6	%10100110	#167	$A7	%10100111
#168	$A8	%10101000	#169	$A9	%10101001
#170	$AA	%10101010	#171	$AB	%10101011
#172	$AC	%10101100	#173	$AD	%10101101
#174	$AE	%10101110	#175	$AF	%10101111
#176	$B0	%10110000	#177	$B1	%10110001
#178	$B2	%10110010	#179	$B3	%10110011
#180	$B4	%10110100	#181	$B5	%10110101
#182	$B6	%10110110	#183	$B7	%10110111
#184	$B8	%10111000	#185	$B9	%10111001
#186	$BA	%10111010	#187	$BB	%10111011
#188	$BC	%10111100	#189	$BD	%10111101
#190	$BE	%10111110	#191	$BF	%10111111
#192	$C0	%11000000	#193	$C1	%11000001
#194	$C2	%11000010	#195	$C3	%11000011
#196	$C4	%11000100	#197	$C5	%11000101
#198	$C6	%11000110	#199	$C7	%11000111
#200	$C8	%11001000	#201	$C9	%11001001
#202	$CA	%11001010	#203	$CB	%11001011
#204	$CC	%11001100	#205	$CD	%11001101
#206	$CE	%11001110	#207	$CF	%11001111
#208	$D0	%11010000	#209	$D1	%11010001
#210	$D2	%11010010	#211	$D3	%11010011
#212	$D4	%11010100	#213	$D5	%11010101
#214	$D6	%11010110	#215	$D7	%11010111
#216	$D8	%11011000	#217	$D9	%11011001
#218	$DA	%11011010	#219	$DB	%11011011
#220	$DC	%11011100	#221	$DD	%11011101
#222	$DE	%11011110	#223	$DF	%11011111

Dez.	Hex.	Binär	Dez.	Hex.	Binär
#224	$E0	%11100000	#225	$E1	%11100001
#226	$E2	%11100010	#227	$E3	%11100011
#228	$E4	%11100100	#229	$E5	%11100101
#230	$E6	%11100110	#231	$E7	%11100111
#232	$E8	%11101000	#233	$E9	%11101001
#234	$EA	%11101010	#235	$EB	%11101011
#236	$EC	%11101100	#237	$ED	%11101101
#238	$EE	%11101110	#239	$EF	%11101111
#240	$F0	%11110000	#241	$F1	%11110001
#242	$F2	%11110010	#243	$F3	%11110011
#244	$F4	%11110100	#245	$F5	%11110101
#246	$F6	%11110110	#247	$F7	%11110111
#248	$F8	%11111000	#249	$F9	%11111001
#250	$FA	%11111010	#251	$FB	%11111011
#252	$FC	%11111100	#253	$FD	%11111101
#254	$FE	%11111110	#255	$FF	%11111111

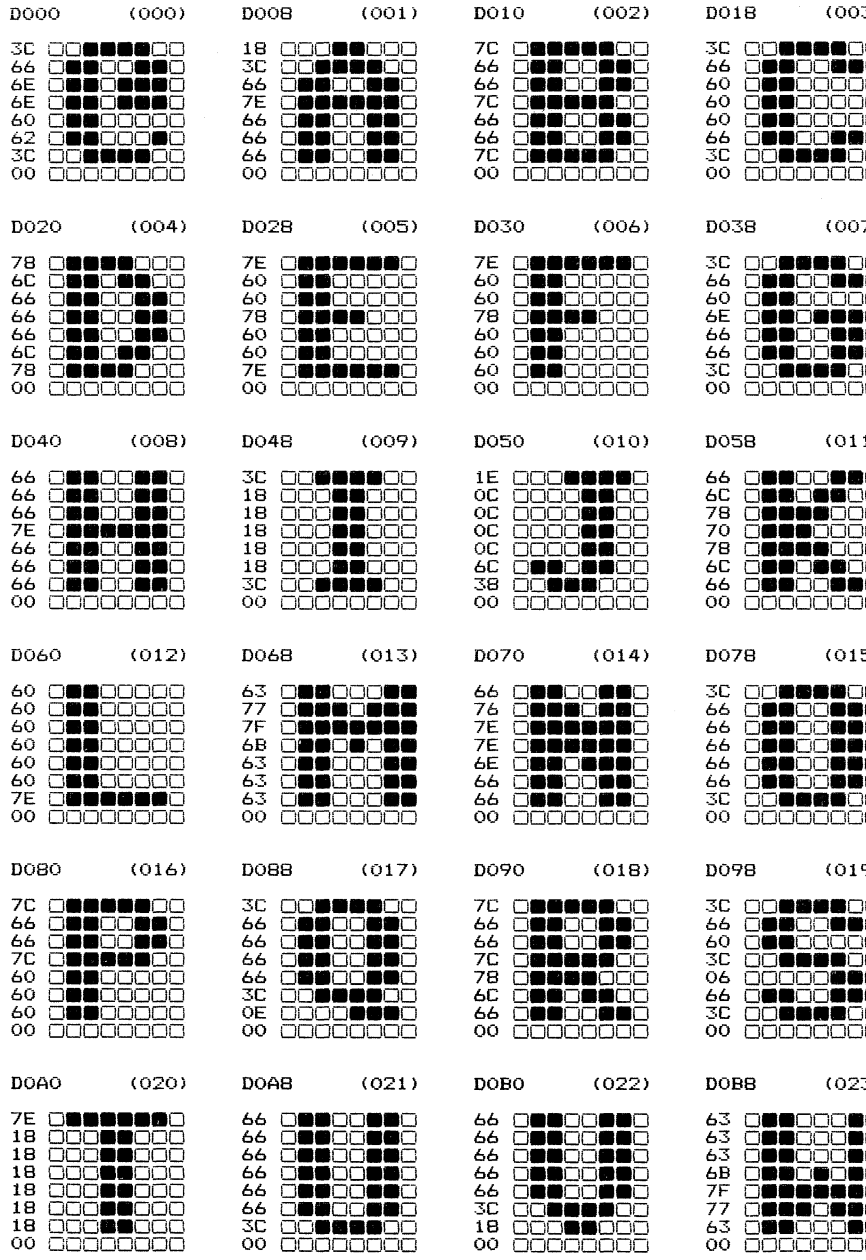
Anhang E: Fehlermeldungen

ER	ERR$
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
6	NOT INPUT FILE
7	NOT OUTPUT FILE
8	MISSING FILE NAME
9	ILLEGAL DEVICE NUMBER
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEF'D STATEMENT
18	BAD SUBSCRIPT
19	REDIM'D ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG
24	FILE DATA
25	FORMULA TOO COMPLEX
26	CAN'T CONTINUE
27	UNDEF'D FUNCTION
28	VERIFY
29	LOAD
30	BREAK
31	CAN'T RESUME
32	LOOP NOT FOUND
33	LOOP WITHOUT DO
34	DIRECT MODE ONLY
35	NO GRAPHICS AREA
36	BAD DISK

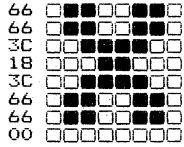
ER	ERR$
37	BEND NOT FOUND
38	LINE NUMBER TOO LARGE
39	UNRESOLVED REFERENCE
40	UNIMPLEMENTED COMMAND
41	FILE READ

Anhang F: Die Zeichensätze

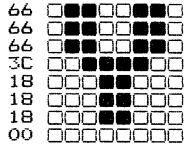
Der C-128 verfügt neben dem Commodore-Standard-ASCII-Zeichensatz über einen DIN-Zeichensatz. Diese beiden Zeichensätze sind im folgenden abgebildet. Sie können durch Betätigen der ASCII/DIN-Taste zwischen den beiden Zeichensätzen hin- und herschalten.


D000 (000)
3C
66
6E
6E
60
62
3C
00
D008 (001)
18
3C
66
7E
66
66
66
00
D010 (002)
7C
66
66
7C
66
66
7C
00
D018 (003)
3C
66
60
60
60
66
3C
00
D020 (004)
78
6C
66
66
66
6C
78
00
D028 (005)
7E
60
60
78
60
60
7E
00
D030 (006)
7E
60
60
78
60
60
60
00
D038 (007)
3C
66
60
6E
66
66
3C
00
D040 (008)
66
66
66
7E
66
66
66
00
D048 (009)
3C
18
18
18
18
18
3C
00
D050 (010)
1E
0C
0C
0C
0C
6C
38
00
D058 (011)
66
6C
78
70
78
6C
66
00
D060 (012)
60
60
60
60
60
60
7E
00
D068 (013)
63
77
7F
6B
63
63
63
00
D070 (014)
66
76
7E
7E
6E
66
66
00
D078 (015)
3C
66
66
66
66
66
3C
00
D080 (016)
7C
66
66
7C
60
60
60
00
D088 (017)
3C
66
66
66
66
3C
0E
00
D090 (018)
7C
66
66
7C
78
6C
66
00
D098 (019)
3C
66
60
3C
06
66
3C
00
D0A0 (020)
7E
18
18
18
18
18
18
00
D0A8 (021)
66
66
66
66
66
66
3C
00
D0B0 (022)
66
66
66
66
66
3C
18
00
D0B8 (023)
63
63
63
6B
7F
77
63
00

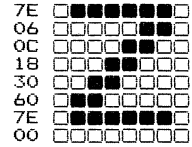
D0C0 (024)



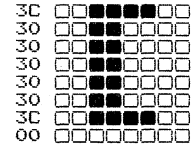
D0C8 (025)



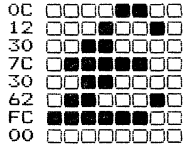
D0D0 (026)



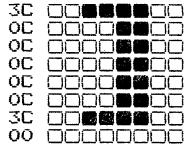
D0D8 (027)



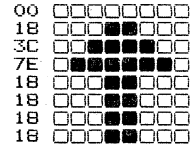
D0E0 (028)



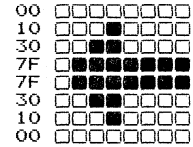
D0E8 (029)



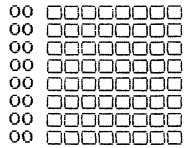
D0F0 (030)



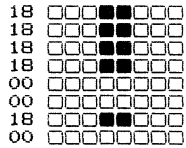
D0F8 (031)



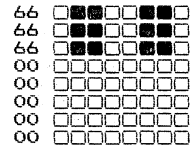
D100 (032)



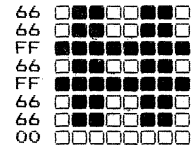
D108 (033)



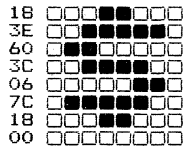
D110 (034)



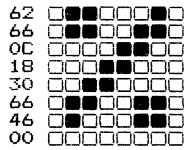
D118 (035)



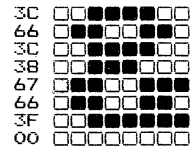
D120 (036)



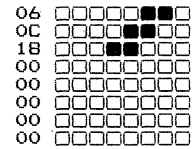
D128 (037)



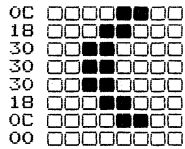
D130 (038)



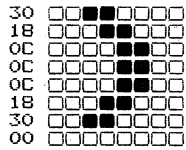
D138 (039)



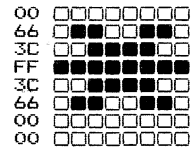
D140 (040)



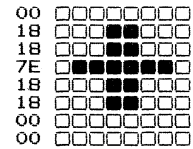
D148 (041)



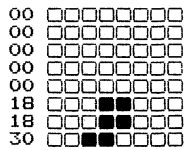
D150 (042)



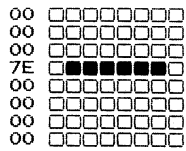
D158 (043)



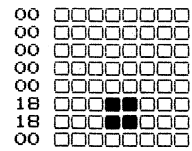
D160 (044)



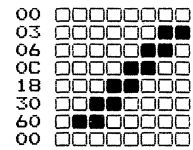
D168 (045)



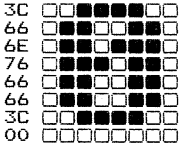
D170 (046)



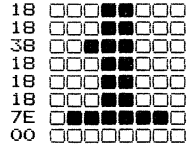
D178 (047)



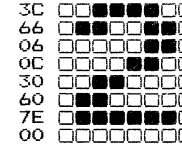
D180 (048)



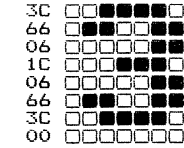
D188 (049)



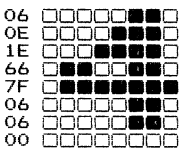
D190 (050)



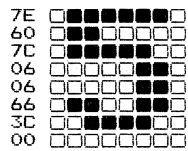
D198 (051)



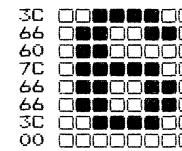
D1A0 (052)



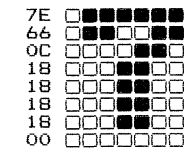
D1A8 (053)



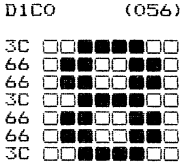
D1B0 (054)



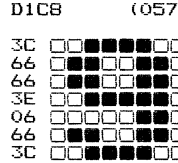
D1B8 (055)



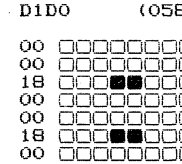
D1C0 (056)



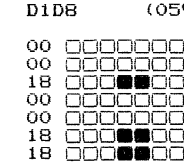
D1C8 (057)



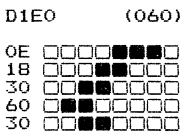
D1D0 (058)



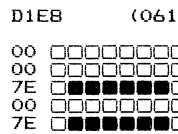
D1D8 (059)



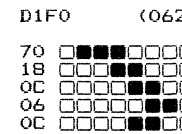
D1E0 (060)



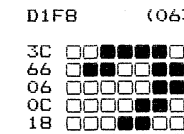
D1E8 (061)



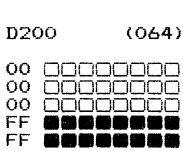
D1F0 (062)



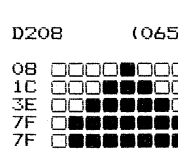
D1F8 (063)



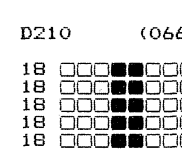
D200 (064)



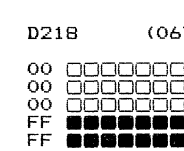
D208 (065)



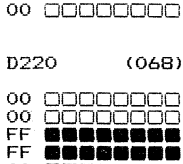
D210 (066)



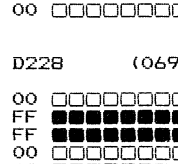
D218 (067)



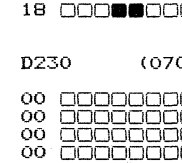
D220 (068)



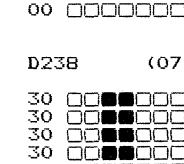
D228 (069)



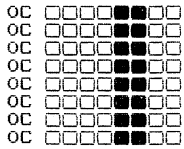
D230 (070)



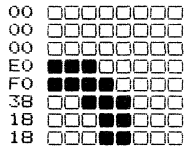
D238 (071)



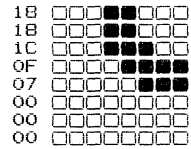
D240 (072)



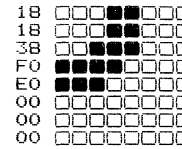
D248 (073)



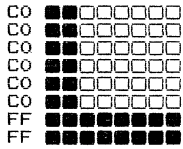
D250 (074)



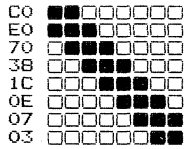
D258 (075)



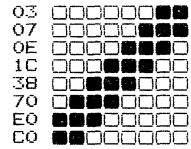
D260 (076)



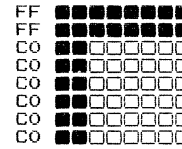
D268 (077)



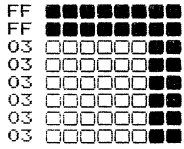
D270 (078)



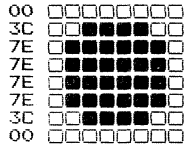
D278 (079)



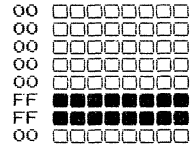
D280 (080)



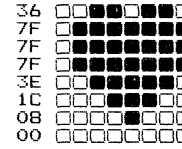
D288 (081)



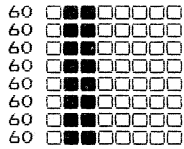
D290 (082)



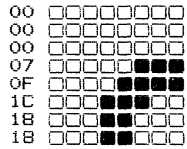
D298 (083)



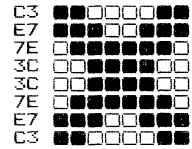
D2A0 (084)



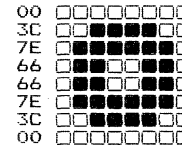
D2A8 (085)



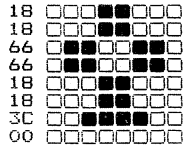
D2B0 (086)



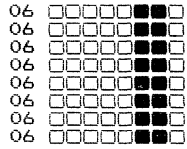
D2B8 (087)



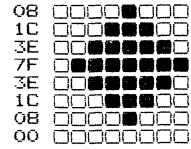
D2C0 (088)



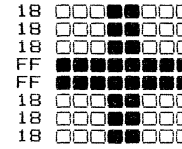
D2C8 (089)



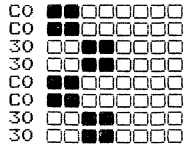
D2D0 (090)



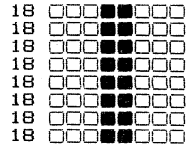
D2D8 (091)



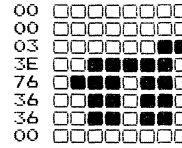
D2E0 (092)



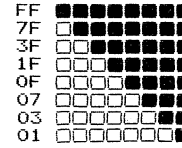
D2E8 (093)

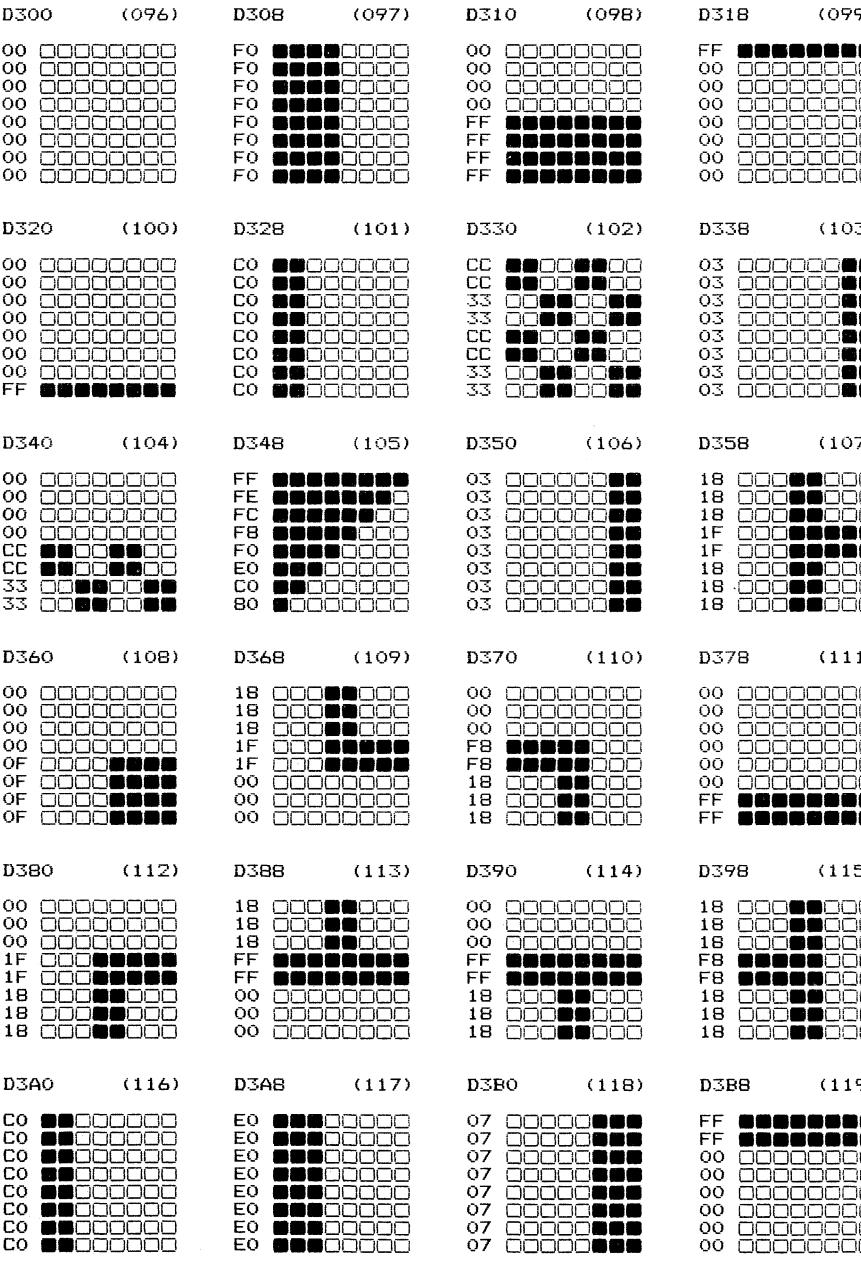


D2F0 (094)

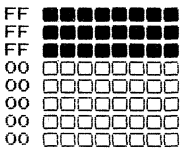


D2F8 (095)

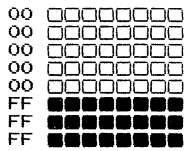



D300 (096)
00 00 00 00 00 00 00 00
D308 (097)
F0 F0 F0 F0 F0 F0 F0 F0
D310 (098)
00 00 00 00 FF FF FF FF
D318 (099)
FF 00 00 00 00 00 00 00
D320 (100)
00 00 00 00 00 00 00 FF
D328 (101)
C0 C0 C0 C0 C0 C0 C0 C0
D330 (102)
CC CC 33 33 CC CC 33 33
D338 (103)
03 03 03 03 03 03 03 03
D340 (104)
00 00 00 00 CC CC 33 33
D348 (105)
FF FE FC F8 F0 E0 C0 80
D350 (106)
03 03 03 03 03 03 03 03
D358 (107)
18 18 18 1F 1F 18 18 18
D360 (108)
00 00 00 00 0F 0F 0F 0F
D368 (109)
18 18 18 1F 1F 00 00 00
D370 (110)
00 00 00 F8 F8 18 18 18
D378 (111)
00 00 00 00 00 00 FF FF
D380 (112)
00 00 00 1F 1F 18 18 18
D388 (113)
18 18 18 FF FF 00 00 00
D390 (114)
00 00 00 FF FF 18 18 18
D398 (115)
18 18 18 F8 F8 18 18 18
D3A0 (116)
C0 C0 C0 C0 C0 C0 C0 C0
D3A8 (117)
E0 E0 E0 E0 E0 E0 E0 E0
D3B0 (118)
07 07 07 07 07 07 07 07
D3B8 (119)
FF FF 00 00 00 00 00 00

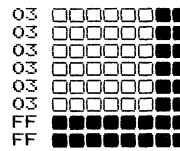
D3C0 (120)



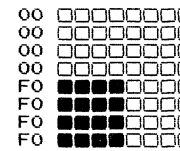
D3C8 (121)



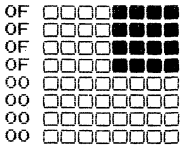
D3D0 (122)



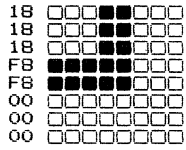
D3D8 (123)



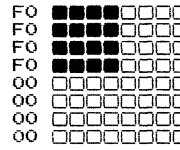
D3E0 (124)



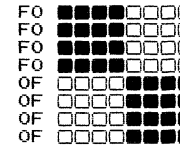
D3E8 (125)

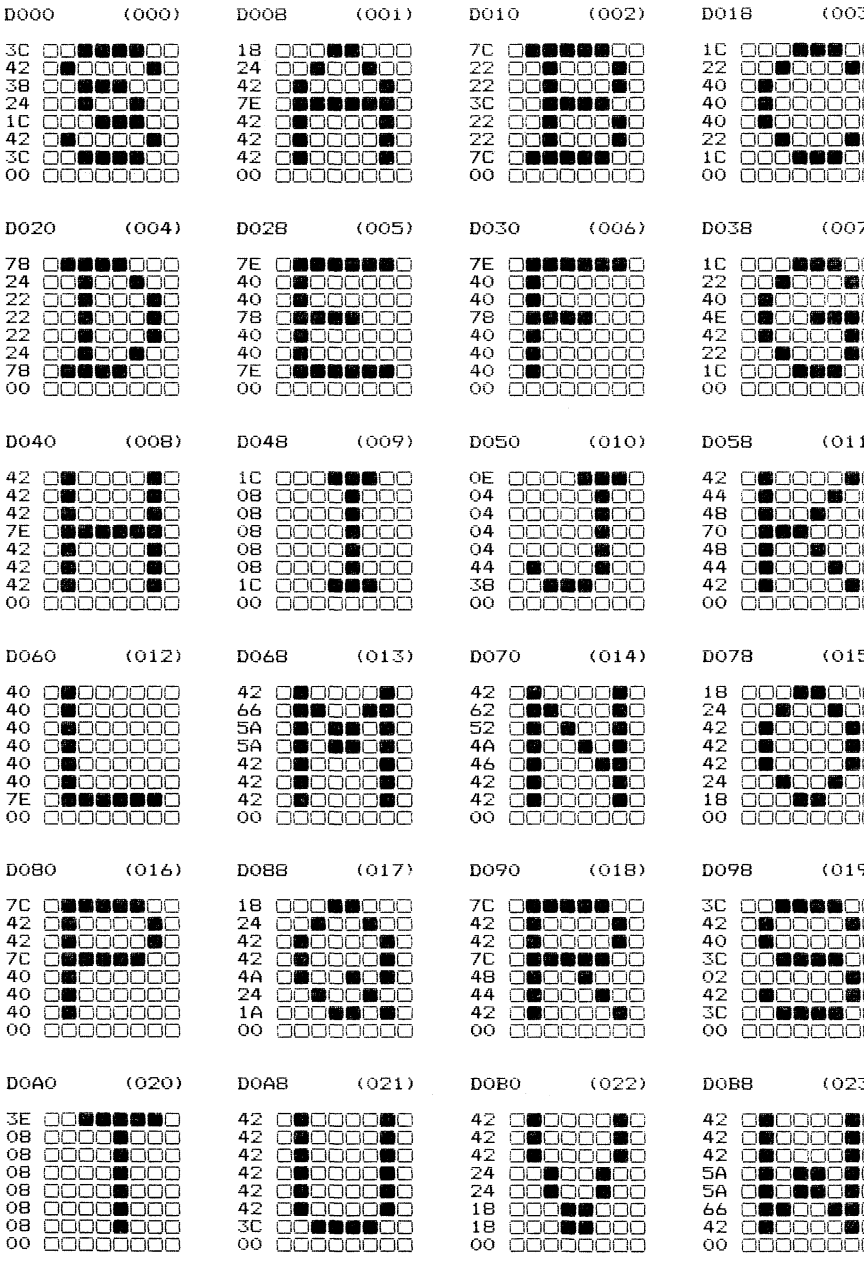


D3F0 (126)

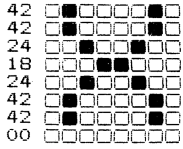


D3F8 (127)

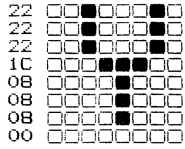



D000 (000)
3C 42 38 24 1C 42 3C 00
D008 (001)
18 24 42 7E 42 42 42 00
D010 (002)
7C 22 22 3C 22 22 7C 00
D018 (003)
1C 22 40 40 40 22 1C 00
D020 (004)
78 24 22 22 22 24 78 00
D028 (005)
7E 40 40 78 40 40 7E 00
D030 (006)
7E 40 40 78 40 40 40 00
D038 (007)
1C 22 40 4E 42 22 1C 00
D040 (008)
42 42 42 7E 42 42 42 00
D048 (009)
1C 08 08 08 08 08 1C 00
D050 (010)
0E 04 04 04 04 44 38 00
D058 (011)
42 44 48 70 48 44 42 00
D060 (012)
40 40 40 40 40 40 7E 00
D068 (013)
42 66 5A 5A 42 42 42 00
D070 (014)
42 62 52 4A 46 42 42 00
D078 (015)
18 24 42 42 42 24 18 00
D080 (016)
7C 42 42 7C 40 40 40 00
D088 (017)
18 24 42 42 4A 24 1A 00
D090 (018)
7C 42 42 7C 48 44 42 00
D098 (019)
3C 42 40 3C 02 42 3C 00
D0A0 (020)
3E 08 08 08 08 08 08 00
D0A8 (021)
42 42 42 42 42 42 3C 00
D0B0 (022)
42 42 42 24 24 18 18 00
D0B8 (023)
42 42 42 5A 5A 66 42 00

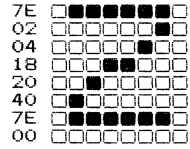
D0C0 (024)



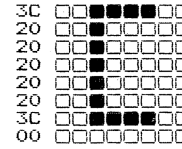
D0C8 (025)



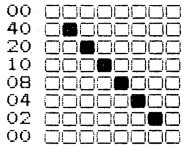
D0D0 (026)



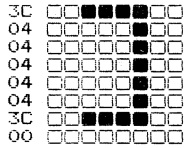
D0D8 (027)



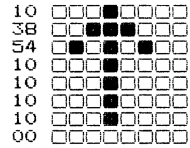
D0E0 (028)



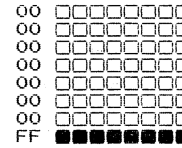
D0E8 (029)



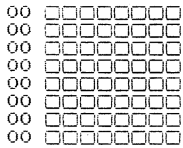
D0F0 (030)



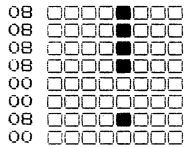
D0F8 (031)



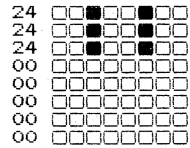
D100 (032)



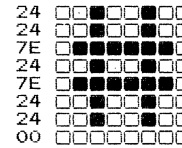
D108 (033)



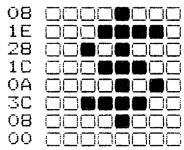
D110 (034)



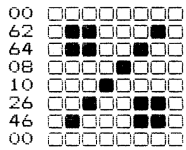
D118 (035)



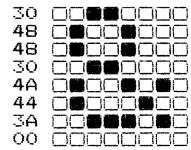
D120 (036)



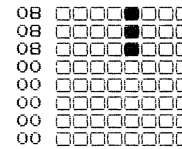
D128 (037)



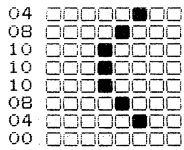
D130 (038)



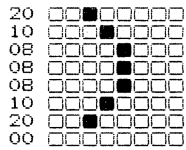
D138 (039)



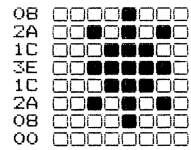
D140 (040)



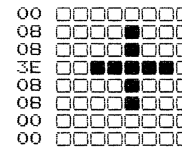
D148 (041)



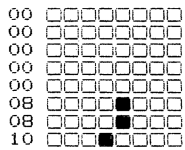
D150 (042)



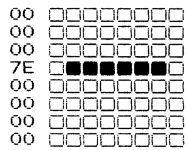
D158 (043)



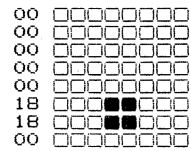
D160 (044)



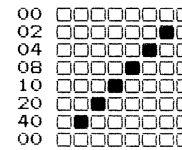
D168 (045)



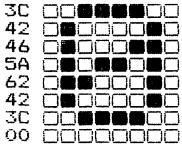
D170 (046)



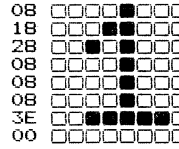
D178 (047)



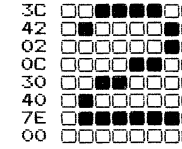
D180 (048)



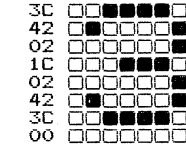
D188 (049)



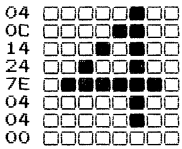
D190 (050)



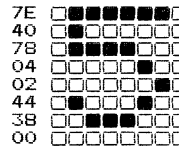
D198 (051)



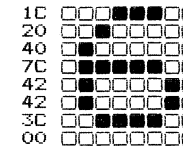
D1A0 (052)



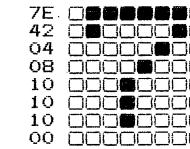
D1A8 (053)



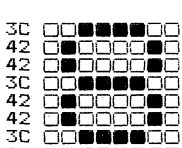
D1B0 (054)



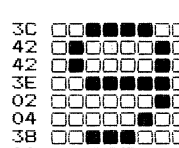
D1B8 (055)



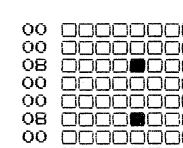
D1C0 (056)



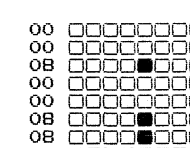
D1C8 (057)



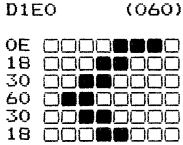
D1D0 (058)



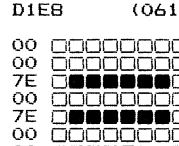
D1D8 (059)



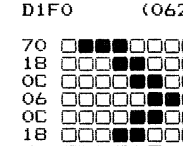
D1E0 (060)



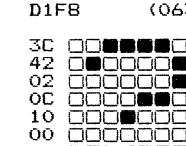
D1E8 (061)



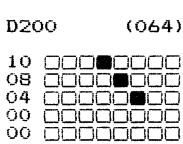
D1F0 (062)



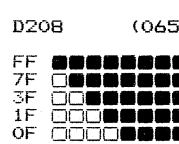
D1F8 (063)



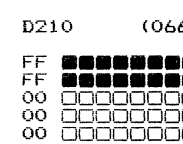
D200 (064)



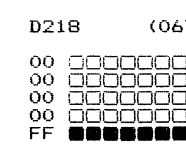
D208 (065)



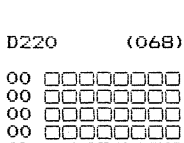
D210 (066)



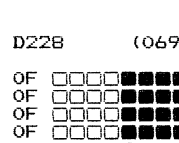
D218 (067)



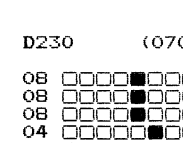
D220 (068)



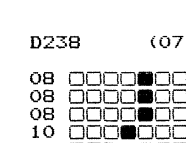
D228 (069)



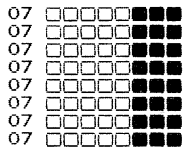
D230 (070)



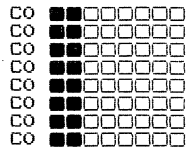
D238 (071)



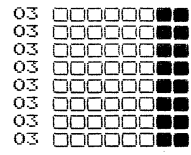
D240 (072)



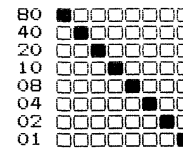
D248 (073)



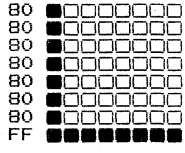
D250 (074)



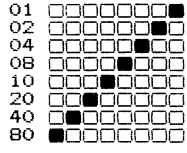
D258 (075)



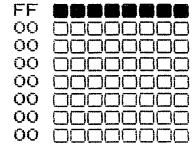
D260 (076)



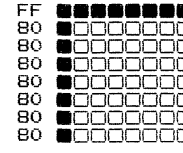
D268 (077)



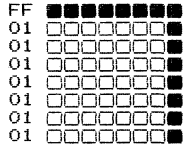
D270 (078)



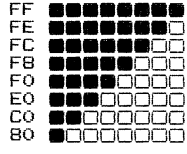
D278 (079)



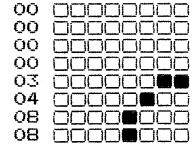
D280 (080)



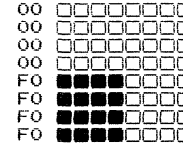
D288 (081)



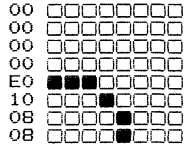
D290 (082)



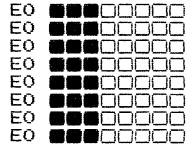
D298 (083)



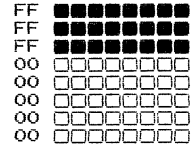
D2A0 (084)



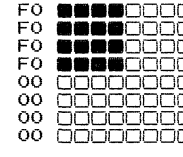
D2A8 (085)



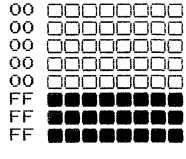
D2B0 (086)



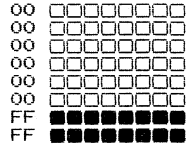
D2B8 (087)



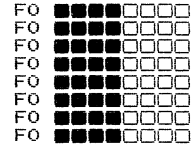
D2C0 (088)



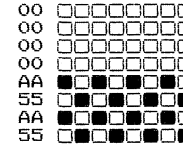
D2C8 (089)



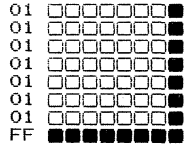
D2D0 (090)



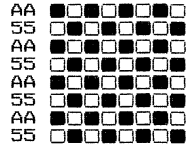
D2D8 (091)



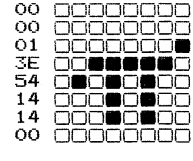
D2E0 (092)



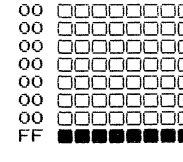
D2E8 (093)



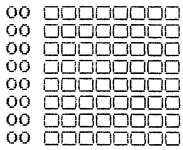
D2F0 (094)



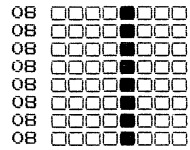
D2F8 (095)



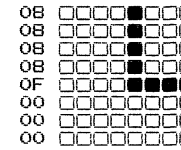
D300 (096)



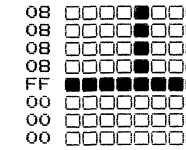
D308 (097)



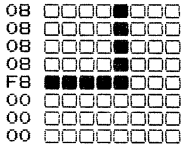
D310 (098)



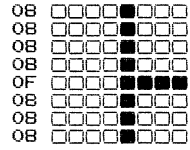
D318 (099)



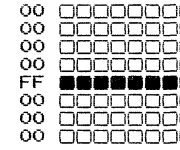
D320 (100)



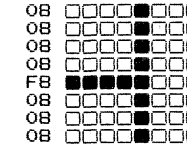
D328 (101)



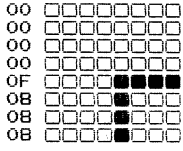
D330 (102)



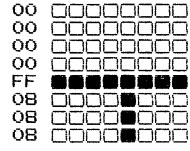
D338 (103)



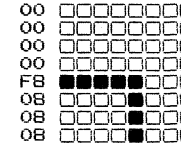
D340 (104)



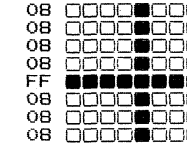
D348 (105)



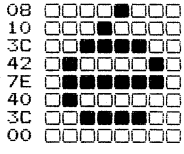
D350 (106)



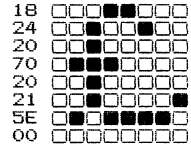
D358 (107)



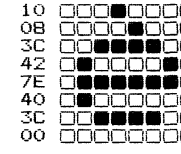
D360 (108)



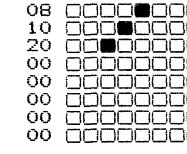
D368 (109)



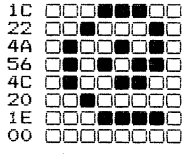
D370 (110)



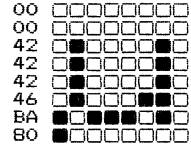
D378 (111)



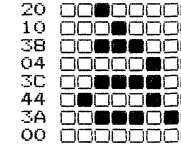
D380 (112)



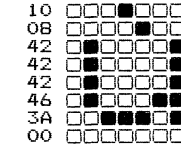
D388 (113)



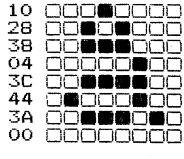
D390 (114)



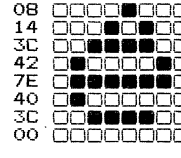
D398 (115)



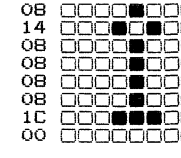
D3A0 (116)



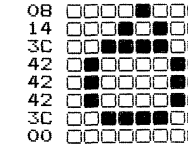
D3A8 (117)



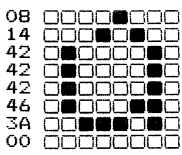
D3B0 (118)



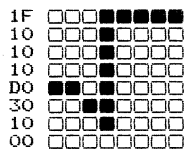
D3B8 (119)



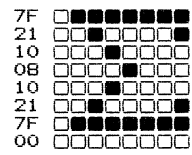
D3C0 (120)



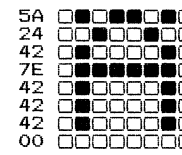
D3C8 (121)



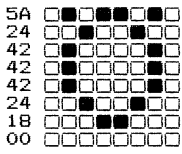
D3D0 (122)



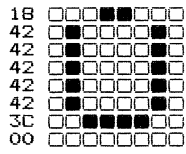
D3D8 (123)



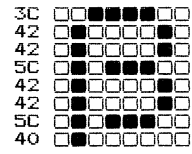
D3E0 (124)



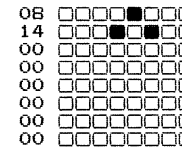
D3E8 (125)



D3F0 (126)



D3F8 (127)



Anhang G: Stichwortregister

Abspeichern eines Programms 143
Algorithmus 17f, 50
Analoguhr 295f
Arrays 188, 191
ASCII-Codes 31
Assemblercode 300
Aussagenlogik 40
Automatische Zeilennumerierung 105

BASIC 17
BASIC-Interpreter 18
Bedingte Programmsprünge 116
Berechnete Sprungbefehle 139
Bildschirmfenster 265ff
Bit 34
Bogenmaß 75f
Boolesche Operatoren 39
Bubble-Sort-Verfahren 270
Byte 34f

Cosinus 75
Cursorpositionierung 254

Dateiverwaltung 216, 274ff
Datenfluß 19f
Datenflußplan 20ff, 51ff, 113, 138
Dezimalsystem 33, 82ff
Direktmodus 57, 159
Dokumentation 28, 53
Dualsystem 32f

Editieren 105
Eindimensionale Felder 188ff, 195ff
Eulersche Zahl 77

Fehlerroutine 153, 156
Felder 188, 191ff
Feldvariable 191

Fenster 268
Flags 272
Formatieren 274
FORTRAN 18
Führende Nullen 307

Ganzzahlvariable 70
Geräteadresse 277
Grafikprogrammierung 291

Hardcopy 306
Hexadezimalsystem 36, 83f
High-Byte 35ff, 303f
Hüllkurve 288f

Indizierte Variable 191
Integer-Variable 70, 79

Konnektor 26, 114, 137

Laufvariable 124ff, 129f
Lineare Programmablaufpläne 50
Logarithmus 78
Logische Operatoren 39ff
Low-Byte 35ff, 303f

Maschinensprache 18, 300
Mehrdimensionale Felder 206ff
Menü 149, 221f, 238ff, 244
Monitor 300f
Multistatements 53
Musikprogrammierung 286ff, 291

Nibble 38

Priorität der Logischen Operatoren 45
Priorität der Rechenoperationen 72
Programm 17
Programmablauf 19, 24
Programmablaufplan 20, 25, 52f, 114, 136, 142

Programmdokumentation 28
Programmierschablone 20f
Programmodus 57

Real-Variable 70
Relative Datei 276, 282ff
Reservierte Wörter 71, 426

Schachtelung von Schleifen 126ff
Schleifen 27, 51, 119, 124ff
Sequentielle Datei 276ff, 308
SID 286
Sinus 75
Speicherplatz 171
Starten des Programms 27, 58
Steuerzeichen 85f
Strings 91ff
Stringvariable 70, 84f, 91ff, 101
Strukturierte Programmierung 121
Stufen des Programmierens 29, 50ff

Tastaturpuffer 160
Token 302, 430

Unbedingte Programmsprünge 112
Unterprogramm 216ff
Unterprogramme 51
Utilities 306

Variable 54f, 70ff

Zufallszahlen 81f
Zwischenspeicherung von Variablen 151, 229

Bücher zum Commodore 128

Ein Standardwerk zum COMMODORE 128, das für jeden unentbehrlich ist, der tiefer in den COMMODORE 128 einsteigen will. Das gesamte Betriebssystem ist ausführlich und gründlich kommentiert, Grafik, Soundbausteine, Prozessor und Peripherieanschlüsse sind genauestens beschrieben. Ein Buch, das für den professionellen Programmierer sehr schnell unentbehrlich wird.

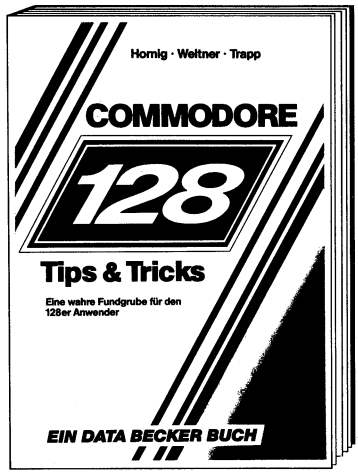
Aus dem Inhalt:

- Der VIC-Chip
  Registerbelegung
  Betriebsarten
  Zeichendarstellung, Graphik, Sprites und Soft-Scrolling
- Ein- und Ausgabesteuerung
  Die CIAs im COMMODORE 128
  Echtzeituhr
  Der serielle IEC-Bus des COMMODORE 128
- Der Sound-Chip SID
- Der 8563-VDC-Chip
  Pinbelegung
  Nutzung der VDC-Register
  Hires-Graphik mit 640×200 Punkten
- Das Memory-Management, die MMU
- Assemblerprogrammierung (Nutzung der Kernal-Routinen)
- Einbinden neuer BASIC-Befehle
- BASIC-Tokens
- Die CPU 8502
- Zeilenweise dokumentiertes Kernal-ROM
- ausführlich dokumentiertes BASIC 7.0
- Z-80-ROM dokumentiert (Boot-Sektor)
- Betriebssystem und Monitorlisting
- Die Hardware

Schieb, Thrun, Wrobel
Commodore 128 Intern
Hardcover, 841 Seiten, DM 69,–
ISBN 3-89011-098-3

Bücher zum Commodore 128

128 Tips & Tricks ist eine riesige Fundgrube für jeden 128er-Besitzer, der mehr mit seinem Rechner machen will. Dieses Buch enthält nicht nur viele Beispielprogramme, sondern erläutert auch leichtverständlich den Aufbau des Rechners und seine Programmierung.



Aus dem Inhalt:

- Grafik auf dem Commodore 128
- Arbeiten mit mehreren Bildschirmen
- Eigener Zeichensatz
- Sprite-Handling
- Grafik mit den eingebauten Befehlen
- Simulation mehrerer Windows
- Listing-Konverter
- Modifiziertes Input
- Software-Schutz auf dem Commodore 128
- Zeilen einfügen
- Rund um die Tastatur
- Befehlserweiterung – selbst gemacht
- Banking
- Weitere Möglichkeiten der MMU
- Autostart
- Der Speicher
- Wechseln des Betriebssystems
- Der 64er-Modus auf dem C-128
- Die 10er-Tastatur am C-64
  und vieles mehr

Hornig, Weltner, Trapp
Commodore 128 Tips & Tricks
Hardcover, 327 Seiten, DM 49,–
ISBN 3-89011-097-5

DAS STEHT DRIN:

Das große BASIC-Buch zum Commodore 128 ist eine ausführliche, didaktisch gut geschriebene Einführung in das CBM BASIC 7.0. Von den BASIC-Befehlen über die Problemanalyse bis zum fertigen Algorithmus lernt man schnell und sicher das Programmieren. Übungsaufgaben helfen, das Gelernte zu vertiefen. Gleichzeitig erhält der BASIC-Programmierer ein praxisbezogenes Nachschlagewerk.

Aus dem Inhalt:

- Datenfluß- und Programmablaufpläne
- Fortgeschrittene Programmiertechniken
- Menüerstellung
- Grafikprogrammierung
- Mehrdimensionale Felder
- Sortierroutinen
- Dateiverwaltung
- Windowprogrammierung
- BASIC intern
- Tokentabelle
- Der Monitor
  und viele nützliche Utilities

UND GESCHRIEBEN HAT DIESES BUCH:

Frank Kampow ist Programmierer mit vielfältigen Erfahrungen als Seminarleiter zu verschiedensten EDV-Themen. Sein pädagogisches Geschick und jahrelange Programmierpraxis machen dieses Buch zu einer nützlichen Hilfe für jeden C128-Anwender.

ISBN 3-89011-114-9