Kampow · Szczepanowski

Das große BASIC Buch zum ATARI® ST

EIN DATA BECKER BUCH

Kampow · Szczepanowski

Das große BASIC Buch zum ATARI® ST

EIN DATA BECKER BUCH

ISBN 3-89011-121-1

Copyright © 1985 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Programms darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

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 1

Kapitel 1:

1. Grundlagen des Programmierens 4
1.1 Algorithmus und Programm 4
1.2 Die Computersprache BASIC 4
1.3 Datenfluß- und Programmablaufpläne 6
1.3.1 Datenflußpläne 9
1.3.2 Programmablaufpläne 12
1.4 ASCII-Codes 17
1.5 Zahlensysteme 17
1.5.1 Das Dualsystem 19
1.5.2 Bit und Byte 20
1.5.3 Das Hexadezimalsystem 21
1.6 Die logischen Operatoren 24
1.6.1 NOT 26
1.6.2 AND 27
1.6.3 OR 28
1.6.4 XOR 30
1.6.5 IMP 32
1.6.6 EQV 32

Aufgaben 34

Kapitel 2:

2. Einführung in das Programmieren mit BASIC 36
2.1 Das erste BASIC-Programm 36
2.1.1 Eingabe von Werten mit INPUT 39
2.1.2 Wertzuweisung mit LET 41
2.1.3 Ausgabe mit PRINT 42
2.1.3.1 PRINT USING 47
2.1.4 Kommentare mit REM 56
2.2 Variablen und deren Verwendung 57

2.2.1 Rechenoperationen mit Variablen 59
Aufgaben 61
2.3 Numerische Funktionen 62
2.3.1 Funktionen mit DEF FN 67
2.3.2 Zufallszahlen 68
2.3.3 Noch mehr Befehle für Variablen 71
2.3.4 ASC(X$) und CHR$(X) 73
Aufgaben 75
2.4 TAB und SPC 76
2.5 Strings 77
2.5.1 LEFT$ 78
2.5.2 RIGHT$ 80
2.5.3 MID$ 81
2.5.4 LEN(X$) 82
2.5.5 VAL(X$) 83
2.5.6 STR$(X) 85
2.5.7 INSTR 85
2.5.8 STRING$ 87
2.5.9 SPACE$ 87
Aufgaben 88
2.6 Editieren von Programmen 90
2.7 Die Bildschirmfenster 95

Kapitel 3:

3. Erweiterte Programmstrukturen 98
3.1 Unbedingte Programmsprünge 98
3.2 Bedingte Programmsprünge 101
3.2.1 IF.. THEN...ELSE 101
3.2.2 Sprungmarken / Labels 105
Aufgaben 107
3.2.3 FOR...TO...NEXT 108
3.2.4 Schleifen mit WHILE...WEND 115
3.3 Berechnete Sprungbefehle 121
3.3.1 Beispielprogramm "Rechenlehrgang" 125
3.3.2 Programmsprünge mit ON ERROR 134
Aufgaben 137
3.4 Die Abfrage Tastatur 138
3.5 FRE, POS, CALL, WAIT 139

3.6 READ, DATA und RESTORE 141

Kapitel 4:

4. Komplexere BASIC-Anwendungen 149
4.1 Felder 150
4.1.1 Eindimensionale Felder 150
4.1.2 Beispiele zu eindimensionalen Feldern 150
Aufgaben 168
4.1.3 Mehrdimensionale Felder 169
Aufgabe 177
Lösung 177
4.2 Unterprogramme 179
Aufgabe 196
Lösung 197
4.3 Menütechniken 200
4.3.1 Verwendung von Eingabe-Routinen im Menü 205
4.3.2 Cursorpositionierung mit GOTOXY 212
4.4 Sortierverfahren 213

Kapitel 5:

5. Das Arbeiten mit dem Diskettenlaufwerk 217

5.1 Programmverwaltung 218
5.1.1 Speichern von Programmen 218
5.1.2 Laden von Programmen 219
5.1.3 Anzeigen des Disketteninhalts 220
5.1.4 Löschen von Files 221
5.1.5 Umbenennen von Files 222
5.2 Sequentielle Dateiverwaltung 222

Kapitel 6:

6. Musik und Grafik 227
6.1 Musik 228
6.2 Grafik 231

6.2.1 Linien 232
6.2.2 Kreise 233
6.2.3 Ellipsen 233
6.2.4 Ausgefüllte Flächen 234

Kapitel 7:

7. GEM-Funktionen unter Basic 237

7.1 GEM-Grundlagen 238
7.2 Paramaterübergabe an GEM-Routinen 239
7.2.1 VDI-Aufrufe 240
7.2.2 AES-Aufrufe 241
7.3 VDI-Beispiele 243
7.4 AES-Beispiele 244

Kapitel 8:

8. Lösungen der Aufgaben 247

Anhang

Befehlsübersicht 265

Einleitung

Mit dem ATARI 520 ST haben Sie eine überaus leistungsfähige BASIC-Maschine erworben. Wenn die im Lieferumfang enthaltene Begleitliteratur zum ST-BASIC nur annähernd die Qualität des Interpreters hätte, stände einer komfortablen Erstellung leistungsfähiger Programme nichts mehr im Wege. Dieses Buch soll nicht nur diese Lücke schließen, sondern fundamentale BASIC-Grundkenntisse von einfacher Bildschirmausgabe mit PRINT bis zu komplexeren Programmieralgorithmen wie z.B. das Sortieren vermitteln. Weiterhin macht das Buch auf vorhandene Fehler in der vorliegenden BASIC-Version aufmerksam.

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

zuerst 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 in das Prinzip der Dateiverwaltung und den Umgang mit dem Diskettenlaufwerk ein.

Der 6. Teil erklärt anhand von Beispielen einige Grafik- und Musikbefehle des Atari ST.

Der 7. Teil befaßt sich mit dem Aufruf von GEM-Funktionen innerhalb von BASIC-Programmen. Auch hier finden Sie wieder einige Beispiele.

Der 8. Teil schließlich ist der Lösungsteil, in dem die Lösungen ausführlich besprochen werden.

Wir weisen darauf hin, daß das Buch auf der BASIC-Version vom 18-07-85 (138944 Bytes) basiert. Abweichungen zu evt. später erscheinenden Versionen sind daher möglich.

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".

Ihre Autoren

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. Einige Computer haben eine Programmiersprache fest eingebaut. Beim Atari ST muß diese Sprache jedoch erst von einer Diskette in den Rechner geladen werden. Trotzdem können Sie dann 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, mit der wir uns befassen wollen, ist im Falle des Atari ST 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 Atari ST nicht direkt auf andere Computer anwendbar ist. So besitzen z.B. andere Rechner weniger leistungsfähige Befehle als der Atari ST. Auf dem Atari ST erstellte Programme müssen also im Hinblick auf die Basicversionen anderer Computer entsprechend angepaßt werden.

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**, welcher ja zuvor geladen werden muß. 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 $\mathbf{V=4PIr^3/3}$, und in den Computer gehämmert. Das Programm könnte dann ungefähr so aussehen:

```
10 PI=3.14159265
20 FOR I=1 TO 20
30 INPUT"WELCHER RADIUS (IN CM)";R
40 V=4*PI*R^3/3
50 PRINT"DAS VOLUMEN BETRAEGT ";V;" ccm"
60 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

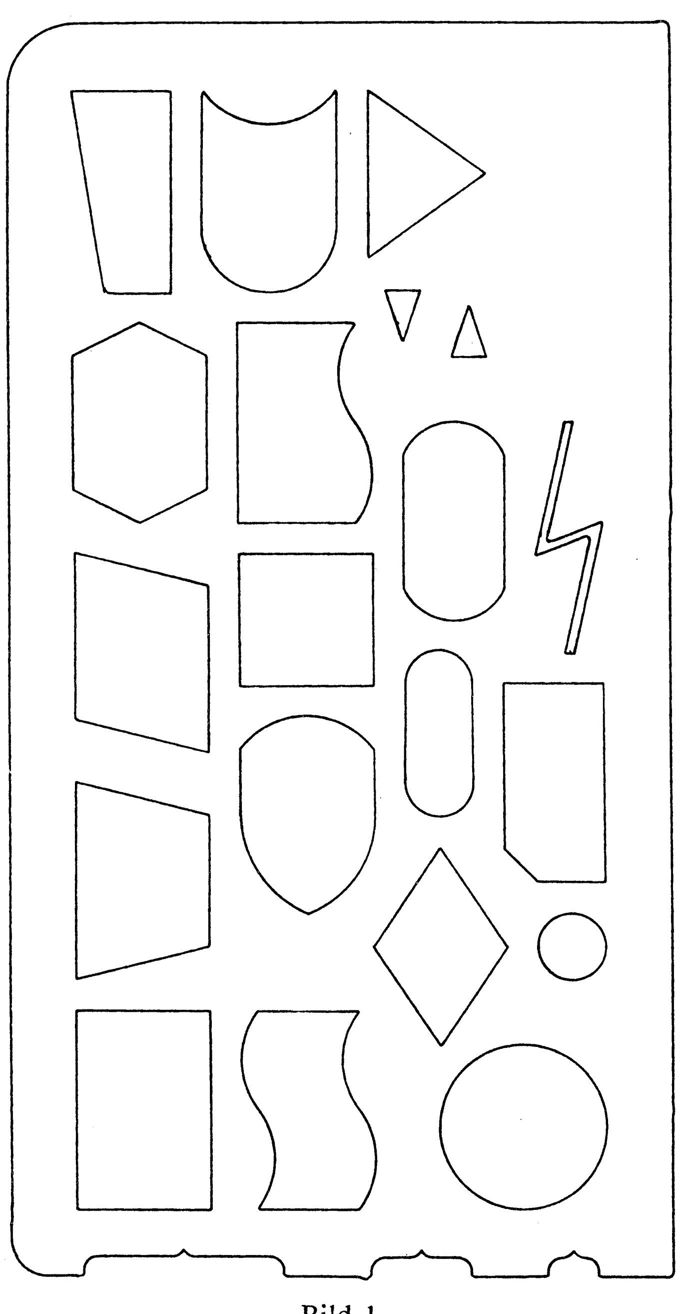


Bild 1

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.

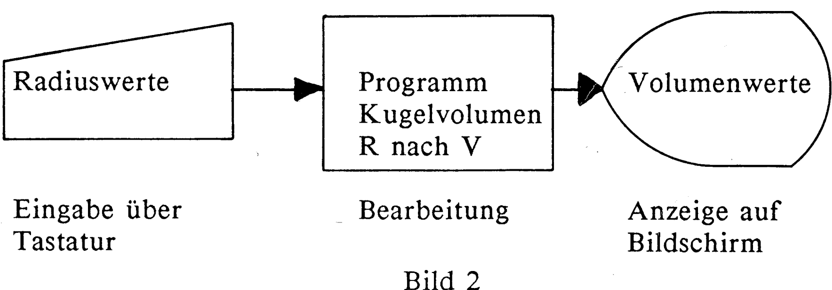


Bild 2

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.

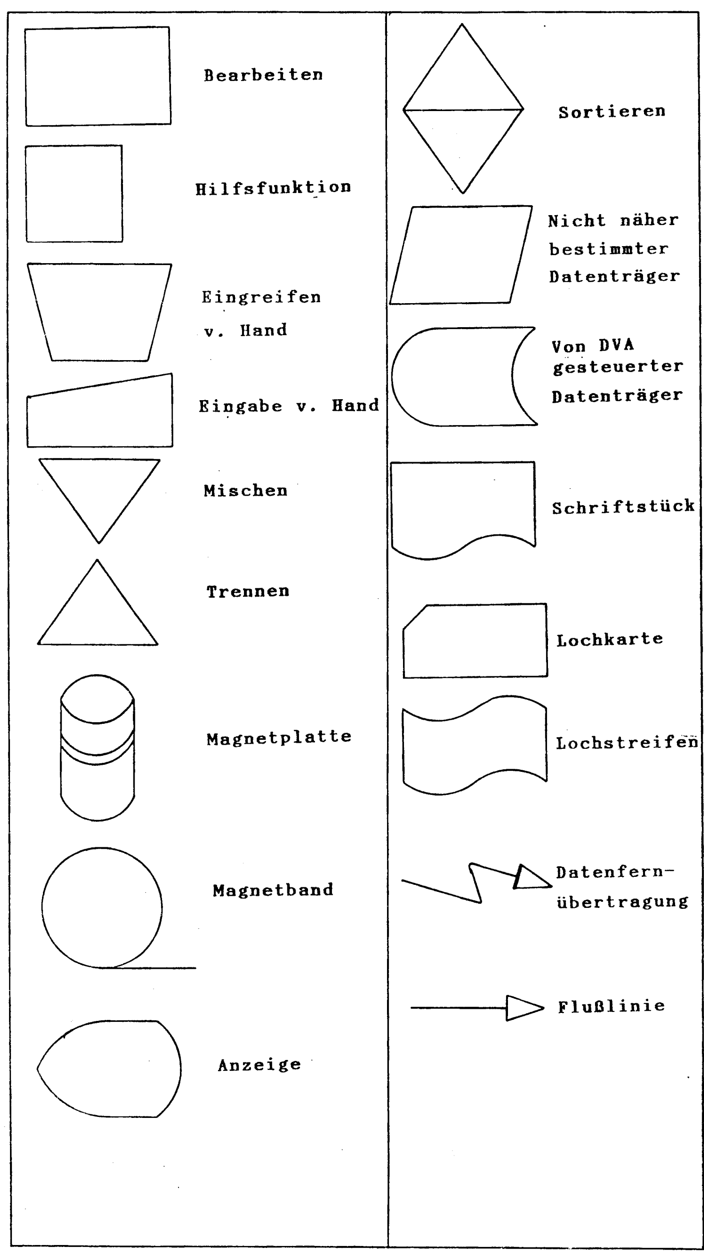


Bild 3

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.

Interne Verarbeitung	Logische Verzweigung
Ein- oder Ausgabe	Unterprogramm-aufruf
Grenzstelle	Kommentarsymbol
Konnektor	Ablauflinie

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 Programmierschablone zu finden sind. Diese Symbole sind in Bild 4 näher erläutert.

An unserem bekannten Beispiel wollen wir nun an die Erstellung unseres ersten Programmablaufplans gehen. Selbst bei solch kleineren Programmen sollte man ruhig dazu übergehen, sich PAPs zu erstellen, damit man nicht später bei komplexeren Programmen Schwierigkeiten mit der Umsetzung bekommt. Auch hier gilt der alte Spruch "Übung macht den Meister".

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

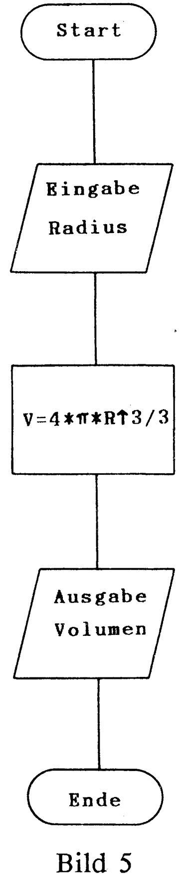


Bild 5

Das Start- bzw. Ende-Symbol kann man nicht in Basic übersetzen. Das Eingabesymbol "EINGABE RADIUS" kann man mit dem BASIC-Befehl INPUT übersetzen. Dieser kann noch mit einem Kommentar versehen werden wie

"WELCHER RADIUS IN CM".

Die Formel für die Berechnung des Kugelvolumens kann direkt in das Symbol für die interne Verarbeitung übernommen werden. Für das Ausgabesymbol "Ausgabe Volumen" benutzen wir den PRINT-Befehl, der mit einem entsprechenden Text versehen wurde. Im Gegensatz zu unserem 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 zu 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.

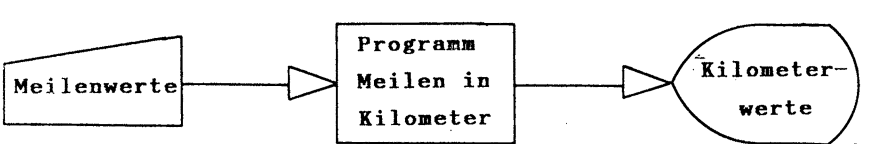
1. *Definition des Problems (Erarbeiten der Problemstellung, Problemanalyse)*

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

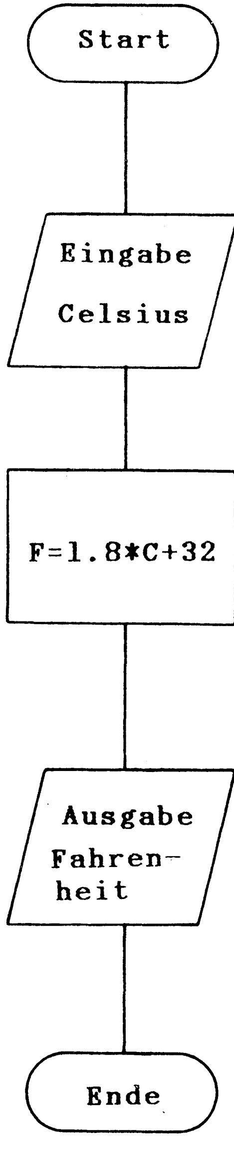


Bild 7

1.4 ASCII-Codes

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

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

Beim Standard-ASCII-Code werden die Zahlen 32-90 für Großbuchstaben und die Zahlen 91-127 für Kleinbuchstaben und einige andere Zeichen verwendet. Der ASCII-Code des ATARI ST ist nur im Bereich zwischen den Zahlen 32-91 und noch mit wenigen anderen Zahlen mit dem Standard-ASCII-Code identisch.

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 $\mathbf{1011}_2$ 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 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**. Alle **acht Bits** zusammengefaßt nennt man **BYTE**.

1.5.3 Das Hexadezimalsystem

Im **Hexadezimalsystem** verwendet man als **Basis** die Zahl **16**. Somit benötigt man auch **15 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

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:

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**.

$2\ E\ 0\ C_{16}$

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

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

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

$0\ A\ B\ C_{16}$

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

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

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**. Die zweistellige Hexadezimalzahl dürfte dann leicht in eine Dezimalzahl zu überführen sein.

> **Anmerkung**: Diese Hälften zu je **vier Bits** nennt man auch **NYBBLE** oder **NIBBLE** (beide Schreibweisen sind gebräuchlich).

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 des Atari ST stellt Ihnen die **drei Grundverknüpfungen**

NOT, AND u. 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 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

[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.

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

00001010.

Diese zwei Bitmuster werden nun mit AND verküpft.

```
    00010111
            > AND
    00001010

  = 00000010 = 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

00001010.

Diese zwei Bitmuster werden nun mit OR verküpft.

```
00010111
         > OR
00001010

= 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 Funktionen **XOR** (eXklusives **OR**), **EQV** (Äquivalenz) und **IMP** (**IMP**likation) besprechen, da diese in ihrer Wirkungsweise unmittelbar mit diesem Kapitel zu tun haben.

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 verknüpfenden **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 0 XOR 0,0 XOR 1,1 XOR 0,1 XOR 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.

1.6.5 EQV

Der logische Operator EQV ist quasi die Negation des Operators XOR. Damit sieht die Tabelle für EQV wie folgt aus:

Operator	Wert 1	Wert 2	Ergebnis
EQV	0	0	-1
	0	-1	0
	-1	0	0
	-1	-1	-1

Somit erhält man bei **EQV** nur dann ein **wahre Aussage**, wenn die **beiden** zu verknüpfenden **Aussagen** einen **gleichen Wahrheitsgehalt** besitzen.

1.6.6 IMP

Der logische Operator IMP erzeugt in drei Fällen eine wahre und in nur einem Fall eine falsche Aussage, so daß die Tabelle für diesen Operator wie folgt aussieht:

Operator	Wert 1	Wert 2	Ergebnis
IMP	0	0	-1
	0	-1	-1
	-1	0	0
	-1	-1	-1

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 am Ende 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 ATARI ST 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:

$$S = 4PIr^2$$

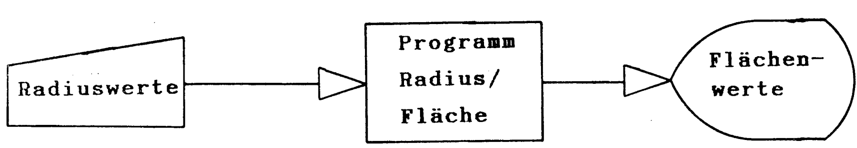
2. Entwurf des Algorithmus' zur Lösung

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

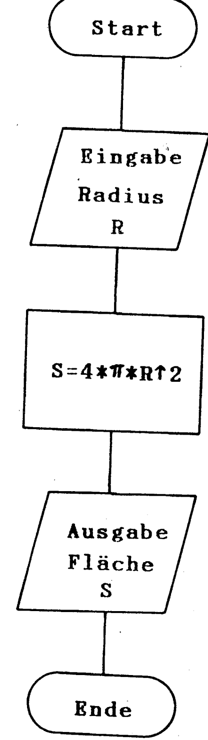
Im folgenden sehen Sie den dazu gehörenden Datenflußplan sowie den Programmablaufplan. Dieser Programmablaufplan zählt zu den **linearen** Programmablaufplänen, d.h. es finden keine Verzweigungen in Form von Unterprogrammen oder Schleifen

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



Programmablaufplan



3. Erstellen des Programms

```
10 INPUT"WELCHER RADIUS IN CM";R
20 LET S=4*3.14159*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 erscheinen im Output-Fenster ein Fragezeichen und 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.

Eine weitere Form des INPUT-Befehls ist der Befehl

LINE INPUT.

Er unterscheidet sich vom INPUT-Befehl eigentlich nur darin, daß man der folgenden Variablen auch das Komma übergeben kann. Weiterhin kann dem LINE INPUT nur **eine** Variable zugeordnet werden, wogegen dem INPUT-Befehl eine Variablenliste folgen kann. Weiterhin dürfen beim LINE INPUT nur Text- bzw. Stringvariablen benutzt werden. Ein Fragezeichen wird nicht automatisch mit ausgegeben.

Wichtig:

Dem LINE INPUT-Befehl kann nur eine Stringvariable zugeordnet werden. Bei der Ausgabe erscheint kein Fragezeichen. Komma und Semikolon können mit übergeben werden.

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 Atari ST 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 des ATARI ST. 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.

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 ATARI ST 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 im Output-Fenster 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 Sonderzeichen, z.B. für die ESC- oder TAB-Taste. Verwenden Sie also innerhalb der Anführungszeichen eine diese Tasten, so erscheint zwar **bei der Eingabe** ein Zeichen, aber diese **Zeichen** werden **nicht** bei der Ausgabe im **Output-Fenster sichtbar.**

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 Atari ST hat jede Ausgabezeile nochmals in Bereiche zu je 14 Zeichen unterteilt, sogenannte Tabulatoren. Wird nun das Komma zwischen zwei Variablen verwendet, so wird die zweite Variable am Anfang des zweiten Tabulators ausgegeben, also ab der 15. 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 der PRINT USING-Anweisung folgende Steuerzeichen zur Verfügung:

Für numerische Ausgaben:

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.

** statt mit Leerzeichen, wird mit * aufgefüllt.

$$ als erstes Zeichen wird $ ausgegeben.

**$ kombinierter Einsatz von $ und *.

, jede dritte Stelle vor dem Dezimalpunkt wird durch Komma abgetrennt.

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

_ unterdrückt für das nächste Zeichen Steuerfunktion, z.B. für #.

Für Textausgaben:

! es wird nur das erste Zeichen der Stringvariablen ausgegeben.

\ \ es werden so viele Zeichen des Textes ausgegeben, wie Leerzeichen plus der beiden Schrägstriche angegeben werden.

& es wird die komplette Stringvariable 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 = -520 *<RETURN>*
D$ = "Atari ST" *<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 wird vor der Zahl ein Prozentzeichen ausgegeben. Das angegebene Format kann hierbei nicht berücksichtigt werden. So würde die folgende PRINT USING-Anweisung

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

diese *Ausgabe* verursachen:

```
%12345.60
```

Durch diese Ausgabe gibt Ihnen der Atari ST 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:

520.00-

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

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

PRINT USING "#####.##";A**

Ausgabe:

****12345.68**

PRINT USING "#####.##";B**

Ausgabe:

*****34.35

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

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

Ausgabe:

$12345.68

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

Ausgabe:

$34.35

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

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

Ausgabe:

*******$34.35**

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

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

Ausgabe:

****12,345.68 DM**

Wollen Sie die Zahlenwerte in der Exponentialschreibweise ausgeben lassen, so dient dazu die folgende Form 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:

12.34E+03

Das '_'-Zeichen wird benutzt, um auch Steuerzeichen wie '#' mit ausgeben zu können.

PRINT USING "DATEINUMMER _###";B

Ausgabe:

DATEINUMMER #34

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

PRINT USING "!";D$

Ausgabe:

A

Bei Benutzung des Ausrufezeichens wird also nur das erste Zeichen der Text- bzw. Stringvariablen 'D$' ausgegeben. Wir

erinnern uns, daß wir 'D$' die Zeichenkette "Atari ST" übergeben hatten.

Die nächste Form erlaubt es uns, eine beliebige Anzahl von Zeichen einer Zeichenkette ausgeben zu lassen.

PRINT USING "\   \";D$

Ausgabe:

Atari

Oben wurden zwischen den Anführungszeichen insgesamt 5 Zeichen eingegeben (einschließlich der beiden Schrägstriche). Somit werden auch von der Textvariablen 'D$' nur die ersten 5 Zeichen ausgegeben. Diese Anwendung von PRINT USING ist verwandt mit der LEFT$-Funktion, die zu einem späteren Zeitpunkt besprochen wird. Diese spezielle Art des Schrägstrichs erhalten Sie übrigens durch die Betätigung der Tastenkombination *SHIFT*, *ALTERNATE* und *Ü*.

Zum Schluß wollen wir noch die Möglichkeit besprechen, wie man sich eine Textvariable komplett ausgeben lassen kann.

PRINT USING "&";D$

Ausgabe:

Atari ST

Dies könnte man auch durch

PRINT D$

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

Der Vollständigkeit halber sei hier noch der Befehl

WRITE

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

WRITE "Atari ST"

und betätigen die RETURN-Taste, so wird

"Atari ST"

im Output-Fenster ausgegeben. Die Anführungszeichen werden also im Gegensatz zum PRINT-Befehl mit angezeigt. Folgen Zahlen oder Variablen durch Kommata getrennt dem WRITE-Befehl, so hat das Komma keine Tabulatorfunktion, sondern wird mit angezeigt.

WRITE 2,3,4

Ausgabe:

2, 3, 4

Damit hätten wir soweit 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*3.1415927*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 BASIC des Atari ST 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 -32767 und 32767 zugeordnet werden.

Der zweite Variablentyp, die *REAL-Variable*, wird benutzt, um Dezimalzahlen darstellen zu können. Die dabei verwendeten Variablenbezeichnungen können als Zusatz das Ausrufezeichen erhalten, unbedingt notwendig ist es aber nicht. Erlaubte Bezeichnungen sind z.B. **A!** oder **B2!** (**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

Strings cannot be over 255 characters long

ausgegeben wird.

Bei der Verwendung der Bezeichnungen von Variablen muß allerdings einiges berücksichtigt werden. Der Atari ST erkennt eine Variable nur an den **ersten 31 Zeichen**. Sie dürfen also durchaus einer Variablen die Bezeichnung 'Donaudampfschiffahrtsgesellschaft' geben, wenn Sie dies für unbedingt notwendig erachten. Allerdings bleiben nur die ersten 31 Zeichen signifikant. Viele Microcomputer können solche langen Variablennamen nicht unterscheiden. Durch diese Eigenschaft des Atari ST BASIC werden Programme leichter verständlich und lesbar, da den **Variablen sinnvolle Bezeichnungen** gegeben werden können wie z.B. '*Zahlung*', '*Wechsel*' oder '*Name*'. Auch dürfen sich in den Variablen Basic-Schlüsselwörter befinden. Die Bezeichnung '*Wand*' wäre **zulässig**, obwohl sich in ihr der logische Operator AND befindet.

Weiterhin dürfen zusätzlich Ziffern zur Kennzeichnung benutzt werden, allerdings mit der Einschränkung, daß diese erst an zweiter Stelle auftreten dürfen. Erlaubt sind Bezeichnungen wie *Betrag1*, *A9* oder ähnliche. Es ist nicht erlaubt, die Zahl an die erste Stelle zu setzen.

Z.B. ist 9Betrag **nicht zulässig!**

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

OR, FN oder ABS.

Der ATARI ST benutzt außerdem folgende Variablenbezeichnungen für interne Funktionen:

GB, AS, ALL

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

2.2.1 Rechenoperationen mit Variablen

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

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

Auch für die logischen Operatoren existiert 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)	AUTO
d)	ODER%	e)	IF	f)	GB
g)	4NAME%	h)	255	i)	MONTAG

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

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

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

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

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

2.3 Numerische Funktionen

Bisher haben wir uns mit einfachen Wertzuweisungen von Variablen befaßt, d.h. es wurden nicht die vorhandenen mathematischen Funktionen des ATARI ST 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 ATARI ST 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
25 PI=3.1415927
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. Da im Basic des Atari ST die Variable PI nicht direkt verfügbar ist, muß diese zuerst in Zeile 25 definiert werden. 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.

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
```

Wie Sie sehen, ist gegenüber unserem vorigen Programm die Zeile 25 weggefallen; die Zeile 30 ist etwas verändert worden. Starten Sie nun das Programm und geben Sie den Wert 1.57079633 (entspricht PI/2) ein. Als Ergebnis erhalten Sie wieder Null.

Die Anwendung der anderen Funktionen ist denkbar einfach. Wie 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

2.71828,

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

PRINT LOG(2.71828183)

Nun erhalten Sie wiederum die 1. Wollen Sie den Logarithmus zur *Basis 10* berechnen, so müssen Sie nur **LOG(X) durch LOG10(X) ersetzen**. Der Logarithmus zur *Basis 10* nennt sich auch "*dekadischer Logarithmus*" und der Logarithmus zur *Basis e* "*natürlicher Logarithmus*".

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

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

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

WICHTIG:

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

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

Die Funktion SGN(X) ergibt das Vorzeichen von X. Das Ergebnis ist 1, wenn X positiv ist, 0, wenn X=0 und -1, wenn X negativ

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

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

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

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

Mit der Funktion 'CINT' können Sie ebenfalls Zahlen runden. Jedoch dürfen diese Zahlen sich nur im Intervall von -32768 bis +32767 befinden.

PRINT CINT(-35.6)

Ausgabe:

-36

Die Funktionen **'CSNG'** und **'CDBL'** wandeln Variablen doppelter Genauigkeit (4 Bytes) in Variablen einfacher Genauigkeit (2 Bytes) um und umgekehrt.

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

PRINT 32 MOD 7

ein, so erhalten Sie als Ausgabe den Wert 4 *(32=4*7 <u>Rest 4</u>)*.

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 ATARI ST 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 sich bei einer unteren Grenze von 1, der Ausdruck im ersten Teil der Formel vereinfacht.

Wollen Sie dem Zufallszahlengenerator einen neuen Startwert (Seed) übergeben, so steht Ihnen im BASIC des Atari ST der Befehl

RANDOMIZE

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

RANDOMIZE 3

Geben Sie diesen Befehl ein, so wird dem Zufallszahlengenerator der neue Startwert 3 zugeordnet. Jetzt können Sie mit

A=RND(4)

der Variablen 'A' eine neue Zufallszahl zuordnen.

2.3.3 Noch mehr Befehle für Variablen

Das umfangreiche BASIC des Atari ST besitzt eine Menge an Befehlen, mit denen Variablen bzw. Variablenformate beeinflußt werden können. Diese Befehle bzw. Funktionen sollen im folgenden nun kurz besprochen werden.

Sie erinnern sich bestimmt noch an das Kapitel über die Zahlensysteme. Der Atari ST bietet dem Benutzer zwei Funktionen, die es ihm ermöglichen, dezimale Zahlen in das hexadezimale bzw. oktale 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 -32768 und +32767* annehmen.

Beispiel:

PRINT HEX$(60)

Ausgabe:

3C

Die Funktion

OCT$(X)

wandelt Dezimalzahlen in Zahlen des Oktalsystems (Basis 8) um. 'X' steht hier wiederum für die umzurechnende Zahl. Dabei müssen die *X-Werte im Bereich zwischen -32678 und +32767* der Funktion übergeben werden.

Beispiel:

PRINT OCT$(16)

Ausgabe:

20

In beiden Beispielen können statt Konstanten auch Variablen eingesetzt werden.

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

DEFDBL, DEFINT, DEFSNG, DEFSTR

Definieren Sie in einem Programm z.B.

DEFSTR A-B,

so haben Sie damit alle Variablen, die mit '**A**' oder mit '**B**' beginnen, als *Stringvariable* festgelegt. Sie brauchen dann innerhalb des Programms nicht jedesmal ein "$"-Zeichen an den Variablennamen anzuhängen. Die Anweisung

DEFSNG C-D

definiert alle Variablen, die mit '**C**' oder '**D**' beginnen, als Realvariable. Analog dazu sind die anderen Befehle zu gebrauchen.

Wollen Sie bei der Ausgabe von Zahlenwerten nur den Zahlenteil vor dem Dezimalpunkt ausgeben lassen, so können Sie dazu den Befehl

FIX

benutzen. FIX arbeitet ähnlich wie der INT-Befehl, nur mit dem Unterschied, daß FIX grundsätzlich den Vorkommateil abtrennt, ohne eine Rundung vorzunehmen.

PRINT FIX(-3.99)

Ausgabe:

-3

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

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

Der ATARI ST besitzt die Möglichkeit, durch den PRINT-Befehl Zahlen, Buchstaben und bestimmte Sonderzeichen, wenn diese zwischen den Anführungszeichen stehen, auf dem Bildschirm ausgeben zu lassen.

Nun können aber nicht sämtliche Zeichen mit dieser Methode ausgegeben werden. Abhilfe schafft hier die CHR$-Funktion. Mit dieser Funktion haben Sie die Möglichkeit, jedes Zeichen des Zeichensatzes ausgeben zu lassen. Geben Sie folgendes im Direktmodus ein:

PRINT CHR$(65)

Betätigen Sie die RETURN-Taste, so erscheint auf dem Bildschirm jetzt das Zeichen A.

Die Funktion ASC stellt nun die Umkehrung zur CHR$-Funktion dar. 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. Im Handbuch zum Atari ST können Sie die entsprechenden ASCII-Werte nachschlagen.

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 Aufgaben auf der nächsten Seite.

Aufgaben

1. Schreiben Sie ein Programm, das das Würfeln mit zwei Würfeln simuliert. Die Ergebnisse sollen getrennt in einer Zeile tabelliert ausgegeben 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. **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. 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. Geben Sie 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 Atari ST 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$="Atari ST"

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

Types of values do not match

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 Atari ST 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. 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 nicht nur Zeichen innerhalb eines Strings auslesen lassen, sondern auch ganz bestimmte Zeichen abändern bzw. neu zuordnen. Schreiben Sie z.B.

MID$(A$,7,1)="U"

so ändern Sie das siebte Zeichen in ein "U" um. Es heißt dann nicht mehr "DONAUDAMPF..." sondern "DONAUDUMPF...". Dieses Beispiel sollte vorerst zur Verdeutlichung der Funktionsweise des MID$ Befehls ausreichen.

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 die Funktion, die 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 Leerzeichen beinhaltet. Ist die Zahl positiv, so handelt es sich dabei um ein Leerzeichen. Zwei Beispiele sollen das wieder verdeutlichen.

Beispiele:

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

Ergebnis: 1234

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

Ergebnis: -1234

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

2.5.7 INSTR

Eine weitere nützliche Funktion für den Umgang mit Strings bietet uns das BASIC des Atari ST mit

INSTR

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

INSTR(X,A$,B$)

Dabei steht 'X' für die Position, ab der der String 'A$' durchsucht werden soll. 'B$' 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:

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 STRING$

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

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

so erhalten Sie als Ausgabe eine Zeile mit 40 Sternchen. Der erste Parameter bestimmt, wie viele Zeichen der String beinhalten soll. Auch hier liegt die obere Grenze wieder bei 255.

2.5.9 SPACE$

Die Funktion SPACE$ ist in der Wirkung ähnlich der String$-Funktion. Mit

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

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

```
PRINT SPACE$(12);"Atari ST"
```

Ausgabe:

```
            Atari ST
```

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)"TEST1" TAB(15)"TEST2"

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

a) Der einzige Unterschied liegt darin, daß bei der ersten Befehlsfolge der String "Test1" ein Zeichen weiter rechts ausgegeben wird.

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

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

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

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

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

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

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

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 Editor (Edit-Fenster), und wie man damit z.B. einzelne Zeichen löscht oder einfügt, wird hier allerdings als bekannt vorausgesetzt. Sollten Sie trotzdem noch Schwierigkeiten damit haben, so lesen Sie bitte im Handbuch des Atari ST 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 10,10** in den Rechner und betätigen Sie die RETURN-Taste, so haben Sie damit die automatische Zeilennumerierung in Zehnerschritten eingeschaltet. Der erste Wert, der dem Befehl AUTO mit übergeben wird, gibt die erste Programmzeile an, und der zweite Wert bestimmt das Inkrement, also den Abstand zu den einzelnen Programmzeilen.

Wollen Sie die automatische Zeilennumerierung wieder aufheben, so betätigen Sie die Tasten *Control G*.

Schreiben Sie ein Programm, so benötigen Sie früher oder später den

RENUM

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

10 ....
12 ....
15 ....
19 ....
20 ....
21 ....

Geben Sie jetzt vom Command-Fenster RENUM ein, so werden alle Programmzeilen in Zehnerschritten neu durchnumeriert, so daß Sie im obigen Beispiel jetzt die Zeilennummern von 10 bis 60 hätten. Die Leistungsfähigkeit dieses Befehls zeichnet sich besonders dadurch aus, daß auch Programmsprünge mit den Befehlen *GOTO*, *GOSUB* usw. mit berücksichtigt, d.h. neu berechnet werden.

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

RENUM X,Y,Z

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

RENUM 200,100,5

würde das Programm ab der alten Zeilennummer 100 in Fünferschritten, beginnend mit der neuen Zeilennummer 200, durchnumerieren. Der RENUM-Befehl legt eine Datei mit Namen '*BASIC.WRK*' an. Achten Sie darauf, daß die im Laufwerk befindliche Diskette nicht schreibgeschützt ist.

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

Stop at line *(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 im Command-Fenster ausgegeben. Der Befehl

TROFF

(engl.=*TR*ace *OFF*) schaltet diese Hilfe wieder aus.

Der Befehl

TRACE

zeigt jede Programmzeile, die gerade vom Programm bearbeitet wird, an. Durch Angabe einer Zeilennummer können Sie auch nur bestimmte Zeilen anzeigen lassen. So wird durch

TRACE 40

nur die Zeile 40 ausgegeben. Sie haben bei TRACE die gleichen Kombinationsmöglichkeiten wie beim DELETE- oder LIST-Befehl.

Durch den Befehl

UNTRACE

wird diese Hilfe wieder ausgeschaltet.

Wollen Sie wissen, welche Variablen in den einzelnen Programmzeilen welchen Inhalt haben, so können Sie hierfür den

FOLLOW

Befehl einsetzen. *FOLLOW A* zeigt Ihnen z.B. während des Programmablaufs den Inhalt der Variablen A mit Zeilennummer an.

Durch die Eingabe von

UNFOLLOW

können Sie den Befehl rückgängig machen.

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. Mit **LLIST** können Sie das Programm auf dem Drucker ausgeben lassen.

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

CLEAR

löscht alle Variablen und Arrays (Felder). Sollte Ihnen der Begriff 'Array' noch nicht geläufig sein, so ist das momentan nicht weiter von Bedeutung. In einem späteren Kapitel wird dieser Begriff ausführlich erklärt.

Wollen Sie nur Felder bzw. ein Feld löschen, so müssen Sie dazu den

ERASE

Befehl verwenden. Mit

ERASE A

wird z.B. das vorher mit DIM A(20) erzeugte Feld gelöscht. Anschließend kann das Feld neu dimensioniert werden.

Durch die Eingabe von

EDIT

schalten Sie direkt um in das EDIT-Fenster.

Mit diesen Editierbefehlen steht Ihnen also ein komfortables Hilfsmittel für die Erstellung und Korrektur Ihrer Programme zur Verfügung.

2.7 Die Bildschirmfenster

Die einzelnen Bildschirmfenster (*Edit*, *List*, *Output* und *Command*) können Sie auch von Basic aus schließen, öffnen und löschen. Im folgenden sollen nun die einzelnen Befehle kurz besprochen werden. Den einzelnen Fenstern sind die folgenden Zahlen zugeordnet:

0 = Edit-Fenster
1 = List-Fenster
2 = Output-Fenster
3 = Command-Fenster

Der Befehl

FULLW 2

setzt das Output-Fenster auf die volle Bildschirmgröße.

Wollen Sie den Inhalt eines Fensters löschen, so benutzen Sie dazu den Befehl

CLEARW

CLEARW 3 löscht z.B. den Inhalt des Command-Fensters.

Der Befehl

OPENW

öffnet ein zuvor geschlossenes Fenster.

Mit

CLOSEW

können Sie ein Fenster schließen, d.h. das entsprechende Fenster verschwindet vollkommen vom Desktop.

V O R S I C H T !

Schließen Sie **alle** *Fenster, so reagiert der Rechner auf keine Eingabe mehr. Sie sind dann gezwungen, das System neu zu starten.*

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 stellt wohl der **GOTO**-Befehl dar. 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.

Der Nachteil dieser unbedingten Programmsprünge ist wiederum, daß man mit ihnen eigentlich nur sogenannte Endlosschleifen erzeugen kann. Wurde das Programm dann einmal gestartet, so hat man nur noch die Möglichkeit, es durch *Control G* oder durch 'anklicken' von BREAK 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.

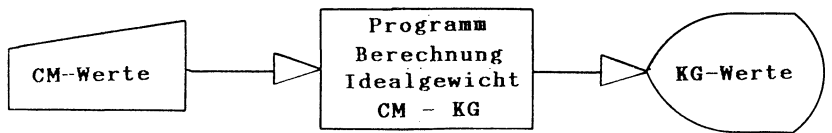
Stellen Sie sich nun vor, Sie geben eine Party und wollen als kleinen Gag dieses Programm vorführen. Jeder der Gäste soll sein Idealgewicht erfahren. Ohne den **GOTO**-Befehl müßte das Programm bei jeder neuen Berechnung erneut gestartet werden. Daher ändern wir das Programm dahingehend ab, daß vor der 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. Die Zeile 80 hätte man nicht einzugeben brauchen, da das Programm ja durch den GOTO-Befehl diese Zeile nie erreicht. Der Datenflußplan wird durch diesen Befehl nicht beeinträchtigt. Das untere Schaubild soll zeigen, wie er auszusehen hätte.

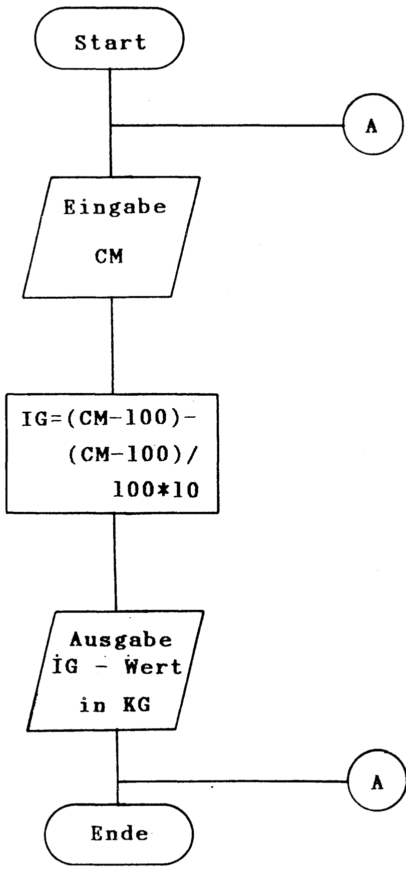
Datenflußplan für Berechnung Idealgewicht



Der Programmablaufplan PAP ändert sich durch diesen Befehl wie im unteren Schaubild dargestellt. Ein Symbol ist hinzugekommen. Es handelt sich dabei um den **Konnektor**. Er bezeichnet die Stelle, 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



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 CLEARW 2:PRINT
30 PRINT"GEBEN SIE ZWEI ZAHLEN FUER "
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 CLEARW 2: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. 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. 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 Sprungmarken / Labels

Sie haben im BASIC des Atari ST die Möglichkeit, Sprungmarken bzw. Labels zu vergeben. Diese Labels können dann mit GOTO oder auch GOSUB angesprungen werden. Sie brauchen sich dadurch sich nicht sämtliche Programmzeilen der einzelnen Routinen in Ihrem Programm zu merken, sondern geben statt

50 GOTO 250

einfach ein:

50 GOTO Ausdruck

Die entsprechende Zeile 250 müßte dann so aussehen:

250 Ausdruck:

Der Name der Sprungmarke bzw. des Labels muß mit einem Doppelpunkt schließen. Sie haben durch die Vergabe von Labels die Möglichkeit, einzelne Programmteile mit Namen zu versehen, und so Ihr Programm leichter lesbar und damit übersichtlicher zu machen. Wollen Sie z.B. in Ihrem Programm eine Sortierroutine aufrufen, brauchen Sie nicht erst umständlich nach der Zeilennummer zu suchen, sondern können dann einfach eingeben:

GOTO Sortier

oder

GOSUB Sortier

Die Namen der Labels dürfen nicht mit einer Zahl beginnen, sondern benötigen als erstes Zeichen immer einen Buchstaben.

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 CLEARW 2:PRINT
30 PRINT "AUSGABE DER ERSTEN 10 QUADRATZAHLEN"
40 FOR I = 1 TO 10
50 PRINT "QUADRATZAHL VON";I;"=";I*I
60 NEXT I
70 PRINT"ENDE"
```

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

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

Dazu einige Beispiele:

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

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

IF Z > 20 THEN 50

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

PRINT Z

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

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

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

Will man, daß die Schleife eine andere Schrittweite (*INKREMENT*) als 1 annimmt, so muß man zusätzlich durch **STEP die Schrittweite bestimmen.** Das folgende Beispiel ergibt in

Schritten von 2 die Ausgabe der geraden Zahlen zwischen 2 und 20.

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

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

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

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

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

Geben Sie dieses Programm ein (natürlich ohne die nebenstehenden Pluszeichen) und starten Sie 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.

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.

FALSCH

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

FALSCH

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:

You are trying to jump into a loop at line *(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 sofort verlassen, 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 für A keine Ausgabe. 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 nicht durchlaufen wird. Das gleiche gilt für negative Schrittweiten.*

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

Diese Regeln beziehen sich nur auf das BASIC des Atari ST. 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 **WHILE...WEND**-Schleife.

3.2.4 Schleifen mit WHILE...WEND

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

Der Beginn der Schleife wird mit **'WHILE'** und das Ende der Schleife mit **'WEND'** gebildet. Der logische Ausdruck, der dem 'WHILE' folgt, wird bei jedem Schleifendurchlauf überprüft. Solange der Ausdruck 'Wahr' ist, wird die Schleife bis 'WEND' durchlaufen. Trifft die Bedingung hinter WHILE nicht mehr zu, wird mit der Ausführung des Programms nach dem WEND fortgefahren. Sie können mit dieser Befehlskombination eine Schleife rein zufällig abbrechen lassen, wie das folgende kleine Beispiel zeigt:

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

Hier wird die WHILE...WEND-Schleife solange durchlaufen, bis A zufällig den Wert 100 erreicht. Wird die Aussage in Zeile 40 wahr, dann wird der nächste Befehl, der dem WEND 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. Ich hoffe, daß Ihnen durch dieses kleine Beispiel die Wirkungsweise der WHILE...WEND-Schleife klarer geworden ist.

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

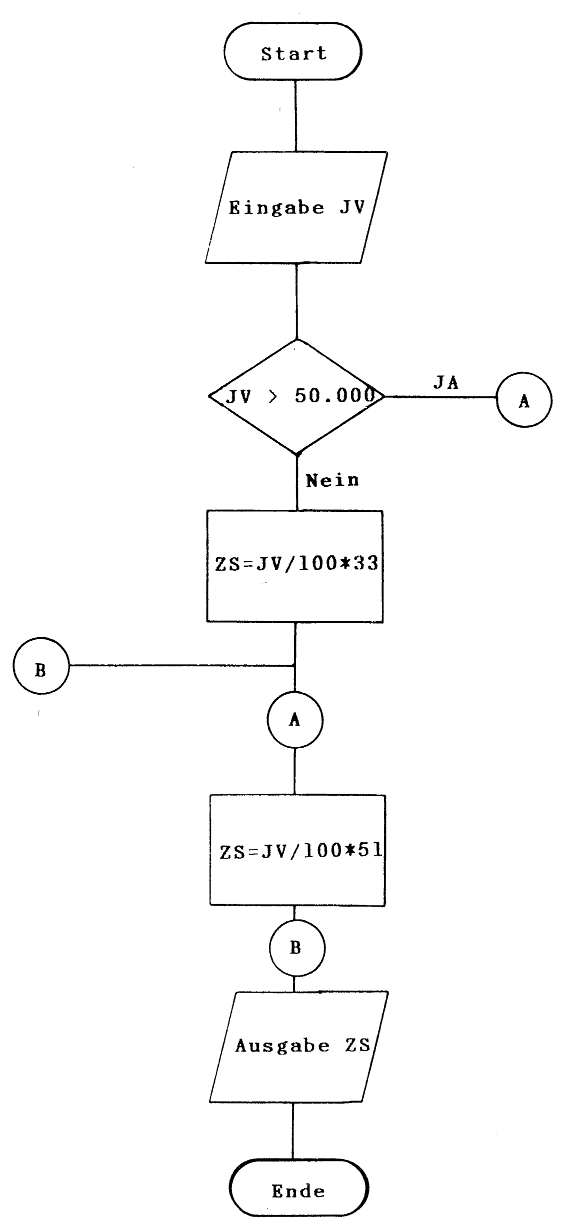
1. *Steht die Anzahl der Wiederholungen der Schleifen von Anfang an fest, verwenden wir die FOR...NEXT-Schleife.*

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

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"

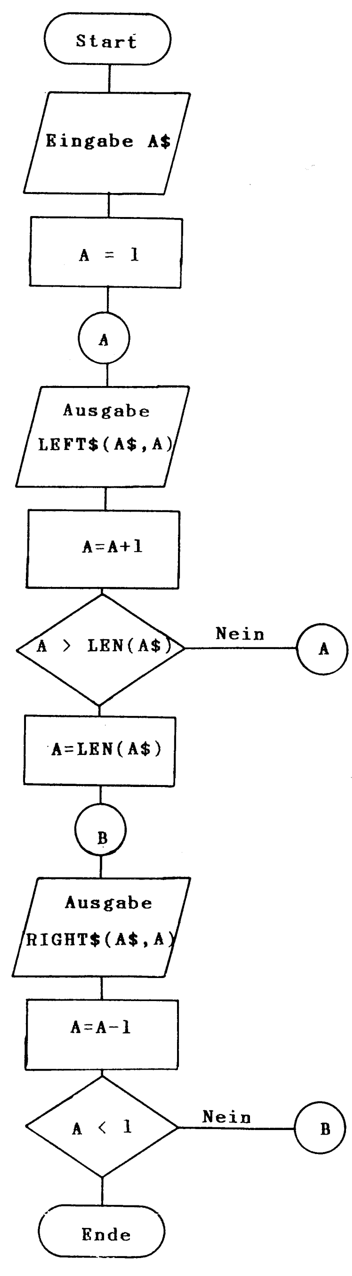


Das Startsymbol sowie das Eingabesymbol in unserem PAP dürften hinreichend bekannt sein. Die Raute ist, wie bereits erwähnt, das Symbol für eine logische Verzweigung. Sie besitzt einen sogenannten JA-Arm und einen NEIN-Arm. Trifft die Bedingung zu, so wird über den Absprungkonnektor A nach dem dazugehörigen 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 114



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 "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 oder Label)*

Dieser erweiterte GOTO-Befehl mit ON veranlaßt, daß das Programm zu einer von mehreren Zeilennummern oder Labels, 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 werden nicht berücksichtigt.

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 Atari ST 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.

Nachdem Sie eine formatierte Diskette in das Laufwerk gelegt haben, geben Sie in den Rechner folgenden Befehl ein:

SAVE LEHRGANG

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- oder OLD-Befehl. Ersetzen Sie einfach an den entsprechenden Stellen SAVE durch LOAD, und das Programm wird in den Rechner geladen.

Im folgenden nun das Programmlisting für den RECHENLEHRGANG.

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

```
260 PRINT TAB(10)"FUER DIE ADDITION EIN."
270 PRINT
290 PRINT TAB(10);:INPUT"GROESSTE";GR
299 REM
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM BERECHNUNG ERGEBNIS
331 REM
340 ER=A1+A2
350 CLEARW 2
360 PRINT
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2  "= ";
380 INPUT ES
390 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0: GOTO450
400 PRINT:PRINT TAB(10)"FALSCH !"
410 FOR I=0 TO 2000:NEXT I
420 F=F+1
430 IF F<=2 THEN 350
440 PRINT
450 FOR I=0 TO 2000:NEXT I
460 PRINT TAB(5)"DAS ERGEBNIS LAUTET" ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
490 INPUT A$
500 IF A$="J" THEN F=0:GOTO 300
510 GOTO 10
600 REM *************
610 REM  SUBTRAKTION
620 REM *************
630 CLEARW 2
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 CLEARW 2
760 PRINT
770 ER=A1-A2
780 PRINT"WIEVIEL ERGIBT" A1 "-" A2 "= ";
790 INPUT ES
800 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0:GOTO860
810 PRINT:PRINT TAB(10)"FALSCH !"
820 FOR I=0 TO 2000:NEXT I
830 F=F+1
840 IF F<=2 THEN 750
850 PRINT
860 FOR I=0 TO 2000:NEXT I
870 PRINT TAB(5)"DAS ERGEBNIS LAUTET";ER
880 FOR I=0 TO 3000:NEXT I
890 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
900 INPUT A$
910 IF A$="J" THEN F=0:GOTO 710
920 GOTO 10
1000 REM **********
1001 REM  DIVISION
1002 REM **********
1010 CLEARW 2
1020 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
1030 PRINT
1040 PRINT TAB(10)"FUER DIE DIVISION EIN."
1050 PRINT
1070 PRINT TAB(10);:INPUT"GROESSTE";GR
1079 REM
1080 REM ERZEUGEN ZUFALLSZAHLEN
1081 REM
1090 A1=INT(GR*RND(1))+1
1100 A2=INT(GR*RND(1))+1
1109 REM
```

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

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

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

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.
In Zeile 420 wurde ein **Zähler gesetzt**, der überprüft, wie viele Falschantworten der Anwender bei dieser Aufgabe bereits gegeben hat. Das Programm soll ja nach drei falschen Antworten das Ergebnis anzeigen. Zeile 430 überprüft, ob bereits drei falsche Antworten gegeben wurden. Ist dies nicht der Fall, verzweigt das Programm nach Zeile 350 und stellt die Aufgabe erneut. Wurden drei falsche Antworten eingegeben, so fährt das Programm mit der Ausführung in Zeile 440 fort.
Wurde inzwischen das richtige Ergebnis eingegeben, so springt das Programm von Zeile 390 in die Zeile 450. Nachdem die Zeitschleife in Zeile 450 durchlaufen ist, wird in Zeile 460 das Ergebnis ausgegeben. Nach einer erneuten Zeitschleife fragt das Programm in Zeile 480 nach, ob eine weitere Aufgabe gestellt werden soll. Gibt der Anwender hier ein "J" ein, so wird zuerst der **Zähler "F"** auf **Null gesetzt** und anschließend nach Zeile 300 verzweigt. Dort werden zwei neue Zufallszahlen gebildet, die für

die neue Aufgabe benötigt werden. Betätigt der Anwender nicht die J-Taste, sondern irgendeine andere, so springt das Programm zurück nach Zeile 10, wo das Menü wieder aufgebaut wird. Der **Zähler "F" muß** deshalb **auf Null zurückgesetzt** werden, da für die **neue Aufgabe** ja **ebenfalls drei Versuche** zur Verfügung stehen sollen. **Wird das vergessen**, so würde der alte Wert von "F" mitgeschleppt, und dann kann es passieren, daß **schon nach zwei oder** nach **einer falschen Antwort** das **Ergebnis ausgegeben wird.** *Achten Sie also darauf, wenn Sie solche Zähler bei Ihren eigenen Programmen verwenden.*

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

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

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

A1=A2 : A2=A1 **< F E H L E R >**

so ginge der **Wert** von **A1 verloren**. Da zuerst A1 gleich A2 gesetzt wird, beinhaltet die Variable A1 jetzt den Wert von A2. Danach versucht man zwar, A2 gleich A1 zu setzen, aber A1 hat ja bereits den Wert von A2. Auf diese Art und Weise schafft man es also nicht, zwei Variablen zu vertauschen. Daher muß man *einen Wert zuerst "retten"*, d.h. *in einer Variablen zwischenspeichern.*

Zuerst bekommt die Variable I den Wert von A1, also

I=A1.

Danach wird der Variablen A1 der Wert von A2 übertragen, also

A1=A2.

Jetzt braucht man nur noch

A2=I

zu setzen, da ja **I** den **Wert von A1** hat, und schon hat die Vertauschung stattgefunden. Diese *Technik der Zwischenspeicherung* ist sehr wichtig. Überzeugen Sie sich daher davon, daß Sie das Prinzip verstanden haben, um es später in eigenen Programmen anwenden zu können.

Für dieses Problem bietet der Atari St allerdings noch eine komfortablere Lösung an ,die aber nicht alle Rechner besitzen. Wollen Sie also Ihre Programme auf andere Basicversionen anpassen, so müssen Sie in dem einen oder anderen Fall, doch die o.a. Lösung verwenden.

Die Anweisung

SWAP (X,Y)

vertauscht die Inhalte der Variablen X und Y. Sie können also durchaus die entsprechenden Zeilen im Programm durch diese Anweisung ersetzen.

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

ganze Zahlen gebildet wurde, nämlich A1 und A2. 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, seien hier 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. *Mit ON..GOTO können mehrere IF...THEN Vergleiche zusammengefaßt werden.*

3.3.2 Programmsprünge mit ON...ERROR

Diese spezielle Art des ON...GOTO-Befehls wird eingesetzt, um innerhalb eines Programms auftretende Fehler selbst per Programm zu behandeln bzw. zu verwalten. Eine typische Programmzeile könnte etwa so aussehen:

```
100 ON ERROR GOTO 1000
```

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 des Atari ST zwei Variablen (sogenannte Systemvariablen), die daraufhin abgefragt werden können. Es handelt sich dabei um die Variablen

ERR

und

ERL.

Mit ERR erfahren Sie den Fehlercode des Fehlers und mit mit ERL die Zeilennummer in der der Fehler aufgetreten ist. Das Ende einer Fehlerroutine wird durch den Befehl

RESUME.

gekennzeichnet. Mit

RESUME *(Zeilennummer)*

bestimmen Sie, in welcher Programmzeile das Programm nach der Fehlermeldung fortfahren soll. *RESUME NEXT* bewirkt, daß mit dem Befehl fortgefahren wird, der dem Befehl folgt, welcher den Fehler verursacht hat.

Sie haben aber auch die Möglichkeit, innerhalb eines Programms die '*Originalfehlermeldung*' des Atari ST ausgeben zu lassen. Dazu müssen Sie die folgende Befehlsfolge verwenden:

ON ERROR GOTO 0

Leider funktioniert der ON ERROR GOTO-Befehl zur Zeit noch nicht einwandfrei, was Sie mit dem folgenden Programm überprüfen können.

```
10 ON ERROR GOTO 100
20 GOTO 40
30 FOR I=1 TO 20
40 PRINT I
50 NEXT
60 END
100 PRINT ERL,ERR
110 RESUME NEXT
```

Das Programm bricht trotz der Zeile 10 mit der Originalfehlermeldung ab. Allerdings wurden sowohl die Fehlernummer als auch die fehlerhafte Zeile den beiden Variablen übergeben. Wollen wir hoffen, daß dieser Fehler noch beseitigt wird.

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(B^2-4AC))/2A
x2 = (-B - SQR(B^2-4AC))/2A

Für B^2-4AC < 0 existieren keine reellen Lösungen. Berücksichtigen Sie das in Ihrem Programm.

3.
```
10 REM TESTAUFGABE MIT ON...GOTO
20 INPUT"GEBEN SIE EINE ZAHL EIN";Z
30 ON Z GOTO 100,150,400:CLEARW 2
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 Die Abfrage der Tastatur

Das BASIC des Atari ST bietet Ihnen verschiedene Möglichkeiten, die Tastatur abzufragen. Die einfachste Möglichkeit, nämlich mit INPUT Zeichen einzulesen und einer Variablen zuzuordnen, haben Sie ja bereits kennengelernt.

Weiterhin ist wohl die INKEY$-Funktion vorgesehen, die aber in der vorliegenden Basicversion zur Zeit leider noch nicht korrekt arbeitet. Geben Sie z.B. die folgende Programmzeile ein

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

und starten dieses Miniprogramm, so werden Sie feststellen, daß das Programm nicht beendet wird, obwohl Sie eine Taste betätigen. Erst durch die 'brutale' Benutzung von *CONTROL C* kommen Sie aus dieser Funktion wieder heraus.

Eine weitere Möglichkeit Tasten zu selektieren besteht in der Anwendung von INPUT$. Mit

```
10 A$=INPUT$(1)
20 A=ASC(A$)
30 PRINT A
```

Der Parameter in Klammern bestimmt wieviele Zeichen Sie einlesen wollen. In unserem Beispiel wird demnach ein Zeichen eingelesen und dessen ASCII-Code ermittelt. Der Vorteil dieser Funktion ist, das Sie nach der Eingabe eines Zeichens nicht noch extra die Return-Taste zu betätigen brauchen.

Allerdings haben alle bisher angesprochenen Funktionen einen Nachteil. Man kann nicht alle Tasten der Tastatur abfragen. So übergeben die Cursortasten und die Funktionstasten bei der Anwendung dieser Funktionen keinen Wert.

Abhilfe schafft hier die Funktion

INP(2)

Der Parameter 2 steht hier für die Tastatur. Zulässige Parameter für INP sind:

```
0 = Drucker
1 = RS-232
2 = Console (Tastatur und Bildschirm)
3 = MIDI-Interface
4 = Tastaturprozessor
```

Das folgende kleine Programm zeigt Ihnen die Codes sämtlicher Tasten an. Mit der ESC-Taste können Sie das Programm beenden.

```
10 A=INP(2)
20 PRINT A,CHR$(A)
30 IF A=27 THEN END
40 GOTO 10
```

Damit wären soweit die wichtigsten Funktionen zur Eingabe von Zeichen über die Tastatur besprochen. Schauen wir uns nun noch einige Befehle bzw. Funktionen an, welche in Programmen nicht so häufig eine 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, CALL, WAIT

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

FRE

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

FRE(X)

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

PRINT FRE(X)

Befindet sich kein Programm im Programmspeicher, so erhalten Sie als Wert den maximalen freien Programmspeicherplatz.

POS

Diese Funktion wird Ihnen innerhalb eines Programms sehr selten begegnen. Mit ihr kann man die aktuelle Cursorposition auf dem Bildschirm erfahren. 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.

CALL

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 Basicbefehle über den Interpreter gesteuert, sondern direkt durch den Maschinencode, der sich nur noch aus Zahlenwerten zusammensetzt. Sie können mit CALL im Prinzip jede Speicherstelle des Rechners ansprechen. Es gibt jedoch viele Adressen bei denen das System "abstürzt". Diese Anweisung setzt also gewisse Kenntnisse des Betriebssystems voraus.

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.

Soweit die Befehle FRE, POS, CALL und WAIT.

3.6 READ, DATA und RESTORE

Bisher haben wir nur die Möglichkeit kennengelernt, Daten von der Tastatur aus einzugeben. 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

READ statement ran out of data at line (*Zeilennumer*)

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

RESTORE

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

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

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

READ statement ran out of data at line 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 in Anführungszeichen setzen. Denken Sie vor allem in Ihren Programmen daran, daß **kein String** einer **numerischen Variablen zugeordnet** werden darf.

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

Function call not allowed at line 20

ab. Daher achten Sie bei Verwendung solcher Variablenkombinationen 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, INPUT$ oder INP 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 Diskette mit abgespeichert! Will man dagegen mittels INPUT eingegebene Daten "*retten*", so muß man diese in einer **separaten Datei** auf **Diskette** abspeichern.

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

So, das war eine Masse an neuen Informationen und Anwendungsmöglichkeiten zu den Befehlen READ, DATA und RESTORE. Versichern Sie sich, daß Sie den Lehrstoff dieses Kapitels verstanden haben.

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:

```
5 CLEARW 2:FULLW 2
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:

```
5 CLEARW 2:FULLW 2
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

?You definded an array more than once

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.

```
5 CLEARW 2:FULLW 2
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

Subscript refers to element outside the array

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)	b)	c)
1	10	2
4	8	4
9	6	8
16	4	16
25	2	32
36	0	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 sehen Sie im Folgenden, 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.

```
5  CLEARW 2:FULLW 2
10 REM VORNAMENLISTE
20 DIM Y$(6)
30 Z=Z+1
40 INPUT"VORNAME";Y$(Z)
50 PRINT"WEITERE EINGABEN J/N ?"
60 A$=CHR$(INP(2))
70 IF A$ < > "J" THEN 100
80 IF Z < 6 THEN 30
90 PRINT"LISTE VOLL !"
100 END
```

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

Subscript refers to element outside the array at line 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 **ON ERROR GOTO** 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.

```
5  CLEARW 2:FULLW 2
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 A$=CHR$(INP(2))
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 A$=CHR$(INP(2))
110 IF A$ <> "J" THEN 150
120 FOR Z=1 TO 3
130 PRINT X(Z,1);X(Z,2);X(Z,3);X(Z,4)
140 NEXT Z
150 END
```

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

FELD A(3,4)

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


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


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

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

```
.
.
80 REM AUSGABE FELD
90 PRINT"FELD ANZEIGEN (J/N) ?"
100 A$=CHR$(INP(2))
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.

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.

```
5 FULLW 2
10 REM DATEN EINLESEN
20 DIM X$(120,3)
30 CLEARW 2
40 Z=Z+1
50 INPUT"VORNAME";X$(Z,1)
60 PRINT
```

```
70 INPUT"NAME";X$(Z,2)
80 PRINT
90 INPUT"TELEFONNR.";X$(Z,3)
100 PRINT
110 PRINT"WOLLEN SIE WEITERE DATEN "
120 PRINT"EINGEBEN (J/N) ?"
130 A$=CHR$(INP(2))
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 ATARI ST hat theoretisch die Möglichkeit, Felder mit bis zu 15 Indizes zu erstellen. Das bedeutet, daß nicht nur zwei-, drei- oder gar vierdimensionale Felder erzeugt werden könnten, sondern Felder mit bis zu 15 Dimensionen. Allerdings nur theoretisch, denn ein dreidimensionales Feld kann man sich 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.

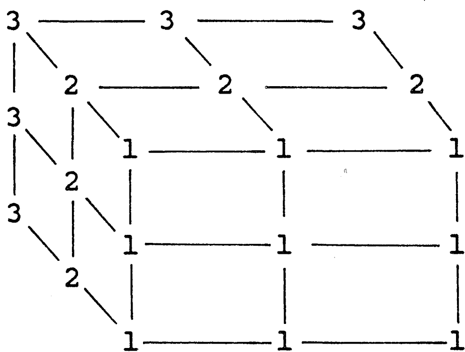
Schauen wir uns trotzdem zum dreidimensionalen Feld noch ein Beispiel an. Wir wollen die Indizes wie folgt definieren:

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 zweidimensionale Felder hintereinandergefügt hat. Das folgende Schaubild soll dies veranschaulichen.



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

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

DIM W(X,Y,Z)

bzw. konkret auf unser Beispiel bezogen:

DIM W(3,3,3)

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

Ihre Aufgabe besteht nun darin, ein Programm zu schreiben, das dieses Feld mit Daten auffüllt. Die Anzahl der Daten ist bekannt. Gehen Sie dabei so vor, daß zuerst die Spaltenwerte einer Zeile, danach die Zeilen selbst *(also wie beim zweidimensionalen Feld)* und schließlich die einzelnen "Scheiben" des Feldes aufgefüllt werden. Schreiben Sie das Programm so, daß genau das obenstehende Feld erzeugt wird. Damit sind die Wertzuweisungen der einzelnen Scheiben gemeint. Sie brauchen nicht den räumlichen Effekt bei der Ausgabe zu erzielen. 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 statement needs matching GOSUB at line
(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 statement needs matching GOSUB at line 150

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

```
110 IF A < 1 THEN GOSUB 130          < R I C H T I G
```

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

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

Geben Sie dieses Programm ein und starten Sie es. Sie werden feststellen, daß nach **15- bzw. 16maligem Durchlaufen des Unterprogramms** der gesamte **Programmablauf** mit der **Fehlermeldung**

?You have nested subroutine calls too deep at line 40

abgebrochen wird. Der Fehler liegt hier im Aufruf 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* 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 16. 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 CLEARW 2
240 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL "
250 PRINT
260 PRINT TAB(10)"FUER DIE ADDITION EIN."
270 PRINT
290 PRINT TAB(10);:INPUT"GROESSTE";GR
299 REM
300 REM ERZEUGEN ZUFALLSZAHLEN
301 REM
310 A1=INT(GR*RND(1))+1
320 A2=INT(GR*RND(1))+1
329 REM
330 REM BERECHNUNG ERGEBNIS
331 REM
340 ER=A1+A2
350 CLEARW 2
360 PRINT
370 PRINT"WIEVIEL ERGIBT" A1 "+" A2 "= ";
380 INPUT ES
390 IF ES=ER 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" ER
470 FOR I=0 TO 3000:NEXT I
480 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
490 INPUT A$
500 IF A$="J" THEN F=0:GOTO 300
```

```
510 GOTO 10
```

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

GEBEN SIE DIE GROESSTE ZAHL

FUER DIE ADDITION EIN.

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

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

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

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

```
.
.
.
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 CLEARW 2:F=0
610 PRINT
620 PRINT TAB(12)"RECHENLEHRGANG"
630 PRINT:PRINT
640 PRINT TAB(12)"WAEHLEN SIE:"
650 PRINT
660 PRINT TAB(12)"FUER "RA$(1)" EINE 1"
670 PRINT
680 PRINT TAB(12)"FUER "RA$(2)" EINE 2"
690 PRINT
700 PRINT TAB(12)"FUER "RA$(3)"EINE 3"
710 PRINT
720 PRINT TAB(12)"FUER "RA$(4)" EINE 4"
730 PRINT
740 PRINT TAB(12)"FUER ENDE EINE 5"
750 PRINT
760 PRINT TAB(12)"WELCHE ZAHL ?"
770 E$=CHR$(INP(2))
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 ein Zeichen eingelesen, wird zunächst in Zeile 780 überprüft, ob es sich um ein zulässiges Zeichen handelt *(zwischen 1 und 5)*. Ist das der Fall, so wird mit **VAL(E$)** der numerische Wert ermittelt und dieser der **Variablen P zugeordnet**. Über P wird dann in die entsprechenden Zeilen weiter verzweigt. Wurde die Addition angewählt, so hat P jetzt den Wert 1. Damit wird nach Zeile 800 verzweigt. Dort beginnt der Programmteil der Addition. Schauen wir uns auch hierfür die Programmzeilen an.

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

In Zeile 830 wird jetzt das erste Unterprogramm aufgerufen. Das ist der Teil, in dem die höchste Zahl eingegeben wird, mit der gerechnet werden soll. Nachfolgend wird dieses erste Unterprogramm aufgeführt.

```
.
.
110 REM   ******************
120 REM   * UNTERPROGRAMME *
130 REM   ******************
140 REM *************************
150 REM * EINGABE HOECHSTE ZAHL *
160 REM *************************
170 CLEARW 2:A$="":B$=""
180 PRINT TAB(10)"GEBEN SIE DIE GROESSTE ZAHL"
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 A$=CHR$(INP(2))
250 IF ASC(A$) < 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 INP wurde schon angesprochen *(s. Kapitel: Eingabe v. Daten mit INP)*. 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 CLEARW 2
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0: GOTO 500
440 PRINT:PRINT TAB(10)"FALSCH !"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
470 IF F <= 2 THEN 390
480 PRINT
490 FOR I=0 TO 2000:NEXT I
500 PRINT:PRINT TAB(5)"DAS ERGEBNIS LAUTET"ER
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
550 INPUT A$
560 RETURN
 .
 .
```

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

WIEVIEL ERGIBT

ausgegeben. Danach werden nacheinander die Werte der **Variablen A1, BE$(P) und A2 ausgegeben.** Diesen Variablen folgt noch das Gleichheitszeichen. Die Zeile schließt dann mit dem Semikolon, damit der INPUT-Befehl aus Programmzeile 420 in der gleichen Bildschirmzeile ausgegeben werden kann. Die allgemeine Form dieser Bildschirmausgabe könnte man so formulieren:

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

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

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

Damit hätten wir den Programmteil, in dem sich die Unterprogramme befinden, abgeschlossen. Sicherlich haben Sie bemerkt, daß wir die Unterprogramme an den Anfang des Programms gelegt haben. In vielen Büchern findet man den Ratschlag, die Unterprogramme an das Ende des Hauptprogramms zu legen. Diese Entscheidung wurde sicherlich im Hinblick auf eine später anzulegende Programmbibliothek getroffen. Man hat dann dort seine Unterprogramme nach Zeilennummern sortiert vorliegen, d.h. das Unterprogramm für die Rundung einer beliebigen Zahl liegt ab Zeilennummer 50000 usw. Wird nun ein neues Programm geschrieben, kann man diese

Routinen über einen speziellen Befehl 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 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 ER=A1+A2** und aus der **Multiplikation ER=A1*A2** direkt in das Unterprogramm übergeben werden können.

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

Bei der Division wollen wir **nur ganzzahlige Ergebnisse** erhalten. Aus diesem Grund wird zunächst durch die Multiplikation der Variablen A1 und A2 das Ergebnis ER berechnet. Im früheren

"*Rechenlehrgang*" konnten wir dann die Aufgabenstellung wie folgt stellen:

WIEVIEL ERGIBT ER / A1 = ?

Es wurde somit das vorher berechnete Ergebnis ER durch die Variable A1 dividiert, was zwangsläufig einen ganzzahligen Wert, nämlich den der Variablen A2, ergeben mußte. In unserem Unterprogramm können wir aber aus **Gründen** der **Allgemeingültigkeit** diese Umstellung **nicht direkt** vornehmen. Das muß wieder im **Programmteil** "*Division*" erfolgen. Dazu dienen die folgenden Programmzeilen:

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

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

```
10 REM *****************
20 REM * PROGRAMMSTART *
30 REM *****************
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 CLEARW 2:A$="":B$=""
180 PRINT AB(10)"GEBEN SIE DIE GROESSTE ZAHL "
190 PRINT
200 PRINT TAB(10)"FUER DIE "RA$(P)" EIN."
210 PRINT
220 PRINT TAB(10)"GROESSTE ? ";
230 FOR I=1 TO 3
240 A$=CHR$(INP(2))
250 IF ASC(A$) < 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 CLEARW 2
400 PRINT
410 PRINT"WIEVIEL ERGIBT";A1;BE$(P);A2;"= ";
420 INPUT ES
430 IF ES=ER THEN PRINT:PRINT TAB(10)"RICHTIG !":F=0: GOTO 500
440 PRINT:PRINT TAB(10)"FALSCH !"
450 FOR I=0 TO 2000:NEXT I
460 F=F+1
```

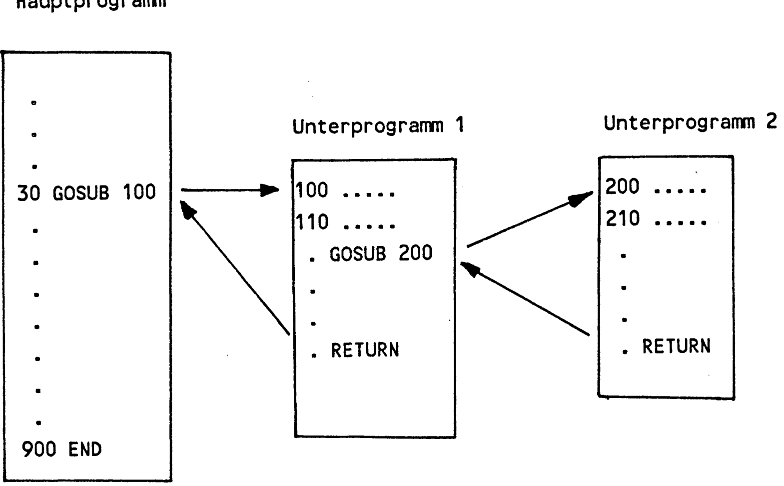
```
470 IF F <= 2 THEN 390
480 PRINT
490 FOR I=0 TO 2000:NEXT I
500 PRINT:PRINT TAB(5)"DAS ERGEBNIS LAUTET"ER
510 F=0
520 FOR I=0 TO 3000:NEXT I
530 PRINT
540 PRINT TAB(5)"NOCH EINE AUFGABE J/N";
550 INPUT A$
560 RETURN
570 REM *****************
580 REM *     MENUE     *
590 REM *****************
600 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 E$=CHR$(INP(2))
780 IF VAL(E$) < 1 OR VAL(E$) > 5 THEN 770
790 P=VAL(E$):ON P GOTO 800,890,990,1090,1180
800 REM ************
810 REM * ADDITION *
820 REM ************
830 GOSUB 110
840 GOSUB 310
850 ER=A1+A2
860 GOSUB 390
```

```
870 IF A$="J" THEN 840
880 GOTO 580
890 REM ***************
900 REM * SUBTRAKTION *
910 REM ***************
920 GOSUB 110
930 GOSUB 310
940 IF A1 < A2 THEN I=A1:A1=A2:A2=I
950 ER=A1-A2
960 GOSUB 390
970 IF A$="J" THEN 930
980 GOTO 580
990 REM  ************
1000 REM * DIVISION *
1010 REM ************
1020 GOSUB 110
1030 GOSUB 310
1040 ER=A1*A2
1050 I=ER:ER=A1:A1=I
1060 GOSUB 390
1070 IF A$="J" THEN 1030
1080 GOTO 580
1090 REM ******************
1100 REM * MULTIPLIKATION *
1110 REM ******************
1120 GOSUB 110
1130 GOSUB 310
1140 ER=A1*A2
1150 GOSUB 390
1160 IF A$="J" THEN 1130
1170 GOTO 580
1180 REM ********
1190 REM * ENDE *
1200 REM ********
1210 CLEARW 2
1220 END
```

Damit haben wir durch den Einsatz von drei Unterprogrammen ca. 43 Programmzeilen eingespart, und das trotz einer höheren Anzahl von REM-Zeilen! Sie sehen, daß die Verwendung von

Unterprogrammen nicht nur komfortabler ist, sondern auch Speicherplatz sparen hilft. **Den Sinn der Zeile 100** haben Sie jetzt bestimmt auch erkannt. Dadurch werden die **Unterprogramme übersprungen** und **direkt ins Menü verzweigt.**

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



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

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

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

Wir hoffen, 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 ER=A1+A2
910 RETURN
920 REM ***************
930 REM * SUBTRAKTION *
940 REM ***************
950 IF A1 < A2 THEN I=A1:A1=A2:A2=I
960 ER=A1-A2
970 RETURN
980 REM  ************
990 REM  * DIVISION *
1000 REM ************
1010 ER=A1*A2
1020 I=ER:ER=A1:A1=I
1030 RETURN
1040 REM ******************
1050 REM * MULTIPLIKATION *
1060 REM ******************
1070 ER=A1*A2
1080 RETURN
1090 REM ********
1100 REM * ENDE *
1110 REM ********
1120 CLEARW 2
1130 END
```

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

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

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

ON P GOSUB

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

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

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

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

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

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

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

4.3 Menütechniken

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

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

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

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

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

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

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

Quadratwurzel
Sinus
Cosinus
Natürlicher Logarithmus
Dekadischer Logarithmus

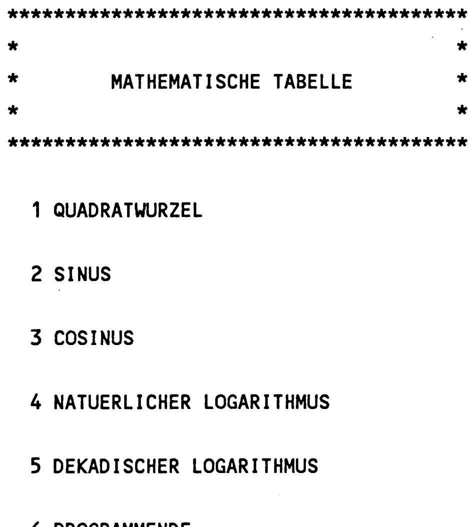
Diese fünf Berechnungen sollen also je nach Auswahl ausgeführt werden.

Ein Punkt fehlt noch in unserem Menü. Es handelt sich um den Punkt Programmende. Sie sollten Ihre Programme so schreiben, daß man sie auf "*normalem*" Wege beenden kann, **und nicht dazu Contrl-C 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 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.



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 INP-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 *****************
50 CLEARW 2:FULLW 2
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 CLEARW 2
240 FOR I=1 TO 40:PRINT"*";:NEXT I:PRINT
250 PRINT"*                                      *"
260 PRINT"*          MATHEMATISCHE TABELLE       *"
270 PRINT"*                                      *"
280 FOR I=1 TO 40:PRINT"*";:NEXT I:PRINT
290 RETURN
300 REM *********
310 REM * MENUE *
320 REM *********
```

```
330 GOSUB 230
390 FOR I=1 TO 3:PRINT:NEXT I
400 FOR I=1 TO 6
410 PRINT M$(I)
420 NEXT I
430 PRINT
450 PRINT "GEBEN SIE IHRE WAHL EIN  (1-6)";
460 W$=CHR$(INP(2))
 .
 .
```

Zunächst sollen die einzelnen Programmzeilen kurz erklärt werden. Die Zeile 50 löscht den Bildschirm. In den Zeilen 60 bis 80 wird das Feld M$ generiert und mit den Daten aus den DATA-Zeilen 90 bis 140 geladen. Danach überspringt das Programm den Programmteil der Unterprogramme und fährt mit der Ausführung in Zeile 330 fort. Von dort springt das Programm in das Unterprogramm ab Zeile 230, in dem die Kopfzeile erzeugt wird.

Danach wird das Unterprogramm verlassen und der Programmablauf wird in Zeile 340 fortgesetzt. 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. 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 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 INP gearbeitet haben, müßten Sie die eingegebenen Werte noch auf Zulässigkeit überprüfen.

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 der Anweisung INP auseinandersetzen.

4.3.1 Verwendung von Eingabe-Routinen im Menü

Dies bisher verwendeten Eingabeoperationen mit INP waren 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 **Ziffernfolge 054** einzugeben. Wollte man eine größere Zahl als 999 eingeben, war dies ebenfalls nicht möglich. Also doch wieder zurück zum INPUT-Befehl?

Es sei hier nochmals erwähnt, daß der INPUT-Befehl normalerweise für eigene Anwendungen ausreicht. Wollen Sie aber Ihre Programme gegen Fehlbedienung absichern, kommen Sie nicht umhin, die INP-Anweisung einzusetzen.

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

```
10 A$=CHR$(INP(2))
```

Damit können Sie jedes Zeichen von der Tastatur aus einlesen und A$ zuordnen. Wir wollen für unseren Fall aber nur Zahlen zulassen. Daher müssen wir diese Tasten selektieren, d.h. andere Eingaben als Zahlen müssen ausgeschlossen werden. Das erreichen wir mit einer IF...THEN-Abfrage:

```
10 A$=CHR$(INP(2))
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
```

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

```
10 A$=CHR$(INP(2))
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 A$=CHR$(INP(2))
15 IF ASC(A$) = 13 THEN 100
20 IF ASC(A$) < 48 OR ASC(A$) > 57 THEN 10
30 Z=Z+1
40 IF Z > 4 THEN 10
```

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

```
10 A$=CHR$(INP(2))
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
```

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.

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

```
10 A$=CHR$(INP(2))
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 A$=CHR$(INP(2))
30 IF ASC(A$) = 13 THEN 130
40 IF ASC(A$) < > 8 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 BACKSPACE** 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 BAXKSPACE-Taste allerdings betätigt, so wird in Zeile 50 überprüft, ob der String B$ noch Zeichen enthält.

Ist der **String** also **leer**, wird die **BACKSPACE-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 BACKSPACE-Taste der CHR$(8)-Code. Dieser bewirkt beim Ausdruck mit PRINT genau das gleiche, als ob Sie die BACKSPACE - Taste direkt benutzen würden. Um das zu veranschaulichen, geben Sie einmal folgende Befehlsfolge in den Rechner:

Damit haben wir nun eine Eingabe-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.

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 **GOTOXY** 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 *GOTOXY* den Cursor zu positionieren.

4.3.2 Cursorpositionierung mit GOTOXY

Mit GOTOXY wird der Cursor an eine bestimmte Position des Bildschirms gesetzt.

Achtung! Dieser Befehl arbeitet in der bei Erstellung des Buches vorliegenden BASIC-Version manchmal unkorrekt.

Die Schreibweise von GOTOXY

GOTOXY X,Y

Die Parameter bedeuten dabei im einzelnen:

```
X = Spaltennummer (0-39)
Y = Zeilennummer  (0-24)
```

Eine anschließende Ausgabe mit PRINT erfolgt meist nach der Positionierung. Das folgende Beispiel soll das verdeutlichen.

```
10 CLEARW 2
20 GOTOXY ,19,12
30 PRINT "AUSGABE MIT GOTOXY!"
40 END
```

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

4.4 Sortierverfahren

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

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

Feld mit 6 Elementen. Zunächst die Programmzeilen, die das Feld dimensionieren und mit Werten auffüllen:

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

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

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

```
240 PRINT F(I)
250 NEXT I
260 END
```

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

Danach wird Z auf eins gesetzt, da eine Vertauschung stattgefunden hat. Wir können also anhand des Zustandes von Z erkennen, ob eine Vertauschung stattgefunden hat oder nicht. Hat Z den Wert 1, so wurde ausgetauscht, hat Z den Wert 0, so wurde nicht ausgetauscht. Dieser "Trick" wird häufig in der Programmierung angewandt, um überprüfen zu lassen, ob bestimmte Vorgänge stattgefunden haben oder nicht. Diese Variablen, wie in unserem Beispiel Z, nennt man auch *FLAGS*. FLAG bedeutet soviel wie Flagge oder Zeichen. Hat Z also nach einem Durchlauf immer noch den Wert Null, so wissen wir, daß keine Vertauschung stattfand und daß wir somit unser Feld sortiert vorliegen haben. Das hat den Vorteil, daß die Sortierung bereits nach einem Durchlauf abgebrochen werden kann, wenn schon zufällig die richtige Reihenfolge der Elemente vorliegt. Dieses Verfahren findet man auch unter dem Namen *"BUBBLE-SORT mit Weiche"*. Weiche deshalb, weil eben jederzeit nach Erreichen der zufälligen Sortierung das Verfahren abgebrochen werden kann. Die letzten Zeilen geben dann das sortierte Feld aus. Wollen Sie beobachten, wie die Sortierung im einzelnen stattfindet, so ändern Sie die letzten Programmzeilen bitte wie folgt ab:

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

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

5

DAS ARBEITEN MIT DEM DISKETTENLAUFWERK

5.1 Programmverwaltung

Wohl kaum ein Computerbesitzer wird seinen Rechner aus- und später in der Hoffnung wieder einschalten, daß seine mühsam erstellten Programmzeilen noch vorhanden sind. Pustekuchen! Der interne Speicher des ST's ist wie der Speicher aller anderen Rechner auch dynamisch, wird im eingeschalteten Zustand ständig "aufgefrischt", da die darin enthaltenen Bits und Bytes sonst verloren gehen. Aus diesem und keinem anderen Grund ist es erforderlich, Programme auf ein externes Speichermedium zu archivieren. Die Zeiten, in denen Kassettenrekorder für diese Arbeit mißbraucht wurden sind anscheinend vorbei. ATARI bietet die Floppy SF354 zum ST zu einem durchaus akzeptablen Preis an.

Auf diesem Diskettenlaufwerk können Sie 360 KByte Ihrer Daten, in Form von Programm- oder sonstigen Dateien ablegen und zu jeder Zeit wieder benutzen. Wie bei vielen anderen Befehlen auch, enthält die Menüleiste des BASIC's alle Befehle, die zum Laden und Speichern von Programmen erforderlich sind. Diese wirklich kinderleichte Bedienung bedarf sicher keiner weiteren Erklärung. Wir wollen uns in diesem Teil den Befehlen zuwenden, die in Verbindung mit der Programmverwaltung stehen.

5.1.1 Speichern von Programmen

Beim ATARI-BASIC gibt es zwei Möglichkeiten, ein Programm abzuspeichern: speichern mit und ohne Überschreiben. Der Befehl SAVE speichert ein Programm, das mit dem angegebenen Namen noch nicht auf der Diskette existiert:

SAVE *name, zeilenbereich*

Der Parameter *name* bestimmt den Programmnamen. Dieser Name ist wie folgt aufgebaut:

xxxxxxxx.yyy

Der eigentliche Filename besteht aus maximal 8 Zeichen. Der Namenszusatz wird hinter einem Punkt angegeben und enthält maximal 3 Zeichen. Mit diesem Zusatz können Files klassifiziert werden. Wird kein Namenszusatz (Extension) angegeben, so wird automatisch der Zusatz **.BAS** eingesetzt.

Der *zeilenbereich* entspricht dem Parameter des **LIST**- oder **DELETE**-Befehls. Wird kein Zeilenbereich angegeben, so wird das gesamte Programm gespeichert.

Dieser Befehl erzeugt eine Fehlermeldung, wenn das Programm unter dem angegebenen Namen bereits auf der Diskette existiert.

Beachten Sie, daß der Programmname nicht in Anführungszeichen gesetzt wird.

Hier nun die zweite Möglichkeit, ein Programm zu speichern:

REPLACE *name, zeilenbereich*

Wie der Name schon sagt, aktualisiert dieser Befehl ein bereits auf der Diskette enthaltenes Programm. Die Parameter entsprechen denen des Befehls SAVE.

5.1.2 Laden von Programmen

Das ST-BASIC stellt auch zum Laden eines Programmes zwei Befehle zur Verfügung, die allerdings vollkomen identisch arbeiten.

OLD *name*

oder

LOAD *name*

Wird kein File-Extension angegeben, so wird das Programm mit dem Extension **.BAS** geladen.

Wird das angegebene Programm nicht gefunden, so werden Sie dies spätestens nach der Fehlermeldung **File not found on disk drive specified** merken.

5.1.3 Anzeigen des Disketteninhaltes

Der Befehl des ATARI-BASIC's zum Anzeigen des Inhaltes der Diskette lautet DIR. Dieser Befehl soll zunächst beschrieben werden:

DIR *laufwerk: maske*

Wird der Befehl ohne Maske angegeben, so wird der gesamte Inhalt angezeigt. Mit Hilfe der Maske können bestimmte Gruppen von Files angezeigt werden. Einige Beispiele:

DIR A:*.BAS zeigt alle Basic-Programme des Laufwerks A an.

DIR B:TEST.* listet alle Files auf, die den Namen TEST tragen.
Das Extension wird dabei ignoriert.

DIR A:A*.B* listet die Fioes auf, deren Name mit A und Extension mit **B** beginnt.

Der Stern bedeutet also, daß **alle** Zeichen beginnend ab dem Stern ignoriert werden sollen.
Ein weiteres Zeichen spielt beim Befehl DIR ebenfalls eine Rolle, das Fragezeichen. Das Fragezeichen bzeichnet alle **einzelne** Zeichen, die bei der Auflistung der Files nicht berücksichtigt werden sollen. Die Maske **?FILE.BAS** z.B. würde die Files **AFILE.BAS, BFILE.BAS, CFILE.BAS** usw. auflisten.

5.1.4 Löschen von Files

Sollte die Diskette einmal derart belegt sein, daß Sie kein Programm mehr speichern können, so sollte diese aufgräumt werden. Meist befinden sich einige alte Versionen von Programmen auf der Diskette, die schon lange nicht mehr benötigt werden.

Dazu setzen Sie den folgenden Befehl ein:

ERA *laufwerk:name*

Der Befehl **ERA A:TEST.BAS** z.B. löscht das Programm **TEST.BAS** auf der Diskette in Laufwerk A. Anstatt dem Filenamen kann auch eine Maske angegeben werden, wie sie beim DIR-Befehl bereits beschrieben wurde.

ERA A:*.* z.B. löscht **alle** Files auf Laufwerk A.

Das das BASIC in unserem ATARI ST mehrere fast identische Befehle hat, ist uns nicht mehr unbekannt. So gibt es auch zu diesem Befehl ein Äquivalent:

KILL "*laufwerk:name*"

Der einzige Unterschied scheinen die Anführungszeichen zu sein, in denen der Filebname beim KILL angegeben wird. Dies hat einen nicht zu übersehenden Vorteil: Die Paramter stellen einen String dar. Demnach ist in Verbindung mit dem KILL-Befehl folgende Befehlsfolge durchaus möglich:

```
B$="PROG1.BAS"
KILL B$
```

Der Vorteil dieses "parametriesierten" Löschens ist die Möglichkeit, innerhalb eines Programms einen Filenamen von der Tastatur einzulesen und diesen dann zu löschen. Dies funktioniert mit dem Befehl ERA nicht.

5.1.5 Umbenennen von Files

Wie jedes gute BASIC, bietet auch das ATARI-BASIC die Möglichkeit, DOS-Operationen durchzuführen. Zu den elementaren DOS-Funktionen gehört auch das Umbenennen von Files. Der entsprechende BASIC-Befehl dazu lautet wir folgt:

NAME *altfile* **AS** *neufile*

Die Parameter erklären sich von selbst. Soll z.B. das Programm **TEXT05.BAS** umgetauft werden in **TEXTPRO.BAS**, so hilft der folgende Befehl weiter:

NAME TEST05.BAS AS TEXTPRO.BAS

Da auch dieser Befehl nicht einer der schwersten ist, gehen wir nun zum nächsten Kapitel über.

5.2 Sequentielle Dateiverwaltung

Sie haben im bisherigen Verlauf des Buches bereits Daten wie z.B. Namen verarbeitet. Diese Daten waren jedoch immer fester Bestandteil des Programms im Rahmen von DATAs. Das dies nicht üblich ist, dürfte klar sein. Man kann einem Anwender eines datenverarbeitenden Programms nicht zumuten, zur Eingabe oder Ändeerung von Daten eine Programmänderung durchzuführen. In anderen Programmiersprachen wäre sogar noch eine vollständige neue Übersetzung des Programms notwendig, welch ein Blödsinn.

Das Prinzip, wie es richtig gemacht wird, dürfte Ihnen bereits eingeleuchtet sein: Die Daten werden in Form einer eigenständigen Datei auf der Diskette verwaltet. Wir wollen in diesem Kapitel die bekannteste Dateiorganisationsform behandeln.

In einer sequentiellen Datei befinden sich die Datensätze hintereinander angeordnet, jeweils durch das ASCII-Zeichen RETURN - CHR$(13) - getrennt. Dieses RETURN wird benötigt, da der entsprechende Befehl zum Lesen eines Eintrags in der sequentiellen Datei die immer bis zu diesem RETURN liest.

Die sequentielle Dateiverwaltung soll anhand einiger Beispiel-Unterprogramme behandelt werden.

Nehmen wir an, wir wollen eine im Rechner in Form einer zweidimensionalen Tabelle gespeicherten Adressenliste auf die Diskette bringen. Jeder Adressensatz besteht aus drei Felder:

A$(n,1) **Name**
A$(n,2) **Vorname**
A$(n,3) **Telefonnummer**

Die gesamte Liste soll 200 Eintragungen haben. In einer derartigen Dateiverwaltung ist es stets erforderlich, einen Satzzähler zu verwalten, der immer auf den letzen Datensatz in der Tabelle zeigt. Nennen wir diesen sinngemäß SZ. Dieser Zähler ist z.B. zur Neueingabe eines Namens erforderlich, da er

die Ermittlung des nächsten freien Tanellenplatzes erlaubt (SZ+1).

Zur Ausgabe dieser Adressentabelle auf dem Bildschirm könnten wir ein Unterprogramm etwa in der folgenden Form einsetzen:

```
1000 REM AUFLISTEN ADRESSEN
1010 :LISTADDR
1020 GOSUB BILDSCHIRMMASKE
1030 FOR X=1 TO SZ
1040 PRINT "NAME:     ";A$(X,1)
1050 PRINT "VORNAME: ";A$(X,2)
1060 PRINT "TELEFON: ";A$(X,3)
1070 PRINT
1080 PRINT "WEITER MIT RETURN"
1090 IF INP(2)<>13 THEN 1090
1100 NEXT X
1110 RETURN
```

Dieses Unterprogramm ist leicht zu "lesen". Wir wollen es daher nicht weiter erklären. Wichtig ist nun, wie wir diese Datensätze auf die Diskette schreiben können. Das Unterprogramm zum Speichern der Datensätze muß eingangs eine Datei öffnen in die die Adressen geschrieben werden sollen. Der Befehl zum Öffnen einer sequentiellen Datei zum Schreieben ist folgender:

OPEN "O",#*filenummer,"dateiname"*

Die Adressendatei soll **ADRESSEN.DAT** heißen, der OPEN-Befehl wäre dann folgender:

OPEN "O",#1,"ADRESSEN.DAT"

Die Filenummer ist erforderlich zur Unterscheidung mehrerer geöffneter Dateien.

Hier zunächst das Unterprogramm zum Schreiben der Adressen:

```
2000 REM SPEICHERN ADRESSEN
2010 :SAVEADDR
2020 OPEN "O",#1,"ADRESSEN.DAT"
2050 PRINT #1,SZ
2030 FOR X=1 TO SZ
2040 FOR Y=1 TO 3
2050 PRINT #1,A$(X,Y)
2060 NEXT Y,X
2070 CLOSE #1
2080 PRINT "ADRESSEN SIND GESPEICHERT!"
2090 PRINT "WEITER MIT RETURN"
2100 IF INP(2) <> 13 THEN 2100
2110 RETURN
```

Nach dem Öffnen der Datei wird zunächst der Satzzähler gespeichert. Er ist später zum gezielten Laden der Adressensätze erforderlich. In einer zweifach verschachtelten Schleife werden die einzelnen Felder der Tabelle gespeichert. Also zunächst Name, Vorname und Telefonnummer der ersten Eintragung, das Name, Vorname, Telefonnummer der zweiten Eintragung, usw. Zum Schreiben in eine sequetielle Datei wird der Befehl PRINT mit der Filenummer als Zusatz benutzt. Nun das Unterprogramm zum Laden dieser Datei:

```
3000 REM LADEN ADRESSEN
3010 :LOADADDR
3020 OPEN "I",#1,"ADRESSEN.DAT"
3030 INPUT #1,SZ
3040 FOR X=1 TO SZ
3050 FOR Y=1 TO 3
3060 INPUT #1,A$(X,Y)
3070 NEXT Y,X
3080 CLOSE #1
3090 PRINT"DATEI IST GELADEN!"
3100 PRINT"WEITER MIT RETURN"
3110 IF INP(2)<>13 THEN 3110
3120 RETURN
```

Der Modus im OPEN-Befehl ist beim Lesen einer Datei **I** **(Input)**. Nachdem der Satzzähler gelesen wurde, wird dieser als Argument für die äußere Schleife benutzt. Das Unterprogramm "weis" also durch diesen Satzzähler, wie weit gelesen werden muß.

Mit Hilfe dieser Unterprogramme, die man leicht auf die individuellen Erfordernisse anpassen kann, sind Sie nun in der Lage, eigene Dateiverwaltungen zu erstellen.

6

MUSIK UND GRAFIK

6.1 Musik

Ein Computer eignet sich sehr gut zur Erzeugung von Tönen. Der Prozessor, das eigentliche Arbeitspferd im Computer, wird von einem Taktgenerator angesteuert. Dieser Generator erzeugt je nach Rechner ca. 1-8 MHz (Millionen Takte pro Sekunde). Der einfachste Weg ist, diesen Takt auf einen Lautsprecher zu legen. Da die dabei entstehenden Töne vom menschlichen Gehör nicht mehr wahrgenommen werden können, muß die Taktfrequenz erheblich tiefer sein. Dies wird erreicht, indem die Taktfrequenz vor Ausgabe auf den Lautsprecher geteilt wird. Die dabei entstehenden Frequenzen werden dann als Töne wahrgenommen.

Dies war die erste Generation der Tonerzeugung mit dem Computer. Da diese Töne, vergleichbar mit den Wecktönen einer Armbanduhr, nicht zufriedenstellend waren, wurde die Tonerzeugung in Heimcomputern ständig weiterentwickelt. Das Resultat ist, daß viele Homecomputer mit sogenannten Synthesizern ausgerüstet sind, die auf einem Chip untergebracht sind. Die damit erzeugten Töne sind denen der "natürlichen" Musikinstrumente verblüffend ähnlich.

Den tonerzeugenden Chip im ATARI ST kann man nicht als Synthesizer im eigentlichen Sinn bezeichnen. Ihm fehlen wichtige Eigenschaften wie mehrere verschiedene Wellenformen (Sinus, Rechteck etc.) und die klangbeeinflussenden Filter. Auch ist das eigentliche Prinzip des Synthesizers, die Frequenz (Höhe) der Töne von einer Spannung abhängig zu machen, nicht gegeben.

Der Tongenerator des ATARI ST verfügt über drei Stimmen. Es können also drei Töne gleichzeitig erklingen. Alles, was dem Tongenerator entlockt wird, erklingt über den im Monitor eingebauten Lautsprecher. Die Lautstärke das eingebauten Lautsprechers kann am Monitor eingestellt werden.

Im folgenden Kapitel lernen Sie den BASIC-Befehl zur Tonerzeugung kennen. Die gesamte, komplexe Tonerzeugung kann an dieser Stelle nicht erläutert werden.

Der SOUND-Befehl

Der Tongenerator des ST wird mit dem Befehl SOUND gesteuert. Diesen Befehl wollen wir zunächst beschreiben:

SOUND *kanal*, *lautstärke*, *note*, *oktave*, *dauer*

Die Parameter:

kanal	Tonkanal (1bis 3)
lautstärke	von 0 bis 15 (0=stumm, 15=laut)
note	von 1 bis 12 (1=C, 2=Cis,... 12=B)
oktave	von 1 bis 8 (1=tief, 8=hoch)
dauer	Dauer des Tons in 1/50 Sekunden

Zunächst einige Grundlagen der Musik. Eine sogenannte Tonleiter besteht aus 12 Tönen, wie Sie der Übersicht einer Klaviatur entnehmen können.

ton-Parameter	Ton in der Tonleiter
1	C
2	Cis
3	D
4	Dis
5	E
6	F
7	Fis
8	G
9	Gis
10	A
11	Ais
12	B

Eine Klaviatur besteht aus mehreren solcher Abschnitte. Diese Tonfolgen nennt man Oktaven. Die erste Oktave umfaßt die tiefen Töne, die letzte Oktave die höchsten Töne.

Das folgende Programm "ST-ORGEL" verwandelt Ihren ST in ein Musikinstrument.

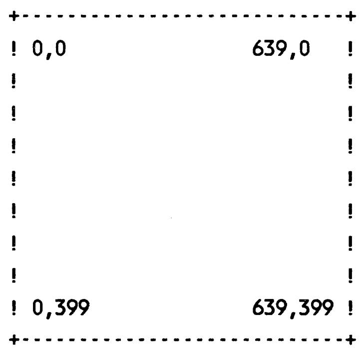
```
10 REM ST-ORGEL
20 DIM ton(256)
30 ton$="Q2W3ER5T6Y7UI9O0P"
40 FOR i=1 TO 17
50    index=ASC(MID$(ton$,i,1))
60    ton(index)=i
70 NEXT i
80 CLEARW 2:FULLW 2
90 PRINT "       ST-ORGEL"
100 PRINT
110 PRINT " 2 3   5 6 7   9 0"
120 PRINT
130 PRINT "Q W E R T Y U I O P"
140 a=INP(2)
150    ton=ton(a)
160 okt=3
170 IF ton>12 THEN ton=ton-12:okt=4
180 SOUND 1,15,ton,okt,20
190 SOUND 1,0,0,0,0
200 GOTO 140
```

Ein kleines Programm mit großer Wirkung! Vielleicht nehmen Sie noch einige Verbesserungen vor, nachdem Sie die Arbeitsweise durchschaut haben.

6.2 Grafik

Für viele Käufer der zahlreich angebotenen Homecomputer ist die Farbdarstellung und vor allem die Grafikfähigkeit des Rechners ein entscheidendes Kaufkriterium. Der ATARI ST bietet in seiner Preisklasse unvergleichbare Farbenpracht und hervorragende Grafikeigenschaften. So verfügt der Besitzer des ATARI ST über unzählige Farben, die er sich selbst aus den drei Grundfarben rot, grün und blau mischen kann. Die Auflösung der Grafik reicht bis zu 640 x 400 Bildpunkten, was für einen Rechner dieser Preisklasse geradezu sensationell ist.

Wie Ihnen sicher bekannt ist, können Sie zwischen drei verschiedenen Grafikauflösungen mit unterschiedlicher Farbenanzahl wählen. Zum Zeichnen von Figuren gibt es einige Befehle, die ausschließlich am Beispiel der hochauflösenden Grafik vorgestellt werden. Die hochauflösende Grafik besteht aus 640 x 400 Bildpunkten und den Farben scharz und weiß. Damit jeder Punkt angesprochen werden kann, muß man sich den Bildschirm als Koordinatensystem vorstellen. Ähnlich wie beim bereits bekannten Befehl GOTOXY, der den Cursor an eine beliebige Position bringen kann, wird auch bei den Grafikbefehlen eine Position bestimmt, nämlich mit zwei Werten. Der erste Wert bestimmt die Position auf der waagerechten Achse (X-Achse) des Bildschirms von links nach rechts orientiert. Der zweite Wert entspricht der Position auf der senkrechten Achse (Y-Achse) von oben nach unten zählend. Die vier Eckpunkte haben folgende Koordinaten:



Wenn Sie sich mit diesem System vertraut gemacht haben, können wir uns mit den ersten Befehlen zur Erzeugung der Grafik beschäftigen.

6.2.1 Linien

LINEF *vonx, vony, nachx, nachy*

Die Parameter sind hier wieder selbsterklärend. Eine Linie von der linken oberen zur rechten unteren Ecke des Bildschirms erzeugt z.B. der folgende Befehl:

LINEF 0,0,639,399

Interessant wird das Zeichnen von Linien in Verbindung mit Schleifen. Überzeugen Sie sich selbst:

```
10 CLEARW 2:FULLW 2
20 FOR i=0 TO 399 STEP 10
30 LINEF 399,i,0,399-i
40 NEXT i
```

Ein einfaches Programm mit großer Wirkung. Weil's so schön war, folgt noch ein weiteres:

```
10 CLEARW 2:FULLW 2
20 FOR i=0 TO 399 STEP 10
30 LINEF I,0,399,I
40 LINEF 399,I,399-i,399
50 LINEF 399-i,399,0,399-i
60 LINEF 0,399-i,i,0
70 NEXT i
```

Ist es nicht bewundernswert, wenn derartige Bilder auf dem Bildschirm entstehen, noch dazu wenn sie aus Ihrer eigenen Feder stammen? Vielleicht versuchen Sie sich auch einmal an derartigen Fantasiegrafiken. Es macht sehr viel Spaß, auch wenn es nicht auf Anhieb klappt. Im Vertrauen, ich habe mir an diesem Bild

auch den Kopf zerbrochen. Die Schwierigkeit ist, die Bilder, die man sich vorstellt, in das Koordinatensystsem umzurechnen, also den Algorithmus zu bilden. Das artet zu echten Denksportaufgaben aus.

6.2.2 Kreise

Zum Zeichnen eines Kreises müssen der Kreismittelpunkt sowie der Radius angegeben werden. Weiterhin besteht die Möglichkeit, Kreisausschnitte durch Angabe des Start- und Endwinkels zu erzeugen. Der Befehl hat die folgende Syntax:

CIRCLE *xmitte*, *ymitte*, *radius*, *startwinkel*, *zielwinkel*

Die Winkel werden in 1/10 Grad (0-3600) angegeben.

Das folgende Beispiel soll einen Kreis mit dem Radius von 50 Punkten in der Bildschirmmitte zeichnen:

CIRCLE 320,200,50

Hier nun ein Programm, das die Erstellung von Kreisausschnitten demonstriert:

```
10 FOR I=10 TO 100 STEP 5
20 CIRCLE 320,200,i,0,900
30 NEXT I
```

6.2.3 Ellipsen

Der Befehl zum Zeichnen von Ellipsen entspricht dem CIRCLE, bis auf die Unterscheidung von X- und Y-Radius.

ELLIPSE *xmitte,ymitte,xradius,yradius,startwinkel,zielwinkel*

Eine Ellipse in der Bildschirmmitte mit dem X-Radius von 100 und dem Y-Radius von 50 wird z.B. wie folgt gezeichnet:

ELLIPSE 320,200,100,50

Es folgt auch zu diesem Befehl ein begleitendes Demo-Programm:

```
10 FOR X=50 TO 200 STEP 10
20 FOR Y=50 TO 125 STEP 5
30 ELLIPSE 320,200,X,Y
40 NEXT Y,X
```

6.2.4 Ausgefüllte Flächen

Kreise und Ellipsen können auch farbig oder mit Rastern ausgefüllt werden. Die beiden Befehle dazu lauten

PCIRCLE

und

PELLIPSE

Die Parameter stimmen mit den bereits beschriebenen Befehlen überein. Bevor die Füll-Befehle ausgeführt werden können, müssen bestimmte Attribute eingestellt werden. Dies geschieht mit dem Befehl COLOR, dessen Syntax zunächst beschrieben wird:

COLOR *textfarbe*, *füllfarbe*, *linienfarbe*, *style*, *index*

Hier die Beschreibung der Parameter:

textfarbe	Farbe der Textausgabe
füllfarbe	Farbe des Füllmusters
linienfarbe	Farbe der Linien, Kreise und Ellipsen
style	Füllmuster
index	Fülltyp (2=Punktraster, 3= Schraffierungen)

Die ersten drei Parameter sind bei einem Monochrom-Monitor immer 1 (schwarz) oder 0 (weiß). Es besteht also die Möglichkeit, gezeichnete Figuren wieder zu löschen, indem Sie in der

Hintergrundfarbe neu gezeichnet werden. Zum Zeichnen der Grafik wird die Linienfarbe auf schwarz (COLOR 1,1,1), zum Löschen auf weiß (COLOR 1,1,0) gesetzt.

Die Füllmuster werden mit den Parametern *style* und *index* bestimmt. Das folgende Programm demonstriert die 24 verschiedenen Rastermuster:

```
10 CLEARW 2:FULLW 2
20 FOR I=1 TO 24
30 COLOR 1,1,1,I,2
40 PCIRCLE 25*(I-1)+15,100,20
50 NEXT I
```

Gleich danach die 12 verschiedenen Schraffurmuster:

```
10 CLEARW 2:FULLW 2
20 FOR I=1 TO 12
30 COLOR 1,1,1,I,3
40 PCIRCLE 25*(I-1)+15,100,20
50 NEXT I
```

Unabhängig vom Parameter *style* erzeugt der Fülltyp 0 weiß und der Typ 1 schwarz ausgefüllte Flächen. Fülltyp 4 hält das ATARI-Logo als Füllmuster bereit.

Dieses Kapitel sollte für Sie eine Anregung zu eigenen interessanten Grafikexperimenten sein. Wir hoffen, Sie ausgiebig inspiriert zu haben.

7

GEM-FUNKTIONEN

UNTER BASIC

7.1. GEM-Grundlagen

Vor dem Einsatz von GEM-Funktionen sollte das Prinzip des Grafik-Betriebssystems GEM bekannt sein. GEM besteht grob strukturiert aus den beiden Teilen

VDI (Virtual Device Interface)

AES (Applikation Environment System)

AES und VDI sind nichts anderes als riesige Sammlungen von Unterprogrammen, ebenso effektiv wie komfortable Funktionen. Diese Funktionen werden hauptsächlich von Assembler oder C aus in die Applikationen integriert und beim Compilieren oder Assemblieren dazugebunden. Zwar ist es nicht empfehlenswert, komplette GEM-Anwedungen in BASIC zu schreiben, jedoch können diverse Funktionen durchaus sinnvoll eingesetzt werden.

GEM-VDI

Das VDI enthält alle grundlegenden Grafikfunktionen wie Zeichnen von Linien, Kreisen usw. Die Funktionen, die das GEM-typische, optische Erscheinungsbild der Applikationen mit Fenstern, Boxes usw. prägen, sind in dem AES enthalten. Wird z.B. die Funktion zum Öffnen eines Fensters eingesetzt, so ist das AES dafür verantwortlich. AES wiederum wird unterstützt von den VDI-Routinen, die z.B. den Rahmen des Fensters erzeugen. VDI letztendlich setzt DOS-Funktionen ein. Die korrekte Hierarchie des GEM ist also

DOS (Disk Operating System)

VDI (Virtual Device Interface)

AES (Application Environment System)

Die Aufgabe des VDI ist es, das Erstellen von Grafikanwendungen durch Grafik-Funktionen zu erleichtern. Der Clou dabei ist, daß die Funktionen unabhängig von dem Grafik-Ausgabegerät sind. Die Gerätetreiber, ebenfalls Bestandteil des VDI, sorgen für die gerätespezifische Ansteuerung.

GEM-AES

Zur Erklärung, was AES eigentlich ist, zieht man am besten den Namen heran: Applikation Environment Manger. Auf Deutsch heißt dies etwa: Verwaltung der Umgebung für Anwenderprogramme. Diese "Umgebung" ist in GEM grafikorientiert, d.h. daß die Kommunikation zwischen dem Rechner und dem Anwender in erster Linie über grafische Elemente (z.B. Fenster und Piktogramme) erfolgt.

Die Grafikumgebung des AES ist ein besonders mächtiges Betriebssystem, das sich in mehrere Komponenten gliedert:

Die Routinen-Bibliotheken bieten Funktionen, mit denen alle Elemente des AES-Systems kontrolliert und genutzt werden können. Das Multitasking ermöglicht, daß mehrere Programme scheinbar gleichzeitig laufen. Die Shell stellt die Verbindung zum eigentlichen Betriebssystem, dem TOS, dar. Damit die Verwaltung von Bildschirmelementen möglich ist, verwaltet das AES Buffer, die zur Zwischenspeicherung von Bildschirmteilen dienen.

7.2 Parameterübergabe an GEM-Routinen

Die GEM-Routinen sind derart komplex, daß sie nicht nur einzelne Variablen, sondern ganze Tabellen (Arrays) von Ein- und Ausgabeparametern benötigen. Die VDI-Funktion zum Öffnen eines Arbeitsgerätes z.B. umfaßt nicht weniger als 14 Ein- und 60 Ausgabeparameter!

Doch keine Angst, wir werden von den zahlreichen GEM-Funktionen nur wenige demonstrieren. Das Prinzip der Übergabe und Übernahme der Parameter soll zuvor noch erklärt werden.

Die VDI-Funktionen werden über 5 Ein/Ausgabearrays gesteuert:

contrl	Ein/Ausgabearray
intin	Eingabearray
intout	Ausgabearray
ptsin	Eingabearray
ptsout	Ausgabearray

Das AES hat anstelle von **ptsin** und **ptsout** die Arrays **addrin** und **addrout**. Weiterhin gibt es ein weiteres Array mit Namen **global**.

7.2.1 VDI-Aufrufe

Die Anfangsadressen der VDI-Arrays befinden sich in den reservierten Variablen **contrl**, **intin**, **intout**, **ptsin** und **ptsout**. Mit dem Befehl POKE werden die Eingabeparameter in die Arrays gefüllt. Dabei muß beachtet werden, daß die Arrays **contrl**, **intin** und **intout** 2-Byte-Werte (Worte) enthalten. Ein in der GEM-Literatur als z.B. **intin(3)** bezeichneter Parameter wird demnach mit dem Befehl

POKE INTIN+6,*wert*

gesetzt. Dies muß unbedingt berücksichtigt werden!

Die VDI-Funktionen sind numeriert. Diese Funktionsnummer wird vor dem VDI-Aufruf in **contrl(0)** mit dem Befehl

POKE CONTRL,*funktionsnummer*

gespeichert. VDI wird schließlich mit dem Befehl

VDISYS

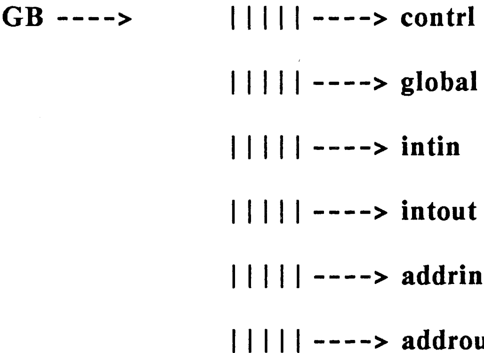
aufgerufen. Nach dem Aufruf der Funktion können eventuelle Rückgabeparameter gelesen werden. Gibt die VDI-Funktion z.B. in **intout(2)** einen Wert zurück, so kann dieser mit dem Befehl

variable=**PEEK(INTOUT)**

einer Variablen zugeordnet werden.

7.2.2 AES-Aufrufe

AES-Aufrufe sind im Vergleich zu VDI-Aufrufen nicht so leicht zu handhaben. Hier stehen nicht alle Array-Adressen in reservierten Variablen zur Verfügung. Es gibt lediglich die Variable **GB**, die die Adresse der Adressenliste der AES-Arrays enthält. Dies hört sich kompliziert an, wird aber sofort näher erklärt. Sehen Sie dazu zunächst die folgende Abbildung:



Die Adresse von GB also zeigt auf einen Bereich von 6 mal 4 Bytes. Der Bereich enthält also 6 Adressen. Dies sind die Adressen der AES-Arrays.

Wichtig! Die Arrays intin und intout sind nicht identisch mit den VDI-Arrays!

Aus diesem Grund werden in der folgenden AES-Initialierung andere Bezeichnungen eingesetzt.

```
1000 REM AES-INITIALISIERUNG
1010 A#=GB
1020 GCONTRL  = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Nach Aufruf dieses Unterprogramms stehen die Adressen aller Arrays in den Variablen zur Verfügung.

Beachten Sie, daß die Arrays **addrin** und **addrout** 4-Byte-Arrays darstellen. **addrin(3)** z.B. wird mit dem Befehl **POKE ADDRIN+12,***wert* gesetzt.

Der Aufruf des AES lautet

GEMSYS (*funktionsnummer***)**

Anders wie beim VDI wird beim AES die Funktionsnummer nicht in **contrl(0)** abgelegt, sondern beim Aufruf des AES mit übergeben. Es ist also nicht erforderlich, die Funktionsnummer beim AES in das **contrl**-Array zu setzen.

Nach Aufruf des AES können die Rückgabeparameter wie beim VDI beschrieben ausgelesen werden.

7.3 VDI-Beispiele

Das folgende Programm zeichnet ein abgerundetes Rechteck. Diese Grafikoperation steht nicht im BASIC zur Verfügung!

```
10 REM ABGERUNDETES RECHTECK
20 COLOR 1,1,1,1,1
30 FULLW 2
40 POKE CONTRL,11      :REM FUNKTIONSNUMMER FÜR GRAFIKOPERATIONEN
50 POKE CONTRL+2,2     :REM ANZAHL DER KOORDINATEN IN PTSIN
60 POKE CONTRL+6,0     :REM ANZAHL DER INTIN-EINTRAGUNGEN
70 POKE CONTRL+10,8    :REM FUNKTIONS-KENNUNG FÜR ABGERUNDETES RECHTECK
80 POKE PTSIN,50       :REM X-KOORDINATE OBERE LINKE ECKE
90 POKE PTSIN+2,20     :REM Y-KOORDINATE OBERE LINKE ECKE
100 POKE PTSIN+4,150 :REM X-KOORDINATE UNTERE RECHTE ECKE
110 POKE PTSIN+6,100 :REM Y-KOORDINATE UNTERE RECHTE ECKE
120 VDISYS
130 RETURN
```

Die Bemerkungen hinter den Parametern sind nicht etwa übertrieben, denn nur so bleibt der Überblick erhalten.

Das abschließende Beispielprogramm erlaubt die Gestaltung des Textes auf dem Bildschirm. Folgende Schriftaren sind möglich:

1	**fett**
2	**hell**
4	**kursiv**
8	**unterstrichen**
16	**auseinandergezogen**
32	**normal**

Es sind auch beliebige Kombinationen aus diesen Schriftarten erlaubt. Sie addieren dann einfach die Zahlen der Schriftart. Eine fette und kursive Schrift erhalten Sie so z.B. mit der Kennung 5.

Doch nun das Programm:

```
10 REM SCHRIFTARTENGESTALTUNG
20 REM FULLW 2
30 INPUT"SCHRIFTART-KENNUNG:";A
40 IF A<1 OR A>63 THEN 30
50 POKE CONTRL,106      :REM FUNKTIONSNUMMER
60 POKE CONTRL+2,0      :REM EINTRAEGE IN INTIN
70 POKE CONTRL+6,1      :REM ANZAHL INTIN-EINTRAGUNGEN
80 VDISYS
90 END
```

7.4 AES-BEISPIELE

Das folgende Demoprogramm setzt die AES-Routinen 73 und 74 zum Wachsen und Schrumpfen einer Box ein. Diese Funktion wird zur optischen Beeindruckung des Anwenders oft in GEM-Programmen eingesetzt. Sie erwecken den Eindruck, daß die Box z.B. aus dem Hintergrund herangeflogen kommt.

```
10 REM WACHSEN UND SCHRUMPFEN EINER BOX
20 GOSUB 1000
30 FOR I=0 TO 7
40 READ A:POKE GINTIN+2*I,A
50 NEXT I
60 DATA 315,200,5,5,0,0,640,400
70 GEMSYS (73):GEMSYS(74)
80 IF INP(2)<>13 THEN 70
90 END
1000 REM AES-INITIALISIERUNG
1010 A#=GB
1020 GCONTRL  = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Das GINTIN-Array besteht aus 8 Eintragungen. Die ersten vier Eintragungen geben die X-Koordinate der oberen linken Ecke, die entsprechende Y-Koordinate, die Breite und die Höhe des basierenden (kleinen) Rechteckes an. Die zweiten vier Eintragungen geben die Maße des resultierenden Rechteckes in der gleichen Form an. Das Programm wiederholt sich mit jedem Tastendruck und bricht mit der RETURN-Taste ab.

Das zweite Beispielprogramm soll die Form der Maus ändern. Folgende Formen dürfen eingegeben werden:

0	**Pfeil**
1	**Cursor**
2	**Biene**
3	**Hand mit Zeigefinger**
4	**flache Hand**
5	**dünnes Fadenkreuz**
6	**dickes Fadenkreuz**
7	**Fadenkreuz als Umriß**

Hier das Programm:

```
10 REM EINSTELLEN MAUSFORM
20 FULLW 2
30 INPUT "MAUSFORM:";F
40 IF F<0 OR F>7 THEN 30
50 GOSUB 1000
60 POKE GINTIN,F
70 GEMSYS (78)
80 END
```

```
1000 REM AES-INITIALISIERUNG
1010 A#=GB
1020 GCONTRL  = PEEK(A#)
1030 GLOBAL  = PEEK(A#+4)
1040 GINTIN  = PEEK(A#+8)
1050 GINTOUT = PEEK(A#+12)
1060 ADDRIN  = PEEK(A#+16)
1070 ADDROUT = PEEK(A#+20)
1080 RETURN
```

Diese Beispiele können sicherlich nur einen kleinen Einblick in die vielen Möglichkeiten des Einsatzes von GEM-Funktionen innerhalb von BASIC-Programmen geben. Wenn Sie sich weiter mit diesem interessanten Gebiet beschäftigen möchten, so empfehlen wir Ihnen DATA BECKER's "Das große GEM-Buch" (Szczpanowski/Günther).

8

LÖSUNGEN

8. Lösungen

Lösungen zu Seite 34

1.

a)	6C	b)	92
c)	BA	d)	F0
e)	C	f)	C9

2.

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

3.

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

4.

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

Lösungen zu Seite 61

zu 1.

a) zulässig
b) zulässig
c) AUTO unzulässig, Atari St Befehl.
d) zulässig
e) IF unzulässig, siehe c).
f) GB unzulässig (Systemvariable).
g) 4NAME% unzulässig, da Bezeichnung mit einer Ziffer beginnt.
h) 255 unzulässig, siehe g).
i) zulässig

zu 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.

zu 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
```

zu 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.

zu 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.

zu 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 75

1.

```
10 REM LOESCHEN BILDSCHIRM
20 CLEARW 2
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 das Output-Fenster mit dem CLEARW 2 Befehl gelöscht. Die Zeilen 40 und 50 ordnen den Variablen W1 und W2 jeweils neu erzeugte Zufallszahlen zu. Sollten Sie mit der Erzeugung der oberen und unteren Grenze Schwierigkeiten gehabt haben, so lesen Sie noch einmal im Kapitel über die Zufallszahlen nach.

2.

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

Bei diesem Programm mußten Sie beachten, daß zuerst S berechnet werden muß, da diese Variable bei der Berechnung der Fläche schon mit verwendet wird. Die Umsetzung der Formel in BASIC dürfte keine Schwierigkeiten bereitet haben. Achten Sie trotzdem bewußt darauf, daß Sie in Ihren Programmen nicht in die mathematische Schreibweise der Formeln verfallen. Dies geschieht nur allzu leicht.

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 wird aber vom Rechner eine Fehlermeldung 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 ebenfalls eine Fehlermeldung ausgegeben. Wie man

dieses Problem bei der Eingabe umgeht, wird in einem späteren Kapitel erklärt (siehe INP).

4.

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

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

5.

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

Dieses Programm braucht wohl nicht näher erläutert zu werden. In seiner Einfachheit erklärt es sich fast von selbst.
Haben Sie diese Aufgaben selbständig zu Ihrer eigenen Zufriedenheit gelöst, so können Sie jetzt getrost zum nächsten Kapitel übergehen. Bei Unsicherheiten schlagen Sie noch einmal in den entsprechenden Passagen nach.

Lösungen zu Seite 88

1. a) ist richtig.

2. Man erhält den Ausdruck BEIN.

3. Man erhält wieder den Ausdruck ROTOR.

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

Das ist eine mögliche Lösung.

Lösungen zu Seite 107

1.

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

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

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

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

In Zeile 20 haben wir unseren Zähler für die einzelnen Summanden von 1 bis 100. Zeile 30 berechnet die Summe der bis dahin aufgetretenen Werte von A, also 1+2+3+4 usw. Zeile 40 führt den bekannten Vergleich aus und in Zeile 50 wird schließlich die Summe S der einzelnen Summanden von 1 bis 100 ausgegeben.
Die zweite Lösung ergibt sich aus der Tatsache, daß wir es hier mit einer *arithmetischen Reihe* zu tun haben, d.h. die Differenz zwischen den einzelnen Gliedern ist konstant. Die Summe läßt sich nach der Formel

$$Sn=n/2(A1+An)$$

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

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

3.

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

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

4.

Bei dieser Aufgabe mußten Sie erkennen, daß jeweils nur die letzten Werte von A und Z ausgegeben werden. Der PRINT-Befehl steht nämlich außerhalb der eigentlichen Schleife. Damit hätte Ihre Lösung lauten müssen:

52 9

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

5.

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

Diese Aufgabe war, zugegeben, schon recht schwierig. Ihr Programm muß dem obigen nun nicht aufs Haar gleichen. Jedoch sollte es die Erzeugung des Vergleichsstrings in Zeile 50 in etwa beinhalten, da das ja die eigentliche Schwierigkeit ist. Die Aufgabenstellung bezog sich auf einen beliebigen String, d.h. unabhängig davon, nach welchen und nach wieviel Zeichen der ursprüngliche String durchsucht werden sollte. So mußte 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 137

1.

```
10 REM HARMONISCHE REIHE
20 CLEARW 2
30 PRINT"BIS ZU WELCHER SUMME SOLL"
40 PRINT:PRINT"ADDIERT WERDEN ?"
50 PRINT
60 INPUT S
70 Z=1
80 SH=SH + 1/Z
90 Z=Z+1
100 IF Z = 50 * INT(Z/50) THEN PRINT Z;"ADDITIONEN"
110 IF SH < S THEN 80
120 PRINT"NACH" Z "GLIEDERN IST DIE SUMME" SH
```

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

2.

```
10 REM QUADRATISCHE GLEICHUNG
20 CLEARW 2
30 PRINT"EINGABE DER KOEFFIZIENTEN A,B,C"
40 PRINT
50 INPUT A,B,C
60 IF A=0 THEN 20:REM A MUSS <> 0 SEIN
70 D=B*B-4*A*C
80 IF D < 0 THEN 140
90 X1 =(-B+SQR(D))/(2*A)
100 X2 =(-B-SQR(D))/(2*A)
110 PRINT"LOESUNG FUER X1 =";X1
120 PRINT"LOESUNG FUER X2 =";X2
130 GOTO 150
140 PRINT"KEINE REELLEN NULLSTELLEN !!"
150 END
```

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

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

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

Lösungen zu Seite 168

1.

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

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

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

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

2.

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

Dieses Programm hat von der Struktur her den gleichen Aufbau wie das Programm aus Aufgabe 1. Hatten Sie also die Lösung von Aufgabe 1, so hatten Sie auch gleichzeitig die Lösung der Aufgabe 2. Der Unterschied besteht lediglich in der Art des Feldes (numerisch) und der Zufallszahlengenerierung in Zeile 40. Die Vergleiche zur Auffindung der größten Zahl basieren auf dem gleichen Prinzip wie in Aufgabe 1. In Zeile 90 wird nur auf

größer/gleich untersucht, da wir ja die größte Zahl ausfindig machen wollten.

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

3.

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

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

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

ANHANG

Der ATARI 520 ST wurde zu Anfang mit einer vorläufigen BASIC-Referenzkarte, also ohne dem BASIC-Sourcebook ausgeliefert, die weder vollständig noch fehlerfrei war. Die folgende Befehlsübersicht enthält alle Befehle, die in der Referenzkarte entweder fehlerhaft oder überhaupt nicht beschrieben wurden.

CLEARW

Schreibweise: CLEARW *windownr*
Funktion: Löscht den Inhalt eines Bildschirmfensters. *windownr* darf Werte zwischen 0 und 3 annehmen.

CLOSEW

Schreibweise: CLOSEW *windownr*
Funktion: Schließt ein Bildschirmfenster, d.h. das Fenster verschwindet vom Desktop.

CONTRL

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

EQV

Schreibweise: X=*op1* EQV *op2*
Funktion: Verknüpft die beiden Operanden *op1* und *op2* im negierten Exklusiv-Oder.

FILL

Schreibweise: FILL *x,y*
Funktion: Füllt die Fläche aus, auf die die Koordinate zeigt

FULLW

Schreibweise: FULLW *windownr*
Funktion: Setzt ein Bildschirmfenster auf die volle Bildschirmgröße.

GB

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

GEMSYS

Schreibweise: GEMSYS (*funktnr*)
Funktion: Ruft die AES-Funktion auf, deren Opcode mit *funktnr* spezifiziert wird.

GOTOXY

Schreibweise: GOTOXY *zeile*, *spalte*
Funktion: Positioniert den Cursor auf die mit *zeile* und *spalte* bestimmte Bildschirmposition

IMP

Schreibweise: X=*op1* IMP *op2*
Funktion: Verknüpft die beiden Operanden *op1* und *op2* nach den Regeln der Implikation.
(s. Kap. 1.6)

INKEY$

Schreibweise: A$=INKEY$
Funktion: Liest ein Zeichen von Tastatur und speichert es in A$.
In der vorliegenden BASIC-Version nicht lauffähig!

INTIN

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

INTOUT

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

MOD

Schreibweise: X MOD Y
Funktion: Ergibt den Restwert der Division von X / Y.

OPEN

Schreibweise: OPEN "*modus*",#*dateinummer*, "*dateiname*", *satzlänge*
Funktion: Öffnet eine Datei. Wichtig: Der Modus (I, O oder R) muß in Anführungszeichen gesetzt werden. Die Satzlänge betrifft nur relative Dateien (Modus R).

OPENW

Schreibweise: OPENW *windownr*
Funktion: Öffnet ein Bildschirmfenster

PTSIN

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

PTSOUT

Funktion: Reservierte Adressvariable zum GEM-Aufruf.

SOUND

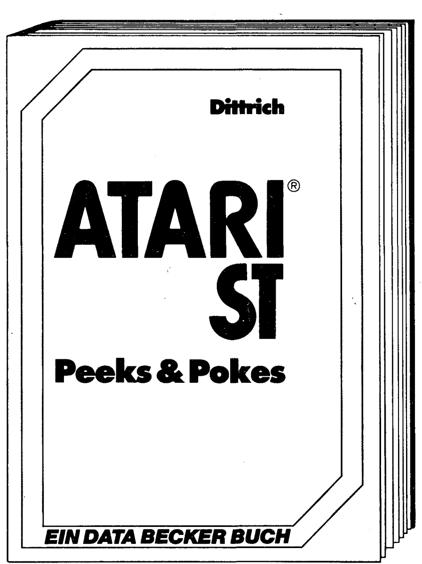
Schreibweise: SOUND *stimme*, *lautst*, *note*, *oktave*, *dauer*
Funktion: Steuert die Tongeneratoren (siehe Kapitel MUSIK).

VDISYS

Schreibweise: VDISYS
Funktion: Ruft eine VDI-Funktion auf. Die Funktionsnummer wird im **contrl**-Array übergeben.

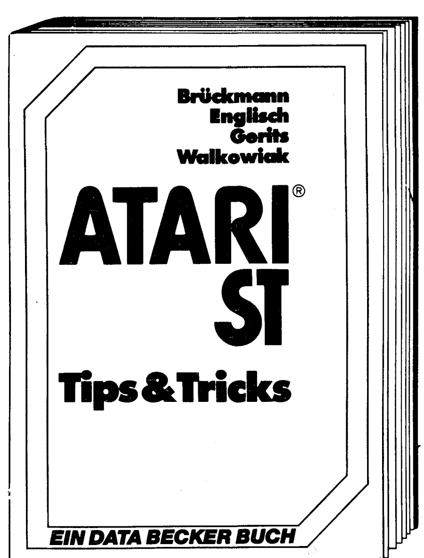
WAVE

Schreibweise: WAVE *mixreg*,*envreg*,*shape*,*periode*,*delay*
Funktion: Dieser Befehl steuert die Wellenform der mit SOUND erzeugten Töne.



Schlagen Sie dem Betriebssystem Ihres ATARI ST ein Schnippchen. Wie? Mit PEEKS & POKES natürlich! Dieses Buch erklärt Ihnen leichtverständlichh den Umgang damit. Mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Dabei wird der Aufbau Ihres ST's prima erklärt: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks sind nur einige Stichworte dazu. Der erste Schritt hin zur Maschinensprache!

PEEKS & POKES zum ATARI ST
ca. 200 Seiten, DM 29,–
ISBN 3-89011-148-3
Erscheint ca. Januar

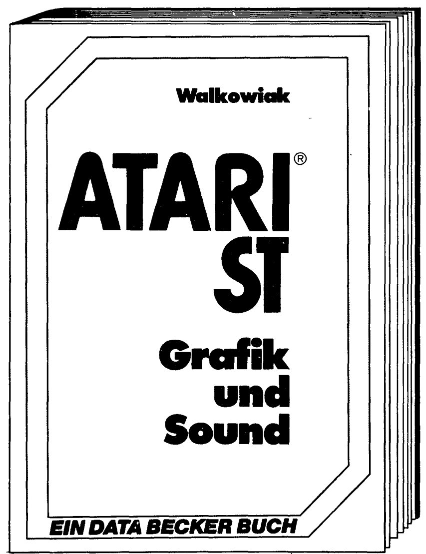


Eine riesige Fundgrube faszinierender Tips & Tricks um Ihren ATARI ST voll auszunutzen! Von phantastischen Grafiken über raffinierte Programme in BASIC, Assembler und C bis hin zu fortgeschrittenen Anwendungsmöglichkeiten. Ein fantastisches Buch zu einem fantastischen Rechner!

Gerits/Englisch/Brückmann/ Walkowiak
ATARI ST Tips & Tricks
256 Seiten, DM 49,–
ISBN 3-89011-118-1

Ein Buch für jeden, der unter GEM Programme erstellen will! Arbeiten mit der Maus, Icons, Virtual Device Interface, Application Environment Services und Graphics Device Operating System. Ein besonderer Schwerpunkt liegt im Einbinden von GEM-Routinen in BASIC und 68000-Assembler und der Programmierung in diesen Sprachen. GEM – das Betriebssystem der Zukunft!

Szczepanowski/Günther
Das große GEM-Buch zum ATARI ST
ca. 459 Seiten, DM 49,–
ISBN 3-89011-125-4



Grafik und Sound auf dem ATARI ST. Ein Traum wird wahr! Grafikgrundlagen, Animationsgrafik, Funktionsdiagramme, 2-D/3-D Grafik, CAD, Soundgrundlagen und das MIDI-Interface sind nur einige Schwerpunkte dieses Buches. Alle Beispiele sind gründlich erklärt und mit vielen Beispielprogrammen verdeutlicht. Werden Sie zum Bildschirmkünstler und Computerdirigenten.

Walkowiak
ATARI ST Grafik & Sound
ca. 300 Seiten, DM 49,–
ISBN 3-89011-123-8
Erscheint ca. Januar

DAS STEHT DRIN:

Das große BASIC-Buch zum ATARI ST ist eine ausführliche, didaktisch gut geschriebene Einführung in das BASIC des ATARI ST. 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 und eine Übersicht der in der Referenzkarte nicht berücksichtigten Befehle.

Aus dem Inhalt:

- Datenfluß- und Programmablaufpläne
- fortgeschrittene Programmiertechniken
- Grafikprogrammierung
- Mehrdimensionale Felder
- Sortierverfahren
- Dateiverwaltung
- BASIC unter GEM

  und vieles mehr

UND GESCHRIEBEN HABEN DIESES BUCH:

Frank Kampow ist EDV-Fachmann mit jahrelanger Programmierpraxis und Autor etlicher DATA BECKER Bücher, ebenso wie Norbert Szczepanowski, der als Datenverarbeitungs-Kaufmann seine fundierten Kenntnisse im Programmieren verschiedener Rechner eingebracht hat.

ISBN 3-89011-121-1