HAPPY COMPUTER

Carsten Straush
Hartmut Pick

CPC 464

FÜR EIN- UND UMSTEIGER

Eine Reise durch die Schneider-CPC 464-Welt mit Anleitungen zur BASIC-Programmierung

Ein Markt & Technik Buch

CPC 464 für Ein- und Umsteiger

Carsten Straush
Hartmut Pick

CPC 464 für Ein- und Umsteiger

Eine Reise durch die Schneider CPC 464-Welt mit Anleitungen zur BASIC-Programmierung

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Straush, Carsten:
CPC 464 für Ein- und Umsteiger : Eine Reise durch die Schneider-CPC 464-Welt
mit Anleitungen zur BASIC-Programmierung /
Carsten Straush ; Hartmut Pick. —
Haar bei München : Markt-und-Technik-Verlag, 1985.
(Happy computer)
ISBN 3-89090-090-9
NE: Pick, Hartmut:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
89 88 87 86 85

ISBN 3-89090-090-9

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Vorwort

Der Schneider CPC 464 ist ein exzellenter Computer - wenn man mit ihm umgehen kann. Allerdings bereitet das Aufspüren seiner Vorzüge einige Probleme. Das Handbuch, obwohl ziemlich ausführlich gehalten, kann hier doch aufgrund der vielfältigen Möglichkeiten teilweise nur Andeutungen geben. Gerade bei den etwas komplexeren und damit meist auch schwierigeren Befehlen ist der Benutzer häufig auf sich allein gestellt. Viele der Tricks und Schliche der Maschine bleiben so leider im Verborgenen.

Nachdem wir einige Monate mit der Anleitung verbracht hatten, entschlossen wir uns, "CPC für Einsteiger und Umsteiger" zu schreiben. Dieses Buch soll anfängliche Frustrationen ersparen und eine Menge nützlicher Anregungen geben.

Am Anfang jedes Kapitels steht eine Zusammenfassung der jeweiligen Befehle und ihres Zusammenhangs. Im Anschluß daran werden in einer Fülle von Programmen Anwendungs- und Lösungsmöglichkeiten aufgezeigt. Jedes der über fünfzig Programme ist ausführlich beschrieben. Es zeigt die Funktion der einzelnen Befehle und ist gleichzeitig als Haupt- oder Unterprogramm für eigene Problemstellungen des Lesers nutzbar. Die Programmpalette ist dabei sehr weit gespannt. Sie reicht von den Standardproblemen der Text- und Datenverarbeitung bis zur grafischen Darstellung und auch die klanglichen Qualitäten kommen nicht zu kurz.

Den Abschluß jedes Kapitels bildet eine Sammlung von Hintergrundinformationen und Tips, die den Leser zu einer besseren Beherrschung des Computers führen. Viel Spaß dabei wünschen

Die Autoren

Inhaltsverzeichnis

1 Einleitung

Der CPC 464 wird als Personalcomputer angepriesen. Grund genug, sich mit seinen Fähigkeiten auf dem Gebiet der Text- und Dateiverwaltung etwas näher zu beschäftigen.

Dieser Bereich bildet dann auch den ersten Haltepunkt auf unserer Reise durch das BASIC des CPC 464. Wir untersuchen die Möglichkeiten der Abspeicherung und des Lesens von Daten und ganzen Dateien. Wir beschäftigen uns näher mit den gebräuchlichsten Sortierroutinen und lernen einiges über die Ausgabe von Texten an bestimmten Cursorpositionen. Der Trick, wie man den Cursor zum Blinken bekommt, wird ebenso erläutert, wie das Einfügen von Texten auf dem Bildschirm. Wir zeigen auch, wie man auf dem CPC einen FULL-SCREEN-EDIT erzeugen kann. Die gebräuchlichsten STRING-Befehle werden in diesem Teil eingeführt und auch auf Spezialbefehle aus dem String-Bereich gehen wir ein. So wird das Umformen von Groß- in Kleinbuchstaben und umgekehrt ebenso gezeigt, wie das Suchen von Zeichenketten mit dem INSTR-Befehl. Die häufigsten Sortierverfahren für Zahlen und Strings werden erläutert und Anwendungsmöglichkeiten aufgezeigt.

Darauffolgend wenden wir uns der Abspeicherung der Befehle zu und lernen einen Befehl kennen, den es gibt und doch nicht gibt. Wir beschäftigen uns mit einem Schutz gegen das Auflisten unserer Programme und lernen einiges über den BASIC-Speicher des CPC.

Wir erweitern unser Blickfeld und beschäftigen uns mit dem Verkürzen von Programmen und der Technik der strukturierten Programmierung. Das Ende dieses Kapitels bildet eine Spezialität, die der CPC von größeren Rechnern gelernt hat: die automatischen Fehlerbehandlungrsoutinen.

Wir verlassen dann den Machtbereich des BASIC-Interpreters und wenden uns den CPC-spezifischen Befehlen zu. Unser erster Stop liegt im Bereich der Grafik. Zuerst gehen wir auf die Grundlagen der Grafik, Farbe und Zeichengröße, ein. Wir beschäftigen uns mit der Abspeicherung der Farbe und damit, wie wir mit sich verändernden Farben sehr effektvolle Warnmeldungen produzieren können. Auch die Wahl der richtigen Farbkombination und eine Hilfe beim Arbeiten mit Farbe, der von uns entwickelte Farbplaner, werden hier besprochen.

Im nächsten Unterpunkt behandeln wir das Verhältnis von Text und hochauflösender Grafik. Auf dem Plan steht dabei die Positionierung von Texten und ihre Verwendung zur Erzeugung von Laufschriften bei der Titeldarstellung. Hier beschäftigen wir uns auch mit einer etwas ungewohnten Art der Textpositionierung, den Systemsteuerzeichen. Durch Bewegung des Cursors ist es relativ einfach möglich, Texte zu verschieben und Teile des Bildschirms zu löschen. Mit der Ausgabe von Texten und Zahlen in vorher festgelegten festen Formaten beschäftigen wir uns im Anschluß daran.

Daraufhin wenden wir uns einer Spezialität des CPC zu, die ebenfalls von größeren Brüdern übernommen wurde, den WINDOWS. Wir sagen einiges zur Definition von WINDOWS und gehen im Anschluß daran auf die verschiedenen Anwendungsmöglichkeiten ein. In diesen Bereich gehören das Erstellen von Grafiken mittels WINDOWS ebenso, wie eine optisch wirksame Aufteilung des Bildschirms in verschiedene Bildschirmzonen. Als nächstes wenden wir uns dem Zeichensatz zu und beschäftigen uns mit der Schöpfung neuer Zeichen.

Dann verlassen wir den Bereich der normalen Grafik und gehen auf die hochauflösende Grafik näher ein. Wir beschäftigen uns mit dem Befehlssatz, mit der grafischen Statistik, dem Entwurf von Diagrammen und dem Malen und Zeichnen mit dem Computer.

Als nächstes lassen wir den CPC Töne erzeugen. Da der Sound des 464 gewöhnungsbedürftig ist, erläutern wir erst einmal die Grundlagen. Sie werden im Handumdrehen zum fast perfekten "Tontechniker"; das gleichzeitige Erzeugen von mehreren Tönen sowie deren geschickte und wirkungsvolle Integration in BASIC-Programme werden an speziell ausgewählten Beispielen nahegebracht. Gerade für den Newcomer ist es von großer Bedeutung, sich von Anfang an mit einer speicherplatzsparenden Programmierung (hier im SOUND-Bereich) vertraut zu machen und sich diese Methode konsequent anzugewöhnen. Ist man später ein Routinier, bastelt dann an entsprechend anspruchsvollen und damit längeren Programmen, ist man über jedes eingesparte und an anderer Stelle dringend benötigte Byte Speicherplatz froh. Daher haben wir viel Mühe darauf verwendet, anhand dieses speziellen Bereiches eine sinnvolle und möglichst effiziente Programmierung bzw. deren Grundlagen zu verdeutlichen.

Ein zum Verständnis des CPC 464 sehr wichtiges Kapitel folgt nun; es trägt die Überschrift "Der CPC und die Außenwelt". Hier beschäftigen wir uns mit der Tastatur und deren Umdefinition. Wir legen Blockgrafik und sogar ganze Programmzeilen auf Tasten.

Dann geht es zur Kassette. Mit dem automatischen Einfügen von Programmen hat auch sie einige Spezialitäten parat. Wir gehen auf das Beheben von Ladefehlern ein und sehen uns im nachfolgenden Kapitel mit Zeitproblemen konfrontiert. Wie mehrere Programme gleichzeitig parallel ablaufen können, wird hier ebenso erläutert, wie eine auf dem Bildschirm permanent laufende Uhr. Den Abschluß des Buchs bildet dann ein Anhang, der neben einigen nützlichen Tabellen auch einen Vergleich des CPC-BASICs mit dem des Commodore 64 enthält, um das Umschreiben von VC-64-Programmen zu ermöglichen.

Starten wir nun zum ersten Punkt unserer Reise, dem BASIC des CPC. Am Anfang soll hier eine Kurzbeschreibung einiger Befehle stehen. Sie sind nach den einzelnen Anwendungsbereichen in die Gebiete Bildschirmansteuerung, Datenverwaltung, Programmablauf, STRING-Verarbeitung und Datenspeicherung unterteilt. Damit ist es relativ schnell möglich, die Erklärung zu einem Befehl anhand seiner Anwendungsmöglichkeiten zu finden.

2

Der BASIC-Interpreter

2 Der BASIC-Interpreter

2.1 Kurzbeschreibung der BASIC-Befehle

Im folgenden wollen wir uns zunächst mit den BASIC-Befehlen und deren Syntax sowie deren Möglichkeiten beschäftigen.

2.1.1 Bildschirmansteuerung

PRINT

Mit diesem Befehl wird auf den Bildschirm oder auf ein Peripheriegerät geschrieben. Dazu wird der Befehl um die Gerätenummer ergänzt: PRINT#GN. Für ein GN zwischen 0 und 7 erscheint die Ausgabe auf dem Bildschirm. Bei GN=8 wird auf einen angeschlossenen Drucker ausgegeben und bei GN=9 wird der Kassettenrecorder bzw. das Diskettenlaufwerk angesprochen. Dazu muß aber vorher mit OPENOUT ein FILE eröffnet werden.

Wird keine Gerätenummer angegeben, erscheint die Ausgabe nur auf dem Bildschirm. Der Computer reagiert dann so, als wäre PRINT#0 eingegeben worden. Für Gerätenummern zwischen 1 und 7 erfolgt die Ausgabe in den jeweils definierten WINDOWS (WINDOW = Fenster). (Das Ansprechen der WINDOWS ist in Kapitel 3 beschrieben.) Wurde kein WINDOW definiert, erfolgt die Ausgabe so, als wäre keine Gerätenummer (GN) angegeben.

Die auszugebenden Texte müssen in Anführungszeichen stehen oder in einem String abgelegt sein. Als Trennungszeichen werden Komma und Semikolon verwendet. Bei ";" werden die auszugebenden Daten hintereinander, bei "," mit einem Zwischenraum ausgegeben. Hinter der Gerätenummer GN ist immer ein Komma erforderlich.

WRITE

Dieser Befehl ist dem PRINT-Befehl sehr ähnlich. Er druckt allerdings alle Zeichen einschließlich Anführungszeichen und Trennzei-

chen (Komma, Semikolon) wie in der angegebenen Liste aus. Der Unterschied wird bei folgendem Beispiel sofort klar:

```
PRINT#0,"Schneider","CPC  464"
```

erscheint auf dem Bildschirm als :

```
Schneider          CPC  464
```

und

```
WRITE#0,"Schneider", "CPC  464"
```

als:

```
"Schneider", "CPC 464"
```

Ansonsten sind beide Befehle identisch.

CLS

Mit diesem Befehl wird der Bildschirm gelöscht und der Cursor in die obere linke Ecke zurückgesetzt. Es ist auch möglich, bestimmte WINDOWS zu löschen. Dazu wird dann lediglich der Name mit angegeben. Beispiel: CLS#1. Hier wird das WINDOW Nr. 1 gelöscht, der Rest des Bildschirms bleibt erhalten. Die Möglichkeiten, die sich mit diesem Befehl ergeben, um Farbflächen auf dem Bildschirm zu erzeugen, werden in Kapitel 3 dargestellt.

LOCATE

Der in vielen BASIC-Dialekten vorkommende Befehl PRINT AT ist hier durch den Befehl LOCATE ersetzt. Damit wird der Cursor an eine bestimmte Stelle des Bildschirms bzw. eines WINDOWS gesetzt. Er wird folgendermaßen angesprochen: LOCATE#1,S,Z (S = Spalte, Z = Zeile).

Zur Darstellung an dieser Stelle muß denn der PRINT- oder WRITE-Befehl verwendet werden.

2.1.2 Datenverwaltung

DATA

Hinter diesem Befehl folgt eine Liste von Daten, die in einem Programm zur Verfügung stehen sollen. Die Daten können numerisch oder alphanumerisch sein. Sie werden durch Kommas getrennt. Der Befehl kann nur in einem Programm und nicht als Direktbefehl verwendet werden.

READ

Mit READ werden Daten aus einem Datenfeld gelesen, das mit dem Befehl DATA erstellt wurde. Dem Befehl folgt die Bezeichnung der Variablen, bzw. des STRINGS, dem die DATA-Anweisung zugeordnet werden soll. Sehen wir uns dazu ein Beispiel an:

```
10 DATA Schneider, CPC 464
20 READ A$:READ B$
30 PRINT A$;B$
```

Hier wurden die Worte "Schneider" und CPC 464" in einer DATA-Anweisung abgelegt, danach gelesen und dann auf dem Bildschirm ausgedruckt.

RESTORE

Bei der mehrmaligen Ausführung des READ-Befehls wird immer eine DATA-Anweisung nach dem anderen gelesen. Dies ist aber nicht immer erwünscht. Oft müssen auch bestimmte Daten mehrfach gelesen werden. Dazu kann mit RESTORE ZN (ZN = Zeilennummer) bestimmt werden, in welcher Zeile (ZN) weitergelesen wird. Wird kein ZN angegeben, wird zur ersten DATA-Zeile gesprungen.

2.1.3 Programmablauf

RUN

Der Befehl kann in drei Versionen verwendet werden:

1) ohne Zusatz (RUN):

Ein im Speicher befindliches Programm wird ab der ersten Programmzeile durchgeführt. Vorher wird ein CLEAR ausgeführt, das heißt alle Variablen und STRINGS werden gelöscht.

2) mit Angabe einer Zeilennummer (RUN 100):

Ein im Speicher befindliches Programm wird ab der angegebenen Zeile gestartet, ebenfalls wird vorher ein CLEAR durchgeführt.

3) mit Angabe eines Programmnamens (RUN"Programm"):

Ein Programm mit dem angegebenen Namen wird geladen und gestartet. Ein eventuell im Speicher befindliches Programm wird ebenso wie im Speicher befindliche Daten gelöscht.

NEW

Bei der Eingabe von NEW wird der gesamte BASIC-Speicher überschrieben und das Programm gelöscht. Die Programmsteuerung wird auf den Anfang des BASIC-RAM zurückgesetzt. Ein einmal eingegebenes NEW ist nicht mehr korrigierbar, da jede Zelle des Speichers danach mit einer Null beschrieben ist, so daß das Programm wirklich verloren ist.

Ähnlich verhält es sich bei der Durchführung eines RESTOREs durch Drücken der Tasten SHIFT, CONTROL und ESC gleichzeitig.

END/STOP

Trifft der Computer bei der Programmausführung auf einen dieser Befehle, wird die Programmausführung sofort abgebrochen. Diese

Befehle sollten verwendet werden, wenn das logische Ende eines Programms mit dem physikalischen Ende übereinstimmt.

IF ... THEN ... ELSE

In ein Programm sollen oft Bedingungen auf Zutreffen oder Nichtzutreffen überprüft werden. Mit einer IF-Abfrage ist das möglich.

Beispiel:

```
10 IF I = 1 THEN PRINT "CPC" : END
20 IF I <> 1 THEN I = 1
30 GOTO 10
```

In Zeile 10 wird der Computer angewiesen, "CPC" auszudrucken, wenn I + 1 gilt. Es ist auch möglich, ihm durch den ELSE-Befehl zusätzlich mitzuteilen, was geschehen soll, falls die Bedingung nicht gilt.

Beispiel:

```
10 IF I=1 THEN PRINT "CPC" ELSE I=1:GOTO 10
```

Hier wird dem Computer mitgeteilt, daß er, falls I=1 gilt, "CPC" ausdrucken, andernfalls der ELSE-Anweisung folgen, I=1 setzen und die Abfrage erneut durchführen soll. Beim zweiten Durchgang gilt dann auf jeden Fall I=1 und das Programm ist beendet. Der GOTO 10-Befehl wird nicht mehr erreicht.

FOR ... NEXT .../ WHILE ... WEND

Die einfachste Art, einen Programmteil mehrfach hintereinander durchlaufen zu lassen, ist die Bildung einer FOR-NEXT-Schleife. Hier muß die Anzahl der Durchläufe von vornherein bekannt sein. Soll ein Programmteil zum Beispiel dreimal durchlaufen werden, würde folgende Befehlsfolge möglich:

```
10 FOR I+ 1 TO 3
30 REM PROGRAMM
.
.
.
50 NEXT I
```

Die Variable I kann dabei im Programm benutzt werden (zum Beispiel zur Indizierung von Datensätzen).
Beispiel:

```
10 FOR I = 1 TO 10
20 PRINT A(I)
30 NEXT I
```

Oft kommt es aber auch vor, daß die verschiedenen I's nicht in einem Abstand von 1 gebraucht werden, sondern zum Beispiel eine 2 zur bisherigen Variablen dazuaddiert werden soll. Das ist möglich, indem man mit STEP eine Schrittweite angibt, mit dem der Abstand der Variablen definiert wird.

Beispiel:

```
10 FOR I = 1 TO 10 STEP 2
20 PRINT A(I), A(I+1)
30 NEXT
```

Hier sollen 2 Elemente einer Datei nebeneinander gedruckt werden. Dazu ist es erforderlich, daß in der FOR-NEXT-Schleife die Indizierung immer 2 Einheiten weiterzählt. Durch Eingabe einer negativen Zahl hinter STEP zählt das System rückwärts. Es können auch "krumme" Zahlen (z.B. 0.7) als Zählindex (STEP) benutzt werden.

Alle diese Möglichkeiten verlangen die genaue Kenntnis der Anzahl der Wiederholungen. Oft ergibt sich diese Anzahl erst aus dem Programmablauf. Meist soll eine Wiederholung stattfinden, wenn Bedingung erfüllt oder nicht erfüllt ist. Im einfachen BASIC müßte dies folgendermaßen programmiert werden:

```
10 I = I+1
20 IF I>= 10 THEN GOTO 10
```

Diese Abfrage verbraucht viel Speicherplatz, da viele BASIC-Befehle benutzt werden. Außerdem ist dieser Stil nicht gerade elegant. Aus der Programmiersprache PASCAL, die auf die Programmstruktur Wert legt, wurde im LOCOMOTIVE-BASIC des CPC 464 die Befehlskette WHILE ... WEND entlehnt. Diese beiden Befehle erfüllen den oben beschriebenen Effekt voll und ganz. Das obige Beispiel würde dann folgendermaßen programmiert:

```
10 WHILE I <= 10
20 I = I + 1
30 WEND
```

Mit WHILE wird die Bedingung definiert, unter der eine Wiederholung stattfinden soll, und WEND gibt dann die Stelle an, zu der zurückgesprungen werden soll, wenn die Bedingung aus WHILE dies erfordert.

Im allgemeinen ist eine solche Schleifenbildung eleganter als eine mit Hilfe von GOTO-Befehlen. Durch diese Befehle wird das BASIC des CPC der Sprache PASCAL angenähert. PASCAL-Programme gelten im allgemeinen als übersichtlicher als BASIC-Programme. Dies kommt daher, daß solche Strukturen bisher nur in PASCAL programmiert werden konnten. Wird eine solche Programmstrukturierung auch in BASIC möglich, ist dies nur zu begrüßen und die Befehlsfolge sollte bei entsprechender Aufgabenstellung angewandt werden.

2.1.4 STRING-Verarbeitung

Zur Stringverarbeitung (string = Zeichenkette) stehen in LOCOMOTIVE-BASIC eine Vielzahl von Befehlen zur Verfügung.

ASC

ASC("X") gibt den ASCII-Code des in Anführungsstrichen angegebenen Zeichens an.

CHR$

CHR$(ASC) gibt das Zeichen, das mit dem ASCII-Code in Klammern steht, an. Es handelt sich also um die Umkehrfunktion von ASC.

VAL

Dieser Befehl wird eingesetzt, um den Wert eines STRING zu bestimmen. Allerdings ist dabei zu beachten, daß nur numerische

STRINGS in einen Wert umgewandelt werden können. Ist in einem STRING ein Buchstabe enthalten, werden bei VAL nur die links von dem Buchstaben stehenden Zahlen berücksichtigt. Der umzuwandelnde STRING folgt in Klammern dem Befehl : VAL(D$).

STR$

Dies ist der Umkehrbefehl zu VAL. Er wandelt eine Zahl in einen STRING um. Sinnvoll wird dieser Befehl zum Beispiel dann, wenn ein STRING um eine Zahl erweitert werden soll.

Beispiel:

```
10 A$="Schneider CPC"
20 B = 464 : B$=STR$(B)
30 C$= A$+B$
40 PRINT C$
```

Auf dem Bildschirm erscheint dann:

```
Schneider CPC 464
```

Dies wäre ohne STR$ so nicht möglich gewesen, es sei denn "464" wäre sofort im Programm durch B$="464" als STRING eingesetzt worden.

CLEAR

Dieser Befehl löscht alle Datenfelder, auch Stringarrays. Er wird auch ausgeführt, wenn RUN eingegeben wird.

DIM

Die Dimensionierung eines Datenfelds erfolgt mit DIM. Wird ein Stringarray angesprochen, ohne daß eine Dimensionierung vorgenommen wurde, führt das Betriebssystem die Dimensionierung auf 10 durch. Konkret heißt dies, wird irgendwo im Programm A$(1) angesprochen, reserviert das System Speicherstellen für A$(0) bis A$(10), also insgesamt 11 Plätze. Kommt später einmal A$(11) vor, erfolgt eine Fehlermeldung mit Programmabbruch. Es ist auch nicht möglich, im nachhinein die Dimensionierung zu ändern. Nachdem

einmal A$(1) angesprochen wurde, führt jede Dimensionierung von A$ zu einer Fehlermeldung. Die Dimensionierung muß also in jedem Fall vorher geschehen. Am Sinnvollsten werden alle in einem Programm verwendeten Felder zu Beginn des Programms dimensioniert. Dies gilt sowohl für Stringarrays als auch für Zahlenfelder. Es können dreidimensionale Felder erzeugt werden.

Beispiel:

```
DIM A$(10,10,10).
```

Damit werden 11 x 11 x 11 STRING-Speicherstellen reserviert.

ERASE

Dieser Befehl ist die Umkehrung von DIM. Er teilt dem Betriebssystem mit, daß eine Variable nicht mehr benötigt wird und schafft somit Platz im Speicher.

LEN

LEN(A$) gibt an, wieviele Stellen A$ hat. Es wird also die Länge des STRINGS A$ ermittelt. Leerzeichen werden dabei als belegte Stellen aufgezählt.

INSTR

INSTR untersucht, ob ein STRING in einem anderen enthalten ist. Er ermittelt die erste Stelle, ab der der enthaltene STRING im ursprünglichen STRING anfängt.

Beispiel:

```
10 A$="Schneider CPC"
20 A =INSTR(A$,CPC)
30 Print A
```

Das Ergebnis muß 11 sein, da CPC an der 11. Stelle von A$ beginnt. Dieser Befehl ist geeignet zur Suche von STRINGS. In Schleifen läßt sich ein gesamtes Datenfeld nach einem bestimmten Wort oder Wortteil durchsuchen.

LEFT$

Dieser Befehl teilt einen STRING. Es kann festgelegt werden, daß der linke Teil eines STRINGS verarbeitet werden soll. In der Klammer hinter dem Befehl muß der STRING und die Anzahl der linksseitigen Stellen enthalten sein, die betrachtet werden sollen.

Beispiel:

```
10 A$=Schneider CPC
20 B$=Left$(A$,9)
30 PRINT B$
```

Das Ergebnis muß "Schneider" sein, da dies die 9 letzten Stellen von A$ sind.

RIGHT$

Dies ist der Umkehrbefehl zu LEFT$. Hier werden die rechts liegenden Stellen betrachtet.

MID$

MID$ stellt eine Konstruktion von RIGHT$ und LEFT$ dar. Hier wird ein Teil aus der Mitte eines STRINGS genommen.

Beispiel:

```
10 A$="Schneider CPC 464"
20 B$=MID$(A$,11,3)
30 PRINT B$
```

Als Ergebnis muß "CPC" auf dem Bildschirm erscheinen, da CPC an der 11. Stelle von A$ anfängt und 3 Zeichen lang ist.

LOWER$

Dieser Befehl ist interessant, da er in einem STRING enthaltene Großbuchstaben in Kleinbuchstaben verwandelt. Für Vergleiche von STRINGS ist das sehr nützlich, da der BASIC-Interpreter Groß- und

Kleinbuchstaben unterscheidet. Dies ist nicht immer erwünscht. So können beim Vergleich STRINGS in Kleinbuchstaben verwendet werden, so daß überhaupt keine Großbuchstaben mehr vorkommen.

UPPER$

Dies ist die Umkehrung von LOWER$, da hier alle Kleinbuchstaben in Großbuchstaben verwandelt werden können. Er ist in ähnlicher Weise zu benutzen wie seine Umkehrung.

SPACE$

Wenn eine bestimmte Anzahl von Leerzeichen hintereinander benötigt werden, kann dieser Befehl benutzt werden. Dies ist immer dann sinnvoll, wenn ein Freiraum geschaffen werden soll. So können bei der Druckerausgabe eine Anzahl von Spalten übersprungen oder der Bildschirm gezielt teilweise gelöscht werden. So löscht beispielsweise der Befehl PRINT SPACE$(1000) den Bildschirm ganz.

STRING$

Dieser Befehl ist SPACE$ sehr ähnlich, nur mit dem Unterschied, daß jetzt ein beliebig langer STRING auch für ein anderes Zeichen erstellt werden kann. So ist es möglich, einen Teil des Bildschirms (bzw. eines WINDOWS) oder den ganzen Bildschirm mit einem ausgesuchten Zeichen zu füllen.

2.1.5 Datenspeicherung

SAVE

Mit SAVE können Programme auf Kassette oder Diskette gespeichert werden. Ein Name muß in Anführungstrichen angegeben werden. Dieser Befehl kann nicht verwendet werden, um Daten zu speichern.

LOAD

LOAD lädt ein Programm von Diskette oder Kassette. Auch hier muß ein Name in Anführungsstrichen angegeben werden. Wird der Kassettenrecorder benutzt, genügt es, nur ein Anführungszeichen einzugeben. Dann wird das nächste Programm auf der Kassette geladen.

OPENOUT

Dieser Befehl wird benötigt, um Daten auf Kassette oder Diskette zu speichern. Mit OPENOUT wird dem Interpreter mitgeteilt, daß Daten auf ein peripheres Speichergerät geschrieben werden sollen. Die Ausgabe der Daten geschieht mit einem PRINT-Befehl und zwar mit PRINT#9. Die Daten werden zunächst in einen 2K-großen Speicher des Computers geschrieben und erst dann auf das Peripheriegerät, wenn der Speicher voll ist oder wenn mit CLOSEOUT dem System mitgeteilt wird, daß die Datenausgabe beendet ist. Dem OPENOUT-Befehl muß ein Dateiname in Anführungsstrichen folgen.

OPENIN

Dies ist die Umkehrung von OPENOUT. Hier wird dem System mitgeteilt, daß Daten von einem Periphergerät eingelesen werden sollen. Diese werden dann erst in den BUFFER des Computers gelesen und von dort weiterverarbeitet. Das Einlesen geschieht mit INPUT#9 oder LINE INPUT#9.

EOF

Mit EOF kann überprüft werden, ob eine Datei (engl. FILE), die gerade eingelesen wird, an ihrem logischen Ende angelangt ist. Dies ist zum Beispiel folgendermaßen zu verwenden:

```
1000 OPENIN"DATEN"
1010 INPUT#9,A$(I)
1020 IF EOF THEN RETURN
1030 I=I+1:GOTO 1010
```

In Zeile 1020 wird überprüft, ob die Datei am Ende ist. Ist das der Fall, wird mit RETURN zurückgesprungen, andernfalls wird in Zeile 1030 die Indizierung um 1 erhöht und dann in Zeile 1010 das

nächste Element eingelesen.

SPEEDWRITE

Der Kassettenrecorder läßt die Möglichkeit zu, in 2 verschiedenen Geschwindigkeiten Daten oder Programme auf Band zu speichern. Es sollte dabei allerdings beachtet werden, daß bei Verwendung der schnellen Speicherung (SPEEDWRITE 1) die Wahrscheinlichkeit für Lesefehler erhöht wird. Diese Möglichkeit der schnellen Speicherung sollte also nur verwendet werden, wenn erstklassiges Bandmaterial in der Kassette zur Verfügung steht und wenn es auf die Geschwindigkeit der Speicherung ankommt. Ansonsten sollte der langsamere und sichere Modus (SPEEDWRITE 0) benutzt werden.

2.2 Dateiverwaltung selbst aufgebaut

Wer schon öfter vor der Aufgabe stand, eine Vielzahl von Daten zu erfassen, zu sammeln und später wieder abzurufen, möchte dafür gerne seinen Computer einsetzen. Folglich ist die Dateiverwaltung eine der wichtigsten Programmieraufgaben für jeden Computer und zugleich auch eine der schwierigsten. Wir wollen in diesem Kapitel einen Einblick in die Dateiverwaltung geben und anhand eines kleinen Adressenverwaltungsprogramms die Programmiertechnik einer Dateiverwaltung erklären. Auf den Unterschied in der Behandlung von numerischen und alphanumerischen Daten (STRINGS) gehen wir dann anschließend anhand einer numerischen Dateiverwaltung ein. Doch wenden wir uns zunächst einigen grundlegenden Begriffen zu.

2.2.1 Einführung

In der Computerwelt wird häufig der Begriff "Datei" verwendet. Was hinter diesem Begriff steckt, wird uns sofort klar, wenn wir ihn durch den Begriff "Kartei" ersetzen. Der Karteikasten dabei ist unser Computer, die Kartei selbst besteht aus vielen Einzeldaten. Eine Kartei besteht in der Regel aus vielen einzelnen Kar-

teikarten. Auf jeder diesen Karten steht die gesamte Information über einen bestimmten Gegenstand oder über eine bestimmte Person.

Durch Entnehmen einer Karteikarte ist jederzeit der Zugriff auf diese Informationen möglich. Allerdings kann auf eine Karteikarte nicht sofort zugegriffen werden, man muß erst wissen, wo sie steht. Um das Suchen zu vereinfachen, werden die Karteikarten deshalb meistens sortiert. In der Regel wird dafür eine alphabetische Reihenfolge gewählt. Es werden allerdings auch andere Sortierarten verwendet, je nach Aufgabenstellung der Kartei.

In der elektronischen Datenverarbeitung heißt jede dieser Karteikarten Datensatz. In einem Datensatz stehen also alle diese Informationen, die in der herkömmlichen Kartei auf einer Karteikarte stehen. Sind die Karteikarten einmal nach einem bestimmten Schlüssel sortiert, ist es in der Regel nicht so schnell möglich, diesen Schlüssel zu ändern. Die Kartei bleibt also immer in der einmal vorgegebenen Form erhalten. Dies ist der große Vorteil des Computers. Er ist in der Lage, die Datensätze einer Datei beliebig oft und nach beliebig vielen Schlüsseln umzusortieren.

Jetzt müssen wir uns noch mit den einzelnen Einträgen auf einer solchen Karteikarte beschäftigen. In der Computersprache heißt jeder dieser Einträge Datenfeld. Während es bei der herkömmlichen Kartei möglich ist, auf einer Karteikarte noch irgendwo handschriftlich zusätzliche Informationen unterzubringen, ist dies bei einer Computerdatei nicht mehr möglich. Hier enthält jede Karteikarte, um bei unserem Vergleich zu bleiben, die gleichen Informationen und die gleichen Einträge. Wir müssen also beim Aufbau einer Dateiverwaltung entscheiden, welche Datenfelder benötigt werden.

Fassen wir das Gesagte noch einmal kurz zusammen: Der Oberbegriff ist die Datei. Dies wäre im herkömmlichen Sinne zum Beispiel eine Adressenkartei. Jeder einzelnen Karteikarte dieser Adressenkartei entspricht in der EDV der Datensatz, die Datenfelder wären dann zum Beispiel Name, Anschrift und Telefonnummer. Wenn Ihnen diese Zusammenhänge noch nicht ganz klar sind, sollten Sie sich das Prinzip noch einmal anhand folgender Grafik verdeutlichen.

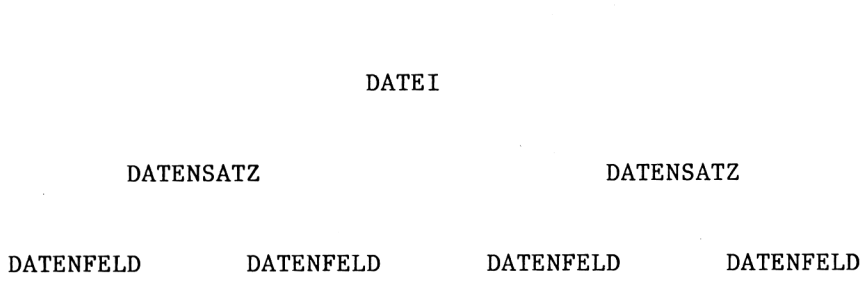


Bild 2.1. Darstellung eines Dateiaufbaus

Das Verständnis dieser Zusammenhänge ist unbedingte Voraussetzung, um den Aufbau einer Dateiverwaltung nachvollziehen zu können. Im folgenden werden wir uns mit der Art der Datenspeicherung und dem Aufbau einer Adressenverwaltung beschäftigen.

2.2.2 Das Prinzip der Datenverwaltung: sequentielle Datei

Nachdem wir uns bisher mit dem formalen Aufbau einer Datei beschäftigt haben, betrachten wir nun die Art der Datenspeicherung. Hier gibt es eine Vielzahl von Möglichkeiten, um Daten auf einem externen Speicher abzulegen. Mit einem Kassettenrecorder lassen sich nur sequentielle Dateien verwirklichen. Da diese Art der Datenspeicherung aber auch mit einer FLOPPY DISK möglich ist, beschränken wir uns an dieser Stelle auf die Darstellung der sequentiellen Datei.

Übersetzt bedeutet sequentiell "nacheinander". Für eine Datei bedeutet dies, daß die einzelnen Datensätze hintereinander abgespeichert sind. So ähnlich, wie bei einem Tonband die Musikstücke hintereinander aufgenommen werden, befinden sich, um bei unserem Beispiel zu bleiben, bei einer sequentiellen Datenspeicherung die einzelnen Adressen nacheinander auf dem Band. Um eine bestimmte Adresse zu finden, müßte man den genauen Zählerstand des Bandzählwerkes kennen oder sich von vorne bis zu der betreffenden Stelle alle gespeicherten Daten anhören. Die Suche über den Zäh-

lerstand dürfte wohl etwas zu umständlich sein, denn hier müßte, damit der Zählerstand genau abgelesen werden kann, jedesmal das Band an seinen Anfang zurückgespult, der Zählerstand auf Null gesetzt und dann zur entsprechenden Stelle vorgespult werden. Da dieses Verfahren nicht nur zu aufwendig ist, sondern auch noch eine handschriftliche Datei der Zählerstände vorhanden sein muß, können wir dieses Verfahren so nicht anwenden.

Um unseren Computer besser einsetzen zu können, gibt es prinzipiell zwei Möglichkeiten: Erstens kann der Computer, wenn nach einer bestimmten Adresse gesucht wird, den ersten Datensatz einlesen und mit dem Suchwort vergleichen. Findet er keine übereinstimmung, liest er den nächsten Datensatz ein und vergleicht usw., bis die zu suchende Adresse gefunden ist oder die ganze Datei durchgelesen wurde. Bei der zweiten Möglichkeit werden einmal alle Datensätze in den Speicher des Computers eingelesen und können dort weiterverarbeitet werden.

Beide Möglichkeiten haben ihre Vor- und Nachteile. Bei der ersten Methode kann die Suche nach einem Datensatz unter Umständen so lange dauern, daß es sich lohnt, zwischendurch einen Kaffee trinken zu gehen. Bei der zweiten Möglichkeit ist die Größe der Datei auf den Speicherplatz im Computer begrenzt. Diese Begrenzung gibt es teigentlich bei der ersten Art nicht, dafür ist die Geschwindigkeit der zweiten Methode um ein Vielfaches höher.

Wir entscheiden uns hier für die zweite Methode, da sie für kleinere Datenmengen die praktikabelste Lösung darstellt. Wenn Sie so große Datenmengen verarbeiten wollen, daß diese nicht mehr in den Speicher des Schneider CPC passen, ist dies bei dieser Art der Datenspeicherung und Datenverarbeitung auch möglich. Sie könnten so zum Beispiel Ihre Adressenkartei in mehrere Unterdateien gliedern, die jede für sich in den Speicher paßt. So wäre es zum Beispiel möglich, eine Adressenkartei, die alphabetisch sortiert ist, aufzuspalten in die Dateien A bis H und I bis Z. So haben Sie zwei unterschiedliche Dateien und können die doppelte Datenmenge verarbeiten, als wenn Sie diese Aufteilung nicht vornehmen würden. Mit etwas Geschick läßt sich so auch mit dem Kassettenrecorder eine komfortable Datenverwaltung aufbauen.

Bevor die Daten auf Band oder Diskette geschrieben werden können, muß zunächst die Datei eröffnet werden. Vorher müssen wir allerdings noch festlegen, welchen Namen die Datei erhalten soll. Nennen wir sie hier einfach mal Datei. Dann wird die Datei eröffnet

mit dem Befehl OPENOUT"DATEI". Mit PRINT#9 können jetzt Daten auf Band geschrieben werden. Ist die Datenausgabe beendet, muß die Datei mit CLOSEOUT geschlossen werden. Sehen wir uns das an einem kleinen Beispiel einmal an. Wir wollen jetzt mit einem kleinen Programm eine Adresse auf Band schreiben.

```
10 REM ================================
20 REM SCHREIBEN EINER ADRESSE AUF BAND
30 REM ================================
40 CLS
50 INPUT "Name:";a$(1)
60 INPUT "Strasse";a$(2)
70 INPUT "Ort:";a$(3)
80 OPENOUT "DATEI"
90 FOR I=1 TO 3 : PRINT#9,a$(I) : NEXT
100 CLOSEOUT
```

Bei diesem Programm werden in den Zeilen 50 bis 70 die drei Datenfelder abgefragt, die auf Band gespeichert werden sollen. In Zeile 80 wird die Datei eröffnet und in Zeile 90 wird in einer FOR-NEXT-Schleife die Datei, die aus drei Datenfeldern besteht, abgespeichert, in Zeile 100 wird die Datei geschlossen. Das Schließen der Datei ist unbedingt notwendig, da mit dem PRINT#9-Befehl die Daten zunächst in einen Datenspeicher im Computer geschrieben werden, der entweder bei Überschreiten dessen Speicherkapazität (2000 BYTE) oder durch den Befehl CLOSEOUT auf Band geschrieben wird.

Auf Band gespeicherte Daten müssen aber auch wieder in den Speicher gelesen werden. Dazu muß wieder eine Datei eröffnet, danach über den Befehl INPUT die Daten eingelesen und mit dem Befehl CLOSEIN die Datei wieder geschlossen werden. Schauen wir uns das Programm als Pendant zur oben abgedruckten Routine an.

```
10 REM ============================
20 REM LESEN EINER ADRESSE VON BAND
30 REM ============================
40 CLS
50 OPENIN "DATEI"
60 FOR I=1 TO 3 : INPUT#9,a$(I) : NEXT
70 CLOSEIN
80 PRINT "Name:";a$(1)
90 PRINT "Strasse:";a$(2)
100 PRINT "Ort:";a$(3)
```

Hier wird jetzt in Zeile 50 die Datei eröffnet, in Zeile 60 mit einer FOR-NEXT-Schleife die drei Datenfelder eingelesen, und in Zeile 70 wird die Datei geschlossen. Die Zeilen 80 bis 100 schreiben die drei Datensätze auf den Bildschirm.

Damit wäre das Prinzip der Datenspeicherung in seinen Grundzügen erklärt. Für die Besitzer einer Floppy-Diskstation sei noch hinzugefügt, daß sich dieses Verfahren analog auch für eine Datenspeicherung auf Diskette verwenden läßt. Dazu muß zu Beginn des Programms dem Computer lediglich mitgeteilt werden, daß die Datenspeicherung auf Diskette erfolgen soll. Ansonsten bleiben alle Befehle identisch. Dieses Verfahren werden wir später dann bei der Entwicklung unserer Adressenkartei benutzen.

2.2.3 Entwicklung eines Adressenprogramms

2.2.3.1 Einige Vorüberlegungen

Nachdem wir uns jetzt die theoretischen Grundlagen einer Dateiverwaltung erarbeitet haben, wollen wir ein einfaches Dateiverwaltungsprogramm schreiben. Dazu sind einige Vorüberlegungen notwendig. Wir müssen zunächst einmal festlegen, was das Programm alles können soll. Es ist selbstverständlich, daß es möglich sein muß, Daten einzugeben, Daten auf Kassette zu speichern, die abgespeicherten Daten wieder einzulesen, Daten zu suchen, zu sortieren und ausdrucken zu lassen. Das Programm soll also insgesamt 6 Funktionen beinhalten.

Wir müssen außerdem noch festlegen, wie jeder einzelne Datensatz beschaffen sein soll. Wir entscheiden uns hier für folgende Felder:

Name, Vorname, Straße, Postleitzahl, Ort und Telefonnummer

Das sind 6 Datenfelder in jedem Datensatz. Da es sich bei diesen Datenfeldern um alphanumerische Daten handelt, müssen wir sogenannte STRINGS verwenden. Dem Computer wird mitgeteilt, daß es sich um einen STRING handelt, indem der Variablenname um das Dollarzeichen ergänzt wird. Zum Beispiel: A$. Da wir mehr als einen

STRING im Speicher benötigen, müssen wir die STRINGS mit einer Numerierung versehen. Dies bezeichnet man als Indizierung. Die Nummer des Datensatzes wird beim STRING-Namen in Klammern gesetzt, zum Beispiel: A$(1).

Bei unserem Problem benötigen wir aber mehrere Datensätze mit jeweils 6 Feldern. Deshalb kommt für uns nur eine zweidimensionale Indizierung in Frage. Dabei legen wir uns so fest, daß die erste Zahl in der Klammer die Datensatznummer angibt und die zweite die Nummer des Datenfeldes, zum Beispiel: D$(1,1) soll das erste Datenfeld im ersten Datensatz angeben. In unserem Beispiel ist das erste Datenfeld der Name. Deshalb bedeutet D$(1,1) der erste abgespeicherte Name.

Wir müssen jetzt dem Computer noch mitteilen, wieviele Datensätze und Datenfelder wir insgesamt verarbeiten wollen, das heißt wir müssen unsere Datei zunächst dimensionieren. Dies ist notwendig, da ohne die Dimensionierung nur insgesamt 11 Datensätze verarbeitet werden könnten. Wir dimensionieren unsere Datei zunächst auf 200 Datensätze mit 6 Datenfeldern.

Wir schreiben uns also zunächst ein kleines Vorspannprogramm, in dem in Zeile 80 die Dimensionierung vorgenommen wird. Vorher werden noch einige Farbregister gesetzt und 3 WINDOWS definiert. Dies ist notwendig, damit wir einen ansprechenden Bildaufbau erhalten.

Bei der Definition der WINDOWS geben die ersten beiden Zahlen nach der WINDOW-Nummer an, wie breit das WINDOW definiert werden soll. Alle 3 WINDOWS werden hier von Spalte 1 bis Spalte 40 definiert, umfassen also die gesamte Bildschirmbreite. Die nächsten beiden Zahlen geben jeweils die Zeilen des WINDOWS an. Dabei geht das erste WINDOW von Zeile 1 bis Zeile 5, das zweite WINDOW von Zeile 6 bis Zeile 20 und das dritte WINDOW von Zeile 21 bis Zeile 25. Sie sehen also, der gesamte Bildschirm von 40 Spalten und 25 Zeilen ist so in drei WINDOWS aufgeteilt worden.

Im oberen Teil des Bildschirms (WINDOW 1) soll uns ab sofort mitgeteilt werden, in welchem Teil des Programmes wir uns befinden, im mittleren Teil (WINDOW 2) sollen Daten ein- bzw. ausgegeben werden und im unteren Teil (WINDOW 3) soll durch das gesamte Programm geführt werden. Damit die Bildschirmbereiche gut voneinander unterschieden werden können, wurden sie in verschiedenen Farben gestaltet. Um diesen Effekt zu erreichen, wurden die Zeilen

50 bis 70 eingefügt. Es folgt nun der erste Teil unseres kleinen Adressenprogramms, das genau die oben aufgeführten Funktionen erfüllt. Es handelt sich dabei lediglich um den Vorspann und hat für sich betrachtet noch keine große Funktionen.

```
10 REM ===========
20 REM VORSPANNADR
30 REM ===========
40 CLS
50 INK 0,1 : INK 1,24 : INK 2,3 : BORDER 1
60 WINDOW#1,1,40,1,5 : WINDOW#2,1,40,6,20: WINDOW#3,1,40,21,25
70 PAPER #1,1 : PEN#1,3 : PAPER#3,1: PEN#3,3
80 DIM D$(200,6)
```

Die Fortsetzung dieses Programms können Sie den nächsten Abschnitten entnehmen oder das gesamte Programmlisting am Ende dieses Kapitels betrachten.

2.2.3.2 Menüsteuerung

Im vorherigen Abschnitt haben wir festgelegt, daß unser Programm insgesamt 6 Funktionen erhalten soll. Es muß natürlich auch irgendwie möglich sein, nach Belieben die Funktion aufzurufen, welche gerade gewünscht wird. Es ist natürlich nicht erwünscht, daß die Funktionen alle hintereinander ablaufen, denn meist wird nur eine dieser Funktionen benötigt.

Die freundlichste Lösung für dieses Problem ist die sogenannte Menüsteuerung. Dabei gibt es in einem Programm ein Hauptprogramm und mehrere Unterprogramme, die jeweils vom Hauptprogramm aufgerufen werden können. So müssen wir es in unserem Beispiel auch durchführen, indem wir ein Hauptprogramm und 6 Unterprogramme schreiben, das heißt für jede Problemlösung ein Unterprogramm. In dem Hauptprogramm müssen auf dem Bildschirm die einzelnen Unterprogramme aufgelistet sein, und wir müssen über die Tastatur eingeben, welches Unterprogramm gewünscht wird. Die Auflistung dieser Unterprogramme nennt man auch Auswahlliste oder, eleganter, Menü. Daher kommt auch der Name Menü-Technik. Zunächst wollen wir einmal unser Menüprogramm eingeben.

```
100 REM ========
110 REM MENUEADR
120 REM ========
130 CLS#1: CLS#2 : CLS#3
140 LOCATE#1,16,3: PRINT#1,"M E N U E"
150 LOCATE#2,13,3:PRINT#2,"1  DATEN EINLESEN"
160 LOCATE#2,13,5:PRINT#2,"2  DATEN SPEICHERN"
170 LOCATE#2,13,7:PRINT#2,"3  EINGEBEN"
180 LOCATE#2,13,9: PRINT#2,"4  SUCHEN"
190 LOCATE#2,13,11:PRINT#2,"5  SORTIEREN"
200 LOCATE#2,13,13:PRINT#2,"6  DRUCKEN"
210 CLS#3:LOCATE#3,3,3: INPUT#3,"WELCHE NUMMER";A$
220 A=VAL(A$): IF A<1 OR A>6 THEN 210
230 ON A GOSUB 1000,2000,3000,4000,5000,6000
240 GOTO 130
1000 RETURN
```

Dies ist nun unser Hauptprogramm, von dem aus alle Unterprogramme aufgerufen werden. Wie geschieht das? Als erstes werden in Zeile 130 3 WINDOWS gelöscht; Zeile 140 schreibt in WINDOW 1 den Programmteil. Wir befinden uns derzeit im Menü. Zeile 150 bis Zeile 200 gibt unsere Auswahlliste im mittleren Fenster aus. Mit Hilfe der LOCATE-Befehle wird jeweils Zeile und Spalte bestimmt, an der die jeweilige Zeile des Menüs stehen soll. Jede Zeile des Menüs beginnt in Spalte 13. Die erste Zeile beginnt in Zeile 3 des WINDOWS 2, die anderen folgen mit jeweils zweizeiligem Abstand.

In Zeile 210 wird dann in WINDOW 3 gefragt, welche Nummer gewünscht wird. Diese Nummer wird über INPUT abgefragt und dem STRING A$ zugeordnet. In Zeile 210 wird dieser STRING in eine Zahl umgewandelt und überprüft, ob Zahlen eingegeben wurden, die nicht den Zahlen der Auswahlliste entsprechen. Zulässig sind nur Zahlen zwischen 1 und 6. Wird eine größere Zahl eingegeben, wird die Abfrage der Zahlen noch einmal durchgeführt. Wird ein Buchstabe eingegeben, wird der Zahl A über den VAL-Befehl eine Null zugeordnet, dann würde auch die Abfrage wiederholt werden. Die wichtigste Zeile in unserem Programm ist die Zeile 230. Sie gibt an, welches Unterprogram aufgerufen werden soll, wenn eine bestimmte Zahl gedrückt ist.

Hinter dem Befehl ON A GOSUB stehen insgesamt 6 Zahlen. Da A einen Wert zwischen 1 und 6 annehmen kann, wählt der Interpreter die Zahl aus, die an der A. Stelle in dieser Liste steht. Wird die Zahl 4 eingegeben, wählt der Interpreter die 4. Zahl, das ist hier die Zahl 4000 und springt in ein Unterprogramm, das ab der Zeilennummer 4000 beginnt. Da es sich um eine Subroutine handelt, die mit RETURN abgebrochen wird, springt das Programm bei Beendigung dieser Subroutine zu Zeile 240. Wir müssen dort angeben, wo das Programm fortgesetzt werden soll. Wir wollen natürlich unser Programm mit dem Menü fortsetzen, da wir unter Umständen ein neues Unterprogramm aufrufen wollen. Wir müssen von dort wieder zu Zeile 130 springen und damit im Prinzip das Programm neu beginnen.

Wenn unser Programm später von Ihnen einmal um zusätzliche Funktionen erweitert werden soll, ist dies ganz einfach möglich, indem in unserer Auswahlliste ein neuer Punkt, zum Beispiel 7, eingefügt wird, in Zeile 220 die Begrenzung von 6 auf 7 geändert und in Zeile 230 eine zusätzliche Zeilennummer hinten angehängt wird. Unser Programm besteht aus einzelnen Modulen, die lediglich zusammengefügt werden. Dadurch ist eine Erweiterung kein Problem mehr, es muß dann nur noch eine Möglichkeit geschaffen werden, sie aus dem Menü heraus aufrufen zu können.

Sollte der Platz auf dem Bildschirm für zusätzliche Programmpunkte nicht ausreichen, kann auch an einer freien Stelle im Programm ein neues Menü eingefügt werden, das wiederum von diesem Menü aufgerufen wird, um dann von dort aus die Unterprogramme aufzurufen. Es muß nur immer wieder darauf geachtet werden, daß in das ursprüngliche Menü zurückgegangen wird. So ist es möglich, mit sich überlappenden Menüs auf mehreren Ebenen zu arbeiten und so eine sukzessive Auswahl innerhalb komplizierter Programme durchzuführen.

Die Untermenüs sollten selbstverständlich thematisch geordnet sein. So wäre es zum Beispiel möglich, unserem Aufruf "Drucken" ein neues Menü einzufügen, das mehrere verschiedene Ausdruckmodi zur Verfügung stellt. In unserem Programm werden wir unter diesem Programmpunkt lediglich eine Adressenliste drucken lassen. Eine eventuelle Erweiterung bleibt dabei Ihnen überlassen.

2.2.3.3 Eingabe der Daten

In diesem Abschnitt wollen wir uns mit dem Unterprogramm 3, Eingabe, beschäftigen. Sie soll im Programm ab der Zeilennummer 3000 zu finden sein. Bevor wir das Programmlisting betrachten, müssen wir aber noch einen Begriff vorher klären, nämlich den Begriff der Eingabemaske.

Bei der Eingabe von Daten in einen Computer spricht man häufig von der Eingabemaske. Diese Maske ist ein wesentliches Element eines Dateiverwaltungsprogramms und läßt schon einige Rückschlüsse auf den Komfort eines Programms zu. Was steckt hinter diesem Begriff? Wir wissen, daß in unserem Programm insgesamt 6 Datenfelder abgefragt werden sollen. Die Eingabemaske stellt jetzt den Bildaufbau dar, innerhalb dessen die Bilder eingegeben werden können.

Bei einer komfortablen Eingabemaske werden zunächst alle Namen der Datenfelder auf dem Bildschirm dargestellt und darüber hinaus ein Raum angegeben, in dem die Eingabe für dieses Datenfeld erfolgen soll. Die Ausgabe aller Datenfeldnamen ist insbesondere deshalb wichtig, damit der Bediener eines solchen Programms nicht immer im Kopf haben muß, welche Felder noch eingegeben werden müssen, sondern es auf einen Blick am Bildschirm sehen kann. In unserem Programm wollen wir eine solche Eingabemaske definieren. Sie soll so aussehen, daß hinter dem Namen des Datenfeldes eine Linie andeutet, wo die Eingabe in dieses Datenfeld erfolgen soll. In der 1. Zeile soll selbstverständlich der Name, in der 2. Zeile der Vorname, in der 3. Zeile die Straße und in der 4. Zeile der Ort abgefragt werden. Da Postleitzahl und Telefonnummer weniger Platz beanspruchen als die anderen Zeilen, ist es möglich, beide gemeinsam in einer Bildschirmzeile abzufragen. Dabei muß natürlich nach dem Drücken der ENTER-Taste der Cursor zur Eingabe auf das nächste Feld springen. In der letzten Bildschirmzeile möchten wir noch gerne die Nummer des Datensatzes haben, der gerade eingegeben wird.

Mit Hilfe des LOCATE-Befehls sind wir in der Lage, die Datenfeldnamen an ihren vorgesehenen Orten ausdrucken zu lassen. Die Linie zur Eingabe des Datenfeldes kann mit Hilfe des STRING$-Befehls kreiert werden. Wir wählen für die Linie das Zeichen 195 des ASCII-Codes aus.

Mit dieser Beschreibung sind eigentlich schon die Zeilen 3050 bis 3100 des folgenden Unterprogrammes erklärt. Diese Zeilen stellen die eigentliche Bildschirmmaske dar.

```
3000 REM =============
3010 REM E I N G A B E
3020 REM =============
3030 a$="E I N G A B E ":I=ANZAHL+1
3040 CLS #1:CLS #2:CLS #3:LOCATE#1,INT((40-LEN(A$))/2),3:PRINT#1,A$
3050 LOCATE#2,3,3:PRINT#2,"Name:";STRING$(32,95)
3060 LOCATE#2,3,5:PRINT#2,"Vorname:";STRING$(29,95)
3070 LOCATE#2,3,7:PRINT#2,"Strasse:";STRING$(29,95)
3080 LOCATE#2,3,9:PRINT#2,"Ort:";STRING$(24,95)
3090 LOCATE#2,3,11:PRINT#2,"PLZ:";STRING$(8,95):LOCATE#2,15,11:PRINT#2
,"Tel.:";STRING$(15,95)
3100 LOCATE#2,3,13:PRINT#2,"Dateinummer:";i
3110 LOCATE#2, 8,3: INPUT#2,"",D$(I,1)
3120 LOCATE#2, 11,5: INPUT#2,"",D$(I,2)
3130 LOCATE#2, 11,7: INPUT#2,"",D$(I,3)
3140 LOCATE#2, 7,9: INPUT#2,"",D$(I,5)
3150 LOCATE#2, 7,11: INPUT#2,"",D$(I,4)
3160 LOCATE#2, 20,11: INPUT#2,"",D$(I,6)
3170 LOCATE #3,2,3:INPUT#3,"RICHTIG (J/N)",J$:IF J$="N" OR J$="n" THEN
 CLS#2: GOTO 3050
3180 CLS#3:LOCATE #3,2,3:INPUT#3,"Weitere Eingabe ? (J/N)",J$:IF J$="N
" OR J$="n" THEN ANZAHL=I:RETURN
3190 IF j$ ="j" OR j$="J" THEN CLS#2:CLS#3: I=I+1: GOTO 3050
3200 GOTO 3180
```

Eine Eingabemaske allein reicht für eine Eingabe von Daten noch nicht aus. Wir brauchen jetzt noch eine Routine, die die Daten an den jeweils vorgesehenen Feldern abfragt. Zur Datenabfrage gibt es eine Vielzahl von Möglichkeiten. Am einfachsten ist eine Eingabe mit Hilfe des INPUT-Befehls. Wir setzen zur Eingabe mit Hilfe von LOCATE-Befehlen den Cursor an die jeweils vorgesehenen Stellen und fragen dann mit einem leicht modifizierten INPUT-Befehl die Daten ab.

Es wird Sie sicherlich wundern, warum der INPUT-Befehl in dieser Form gewählt wurde. Dies hat seine Begründung in der Syntax des Befehls. Es ist möglich, mit Hilfe des INPUT-Befehls noch einen Text auf dem Bildschirm ausdrucken zu lassen. Wird diese Möglichkeit in Anspruch genommen, erhält der Benutzer die Auswahl, hin-

ter diesem Text ein Komma oder ein Semikolon als Trennzeichen zu benutzen. Wird ein Semikolon benutzt, wird bei der INPUT-Abfrage ein Fragezeichen auf dem Bildschirm ausgegeben; dagegen erfolgt bei der Benutzung des Kommas eine Abfrage ohne dieses Fragezeichen.

In unserer Eingabemaske würde das Fragezeichen im INPUT-Befehl stören. Jetzt ist es aber ohne Ausgabe von Text nicht möglich, das Fragezeichen durch Benutzung des Kommas als Trennzeichen zu verwenden. Bitte verwechseln Sie dieses Komma nicht mit dem Komma, das auf den INPUT#2-Befehl folgt. Dieses Komma wird nötig, da es sich um einen INPUT in einem WINDOW handelt. Es stellt also kein Trennzeichen dar, sondern gehört zum INPUT-Befehl. Die Alternative, die sich uns stellt, ist entweder das Fragezeichen in der Eingabemaske in Kauf zu nehmen oder einen Text auf dem Bildschirm zusätzlich mit ausdrucken zu lassen. Keine von beiden Alternativen ist für uns befriedigend. Deshalb müssen wir den Computer überlisten, indem wir ihm den Befehl geben, einen Text auszudrucken, der aus keinem Zeichen besteht. Dies wird dadurch gewährleistet, daß hinter dem INPUT #2-Befehl zwei Anführungszeichen direkt hintereinandergesetzt werden. Der auszudruckende Text steht ja bekanntlich immer zwischen zwei Anführungszeichen. Wenn die Anführungszeichen direkt aufeinanderfolgen, bedeutet dies, daß der Text leer ist. Man spricht dann von einem sogenannten LEERSTRING. Da wir jetzt einen auszudruckenden STRING haben, ist es möglich, das Komma als Trennzeichen zu verwenden und somit das Fragezeichen in der INPUT-Abfrage zu vermeiden.

Nach erfolgter Eingabe der 6 Datenfelder muß es möglich sein, die Eingabe auf Fehler zu untersuchen und etwaige Fehler zu korrigieren. Um diesen Zweck zu erreichen, wird in Zeile 3170 gefragt, ob die Eingabe richtig war. Wenn die Eingabe fehlerhaft war und bei der Abfrage N eingegeben wird, wiederholt das Programm die Eingaberoutine für diese Dateinummer. Wird gar nichts eingegeben, das heißt wird lediglich die ENTER-Taste gedrückt oder wird ein J eingegeben, fragt das Programm, ob die Eingabe fortgesetzt werden soll. Ist dies der Fall, zählt das Programm den Index I um eins weiter und fragt dann die nächste Dateinummer ab. Soll das Programm beendet werden, wird die Zahl I, das ist die Anzahl der bisher eingegebenen Daten, der Variablen Anzahl zugeordnet, und das Programm springt zum Menü zurück.

2.2.3.4 Datenspeicherung

Nachdem nun die Eingaberoutine erstellt ist, wollen wir die Daten auf Band abspeichern. Dazu benutzen wir das oben beschriebene Verfahren der sequentiellen Datenspeicherung. Als Namen für die Datei wählen wir "Adressendatei" und eröffnen mit diesem Namen die Datei auf Band. Da wir ein zweidimensionales Datenfeld haben, müssen wir zur Abspeicherung zwei ineinander verschachtelte FOR-NEXT-Schleifen benutzen. Die erste Schleife soll die Anzahl der Datensätze durchlaufen, während die zweite Schleife die Anzahl der einzelnen Datenfelder durchläuft, d.h. jeweils bis 6 zu zählen hat. Nachdem alle Daten abgespeichert sind, wird das Datenfeld mit CLOSEOUT geschlossen und mit RETURN wieder zum Menü zurückgesprungen.

```
2000 REM ============
2010 REM SPEICHERNADR
2020 REM ============
2030 CLS#1 : CLS#2 : CLS#3
2040 LOCATE 3,12:LOCATE #1,15,3: PRINT#1,"SPEICHERN"
2050 OPENOUT"ADRESSENDATEI"
2060 FOR I=1 TO ANZAHL
2070 FOR J=1 TO 6
2080 PRINT#9,D$(I,J)
2090 NEXT J : NEXT I
2100 CLOSEOUT
2110 RETURN
```

Beim Speichern der Daten gibt der Computer die gleichen Kommandos aus wie beim Speichern eines Programms, das heißt er meldet sich mit der Aufforderung PRESS REC AND PLAY THEN ANY KEY; und gibt während des Speicherns die Anzahl der zu speichernden Blöcke aus, wie wir es bei einer Programmspeicherung gewohnt sind. Diese Meldung möchten wir gerne im mittleren Teil des Bildschirms haben. Dafür ist es notwendig, den Cursor auf die Bildschirmmitte zu setzen. Dies geschieht in Zeile 2040 mit dem ersten LOCATE-Befehl. Die nachfolgenden Befehle LOCATE und PRINT schreiben lediglich in den oberen Bereich des Bildschirms.

2.2.3.5 Daten einlesen

Um die auf Band gespeicherten Daten wieder in den Computer einlesen zu können, müssen wir eine eigene Routine schreiben. Das Prinzip des Einlesens ist bereits beschrieben worden. An dieser Stelle wollen wir die Routine für dieses Programm entwickeln.

Da wir beim Abspeichern der Daten die genaue Anzahl der zu speichernden Daten kannten, war es möglich, die Speicherung durch eine FOR-NEXT-Schleife vornehmen zu lassen. Da es aber möglich ist, daß auf dem Band unterschiedlich viele Daten stehen und wir diese Zahl von vornherein nicht wissen, ist dieses einfache Verfahren beim Einlesen nicht möglich.

Das LOCOMOTIVE-BASIC bietet uns für diese Problemstellung eine andere Art von Schleife an, die es in den wenigsten BASIC-Dialekten gibt. Diese Schleifenbildung wird mit den Befehlen WEND und WHILE vorgenommen und ist der Programmiersprache PASCAL entlehnt. Bei dieser Schleifenbildung wird mit dem Befehl WHILE die Bedingung festgelegt, unter der die Schleife wiederholt werden soll, mit dem Befehl WEND dann die Wiederholung durchgeführt. So gesehen ist es auch logisch, daß der WHILE-Befehl am Schleifenbeginn stehen muß, der WEND-Befehl zu Schleifenende, ähnlich wie bei FOR NEXT der FOR TO-Befehl zu Beginn der Schleife steht und der NEXT-Befehl am Ende.

Wir wollen jetzt eine INPUT-Schleife schreiben, die so oft durchlaufen wird, bis das Ende der Datei auf Band erreicht ist. Zur Überprüfung des Endes gibt uns der Computer auch einen Befehl. Dieser Befehl heißt EOF. Das ist die Abkürzung für END OF FILE. EOF nimmt den Wert Null an, wenn das Ende noch nicht erreicht ist und den Wert -1, wenn das Ende erreicht ist. Also müssen wir unsere Schleife solange durchlaufen, wie EOF den Wert Null hat. Dies wurde in Zeile 1070 mit dem Befehl WHILE EOF = 0 formuliert. In Zeile 1110 steht dann der WEND-Befehl. So wird es möglich, eine beliebige Anzahl von Datensätzen einzulesen. Die Datensätze werden mit dem Index I versehen. Da jeder Datensatz aus genau 6 Datenfeldern besteht, können diese einzelnen Datenfelder mit einer FOR-NEXT-Schleife eingelesen werden. Das Datenfeld wird mit dem Index J versehen.

```
1000 REM ========
1010 REM LADENADR
1020 REM ========
1030 CLS#1 : CLS#2 : CLS#3
1040 LOCATE 3,12:LOCATE #1,15,3: PRINT#1,"L A D E N"
1050 I=ANZAHL+1
1060 OPENIN"ADRESSENDATEI"
1070 WHILE EOF=0
1080 FOR J=1 TO 6
1090 INPUT#9,D$(I,J)
1100 NEXT J: I=I+1
1110 WEND
1120 CLOSEIN
1130 ANZAHL=I-1:RETURN
```

Da es sich hier um ein Unterprogramm handelt, muß es mit dem RETURN-Befehl abgeschlossen werden. Vorher wird allerdings der Variablen ANZAHL die Anzahl der insgesamt eingelesenen Datensätze zugeordnet. Zu beachten ist noch, daß eventuell im Speicher vorhandene Datensätze einer Datei nicht überschrieben werden, da in Zeile 1050 der Index I die Zahl ANZAHL + 1 zugeordnet wird. Dies bedeutet nichts anderes, als daß eine eingelesene Datei an eine schon im Speicher befindliche Datei angehängt wird. Sollte dies nicht erwünscht sein, müßte die Zeile 1050 durch I = 1 ersetzt werden; dann werden im Speicher befindliche Dateien zumindest teilweise überschrieben.

2.2.3.6 Daten ausdrucken

Wenden wir uns nun dem Ausdruck einer Liste unserer Datei zu. Die Liste soll folgendes Aussehen haben: Als Überschrift der Liste soll zunächst das Wort Adressendatei ausgedruckt und unterstrichen werden. Danach soll eine Leerzeile gedruckt werden und dann die Adressen. Die Adressen sollen zweizeilig gedruckt sein. In der ersten Zeile befinden sich Name, Vorname und Telefonnummer, während in der zweiten Zeile Straße, Postleitzahl und Ort zu finden sein sollen. Selbstverständlich sollen die einzelnen Datenfelder durch Kommas voneinander getrennt sein. Lediglich vor

der Telefonnummer soll noch einmal das Wort Tel. erscheinen, und Postleitzahl und Ort sollen selbstverständlich nur durch ein Leerzeichen getrennt sein. Diese Problemstellung erfüllt das folgende Programm:

```
6000 REM ==========
6010 REM DRUCKENADR
6020 REM ==========
6030 CLS#1 : CLS#2 : CLS#3
6040 LOCATE#1,15,3:PRINT#1,"D R U C K E N"
6050 LOCATE #3,2,3: INPUT#3,"IST EIN DRUCKER ANGESCHLOSSEN (J/N)";A$

6060 IF LOWER$(A$)<>"j" THEN RETURN
6070 PRINT#8,"A D R E S S E N D A T E I"
6080 PRINT#8,"========================="
6090 PRINT#8
6100 FOR I=1 TO ANZAHL
6110 PRINT#8,D$(I,1)", "D$(I,2)",  TEL.:"D$(I,6)
6120 PRINT#8,D$(I,3)", "D$(I,4)" "D$(I,5)
6130 NEXT
6140 RETURN
```

Die wichtigsten Zeilen sind hierbei die Zeilen 6110 und 6120. In ihnen wird die Liste gedruckt. Hier wird jedesmal ein Datenfeld und dann die Trennung zum nächsten Datenfeld (ein Komma, das in diesem Fall in Anführungszeichen gesetzt sein muß) gedruckt, anschließend das nächste Datenfeld und dann wieder in Anführungszeichen die Trennung zum nächsten Datenfeld. Wie Sie in diesem Beispiel sehen, ist es nicht unbedingt notwendig, daß Trennzeichen zwischen dem jeweiligen Datenfeld und den auszudruckenden Trennungen stehen. Das Programm läuft genauso ab, wenn dort jeweils noch ein Semikolon steht. Setzen Sie ein Komma an diese Stelle, fügt das Programm einige Leerzeichen ein und bringt damit Ihre Formatierung durcheinander.

Diese Druck-Routine wird mit Hilfe einer FOR-NEXT-Schleife ANZAHLmal wiederholt, das heißt so oft, wie Datensätze im Speicher des Programms vorhanden sind. In dieser Variablen war sowohl bei der Eingabe als auch beim Einlesen von Datenträgern die Anzahl der Datensätze abgelegt worden.

In Zeile 6050 wird noch abgefragt, ob überhaupt ein Drucker angeschlossen ist. Dies ist deshalb nötig, da bei nicht angeschlossenem oder ausgeschaltetem Drucker (er befindet sich nicht im ON-LINE-Zustand), der Computer auf dessen READY-Meldung wartet. Das Programm wird nicht mit einer Fehlermeldung abgebrochen, sondern der Computer wartet so lange, bis der Drucker ihm mitteilt, daß er aufnahmebereit ist. Das ist natürlich bei einem nicht angeschlossenen Drucker nicht möglich. Deshalb würde das Programm endlos warten. Um dies zu verhindern, wurde die Abfrage eingefügt, ob überhaupt ein Drucker angeschlossen ist. Falls einmal aus Versehen diese Routine aufgerufen wird, kann man durch das Drücken einer Taste (mit Ausnahme von J) sofort wieder in das Menü zurückgespringen. Da der Computer ein J und ein j unterscheidet, ist die Abfrage in Zeile 6060 mit LOWER$ nötig. Alternativ wäre es möglich, A$ sowohl auf ein J als auch auf ein j zu untersuchen.

2.2.3.7 Suchen von Daten

Beim Suchen von Daten kommt es darauf an, daß das Programm selbständig erkennt, ob eine eingegebene Buchstabenkombination in einem der Datensätze vorkommt. Um dies zu erreichen, gibt es eine Vielzahl von Algorithmen, die sich aber meist nur bei vorsortierten Datensätzen anwenden läßt. Sind die Adressen bereits alphabetisch sortiert und suche ich nach dem Namen Meier, ist es möglich, daß ich zunächst einmal in der Mitte einen Datensatz herausnehme, mir anschaue, ob das entsprechende Datenfeld größer oder kleiner als mein Suchwort ist und dann in der entsprechenden Richtung weitersuche. Werden mehrere solcher zufälligen Zugriffe durchgeführt, läßt sich sehr schnell der Ort des zu suchenden Datensatzes einkreisen. Diese Methode ist sehr schnell, allerdings nur möglich, wenn die Datensätze bereits sortiert sind. Desweiteren muß bei dieser Methode bereits von vornherein bekannt sein, in welchem Datenfeld die gesuchte Buchstabenkombination vorkommen kann. In dieser Methode gibt es eine Vielzahl von Varianten, die alle mehr oder weniger komfortabel sind.

Für unser Problem hier wollen wir einen Such-Algorithmus verwenden, bei dem man die Datensätze nicht vorsortieren braucht und bei dem noch nicht klar ist, in welchem Datenfeld die gesuchte Kombination zu finden ist. Das Programm soll also in allen Daten-

feldern gleichzeitig suchen und alle Datensätze dabei berücksichtigen. Das einfachste Verfahren dabei ist, alle Datensätze nacheinander abzufragen und in jedem Datensatz alle Datenfelder nacheinander zu durchsuchen. Das bedeutet, daß wir zwei ineinander verschachtelte FOR-NEXT-Schleifen benutzen müssen, ähnlich wie beim Abspeichern von Daten.

Wir nehmen uns den ersten Datensatz und durchsuchen alle 6 Datenfelder. Bei Übereinstimmung mit unserer eingegebenen Buchstabenkombination soll der Datensatz auf dem Bildschirm ausgegeben werden. Bei Nichtübereinstimmung soll das nächste Datenfeld genommen werden. Sind alle Datenfelder durchgesucht, gehen wir an den nächsten Datensatz usf., bis alle Datensätze durchgesucht worden sind bzw. bis die Buchstabenkombination gefunden wurde.

Ist eine Übereinstimmung festgestellt, und ist ein Datensatz auf dem Bildschirm ausgegeben worden, so muß es möglich sein, auch noch die restlichen Datensätze zu durchsuchen. Dazu soll das Programm dann fragen, ob die Suche fortgesetzt werden soll. Wird das mit Ja beantwortet, werden die restlichen Datensätze auch noch durchsucht. Wird es mit Nein beantwortet, soll die Suche an dieser Stelle abgebrochen werden.

Zum Suchen bietet uns das komfortable BASIC des SCHNEIDER einen speziellen Befehl an. Dieser Befehl heißt INSTR. Er vergleicht jeweils zwei STRINGS und gibt an, ab welcher Position der zweite STRING im ersten STRING vorkommt. Wir erhalten bei dem Vergleich eine Zahl. Bei Übereinstimmung ist diese Zahl positiv, bei Nichtübereinstimmung ist diese Zahl Null. Dies können wir bei der Suche geschickt ausnutzen, indem wir in jedem Datenfeld mit Hilfe von INSTR abfragen, ob die zu suchende Buchstabenkombination in diesem Datenfeld enthalten ist. Ist sie enthalten, erhalten wir als Ergebnis eine positive Zahl; ist sie nicht enthalten, erhalten wir eine Null. Folglich brauchen wir lediglich bei unserer Abfrage zu überprüfen, ob die INSTR-Funktion ungleich Null ist, um eine Übereinstimmung festzustellen und dann den entsprechenden Datensatz auf dem Bildschirm auszugeben. Diese Technik wird in folgendem Unterprogramm angewandt.

```
4000 REM ======
4010 REM SUCHEN
4020 REM ======
4030 CLS#1 : CLS#2 : CLS#3
4040 LOCATE #1,13,3: PRINT#1,"S U C H E N"
4050 LOCATE#2,3,7:PRINT#2,"WONACH SOLL GESUCHT WERDEN?"
4060 LOCATE#2,3,9: INPUT#2,"",A$
4070 FOR I=1 TO ANZAHL
4080 FOR J=1 TO 6
4090 IF INSTR(D$(I,J),A$)<>0 THEN GOSUB 4140
4100 IF INSTR(D$(I,J),A$)<>0 THEN CLS#3:LOCATE#3,3,3: INPUT#3,"WEITERS
UCHEN (J/N)";B$: IF UPPER$(B$)<>"J"THEN 4120
4110 NEXT J : NEXT I
4120 CLS#3:LOCATE#3,3,3: INPUT#3,"NOCHMAL SUCHEN (J/N)";B$: IF UPPER$(
B$)<>"J" THEN RETURN
4130 GOTO 4030
4140 CLS#2: LOCATE#2,3,3: PRINT#2,"NAME: ";D$(I,1)
4150 LOCATE#2,3,5: PRINT#2,"VORNAME: ";D$(I,2)
4160 LOCATE#2,3,7: PRINT#2,"STRASSE: ";D$(I,3)
4170 LOCATE#2,3,9: PRINT#2,"ORT: ";D$(I,5)
4180 LOCATE#2,3,11: PRINT#2,"PLZ: ";D$(I,4);"   TEL.: ";D$(I,6)
4190 LOCATE#2,3,13: PRINT#2,"DATEINUMMER: ";I
4200 RETURN
```

Zur Ausgabe der Daten verwenden wir eine Subroutine, die ab Zeile 4140 beginnt und die in ihrem Aussehen der Eingabemaske, die wir oben entwickelt haben, entspricht. Wir müssen uns nicht an eine neue Bildschirmaufteilung gewöhnen und finden jedes Datenfeld an dem Platz, an dem wir es auch eingetippt haben.

Der eigentliche Suchalgorithmus des Programms befindet sich in den Zeilen 4070 bis 4110. Die eigentliche Überprüfung findet in Zeile 4090 statt. A$ ist der zu suchende STRING, D$ ist unsere Datei. Wenn eine Übereinstimmung festgestellt wird, springt das Programm in unsere Subroutine, die die Ausgabemaske darstellt und fragt über Zeile 4100, ob die Suche fortgesetzt werden soll.

Dieser Algorithmus ist sicherlich nicht der schnellste Suchalgorithmus, der geschrieben werden könnte. Er erfüllt jedoch alle Anforderungen an einen Suchalgorithmus und ist für alle Problemstellungen verwendbar. Allerdings muß gesagt werden, daß es unter Umständen bei großen Datenmengen lange dauern kann, bis der ent-

sprechende Datensatz gefunden wird, erst recht dann, wenn er sich am Ende der Datei befindet. Wenn es schneller gehen soll, sollten Sie versuchen, den oben beschriebenen Algorithmus zu programmieren. Achten Sie darauf, daß vor einer Suche Ihre gesamte Datei sortiert worden ist.

Mit dem Sortieren wollen wir uns im nächsten Abschnitt beschäftigen. Bei unserem oben vorgestellten Vorschlag ist Sortieren nicht notwendig. Eine andere Möglichkeit, um das Verfahren zu beschleunigen, ist, die Suche auf ein oder zwei Datenfelder zu beschränken. Wenn Sie zum Beispiel wissen, daß Sie nur nach Nachnamen suchen werden, könnten Sie die "FOR J = 1 TO 6"-Schleife weglassen und anstelle der J's im Programm jeweils eine 1 einsetzen. Dann würde die Suche nur in diesem einen Datenfeld durchgeführt, was die Suche natürlich beschleunigt. Das Anpassen der Routine bleibt Ihnen überlassen; unsere Routine wurde allgemein formuliert.

2.2.3.8 Sortieren von Daten

Über das Problem, Daten zu sortieren, ließe sich eigentlich ein eigenes Buch schreiben. Es gibt wahrscheinlich so viele Verfahren und Varianten der einzelnen Algorithmen, wie es Software-Autoren und Buchautoren gibt. Jeder dieser Autoren hat entweder ein eigenes Verfahren entwickelt oder ein gängiges Verfahren für seine eigenen Zwecke so abgewandelt, daß es als neue Variante betrachtet werden kann.

Beim Sortieren geht es darum, Daten nach einer bestimmten Reihenfolge zu ordnen. Dies ist eigentlich nicht schwieriger, als Karten zu sortieren. Jeder hat dies sicher schon einmal gemacht und dabei auch seine eigene Methode entwickelt. Da jeder in der Lage ist, Karten zu sortieren, müßte eigentlich auch jeder in der Lage sein, ein eigenes Verfahren zum Sortieen von Daten zu entwickeln. Sicherlich wird Ihnen auch auf Anhieb ein solches Verfahren einfallen.

Brisant wird dieses Thema eigentlich erst dann, wenn die Frage nach dem Ablauf der Geschwindigkeit und nach der Länge des Programms gestellt wird. Wünschenswert wäre sicherlich ein kurzes Programm, das innerhalb kürzester Zeit alle Daten alphabetisch sortiert. Außerdem müßte es in der Lage sein, bereits vorsortier-

te Dateien schneller zu sortieren, als eine Datei, die so durcheinander ist, wie es ein Kartenspiel sein sollte, bevor die Karten ausgegeben werden. Wenn Ihnen dieses ideale Verfahren auf Anhieb nicht einfällt, sollten Sie sich keine grauen Haare wachsen lassen, andere Programmierer haben es auch noch nicht entdeckt. Aus diesem Grund möchten wir Ihnen an dieser Stelle mehrere Sortierverfahren vorstellen, die alle ihre Vor- und Nachteile haben.

BUBBLE-SORT-Verfahren

Beginnen wir gleich mit dem einfachsten Sortierverfahren, dem BUBBLE-SORT-Verfahren. Der Name dieses Verfahrens rührt daher, daß hier, so ähnlich wie Luftblasen in Wasser nach oben steigen, um schwereren Teilen unten Platz zu machen, in der Datei nach und nach die kleineren (leichteren) Daten nach oben geschoben werden, um so für die größeren (schwereren) Daten Platz zu machen. Es gibt böse Zungen, die behaupten, der Name BUBBLE-SORT sei nur dadurch gerechtfertigt, daß sich die Sortierzeit mit zunehmender Datenmenge wie eine Seifenblase aufbläst.

In der Tat steigt die Sortierzeit bei diesem Verfahren bei großen Datenmengen sprunghaft an. Sind zum Beispiel n Elemente zu sortieren, müssen insgesamt n x n-1 Paarvergleiche durchgeführt werden. Dieses Verfahren hat auch seine Vorteile. Doch bevor wir darauf kommen, wollen wir zunächst einmal das Verfahren durchleuchten.

Erinnern Sie sich bitte noch einmal an das Sortieren von Karten. Wenn Sie die Karten in Ihrer Hand halten und diese sortieren wollen, stecken Sie jeweils die größte nach links. Dadurch wandern natürlich die kleineren Zahlen automatisch nach rechts. Allerdings haben Sie beim Kartensortieren in Ihrer Hand die Übersicht, wo die größte Karte gerade sitzt und stecken diese an den linken Rand. Dies kann so im Computer natürlich nicht durchgeführt werden, da der Computer diesen Überblick zunächst einmal nicht hat. Sie hätten diesen Überblick natürlich auch nicht, wenn Sie ein gesamtes Kartenspiel sortieren wollten. Also müssen wir in unserem Programm systematisch vorgehen.

Wir fangen zunächst an und vergleichen die linke Karte mit der daneben stehenden, schauen, welche größer ist und stecken die größere von beiden nach links. Anschließend vergleichen wir dann die zweite mit der dritten Karte usw., bis alle Karten durch

sind. Jetzt ist unser Kartenspiel aber noch lange nicht sortiert. Es ist immer noch möglich, daß das größte Element in der Mitte sitzt, da wir ja bisher immer nur zwei Karten verglichen und gegeneinander ausgetauscht haben. Allerdings ist das größte Element um einen Platz weiter nach links gerutscht. Dieses Verfahren muß also mehrfach wiederholt werden, und zwar genau einmal weniger, als Daten insgesamt zur Verfügung stehen. Bei einem Kartenspiel, das 32 Karten beinhaltet, muß dieses Verfahren insgesamt 31mal durchgeführt werden. Dann haben wir die Garantie, daß unsere Daten in der richtigen Reihenfolge stehen, denn spätestens beim 31. Mal ist das größte Element am ersten Platz und alle anderen kleineren sind rechts davon auch in der richtigen Reihenfolge sortiert.

Wir haben jetzt das eigentliche BUBBLE-SORT-Verfahren hinreichend beschrieben. Es hat noch einen Fehler, der aber sehr leicht behebbar ist. Es ist durchaus möglich, daß beim Sortieren unseres Kartenspiels weniger als die angegebenen 31 Durchgänge gebraucht werden. Dies ist sogar sehr wahrscheinlich, da nur 31 Durchgänge gebraucht werden, wenn die größte Karte am letzten Platz liegt. Da dies aber meistens nicht der Fall ist, genügen auch weniger Durchgänge.

Wenn wir uns jetzt überlegen, was in den letzten Durchgängen, in denen die Liste bereits vollständig sortiert ist, noch passieren würde, stellen wir fest, es wird nichts passieren, da bei jedem Paarvergleich immer nur die richtige Reihenfolge festgestellt wird und kein Austausch mehr stattfindet. Diese Erkenntnis können wir uns zunutze machen, indem wir eine Variable Z mit in unser Programm einbauen, die uns angibt, ob ein Austausch stattgefunden hat oder nicht.Zunächst setzen wir die Variable auf Null. Wenn dann whrend des Programms ein Austausch stattfindet, wird die Variable auf 1 gesetzt. Wenn jetzt ein neuer Durchlauf begonnen werden soll, können wir zunächst feststellen, ob im vorangegangenen Durchlauf überhaupt ein Austausch stattgefunden hat. Wenn dies nicht der Fall ist, sind wir schon fertig. Ist dies aber der Fall, können wir die Variable wieder auf Null setzen und das ganze Verfahren weiterlaufen lassen. Dies würde auch jeder vernünftige Kartenspieler so machen, der sein Kartenspiel sortieren muß. Er würde nicht mitzählen, ob er auch wirklich 31 Sortierdurchgänge durchgeführt hat sondern würde feststellen, ob die Karten sortiert sind.

Mit dieser kleinen Abwandlung des BUBBLE-SORT-Verfahrens ist dies natürlich auch im Computer möglich. Doch zunächst folgt jetzt erst einmal das Listing. Es wurde so geschrieben, daß es als Unterprogramm zu unserem Adressenprogramm verwendet werden kann, das heißt es beginnt mit der Zeile 5000, benutzt die gleichen Variablen wie unser Adressenprogramm und endet mit dem RETURN-Befehl.

```
5000 REM ========================
5010 REM SORTIEREN MIT BUBBLE-SORT
5020 REM ========================
5030 CLS#1 : CLS#2 : CLS#3
5040 LOCATE#1,15,3:PRINT#1,"SORTIEREN"
5050 LOCATE#2,13,8:PRINT#2,"BITTE WARTEN !"
5060 Z=0
5070 FOR I=1 TO ANZAHL-1
5080 IF D$(I,1)<=D$(I+1,1) THEN NEXT I
5090 FOR J=1 TO 6
5100 T$(J)=D$(I,J)
5110 NEXT J
5120 FOR J=1 TO 6
5130 D$(I,J)=D$(I+1,J)
5140 NEXT J
5150 FOR J=1 TO 6
5160 D$(I+1,J)=T$(J)
5170 NEXT J
5180 Z=1
5190 NEXT I
5200 IF Z=1 THEN 5060
5210 RETURN
```

Da wir in unserem Adressenprogramm nicht nur ein Datenfeld pro Datensatz haben, sondern 6 Datenfelder pro Datensatz, genügt es bei dem Austausch natürlich nicht, wie zum Beispiel bei dem Kartenspiel, daß nur eine Karte gegen die andere ausgetauscht wird. Wir müssen hier jeweils 6 Karten gegen 6 andere Karten austauschen, damit unser Datensatz so erhalten bleibt, wie er eingegeben wurde. Dies ist einfach mit einer FOR-NEXT-Schleife zu erreichen. Soll ein Austausch stattfinden, wird zunächst einmal unsere größere Karte auf dem Tisch abgelegt. Im Computer bedeutet das, wir speichern sie in einem anderen Datensatz zwischen, um sie nachher wieder einsortieren zu können. Den Zwischenspeicher

nennen wir in Anlehnung an den Tisch beim Kartenspiel T$. Da wir insgesamt 6 Zwischenspeicher brauchen, für jedes Datenfeld unseres Datensatzes einen, muß T$ von 1 bis 6 indiziert werden. Ist die Zwischenspeicherung erfolgt, können wir unseren zweiten Datensatz an die Stelle des ersten setzen und dann unseren Zwischenspeicher dort einsortieren, so der zweite Datensatz ist. Wir brauchen insgesamt 3 FOR-NEXT-Schleifen mit jeweils 6 Durchgängen. In unserem Programm nehmen diese 3 Schleifen den meisten Platz in Anspruch. Das heißt natürlich auch, daß unser eigentliches Sortierprogramm sehr kurz ist. Würde jeweils nur ein Datenfeld sortiert werden müssen und würden wir auf einen unserem Programm entsprechenden Bildaufbau verzichten, bestünde das Programm nur aus 7 Befehlen.

Unsere Anforderung an einen geringen Speicherbedarf des Programms wäre somit erfüllt. Ebenso wäre unsere Anforderung erfüllt, daß bereits vorsortierte Daten schneller sortiert werden sollen als Daten in beliebiger Reihenfolge. Dies wurde durch Einfügen der Variablen Z erreicht. Allerdings steigt mit der Anzahl der Datensätze die Sortierzeit quadratisch an. Bei kleinen Datenmengen ist dieses Verfahren sicherlich ein effizientes und auch verhältnismäßig schnelles Verfahren. Dies kann bei großen Datenmengen nicht mehr behauptet werden. Dafür sollten wir uns andere Sortierverfahren überlegen.

Sortieren durch Austauschen

Zur Erklärung dieses Verfahrens bleiben wir zunächst bei unserem Beispiel des Kartenspiels. Hierbei wird zunächst die erste Karte mit allen nachfolgenden Karten verglichen und, wenn eine größere Karte gefunden wird, werden beide Karten ausgetauscht und dann das Verfahren fortgesetzt. So kann es vorkommen, daß eine Karte mehrfach ausgetauscht wird, also während des Sortierverfahrens mal vorne, mal in der Mitte oder auch mal hinten in der Liste steht.

Der Vorteil des Verfahrens ist, daß nur halb so viele Vergleichsoperationen wie beim BUBBLE-SORT-Verfahren durchgeführt werden müssen. Allerdings ist es mit diesem Verfahren nicht möglich, daß durch eine zufällige vorausgegangene Sortierung das Verfahren früher abgebrochen werden kann. Auch eine vorsortierte Liste würde nicht dazu führen, daß die Anzahl der Paarvergleiche zurückgehen würde. Wir haben also, egal wie unsere Datei beschaffen ist,

bei gleicher Anzahl von Datensätzen immer die gleiche Anzahl von Vergleichen durchzuführen, so daß selbst bei einer vollständig sortierten Liste das Verfahren genau so lange dauern würde wie bei einer vollkommen unsortierten Liste.

Dadurch ist dieses Verfahren nicht gerade vorteilhaft. Allerdings ist zu beachten, daß die Sortierzeit unter Umständen wesentlich kürzer ist als beim BUBBLE-SORT-Verfahren. Das "unter Umständen" bezieht sich darauf, daß beim BUBBLE-SORT-Verfahren nur die maximal benötigte Zahl von Paarvergleichen ausgerechnet werden kann, während bei diesem Verfahren, das immer die Hälfte der beim BUBBLE-SORT-Verfahren maximalen Anzahl der Paarvergleiche braucht, eine Zahl ausgerechnet werden kann, die immer zutrifft.

So ist es schon eine Überlegung wert, ob dieses Verfahren dem BUBBLE-SORT-Verfahren vorgezogen werden soll. Eine Wertung soll an dieser Stelle nicht erfolgen und Ihnen überlassen bleiben. Das nachfolgende Listing wurde wiederum so erstellt, daß es als Unterprogramm in unserem Adressenprogramm verwendet werden kann.

```
5000 REM ==========================
5010 REM SORTIEREN DURCH AUSTAUSCH
5020 REM ==========================
5030 CLS#1 : CLS#2 : CLS#3
5040 LOCATE#1,15,3:PRINT#1,"SORTIEREN"
5050 LOCATE#2,13,8:PRINT#2,"BITTE WARTEN !"
5060 FOR I=1 TO ANZAHL-1
5070 L=I+1
5080 FOR K=I TO N
5090 IF D$(I,1)>D$(L,1) THEN NEXT K
5100 L=K
5110 FOR J=1 TO 6
5120 T$(J)=D$(I,J)
5130 NEXT  J
5140 FOR J=1 TO 6
5150 D$(L,J)=D$(I,J)
5160 NEXT  J
5170 FOR J=1 TO 6
5180 D$(I,J)=T$(J)
5190 NEXT  J
5200 NEXT I
5210 RETURN
```

QUICK-SORT

Dieses Verfahren trägt seinen Namen zu Recht, denn es ist das schnellste Sortierverfahren von Daten, wenn große Datenmengen vorliegen. Dies gilt natürlich nur durchschnittlich, da wir, wie bereits gesehen, bei vorsortierten Dateien mit unserem BUBBLE-SORT-Verfahren recht schnell Ergebnisse erreichen. In diesem Falle wäre dann BUBBLE-SORT QUICK-SORT überlegen, im allgemeinen wird dies umgekehrt der Fall sein. Die Sortierzeiten beim QUICK-SORT steigen nicht wie beim BUBBLE-SORT quadratisch, sondern fast linear an, wodurch sich das Verfahren für große Datenmengen besser eignet.

Es basiert auf folgender Idee: Beim Sortieren von Karten würden zunächst die Karten am linken und am rechten Rand mit einer Karte in der Mitte verglichen. Vom linken Rand ausgehend, wird eine Karte nach der anderen bis zur Mitte hin mit der Karte in der Mitte verglichen. Wird eine gefunden, die kleiner ist als die Karte in der Mitte, wird sie zwischengespeichert. Zwischenspeichern heißt, daß Sie sich diese Karte merken müssen. Dann wird der rechte Teil mit der Karte in der Mitte verglichen und zwar von rechts nach links. Wenn Sie dort eine Karte finden, die größer als die Karte in der Mitte ist, werden beide vertauscht. Dann geht das Verfahren wieder auf der linken Seite weiter. Wenn Sie auch dort wieder eine kleinere Karte finden, merken Sie sich auch die und setzen das Verfahren dann auf der rechten Seite wieder fort.

Dieses Verfahren wird so lange durchgeführt, bis sich der rechte und der linke Rand im Vergleich treffen. Dies war der erste Durchgang. Die Karten sind noch nicht sortiert. Allerdings sind links von dem gedachten Treffpunkt alle Karten, die kleiner sind und rechts davon alle, die größer sind. Nun kann jede dieser Teillisten wieder nach der gleichen Methode sortiert werden, so daß sich immer kleinere Teillisten bilden, bis dann schließlich nur noch zwei Elemente miteinander verglichen zu werden brauchen.

Dies hört sich kompliziert an, weil Sie sich eine ganze Menge merken müssen. Zugegeben, beim Kartensortieren würde es eine schwierige Aufgabe sein, sich alle die Karten zu merken. Für einen Computer ist das kein Problem. Hier können wir die Werte einfach zwischenspeichern. Wir nehmen dazu zwei Variablen, die wir K nennen. K wählen wir in Anlehnung an den Begriff Keller, da wir uns vorstellen, daß die Daten im Keller abgelegt werden.

```
5000 REM =========
5010 REM QUICKSORT
5020 REM =========
5030 CLS#1: CLS#2: CLS#3
5040 LOCATE#1,15,3:PRINT#1,"SORTIEREN"
5050 LOCATE#2,13,8:PRINT#8,"BITTE WARTEN !"
5060 N1=1:N2=ANZAHL:I=1
5070 L1=N1:L2=N2
5080 IF D$(L1,1) <=D$(L2,1) THEN GOTO 5300
5090 FOR J=1 TO 6
5100 T$(J)=D$(L1,J)
5110 NEXT
5120 FOR J=1 TO 6
5130 D$(L1,J)=D$(L2,J)
5140 NEXT
5150 FOR J=1 TO 6
5160 D$(L2,J)=T$(J)
5170 NEXT
5180 L1=L1+1
5190 IF L1=L2 THEN GOTO 5320
5200 IF D$(L1,1) <=D$(L2,1) THEN GOTO 5180
5210 FOR J=1 TO 6
5220 T$(J)=D$(L1,J)
5230 NEXT
5240 FOR J=1 TO 6
5250 D$(L1,J)=D$(L2,J)
5260 NEXT
5270 FOR J=1 TO 6
5280 D$(L2,D)=T$(J)
5290 NEXT
5300 L2=L2-1
5310 IF L2<>L1 THEN GOTO 5080
5320 L2=L2+1:IF L2 >=N2 THEN GOTO 5350
5330 K(I,1)=L2:K(I,2)=N2
5340 I=I+1
5350 L1=L1-1
5360 IF N1 >=L1 THEN GOTO 5390
5370 N2=L1
5380 GOTO 5070
5390 I=I-1
5400 N1=K(I,1):N2=K(I,2)
5410 IF I >0 THEN GOTO 5070
5420 RETURN
```

Insgesamt ist dieses Verfahren einfacher, als es im ersten Moment aussieht. Durch die Aufspaltung in Teillisten wird immer wieder der gleiche Ablauf auf immer kleinere Datenmengen angewandt. Vielleicht probieren Sie das Verfahren mit einem kleinen Kartenspiel aus. Sie werden sehen, nach zwei Durchgängen sind die einzelnen Häufchen so klein geworden, daß sich das ganze leicht überblicken läßt.

Sie sehen, dieses Listing ist bedeutend länger als die vorhergehenden Routinen. Es belegt also etwas mehr Speicherplatz. Da bei kleinen Datenmengen der Geschwindigkeitsvorsprung von QUICK-SORT nicht bedeutend ist, bzw. BUBBLE-SORT da sogar noch überlegen ist, hat auch diese Routine ihre Nachteile. Insgesamt haben wir jetzt drei Verfahren vorgestellt (es gibt noch eine Vielzahl mehr), die einen groben Überblick über die Möglichkeiten von Sortier-Algorithmen geben sollen. Welches der vorgestellten Verfahren für Ihre Zwecke in Frage kommt, sollten Sie selbst entscheiden.

2.2.3.9 Das fertige Programm

An dieser Stelle werden wir das gesamte Programm zusammensetzen. Wenn wir bisher die einzelnen Teile eingegeben und diese Teile jeweils auf Kassette gespeichert sind, ist es möglich, die einzelnen Teile jetzt mit dem MERGE-Befehl zusammenzusetzen. Spulen Sie dazu das Band an den Anfang zurück, und geben Sie MERGE mit zwei folgenden Anführungszeichen ein, damit das nächste Programm nachgeladen wird. So können dann alle Programmteile zusammengesetzt werden.

Sollten Sie die einzelnen Programmteile noch nicht eingetippt haben und möchten das ganze Programm mit einmal eingeben, haben Sie jetzt dazu Gelegenheit. Nachfolgend wird das gesamte Listing unseres Adressenprogramms in lauffähiger Version abgedruckt. Von den drei Sortierroutinen haben wir die QUICK-SORT-Routine ausgewählt, was allerdings nicht heißen soll, daß wir für jede Problemstellung diese Routine bevorzugen würden.

Bedenken Sie bitte bei unserem Programm, daß es vom Ansatz her ein sehr leistungsfähiges Programm ist und in seinen Routinen schon einen gewissen Service bietet. Dennoch könnte man mit einem wesentlich längeren Programm noch einige Funktionen mehr erfüllen. Da diese Funktionen dann sehr speziell sind, beschäftigen wir uns damit nicht weiter. Das Programm hat einen modularen Aufbau, deshalb sind Sie jederzeit in der Lage, es um eigene Routinen zu erweitern. Es ist natürlich auch möglich, Routinen des Programmes durch andere, selbstgeschriebene Routinen zu ersetzen. Für die meisten Anwendungen dürfte unser Programm ausreichend sein. Sie können sich damit eine komplette Adressenverwaltung aufbauen. Sie sollten keine Scheu davor haben, das Programm zu erweitern.

```
1 REM =============
2 REM ADRESSENDATEI
3 REM =============
10 REM ========
20 REM VORSPANN
30 REM ========
40 CLS
50 INK 0,1 : INK 1,24 : INK 2,3 : BORDER 1
60 WINDOW#1,1,40,1,5 : WINDOW#2,1,40,6,20: WINDOW#3,1,40,21,25
70 PAPER #1,1 : PEN#1,2 : PAPER#3,1: PEN#3,2
80 DIM D$(200,6)
100 REM =====
110 REM MENUE
120 REM =====
130 CLS#1: CLS#2 : CLS#3
140 LOCATE#1,16,3: PRINT#1,"M E N U E"
150 LOCATE#2,13,3:PRINT#2,"1  DATEN EINLESEN"
160 LOCATE#2,13,5:PRINT#2,"2  DATEN SPEICHERN"
170 LOCATE#2,13,7:PRINT#2,"3  EINGEBEN"
180 LOCATE#2,13,9: PRINT#2,"4  SUCHEN"
190 LOCATE#2,13,11:PRINT#2,"5  SORTIEREN"
200 LOCATE#2,13,13:PRINT#2,"6  DRUCKEN"
210 CLS#3:LOCATE#3,3,3: INPUT#3,"WELCHE NUMMER";A$
220 A=VAL(A$): IF A<1 OR A>6 THEN 210
230 ON A GOSUB 1000,2000,3000,4000,5000,6000
240 GOTO 130
1000 REM =====
1010 REM LADEN
1020 REM =====
1030 CLS#1 : CLS#2 : CLS#3
1040 LOCATE 3,12:LOCATE #1,15,3: PRINT#1,"L A D E N"
1050 I=ANZAHL+1
1060 OPENIN"ADRESSENDATEI"
1070 WHILE EOF=0
1080 FOR J=1 TO 6
1090 INPUT#9,D$(I,J)
1100 NEXT J: I=I+1
1110 WEND
1120 CLOSEIN
1130 ANZAHL=I-1:RETURN
2000 REM =========
```

```
2010 REM SPEICHERN
2020 REM =========
2030 CLS#1 : CLS#2 : CLS#3
2040 LOCATE 3,12:LOCATE #1,15,3: PRINT#1,"SPEICHERN"
2050 OPENOUT"ADRESSENDATEI"
2060 FOR I=1 TO ANZAHL
2070 FOR J=1 TO 6
2080 PRINT#9,D$(I,J)
2090 NEXT J : NEXT I
2100 CLOSEOUT
2110 RETURN
3000 REM =============
3010 REM E I N G A B E
3020 REM =============
3030 a$="E I N G A B E ":I=ANZAHL+1
3040 CLS #1:CLS #2:CLS #3:LOCATE#1,INT((40-LEN(A$))/2),3:PRINT#1,A$
3050 LOCATE#2,3,3:PRINT#2,"Name:";STRING$(32,95)
3060 LOCATE#2,3,5:PRINT#2,"Vorname:";STRING$(29,95)
3070 LOCATE#2,3,7:PRINT#2,"Strasse:";STRING$(29,95)
3080 LOCATE#2,3,9:PRINT#2,"Ort:";STRING$(24,95)
3090 LOCATE#2,3,11:PRINT#2,"PLZ:";STRING$(8,95):LOCATE#2,15,11:PRINT#2
,"Tel.:";STRING$(15,95)
3100 LOCATE#2,3,13:PRINT#2,"Dateinummer:";i
3110 LOCATE#2, 8,3: INPUT#2,"",D$(I,1)
3120 LOCATE#2, 11,5: INPUT#2,"",D$(I,2)
3130 LOCATE#2, 11,7: INPUT#2,"",D$(I,3)
3140 LOCATE#2, 7,9: INPUT#2,"",D$(I,5)
3150 LOCATE#2, 7,11: INPUT#2,"",D$(I,4)
3160 LOCATE#2, 20,11: INPUT#2,"",D$(I,6)
3170 LOCATE #3,2,3:INPUT#3,"RICHTIG (J/N)",J$:IF J$="N" OR J$="n" THEN
 CLS#2: GOTO 3050
3180 CLS#3:LOCATE #3,2,3:INPUT#3,"Weitere Eingabe ? (J/N)",J$:IF J$="N
" OR J$="n" THEN ANZAHL=I:RETURN
3190 IF j$ ="j" OR j$="J" THEN CLS#2:CLS#3: I=I+1: GOTO 3050
3200 GOTO 3180
4000 REM ======
4010 REM SUCHEN
4020 REM ======
4030 CLS#1 : CLS#2 : CLS#3
4040 LOCATE #1,13,3: PRINT#1,"S U C H E N"
4050 LOCATE#2,3,7:PRINT#2,"WONACH SOLL GESUCHT WERDEN?"
```

```
4060 LOCATE#2,3,9: INPUT#2,"",A$
4070 FOR I=1 TO ANZAHL
4080 FOR J=1 TO 6
4090 IF INSTR(D$(I,J),A$)<>0 THEN GOSUB 4140
4100 IF INSTR(D$(I,J),A$)<>0 THEN CLS#3:LOCATE#3,3,3: INPUT#3,"WEITERS
UCHEN (J/N)";B$: IF UPPER$(B$)<>"J"THEN 4120
4110 NEXT J : NEXT I
4120 CLS#3:LOCATE#3,3,3: INPUT#3,"NOCHMAL SUCHEN (J/N)";B$: IF UPPER$(
B$)<>"J" THEN RETURN
4130 GOTO 4030
4140 CLS#2: LOCATE#2,3,3: PRINT#2,"NAME: ";D$(I,1)
4150 LOCATE#2,3,5: PRINT#2,"VORNAME: ";D$(I,2)
4160 LOCATE#2,3,7: PRINT#2,"STRASSE: ";D$(I,3)
4170 LOCATE#2,3,9: PRINT#2,"ORT: ";D$(I,5)
4180 LOCATE#2,3,11: PRINT#2,"PLZ: ";D$(I,4);"   TEL.: ";D$(I,6)
4190 LOCATE#2,3,13: PRINT#2,"DATEINUMMER: ";I
4200 RETURN
5000 REM =========
5010 REM QUICKSORT
5020 REM =========
5030 CLS#1: CLS#2: CLS#3
5040 LOCATE#1,15,3:PRINT#1,"SORTIEREN"
5050 LOCATE#2,13,8:PRINT#2,"BITTE WARTEN !"
5060 N1=1:N2=ANZAHL:I=1
5070 L1=N1:L2=N2
5080 IF D$(L1,1) <=D$(L2,1) THEN GOTO 5300
5090 FOR J=1 TO 6
5100 T$(J)=D$(L1,J)
5110 NEXT
5120 FOR J=1 TO 6
5130 D$(L1,J)=D$(L2,J)
5140 NEXT
5150 FOR J=1 TO 6
5160 D$(L2,J)=T$(J)
5170 NEXT
5180 L1=L1+1
5190 IF L1=L2 THEN GOTO 5320
5200 IF D$(L1,1) <=D$(L2,1) THEN GOTO 5180
5210 FOR J=1 TO 6
5220 T$(J)=D$(L1,J)
5230 NEXT
```

```
5240 FOR J=1 TO 6
5250 D$(L1,J)=D$(L2,J)
5260 NEXT
5270 FOR J=1 TO 6
5280 D$(L2,J)=T$(J)
5290 NEXT
5300 L2=L2-1
5310 IF L2 <>L1 THEN GOTO 5080
5320 L2=L2+1:IF L2 >=N2 THEN GOTO 5350
5330 K(I,1)=L2:K(I,2)=N2
5340 I=I+1
5350 L1=L1-1
5360 IF N1 >=L1 THEN GOTO 5390
5370 N2=L1
5380 GOTO 5070
5390 I=I-1
5400 N1=K(I,1):N2=K(I,2)
5410 IF I >0 THEN GOTO 5070
5420 RETURN
6000 REM =======
6010 REM DRUCKEN
6020 REM =======
6030 CLS#1 : CLS#2 : CLS#3
6040 LOCATE#1,15,3:PRINT#1,"D R U C K E N"
6050 LOCATE #3,2,3: INPUT#3,"IST EIN DRUCKER ANGESCHLOSSEN (J/N)";A$

6060 IF LOWER$(A$)<>"j" THEN RETURN
6070 PRINT#8,"A D R E S S E N D A T E I"
6080 PRINT#8,"========================="
6090 PRINT#8
6100 FOR I=1 TO ANZAHL
6110 PRINT#8,D$(I,1)", "D$(I,2)",  TEL.:"D$(I,6)
6120 PRINT#8,D$(I,3)", "D$(I,4)" "D$(I,5)
6130 NEXT
6140 RETURN
```

2.2.4 Numerische Dateien

Nachdem wir uns im letzten Kapitel ausgiebig mit den Möglichkeiten von STRINGS beschäftigt haben, wollen wir uns nun dem numerischen Bereich zuwenden. Der CPC 464 verfügt im wesentlichen über zwei Typen von numerischen Variablen, INTEGER- und REAL-Variablen. Im Unterschied zu STRINGS können in diesen nicht beliebige Zeichen, sondern nur Zahlen gespeichert werden.

Der Unterschied zwischen INTEGER- und REAL-Variablen liegt in der Art der speicherbaren Zahlen. Bei INTEGER-Variablen sind nur Ganzzahlen speicherbar und auch der Bereich ist hier beschränkt. INTEGER-Variablen können nur Werte zwischen +32.768 und -32.768 annehmen. In einer REAL-Variablen können dagegen auch gebrochen rationale Zahlen, Dezimalzahlen etc. gespeichert werden. Auch ist der Bereich hier etwas weiter gefaßt, er reicht von 1E+38 bis -1E+38, womit fast jede Zahl darstellbar ist. Dies geht allerdings auf Kosten des Speicherplatzes und der Schnelligkeit.

REAL-Variablen benötigen erheblich mehr Speicherplatz, und ihre Bearbeitung kostet mehr Zeit. Speziell in Programmen, die mit größeren Datenmengen arbeiten und komplizierte, das heißt zeitintensive Sortier-Algorithmen anwenden, ist es deshalb sinnvoll zu prüfen, ob nicht gegebenenfalls die Verwendung von INTEGER-Variablen ausreicht. Die Definition des Variablentyps nehmen wir durch die Auswahl des Variablennamens vor. Für den Variablennamen ist eine Kombination aus Buchstaben und Zahlen zulässig. Fügen wir an diese Kombination das Dollar-Zeichen an, erhalten wir einen STRING. Ein Ausrufezeichen am Ende unseres Variablennamens definiert den Variablentyp als REAL-Variable. Geben wir keines dieser Typenkennzeichen an, faßt der CPC die Variable als REAL-Variable auf.

Es gibt noch eine andere, globalere Methode, um die Variablentypen festzulegen. Diese ist aus anderen Programmiersprachen, wie zum Beispiel PASCAL und FORTRAN bekannt. Mit dem Befehl DEFINT werden die nachfolgend aufgeführten Variablennamen als INTEGER festgelegt. DEFREAL bzw. DEFSTR leisten dasselbe fü den REAL- bzw. STRING-Bereich. So wird zum Beispiel mit DEFINT I-N festgelegt, daß die Variablen-Typen mit den Anfangsbuchstaben I bis N vom Computer als INTEGER aufzufassen sind, solange sie nicht anderweitig durch Benutzung der Variablentypen definiert wurden. Ein DEFREAL A definiert alle Variablen mit dem Anfangsbuchstaben

A als REAL. Geben wir danach ein AB% = 14 ein, haben wir damit dennoch den Wert der INTEGER-Variablen auf 14 gesetzt. Die Typkennzeichen stehen also in der Hierarchie über den DEF-Befehlen. Im Verlauf unseres Programms sind wir nicht an diese einmal vorgegebenen Typdefinitionen gebunden. Wir können nämlich INTEGER- und REAL-Variable mit den Kommandos

CINT und CREAL

untereinander konvertieren. CINT wandelt eine REAL-Variable zu einer INTEGER-Variablen um. Dabei wird der Wert der REAL-Variablen kaufmännisch gerundet. Geben wir

```
N=4.7:PRINT CINT (N)
```

ein, so erhalten wir das Ergebnis fünf.

Beim Arbeiten mit numerischen Variablen sind wir nicht nur auf einfache Variablen beschränkt. Wie schon im letzten Kapitel können wir Felder definieren, die eine, zwei, drei, usw. Dimensionen beeinhalten. Diese wären dann zum Beispiel als

A(I,J,K) oder B%(I)

zu schreiben. Wollen wir ein mehrdimensionales Feld eröffnen, oder mit einem eindimensionalen Feld arbeiten, dessen Index größer als elf wird, müssen wir dieses wie bei den STRINGS mit dem DIM-Befehl vorab dimensionieren. Auslöschen erfolgt auch hier mit dem ERASE-Kommando. Die Verwendung des DIM-Befehls kann auch bei kleineren eindimensionalen Feldern sinnvoll sein.

Will man zum Beispiel nur mit fünf Variablen arbeiten, die man mit A(1) bis A(5) bezeichnet, spart das Dimensionieren von A auf fünf mit

```
DIM A(5)
```

wiederum Speicherplatz. Das Abspeichern und Laden von numerischen Daten geht wie bei den STRING-Daten vor sich. Haben wir zum Beispiel ein ARRAY von der Größe 3 mal 4 vor uns, das wir mit

```
DIM A(3,4)
```

dimensioniert hatten (die Matrizen mit den Indices 0 sollen hier nicht verwendet werden), können wir es mit folgenden Zeilen abspeichern:

```
10 OPENOUT"PRG-NAME"
20 FOR I=1 TO 3
30 FOR J=1 TO 4
40 PRINT#9,A (I,J)
50 NEXT J,I
```

Zum Einlesen ersetzen wir wiederum das PRINT-Kommando durch den INPUT-Befehl, wobei wir vorher das entsprechende FILE (= engl. Name für Datei) mit OPENIN"PRG-NAME" zum Lesen öffnen sollten; eine Operation, die wir beim Schreiben umgekehrt ausgeführt hatten.

Schauen wir uns nun an, welche Sortierroutinen bei numerischen Variablen sinnvoll nutzbar sind. Beim Arbeiten mit STRINGS ist es darum gegangen, diese alphabetisch zu ordnen, beziehungsweise das Übereinstimmen von Teilstücken dieser STRINGS mittels der INSTR-Routine zu überprüfen. Beim Arbeiten mit numerischen Variablen ist die Fragestellung meist anders. Hier geht es darum, diese Zahlen der Größe nach zu ordnen, beziehungsweise ihre Minimal- und Maximalwerte festzustellen. Wie dies geht, wollen wir mit den nächsten beiden Programmen demonstrieren. Zunächst zur Berechnung von MINIMA und MAXIMA. Dazu stellt uns das CPC-BASIC die Befehle

MIN und MAX

zur Verfügung. Diese berechnen aus einer gegebenen Anzahl von Variablen den größten, beziehungsweise kleinsten Wert. Nun wäre es möglich, alle zu berücksichtigten Variablen in den MAX-Befehl hineinzuschreiben und somit auf einen Schlag das Maximum der Wertegruppe zu erhalten. Einfacher und weniger kompliziert geht es, wenn wir Schritt für Schritt in einer Schleife die Extremwerte berechnen. Dazu wenden wir einen kleinen Trick an. Wir setzen nämlich unsere Variable ma, die am Ende den Maximalwert enthalten soll, gleich dem Maximalwert von ihr selbst und den zu vergleichenden Feldvariablen z(I). Da i über alle Werte unseres Feldes läuft, ist mit dieser Methode sichergestellt, daß wir auch wirklich das Maximum unserer Werte in ma erhalten. Dieselbe Überlegung läßt sich analog auch zur Bestimmung des Minimums durchführen. Um eine beliebige Anzahl von Werten verarbeiten zu können, wurde diese Routine für ein variables Feld vorbereitet. Das

heißt, die Variable z wird erst im Programm entsprechend der eingegebenen Werte dimensioniert. Somit sind zwei oder auch zweihundert Werte je nach dem Willen des Benutzers bearbeitbar. Nachdem die einzelnen Variablen in Zeile sechzig bis achzig eingegeben wurden, erfolgt dann die Errechnung der Extremwerte in der schon beschriebenen Art. Im Anschluß daran werden diese ausgegeben.

```
10 REM =============
20 REM * Min - Max *
30 REM =============
40 CLS
50 mi=1E+38:ma=-1E+37
60 INPUT"Wie viele Werte";w
70 DIM z(w)
80 PRINT
90 FOR i= 1 TO w
100 PRINT"Bitte geben Sie den ";i;"ten Wert ein."
110 INPUT z(i):NEXT i
120 FOR i=1 TO w
130 ma=MAX(ma,z(i))
140 mi=MIN(mi,z(i))
150 NEXT i
160 PRINT
170 PRINT"Der Maximalwert betraegt:";ma
180 PRINT"Der Minimalwert betraegt:";mi
190 PRINT
```

Etwas schwieriger wird es, wenn wir unser Zahlenfeld nach der Größe ordnen wollen. In diesem Programm arbeiten wir mit zwei Feldern. Das Eingabefeld enthält die zuzuordnenden Daten, das zweite Feld, wir haben hier y gewählt, enthält dann die geordneten Daten in absteigender Reihenfolge. Die Eingabeprozedur ist dieselbe wie in dem oben bereits behandelten MIN/MAX-Programm. Der Sortier-Algorithmus ist jedoch wesentlich anders. Das Programm beginnt mit J=1 und sucht den Maximalwert des Zahlenfelds z. Darauf werden für alle I die Elemente z(I) daraufhin untersucht, ob sie größer als y(1) sind und gegebenenfalls y(1) auf diesen Wert heraufgesetzt. Dieses ist derselbe Gedankengang wie bei unserem MAX-Programm. Im Unterschied zu diesem wird die Zahl des Elementes, das gespeichert wurde, in der Variablen y festgehalten. Damit ist es nach Durchlaufen der I-Schleife möglich, festzustellen, welches Element das größte war. Dieses wird nun

auf -1E+37 herabgesetzt, womit es beim nächsten Durchlauf als größtes Element ausscheidet. Für J=2 wird somit das zweitgrößte Element bestimmt usw. Am Ende des Programms werden die Zahlen dann noch einmal in fortlaufender Folge ausgegeben.

```
10 REM ===========
20 REM * Numsort *
30 REM ===========
40 CLS
50 INPUT "Wieviele Werte";w
60 DIM z(w)
70 FOR i= 1 TO w
80 PRINT"Bitte geben Sie den ";i;"ten Wert ein."
90 INPUT z(i):NEXT i
100 DIM y(w)
110 FOR i=1 TO w
120 y(i)=-1E+37:NEXT i
130 REM =============
140 REM * Sortieren *
150 REM =============
160 FOR j=1 TO w
170 FOR i= 1 TO w
180 IF z(i)>y(j) THEN y(j)=z(i):y=i
190 NEXT i
200 z(y)=- 1E+37:NEXT j
210 CLS
220 PRINT:PRINT"Zahlen in fortlaufender Reihenfolge:":PRINT
230 FOR i= 1 TO w:PRINT i,y(i):NEXT i
```

2.3 Textverarbeitung mit dem CPC 464

Neben der Dateiverwaltung und der Verarbeitung von Zahlen, die wir beide in den vorhergehenden Abschnitten beschrieben haben, stellt die Textverarbeitung eines der wichtigsten Aufgabenfelder für Mikrocomputer dar, und gleichzeitig ist sie eine der interessantesten und vielfältigsten Problemstellungen, die sich einem Programmierer bieten.

Es gibt eine sehr große Zahl verschiedener Anforderungen, die ein Textprogramm erfüllen soll. Diese reichen vom einfachen Briefeschreibprogramm bis zur umfangreichen Textverwaltung, mit der es möglich ist, selbst ganze Bücher zu schreiben. Für jeden Anwendungszweck gibt es ein mehr oder weniger umfangreiches Textprogramm und so groß wie die Zahl der Anforderungen an ein solches Programm, ist auch die Anzahl der Lösungsmöglichkeiten. Deshalb ließe sich über die Textverarbeitung nicht nur ein Buch schreiben, sondern eine ganze Bibliothek füllen. Wir müssen uns hier deshalb auf einige wesentliche Punkte beschränken. Insbesondere wollen wir es uns zur Aufgabe stellen, ein kleines Textverarbeitungsprogramm selbst zu entwickeln.

2.3.1 Aufbau des Programms

Zunächst müssen wir wieder einmal einige Vorüberlegungen zu unserem Programm anstellen. Wir müssen uns als erstes fragen, was unser Programm alles können soll.

An erster Stelle steht natürlich, und das muß eigentlich gar nicht mehr erwähnt werden, die Eingabe von Text. Fast noch wichtiger ist die Möglichkeit, den eingetippten Text anschließend korrigieren und verändern zu können. Wir wollen natürlich auch unseren Kassettenrecorder benutzen und die geschriebenen Texte abspeichern und wieder einlesen. Nun fehlt nur noch eine Routine zum Ausdrucken und unser Textprogramm ist fertig.

Damit das Programm anwenderfreundlich gestaltet ist, sollen die einzelnen Routinen wiederum durch ein Menü aufgerufen werden. Der wichtigste Zweck, den unser Programm erfüllen soll, ist das Verfassen von Briefen und anderen Texten. Ein normaler Drucker verarbeitet in einer Zeile bis zu achzig Zeichen, da aber dann eine DIN A4 Seite vollständig von links bis rechts bedruckt ist, wollen wir hier eine etwas schmalere Zeile wählen. Das beste Druckbild erhalten wir, wenn wir in einer Zeile sechzig Zeichen unterbringen.

Die meisten Textverarbeitungsprogramme sind zeilenorientiert, das heißt sie schauen sich nur an, wie viele Zeilen in einem Text vorkommen. Da der Mensch sich Text meistens nicht in Zeilen, sondern eher in Seiten vorstellen kann, ist es besser, eine Textver-

arbeitung aufzubauen, die seitenorientiert ist. Auch dies hat seine Nachteile, wenn man später einmal ein solches Programm um sogenannte Blockverschieberoutinen erweitert, die es ermöglichen, daß Absätze getauscht werden, oder daß Zeilen eingefügt werden können.

Weil es für den Benutzer angenehmer ist, wollen wir hier gerne seitenorientiert arbeiten, uns aber gleichzeitig die Option einer Zeilenorientierung offenhalten. Deshalb versuchen wir eine Kombination aus beiden Möglichkeiten, indem wir unseren Text als zweidimensionale Datei abspeichern. Dabei soll die erste Dimension die Seite darstellen und die zweite die Zeile. Da es sich bei dieser Art Speicherung um einen Zeilen-STRING handelt, wollen wir diesen STRING Z$ nennen.

Damit sind wir eigentlich schon mitten im Vorspann zu unserem Programm, in dem die Farbregister gesetzt und die Variablen definiert werden. Dies müßte dann so aussehen:

```
10 REM ============
20 REM VORSPANNTEXT
30 REM ============
40 MODE 1
50 INK 0,1: INK 1,24: INK 2,3 : BORDER 1
60 PAPER #2,1 : PEN #2,2 : PAPER #3,1 : PEN #3,2 : PAPER #4,1 : PEN #4
,2 : PAPER #5,1 : PEN #5,2
70 DIM Z$(60,60): AN=1 : ZEI=0 : ZL=60 : KL=53 : SL=60
```

Die wichtigste Zeile dieses Programmteils ist sicherlich Zeile 70. Hier wird zunächst einmal Z$ dimensioniert. Wir gehen davon aus, daß auf jeder Seite eines Textes sechzig Zeilen untergebracht werden. Dies entspricht einer normalen DIN A4 Seite bei einzeiliger Beschriftung. Wir glauben darüberhinaus, daß wir mit circa sechzig Seiten immer auskommen werden; deshalb dimensionieren wir unser Z$ auf sechzig in der ersten Dimension und sechzig in der zweiten Dimension.

Selbstverständlich brauchen wir noch eine Anzahl anderer Variablen, die hier definiert werden. AN steht für die Anzahl der Seiten, die im Speicher enthalten sind. Sie wird zunächst einmal auf Eins gesetzt, damit wir überhaupt mit der ersten Seite anfangen. Mit ZL wird die Zeilenlänge auf sechzig definiert. Wir haben oben

schon darüber gesprochen, daß dies einer normalen Beschriftung einer DIN A4 Seite entspricht.

Selbstverständlich ist es möglich, an dieser Stelle auch eine andere Zahl für die Zeilenlänge einzusetzen. In dem Programm wird immer mit der Variablen ZL gearbeitet, so daß diese Variable ohne weiteres auf eine andere Zeilenlänge eingestellt werden kann. Danach arbeitet das Programm immer noch ordnungsgemäß.

Mit SL soll die Seitenlänge bezeichnet werden. Wer lieber 1-1/2-zeiligen Zeilenabstand hat statt 1-zeiligen und auch seinen Drukker daraufhin einstellen kann, kann diese Variable in die Zahl 40 ändern, was einer 1-1/2-zeiligen Beschriftung einer kompletten DIN A4 Seite entsprechen würde.

Wir wollen bei unserer Eingabe noch einige kleine Besonderheiten erreichen. So muß es zum Beispiel möglich sein, an jeder beliebigen Stelle des Textes die Eingabe abzubrechen und dann auch wieder dort fortzusetzen. Für diesen Zweck müssen wir uns merken, in welcher Zeile einer Seite wir die Eingabe abgebrochen haben; dazu wird die Variable ZEI benutzt. In ihr wird dann später bei Beendigung der Eingabe die aktuelle Zeilennummer abgelegt. Zu Beginn des Programms ist sie selbstverständlich auf Null gesetzt.

Von der Schreibmaschine her sind wir es gewohnt, daß bei Erreichen einer bestimmten Stelle ein Klingelzeichen ertönt, das uns anzeigt, daß die Zeile jetzt bald vollgeschrieben ist. Dies können wir selbstverständlich auch mit dem Computer erreichen. Dazu müssen wir dem Rechner lediglich mitteilen, wann er sich akustisch bemerkbar machen soll. Diese Stelle darf nicht sehr weit vom Zeilenende entfernt sein. Wir wählen dafür die Stelle 53 und ordnen der Variablen KL (für Klingel) die Zahl 53 zu.

Später bei der Texteingabe werden wir diese Variablen alle noch einmal benötigen. Doch zunächst wollen wir uns einmal den Menüaufbau des Programms anschauen.

Die folgenden Zeilen stellen das eigentliche Hauptprogramm für unsere Textverarbeitung dar. Von ihm aus werden alle Möglichkeiten, die unser Programm bieten soll, aufgerufen.

```
100 REM ==========
110 REM MENUETEXT
120 REM =========
130 CLS : MODE 1
140 WINDOW#2,1,40,1,5 : WINDOW#1,1,40,6,20: WINDOW#3,1,40,21,25
150 CLS#1: CLS#2 : CLS#3
160 LOCATE#2,16,3: PRINT#2,"M E N U E"
170 LOCATE#1,13,3:PRINT#1,"1  TEXT EINGEBEN"
180 LOCATE#1,13,5:PRINT#1,"2  TEXT SPEICHERN"
190 LOCATE#1,13,7:PRINT#1,"3  TEXT LADEN"
200 LOCATE#1,13,9:PRINT#1,"4  KORRIGIEREN"
210 LOCATE#1,13,11:PRINT#1,"5  DRUCKEN"
220 CLS#3:LOCATE#3,3,3: INPUT#3,"WELCHE NUMMER";A$
230 A=VAL(A$): IF A<1 OR A>5 THEN 220
240 ON A GOSUB 1000,2000,3000,4000,5000
250 GOTO 150
```

Für unseren Bildschirm wählen wir wieder die bekannte DREI-WINDOW-Technik, die in diesem Buch an mehreren Stellen besprochen wird und wollen unsere Auswahlliste in das mittlere WINDOW schreiben. Das obere und untere WINDOW soll lediglich einen großen Balken darstellen. Im oberen WINDOW steht dann jeweils der Teil des Programms, in dem wir uns befinden, und im unteren WINDOW erfolgt dann die Dateneingabe, beziehungsweise die Weiterführung im Programm.

Der Aufbau eines Menüs müßte eigentlich mittlerweile durch die Beschreibung und Erklärung der Programme in den vorhergehenden Abschnitten hinreichend bekannt sein. Deshalb soll an dieser Stelle darauf nicht noch einmal explizit eingegangen werden.

2.3.2 Die Eingaberoutine

Den wichtigsten und zugleich programmtechnisch schwierigsten Teil einer Textverarbeitung stellt die Eingaberoutine dar. Die einfachste Form einer Eingabe haben wir bereits an anderer Stelle durch die INPUT-Funktion kennengelernt. Leider ist diese Funktion für die Textverarbeitung nicht geeignet. Deshalb müssen wir eine aufwendigere Eingaberoutine programmieren. Schauen Sie sich zu-

nächst einmal das Listing an, das wir danach besprechen werden.

```
1000 REM ===========
1010 REM TEXTEINGABE
1020 REM ===========
1030 MODE 2 :WINDOW#1,1,80,7,21 : WINDOW#2,1,80,1,5: WINDOW#3,11,15,3,
3:WINDOW#4,73,77,3,3:WINDOW#5,1,80,23,25
1040 CLS#1 :CLS#2: CLS#3 :CLS#4 :CLS#5
1050 LOCATE#2,30,3:PRINT#2, "T E X T E I N G A B E"
1060 LOCATE#2,4,3:PRINT#2,"ZEILE:": LOCATE#2,66,3: PRINT#2,"SEITE:"
1070 LOCATE #5,23,2: PRINT#5,"EINGABE BEENDEN DURCH : ~CTRL ↑~"
1080 S=AN
1090 FOR Z=ZEI+1 TO SL:Z$(S,Z)=""
1100 PRINT#4,S:PRINT#3,Z
1110 IF Z >SL THEN S=S+1:ZEI=0 : GOTO 1090
1120 PRINT#1,  CHR$(143);
1130 A$=INKEY$: IF A$=""THEN 1130
1140 T=ASC(A$):IF T=13 THEN PRINT#1,  CHR$(8);: PRINT#1," ":GOTO 1240
1150 IF T=30 THEN AN=S:ZEI=Z:Z$(S,Z+1)="§":RETURN
1160 IF T=127 AND LEN(Z$(S,Z))=0 THEN GOSUB 1260:GOTO 1130
1170 IF T=32 AND LEN(Z$(S,Z)) >KL THEN PRINT#1, CHR$(8);" ";CHR$(13):
GOTO 1240
1180 IF T=127  THEN Z$(S,Z)=LEFT$(Z$(S,Z),LEN(Z$(S,Z))-1) : PRINT#1, C
HR$(8);" ";CHR$(8);CHR$(8);CHR$(143);
1190 IF LEN(Z$(S,Z))=ZL THEN GOSUB 1260:GOTO 1130
1200 IF T<32 OR T>126 THEN 1130
1210 PRINT#1, CHR$(8);: PRINT#1, A$;:PRINT#1, CHR$(143); :Z$(S,Z)=Z$(S
,Z)+A$
1220 IF LEN(Z$(S,Z))=KL THEN GOSUB 1260
1230 GOTO 1130
1240 NEXT
1250 S=S+1:ZEI=0:GOTO 1090
1260 PRINT CHR$(7):RETURN
```

Bei der Textverarbeitung erweist sich die Möglichkeit des CPC 464, achtzig Zeichen auf dem Bildschim darzustellen, als sehr nützlich. Dadurch sind wir in der Lage, eine komplette Zeile in der Form, in der sie auf dem Drucker ausgegeben wird, auch auf dem Bildschirm anzuzeigen. Diese Darstellungsweise rufen wir mit dem Befehl

```
MODE 2
```

auf. Dieser Befehl hat allerdings einen Nachteil. Durch einen MODE-Befehl werden sämtliche vorhergehenden WINDOW-Definitionen gelöscht. Wir müssen uns also jetzt neue WINDOWS definieren.

Die Notwendigkeit, neue WINDOWS zu definieren, benutzen wir jetzt gleichzeitig, um noch einige weitere Funktionen auf unserem Bildschirm zu installieren. Wir möchten nämlich im oberen WINDOW einige Informationen über den Stand der Texteingabe erhalten. So möchten wir gerne permanent angezeigt haben, auf welcher Seite wir uns befinden und in welcher Zeile. Dazu definieren wir im oberen WINDOW zwei weitere WINDOWS, in denen dann die Seitenzahl und die Zeilennummer ausgegeben werden.

Die beiden WINDOWS müssen natürlich so klein gehalten werden, daß sie nur diese Zahlen ausgeben können und jeder Versuch, eine Zahl in dieses WINDOW zu schreiben dazu führt, daß alles vorherige, was darin ausgegeben wurde, gelöscht wird. In unserer Definition erfüllt dieses Erfordernis WINDOW Nummer drei und WINDOW Nummer vier.

Unsere Eingaberoutine ist eigentlich eine ganz große FOR-NEXT-Schleife. Der FOR-Befehl steht in Zeile 1090, der NEXT-Befehl befindet sich in Zeile 1240. In diesem FOR-NEXT-Befehl werden alle sechzig Zeilen einer Seite abgefragt. Sind diese sechzig Zeilen eingegeben, wird die Seitenzahl um eins erhöht und die Abfrage der sechzig Zeilen beginnt von neuem. Dies geschieht in Zeile 1250.

Die eigentliche Abfrage der Tastatur und damit die Eingabe des Textes erfolgt in Zeile 1130. Mit dem Befehl

```
A$=INKEY$
```

wird der Variablen A$ das gerade über die Tastatur eingegebene Zeichen zugeordnet. Da wir natürlich auch bei der Texteingabe wissen wollen, an welcher Stelle wir gerade sind, schreiben wir mit der Zeile 1120 einen Cursor auf den Bildschirm.

Bisher war die Sache ja noch relativ einfach, aber jetzt wird es kompliziert. Die verschiedenen Tasten haben selbstverständlich auch verschiedene Funktionen. Die einzelnen Buchstaben und Zahlen sollen als Buchstaben und Zahlen angenommen und in der Zeile ausgegeben werden. Wird aber zum Beispiel die DEL-Taste gedrückt, so soll das letzte Zeichen gelöscht werden. Wir müssen also jetzt

die unterschiedlichen Tasten bei der Eingabe unterscheiden. Nun wissen wir, daß die DEL-Taste den ASCII-Code 127 beinhaltet. In Zeile 1160 fragen wir ab, wenn diese Taste gedrückt wird und bisher noch gar keine Zeile existiert (das heißt, der Cursor steht in der ersten Spalte) soll nichts passieren, außer daß dies durch einen Piepton angezeigt wird. Dazu springen wir in die Subroutine in Zeile 1260.

Diese Subroutine gibt lediglich einen Piepton aus. Sie werden sich sicherlich wundern, daß dies mit einem PRINT-Befehl geschieht. Das ist möglich, da im ASCII-Code das Zeichen Nummer sieben die Ausgabe eines Pieptons beinhaltet. Deshalb kann durch den Befehl

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

ein Piepton erzeugt werden. Probieren Sie es einmal aus. Wenn bereits eine Zeile besteht und die DEL-Taste gedrückt wird, wird in Zeile 1180 der bis dahin erzeugte STRING um eine Zeile verkürzt. Mit Hilfe des Zeichens

```
CHR$(8)
```

wird die aktuelle Bildschirmposition des Cursors um eine Stelle nach links gesetzt. Dadurch wird es möglich, das letzte Zeichen zu löschen und durch nochmalige Eingabe dieses ASCII-Zeichens den Cursor auf seine richtige Position zu setzen.

In Zeile 1210 wird dann unser STRING aus dem bisherigen Zeilen-STRING und dem eingegebenen Buchstaben zusammengesetzt und richtig auf dem Bildschirm ausgegeben. In der nächsten Zeile wird dann überprüft, ob die Länge gerade unser Klingelzeichen erreicht hat; wenn dies der Fall ist, wird in die Subroutine 1260 gesprungen, und es ertönt ein Piepton.

Wir möchten jetzt noch erreichen, daß bei Überschreiten dieses Pieptons und einer danach folgenden Eingabe eines Leerzeichens automatisch in die nächste Zeile gesprungen wird, so daß nicht am Ende jeder Zeile die ENTER-Taste gedrückt werden muß. Diese Funktion erfüllt Zeile 1170, in der abgefragt wird, ob gerade die SPACE-Taste (= Leertaste) gedrückt wurde (ASCII Zeichen Nummer 32) und ob die Länge größer als unser Klingelzeichen ist.

In Zeile 1150 wird abgefragt, ob die Eingabe beendet ist. Das erreicht man durch gleichzeitiges Drücken der CTRL- und der "Pfeil nach oben"-Taste. Auf diese Art erhält man den ASCII-Code 30.

Beachten Sie bitte bei diesem Programm, daß das NEXT in unserer FOR-NEXT-Schleife nur dann durchgeführt wird, wenn mit Hilfe der ENTER-Taste oder mit der Leertaste bei Überschreiten der Zeilenlänge von 53 in die nächste Zeile gesprungen wird. Ansonsten befindet sich das Programm immer in einer Zeile und springt immer wieder zur Zeile 1130, in der die Tastatur abgefragt wird, zurück. Solange kein Zeichen eingegeben wird, befindet sich das Programm permanent in einer Warteschleife in der Zeile 1130.

2.3.3 Die Korrekturroutine

Fast so wichtig und eigentlich noch schwerer zu programmieren als die Eingaberoutine ist eine Korrekturroutine. Da bei der Eingabe der Text normalerweise nacheinander getippt wird, sollte eine Korrekturroutine sicherstellen, daß an jeder Stelle des Textes beliebige Änderungen angebracht werden können. Wir müssen deshalb in der Lage sein, mit dem Cursor über den gesamten Bildschirm zu fahren und an jeder Stelle Text eingeben oder Text löschen können. Eine solche Möglichkeit heißt in der Fachsprache FULL-SCREEN-EDIT. Bevor wir das Programm besprechen, sollten Sie es sich erst einmal genau anschauen.

```
4000 REM ===========
4010 REM KORRIGIEREN
4020 REM ===========
4030 MODE 2 : WINDOW#0,1,80,7,21 : WINDOW #2,1,80,1,5 : WINDOW #3,1,80
,22,25
4040 CLS#0 : CLS#2 : CLS#3
4050 LOCATE #2,27,3: PRINT#2,"K O R R I G I E R E N"
4060 LOCATE #3,3,2
4070 INPUT#3,"WELCHE SEITE";S
4080 LOCATE #3,3,3
4090 INPUT#3, "AB WELCHER ZEILE";ZZ
4100 CLS#3: LOCATE#3,23,3: PRINT#3,"BEENDEN MIT ~CTRL ↑~"
```

```
4110 L$=SPACE$(79):  L=ZZ+10:IF L >60 THEN L=60
4120 IF S=AN AND L>ZEI THEN L=ZEI
4130 : FOR Z=ZZ TO L : PRINT Z$(S,Z) : NEXT :LOCATE 1,1 : Z=ZZ  : C=1
: P=1
4140 K$=Z$(S,Z)+"  ":K1$="" :K2$=K$ : LOCATE 1,C: CALL &BB8D
4150 A$=INKEY$ : IF A$="" THEN 4150
4160 IF A$=CHR$(30) THEN Z$(S,Z)=K$ : RETURN
4170 k1=LEN(k$)-p+1: IF k1<1 THEN k1=1
4180 A=ASC(A$) :  IF A=&F1 OR A= &F5 OR A=&F9 OR A=13 THEN GOSUB 4360
:IF Z<L   THEN  Z=Z+1 : C=C+1 : LOCATE 1,C  : GOTO 4140
4190 IF A=&10 THEN P=P-1 : IF P<=0 THEN P=0
4200 IF A=&10 THEN GOTO 4150
4210 IF A=&F1 OR A= &F5 OR A=&F9 OR A=13 THEN GOTO 4140
4220 IF A=&F0 OR A= &F4 OR A=&F8  THEN  GOSUB 4360 :IF C<=1 THEN C=1 :
 LOCATE 1,C  : GOTO 4140
4230 IF A=&F0 OR A= &F4 OR A=&F8 THEN Z=Z-1 : C=C-1 : LOCATE 1,C  : GO
TO 4140
4240 IF A=&F3 OR A=&F7 OR A=&F8 THEN P=P+1 : IF P>LEN(K$) THEN P=1
4250 IF A=&F3 OR A=&F7 OR A=&FB THEN CALL &BB8D : LOCATE P,C : CALL &B
B8A : GOTO 4150
4260 IF A=&10 THEN P=P-1 : IF P<=0 THEN P=0
4270 IF A=&10 THEN 4150
4280 IF A=&F2 OR A=&F6 OR A=&FA THEN P=P-1 : IF P<1 THEN P=1 : GOTO 41
50
4290 IF A=&F2 OR A=&F6 OR A=&FA THEN CALL &BB8D : LOCATE P,C : CALL &B
B8A : GOTO 4150
4300 IF A=127 AND P>=1 THEN  K$=LEFT$(K$,P-1)+RIGHT$(K$,k1-1) :CALL &B
B8D : LOCATE 1,C : PRINT K$;" "; :LOCATE P,C : CALL &BB8A
4310 IF A=127 THEN  GOTO 4150
4320 IF A<32 OR A> 127 THEN 4150
4330 K$=LEFT$(K$,P)+A$+RIGHT$(K$,LEN(K$)-P) : IF LEN(K$)>=77 THEN K$=L
EFT$(K$,77)
4340 P=P+1 : IF P>77 THEN P=77
4350 CALL &BB8D : LOCATE 1,C : PRINT K$ : LOCATE P,C : CALL &BB8A : GO
TO 4150
4360 P=1 :LOCATE 1,C: PRINT K$;" "
4370 IF RIGHT$(K$,1)=" "THEN K$=LEFT$(K$,LEN(K$)-1): GOTO 4370
4380 Z$(S,Z)=K$ : RETURN
```

Wir wollen hier jeweils zehn Zeilen unseres Textes auf dem Bildschirm ausgeben und beliebig korrigieren können. Dazu wird zunächst abgefragt, in welcher Seite sich der zu ändernde Text befindet, und ab welcher Zeile er beginnt. Danach werden dann die folgenden zehn Zeilen auf dem Bildschirm dargestellt. Der Cursor befindet sich oben links in der Ecke. Da es in BASIC zu lange dauern würde, den Cursor immer wieder zu setzen, müssen wir uns einer kleinen Maschinenspracheroutine bedienen. Diese Routine ist bereits im BASIC-Interpreter vorgesehen. Dort befinden sich zwei Einsprungadressen, mit denen jeweils der Cursor gesetzt, beziehungsweise aufgehoben werden kann. So kann mit dem Befehl

```
CALL &BB8A
```

der Cursor an der Stelle gesetzt werden, die vorher mit dem LOCATE-Befehl definiert wurde. Mit dem Aufruf

```
CALL &BB8B
```

wird der Cursor wieder gelöscht. Diese beiden Routinen können wir uns in unserem Programm sehr gut zunutze machen. Wir brauchen jetzt nicht jedes Mal unseren Zeilen-STRING zu zerteilen, um den Cursor an die gewünschte Stelle zu setzen, sondern sind in der Lage, mit einem einfachen LOCATE und einem CALL-Aufruf den Cursor über die bestimmte Stelle des STRINGS zu setzen. Dadurch wird die Bewegungsmöglichkeit des Cursors wesentlich verbessert.

Für uns sind jetzt mehrere Tasten wichtig. Als erstes sind da einmal die Cursorsteuerungstasten oben rechts auf der Tastatur. Werden diese Tasten gedrückt, soll sich der Cursor in entsprechender Weise bewegen. Befindet er sich in der obersten Zeile, darf er diese nicht überschreiten. Dasselbe gilt in gleicher Form, wenn er sich in der untersten Zeile des Bildschirms befindet. Sonst muß aber jede Bewegung des Cursors mit diesen vier Tasten steuerbar sein. Wir führen zu diesem Ziel zwei weitere Laufvariablen P und C ein. C soll die jeweilige Zeile angeben, in der wir uns gerade befinden, mit P wird die Position innerhalb der Zeile beschrieben.

Wenn wir die Taste "Cursor rechts" drücken, soll P um eins erhöht werden. Anschließend soll mit Hilfe des LOCATE-Befehls der Cursor neu gesetzt werden. Bei der "Cursor links"-Taste muß analog P um eins erniedrigt werden, bei der "Cursor nach oben"- oder "Cursor nach unten"-Taste soll P den Wert eins erhalten und C wird je-

weils um eins erhöht oder erniedrigt. Das Drücken der ENTER-Taste muß natürlich die gleiche Wirkung haben wie das Drücken der "Cursor nach unten"-Taste. Wird die DEL-Taste gedrückt, soll das Zeichen unter dem Cursor gelöscht werden. Wird ein Buchstabe oder eine Zahl eingegeben, so soll dieses Zeichen in unseren Zeilen-STRING eingefügt werden. Das bedeutet, daß die nachfolgenden Buchstaben jeweils um ein Zeichen nach rechts verschoben werden müssen. Die Beendigung unserer Korrektur soll analog zur Eingaberoutine wiederum mit der CTRL- und gleichzeitig gedrückter "Pfeil nach oben"-Taste erfolgen. Die Abfrage dazu geschieht in Zeile 4160.

Dieses Programm sollten Sie sich wirklich einmal sehr genau anschauen. Das Programm sieht auf den ersten Blick sehr kompliziert aus, doch davon sollten Sie sich nicht abschrecken lassen. Wenn Sie einmal genau nachvollziehen, was die Tastatur-Codes bedeuten und welche Wirkungen sie auf die Variablen C und P haben, werden Sie sicherlich sofort verstehen, wie ein solcher FULL-SCREEN-EDIT funktioniert.

2.3.4 Sonstige Funktionen

Sicherlich brauchen wir an dieser Stelle nicht nocheinmal die Routinen für das Speichern und Laden von Daten besprechen. Deshalb sind sie hier nur abgedruckt.

```
2000 REM =============
2010 REM SPEICHERNTEXT
2020 REM =============
2030 CLS#1 : CLS#2 : CLS#3
2040 LOCATE 3,12:LOCATE #2,15,3: PRINT#2,"SPEICHERN"
2050 LOCATE #3,3,3: INPUT#3,"DATEINAME";A$
2060 OPENOUT A$
2070 FOR S=1 TO AN
2080 IF S=AN THEN 2120
2090 FOR Z=1 TO SL
2100 PRINT#9,Z$(S,Z)
2110 NEXT Z : NEXT S
2120 FOR Z=1 TO ZEI
2130 PRINT#9,Z$(S,Z)
2140 NEXT Z
2150 CLOSEOUT
2160 RETURN




3000 REM =========
3010 REM LADENTEXT
3020 REM =========
3030 MODE 2 : WINDOW#1,1,80,7,21 : WINDOW #2,1,80,1,5 : WINDOW #3,1,80
,22,25
3040 CLS#1 : CLS#2 : CLS#3
3050 LOCATE 3,12:LOCATE #2,32,3: PRINT#2,"T E X T  L A D E N"
3060 LOCATE #3,3,3: INPUT#3,"DATEINAME";A$
3070 S=1
3080 OPENIN A$
3090 CLS#1
3100 WHILE EOF=0
3110 FOR Z=1 TO SL
3120 INPUT#9,Z$(S,Z)
3130 IF EOF=-1 THEN 3170
3140 PRINT#1,Z$(S,Z)
3150 NEXT Z: S=S+1
3160 WEND
3170 CLOSEIN
3180 AN=S : ZEI=Z
3190 RETURN
```

Was wäre eine Textverarbeitung ohne eine Ausdruckroutine. Da wir hier eine Routine verwenden, die nicht sehr kompliziert ist, sei auch diese ohne größeren Kommentar abgedruckt. Es handelt sich um einfache FOR-NEXT-Schleifen und den Ausdruck der einzelnen Zeilen auf dem Drucker.

```
5000 REM ===========
5010 REM DRUCKENTEST
5020 REM ===========
5030 CLS#1 : CLS#2 : CLS #3
5040 LOCATE #2,13,3: PRINT#2,"DRUCKEN"
5050 LOCATE #3,3,3: INPUT#3,"IST EIN DRUCKER ANGESCHLOSSEN";A$
5060 IF LEFT$(UPPER$(A$),1)<>"J" THEN RETURN
5070 FOR S=1 TO AN
5080 FOR Z=1 TO SL
5090 IF S=AN AND Z=ZEI THEN RETURN
5100 PRINT#8,Z$(S,Z)
5110 NEXT Z
5120 NEXT S
```

Wir sollten uns jetzt noch damit beschäftigen, wie dieses kleine Programm, das schon eine beachtliche Leistungsfähigkeit aufweist, um weitere Funktionen erweitert werden kann.

Die meisten Textprogramme können im Blocksatz drucken. Das bedeutet, daß beim Ausdrucken jeweils rechts und links der Rand bündig ist. Der Ausdruck sieht dann so ähnlich aus wie im Buchsatz. Am einfachsten erreicht man dies, indem man in eine Zeile zusätzlich so viele Leerzeichen einzufügt, bis alle Zeilen die Maximallänge erreicht haben. Dies zu programmieren, dürfte Ihnen jetzt nicht mehr schwer fallen, da Sie ja bereits über einige Erfahrung verfügen. Wir möchten hier nur einige Tips geben, wie ein solches Problem gelöst werden kann. Sie könnten zunächst einmal mit Hilfe des INSTR$-Befehls die Stelle eines Leerzeichen heraussuchen, und über LEFT$ und RIGHT$ an dieser Stelle ein Leerzeichen einfügen. Wie Zeichen in einen bestehenden String eingefügt werden, können Sie oben in der Korrekturroutine ersehen. Danach können Sie die Länge der Strings abfragen, und wenn er immer noch zu kurz ist, kann die Routine wiederholt werden. Damit müßten dann alle Zeilen durchgegangen werden. Das Prinzip ist recht einfach. Probieren Sie es doch einmal aus!

Viele der heute erhältlichen Drucker beherrschen auch den deutschen Zeichensatz. Dort ist für die Umlaute und das Zeichen "ß" jeweils ein ASCII-Zeichen benutzt worden. Leider werden diese Zeichen in ihrer ursprünglichen amerikanischen Form auf dem Bildschirm dargestellt. Für Textverarbeitungszwecke ist es in der Regel sinnvoll, auch die deutschen Zeichen auf dem Bildschirm darzustellen. Wenn dies nicht geschieht, ist es unter Umständen sehr mühseelig, die Umlaute und das "ß" zu finden. Wenn Sie die deutschen Umlaute auf dem Bildschirm sichtbar machen wollen, können Sie dazu folgendes Programm verwenden, das den Zeichensatz umdefiniert.

```
100 REM ========================
110 REM DEUTSCHER ZEICHENSATZ
120 REM ====================
130 A=&5B : B=&7B :O=&5C:P=&7C : U=&5D : V=&7D : S=&7E
140 UEG1=&X1100110 : UEG2=&X0 : UEG3=&X1100110 : UEG4=UEG3 : UEG5=UEG3
 : UEG6=UEG3 : UEG7=&X111100
150 UEK1=&X1100110 : UEK2=0 : UEK3=UEK1 : UEK4=UEK1 : UEK5=UEK1 : UEK6
=UEK1 : UEK7=&X111110: UEK8=0
160 AEG1=&X1100110 : AEG2=0 : AEG3=&X11000 : AEG4=&X111100 : AEG5=&X11
00110 : AEG6=&X1111110 : AEG7=AEG5 : AEG8=0
170 AEK1=AEG1 : AEK2=0 : AEK3=&X1111000 : AEK4=&X1100: AEK5=&X1111100
: AEK6=&X11001100: AEK7=&X111011 : AEK8=0
180 OEK1=AEK1 : OEK2=0 : OEK3=&X1111000 : OEK4=&X11001100 : OEK5=OEK4
: OEK6=OEK4 : OEK7=OEK3 : OEK8=0
190 OEG1=OEK1: OEG2=&X111000 : OEG3=&X1101100 : OEG4=&X11000110 : OEG5
=OEG4 : OEG6=OEG3 : OEG7=OEG2 : OEG8=0
200 SZ1=&X111100 : SZ2=&X1100110 : SZ3=SZ2 : SZ4=&X1111100 : SZ5=SZ2 :
 SZ6=SZ2 : SZ7=&X1100100 : SZ8=&X1100000
210 SA=MIN (A,B,O,P,U,V,S)
220 SYMBOL AFTER SA
230 SYMBOL A,AEG1,AEG2,AEG3,AEG4,AEG5,AEG6,AEG7,AEG8
240 SYMBOL B,AEK1,AEK2,AEK3,AEK4,AEK5,AEK6,AEK7,AEK8
250 SYMBOL O,OEG1,OEG2,OEG3,OEG4,OEG5,OEG6,OEG7,OEG8
260 SYMBOL P,OEK1,OEK2,OEK3,OEK4,OEK5,OEK6,OEK7,OEK8
270 SYMBOL U,UEG1,UEG2,UEG3,UEG4,UEG5,UEG6,UEG7,UEG8
280 SYMBOL V,UEK1,UEK2,UEK3,UEK4,UEK5,UEK6,UEK7,UEK8
290 SYMBOL S,SZ1,SZ2,SZ3,SZ4,SZ5,SZ6,SZ7,SZ8
```

2.3.5 Das komplette Programm

Zum Schluß dieses Abschnitts wollen wir das komplette Textverarbeitungsprogramm in der Form, wie oben entwickelt wurde, zusammensetzen. Die angesprochenen Erweiterungsmöglichkeiten wurden noch nicht berücksichtigt. Diese Erweiterungen sollen Ihnen überlassen bleiben. Sie sollten zugleich Ansporn und Herausforderung sein, um ein solches Programm, das schon in seiner Grundstruktur sehr leistungsfähig ist, weiterzuentwickeln. Hier das komplette Listing:

```
1 REM ============
2 REM TEXTPROGRAMM
3 REM ============
10 REM ========
20 REM VORSPANN
30 REM ========
40 MODE 1
50 INK 0,1: INK 1,24: INK 2,3 : BORDER 1
60 PAPER #2,1 : PEN #2,2 : PAPER #3,1 : PEN #3,2 : PAPER #4,1 : PEN #4
,2 : PAPER #5,1 : PEN #5,2
70 DIM Z$(60,60): AN=1 : ZEI=0 : ZL=60 : KL=53 : SL=60
100 REM =====
110 REM MENUE
120 REM =====
130 CLS : MODE 1
140 WINDOW#2,1,40,1,5 : WINDOW#1,1,40,6,20: WINDOW#3,1,40,21,25
150 CLS#1: CLS#2 : CLS#3
160 LOCATE#2,16,3: PRINT#2,"M E N U E"
170 LOCATE#1,13,3:PRINT#1,"1  TEXT EINGEBEN"
180 LOCATE#1,13,5:PRINT#1,"2  TEXT SPEICHERN"
190 LOCATE#1,13,7:PRINT#1,"3  TEXT LADEN"
200 LOCATE#1,13,9:PRINT#1,"4  KORRIGIEREN"
210 LOCATE#1,13,11:PRINT#1,"5  DRUCKEN"
220 CLS#3:LOCATE#3,3,3: INPUT#3,"WELCHE NUMMER";A$
230 A=VAL(A$): IF A<1 OR A>5 THEN 220
240 ON A GOSUB 1000,2000,3000,4000,5000
250 GOTO 130
1000 REM ============
1010 REM DATENEINGABE
1020 REM ============
1030 MODE 2 :WINDOW#1,1,80,7,21 : WINDOW#2,1,80,1,5: WINDOW#3,11,15,3,
3:WINDOW#4,73,77,3,3:WINDOW#5,1,80,23,25
1040 CLS#1 :CLS#2: CLS#3 :CLS#4 :CLS#5
1050 LOCATE#2,30,3:PRINT#2, "T E X T E I N G A B E"
1060 LOCATE#2,4,3:PRINT#2,"ZEILE:": LOCATE#2,66,3: PRINT#2,"SEITE:"
1070 LOCATE #5,23,2: PRINT#5,"EINGABE BEENDEN DURCH : 'CTRL ↑'"
1080 S=AN
1090 FOR Z=ZEI+1 TO SL:Z$(S,Z)=""
1100 PRINT#4,S:PRINT#3,Z
1110 IF Z >SL THEN S=S+1:ZEI=0 : GOTO 1090
1120 PRINT#1,  CHR$(143);
```

```
1130 A$=INKEY$: IF A$=""THEN 1130
1140 T=ASC(A$):IF T=13 THEN PRINT#1,  CHR$(8);: PRINT#1," ":GOTO 1240
1150 IF T=30 THEN AN=S:ZEI=Z:Z$(S,Z+1)="§":RETURN
1160 IF T=127 AND LEN(Z$(S,Z))=0 THEN GOSUB 1260:GOTO 1130
1170 IF T=32 AND LEN(Z$(S,Z))>KL THEN PRINT#1, CHR$(8);" ";CHR$(13):
GOTO 1240
1180 IF T=127  THEN Z$(S,Z)=LEFT$(Z$(S,Z),LEN(Z$(S,Z))-1) : PRINT#1, C
HR$(8);" ";CHR$(8);CHR$(8);CHR$(143);
1190 IF LEN(Z$(S,Z))=ZL THEN GOSUB 1260:GOTO 1130
1200 IF T<32 OR T>126 THEN 1130
1210 PRINT#1, CHR$(8);: PRINT#1, A$;:PRINT#1, CHR$(143); :Z$(S,Z)=Z$(S
,Z)+A$
1220 IF LEN(Z$(S,Z))=KL THEN GOSUB 1260
1230 GOTO 1130
1240 NEXT
1250 S=S+1:ZEI=0:GOTO 1090
1260 PRINT CHR$(7):RETURN
2000 REM =========
2010 REM SPEICHERN
2020 REM =========
2030 CLS#1 : CLS#2 : CLS#3
2040 LOCATE 3,12:LOCATE #2,15,3: PRINT#2,"SPEICHERN"
2050 LOCATE #3,3,3: INPUT#3,"DATEINAME";A$
2060 OPENOUT A$
2070 FOR S=1 TO AN
2080 IF S=AN THEN 2120
2090 FOR Z=1 TO SL
2100 PRINT#9,Z$(S,Z)
2110 NEXT Z : NEXT S
2120 FOR Z=1 TO ZEI
2130 PRINT#9,Z$(S,Z)
2140 NEXT Z
2150 CLOSEOUT
2160 RETURN
3000 REM =====
3010 REM LADEN
3020 REM =====
3030 MODE 2 : WINDOW#1,1,80,7,21 : WINDOW #2,1,80,1,5 : WINDOW #3,1,80
,22,25
3040 CLS#1 : CLS#2 : CLS#3
3050 LOCATE 3,12:LOCATE #2,32,3: PRINT#2,"T E X T  L A D E N"
```

```
3060 LOCATE #3,3,3: INPUT#3,"DATEINAME";A$
3070 S=1
3080 OPENIN A$
3090 CLS#1
3100 WHILE EOF=0
3110 FOR Z=1 TO SL
3120 INPUT#9,Z$(S,Z)
3130 IF EOF=-1 THEN 3170
3140 PRINT#1,Z$(S,Z)
3150 NEXT Z: S=S+1
3160 WEND
3170 CLOSEIN
3180 AN=S : ZEI=Z
3190 RETURN
4000 REM ===========
4010 REM KORRIGIEREN
4020 REM ===========
4030 MODE 2 : WINDOW#0,1,80,7,21 : WINDOW #2,1,80,1,5 : WINDOW #3,1,80
,22,25
4040 CLS#0 : CLS#2 : CLS#3
4050 LOCATE #2,27,3: PRINT#2,"K O R R I G I E R E N"
4060 LOCATE #3,3,2
4070 INPUT#3,"WELCHE SEITE";S
4080 LOCATE #3,3,3
4090 INPUT#3, "AB WELCHER ZEILE";ZZ
4100 CLS#3: LOCATE#3,23,3: PRINT#3,"BEENDEN MIT 'CTRL ↑'"
4110 L$=SPACE$(79):  L=ZZ+10:IF L >60 THEN L=60
4120 IF S=AN AND L>ZEI THEN L=ZEI
4130 : FOR Z=ZZ TO L : PRINT Z$(S,Z) : NEXT :LOCATE 1,1 : Z=ZZ  : C=1
: P=1
4140 K$=Z$(S,Z)+"  ":K1$="" :K2$=K$ : LOCATE 1,C: CALL &BB8D
4150 A$=INKEY$ : IF A$="" THEN 4150
4160 IF A$=CHR$(30) THEN Z$(S,Z)=K$ : RETURN
4170 k1=LEN(k$)-p+1: IF k1<1 THEN k1=1
4180 A=ASC(A$) :  IF A=&F1 OR A= &F5 OR A=&F9 OR A=13 THEN GOSUB 4360
:IF Z<L   THEN  Z=Z+1 : C=C+1 : LOCATE 1,C  : GOTO 4140
4190 IF A=&10 THEN P=P-1 : IF P =0 THEN P=0
4200 IF A=&10 THEN GOTO 4150
4210 IF A=&F1 OR A= &F5 OR A=&F9 OR A=13 THEN GOTO 4140
4220 IF A=&F0 OR A= &F4 OR A=&F8  THEN  GOSUB 4360 :IF C<=1 THEN C=1 :
 LOCATE 1,C  : GOTO 4140
```

```
4230 IF A=&F0 OR A= &F4 OR A=&F8 THEN Z=Z-1 : C=C-1 : LOCATE 1,C  : GO
TO 4140
4240 IF A=&F3 OR A=&F7 OR A=&F8 THEN P=P+1 : IF P>LEN(K$) THEN P=1
4250 IF A=&F3 OR A=&F7 OR A=&FB THEN CALL &BB8D : LOCATE P,C : CALL &B
B8A : GOTO 4150
4260 IF A=&10 THEN P=P-1 : IF P<=0 THEN P=0
4270 IF A=&10 THEN 4150
4280 IF A=&F2 OR A=&F6 OR A=&FA THEN P=P-1 : IF P<1 THEN P=1 : GOTO 41
50
4290 IF A=&F2 OR A=&F6 OR A=&FA THEN CALL &BB8D : LOCATE P,C : CALL &B
B8A : GOTO 4150
4300 IF A=127 AND P>=1 THEN  K$=LEFT$(K$,P-1)+RIGHT$(K$,k1-1) :CALL &B
B8D : LOCATE 1,C : PRINT K$;" "; :LOCATE P,C : CALL &BB8A
4310 IF A=127 THEN  GOTO 4150
4320 IF A<32 OR A>127 THEN 4150
4330 K$=LEFT$(K$,P)+A$+RIGHT$(K$,LEN(K$)-P) : IF LEN(K$)>=77 THEN K$=L
EFT$(K$,77)
4340 P=P+1 : IF P>77 THEN P=77
4350 CALL &BB8D : LOCATE 1,C : PRINT K$ : LOCATE P,C : CALL &BB8A : GO
TO 4150
4360 P=1 :LOCATE 1,C: PRINT K$;" "
4370 IF RIGHT$(K$,1)=" "THEN K$=LEFT$(K$,LEN(K$)-1): GOTO 4370
4380 Z$(S,Z)=K$ : RETURN
5000 REM =======
5010 REM DRUCKEN
5020 REM =======
5030 CLS#1 : CLS#2 : CLS #3
5040 LOCATE #2,13,3: PRINT#2,"DRUCKEN"
5050 LOCATE #3,3,3: INPUT#3,"IST EIN DRUCKER ANGESCHLOSSEN";A$
5060 IF LEFT$(UPPER$(A$),1)<>"J" THEN RETURN
5070 FOR S=1 TO AN
5080 FOR Z=1 TO SL
5090 IF S=AN AND Z=ZEI THEN RETURN
5100 PRINT#8,Z$(S,Z)
5110 NEXT Z
5120 NEXT S
```

2.4 BASIC-Speicher und BASIC-TOKENS

BASIC-Programme legt der Schneider CPC 464 in seinem Speicher ab der Adresse 170 HEX ab. In Dezimalzahlen gerechnet beginnt der BASIC-Speicher bei der Speicherstelle 368. Um diesen Speicherbereich aktiv nutzen, sollte man sich ein wenig damit beschäftigen, wie der Schneider seine Programme abspeichert.

Jedes eingegebene BASIC-Programm wird Zeile für Zeile in einem besonders hierfür reservierten RAM-Bereich abgelegt. Wir wollen uns nun einmal anschauen, wie das aussieht. Dazu geben Sie bitte folgendes kleine Programm ein:

```
10 DATA
20 PRINT
30 GOTO 10
```

Dieses Programm liegt jetzt im Speicherbereich von 368 bis 389. Schauen wir uns diesen Speicherbereich einmal an. Dazu geben Sie bitte folgende Zeile ein:

```
For i = 368 to 389 : PRINT I, PEEK (i) : NEXT
```

Nach Drücken der ENTER-Taste erhalten Sie folgendes Bild auf dem Bildschirm:

368	6
369	0
370	10
371	0
372	140
373	0
374	6
375	0
376	20
377	0
378	191
379	0
380	10
381	0
382	30
383	0
384	160
385	32
386	30
387	10
388	0
399	0

Wenn bei Eingabe dieser Zeile Ihr Rechner einen Syntaxfehler (Syntax error) meldet, sollten Sie darauf achten, daß nach jedem BASIC-Befehl ein Leerzeichen stehen muß. Wenn Sie diese Leerzeichen weglassen, kann der Rechner die BASIC-Befehle nicht exakt abgrenzen. In der oben angegebenen FOR-NEXT-Schleife muß jeweils der Anfangs- und Endwert des Speicherbereichs stehen, den Sie sich gerade anschauen wollen. Durch Ändern dieser beiden Zahlen können Sie sich jeden Speicherbereich ansehen.

Was bedeutet unsere Liste? Wenn Sie genau hinschauen, sehen Sie, daß in Zeile 370 eine 10, in Zeile 376 eine 20 und in Zeile 382 eine 30 steht. Das sind die Zeilennummern des oben eingegebenen Programms. Wir erinnern uns: wir hatten drei Zeilen eines kleinen Programms eingegeben mit den Zeilennummern 10, 20 und 30. Schauen wir uns die erste Zeile nochmal an:

```
10 DATA
```

Unsere Liste ab Zeile 368 beginnt allerdings nicht mit der Nummer 10 sondern vorher steht noch eine 6. Diese 6 gibt die Länge des

Speicherbereichs an, den die Zeile 10 im Speicher belegt.

Jetzt können wir auch schon sofort sehen, warum BASIC-Zeilen nur 256 Zeichen enthalten können. In jeder Speicherzelle kann eine Zahl zwischen 0 und 255 gespeichert werden. In Zelle 369 steht eine Null. Diese Null befindet sich in jeder Zeile zwischen der Angabe der Länge und der Zeilennummer. In Zelle 370 steht jetzt die Zeilennummer, in 371 eine 0. Für die Zeilennummer werden beide Speicherzellen benötigt, da sonst nur Zeilennummern bis 255 möglich wären.

Wird jetzt eine Zeilennummer größer als 255 gewählt, berechnet der Interpreter die Zeilennummer folgendermaßen: Den Wert der Zelle 371 multipliziert er mit 256 und addiert dazu den Wert der Zelle 370. Wenn Sie dies durchführen, kommen Sie auf den Wert 10, da 0 x 256 0 ergibt. Wenn Sie sich dies einmal mit größeren Zeilennummern ansehen wollen, geben Sie einfach ein: RENUM 1000. Danach belegt Ihr Programm die Zeilennummern 1000, 1010 und 1020. Wenn Sie jetzt noch einmal die oben angegebene FOR-NEXT-Schleife eingeben, erhalten Sie in Zelle 370 den Wert 232 und in Zelle 371 den Wert 3. Wenn Sie nachrechnen, ergibt 3 x 256 + 232 genau die Zahl 1000.

In der Fachsprache nennt man diese Aufspaltung einer Zahl in zwei Bytes eine Aufspaltung in LOWER BYTE und HIGHER BYTE. Dabei ist das HIGHER BYTE die Zahl, die mit 256 multipliziert wird und das LOWER BYTE die Zahl, die addiert wird.

Doch bleiben wir bei unserer ursprünglichen Liste. Nach der Zeilennummer, die zwei Speicherplätze belegt, finden wir in Zelle 372 die Zahl 140. Dies ist der Code, mit dem der BASIC-Befehl DATA im Speicher abgelegt wird. Die folgende Null zeigt, daß die Zeile beendet ist, die 6 in Zeile 374 gibt wiederum die Länge der nächsten BASIC-Zeile an. Dort finden wir den Befehl PRINT in Zelle 378 mit dem Wert 191 abgespeichert. Ab Zelle 380 beginnt dann die 3. Zeile. Sie belegt insgesamt 10 Speicherplätze, wie uns die Zelle 380 anzeigt. In 384 finden wir 160. Dies ist der Code für den GOTO-Befehl. Die 32 in Zelle 385 stellt das Leerzeichen nach dem GOTO-Befehl dar.

Die 30 in Zelle 386 zeigt an, daß nun die Zeilennummer kommt, in die gesprungen werden soll. Die Zeilennummer wird dann in Zelle 387 und 388 als LOWER BYTE und HIGHER BYTE abgespeichert. Da in unserem Programm zur Zeile 10 gesprungen werden soll, finden Sie

in Zelle 387 eine 10 und in 388 eine Null. Ab Zelle 389 werden Sie nur noch Nullen finden, die anzeigen, daß das Programm beendet ist.

Wir haben oben gesehen, daß die BASIC-Befehle mit einem Code abgespeichert werden. Diesen Code nennt man in der Fachsprache TOKEN. Wir haben auch gesehen, daß das Leerzeichen mit der Zahl 32 abgespeichert wird. 32 ist die Zahl, die das Leerzeichen im ASCII-Code beinhaltet. Verallgemeinernd läßt sich feststellen, daß eingegebene Zeichenketten mit ihrem ASCII-Code abgespeichert werden. Da der ASCII-Code die Zahlen kleiner 128 belegt, bleiben für die BASIC-TOKENS lediglich die Zahlen von 128 bis 255 übrig, da in jedem Byte lediglich eine Zahl zwischen 0 und 255 gespeichert werden kann. Folglich stehen für die BASIC-TOKENS 128 Zahlen zur Verfügung. Dies reicht bei weitem für den umfangreichen Wortschatz des LOCOMOTIVE-BASIC nicht aus.

Der Schneider beherrscht insgesamt 170 BASIC-Befehle. Dadurch ist es nötig, BASIC-TOKENS mehrfach zu belegen. Der Rechner belegt lediglich das TOKEN 255 doppelt. Dies geschieht so, daß in der ersten Zelle, in der der Befehl abgespeichert wird, eine 255 steht und direkt dahinter eine andere Zahl, die angibt, um welchen Befehl es sich handelt. Diese Zahl liegt immer im Bereich zwischen 0 und 127. Diese Befehle belegen immer zwei Bytes, während andere Befehle nur ein Byte im Speicher belegen.

Die Codes aller BASIC-Befehle können Sie der Liste der BASIC-TOKENS in diesem Kapitel entnehmen. Diese Liste ist vollständig und enthält drei Befehle, die nicht im Handbuch beschrieben sind. Der Befehl DEC$(255,114) ist zwar als TOKEN vorhanden, allerdings ist der Interpreter nicht in der Lage, diesen Befehl zu verarbeiten. Dies wäre auch unsinnig, da die Syntax des Befehls darauf schließen läßt, daß dieser Befehl eine Binärzahl oder eine Hexadezimalzahl in eine Dezimalzahl umwandeln soll. Dies geschieht aber bei dem Interpreter automatisch, ohne daß ein Befehl nötig ist. Da dieser Befehl als BASIC-TOKEN vorhanden ist, gehört DEC$ zu den reservierten Wörtern und ist nicht als Variable verwendbar. Dies geht leider aus dem Handbuch nicht hervor. Der Befehl mit dem TOKEN 149 ist ebenfalls im Handbuch nicht erklärt. Das ist auch verständlich, da es sich um einen Steuerungsbefehl für die Floppy-Disk handelt.

Tabelle 2.1 BASIC-TOKENS 128-207

128 AFTER	155 ERASE	182 OPENIN
129 AUTO	156 ERROR	183 OPENOUT
130 BORDER	157 EVERY	184 ORIGIN
131 CALL	158 FOR	185 OUT
132 CAT	159 GOSUB	186 PAPER
133 CHAIN	160 GOTO	187 PEN
134 CLEAR	161 IF	188 PLOT
135 CLG	162 INK	189 PLOTR
136 CLOSEIN	163 INPUT	190 POKE
137 CLOSEOUT	164 KEY	191 PRINT
138 CLS	165 LET	192 '
139 CONT	166 LINE	193 RAD
140 DATA	167 LIST	194 RANDOMIZE
141 DEF	168 LOAD	195 READ
142 DEFINT	169 LOCATE	196 RELEASE
143 DEFREAL	170 MEMORY	197 REM
144 DEFSTR	171 MERGE	198 RENUM
145 DEG	172 MID$	199 RESTORE
146 DELETE	173 MODE	200 RESUME
147 DIM	174 MOVE	201 RETURN
148 DRAW	175 MOVER	202 RUN
149 DRAWR	176 NEXT	203 SAVE
150 EDIT	177 NEW	204 SOUND
151 ELSE	178 ON	205 SPEED
152 END	179 ON BREAK	206 STOP
153 ENT	180 ON ERROR GOTO	207 SYMBOL
154 ENV	181 ON SQ	

Tabelle 2.2 BASIC-TOKENS 208-255

208 TAG	227 ERL	242 <>
209 TAGOFF	228 FN	243 <=
210 TROFF	229 SPC	244 +
211 TRON	230 STEP	245 -
212 WAIT	231 SWAP	246 *
213 WEND	232-233 unbenutzt	247 /
214 WHILE	234 TAB	248 ^
215 WIDTH	235 THEN	249 \
216 WINDOW	236 TO	250 AND
217 WRITE	237 USING	251 MOD
218 ZONE	238 >	252 OR
219 DI	239 =	253 XOR
220 EI	240 >=	254 NOT
221-226 unbenutzt	241 <	255 (Tabelle 2.3)

Tabelle 2.3 BASIC-TOKEN 255

0	ABS	19	REMAIN	71	XPOS
1	ASC	20	SIGN	72	YPOS
2	ATN	21	SIN	73-112	unbenutzt
3	CHR$	22	SPACE$	113	BIN$
4	CINT	23	SQ	114	DEC$
5	COS	24	SQR	115	HEX$
6	CREAL	25	STR$	116	INST
7	EXP	26	TAN	117	LEFT$
8	FIX	27	UNT	118	MAX
9	FRE	28	UPPER$	119	MIN
10	INKEY	29	VAL	120	POS
11	INP	30-63	unbenutzt	121	RIGHT$
12	INT	64	EOF	122	ROUND
13	JOY	65	ERR	123	STRING$
14	LEN	66	HIMEM	124	TEST
15	LOG	67	INKEY$	125	TESTR
16	LOG10	68	PI	126	unbenutzt
17	LOWER$	69	RND	127	VPOS
18	PEEK	70	TIME		

Um einen interessanten zusätzlichen Befehl handelt es sich bei BASIC-TOKEN 192. Es handelt sich um den Apostroph, der auf der Tastatur oberhalb der 7 zu finden ist. Dieser Befehl reagiert genau gleich mit dem REM-Befehl. Mit ihm lassen sich Kommentare mit dem Druck einer Taste einfügen, die dann vom Interpreter nicht beachtet werden. Sobald der Interpreter dieses Zeichen in einer Zeile findet, springt er automatisch in die nächste Zeile, genau wie es auch beim REM-Befehl geschieht. Man kann also dieses Zeichen als Abkürzung für den REM-Befehl benutzen.

Die Frage stellt sich, wofür wir die Kenntnisse über die Speicherbelegung und die BASIC-TOKENS in einem Programm benutzen können. Da wir mit dem Befehl POKE Zahlen direkt in den Speicher schreiben können, sind wir in der Lage, ein Programm während seines Laufs zu verändern, wenn wir wissen, wo welche Zahl stehen muß. So ist zum Beispiel möglich, häufig benötigte Routinen sowohl als INPUT- oder auch als PRINT-Routine zu benutzen, indem während des Programmlaufs durch einen POKE-Befehl der PRINT-Befehl in einen INPUT-Befehl verwandelt wird oder umgekehrt.

Interessant wird dies zum Beispiel bei einer Adressenverwaltung, bei der Anschriften sowohl eingegeben als auch ausgegeben werden. Wir wollen uns das anhand eines kleinen Programms einmal ansehen. Geben Sie bitte das folgende kleine Programm so ein, wie es hier steht, sonst funktioniert es nicht. (Denken Sie auch an die Leerräume!)

```
10 INPUT"NAME:";A$
20 INPUT"STRASSE:";B$
30 INPUT"ORT:";C$
40 PRINT
50 FOR I=1 TO 3 : READ A : POKE A,191 : NEXT : POKE 428,152
60 GOTO 10
70 DATA 372,390,411
```

Wenn Sie das Programm starten, fragt der Rechner erst die Variablen, Namen, Straße und Ort ab, druckt danach die drei Variablen aus, die Sie eingegeben haben und beendet dann das Programm. Wenn Sie jetzt das Programm listen, sehen Sie, daß sich das Aussehen grundlegend verändert hat. Es sieht nun folgendermaßen aus: Jeder INPUT-Befehl wurde in dem Programm durch den PRINT-Befehl ersetzt und in Zeile 40 findet sich der END-Befehl.

Was ist passiert? Schauen wir uns das Programm einmal näher an: In Zeile 70 sind drei Zahlen als DATAs abgespeichert. Diese drei Zahlen stellen die Speicherstellen dar, in denen jeweils der INPUT-Befehl gespeichert ist. In Zeile 50 wird in einer POKE-Schleife in jede dieser Speicherstellen eine neue Zahl geschrieben. Diese Zahl ist 191 und stellt das BASIC-TOKEN für den PRINT-Befehl dar. Danach wird mit einem POKE-Befehl in Zeile 40 der PRINT-Befehl in einen END-Befehl verwandelt. Deshalb sieht das Programm jetzt folgendermaßen aus:

```
10 PRINT"NAME:";A$
20 PRINT"STRASSE:";B$
30 PRINT"ORT:";C$
40 END
```

Diese Technik, die hier in einem kleinen Programm vorgeführt wurde, läßt sich bei großen und komplizierten Algorithmen sehr gut verwenden, um dadurch Speicherplatz und Programmierzeit zu sparen. Wenn bei Ihnen der Programmlauf nicht wie vorgesehen funktioniert, kann das daran liegen, daß Sie zusätzliche Leerzeichen eingegeben haben oder durch Korrekturen das Programm vom Interpreter in leicht veränderter Form abgelegt wurde. Dann sollten Sie sich die Speicherbelegung anschauen und nachsehen, ob die angegebenen Speicherstellen richtig sind, bzw. sich die richtigen Speicherstellen heraussuchen und dann die Zeile 70 und die Zeile 50 anpassen.

2.5 Ein einfacher LIST-Schutz

Die Kenntnis der BASIC-TOKENS ermöglicht es uns auch, einen einfachen LIST-Schutz einzuführen. Damit der Interpreter das Listen abbricht, muß nur ein BASIC-TOKEN in das Programm geschrieben werden, das der Interpreter nicht erkennen kann. Dann bricht er das Listen mit der Meldung SYNTAX ERROR ab. Aus unserer Liste der BASIC-TOKENS sehen wir, daß zum Beispiel die Nummer 232 nicht belegt ist. Um das zu demonstrieren, geben Sie bitte folgendes kleine Programm ein:

```
10 PRINT "LISTSCHUTZ"
15 GOTO 30
30 PRINT
40 PRINT "OK"
```

Geben Sie jetzt bitte den Befehl: POKE 401,232 ein und versuchen Sie dann, das Programm zu listen. Sie werden sehen, daß nur noch die ersten beiden Zeilen des Programms erscheinen und danach die Meldung SYNTAX ERROR ausgegeben wird. Wenn Sie das Programm starten, läuft es wunschgemäß ab. Dies ist nur möglich durch den Befehl in Zeile 15: GOTO 30. Dadurch wird die Stelle, in der ein BASIC-Befehl steht, den der Interpreter nicht erkennen kann, übersprungen. Folglich kann beim Programmlauf auch dieser Befehl nicht erkannt werden, beim Listing wird an dieser Stelle das Listen abgebrochen. Auf diese Art und Weise ist es möglich, einen einfachen LIST-Schutz einzufügen.

2.6 Tips zur BASIC-Programmierung

2.6.1 BASIC-Programme kürzer und schneller

Der Schneider CPC 464 ist zwar von Haus aus reichlich mit RAM-Speicher ausgerüstet, und der BASIC-Interpreter ist sehr schnell. Dennoch empfiehlt es sich, bei größeren Programmen im Interesse der Programmablaufgeschwindigkeit und des verfügbaren Speicherplatzes einige Hinweise zu beachten.

Zunächst sollen einmal einige Tips zur Verkürzung von BASIC-Programmen gegeben werden. Sie gelten allgemein und nicht speziell für den Schneider CPC 464.

Folgende Hinweise sollten Sie grundsätzlich beachten:

1. Lassen Sie REM-Anweisungen weg.

2. Speichern Sie häufig benutzte Konstanten (zum Beispiel 3.78) einmal in einer Variablen ab. Arbeiten Sie dann nur noch mit dieser Variablen.

3. Benutzen Sie dieselben Variablen immer wieder für verschiedene Aufgabenstellungen.

4. Schreiben Sie häufig gebrauchte Programmteile als Subroutinen.

5. Packen Sie möglichst viele Befehle in eine Zeile.

6. Lassen Sie hinter NEXT den Variablennamen weg.

7. Lassen Sie in der letzten Zeile die END-Anweisung weg.

8. Benutzen Sie bei indizierten Variablen auch immer Index 0 mit.

9. Wählen Sie kurze Variablennamen.

Einige Punkte sollten etwas näher betrachtet werden.

1. Jede Zeilennummer wird im BASIC in 2 Bytes abgespeichert. Dazu kommen dann noch 2 Bytes für die Zeilenlänge und 1 Byte für das Zeilenende. Wenn die BASIC-Zeilen mit möglichst vielen Befehlen vollgepackt werden, können zumindest die Zeilennummern gespart werden. Der Doppelpunkt als Trennzeichen belegt ein Byte. Jede Zeile belegt aber mehr als ein Byte, da die Zeilennummer schon mehrere Bytes belegt und darüber hinaus noch einige Bytes für die Verkettung der einzelnen Zeilen benötigt werden. Der BASIC-Interpreter des Schneider ist in der Lage, bis zu 255 Zeichen in einer Zeile zu verarbeiten. Das bedeutet, daß für ein Programm, das eine Länge von einem Kilobyte hat, lediglich 4 Zeilen benötigt werden.

2. Achten Sie darauf, daß Routinen, die öfter gebraucht werden, als Subroutine geschrieben werden. Die Beendigung einer Subroutine erfolgt mit RETURN. Dieser Befehl verbraucht lediglich ein Byte. Sollten Sie aber mit einer Sprunganweisung das Unterprogramm beenden, müssen Sie wissen, daß der Befehl GOTO 100 vier Bytes in Anspuch nimmt. Sprungbefehle sind wahre Bytefresser. Deshalb arbeiten Sie am besten immer mit einer Subroutine.

3. Planen Sie Ihr Programm von vornherein. Legen Sie möglichst fest, wieviele Variablen gebraucht werden. Viele Variablen werden lediglich lokal benötigt, so daß diese an anderen Stellen wieder zur Verfügung stehen können. Beachten Sie auch, daß Komfort Platz verbraucht. Mehrstellige Variablen belegen auch mehrere Speicherstellen. In der Regel reicht das Alphabet für die benötigten Variablen aus. Achten Sie bei Ihrer Planung auch darauf, ob in DATA-Zeilen sich ständig wiederholende Zahlenketten vorkommen. Die Erfahrung zeigt insbesondere, daß ein großer Teil der DATA-Zeilen aus Nullen besteht, die dann mit in dem Programm eingelesen werden. Es ist möglich, sich die Vielzahl der Nullen zu sparen, indem in der Leseroutine nur bis zu dem Beginn der Nullenreihe gelesen wird, danach in einer Programmschleife die Nullen in die Variablen eingelesen werden und dann das Lesen aus den DATA-Zeilen fortgesetzt wird. Dabei kann auch mit einbezogen werden, daß mit Hilfe von RESTORE DATA-Zeilen mehrfach gelesen werden können, so daß es sich erübrigt, gleiche DATA-Zeilen mehrfach zu schreiben. All dies benötigt eine intensive Vorbereitung des Programms.

Wenn Sie diese Hinweise beachten, werden Sie feststellen, daß Ihre Programme weniger Platz brauchen und in der Regel auch schneller laufen, allerdings auch unübersichtlich sind. Deshalb sollten Sie ein so geschriebenes Programm intensiv dokumentieren, um jederzeit nachvollziehen zu können, wie das Programm entwikkelt wurde. Es erweist sich in der Regel als schwierig, nach einer längeren Zeit ein solches Programm zu verändern. Deshalb sollten Sie diese Methode nur anwenden, wenn Sie lange Programme schreiben oder es auf die Ausführungszeit ankommt. Ansonsten empfiehlt sich eine strukturierte Programmierung, die Wert auf große Übersichtlichkeit und gute Dokumentation des Programms in REM-Zeilen legt.

2.6.2 Strukturiertes Programmieren

Sicherlich haben Sie schon einmal eine Idee für ein neues Programm gehabt und wollten dieses dann verwirklichen. Sie setzten sich dann sofort an die Maschine und begannen, das Programm einzutippen. Bei den ersten Zeilen ging alles noch ganz gut: sie konnten immer alle Zeilen auf dem Bildschirm gleichzeitig sehen

und so das Programm kontrollieren. Mit zunehmender Länge wurde es aber immer unübersichtlicher.

Wahrscheinlich haben Sie auch zwischen einige Zeilen neue eingefügt, bis schließlich die Zahlen zwischen den einzelnen Zeilen vollständig ausgefüllt waren. Mit der RENUMBER-Funktion konnten Sie sich glücklicherweise über die Runden retten. Mit Sicherheit haben Sie bald Schwierigkeiten bekommen, das Programm fehlerfrei zum Laufen zu bringen und es ist anzunehmen, daß Sie bald keine Lust mehr am Programmieren verspürt haben.

Dies alles können und sollten Sie sich ersparen, indem Sie Ihr Programm von vornherein planen. Man benutzt dazu die Methode der strukturierten Programmierung. Diese wollen wir uns in diesem Kapitel etwas näher betrachten, und Sie werden sehen, wenn Sie diese Methode anwenden, werden Ihre Programme übersichtlicher, die Programmierzeiten werden kürzer und Ihre Lust am Programmieren wird wieder steigen. Außerdem nimmt die Fehlerwahrscheinlichkeit ab, weil Fehler besser eingegrenzt werden können. Beschäftigen wir uns nun etwas näher mit dieser Methode.

2.6.2.1 Die Programmidee

Das Wichtigste bei einem Programm ist die Programmidee. Viele Computerbesitzer sind ständig auf der Suche nach neuen Ideen, die sie in ein Programm umsetzen können. Sie werden sicherlich auch schon die eine oder andere Idee gehabt haben.

Wenn Sie jetzt eine neue Idee haben, sollten Sie sich nicht sofort an die Maschine setzen und drauflosprogrammieren (denn dies ist eigentlich kein Programmieren, sondern lediglich Codieren), sondern zunächst einmal auf einem Zettel die Programmidee in all ihren wichtigen Punkten festhalten. Nachdem Sie dies gemacht haben, können Sie sicher sein, daß Ihnen Ihre Idee so schnell nicht wieder verlorengeht, es sei denn, Sie verlieren den Zettel.

Wenn Sie sich dann später mit dieser Idee erneut beschäftigen wollen, sollten Sie sich auch nicht sofort an die Maschine setzen und drauflostippen, sondern zunächst einmal an Ihrem Schreibtisch exakt definieren, was Ihr Programm können soll.

Wenn Sie sich erinnern, wie wir zum Beispiel unser Adreßprogramm entwickelt haben, werden Sie feststellen, daß wir zunächst auch einmal gefragt haben: Was soll unser Programm alles können. Wir hatten uns damals dafür entschieden, daß dieses Programm Daten speichern, einlesen und sortieren sollte. Außerdem mußten die Daten natürlich eingegeben werden. Weiterhin sollte nach den Daten gesucht und sie ausgedruckt werden. Mit dieser Definition war eigentlich die Aufgabenstellung schon vollkommen klar.

Häufig stellt man fest, daß die Aufgabenstellung eines Programms bereits in einem anderen Programm, das früher schon einmal angefertigt wurde, gelöst ist und man sollte dann ruhig den Mut haben, seine Routine von früher noch einmal anzuschauen und daraufhin zu überprüfen, ob man sie hier gebrauchen kann. Meist ist dies der Fall, so daß die Routine entweder mit keinen oder nur mit leichten Änderungen übernommen werden kann. Dies erspart natürlich wertvolle Programmierzeit.

Nachdem jetzt die Problemstellung klar und exakt umrissen ist, müssen wir weiter überlegen. Bei unserem Programm sollte eine Datei verwendet werden. Wir mußten zunächst einmal festlegen, welche Art von Datei verwendet werden sollte. Für uns kam hier nur die sequentielle Datei in Frage. Diese Frage war also leicht zu beantworten. Steht zum Beispiel ein Floppy-Laufwerk zur Verfügung, ist diese Frage nicht so leicht zu beantworten. Man sollte sich dann schon einige Gedanken darüber machen, welche Dateiart verwendet werden soll. Viel wichtiger ist die Festlegung der einzelnen Datensätze. Wir hatten uns dafür entschieden, daß jeder Datensatz aus 6 Datenfeldern bestehen sollte.

Nachdem diese Dinge alle festgelegt sind, haben Sie eigentlich schon die Vorarbeit zum Programmieren weitgehend erledigt.

Wollen Sie ein Programm entwickeln wollen, das keine Datei enthält (zum Beispiel ein Spiel), muß eine Datei natürlich nicht festgelegt werden. Aber auch hier sind sehr oft Variablen zu benutzen, zum Beispiel Punktzahlen, die auch vorgeplant werden müssen. In unserem Adressenprogramm waren auch einige Variablen verwendet worden, die nicht in einer Datei aufgenommen werden sollten. So mußte für die Datensatznummer eine Variable eingeführt werden und eine zusätzliche für die Datenfeldnummer. Wir haben dafür die Variablen I und J gewühlt und diese im gesamten Programm dafür benutzt. Der Vorteil davon liegt klar auf der Hand: Wir wissen zu jedem Zeitpunkt im Programm, welche Bedeutung die

Variable I und welche Bedeutung die Variable J hat. Auch diese Dinge mußten von vornherein festgelegt werden.

In diesem Beispiel mag es noch sehr einfach zu überblicken sein, welche Variablen im gesamten Programm häufiger benutzt werden. Bei manchen Programmen mag dies nicht so ganz einfach möglich sein, aber auch hier hilft eine gute Vorarbeit, spätere Überraschungen zu vermeiden. Werden Variablen nur einmal benutzt, sollte man auch von vornherein eine Reihe von Variablen vorsehen, die dafür zur Verfügung stehen können. Man sollte diese Variablen dann immer wieder benutzen, um Speicherplatz zu sparen.

Als sinnvoll erweist es sich, von vornherein eine Variablenliste anzulegen und dort immer einzutragen, welche Variable wo benutzt werden soll. Haben Sie diese Liste vor Augen, wird es viel einfacher, Variablen zu dimensionieren, wenn Sie sich später beim Programmieren wirklich an die Maschine setzen.

Wir haben jetzt also festgelegt, was unser Programm können soll, welche Variablen verwendet werden sollen und wie unsere Datei aussehen soll. Nun können wir uns damit beschäftigen, wie unser Programm aufgebaut sein soll.

2.6.2.2 Die Programmablaufplanung

Nach unseren bisherigen Vorarbeiten müßte es eigentlich leicht sein, den Programmablauf exakt planen zu können. Da wir bei unserer Programmidee schon genau definiert haben, was unser Programm alles können soll, wissen wir schon mehr über das Programm, als wir zunächst angenommen haben. Wir müssen jetzt entscheiden, wie das Programm gesteuert werden soll. Wir haben uns bei der Planung des Adressenprogramms für eine Menüsteuerung entschieden. Wir hatten in diesem Abschnitt auch erwähnt, daß die Menüsteuerung auf der Idee beruht, daß mehrere Programmteile sich gegenseitig aufrufen. Hier haben wir ein Hauptprogramm (Menüprogramm), das die Unterprogramme aufruft. Wir wissen jetzt schon, daß unser Programm aus mehreren Unterprogrammen bestehen muß. Jedes dieser Einzelprogramme müssen wir jetzt noch etwas genauer planen.

Professionelle Programmierer verwenden zur Planung des Programmablaufs eine grafische Symbolsprache. Diese Symbolsprache wird

Flußdiagramm genannt. In diesem Flußdiagramm gibt es für jede Art von Problemstellung innerhalb eines Programms ein eigenes Symbol. Eine Aufstellung der wichtigsten Symbole finden Sie in Tabelle 2.4. Betrachten wir die Symbole etwas näher.

Mit dem ersten Symbol werden Anfang und Ende eines Programms gekennzeichnet. Man kann damit sofort in einem Flußdiagramm erkennen, wo ein Programm beginnt bzw. wo mit dem "Lesen" des Flußdiagramms begonnen werden soll. Mit dem zweiten Zeichen werden alle Tätigkeiten, die der Computer durchführen soll, gekennzeichnet. Diese Tätigkeiten können vielseitiger Natur sein und im Prinzip alle Aufgabenstellungen umfassen, die in einem Programm vom Computer selbst durchgeführt werden sollen.

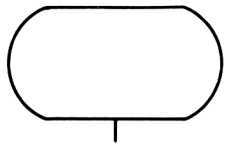
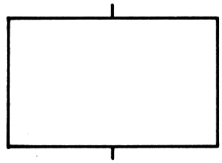
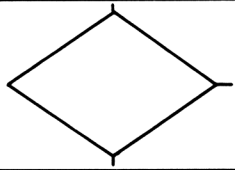
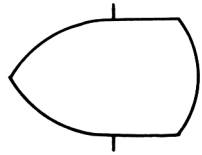
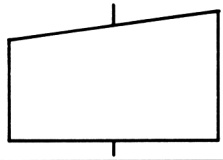
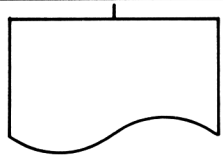
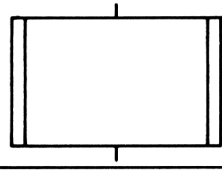
Das nächste Symbol gibt uns an, ob ein Programm verzweigt werden soll oder nicht. An dieser Stelle wird jeweils eine Bedingung daraufhin überprüft, ob sie zutrifft oder nicht. Bei Zutreffen soll an einer bestimmten Stelle weitergearbeitet werden, bei Nichtzutreffen an einer anderen Stelle. Bei diesem Symbol ist es unwichtig, ob die Überprüfung der Bedingung vom Mensch oder vom Computer vorgenommen wird.

Mit dem vierten Symbol soll eine Ausgabe auf dem Bildschirm angezeigt werden. Allerdings muß hier ergänzt werden, daß auch eine akustische Ausgabe (zum Beispiel Erklingen eines Tons) mit diesem Symbol gekennzeichnet wird.

Das fünfte Zeichen deutet an, daß an dieser Programmstelle eine Eingabe über die Tastatur erfolgen soll. Da in einem Programm oftmals Eingaben über die Tastatur erfolgen, wird dieses Symbol auch sehr oft in einem Flußdiagramm auftauchen.

Das Symbol für die Druckerausgabe versteht sich eigentlich von selbst. Beim Zeichen für den Unterprogrammaufruf ist sofort einzusehen, daß dieses Zeichen in einem Flußdiagramm nur sinnvoll ist, wenn dieses Unterprogramm separat beschrieben wird. Wird ein kompletter Programmablaufplan in einem Flußdiagramm eingezeichnet, in dem alle Unterprogramme enthalten sind, kann natürlich dieses Zeichen nicht in einem Flußdiagramm vorkommen. Meist ist aber die Technik so, daß, wie zum Beispiel bei dem Menüprogramm, das wir im Moment beschreiben, jedes Unterprogramm ein Flußdiagramm für sich erhält und im Hauptprogramm dieses Symbol für den Aufruf des entsprechenden Unterprogramms eingezeichnet ist.

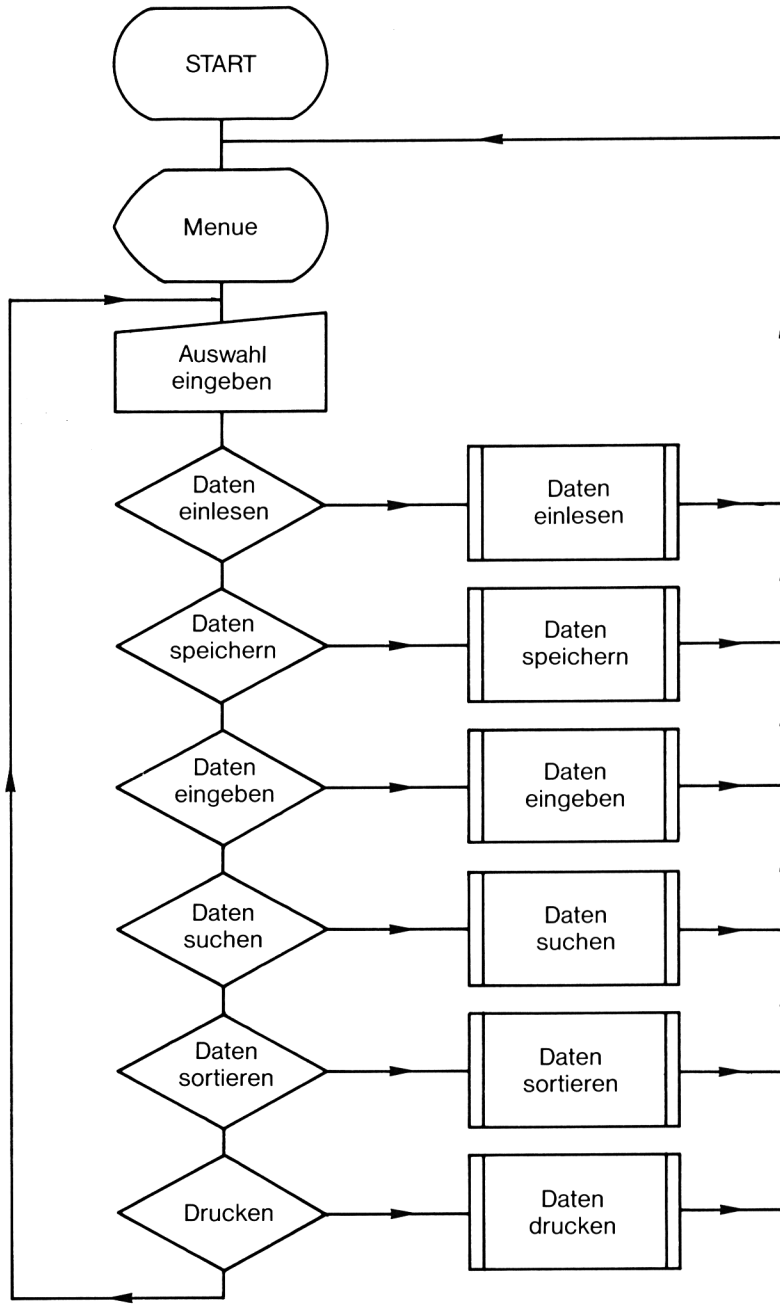
Tabelle 2.4 Die wichtigsten Flußdiagrammsymbole

Symbol	Erklärung
	Symbol für Programmstart, -ende und Programmein- und -aussprünge
	Symbol für die Ausführung von Tätigkeiten (z.B. öffnen einer Datei)
	Zeichen für eine Verzweigung, die von einer Bedingung (Ja/Nein) abhängt.
	Symbol für Bildschirmausgabe
	Zeichen für Tastatureingabe
	Symbol für Druckerausgabe
	Symbol für Unterprogrammausführung

Jetzt wollen wir für unser Aufrufen des Menüprogramms einmal ein solches Flußdiagramm zeichnen. Bei diesem Flußdiagramm soll das Zeichen für Unterprogrammaufruf für jedes Unterprogramm verwendet werden.

In diesem Flußdiagramm sehen wir sofort unsere 6 Unterprogramme, die wir in diesem Programm installieren wollten bzw. installiert haben. Nach dem Start erfolgt die Ausgabe eines Menüs, danach wird die Auswahl eingegeben. Bei unserem Programm genügt es, hierzu eine Ziffer einzugeben. In den folgenden Rauten wird dann überprüft, ob eine Bedingung erfüllt ist oder nicht oder ob weitergegangen oder ins Unterprogramm gesprungen werden soll. Die Bedingung, die hier zum Beispiel mit Dateneinlesen bezeichnet wird, wird in dem Programm dann nachher nur noch die Überprüfung einer Ziffer bedeuten. Nach Beendigung der Unterprogramme soll ins Menü zurückgesprungen werden. Deshalb sind die Pfeile auf der rechten Seite der Unterprogrammaufrufe angebracht, mit denen dann vor die Menüausgabe zurückgegangen wird. Sind alle Auswahlmöglichkeiten durchlaufen, soll die Tastatureingabe nochmal abgefragt werden. Diesen Zweck erfüllt der Pfeil auf der linken Seite.

Flußdiagramm für Hauptprogramm



Mit einem solchen Flußdiagramm (sei es auf einem Blatt Papier, oder sei es auch nur theoretisch im Kopf) fällt es verständlicherweise viel leichter, ein Menüprogramm zu schreiben. Die Aufgabe, die sich jetzt lediglich stellt, ist, jeden einzelnen Punkt dieses Flußdiagramms in den dazugehörigen Programmteil zu übersetzen. Auch hierzu sind wieder einige Vorüberlegungen notwendig, die die Arbeit erleichtern.

Da in einem Programm öfter eine Bildschirmausgabe erforderlich ist, sollte zunächst einmal festgelegt werden, wie eine Bildschirmmaske aufgebaut sein soll. Wir hatten uns bei unserem Adressenprogramm (das hier als Beispiel dienen soll) dazu entschlossen, die Drei-WINDOW-Technik anzuwenden. Diese Technik wird an anderer Stelle in diesem Buch ausführlich beschrieben. Damit war unser Bildaufbau (die Bildschirmmaske) hinreichend beschrieben, und wir konnten uns an die Programmierung setzen.

Beim Programmieren achten wir jetzt darauf, daß wir vor jeden Programmteil, den wir schreiben, eine REM-Zeile setzen, die uns angibt, welche Funktion der folgende Programmteil erfüllen soll. Das erleichtert das spätere Wiederauffinden dieses Programmteils. Ebenso achten wir darauf, daß wir unsere Zeilen nicht mit BASIC-Befehlen überfüllen sondern entweder nur einen Befehl in jeder Zeile oder logisch zusammengehörende Befehle verwenden. Als Beispiel für solche logisch zusammenhängende Befehle wäre eine Zeitschleife zu nennen. Wir können durchaus in einer Zeile schreiben:

```
FOR I = 1 TO 1000: NEXT
```

Diese Befehle gehören logisch zusammen, da mit dieser Befehlszeile nur ein Ziel erreicht werden soll, nämlich eine Warteschleife zu bilden. Hier kann es sinnvoll sein, diese Befehle in eine Zeile zu setzen. Bei Befehlen, die inhaltlich nichts direkt miteinander zu tun haben, ist dies nicht unbedingt sinnvoll, da die Übersichtlichkeit des Programmnamens darunter stark leidet.

Wir haben es deshalb in unserem Menü auch vermieden, die BASIC-Zeilen vollzuschreiben sondern haben versucht, nur zusammengehörende Teile in jeder Zeile vorkommen zu lassen. Wenn Sie sich des Programm noch einmal anschauen, werden Sie feststellen, daß zum Beispiel in einer Zeile alle WINDOWS definiert oder in einer anderen Zeile die Farbregister alle gesetzt wurden. Darüber hinaus finden Sie in einer Zeile einen LOCATE- und einen PRINT-Befehl. Beide Befehle gehören logisch zusammen, da der LOCATE-Be-

fehl lediglich ein Vorbefehl zum PRINT-Befehl ist, weil er angibt, an welcher Stelle die Ausgabe erfolgen soll. Da in dem Menü des Programms bei der Bildschirmausgabe jede einzelne Zeile einen bestimmten Anfangspunkt haben muß, damit das Bild akzeptabel aussieht, gehört der LOCATE-Befehl somit zum entsprechenden PRINT-Befehl. Insofern war es sinnvoll, beide Befehle in einer Zeile zusammenzufassen.

Wir müssen allerdings ehrlich sein und feststellen, daß dieses Programm nicht bis in die letzte Möglichkeit hinein strukturiert programmiert ist. Wir sind mit einem strukturierten Ansatz an dieses Programm herangegangen, haben das Problem von vornherein definiert und das Programm nach der Problemstellung exakt und sinnvoll aufgbaut. Außerdem ist das Programm auch ziemlich übersichtlich und in seinem modularen Aufbau jederzeit erweiterbar. Diese Funktion sollte jede strukturierte Programmierung erfüllen.

Man könnte selbstverständlich, wie alles im Leben, auch dieses Programm in seiner Struktur noch verbessern. Oft ist es so, daß ein logischer Programmabschnitt über mehrere Zeilen hinweg programmiert wird. Dies ist in den meisten FOR-NEXT-Schleifen der Fall. Jetzt wäre es natürlich schön, sofort zu sehen, an welcher Stelle ein solcher logischer Programmteil beginnt und wo dieser Programmteil aufhört. Dazu verwendet man den Trick des Einrückens von Befehlen. So wird dann der Programmteil in der ersten Spalte begonnen, und das Ende liegt ebenfalls dort. Bei dem Beispiel einer FOR-NEXT-Schleife würde der FOR-TO-Befehl und der NEXT-Befehl in der gleichen Spalte erscheinen. Die dazwischenliegenden Befehle wären jetzt eingerückt, so daß man sofort sehen könnte, wo der Programmteil beginnt und wo er endet. Dies könnte zum Beispiel folgendermaßen aussehen:

```
10 FOR I = 1 TO 20
20   PRINT A(I)
30   PRINT B(I)
40 NEXT I
```

Hier sieht man sofort, wo die Routine beginnt und wo die Routine endet. Selbstverständlich können auch mehrere solcher Routinen ineinander eingebaut werden, dann muß lediglich bei jeder weiteren Unterroutine der Befehl weiter eingerückt werden. So wäre es möglich, daß anstelle des PRINT-Befehls ein neuer FOR-Befehl stehen könnte, und der PRINT-Befehl dann in dieser Schleife erfolgen sollte. Dann müßte der PRINT-Befehl natürlich um mindestens eine

Stelle weiter eingerückt werden und der NEXT-Befehl wieder an der Stelle stehen, wo jetzt der PRINT-Befehl steht.

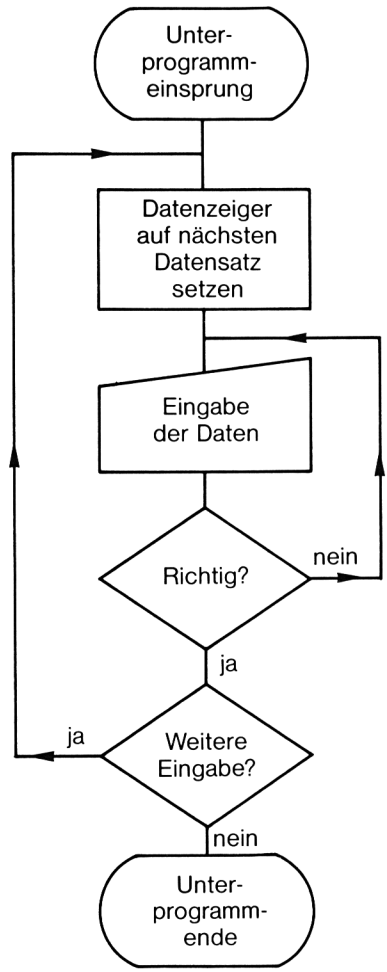
Diese Programmierart ist recht umständlich und zu aufwendig. In der Regel reicht es aus, wenn ein Programm in der Art programmiert wird, wie wir es oben vorgeführt haben. Wenn Sie diese Hinweise, die wir hier gegeben haben, befolgen, werden Sie ein übersichtliches Programm entwickeln können. Wenn Ihr Programm später einmal von einem anderen Programmierer verändert werden soll, wäre es besser, wenn Sie das Programm auch noch durch Einrücken der Befehle strukturieren würden. Das würde jedem anderen Programmierer die Arbeit erleichtern. Er wäre dann in der Lage, das Programm besser zu überblicken und zu sehen, wo welche Routine beginnt und wo sie aufhört.

Sie werden jetzt sicherlich sagen, dies ist nicht der Fall, mein Programm soll von keinem anderen Programmierer nachvollzogen werden können. Aber so einfach sollte man das nicht formulieren. Oft werden Programme auch für andere Anwender geschrieben, und da können Sie nie sicher sein, ob diese Anwender nicht das Programm von einem weiteren Programmierer überarbeiten lassen. Außerdem können Sie auch nie sicher sein, ob Sie noch in 2 oder 3 Jahren, wenn Sie noch einmal versuchen, dieses Programm nachzuvollziehen und zu überarbeiten, noch genau verstehen, was Sie damals gemacht haben.

In jedem Fall sollten Sie das Programm mit so vielen REM-Zeilen versehen, daß Sie jederzeit sehen können, an welcher Stelle des Programme welche Problemlösung steht. Dies ist auch unter Umständen im Hinblick auf spätere, andere Programme wichtig, da es möglich ist, daß Sie eine Routine, die Sie in diesem Programm entwickelt haben, in einem anderen Programm ebenfalls einsetzen können.

Also versuchen Sie es einmal mit der strukturierten Programmierung und achten Sie darauf, daß Sie auch vorher das Flußdiagramm Ihres Programms zumindest im Kopf haben. Zur Übung sollten Sie einmal die Flußdiagramme von allen Unterprogrammen zu unserem Adressenprogramm entwickeln. Als kleine Hilfestellung dazu zeigen wir Ihnen hier noch einmal ein Flußdiagramm für eines der Unterprogramme. Die anderen Flußdiagramme sollten Sie dann selbst entwickeln können. Zeichnen Sie zur Übung alle noch fehlenden Flußdiagramme.

Flußdiagramm: Unterprogramm Daten eingeben



2.6.3 Fehler und Unterbrechungsbehandlung mit dem CPC

Daß der CPC 464 über eine Reihe von Kommandos verfügt, die bei normalen BASIC-Computern nicht üblich sind, haben wir schon mehrmals erfahren. In diesen Bereich gehören auch die Kommandos

ON ERROR GOTO

und

ON BREAK GOSUB

Sie dienen zur Fehler- und Unterbrechungsbehandlung. Normalerweise führt das Auftreten eines Fehlers dazu, daß das Programm mit der entsprechenden Fehlermeldung abgebrochen wird. Dies ist häufig nicht im Sinne des Benutzers, da eine Reihe häufig auftretender Fehler vom Computer theoretisch selbst behoben werden könnte. Ein einfaches Beispiel dafür ist eine Division durch Null bei der Darstellung einer Funktion auf dem Bildschirm. Hier besteht eigentlich kein Grund, warum der CPC 464 anhalten, und seine schön gemalte Grafik durch ein

DIVISION BY ZERO ERROR

verunstalten sollte. Es wäre durchaus möglich, diesen Funktionswert einfach zu überspringen, um mit dem nächsten fortzufahren, ohne daß die Grafik an Aussagegehalt verlieren würde. Als ein weiteres Beispiel bietet sich das Laden von Dateien an. Hat man hier den Index der zu ladenden Variablen zu groß werden lassen, stehen dem Computer in der gerade eingelesenen Datei vom Band nicht mehr genügend Variablen zur Verfügung. Er meldet sich dann mit einem unfreundlichen

EOF met

Mit dem ON ERROR GOTO-Befehl können wir derartige Probleme beheben. Er bietet uns die Möglichkeit, beim Auftreten eines Fehlers ein ganz bestimmtes Unterprogramm anzuspringen. In dieser muß dann die Behandlung des entsprechenden Fehlers erfolgen, wonach dann eine Rückkehr zur normalen Arbeit möglich wird. Im ersten Beispiel wäre hier zum Beispiel zu prüfen, ob eine Division durch Null stattgefunden hat und der entsprechende Wert wäre dann zu überspringen. Im zweiten Fall sollte aus der entsprechenden

Schleife ausgebrochen werden. Dies könnten wir am einfachsten dadurch erreichen, daß wir die entsprechende Zählvariable über das Schleifenmaximum hinaussetzen. Haben wir zum Beispiel in der Schleife

```
FOR I=1 TO 40
```

beim Stand von I=35 ein EOF bekommen (dies stellt die Abkürzung für END OF FILE oder Ende des Datenfelds dar), müssen wir nun einfach I gleich 42 setzen. Damit wird unsere Schleife nicht weiter abgearbeitet, sondern beendet, und wir brauchen uns um den weiteren Fortgang des Programms keine Sorgen zu machen, solange wir nicht auf die restlichen fünf Daten angewiesen sind. Eine andere Möglichkeit könnte darin bestehen, einen entsprechenden Erklärungstext auszugeben, der auf den Fehler hinweist oder den Benutzer mit einer Eingabe aufzufordern, auf diesen Fehler zu reagieren. All diese Möglichkeiten sind sehr viel komfortabler als der normale Programmabbruch. Wir wollen im folgenden zeigen, wie wir diese Möglichkeiten miteinander kombinieren können, um deutsche Fehlermeldungen und eine entsprechende Eingriffsmöglichkeit in unser Programm zu erhalten.

2.6.3.1 Deutsche Fehlermeldungen auf dem CPC

Bevor wir uns mit einem Programm, das uns deutsche Fehlermeldungen auf den Bildschirm bringt, etwas näher beschäftigen, müssen wir noch auf das ERROR-Kommando eingehen.

Wir haben schon gesagt, daß wir mit

```
ON ERROR GOTO
```

in unsere Fehlerbehandlungsroutine springen können. Das Ende unserer Fehlerbehandlungsroutine bildet der Befehl

```
RESUME
```

Nachdem der Computer diesen erreicht hat, macht er an der Stelle weiter, an der der Fehler aufgetreten ist. Wir können in den Zeilen 1 oder 5

```
ON ERROR GOTO 60000
```

schreiben, und nachdem wir ab Zeile 60000 unsere Fehlerbehandlungsroutine ausgeführt haben, springt der CPC nun nicht nach Zeile 5 zurück, sondern zum Beispiel nach Zeile 100, wo der Fehler aufgetreten ist. Wir brauchen uns um den Rücksprung zur richtigen Zeile und zum richtigen Befehl nicht zu kümmern. Sollten wir in unserer Routine die Nummer der Zeile, in der der Fehler aufgetreten ist, benötigen, so können wir diese mit der Variable ERL jederzeit abfragen.

```
PRINT ERL
```

gibt uns also beim Auftreten eines Fehlers die entsprechende Fehlerzeile an, ansonsten ist diese Variable Null. Jeder Fehler hat einen Code, die sogenannte Fehlernummer. Sie beginnt mit eins für das Auftreten eines NEXT-Kommandos ohne vorheriges FOR und endet mit der Nummer dreißig für ein WEND, vor dem kein entsprechendes WHILE eingegeben wurde. Die Variable ERR enthält beim Auftreten eines Fehlers die entsprechende Fehlernummer, sonst ist auch sie gleich Null gesetzt. Mit diesem Vorwissen sind wir nun gut gerüstet, um unser Fehlerbehandlungsprogramm zu konstruieren. Wir haben es mit den Zeilennummern 60010 bis 60360 ganz oben im Speicher abgelegt. Der Abschluß wird durch einen RESUME-Befehl gebildet.

```
10 ON ERROR GOTO 60010
20 FOR i= 1 TO 40
59000 END
59400 REM ******************
59500 REM ** UP:DT.FEHLER **
59600 REM ******************
60010 DATA "NEXT ohne FOR","Syntax-Fehler"
60020 DATA "RETURN ohne Ruecksprungadresse"
60030 DATA "keine DATA mehr","falsches Argument"
60040 DATA Akkumulator voll,Speicher voll
60050 DATA Zeile existiert nicht
60060 DATA Indexbereich ueberschritten
60070 DATA ARRAY schon dimensioniert
60080 DATA Division durch Null
60090 DATA unzulaessiges Direktkommando
60100 DATA Variablentypen verschieden
60110 DATA Stringspeicher voll
60120 DATA String zu lang
60130 DATA Stringausdruck zu komplex
60140 DATA CONT nicht moeglich
60150 DATA unbekannte Benutzerfunktion
60160 DATA RESUME fehlt
60170 DATA RESUME ohne ON ERROR
60180 DATA Programmzeile ohne Nummer
60190 DATA fehlender Operant
60200 DATA Zeile zu lang
60210 DATA Fileende ueberschritten
60220 DATA NEXT fehlt
60230 DATA File schon offen
60240 DATA Filetyp falsch
60250 DATA unbekanntes Kommando
60260 DATA WEND fehlt
60270 DATA WEND ohne WHILE
60280 DIM ER$(30)
60290 FOR i=1 TO 30:READ ER$(i):NEXT i
60300 PRINT ER$(ERR);" in Zeile";ERL
60310 PRINT"aendern          (1)"
60320 PRINT"weiterer Versuch (2)"
60330 z$=INKEY$:IF VAL(z$)<1 OR VAL(z$)>2 THEN 60330
60340 IF VAL(z$)=2 THEN ERASE ER$:RESTORE 60010:RESUME
60350 STOP
60360 RESUME
```

Zu Beginn unseres Unterprogramms werden die deutschen Fehlermeldungen in das vorher dimensionierte Fehlerfeld ER$ eingelesen. Danach ist es ein leichtes, in Abhängigkeit von der Fehlernummer ERR und der Fehlerzeile ERL einen deutschen Fehlertext in Zeile 60300 auszugeben. Nun wird der Benutzer gefragt, ob eine Änderung erfolgen soll. Ist dem so, so läuft das Programm auf ein STOP-Kommando, die Änderung kann erfolgen, und durch ein

```
CONT
```

ist es dann meist möglich, den Programmablauf fortzusetzen. Im anderen Falle wird noch einmal probiert, die entsprechende Operation auszuführen. Dieser letzte Teil ist natürlich nur optional, das heißt er kann durch andere Routinen ersetzt werden. Hier besteht zum Beispiel die Möglichkeit, sich durch

```
LIST ERL-30,ERL+30
```

in der Umgegend der Fehlerzeile sechs Zeilen auf den Bildschirm bringen zu lassen, um eine ziemlich gute Übersicht über den Ort des Fehlers zu erhalten. Das Operieren mit derartigen LIST-Routinen ist besonders bei langwierigen Programmdurchläufen in der Entwicklung von Dateiprogrammen äußerst zweckmäßig, um versteckten Fehlern auf die Spur zu kommen. Um die Wirkungsweise des Programms zu verdeutlichen, haben wir die fehlerhafte Zeile 30 eingefügt. Da hier zu der eröffneten FOR-TO-Schleife das entsprechende NEXT-Kommando fehlt, gibt der Computer den Fehlertext NEXT FEHLT in Zeile 30 aus. Geben wir in unserem Beispiel eine zwei ein, das heißt weiterer Versuch, wird die Zeile 30 immer wieder angesprungen, und wir erhalten denselben Fehlertext. Dies ist auch selbstverständlich, da wir den Fehler nicht behoben haben. Ein ON ERROR-Kommando kann auch mehrmals zum Sprung in eine Fehlerbehandlungsroutine benutzt werden. Dann ist es zum Beispiel möglich, in einem fehlerhaften Programm auch mehrere abhängige oder unabhängige Fehler nacheinander durch derartige Routinen behandeln zu lassen.

Etwas anders liegt der Sachverhalt mit dem ON BREAK GOSUB-Kommando. Dieses wirkt nur einmal. Normalerweise unterbricht der CPC beim Drücken auf die ESCAPE-Taste den Programmablauf. Es kann wünschenswert sein, stattdessen eine Anzahl von Zeilen zu listen, oder eine Änderung im Programm vorzunehmen. Ein typisches Anwendungsgebiet stellt auch hier die Programmentwicklung dar. Testet man zum Beispiel den Bildaufbau und arbeitet dabei mit dem LOCA-

TE-Kommando, kommt es oft vor, daß eine Ausgabe mehrmals durchlaufen werden muß, um die optimale Position festzustellen. Nachdem der auszugebende Text an der falschen Stelle auf den Bildschirm gebracht wurde, sollte die zu ändernde Zeile erscheinen, wenn wir BREAK eingeben und nicht erst duch ein nochmaliges LIST dazu erst aufgefordert werden müssen. Wir können dies relativ einfach erreichen, indem wir uns ein kleines Unterprogramm, zum Beispiel ab Zeile 10000 schaffen, das nur den Befehl

```
LIST Zeilennummer
```

oder

```
LIST Zeilennummer -30, Zeilennummer +30
```

und den END-Befehl enthält. Geben wir dann zum Beispiel in Zeile 1

```
ON BREAK GOSUB 10000
```

ein, erfolgt auf Drücken der ESCAPE-Taste der Sprung zu diesem Unterprogramm, das uns dann die fraglichen Zeilen listet. Ein anderer Anwendungszweck dieses Kommandos liegt darin, einen fehlerhaften, das heißt unbeabsichtigten Programmabbruch zu verhindern. In unserem Unterprogramm ab 10000 müßten wir dann fragen, ob das Programm wirklich abgebrochen werden soll. Falls nein, müßte dann mit einem einfachen RETURN zum Hauptprogramm zurückgekehrt werden. Falls ja, wäre das Programm mit END abzubrechen. Im Gegensatz zum ON ERROR-Befehl kann der Sprung zu dem Unterprogramm auf ON BREAK hin aus verständlichen Gründen nur ein einziges Mal erfolgen. Würden wir hier, wie bei dem ON ERROR-Kommando, einen fortgesetzten Sprung zu dem entsprechenden Unterprogramm erlauben, bestünde keine Möglichkeit mehr, den Rechner durch Drücken auf die ESCAPE-Taste zum Programmabbruch zu zwingen. Das dadurch notwendige Ausschalten und Wiedereinschalten des Rechners würde natürlich unsere Programme und etwaige Dateien zerstören.

Nachdem wir uns mit den Möglichkeiten des CPC bei der Daten- und Textverarbeitung beschäftigt haben und am Schluß noch Stellung zu einigen Spezialbefehlen genommen haben, wollen wir nun etwas näher auf die systemspezifischen Eigenschaften des CPC 464 eingehen. Als ersten Bereich wenden wir uns dabei der Grafik zu.

3

Grafik

3 Die Grafik

3.1 Einleitung

Grafik ist eine der Stärken des CPC 464, vielleicht sogar seine stärkste Seite. Die Möglichkeiten sind vielfältig, teilweise sogar fantastisch, besonders dann, wenn man die Maschinenebene mit einbezieht.

Schon das "normale" BASIC des CPC, das jedem Benutzer problemlos zugänglich ist, bietet eine Fülle von Möglichkeiten, die sich mit jedem Vergleichsmodell messen können. Bleiben wir deshalb zuerst noch ein wenig in diesem Bereich. Die Beschäftigung mit der Grafik kann man im wesentlichen in zwei Teilbereiche unterteilen. Der eine umfaßt die Beschäftigung mit den grafischen Zeichen, dem Zeichensatz und den Symbolen. Auf der anderen Seite stehen die Darstellungsmöglichkeiten der hochauflösenden Grafik.

Ausgehend von dieser Unterteilung werden wir uns nach einer kurzen Einleitung zuerst mit den Möglichkeiten des CPC bei der Textdarstelllung beschäftigen. Daran anschließend gehen wir in den Bereich der hochauflösenden Grafik und werden uns mit seinen Fähigkeiten auf dem Feld der Erstellung von Grafiken, der grafischen Statistik und dem joystickgesteuerten Zeichnen etwas näher befassen. Wir werden uns mit den gebräuchlichen Darstellungsmitteln der grafischen Statistik, wie Balken-, Stab- und Kreisdiagrammen auseinandersetzen und Realisierungsmöglichkeiten für diese auf dem Schneider-Computer suchen. Mit der hochauflösenden Grafik ist man nicht auf die grafische Darstellung beschränkt. Mit der Grafik ist es auch möglich, individuell zu malen und zu zeichnen. Wir gehen hier auf die Grundzüge ein und geben einige Anregungen weiter. Die Entwicklung der Grundzüge eines Zeichenprogramms auf dem CPC bildet den Abschluß dieses Kapitels.

3.2 Grundlagen der Grafik

Bei der Beschäftigung mit der Grafik kann man zwei Wege gehen. Man fängt ganz groß an - bei Textseiten und ganzen Texten - oder man fängt ganz klein an und beschäftigt sich mit dem Stoff, aus dem die Grafik aufgebaut ist, dem Pixel. Daran schließt sich gleich die erste Frage an:

Was ist ein Pixel?

Als Pixel bezeichnet man die kleinste vom Computer darstellbare grafische Einheit; oder um es etwas konkreter zu sagen: einen Bildpunkt auf dem Schirm des Monitors. Insgesamt kann der CPC 464 128000 solcher Bildpunkte auf dem Bildschirm darstellen. Diese sind in 200 Reihen zu je 640 Punkten nebeneinander angeordnet. Jedes Zeichen ist aus Pixeln aufgebaut, aber auch die hochauflösende Grafik greift auf sie zurück. Jedes Zeichen ist dabei in einer Matrix aus Pixeln definiert. Diese Matrix hat immer dieselbe Höhe: 8 Pixel. Die Anzahl der Pixel in der Waagrechten hängt vom eingeschalteten MODE ab.

Im hochauflösenden 80-Zeichen-MODE, dem MODE 2, wird mit einer Breite von 4 Pixeln je Zeichen gearbeitet. Im Standardmodus, dem MODE 1, wird die Anzahl auf 8 erweitert, so daß sich Breite und Höhe des Zeichens entsprechen. In MODE 0 wird schließlich mit 16 Pixeln in der Breite gearbeitet. Diese Vergrößerung erreicht man dadurch, daß in den niedrigeren MODES zwei, beziehungsweise vier nebeneinanderliegende Pixels gleichgesetzt, beziehungsweise rückgesetzt werden. Dafür ist eine Routine im Betriebssystem verantwortlich, die die Zeichen des CPC dehnt. Wenn Sie ein Zeichen in den verschiedenen Modi betrachten, handelt es sich immer um dasselbe Zeichen, nur daß die Relation von Höhe und Breite geändert wurde. Durch diese Parallelbedienung von Pixeln ändert sich auch die Auflösung der Zeichen in der Zeile. Als Auflösung bezeichnet man die Anzahl der in einer Zeile darstellbaren Zeichen. Je nach MODE können mit abnehmender Auflösung 80, 40 oder 20 Zeichen in einer Bildschirmzeile abgebildet werden.

Wie sieht es mit der Farbe aus?

3.2.1 Farbe und Modi

Der CPC kann insgesamt 28 verschiedene Farben darstellen. Je nach verwendetem Modus kann nur eine begrenzte Anzahl davon gleichzeitig auf dem Bildschirm erscheinen. Die farblichen Möglichkeiten verhalten sich dabei genau umgekehrt zur Auflösung der Zeichen. Im MODE 2 müssen wir uns mit zwei Farben begnügen. Im MODE 1 stehen uns vier zur Verfügung und im MODE 0 haben wir eine ganze Farbpalette mit insgesamt sechzehn Farben zur Verfügung. Der Grund dafür liegt in der Abspeicherung der Grafik durch den CPC. Jedem Pixel ist genau ein Speicherplatz zugeordnet. Fassen wir nun wie oben in den MODES 0 und 1 mehrere Pixel zusammen, so werden ein, beziehungsweise drei Speicherplätze frei, und diese können dann zur zusätzlichen Farbcodierung benutzt werden. Doch davon später mehr. Hier wollen wir uns zunächst mit den praktischen Anwendungsmöglichkeiten der MODES beschäftigen.

Die Wahl des MODES stellt einen Kompromiß zwischen Auflösung und farblichen Möglichkeiten dar. Der Leser muß den gewünschten MODE daher nach seinem Programmierziel wählen. Für die präzise Darstellung von Funktionen in Form von Grafen, für die Textverarbeitung und ähnliche hochauflösende Zwecke bietet sich der MODE 2 an.

Bei der Programmierung von Spielen wird man auf den größeren Farbgenuß des MODE 0 nicht verzichten wollen. Bei gemischter Zielsetzung dagegen bietet sich die Verwendung des Standard-MODES 1 an. Noch ein anderer Punkt ist bei der Entscheidung, welcher MODE gewählt werden soll, zu beachten. Die Auswahl eines MODES hat etwas Endgültiges. Das rührt daher, daß bei einem Wechsel der MODES der Bildschirm und sämtliche WINDOWS gelöscht werden. Außerdem geht auch eine möglicherweise auf dem Schirm vorhandene Grafik verloren. Besonders bei einer aufwendigen Grafik oder bei komplizierteren und textintensiven Programmen kann der erneute Bildaufbau eine Menge Zeit und gegebenenfalls Programmzeilen kosten. Doch keine Panik. Alles ist machbar, oder jedenfalls fast alles. Allerdings ist es einfacher und zeitsparender, sich ein paar Minuten einige Gedanken zu diesem Thema zu machen, als später das halbe Programm umzuschreiben.

3.2.2 Die Farbdefinition

Im folgenden wollen wir die farblichen Möglichkeiten des CPC etwas näher unter die Lupe nehmen. Die Farbgebung des CPC ist etwas kompliziert im Vergleich zu anderen Rechnern, hat aber den Vorteil, für hochauflösende Grafik und in den Text-MODES nach denselben Gesetzen zu funktionieren. Die Basis der Farbgebung bilden sechzehn Farbregister mit den Nummern 0 bis 15. Diese können mit dem INK-Kommando mit ein oder zwei Farben belegt werden. So wird zum Beispiel mit

```
INK 1,2
```

das Farbregister 1 auf mittelblau gesetzt. 2 ist der Code für mittelblau. Geben wir dagegen

```
INK 0,2,6
```

ein, wechselt der Wert des Farbregisters 0 ständig zwischen 2 und 6 und damit zwischen Mittelblau und Rot, das Farbregister blinkt. Auf diese vorher gesetzten Farbregister können wir zurückgreifen, um die Hintergrundfarbe, die Schriftfarbe oder die Farbe für die hochauflösende Grafik zu bestimmen. Beim Arbeiten mit Text setzen wir mit PEN die Schriftfarbe, mit PAPER die Farbe des Hintergrunds. Haben wir davor das Farbregister 1 mit INK 1,24 auf Gelb gesetzt und das Farbregister 0 mit INK 0,0 auf Schwarz, können wir auf diese Werte mit PEN 1 und PAPER 0 zurückgreifen. Wir schreiben dann gelb auf schwarz. In Wirklichkeit haben wir mit INK 1 (gelb = PEN = Schrift) auf INK 0 (schwarz = PAPER = Hintergrund) geschrieben. Das führt zu dem für den Anfänger etwas verwunderlichen Effekt, daß nach INK 0,2 die Hintergrundfarbe auf Blau wechselt und wir jetzt gelb auf blau schreiben. Dabei werden nicht alle zukünftigen, sondern auch die bisherigen Farbsetzungen geändert. Auf der anderen Seite kann dieser Effekt auch sehr gut für Warnmeldungen benutzt werden. Dies demonstriert das Programm auf der nächsten Seite.

```
10 REM ************************
20 REM ** Warnmeldungsdemo 1 **
30 REM ************************
40 INK 0,0:INK 1,24
50 PAPER 0:PEN 1
60 PRINT"Moeglichkeit A  (1)"
70 PRINT"Moeglichkeit B  (2)"
80 PRINT:PRINT"Bitte waehlen Sie."
90 z$=INKEY$
100 IF z$=""THEN 90
110 IF z$<>"1" AND z$<>"2" THEN 130
120 PRINT"o.k.":END
130 INK 0,24:INK 1,0
140 PRINT"Eingabefehler"
150 FOR i=1 TO 1000:NEXT
160 INK 0,0:INK 1,24:PRINT:GOTO 60
```

Nach dem Setzen der Farben in der ersten Zeile und der Ausgabe der beiden wählbaren Möglichkeiten in den Zeilen 60 und 70, wird der Benutzer in Zeile 80 aufgefordert, seine Entscheidung zu treffen. Die nachfolgenden drei Zeilen sind die eigentliche Abfrageroutine. In den Zeilen 90 und 100 wird geprüft, ob eine Taste gedrückt wurde. Wurde sie gedrückt, prüft 110, ob eine gültige Eingabe erfolgt ist. Wenn ja, gibt der Rechner in Zeile 120 ein freundliches "o.k." aus. Ansonsten erfolgt in 110 der Sprung zur Zeile 130, der Rechner stellt die Hintergrundfarbe von 0 auf 24 um, die Vordergrundfarbe von 1 auf 0 und gibt den Text "Eingabefehler" aus. Dies wird dem Benutzer auch noch einmal optisch durch den Farbwechsel klargemacht. Nach der Zeitverzögerungsschleife in 150 werden die Farben wieder zurückgesetzt; es folgt ein Rücksprung in die Abfrageroutine. Damit es auch der letzte unausgeschlafenen Computeranwender mitbekommt, schlagen wir folgende kleine Änderung vor:

```
130 INK 0,24,0:INK 1,2,6
```

Der Rechner entfaltet jetzt ein wahres Feuerwerk von Farben. Der Hintergrund wechselt ständig zwischen schwarz und gelb, und davor blinkt der Vordergrund in blau und rot.

Eine andere Möglichkeit besteht darin, eine neue Schrift oder Hintergrundfarbe für die Ausgabe der Fehlermeldung zu wählen. Dazu bauen wir unser Warnmeldungsprogramm wie folgt um:

```
10 REM ************************
20 REM ** Warnmeldungsdemo 2 **
30 REM ************************
40 INK 0,0:INK 1,24:INK 2,6
50 PAPER 0:PEN 1
60 PRINT"Moeglichkeit A  (1)"
70 PRINT"Moeglichkeit B  (2)"
80 PRINT:PRINT"Bitte waehlen Sie."
90 z$=INKEY$
100 IF z$=""THEN 90
110 IF z$<>"1" AND z$<>"2" THEN 130
120 PRINT"o.k.":END
130 PAPER 2
140 PRINT"Eingabefehler"
150 PAPER 0
160 PRINT:GOTO 60
```

Wir definieren jetzt nicht das entsprechende INK-Register um, sondern greifen für unsere Warnmeldung auf ein neues Farbregister zu. Dieses Farbregister setzen wir warnwirksam auf die Farbe rot. Mit dem PAPER-Befehl in Zeile 130 ändern wir die Hintergrundfarbe auf dieses Register. Unsere Warnmeldung wird nun gelb auf rot ausgegeben. Eine weitere Steigerung können wir auch hier wieder erreichen, indem wir das Farbregister 2 doppelt belegen, so daß der Hintergrund bei der Ausgabe der Warnmeldung zu blinken beginnt. Der PAPER-Befehl in Zeile 150 dient auch hier dazu, den Ausgangszustand wiederherzustellen.

Denselben Trick können wir natürlich auch mit der Vordergrundfarbe durchspielen. Das Farbregister brauchen wir nicht erneut zu definieren; es genügt, wenn wir die Zeilen 130 und 150 durch

```
130 PEN 2
150 PEN 1
```

ersetzen.

Doch Vorsicht! Die letzten beiden Änderungen funktionieren nur in den MODES 0 und 1, denn im MODE 2 haben wir nur die unteren beiden Farbregister zur Verfügung. Benutzen wir höhere als die erlaubten Angaben für PEN und PAPER, setzt der CPC diese automatisch durch Anwendung der MODULO-Funktion herab. Im hochauflösenden MODE 2 nimmt er alle PAPER- und PEN-Werte MODULO 2, im MODE 1 MODULO 4.

Man kann die Wirkung der MODULO-Funktion nachbilden, indem man den PEN oder PAPER-Wert durch den MODULO-Wert dividiert, vom Ergebnis den ganzzahligen Vorkommateil abzieht und das Ergebnis wieder mit dem MODULO-Wert multipliziert. In unserem Fall sähe die Berechnung dann folgendermaßen aus:

```
2 mod 2 :  2/2=1
           1-1=0
           0*2=0
```

Die Anwendung der MODULO-Funktion setzt jede Zahl in den Bereich zwischen 0 und dem MODULO-Wert minus 1 herab. Dies führt dazu, daß statt dem erwünschten Farbkontrast unsere Warnmeldung jetzt mit schwarz auf schwarz geschrieben wird und damit unsichtbar bleibt.

Das nächste Programm demonstriert die Wirkung der MODULO-Funktion. Nach Eingabe des gesetzten MODES und Auswahl des PAPER- beziehungsweise PEN-Werts gibt es das vom Computer ausgewählte Farbregister an. MOD ist die im Rechner intern verfügbare MODULO-Funktion.

```
10 REM ************************
20 REM ** Demo:Modulofunktion **
30 REM ************************
40 INPUT"MODE (0-2)";m
50 INPUT"PAPER-/PEN-Wert";w
60 IF m<0 OR m>2 OR w<0 OR w>15 THEN PRINT"Fehler":GOTO 40
40 IF m=1 THEN s=4
50 IF m=2 THEN s=2
60 IF m=0 THEN s=16
70 PRINT"benutztes Farbregister ";w MOD s
80 PRINT:GOTO 40
```

Die nachfolgende Tabelle stellt die Beziehungen noch einmal im Zusammenhang dar. In der linken Spalte finden Sie den angegebenen PEN- beziehungsweise PAPER-Wert und daneben die in Abhängikeit davon vom Computer ausgewählten Farbregister.

Tabelle 3.1

PEN- bzw. PAPER-Wert	benutztes Farbregister		
	Mode 0	Mode 1	Mode 2
0	0	0	0
1	1	1	1
2	2	2	0
3	3	3	1
4	4	0	0
5	5	1	1
6	6	2	0
7	7	3	1
8	8	0	0
9	9	1	1
10	10	2	0
11	11	3	1
12	12	0	0
13	13	1	1
14	14	2	0
15	15	3	1

Eine andere oft benutzte Möglichkeit im Spiel mit Licht und Farbe ist die Benutzung der REVERS-Darstellung. Bei der REVERS-Darstellung werden Schrift- und Hintergrundfarbe gegeneinander ausgetauscht. Beim CPC 464 können wir das durch einfaches Vertauschen der Werte hinter PAPER und PEN erreichen. Hatten wir also am Anfang mit

PEN 1 auf PAPER 0 geschrieben, so können wir nun mit

PAPER 0 auf PEN 1 eine REVERS-Darstellung erreichen.

Eine andere Möglichkeit ist uns durch den Rückgriff auf das Betriebssystem gegeben. Das Betriebssystem hat eine Reihe nützlicher Routinen gespeichert, die der CPC benutzt, um die von uns in der Hochsprache BASIC vorgegebenen Kommandos zu übersetzen. Eine davon ist die Text-INVERS-Routine. Sie vertauscht die laufenden INKs für PAPER und PEN. Auf diese Routine können wir mit dem CALL-Befehl zurückgreifen. Mit dem CALL-Befehl können wir beliebige Routinen, die in Maschinensprache (= die Sprache des CPC,

die er versteht) geschrieben sind, aufrufen. Probieren Sie die nachfolgende Befehlsfolge aus, die Sie sowohl als Programm, als auch im Direkt-MODE eingeben können. Nach dem Setzen der Farbregister wird zuerst ein normal geschriebener Text ausgegeben, danach folgt der Aufruf der Text-INVERS-Routine. Diese schaltet den REVERS-MODE ein, indem sie die INKs für PEN und PAPER vertauscht. Nochmaliges Aufrufen der Routine bringt uns wieder in den normalen MODE zurück. Diese Routine finden Sie, wie auch viele andere, in Kapitel 6.1 im Anhang zu diesem Buch.

```
10 REM *****************
20 REM ** Revers-Demo **
30 REM *****************
40 INK 0,0:INK 1,6:PAPER 0:PEN 1
50 PRINT"Normal"
60 CALL &BB9C
70 PRINT"Reverse"
80 CALL &BB9C
90 PRINT"wieder normal"
```

Der Umgang mit den Farben beim CPC 464 ist für den Anfänger nicht leicht, jedoch bieten sich, wie wir gesehen haben, fast unerschöpfliche Möglichkeiten. Das Hauptproblem liegt darin, die Übersicht nicht zu verlieren. Besonders beim Arbeiten im MODE 0 kommt es oft vor, daß man vergessen hat, welche Farbregister denn nun mit welcher Farbe belegt wurden. Aus diesem Grund haben wir den auf der nächsten Seite abgedruckten Farbplaner entwickelt. Er ermöglicht es, eine optimale Übersicht über die Belegung der Farbregister und die zur Verfügung stehenden Farben zu behalten. In die Kästchen werden je nach verwendetem MODE die entsprechenden Farbcodes eingetragen. Wer mit der Verwendung der Farbcodes noch nicht so ganz vertraut ist, kann hier daneben auch noch die Farbbezeichnungen eintragen. Für das praktische Arbeiten kann es nützlich sein, sich davon ein paar Kopien anzufertigen.

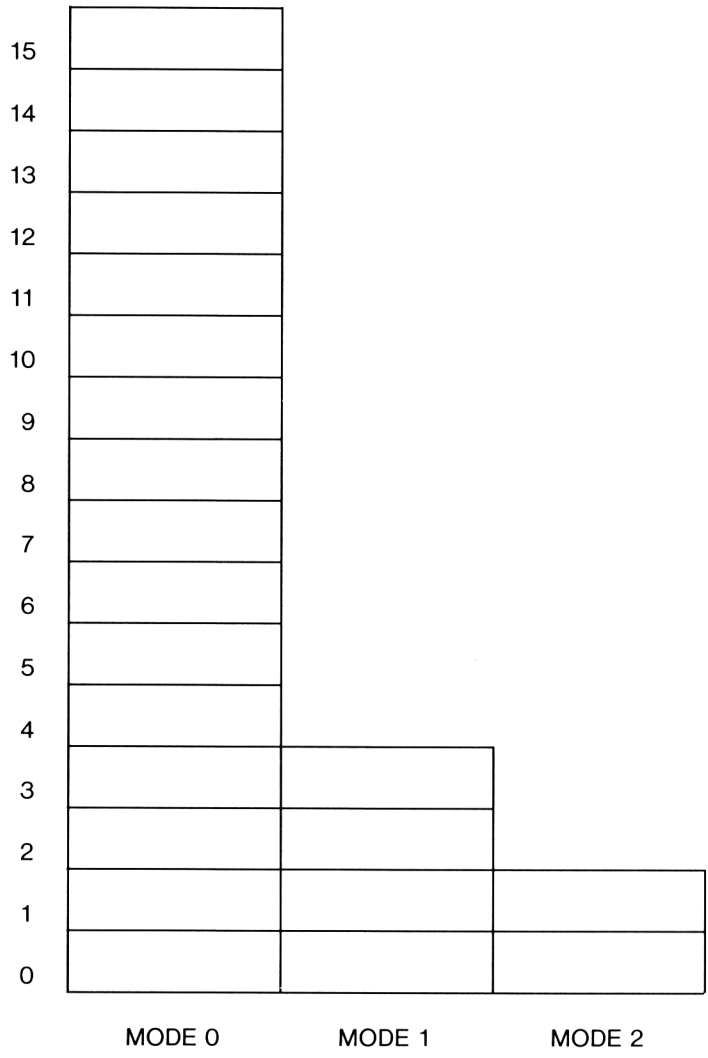


Bild 3.1 Farbplaner

3.2.3 Die Wahl der richtigen Farbe

Eine Tabelle der beim CPC verfügbaren Farben finden Sie im Anhang 6.2. Wir haben die auch im Handbuch abgedruckte Farbtabelle, die die Übersetzung der Farbcodes in Farben angibt, noch um ihr Gegenstück erweitert, eine Übersetzungstabelle von Farbe auf Code. Die Farbcodes sind von oben nach unten nach steigender Helligkeit beziehungsweise Intensität geordnet. Diese Tabellen bieten sich insbesondere für das praktische Programmieren an, um die benötigte Farbe, beziehungsweise Farbschattierung herauszufinden. Um die optimale Farbe herauszufinden, sei an dieser Stelle ein kleiner Trick verraten: Hat man mit

```
INK 0,X:PAPER 0:CLS
```

den Bildschirm auf die Farbe X gesetzt, kann man mit BORDER Y andere Farben dagegen testen. Y steht für den Farbcode der alternativen Farbe. Dieses Vorgehen läßt sich im übrigen auch gut dazu verwenden, um die Übereinstimmung von verschiedenen Farben zu überprüfen, und somit ein ausgewogenes Farbbild zu erhalten.

3.3 Text und Grafik

Nachdem wir uns im letzten Kapitel mit den farblichen Möglichkeiten des CPC 464 beschäftigt haben, wollen wir nun auf die Zusammenhänge zwischen Text und Grafik eingehen. Im Mittelpunkt stehen dabei die Gestaltungsmöglichkeiten des Bildschirms bei der Verwendung von Text. Probleme der Positionierung von Text, der formatierten Textausgabe und eines möglichst optimalen, das heißt benutzerfreundlichen Bildaufbaus sollen hier behandelt werden.

3.3.1 Die Positionierung von Text

Die Aufgabenstellung bei der Positionierung von Text besteht darin, einen gegebenen Text, in der Regel eine Zeichenkette (STRING), zielgerichtet an einer bestimmten Stelle des Bildschirms oder eines Teils des Bildschirms (WINDOW), auszugeben. Wir können dabei zwei Arten der Adressierung unterscheiden: die absolute und die relative Adressierung. Bei der absoluten Adressierung wird die Anfangsposition des auszugebenden STRINGS auf dem Bildschirm direkt festgelegt, das heißt durch Angabe von Bildschirmspalte und Bildschirmzeile. Das Gegenstück dazu stellt die relative Adressierung dar. Hierbei wird die neue Position der Textausgabe in Abhängigkeit von der alten Cursor-Position definiert.

Wir wollen uns zunächst mit den Möglichkeiten einer absoluten Adressierung beschäftigen. Diese wird beim CPC 464 durch die Verwendung des LOCATE-Befehls ermöglicht. Als Ausgabefeld wollen wir zunächst den ganzen Bildschirm unterstellen. Auf das Verhältnis zwischen LOCATE-Befehl und WINDOWS werden wir in dem Kapitel über WINDOWS näher eingehen. Der LOCATE-Befehl legt den Anfangspunkt einer Textausgabe fest. Bezugsgröße sind die Bildschirmzeilen beziehungsweise -spalten. Bei der Textausgabe stehen uns generell 25 Textzeilen zur Verfügung, je nach eingeschaltetem MODE mit 20 (MODE 0), 40 (MODE 1) und 80 (MODE 2) Zeichen. Bei der Angabe der Koordinaten müssen wir den jeweiligen MODE berücksichtigen. So beginnt zum Beispiel bei dem Kommando LOCATE 10,12 im MODE 0 die Textausgabe im Bildschirmzentrum. Im MODE 1 findet die Ausgabe immer noch auf der mittleren Zeile statt, sie beginnt jedoch schon im linken Viertel der Zeile. Im MODE 2 schließlich wird der Beginn auf der mittleren Bildschirmzeile in das erste Achtel des Bildschirms verschoben. Im folgenden wollen wir uns damit beschäftigen, wie mit dem LOCATE-Befehl und gleichzeitiger Verwendung von STRING-Operationen interessante Titelbilder, Überschriften und Laufschriften erzeugt werden können.

3.3.2 Laufschriften und absolute Adressierung

Im ersten Programm wollen wir eine einfache, möglichst große Laufschrift erzeugen. Als Text haben wir dafür "Schneider CPC 464" gewählt. Dieser ist als STRING in der Variablen Z$ in Zeile 70 abgespeichert. Um am Anfang und am Ende eine definierte Anzahl von Leerzeichen zu erhalten, werden an diesen STRING in Zeile 80 mit dem SPACE$-Kommando einige Leerzeichen abgehängt. Die nachfolgenden Zeilen von 90 bis 120 stellen eine Schleife dar. In dieser wird der STRING nun wieder zerpflückt. Mit dem MID$-Kommando trennen wir aus dem STRING beginnend mit dem durch I vorgegebenen Zeichen einen Teil-STRING mit der Länge 19 heraus. Da I von 1 bis 17 läuft, enthält unser Teil-STRING mit zunehmendem I immer weniger Vorleerstellen und einen immer größeren Teil unseres Ausgabe-STRINGS. Dieser wird durch das LOCATE-Kommando in Zeile 100 immer an derselben Stelle ausgegeben. Die später gedruckten STRINGS überdecken somit die älteren. Dies führt zum Eindruck der laufenden Schrift. Zeile 115 verzögert diesen Prozeß durch eine Zeitschleife, um eine gute Lesbarkeit der Schrift sicherzustellen.

```
10 REM *****************
20 REM **Laufschrift 1**
30 REM *****************
40 INK 0,0:INK 1,2
50 BORDER 0:PAPER 0:PEN 1
60 MODE 0
70 Z$="SCHNEIDER CPC 464"
80 Z$=SPACE$(19)+Z$+SPACE$(25)
90 FOR I= 1 TO 19
100 LOCATE 1,12
110 PRINT MID$(Z$,I,19)
115 FOR T=1 TO 150:NEXT T
120 NEXT I
130 GOTO 90
```

Das nächste Programm stellt eine Variante desselben Themas dar. Wir geben uns jetzt nicht damit zufrieden, nur einen wandernden STRING auszugeben und damit eine einfarbige Laufschrift zu erzeugen. Unser Ziel ist es jetzt, diese Laufschrift auch noch mehrfarbig zu erhalten. Basis ist dabei das schon bekannte Programm, daß wir allerdings in einigen wichtigen Punkten erweitert haben. Wir geben den mit MID$ erzeugten STRING nicht mehr direkt aus,

sondern speichern ihn in der Variable Z2$ zwischen. In Zeile 110 und 120 wird dieser Teil-STRING jetzt noch einmal Buchstabe für Buchstabe zerpflückt. Jeder Buchstabe wird in einer anderen Farbe ausgegeben. Dies erreichen wir wieder, indem wir auf unsere Zählvariable die MODULO-Funktion anwenden, die wir schon aus den vorherigen Kapiteln kennen. Indem wir J MODULO 5 nehmen und dazu 1 addieren, erhalten wir für unser Farbregister Werte zwischen 1 und 5. Diese benutzen wir, um mit dem nachfolgenden PEN-Kommando die Schriftfarbe zu setzen. Dadurch erhalten wir ein sich nach fünf Buchstaben wiederholendes Farbband von Zeichen.

```
10 REM *****************
20 REM **Laufschrift 2**
30 REM *****************
40 INK 0,0:INK 1,2:INK 2,6:INK 3,21:INK 4,7:INK 5,24
50 BORDER 0:PAPER 0:PEN 1
60 MODE 0
70 Z$="SCHNEIDER CPC 464"
80 Z$=SPACE$(17)+Z$+SPACE$(18)
90 FOR I=1 TO LEN(Z$)-21
100 Z2$=MID$(Z$,I,20):LOCATE 1,12
110 FOR J=1 TO 19
115 FOR T = 1 TO 150:NEXT T
120 m=(J MOD 5)+1:PEN m:PRINT MID$(Z2$,I,1);:NEXT J
130 NEXT I
140 GOTO 90
```

Wir verlassen nun die Welt der Breitschrift und gehen in den MODE 1 über. In dem Programm auf der folgenden Seite stehen uns mehr Zeichen pro Zeile zur Verfügung, welche aber auf Grund ihrer geringeren Breite weniger plastisch wirken. Um dennoch eine gute Wirkung unserer Schrift zu erhalten, rollen wir diesmal gleich drei Schriftbänder parallel herein. Mit dieser Anzahl schöpfen wir auch die verfügbaren Farbregister voll aus. Die Grundkonzeption ist mit der des ersten Programms weitgehend identisch. Der Hauptunterschied liegt in der eingeschobenen Schleife in den Zeilen 90 bis 110. Der mit MID$ erzeugte Teil-STRING wird jetzt nicht nur in Zeile 12, sondern in Abhängigkeit von J, in den Zeilen 6, 12 und 18 parallel ausgegeben. Dies führt zu drei gleichzeitig hereingerollten Schriftbändern. Auch bei der Auswahl der auszugebenden Größen haben wir eine Änderung vorgenommen. Der CPC gibt jetzt nicht mehr einen Text als Laufschrift aus, sondern eine Reihe von Männchen in verschiedenen Posen. Dadurch entsteht der Eindruck von sich bewegenden Figuren; ein Effekt, der sehr

gut bei der Gestaltung von Action-Spielen eingesetzt werden kann.

```
10 REM *****************
20 REM **Laufschrift 3**
30 REM *****************
40 INK 0,1:INK 1,24:INK 2,6:INK 3,15
50 BORDER 1:PAPER 0:PEN 1:MODE 1
60 FOR I=1 TO 40:Z$=Z$=Z$+CHR(248+3*RND(1)):NEXT I
70 Z$=SPACE$(40)+Z$+SPACE$(40)
80 FOR I= 1 TO LEN(Z$)-39
90 FOR J= 1 TO 3:PEN J
100 LOCATE 1,6*J:MID$(Z$,I,39)
110 NEXT J,I
```

Bei unserem vierten und letzten Laufschriftprogramm verwenden wir eine interessante grafische Anordnung der Textzeilen, um Aufmerksamkeit zu erregen. Der Text wird wiederum in Form von Schriftbändern ausgegeben; diese laufen allerdings mit ein bis zwei Buchstaben pro Satz hintereinander her. Dies erreichen wir, indem wir die als Zählvariable ohnehin benötigte Variable J auch noch in dem mit MID$-Ausdruck zur Verschiebung des Anfangspunkts den Teil-STRINGS einsetzen. Im übrigen wäre noch darauf hinzuweisen, daß die Zeitschleifen völlig willkürlich gewählt wurden und vom Benutzer je nach Belieben zu langsamer oder schnellerer Ausgabe durch Herabsetzen, beziehungsweise Heraufsetzen der Schleifenwerte verändert werden können.

```
10 REM *****************
20 REM **Laufschrift 4**
30 REM *****************
40 INK 0,0:INK 1,24:INK 2,6:INK 3,21
50 BORDER 0:PAPER 0:PEN 1:MODE 0
60 Z$="Schneider CPC 464"
70 Z$=SPACE$(22)+Z$+SPACE$(20)
80 FOR I= 4 TO 24
90 FOR J= 1 TO 3
100 PEN J
110 LOCATE 1,6*J:PRINT MID$(Z$,I-J,19)
120 NEXT J
130 FOR T=1 TO 50:NEXT T
140 NEXT I
150 FOR T=1 TO 500:NEXT T
160 GOTO 80
```

Nachdem wir den LOCATE-Befehl etwas näher betrachtet haben, wol len wir uns jetzt den anderen Befehlen im Bereich der absolute Adressierung zuwenden. Da wäre zunächst der TAB-Befehl. Mit die sem Befehl legen wir die Anfangsspalte der Textausgabe fest. J nach MODE verfügen wir wieder über 20, 40 oder 80 Spalten, in di wir unseren Cursor setzen können. Der Anfang der Ausgabe is nicht ganz unabhängig von der vorherigen Stellung des Cursors Befinden wir uns in der 8. Spalte, schiebt ein TAB(15)-Kommand den Cursor sieben Stellen nach rechts auf die Spalte 15. Endet die letzte Textausgabe dagegen zum Beispiel in Spalte 20, wir der Cursor nicht fünf Zeichen zurückbewegt, sondern es wird di Spalte 15 in der nächsten Zeile angenommen. Das TAB-Kommando is nicht vollkommen unabhängig von der Ausgangsposition.

In den Bereich der Ausgabepositionierung gehört auch noch ein an deres Kommando, dem man seine Kommandoeigenschaften auf den er sten Blick nicht ansehen würde: das Komma.

Im BASIC des CPC - und auch bei vielen anderen BASIC-Maschinen erfüllt das Komma eine Zwitterrolle. Zum einen wird es dazu be nutzt, um die Argumente einer Funktion voneinander zu trenne (DELIMITER). Im Zusammenhang mit der Textausgabe - speziell i PRINT-Kommando - erfüllt es eine andere Funktion. Der Bildschir des CPC 464 ist in mehrere Zonen unterteilt. Diese verlaufen vo oben nach unten mit einer Breite von normalerweise 13 Zeichen Mit dem ","-Kommando bewegen wir nun den Cursor auf den Begin einer solchen Zone vor. Das nachfolgende kleine Programm demon striert die Beziehungen noch einmal im Zusammenhang. Es besteh im wesentlichen aus der Eingabe-Schleife, in der Name, Straße un Ort eingegeben werden und der Ausgabeschleife. Diese gibt die ge speicherten Werte in Zeilen 110 und 120 als Tabelle aus. Am lin ken Rand beginnend wird der Name ausgegeben, 12 Zeichen weite rechts beginnt die Ausgabe der Straße. Durch das vorangestellt Komma in Zeile 120 wird der Anfang der Ortsangabe genau über de Anfang der Straße gesetzt.

```
10 REM ****************
20 REM **Mini-Adresse**
30 REM ****************
40 DIM N$(5) S$(5),O$(5)
50 FOR I=1 TO 5
60 INPUT"Name   ";N$(I)
70 INPUT"Strasse";S$(I)
80 INPUT"Ort    ";O$(I)
90 NEXT I
100 FOR I=1 TO 5
110 PRINT N$(I),S$(I)
120 PRINT ,O$(I):NEXT I
```

Das ","-Kommando eignet sich vorzüglich zur schnellen Erstellung von Tabellen. Abrechnungen und Kalkulationstabellen. Die hier benutzten Adreßlisten sind mit dem KOMMA-Kommando auf einfache Weise erstellbar. Bedingung für eine vernünftige Ausgabe ist daß der auszugebende Text nicht größer als die Zonenbreite - im Normalfall dreizehn Zeichen - ist. Bei größeren Texten springt der CPC nämlich sonst zum nächsten Zonenbeginn und bringt damit unsere Tabelle durcheinander. Wir müssen uns jedoch nicht mit den beim Einschalten vorgegebenen dreizehn Zeichen Zonenbreite abfinden. Mit dem ZONE-Kommando können wir diese jederzeit variieren. Übersteigen zum Beispiel in einer kaufmännischen Rechnung die Zahlen den Wert von 9999.99 nicht, so können wir unsere Zonenbreite auf acht Zeichen vermindern. Statt drei verfügbaren Kolonnen stehen uns dann im MODE 1 fünf zur Verfügung.

3.3.3 Relative Adressierung

Wenden wir uns nun der relativen Adressierung zu. Am einfachsten ist diese vom BASIC aus mit dem SPC-Befehl realisierbar. Dieser schiebt vor der Ausgabe eines Textes eine definierte Anzahl von Leerzeichen ein. Der Unterschied zum TAB-Befehl besteht darin, daß diese Anzahl immer gleich ist. Dadurch wird eine vollkommen andere Wirkung erreicht. Steht der Cursor in Spalte 10, 12 oder 15, bewegt ihn ein TAB(20)-Kommando immer auf Spalte 20 vor. Ein SPC(20)-Kommando dagegen schiebt zwanzig Leerzeichen ein. Dadurch wird der Cursor in Spalte 30, 32 und 35 vorbewegt. TAB, "," und auch das SPC-Kommando wirken nur im Zusammenhang mit dem PRINT-Befehl. Sie sind sowohl reihbar wie auch untereinander kombinier-

bar. Somit sind zum Beispiel folgende Kombinationen möglich:

```
PRINT TAB(5);"Nr.1";TAB(20);"DM 1000,-"

PRINT SPC(10);"10 Schritte","weiter"

PRINT TAB(2);"2";SPC(5);"5 weiter"
```

Doch damit sind die Möglichkeiten der relativen Adressierung noch lange nicht erschöpft. Im Handbuch freilich findet man dazu wenig. Die Rede ist von den Systemsteuerzeichen und ihren Möglichkeiten, die so vielfältig sind, daß wir ihnen ein ganzes Unterkapitel gewidmet haben.

3.3.4 Die Systemsteuerzeichen

Die Systemsteuerzeichen gehören wie der Aufruf von Systemroutinen, mit dem wir uns im Kapitel über Farbe schon etwas näher beschäftigt hatten, in den Bereich des Betriebssystems. Die Unterschiede liegen in der Funktion und der Art des Aufrufs dieser Routinen. Die Ausgabesteuerzeichen haben im wesentlichen mit dem Bildschirm zu tun, während Betriebssystem-Routinen für fast alle Anwendungszwecke verfügbar sind. Während die Betriebssystem-Routinen mit dem CALL-Befehl als Maschinenunterprogramme aufgerufen werden, werden die Ausgabesteuerzeichen einfach mit dem PRINT-Befehl ausgegeben. Stößt der CPC in einem Ausgabetext auf ein solches Steuerzeichen, druckt er nicht dieses Steuerzeichen, sondern geht in die entsprechende Routine des Betriebssystems. Wer schon einmal mit Commodore-Rechnern gearbeitet hat, wird diese Art der Ausgabesteuerung kennen.

Man tippt:

```
PRINT "‹CURSOR-NACH-UNTEN-TASTE›"
```

und der Cursor bewegt sich eine Zeile nach unten. Die meisten Ausgabesteuerzeichen sind über Tastatur verfügbar und zwar in der CONTROL-Ebene. Ein Beispiel:

Drücken der ‹G-Taste› bringt ein kleines g auf den Schirm.
Drücken von ‹SHIFT + G-Taste› erzeugt ein großes G.

Drücken von ‹CTRL + G-Taste› bringt dagegen ein Glockensymbol auf dem Bildschirm.

Dieses Zeichen ist für die Systemsteuerzeichen untypisch, da es mit der Bildschirmgestaltung wenig zu tun hat. Dafür hat es den Vorteil, einen eindeutigen Effekt zu erzeugen. Gibt man

```
PRINT "‹CONTROL-G-TASTE›"
```

ein, läßt der CPC einen kurzen Piepser erklingen, vorausgesetzt, man hat den Lautstärkeregler aufgedreht. Dies ist derselbe Ton, den wir schon bei der Textverarbeitung zur Kennzeichnung des Zeilenendes benutzt haben. Dort hatten wir ihn allerdings mit einer anderen Methode aufgerufen, nämlich mit der CHR$-Funktion.

Jedes auf dem Bildschirm darstellbare Zeichen hat eine Nummer. Diese Nummern laufen von 32 bis 255 und enthalten neben den von der Tastatur aus eingebbaren Zeichen noch eine Reihe weiterer. So ist zum Beispiel 65 der Bildschirmcode für A. CHR$(65) erzeugt einen STRING, der aus einem großen A besteht. Diesen Mini-STRING können wir nun zu einer Zeichenkette addieren oder ganz einfach mit

```
PRINT CHR$(65)
```

ausgeben. Im letzteren Fall druckt der CPC ein großes A auf den Bildschirm. Wir sind mit der CHR$-Funktion nicht nur auf die ohnehin von der Tastatur aus erreichbaren Zeichen beschränkt.

```
PRINT CHR$(251)
```

bringt uns zum Beispiel ein kleines Männchen auf den Bildschirm. Aber auch grinsende Gesichter (CHR $(224)) oder griechische Buchstaben (zum Beispiel: CHR$(191)) sind verfügbar. Eine genaue Liste des Zeichensatzes des CPC 464 finden Sie im Anhang Ihres Handbuchs. Wir können ihn aber auch jederzeit mit

```
FOR I = 32 TO 255:PRINT CHR$(I);:NEXT I
```

ausdrucken. Wir wollen uns an dieser Stelle mit dem darunterliegenden Bereich, den Zeichen von 0 bis 31 beschäftigen, denn dies sind die Systemsteuerzeichen. CHR$(7) entspricht dabei dem CTRL-G (wenn wir in Zukunft CTRL-x schreiben, dann müssen Sie die CONTROL-Taste festhalten und die entsprechende Taste x drücken)

und deshalb erzeugt

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

denselben Effekt. Dieser läßt sich neben der Ausgabe eines Warntons auch gut verwenden, um auf die Ausgabe eines Textes aufmerksam zu machen. Tippen wir

```
PRINT CHR$(7)+"NAME",
```

wird das Wort "Name" ausgegeben und gleichzeitig ertönt ein Piepser aus dem Lautsprecher. Denselben Effekt hätten wir mit

```
PRINT"‹CTRL-G›NAME"
```

erzielen können. Die Zahl in der CHR$-Funktion ist mit der Nummer des Buchstabens in der CONTROL-Funktion im Alphabet identisch. G ist der siebte Buchstabe im Alphabet und somit liefert CHR$(7) den entsprechenden Code. Analog dazu können wir das Drücken von CTRL und F durch ein CHR$(6) ersetzen und so weiter. Dabei ist zu beachten, daß CTRL und C für die Character-Codes 2 und 3 gleichzeitig gelten. Somit entspricht dann ‹CTRL-A› der Funktion CHR$(0) und ‹CTRL-B› der Funktion CHR$(1). Doch schauen wir uns die anderen Systemsteuerzeichen an. Beim Ausprobieren und Kennenlernen der Steuerzeichen sollten Sie möglichst kein Programm mehr im Speicher haben, da der CPC sich bei mehrmaligen Fehlbedienungen gelegentlich aufhängt. Sie bekommen ihn dann meistens mit

‹CTRL› ‹SHIFT› ‹ESC›

oder spätestens durch Aus- und Wiedereinschalten wieder zum Laufen. Ein im Speicher befindliches Programm ist dann in jedem Fall verloren.

Kommen wir zunächst zum niedrigsten vom Benutzer sinnvoll verwendbaren Kommando. Der Zeichen-Code 4 hat denselben Effekt wie das MODE-Kommando. Mit einem zweiten Zeichen geben wir den MODE an.

```
PRINT CHR$(4)   oder   PRINT "‹CTRL-D›"
```

bewirkt das Umschalten in den MODE 1 und ist damit dasselbe, wie der Ausdruck

```
PRINT CHR$(4)+CHR$(1)
```

```
PRINT CHR$(4)+CHR$(0)  bzw.  PRINT CHR$(4)+CHR$(2)
```

sorgen für die Umschaltung in den MODE 0 beziehungsweise den MODE 2. Das nächste Kommando in dieser Folge haben wir schon kennengelernt. Es ist die Funktion CHR$(7), die einen Piepser erzeugt. Als nächstes kommen wir zu den Cursorsteuerzeichen.

```
PRINT CHR$(8)  oder  PRINT "<CTRL-H>"
```

bewegen den Cursor um eine Stelle nach links. Mit CHR$(9) oder CTRL und gleichzeitigem Drücken von I erreicht man dasselbe nach rechts. Geben wir

```
PRINT CHR$(10)  oder  PRINT "<CTRL-J>"
```

ein, geht der Cursor eine Zeile nach unten; mit CHR$(11) oder CTRL und K geht es wiederum in die Gegenrichtung. Die letzten beiden Paare bewegen nicht nur den Cursor. Befindet sich der Cursor auf der letzten Zeile, das heißt in Zeile 25, bewirkt der Ausdruck eines CHR$(10) ein Scrollen des Bildschirms: der Schirm wird eine Zeile nach oben bewegt. Diesen Effekt kennen wir schon vom einfachen Drucken in der letzten Zeile. Interessant ist, daß wir den Bildschirm mittels CHR$(10) in der obersten Zeile zum Scrollen in der Gegenrichtung, das heißt nach unten, veranlassen können. Dies ist sehr nützlich, wenn wir Texte nach unten oder oben rollen wollen, wobei jeweils eine neue Textzeile ausgegeben werden soll. Eine Anwendung dafür stellt unser Textprogramm aus dem zweiten Kapitel dar. Überlegen Sie, wie Sie dieses Rollen einbauen können. Einen anderen Effekt hat

```
PRINT CHR$(12)  oder  PRINT "<CTRL-L>"
```

Bei Eingabe dieses Codes führt der CPC 464 ein CLEAR/HOME aus. Das heißt, er löscht den Bildschirm und geht auf den Anfang der obersten Bildschirmzeile. Dieser Befehl ist das Äquivalent zum BASIC-Befehl CLS in Maschinensprache.

```
PRINT CHR$(13)  oder  PRINT "<CTRL-M>"
```

sind gleichbedeutend mit dem Drücken der RETURN-Taste. Der Cursor wird auf den linken Rand der aktuellen Zeile bewegt. Damit sind sehr schnelle Cursorbewegungen möglich. Auch das Überschreiben von Texten kann auf diese Weise geschehen, da hier kein Zeilenvorschub erfolgt. Wir befinden uns also nach der Ausführung des Befehls wieder am Anfang der Zeile, auf der wir gerade geschrieben haben. Ein breites Anwendungsfeld hierfür bilden wieder Textverarbeitungsprogramme.

Die nächsten Codes stellen die Äquivalente für die BASIC-Befehle PAPER und PEN dar.

```
PRINT CHR$(14)  oder  PRINT "<CTRL-N>"
```

beziehungsweise

```
PRINT CHR$(15)  oder  PRINT "<CTRL-O>"
```

sind mit der Ausführung eines PAPER 1- beziehungsweise PEN 1-Befehls identisch, und damit für den Anwender wohl nicht von allzu großem Interesse. Durch Anhängen eines zweiten Zeichens erhalten wir wiederum die anderen Farbregister.

```
PRINT CHR$(15)+CHR$(2)
```

führt den Befehl PEN 2 aus. Ein Anwendungsfeld hierfür könnte ein Programm sein, das den Bildschirm hauptsächlich aus STRINGS aufbaut. Hier wäre es dann möglich, Farbumschaltungen mit in den Text zu integrieren. Dies bietet ungeahnte Möglichkeiten, ist aber wohl keine Sache für Anfänger. Zur Klarheit sollten Sie die nachfolgenden Befehle ausprobieren:

```
a$="abcd"+CHR$(15)+CHR$(2)+"efgh"+CHR$(15)+CHR$(3)+"ijkl"
PRINT a$
```

Die Kommandos sollten im MODE 1 in der Ausgangsfarbzusammenstellung ausprobiert werden. Gegebenenfalls ist der Computer mit <SHIFT><CTRL><ESC> zuerst wieder in den Einschaltzustand zu versetzen.Interessanter sind die Codes

```
PRINT CHR$(19)  oder  PRINT "<CTRL-S>"
```

beziehungsweise

```
PRINT CHR$(20)  oder  PRINT "<CTRL-T>"
```

Mit diesen ist es möglich, in Abhängigkeit von der Position des Cursors einen Teil des Bildschirms zu löschen. Mit CHR$(19) löschen wir dabei den Bildschirm oberhalb des Cursors. Mit der Ausgabe von CHR$(20) löschen wir bis in die untere rechte Bildschirmecke. Anfangs- bzw. Endpunkt ist immer die aktuelle Cursorposition. Dies ist sehr nützlich, um bei textintensiven Programmen schnell eine Löschung der benötigten Bildschirmbereiche zu erhalten. Der Punkt, an dem sich der Cursor momentan befindet, wird als letzter Punkt mitgelöscht. Wird zum Beispiel der Text in den Bildschirmzeilen 1 bis 10 nicht mehr benötigt, setzt man den Cursor mit dem LOCATE-Kommando auf das Ende der zehnten Zeile, das heißt je nach eingeschaltetem MODE in Spalte 20, 40, oder 80 und führt dann ein

```
PRINT "<CTRL-S>"
```

aus. Damit wird der obere Teil des Bildschirms gelöscht. Mit

```
PRINT CHR$(24)  oder  PRINT "<CTRL-X>"
```

haben wir eine andere Möglichkeit, die im Kapitel über Farben angesprochene REVERS-Darstellung zu erreichen. Es vertauscht die Schrift und die Hintergrundfarbe. Mit

```
PRINT CHR$(30)  oder  PRINT <CTRL-^>
```

wird ein Cursor-HOME ausgeführt, das heißt der Cursor wird auf den Anfang der obersten Bildschirmzeile gesetzt. Im Gegensatz zu

```
PRINT CHR$(12)
```

wird der Bildschirm nicht gelöscht. Dies kann dazu benutzt werden, schnell in den oberen Teil des Bildschirms zu gelangen. Will man den Cursor an einer anderen Stelle am Bildschirm haben, zum Beispiel in der Mitte, so hilft

```
PRINT CHR$(31)
```

Dieser Befehl ist das Äquivalent zum LOCATE-Befehl. Wir müßten eigentlich mit zwei nachfolgenden CHR$-Kommandos die Zeilennummer und die Bildschirmspalte eingeben. Falls wir dies unterlassen, und das ist der Trick dabei, werden sie auf 0,12 gesetzt. Damit

wird der Cursor auf die Zeile 12 gesetzt. Die Weitersteuerung des Cursors kann dann mit den einfachen Cursorsteuerung-Befehlen CHR$(8) bis CHR$(11) erfolgen.

Sie sehen: die richtige und kombinierte Anwendung der Ausgabesteuerzeichen eröffnet eine Vielzahl von Möglichkeiten. Die Liste der oben besprochenen und auf der nächsten Seite noch einmal tabellarisch zusammengefaßten Ausgabesteuerzeichen erhebt keinen Anspruch auf Vollständigkeit. Sie wurde nach dem Kriterium Brauchbarkeit für den durchschnittlichen Benutzer zusammengestellt.

Wer an kniffligen Problemen interessiert ist, hat hier noch ein breites Arbeitsfeld. Für Tüftler hier gleich eine kleine Anregung: Geben Sie auf einer Zeile einen kleineren Text ein und führen Sie dann auf einer anderen, möglichst freien Zeile den Befehl:

```
PRINT CHR$(22)
```

aus. Danach können Sie versuchen, den oben geschriebenen Text mit DELETE zu löschen oder durch Überschreiben zu vernichten. Sind Sie Besitzer eines Farbmonitors und befindet sich dieser noch in der Einschaltfarbzusammensetzung, werden Sie mit Sicherheit einige Probleme bekommen. Ein kleiner Tip am Rande: Mit

```
PRINT CHR$(22)+CHR$(0)
```

läßt sich der Spuk normalerweise beenden.

Bei diesem Befehl handelt es sich um das Ein- und Ausschalten der Transparent-Option, einer besonderen Eigenschaft des CPC 464. Dabei werden die Bildpunkte, die normalerweise mit der Hintergrundfarbe gesetzt würden, nicht gesetzt und nur die mit der Schriftfarbe zu setzenden Punkte werden in den Grafikspeicher übernommen. Dadurch kommt ein Überlagerungseffekt zustande, der es uns ermöglicht, Zeichen übereinander zu drucken. Wollen wir ein Zeichen aus zwei anderen durch Übereinanderdrucken schaffen, ist dies mit der Funktion CHR$ möglich. Sie geben mit demselben LOCATE-Befehl zweimal hintereinander verschiedene Zeichen an derselben Position aus, und schon haben Sie ein neues Bildschirmsymbol erstellt. Dies ist in manchen Fällen einfacher, als mit SYMBOL neue Zeichen zu definieren.

Zum Schluß noch einmal die Tabelle der Systemsteuerzeichen.

Tabelle 3.2 Nützliche Systemsteuerzeichen

Funktion	Erklärung
CHR$(4)	Ersatz für MODE-Befehl
CHR$(7)	Bell; es ertönt ein Piepser im Lautsprecher
CHR$(8)	Cursor links
CHR$(9)	Cursor rechts
CHR$(10)	Cursor runter; ggf. Scrollen nach oben
CHR$(11)	Cursor rauf; ggf. Scrollen nach unten
CHR$(12)	CLEAR/HOME; entspricht CLS
CHR$(13)	RETURN; Sprung auf Zeilenanfang
CHR$(14)	Ersatz für PAPER
CHR$(15)	Ersatz für PEN
CHR$(19)	Löschen von Anfang des WINDOWS bis zum Cursor
CHR$(20)	Löschen von Cursor bis Ende des WINDOWS
CHR$(22)	Transparent-Option EIN/AUS
CHR$(30)	HOME; Cursor auf Bildschirmanfang
CHR$(31)	Ersatz für LOCATE

Mit Hilfe der Systemsteuerzeichen können wir Texte relativ zu anderen Texten positionieren. Basis für die Berechnung der neuen Anfangsposition ist das Ende des vorherigen Textes. So schön dies beim Bildschirmaufbau sein mag, hilft es uns doch wenig, wenn (zum Beispiel in einer Tabelle) verschieden lange Werte mit gleicher Endposition untereinander ausgegeben werden sollen. Wir müßten dann nämlich die Anzahl unserer Cursorbewegungen in Abhängigkeit von der Länge des zweiten Textes variieren können. Auch die Methoden der absoluten Adressierung können uns nur einen identischen Anfangspunkt der zweiten Textspalte unserer Tabelle sichern. Um hier einen gleichmäßigen Abschluß zu erzielen, müßten wir die X-Koordinate unseres LOCATE-Befehls ändern, oder beim Arbeiten mit TAB und "," eine unterschiedliche Anzahl Leerzeichen einschieben. Die Berechnung dieser führt dann meist zu mehrzeiligen komplexen STRING-Ausdrücken, in der Fachsprache auch als "Langholz" bezeichnet. Dies kann durch die Anwendung der formatierten Textausgabe vermieden werden, eine Technik, auf die wir nun zu sprechen kommen.

3.3.5 Die formatierte Ausgabe

Unter formatierter Ausgabe versteht man den Ausdruck eines Textes und einer Variablen in einem vorher vorgegebenen Ausgabeformat. Dies ermöglicht es, den gewünschten Teil eines Textes oder einer numerischen Variablen sichtbar zu machen und uninteressante Ausgaben zu unterdrücken. Ein kurzes Beispiel soll dies verdeutlichen:

In einer Steuerberechnung, in der vom Verkaufspreis die Mehrwertsteuer abgezogen werden soll, ist es naturgemäß vollkommen uninteressant, welchen Wert die dritte oder vierte Nachkommastelle annimmt. Die Ausgabe eines hochpräzisen Zahlenbandwurms würde die Lesbarkeit der Berechnung nur erschweren, ohne zusätzliche Informationen zu bringen. Die Ausgabe einer zu langen Zahl könnte darüberhinaus bei einer Tabellendarstellung dazu führen, daß die letzten Zahlen bereits in die nächste Spalte reichen und dort befindliche Werte durch Überschreiben zerstören. Das Problem liegt also darin, die ausgegebenen Werte um die entsprechende Länge zu verkürzen. Dies könnten wir erreichen, indem wir auf den entsprechenden Variablen eine der Funktionen

ROUND, INT oder FIX

anwenden würden. Dadurch würden wir allerdings den Wert der entsprechenden Variablen verändern. Außerdem würde das Runden auf mehrere Stellen nach dem Komma bei Verwendung dieser Befehle bereits einige Gedankenarbeit erfordern. Zur Lösung derartiger Probleme stellt das CPC-BASIC das

PRINT-USING-Kommando

zur Verfügung. Das PRINT-USING-Kommando besteht aus einem einfachen PRINT-Kommando mit einer angehängten USING-Klausel.

Diese Klausel hat es jedoch in sich, denn sie erlaubt es uns, ein fast beliebiges Ausgabeformat festzulegen. Was das heißt, sollen zwei Beispiele zeigen. Geben wir

```
a=27.75:PRINT a
```

ein, erhalten wir den Wert 27.75 auf dem Bildschirm ausgedruckt. Drucken wir dagegen a unter der USING-Bedingung durch

```
a=27.75:PRINT USING"##.#";a
```

aus, so erscheint der Wert 27.8 auf dem Bildschirm. Der Wert unserer Variablen ist also beim Ausdruck auf eine Nachkommastelle gerundet worden. Die Variable selbst blieb jedoch unverändert, wie wir mit einem einfachen PRINT schnell überprüfen können. Beim nächsten Beispiel zeigt sich noch ein anderer Effekt des angehängten USING. Durch

```
a=27.75:b=7.395
PRINT USING"##.#";a:PRINT USING"##.#";b
```

erhalten wir die Werte 27.8 und 7.4 auf dem Bildschirm. Diese werden aber nun nicht linksbündig ausgegeben, sondern schön mit dem Dezimalpunkt untereinander, genauso, wie wir es in einer Tabellendarstellung benötigen und oben gefordert haben.

Schauen wir uns den PRINT-USING-Befehl bzw. das angehängte USING näher an! Das USING-Kommando besteht aus 3 Teilen. Am Anfang steht der USING-Befehl, am Ende die Liste der unter diesem Kommando auszugebenden Variablen. In der Mitte befindet sich in Anführungszeichen die Angabe des Ausgabeformats. Hierin können bestimmte Zeichen stehen, die dann mit den Variablen ausgegeben werden. Geben wir

```
a=27.75:PRINT USING "DM ##.#";a
```

ein, erhalten wir als Ausgabe den String "DM 27.8". Vor dem Druck des Werts wurde also das in der USING-Klausel angegebene Zeichen mit ausgegeben. Wir können dieses natürlich auch nachstellen. Mit

```
a=27.75:PRINT USING "##.# DM";a
```

erhalten wir ein nachstehendes DM hinter dem Zahlenwert. Bestimmte Zeichen haben jedoch auch eine Sonderfunktion, auch das schon oft verwendete Doppelkreuz. Es stellt einen Platzhalter für numerische Variablen dar. Die nachfolgenden Variablen werden in diesem Format ausgegeben. Ein "#" vor dem Dezimalpunkt symbolisiert eine Vorkommastelle; ein "#" nach dem Punkt leistet dasselbe für die Nachkommastellen der Variablen. Mit der Spezifikation "##.#" haben wir damit für die nachstehenden Variablen eine Nachkomma- und zwei Vorkommastellen festgelegt. Besitzt unsere Variable mehr Stellen nach dem Komma, als durch die USING-Klausel angegeben wurde, wird auf die letzte Stelle, hier die erste Nachkommastel-

le, gerundet. Ist der auszugebende Wert dagegen größer als erlaubt, hier z.B. drei Vorkommastellen, wird das Überlaufzeichen "%" und der volle Variablenwert ausgegeben. Als Anfangspunkt der Ausgabe wird dabei der Beginn der Spezifikation gewählt; das heißt, der Dezimalpunkt verschiebt sich im Verhältnis zu kleineren Variablen um die überschüssigen Stellen nach rechts. Dadurch wird natürlich unsere Tabelle nicht mehr spaltenrichtig ausgegeben. Wir müssen darauf achten, daß wir eine genügende Anzahl Vorkommastellen mit dem Doppelkreuz reservieren. Mit der Angabe der Ausgabestellen und etwaiger zusätzlicher Ausdrücke sind unsere Möglichkeiten noch nicht erschöpft. Als weiteren Zusatz können Vorzeichen definiert werden. Ein "+" gibt dabei immer das Vorzeichen aus. Es erscheint ein "+", falls der Zahlenwert positiv ist, ein "-" bei negativen Zahlen. Das "+" kann dabei vor oder nach den Spezifikatoren stehen.

```
PRINT USING "DM ###.##+";a
```

gibt folgenden Ausdruck aus: "DM 27.8+". Die nicht benötigten Felder werden durch Leerzeichen aufgefüllt. Das nachgestellte "+" bewirkt die Ausgabe eines "+", weil die Variable positiv ist. Das DM-Zeichen wird vor der Variablenausgabe auf den Schirm gebracht, und auch die Leerstelle zwischen "DM" und den Spezifikatoren wird mit ausgegeben. Leerräume werden im PRINT-USING-Kommando mit berücksichtigt! Die beiden Leerzeichen ergeben sich somit als Summe aus dem definierten Space und der einen nicht benötigten Stelle des Ausgabefelds. Würden wir in diesem Format den Wert 7.8 ausgeben, so hätten wir also drei Leerzeichen zwischen Zahlenwert und DM-Kürzel.

Etwas anders dagegen wirkt das "-". Ein vorgestelltes Minus hat den gleichen Effekt wie andere Zeichen, es wird permanent vor dem Zahlenwert ausgegeben. Setzen wir dagegen ein "-" hinter die Kreuze, so erscheint es nur bei negativen Zahlen. Bei positiven Werten wird es durch ein Leerzeichen ersetzt.

Wir kommen nun zu einigen Spezialitäten. Ein vorangestelltes "*" bewirkt die Ausgabe desselben. Zwei Sternchen dagegen füllen die nicht benötigten Stellen auf. Dabei werden vorab zwei Sternchen ausgegeben.

```
PRINT USING"**######";a
```

bringt uns die Ausgabe "*****28" auf den Schirm. Die 5 Sternchen setzen sich aus 2 permanent ausgegebenen Sternchen und den 3 freien Stellen zusammen. Da wir keine Nachkommastellen angegeben hatten, wurde a auf 28 aufgerundet. Dies wird zum Beispiel beim Ausdruck von Bilanzen etc. benutzt, um ein nachträgliches Ändern der Werte zu verhindern. Ein vorangestellter "$" gibt das Dollarzeichen aus.

Für die Arbeit mit großen Zahlen stellt uns das PRINT-USING-Kommando zwei spezielle Spezifikatoren zur Verfügung. Mit dem "," können wir nach drei Stellen ein Komma ausgeben, was das Abzählen längerer Zahlen verbessert. Setzen wir a auf 1000000 und geben diesen Wert dann mit

```
PRINT USING "##,###,###"
```

aus, erhalten wir als Ergebniss " 1,000,000". Die unbenutzte Stelle wurde wieder durch Leerzeichen aufgefüllt, und die Kommas werden stellenrichtig eingefügt. Bei sehr großen Zahlen bietet sich die Exponentialschreibweise an. Dies erreichen wir beim CPC durch Eingabe von vier Hochpfeilen nach den Doppelkreuzen. Der Computer gibt dann die Zahl mit einer Vorkommastelle und Exponentenangabe aus. In unserem Beispiel würde bei zwei Kreuzen die Million als " 1E6" auf dem Bildschirm dargestellt.

Das Hauptanwendungsfeld für den PRINT-USING-Befehl ist die formatierte Zahlenausgabe. Aber auch beim Arbeiten mit Strings kann er benutzt werden. Eine Anwendungsmöglichkeit haben wir schon gesehen, das Einfügen von Leerzeichen. Eine andere Variante besteht darin, mit dem "!" nur das erste Zeichen eines Strings auszugeben. Definieren wir a$ als "abcd", können wir mit

```
PRINT USING "!";a$
```

nur das erste Zeichen dieser Zeichenkette ausgeben.

Mit der formatierten Ausgabe können wir Zahlen und Strings in einem vorher definierten Ausgabeformat, gegebenenfalls unter Einfügung von weiteren Zeichen auf dem Bildschirm darstellen. Wir wollen uns nun damit beschäftigen, wie wir ganze Textblöcke auf dem Bildschirm positionieren können.

3.3.6 WINDOWS

Der WINDOW-Befehl ist eines der stärksten Kommando im Zusammen spiel zwischen Text-Kommandos und Bildschirmaufbau. Durch da WINDOW-Kommando bietet das CPC-BASIC dem Benutzer eine Vielzah von Möglichkeiten, die sonst nur auf größeren Maschinen verfügba sind. Wir haben uns mit WINDOWS bereits im Kapitel über Textver arbeitung etwas näher beschäftigt. Wir wollen diese Kenntniss jetzt vertiefen und um Anwendungsmöglichkeiten bei der grafische Darstellung erweitern.

3.3.6.1 Einführung in die WINDOWS

Im Prinzip handelt es sich bei einem WINDOW (= Fenster) um eine Bildschirm im Bildschirm. Ein Teil des Bildschirms wird als eige ner Unterbildschirm, als sogenanntes Bildschirmfenster definiert In diesem laufen nun alle Bildschirmfunktionen, wie das Setze von Schrift- oder Hintergrundfarbe, das Löschen des Bildschirm oder das Positionieren des Cursors, genau wie auf dem Hauptbild schirm ab. Der Unterschied besteht darin, daß Kommandos, die de Teilbildschirm betreffen, den Hauptbildschirm jedoch unberühr lassen. Damit ist es jetzt zum Beispiel möglich, auf den obere Teil des Bildschirms eine feststehende Erklärung auszugeben, wäh rend im unteren Teil ein Text oder eine Variablenliste erscheint Auch eine Dreiteilung des Bildschirms in die Teile Titel, Ausga beteil und Eingabeteil ist möglich.

Außerdem kann das WINDOW-Kommando gut benutzt werden, um farbig Felder auf dem Bildschirm darzustellen. Anwendungsmöglichkeite bieten sich hier im Bereich der grafischen Statistik oder bei Hervorheben von Texten. Doch zunächst noch einmal zur Definitio des Kommandos: Das WINDOW-Kommando besteht aus fünf Teilen un hat die Form:

```
WINDOW #a,b,c,d,e

a = Nummer des WINDOWS
b = linke Spalte
c = rechte Spalte
d = oberste Zeile
e = unterste Zeile
```

Das WINDOW wird also als Rechteck durch die Angabe seiner Eck-Koordinaten bestimmt. Bezugsgröße ist dabei der Gesamtbildschirm. Damit ist das WINDOW-Kommando abhängig vom eingeschalteten MODE. Ein Beispiel mag dies verdeutlichen:

Bei

```
WINDOW #1,1,20,1,12
```

wird in MODE 0 ein Bildschirmfenster eröffnet, das die obere Hälfte des Gesamtbildschirms überdeckt. Da im MODE 0 jede Zeile aus zwanzig Zeichen besteht, reicht das WINDOW über die volle Zeilenbreite. Die Eckwerte 1 und 12 für die Zeilenzahl im WINDOW definieren die Höhe für die obere Bildschirmhälfte. Im MODE 1 dagegen, würde dieses Bildschirmfenster nur das obere linke Viertel des Gesamtbildschirms einnehmen. Im MODE 2 schließlich schrumpft es auf ein Achtel der verfügbaren Bildschirmfläche zusammen.

Auf die mit dem WINDOW-Kommando definierten Bildschirmfenster können wir nun mit den üblichen Bildschirmbefehlen zugreifen. Für die Ansteuerung verwenden wir dabei das Konzept der

```
STREAM EXPRESSION.
```

Zwischen unsere Befehle und die jeweiligen Werte fügen wir ein "#" und die Nummer des WINDOWS ein, auf das der Befehl wirken soll. So wird zum Beispiel mit

```
PEN #1,1
```

die Schriftfarbe des WINDOWS mit der Nummer eins auf Farbregister eins gesetzt.

```
PAPER #2,3
```

legt die Hintergrundfarbe des zweiten WINDOWS auf Farbregiste drei fest. Für die Wahl der Farbregister gelten dabei die im Ka pitel 3.2.2 gemachten Bemerkungen. Auch hier ist also die Farb wahl MODE-abhängig.

```
CLS #4
```

löscht das vierte WINDOW auf die Hintergrundfarbe, die zuvor mi einem PAPER#4,x festgelegt wurde. Im Zweifelsfall wird hier da Farbregister 0 benutzt.

```
LOCATE #1,2,3
```

schiebt den Cursor in die zweite Spalte und dritte Zeile de WINDOWS Nummer 1. Die Position 1,1 ist dabei nicht mehr, wie be dem normalen LOCATE-Kommando die obere linke Bildschirmecke sondern die obere linke Ecke des zuvor definierten WINDOWS.

Bis jetzt sind wir davon ausgegangen, daß es sich bei dem Bild schirm und den WINDOWS um zwei verschiedene Dinge handelt. Die ist jedoch so nicht richtig. Auch mit dem normalen PRINT- ode LOCATE-Kommando wird nämlich ein WINDOW angesprungen, das WINDO Nummer 0. Um die Eingabe zu vereinfachen, braucht dieses nich bei PRINT oder einer ähnlichen Ausgabe angegeben zu werden. De finieren wir also mit

```
WINDOW #0,1,20,1,10
```

WINDOW Nummer 0 neu, so reichen auch die mit einem normalen PRIN gemachten Ausgaben nicht mehr über den gesamten Bildschirm. Ei gewißer Unterschied zwischen dem WINDOW Nr. 0 und den andere WINDOWS mit den Nummern eins bis sieben besteht dennoch, denn au WINDOW 0 werden die Systemmeldungen, wie

```
Ready
```

oder die Fehlermeldungen ausgegeben und auch die Eingabe der Kom mandos des Benutzers, wie

```
RUN etc.
```

erfolgen hier. Doch auch dieser Zusammenhang ist nicht unumstöß

lich. Mit

```
WINDOW SWAP
```

können wir zwei Text-WINDOWS gegeneinander austauschen. So werden nach WINDOW SWAP 0,1 die Nachrichten des BASICS nicht mehr auf dem gesamten Bildschirm, sondern nur noch auf dem zuvor definierten WINDOW 1 ausgegeben. Definieren wir dieses mit

```
WINDOW #1,1,40,21,25
```

so bleiben die gegebenenfalls störenden Ausgaben auf die unteren vier Bildschirmzeilen beschränkt. Dies hat mannigfaltige Vorteile. Eine Anwendungsmöglichkeit bietet sich zum Beispiel im Zusammenhang mit der Kassettenbearbeitung. Wollen wir zum Beispiel in einem Programm, wie wir es im Kapitel über die Datenverwaltung getan haben, Daten vom Programm gesteuert automatisch laden, so wird normalerweise die Meldung

```
PRESS PLAY THEN ANY KEY
```

ausgegeben. Wenn der Cursor für WINDOW Nr. 0 ungünstig steht, zerstört diese Meldung damit möglicherweise einen gerade ausgegebenen Text. Dies läßt sich nun mit dem WINDOW SWAP-Kommando vermeiden. Nach

```
WINDOW #2,1,40,21,25

WINDOW SWAP 0,2
```

werden derartige Ausgaben in die unteren vier Bildschirmzeilen verbannt. Eine andere sinnvolle Anwendung ergibt sich im Zusammenhang mit der Fehlerbehandlung. Hier wäre es sehr wünschenswert, die Zeilen, in denen man Fehler vermutet, gleichzeitig zu den Ausgaben des Programms auf dem Bildschirm zu haben. Gerade größere und damit meist auch textintensivere Programme schieben in einer Vielzahl von Ausgaben die gelisteten Programmzeilen schnell aus dem Bildschirm wieder heraus. Eine Lösung können wir auch hier mittels der Befehle WINDOW und WINDOW SWAP erreichen. Unter der Annahme, daß uns fünf Programmzeilen zur Kontrolle genügen, definieren wir:

```
WINDOW #1,1,40,1,5
WINDOW #2,1,40,6,25
WINDOW SWAP 0,2
```

Nun können wir mit

```
LIST 100-120,#1
```

die fehlerhaften Programmzeilen im WINDOW 1 ausgeben. Mit de WINDOW SWAP-Kommando haben wir die WINDOWS 0 und 2 vertauscht, s daß die mit einem normalen PRINT erzeugten Ausgaben und auch di Meldungen des BASICs auf den unteren Bildschirmteil beschränk bleiben. Dies funktioniert natürlich nur, sofern das Program seinerseits nicht auch wieder WINDOWS definiert. Doch kommen wi zurück zur Verbindung von WINDOWS und Text.

Wir hatten schon gesehen, daß wir das WINDOW-Kommando im Zusam menhang mit dem CLS-Befehl dazu benutzen können, Teilbereiche de Bildschirms zu löschen. Dieses Löschen kann nun auf zwei Arte erfolgen. Zum einen können wir für den Gesamtbildschirm und da WINDOW dieselbe Hintergrundfarbe nehmen. In diesem Fall wird nu die Schrift aus einem Teil des Gesamtbildschirms herausgelöscht Eine andere Möglichkeit besteht darin, für das WINDOW eine ander Hintergrundfarbe zu wählen. Damit wird nicht nur der alte Tex gelöscht, sondern gleichzeitig ein Quadrat mit der neuen Hinter grundfarbe auf den Bildschirm gesetzt. Die folgenden beiden Ein zeiler verdeutlichen den Zusammenhang. Der obere löscht dabei au eine neue Hintergrundfarbe, der untere behält die Grundfarbe bei Zur besseren Sichtbarkeit des Effektes sollten diese beiden Zei len nacheinander in der Bildschirmmitte eingegeben werden.

```
WINDOW#1,1,20,10,20:INK 2,6:PAPER#1,2:CLS#1
WINDOW#1,1,20,10,20:PAPER#1,0:CLS#1
```

Nach diesem Prinzip können wir nun Texte verschiedenfarbig unter legen. Damit ist es uns möglich, Ausgaben, die mit verschiedene Sachverhalten zu tun haben, auch durch ihre Hintergrundfarbe von einander abzuheben. Auch eine Strukturierung des Bildschirms i mehrere farblich voneinander abgehobene Teilbereiche, die ver schiedenen Zwecken dienen, ist damit möglich. Durch die klar Trennung von Eingaben und Bildschirmausgaben wird die Übersich über das Bildschirmgeschehen wesentlich besser.

Ein Beispiel für diese Technik, ist das von uns oft benutzte Drei-WINDOW-System. Wir unterteilen dabei den Bildschirm in drei Teilbereiche. In Zeile eins bis drei geben wir Titel und Überschriften aus. Die Zeilen vier bis zweiundzwanzig dienen der Hauptausgabe. In den untersten drei Bildschirmzeilen schließlich läuft die gesamte Eingabe, einschließlich Fehlerkorrektur etc., ab. Der obere und untere Teil des Bildschirms ist dabei andersfarbig unterlegt, was eine Umrahmung des Hauptbildschirms zur Folge hat.

Einen fast klassischen Fall der Anwendung dieser Technik stellt das nachfolgende Programm Überschußrechnung dar. Bei der Überschußrechnung geht es darum, die Einnahmen und Ausgaben einer Periode einander gegenüber zu stellen, um einen Überblick über die finanzielle Situation zu erhalten. Dies geschieht meist in tabellarischer Form, wobei nacheinander die einzelnen Posten abgefragt werden. Arbeitet man dabei mit nur einem Bildschirm, so wechseln sich die Tastatureingaben und Bildschirmausgaben ständig ab, der Bildschirm rollt nach oben, und gegebenenfalls sorgen Fehlermeldungen für weitere Verwirrung. Mit dem Drei-WINDOW-System werden diese Probleme vermieden. Wir erhalten im mittleren Teil unseres Bildschirms eine stehende Tabelle, im unteren Teil verarbeiten wir die Eingaben und Ausgaben, und im oberen Teil erklärt uns eine Überschrift, welches Programm wir vor uns haben.

Schauen wir uns das Programm etwas näher an: Die Zeilen 40 bis 90 enthalten die Initialisierung. Hier werden die variablen Größen festgelegt, desgleichen der MODE und die Rahmenfarbe eingeschaltet. Danach folgt die Definition der WINDOWS, die Festlegung der Farbregister und der Hintergrund und Vordergrundfarben. In Zeile 90 werden alle drei Bildschirme dann gelöscht. Wir verfügen nun über einen schwarzen Hauptbildschirm in der Mitte, der von zwei orangefarbenen Farbbändern umrahmt ist, das

DREI-WINDOW-SYSTEM.

```
10 REM ************************
20 REM ** Ueberschussrechnung **
30 REM ************************
40 DIM a$(20),e$(10),a(20),e(10)
50 MODE 1:BORDER 0
60 WINDOW#1,1,40,1,3:WINDOW#2,1,40,4,23:WINDOW#3,1,40,24,25
70 INK 0,27:INK 1,21:INK 2,1:INK 3,6
80 PAPER#1,2:PEN#1,3:PAPER#2,0:PEN#2,1:PAPER#3,2:PEN#3,3
90 CLS#1:CLS#2:CLS#3
100 REM ***************
110 REM ** Einnahmen **
120 REM ***************
130 LOCATE#1,13,2:PRINT#1,"E I N N A H M E N"
140 e$(1)="Lohn/Gehalt"+SPACE$(29)
150 e$(2)="Rente"+SPACE$(35)
160 e$(5)="Mietertraege"+SPACE$(28)
170 e$(4)="Zinsertraege"+SPACE$(28)
180 e$(3)="nebenberufliche Taetigkeit"+SPACE$(14)
190 e$(6)="Versicherungsleistungen"+SPACE$(17)
200 e$(7)="sonstige Einnahmen"+SPACE$(22)
210 PRINT#3,"Bitte geben Sie jetzt die einzelnen"
220 PRINT#3,"Posten ein!"
230 FOR i=1 TO 1000:NEXT i
240 FOR i= 1 TO 7:LOCATE#2,1,i+1:PRINT#2,e$(i)
250 PRINT#3,e$(i);:INPUT#3,e
260 s=s+e:s$=STR$(s)
270 e$(i)=LEFT$(e$(i),36-LEN(STR$(INT(e))))+STR$(e)
280 LOCATE#2,1,i+1:PRINT#2,e$(i)
290 z$=SPACE$(21-LEN(STR$(INT(s))))
300 se$="Gesamteinnahmen"+z$+s$
310 LOCATE#2,1,20:PRINT#2,se$
320 NEXT i:e=s
330 PRINT#3,"o.k.  j/n ?
"
340 a$=LOWER$(INKEY$):IF a$<>"n" AND a$<>"j" THEN 340
350 IF a$="n" THEN RUN
360 REM **************
370 REM ** Ausgaben **
380 REM **************
390 CLS#1:CLS#2:s=0
400 LOCATE#1,12,2:PRINT#1,"A U S G A B E N"
410 a$(1)="Miete"+SPACE$(35)
```

```
420 a$(2)="Heizung"+SPACE$(33)
430 a$(3)="Elektro/Wasser"+SPACE$(26)
440 a$(4)="Raten/Abzahlung"+SPACE$(25)
450 a$(5)="Fahrtkosten/Auto"+SPACE$(24)
460 a$(6)="Versicherungszahlungen"+SPACE$(24)
470 a$(7)="Telefon,Porto"+SPACE$(27)
480 a$(8)="Kleidung"+SPACE$(32)
490 a$(9)="Lebensmittel"+SPACE$(28)
500 a$(10)="Genussmittel"+SPACE$(28)
510 a$(11)="Reinigung/Hygiene"+SPACE$(26)
520 a$(12)="Geschenke"+SPACE$(30)
530 a$(13)="Hobby"+SPACE$(35)
540 a$(14)="Sparen"+SPACE$(34)
550 a$(15)="sonst.Ausgaben"+SPACE$(26)
560 PRINT#3,"
Bitte geben Sie jetzt die einzelnen"
570 PRINT#3,"Posten ein."
580 FOR i=1 TO 1000:NEXT i
590 FOR i= 1 TO 15:LOCATE#2,1,i+1:PRINT#2,a$(i)
600 PRINT#3,a$(i);:INPUT#3,a
610 s=s+a:s$=STR$(s)
620 a$(i)=LEFT$(a$(i),36-LEN(STR$(INT(a))))+STR$(a)
630 LOCATE#2,1,i+1:PRINT#2,a$(i)
640 z$=SPACE$(22-LEN(STR$(INT(s))))
650 sa$="Gesamtausgaben"+z$+s$
660 LOCATE#2,1,20:PRINT#2,sa$
670 NEXT i
680 PRINT#3,"o.k.  j/n ?
"
690 a$=LOWER$(INKEY$):IF a$<>"n" AND a$<>"j" THEN 690
700 IF a$="n" THEN CLS#2:GOTO 390
710 a=s
720 CLS#3
730 CLS#2:PRINT#2:PRINT#2:PRINT#2,es$:PRINT#2,ea$
740 IF e>=a THEN PRINT#2,"Einnahmenueberschuss :";e-a;" DM":GOTO 760
750 PRINT#2,"Ausgabenueberschuss :";a-e;" DM"
760 GOTO 760
```

Im nachfolgenden Teil werden nun zuerst die Einnahmen, dann die Ausgaben abgefragt und schließlich wird die Endsumme ausgegeben. Da sich die ersten beiden Teile von der Programmtechnik her im wesentlichen entsprechen, werden wir nur auf den ersten eingehen. Zeile 130 gibt die Überschrift aus, und zwar im WINDOW NR. 1 in der mittleren Zeile. Die nachfolgenden Zeilen bis Zeile 200 legen die Text für die Eingabe fest. Die angefügten

```
SPACE$()
```

dienen dabei dazu, diese STRINGS auf eine Standardlänge von vierzig Zeichen zu bringen. Was nun folgt ist ein Wechselspiel zwischen Bildschirm WINDOW Nr. 2 und Nr. 3. Dabei wird zunächst jeder einzelne Posten im WINDOW Nr. 3 abgefragt, und daraufhin sofort in das WINDOW Nr. 2 stellenrichtig übernommen. Dazu dienen auch die etwas kompliziert aussehenden Ausdrücke in Zeile 260 und 270. Unser vierzig Zeichen langer Eingabe-STRING wird dabei zunächst einmal um die Zeichenzahl der Eingabe verkürzt. Die zuvor abgezogenen Leerstellen werden dann durch den Eingabewert wieder aufgefüllt. Mit diesem Trick wird sichergestellt, daß alle Eingaben den Dezimalpunkt an derselben Stelle haben. Dieser STRING e$(i) wird nun im WINDOW Nr. 2 in der richtigen Zeile ausgegeben. Ähnliches passiert in Zeile 280-300 mit den Gesamteinnahmen. Diese wurden in Zeile 250 durch den neuen Eingabewert erhöht und müssen nun zu einem neuen STRING zusammengebastelt werden, bevor sie in Zeile 300 auf der Bildschirmzeile 18 ausgegeben werden. Die ganze Prozedur erfolgt dabei in einer Schleife, die von Zeile 230-320 einschließlich reicht. Nach einer Kontrollabfrage wiederholt sich das ganze für die Ausgaben. Das Endergebnis wird dann in den Zeilen 740 und 750 präsentiert.

Nachdem wir uns ausgiebig mit den Zusammenhängen zwischen WINDOWS und Texten beschäftigt haben, wenden wir uns nun zwei Bereichen zu, in denen man auf den ersten Blick gar keine Anwendung für die WINDOW-Technik erwarten sollte:

Dem Schaffen von Bedienfeldern und der grafischen Statistik.

Grundsätzlich gibt es zwei Arten, den Zusammenhang zwischen der Tastatur und bestimmten Programmfunktionen zu verdeutlichen. Die eine läuft über teilweise seitenlange Erklärungstexte; daß heißt, es werden zu jeder Taste die entsprechenden Funktionen aufgelistet oder es wird die Funktion der Taste entsprechend beschrieben. Die andere Methode läuft über Bedienfelder. Dabei wird eine

Kopie der Tastatur oder eines Teils der Tastatur auf dem Bildschirm ausgegeben, und an dieser die Funktion erläutert. Dies bietet sich insbesondere dann an, wenn hinter allen Tasten oder Funktionen ein gleichartiges Prinzip liegt. Ein einfaches Beispiel dafür ist ein Piano-Programm, indem eine Klaviatur auf dem Bildschirm dargestellt wird, und auf jeder Taste der entsprechende Buchstabe der Rechnertastatur angegeben wird. Diese Methode hat gegenüber der ersteren mehrere Vorteile. Zum einen entfallen langwierige Erklärungen; zum anderen kann das Programm nun seine Funktionsweise selbständig demonstrieren, indem es zu den gespielten Tönen gleichzeitig auch noch die Tastaturfelder blinken läßt. Darüberhinaus spart die zweite Variante meistens auch noch Speicherplatz. Das Programm Farbfelddemo demonstriert das Prinzip.

```
10 REM ******************
20 REM ** Farbfelddemo **
30 REM ******************
40 INK 0,0:INK 1,24:INK 2,5:INK 3,6:INK 4,18
50 INK 5,12:INK 6,7:INK 7,10:INK 8,2:INK 9,24
60 PAPER 0:MODE 0:BORDER 0
70 PEN 2:LOCATE 4,5:PRINT"F A R B F E L D"
80 FOR i= 0 TO 9 STEP 3
90 FOR k= 0 TO 3 STEP 3
100 WINDOW#1,5+i,7+i,10+k,12+k
110 PAPER #1,(2+i/3+4*k/3):CLS#1
120 NEXT k,i
130 GOTO 130
```

Zuerst werden in den Zeilen 40 bis 50 die Farbregister gesetzt. Danach erfolgt der Übergang in MODE 0, die Löschung des Bildschirms auf schwarz. Die Zeilen 70 bis 110 erzeugen nun das eigentliche Bedienfeld. Mittels der Doppelschleife, die über I und K läuft, wird WINDOW Nr. 1 achtmal hintereinander an einen anderen Platz gesetzt, viermal nebeneinander und in zwei Reihen untereinander. In Abhängigkeit von I und K wird dann in einhundert jeweils eine andere Farbe ausgewählt und das WINDOW daraufhin gelöscht. Auf dem Bildschirm erscheinen dann acht verschiedenfarbige Kästchen mit einer Größe von drei mal drei Zeichen. In diese Kästchen können wir nun mit PRINT und LOCATE Buchstaben oder Zeichen schreiben, die die entsprechenden Tasten repräsen-

tieren. Außerhalb der Kästchen besteht die Möglichkeit, weitere Erklärungen abzugeben. Außerdem wäre es natürlich möglich, ein anderes WINDOW - zum Beispiel WINDOW Nr. 2 - und den oberen Teil des Bildschirms zu definieren und darin rollende Erläuterungstexte auszugeben, wobei gleichzeitig die entsprechende Taste blinken würde. Das Blinken der Taste wird dabei wiederum dadurch erreicht, daß wir für die Hintergrundfarbe ein Farbregister wählen, auf das wir zwei Farben gelegt hatten. Wollen wir zum Beispiel die zweite in der oberen Reihe blinken lassen, so definieren wir einfach das Farbregister 3 um. Nach

```
INK 2,5,6
```

fängt das entsprechende Farbfeld an zu blinken.

```
INK 2,5
```

stellt den Ausgangszustand wieder her. Eine andere Möglichkeit wäre, die Schriftfarbe mit einer blinkenden INK zu schreiben und damit auf die einfarbige Hintergrundfläche ein blinkendes Zeichen zu setzen.

Wir sehen:

Die Möglichkeiten und Varianten im Spiel mit Farbe, WINDOWS und Text sind sehr vielfältig und auch hier gilt: Übung macht den Meister. Deshalb sollten Sie ruhig einmal an dem einen oder anderen Wert herumspielen und sich von dem Ergebnis beeindrucken lassen.

3.3.6.2 Grafiken mit WINDOWS

Ein weiterer Anwendungsbereich dieser Technik ist die grafische Statistik. Bei der grafischen Statistik geht es darum, Zahlen und das Verhältnis von Zahlen mit grafischen Mitteln darzustellen. Typische Beispiele für derartige Grafiken sind Umsatzzahlen, Zahlen über die wirtschaftliche Entwicklung, die Inflationsrate etc.

Das wohl häufigste Darstellungsmittel ist die Säulengrafik. In dieser Darstellungsart entsprechen den Zahlenwerten verschieden hohe farbige Balken. Wir wollen uns nun damit beschäftigen, wie

wir auf sehr schnelle Art und Weise mit dem CPC und den WINDOW-Kommandos derartige Farbgrafiken erstellen können. Auch dazu haben wir ein kleines Demonstrationsprogramm geschrieben: Säulengrafik.

Das Prinzip ist dasselbe wie im letzten Programm. Ein WINDOW wird gesetzt und danach auf die entsprechende Hintergrundfarbe gelöscht. Der Unterschied besteht nun darin, daß wir nicht mehr mit gleich hohen und gleich breiten WINDOWS arbeiten, sondern die Ausdehnung des WINDOWS in Y-Richtung von den entsprechenden Zahlenwerten abhängig machen. Als Basis nehmen wir dafür Umsatzwerte für die Monate Januar bis Dezember, die Sie sich selber ausdenken können.

Nachdem die Monatsnamen in Zeile 70 eingegeben wurden, geht es in eine Eingabeschleife. In diese werden für die Zahlen von eins bis zwölf die Monatsumsätze hereingeholt und in m(i) gespeichert. Die nachfolgende Schleife ist eine MIN-MAX-Schleife. In dieser wird das Minimum und das Maximum der Zahlenwerte festgestellt und in ma beziehungsweise mi gespeichert. Danach folgt die eigentliche Ausgaberoutine. In Zeile 180 werden die Farbregister gesetzt und die Umrandung auf Farbe 0 gelöscht. Danach kommt die eigentliche Ausgabeschleife. Zunächst wird mit der uns ja schon bekannten MODULO-Funktion die Farbe festgestellt. Wegen der eingeschränkten Benutzbarkeit der Farbregister im MODE 1, in dem dieses Programm arbeitet, müssen wir nämlich die Farbregister mehrfach benutzen. In Zeile 210 wird dann die Höhe des Balkens errechnet, die maximal zugelassene Höhe beträgt dabei sechzehn Zeilen. Danach wird das WINDOW gesetzt, gelöscht und das nächste bearbeitet.

Dieses Programm stellt die Balken beginnend mit dem Wert 0 dar. Bei kleineren Änderungen auf einem ziemlich hohen Niveau kann diese Betrachtungsweise ziemlich störend sein. Es ist dann gegebenenfalls sinnvoller, nur den Schwankungsbereich für die Darstellung zu wählen. Dies erreicht man, indem man Zeile 210 wie folgt ändert:

```
210 h=((m(i)-mi)/(ma-mi))*16
```

```
10 REM *************************
20 REM ** Saeulengrafik WINDOW **
30 REM *************************
40 INK 0,27:INK 1,21:PAPER 0:PEN 1:BORDER 0
50 DATA Januar,Februar,Maerz,April,Mai,Juni
60 DATA Juli,August,September,Oktober,November,Dezember
70 DIM m$(12),m(12)
80 FOR i=1 TO 12:READ m$(i):NEXT i
90 CLS
100 FOR i= 1 TO 12
110 PRINT"Bitte geben Sie den Umsatz fuer den Mo-"
120 PRINT"nat ";m$(i);" ein";:INPUT m(i)
130 NEXT i
140 ma=0:mi=10000000
150 FOR i= 1 TO 12:ma=MAX(ma,m(i)):mi=MIN(mi,m(i)):NEXT i
160 CLS
170 LOCATE 7,2:PRINT"W I N D O W - G r a f i k"
180 INK 0,0:INK 1,2:INK 2,15:INK 3,21:BORDER 0
190 FOR i= 1 TO 12
200 f=(i MOD 3)+1:PAPER#4, f
210 h=m(i)/ma*16
220 WINDOW#4,3*i-1,3*i+1,20-h,20:CLS#4
230 NEXT i
240 PEN 2
250 LOCATE 2,22:PRINT" J  F  M  A  M  J  J  A  S  O  N  D"
260 GOTO 260
```

Man kann natürlich nicht nur den Minimalwert als Bezugspunkt wählen, sondern jeden beliebigen anderen Wert. Je nach Wahl des Bezugswertes verändert sich natürlich dann die Gestalt und auch die Wirkung des Diagramms. Auch hier gilt deshalb, wie bei den anderen Programmen: Spielen Sie doch einmal mit den entsprechenden Werten. Testen Sie doch einmal die Programme in den verschiedenen MODES durch. Sie sollten auch einmal die Farbgebung variieren, um ein gutes Gefühl für das Zusammenwirken der auf dem CPC verfügbaren Farben zu erhalten. Schließlich könnten Sie versuchen, ein Programm zu schreiben, das den Bildschirm vollständig umrahmt. Dieses müßte dann auch seitliche Teile des Bildschirms als eigenständige WINDOWS definieren. Der Effekt, der damit entsteht,

gleicht einem Bildschirm im Bildschirm und läßt sich sehr gut in Spielen verwenden.

3.3.7 Umdefinieren des Zeichensatzes

Der CPC 464 verfügt nicht nur über einen großartigen Zeichensatz, er hat ihn auch vorzüglich versteckt. Die über die Tastatur erreichbaren Zeichen sind auf das notwendigste beschränkt: große und kleine Buchstaben, Zahlen und ein paar Sonderzeichen. Dies ist weniger als die Hälfte der vom CPC ohne Änderungen darstellbaren Zeichen. Neben einer Blockgrafik, die auch Kurvenstücke und Schrägteile umfaßt, sind auch eine Menge Sonderzeichen und die für Spiele so dringend benötigten Raketen und Männchen enthalten. Der CPC hat also in punkto Zeichensatz einiges zu bieten. Das Problem besteht nur darin, diese Möglichkeiten auch schnell, das heißt möglichst über die Tastatur verfügbar zu machen. Mit dieser Problematik werden wir uns in dem Kapitel über die Tastatur näher beschäftigen. An dieser Stelle wollen wir uns zunächst mit dem Zeichensatz und daraus folgend mit einer Veränderung - der Schaffung neuer Zeichen - beschäftigen.

Schauen wir uns zunächst einmal den Zeichensatz an. Mit

```
FOR i=32 TO 255:PRINT CHR$(i);:NEXT i
```

wird er auf dem Bildschirm des CPC ausgegeben. Wir haben

```
224=256-32
```

frei definierbare Zeichen zur Verfügung. Die unteren 32 Characters enthalten die uns schon bekannten Steuerzeichen und sind nicht für die Zeichendarstellung geeignet. Was heißt nun frei definierbar? Frei definierbar bedeutet, daß wir jedes Zeichen vollkommen unabhängig neu definieren oder umdefinieren können. So können wir uns einen Zeichensatz zusammenbasteln, der anstelle des großen A ein Alpha vorweist, oder statt der arabischen Zahlen ägyptische Zeichen enthält. Ein komplettes kyrillisches Alphabet oder auch eine Vielzahl neuer Spielsymbolen - von Spielkartensymbolen über Laserbatterien bis hin zu neuen Zahlensymbolen - sind so realisierbar. Bevor wir uns praktisch mit den dazu benötigten Befehlen und einem sehr nützlichen Hilfsprogramm beschäftigen,

wollen wir noch einmal auf die Grundlagen eingehen.

Wie wir schon gesagt hatten, besteht der Bildschirm des CPC 464 aus 128000 einzelnen Bildpunkten, eingeteilt in 200 Zeilen zu je 640 Punkten. Stellen wir ein Zeichen auf dem Schirm dar, so nimmt dieses immer acht Zeilen, aber je nach MODE eine verschiedene Anzahl von Spalten ein. Diese Ausdehnung der Zeichen in der Breite wird durch eine eigene Routine im Betriebssystem geleistet. Der CPC verfügt also nicht über drei verschiedene Zeichensätze, sondern nur über einen für alle MODES einheitlichen Zeichensatz, der je nach MODE entsprechend ausgedehnt oder komprimiert wird. Jedes Zeichen ist dabei in einer Matrix aus acht mal acht Bildpunkten oder Pixeln definiert. Das Setzen beziehungsweise nicht Setzen eines Zeichens entspricht dabei einer Ja/Nein-Entscheidung und läßt sich damit in einem Bit im Speicher festhalten. Nehmen wir die Punkte einer Zeile, so benötigen wir dafür ein Byte. Um ein komplettes Zeichen zu speichern, benötigen wir somit vierundsechzig Bit oder acht Byte, wobei jede Zeile in einem Byte gespeichert ist. Im Innern des CPC sind diese acht Byte für jedes Zeichen in einem separaten Speicher, dem sogenannten

Zeichengenerator-ROM

abgespeichert. Das BASIC des CPC bietet jedoch die Möglichkeit, auf andere vom Benutzer frei definierbare Zeichen zurückzugreifen. Dies geschieht mit dem Kommandos

SYMBOL und SYMBOL AFTER

SYMBOL AFTER legt dabei fest, ab welchem Zeichen neu definiert werden soll. Wie wir gesehen haben, können wir maximal 224 Zeichen definieren. Beim Einschalten des Rechners sind nur die oberen 16, das heißt die Zeichen von 240 bis 255 umdefinierbar. Wie ein Blick in den Anhang des Handbuchs zeigt, handelt es sich dabei im wesentlichen um Spielsymbole, wie Männchen, Raketen etc. Wollen wir auch auf die Zeichen mit niedrigerem Code-Wert zugreifen, beispielsweise auf die Großbuchstaben des Alphabets, so müssen wir zunächst mit SYMBOL AFTER, in diesem Fall SYMBOL AFTER 64, den Zugriff hierauf ermöglichen. Der Wert nach SYMBOL AFTER enthält dabei immer die Character-Nummer, ab der nach oben hin unbeschränkt umdefiniert werden soll. Mit dem SYMBOL-Befehl geben wir nun unser neues Zeichen ein.

Das SYMBOL-Kommando schleppt einen ganzen Rattenschwanz an Zahlen hinter sich her. Die erste steht dabei für die Nummer des neu zu definierenden Zeichens. Die weiteren acht Zahlen sind die oben schon angesprochenen acht Byte, eins für jede Zeile unseres Zeichens. Die Eingabe dieser Zahlenwerte kann nun auf mehrere Arten erfolgen. Zum einen können wir sie binär eingeben, das heißt, wir geben das Symbol "&" und dann jeweils die acht Bits einer Zeile von links nach rechts, wobei eine Eins bedeutet, daß der Bildpunkt gesetzt ist, eine Null, daß die Hintergrundfarbe benutzt wird. Die andere Möglichkeit besteht darin, daß wir die Zeilenwerte direkt dezimal eingeben. Dies erfordert allerdings einige Rechenarbeit. Wir müssen nämlich dann die Umwandlung von der Binärzahl zur Dezimalzahl selbst vornehmen. Dazu multiplizieren wir die Bits einer jeden Zeile absteigend von links nach rechts mit 128, 64, 32, 16 8, 4, 2 und 1 und bilden dann darüber die Summe. Dies liefert uns den Zeilenwert für jede Zeile. Schauen wir die nachfolgende Tabelle an.

Tabelle 3.3

	128	64	32	16	8	4	2	1
1. Zeile	1	1	1	1	1	0	0	0
2. Zeile	1	1	1	1	1	0	0	0
3. Zeile	0	0	0	1	1	0	0	0
4. Zeile	0	0	0	1	1	0	0	0
5. Zeile	0	0	0	1	1	0	0	0
6. Zeile	0	0	0	1	1	0	0	0
7. Zeile	0	0	0	1	1	0	0	0
8. Zeile	0	0	0	1	1	0	0	0

In dieser Tabelle haben wir ein Winkelstück entworfen, wie wir es zum Beispiel beim Arbeiten mit der Blockgrafik brauchen können. Die Einsen symbolisieren dabei gesetzte Bildpunkte; bei "0" erhält der Bildpunkt die Hintergrundfarbe und wird damit gelöscht.Dieses Zeichen können wir nun mit dem SYMBOL-Befehl definieren. Dazu geben wir die Zeilenwerte nacheinander an. Wir erhalten dann das Kommando:

```
SYMBOL 240,&x11111000,&x11111000,&00011000,&x00011000,
&x00011000,&x00011000,&x00011000,&x00011000
```

Wir können diesen Zahlenwurm etwas abkürzen, indem wir die einzelnen Zeilenwerte ausrechnen. Dabei brauchen wir nur zwei Rechnungen auszuführen, da die Zeilen 1 und 2 und 3 bis 8 identisch sind.

```
&x11111000 = 1 * 128          &x00011000 = 0 * 128
           + 1 *  64                     + 0 *  64
           + 1 *  32                     + 0 *  32
           + 1 *  16                     + 1 *  16
           + 1 *   8                     + 1 *   8
           + 0 *   4                     + 0 *   4
           + 0 *   2                     + 0 *   2
           + 0 *   1                     + 0 *   1
           -------------                 -------------
                 248                           24
```

Als Befehl ergibt sich somit:

```
SYMBOL 240,248,248,24,24,24,24,24,24
```

Nun hatten wir mit mit dem Haken in Tabelle 3.3. noch ein relativ einfaches Zeichen gewählt. Gehen wir dagegen auf kompliziertere Zeichen über, so wird das Definieren neuer Characters schnell zur Sisyphosarbeit, und man verbringt mehr Zeit mit der Suche nach Rechenfehlern als mit dem eigentlichen Entwurf neuer Symbole. Das Definieren neuer Zeichen auf diese Art ist also eine ziemlich eintönige und lästige Sache und eignet sich damit geradezu perfekt für den Computereinsatz. Deshalb haben wir das Programm Zeichendefinition geschrieben. Mit diesem Programm wird das Entwerfen neuer Symbole zur wahren Freude.

Das Programm arbeitet mit zwei WINDOWS, die diesmal den Bildschirm nicht waagerecht, sondern senkrecht unterteilen. Im linken Teil des Bildschirms wird dabei ständig das neu entworfende Symbol ausgegeben, im rechten Teil dagegen die entsprechende Matrix aus Nullen und Einsen. Sie fahren nun mit einem JOYSTICK gesteuerten Cursor über den Bildschirm und können, wenn Sie über einer Zahl der Matrix angelangt sind, diesen durch einfachen Druck auf den Feuerknopf verändern. Einmaliges Drücken kehrt den entsprechenden Wert um, aus Null wird Eins und aus Eins wird Null. Mit nochmaligem Drücken befindet man sich wieder im Ausgangszustand. Nach jeder Änderung errechnet das Programm selbständig die jeweiligen Zeilenwerte und definiert dann mit SYMBOL

255,... das neue Zeichen, welches dann im linken WINDOW ausgegeben wird. Schauen wir uns das Programm etwas näher an.

```
10 REM ***********************
20 REM ** Zeichendefinition **
30 REM ***********************
40 INK 0,0:INK 1,15:INK 2,2:INK 3,21
50 MODE 1:BORDER 1
60 WINDOW#1,1,3,1,25:WINDOW#2,4,40,1,25
70 PAPER#1,0:PEN#1,3:PAPER#2,1:PEN#2,2
80 CLS#1:CLS#2
90 DIM m(8,8),w(8):x=2:y=2
100 FOR i= 1 TO 8:FOR j=1 TO 8
110 LOCATE#2,2+4*j,2+2*i:PRINT#2,m(i,j)
120 NEXT j,i
130 SYMBOL AFTER 240
140 REM **********************
150 REM ** Joystickabfrage **
160 REM **********************
170 IF JOY(0)=2 THEN y=y+1
180 IF JOY(0)=1 THEN y=y-1
190 IF JOY(0)=4 THEN x=x-1:IF x<2 THEN x=37:y=y-1
200 IF JOY(0)=8 THEN x=x+1:IF x>36 THEN x=2:y=y+1
210 y=MAX(y,2):y=MIN(y,22)
220 IF x<7 OR x>35 OR y<4 OR y>19 THEN f=0:GOTO 280
230 REM **********************
240 REM ** Stellenpruefung **
250 REM **********************
260 f=0
270 IF INT((x-3)/4)=(x-3)/4 AND INT((y-2)/2)=(y-2)/2 THEN f=1
280 IF f=1 THEN GOSUB 590 ELSE GOSUB 510
290 IF JOY(0)=16 THEN 340
300 IF INKEY(24)=0 THEN 440 ELSE 170
310 REM ********************
320 REM ** Pixel aendern **
330 REM ********************
340 IF f=0 THEN 170
350 m((x-3)/4,(y-2)/2)=-(m((x-3)/4,(y-2)/2)-1)
360 FOR i= 1 TO 8
370 w(i)=0
380 FOR j=1 TO 8
390 w(i)=2^(8-j)*m(j,i)+w(i)
400 NEXT j,i
410 SYMBOL 255,w(1),w(2),w(3),w(4),w(5),w(6),w(7),w(8)
420 LOCATE#1,2,10:PRINT#1,CHR$(255)
```

```
430 GOTO 170
440 LOCATE#2,1,24:z$="SYMBOL 255"
450 FOR i= 1 TO 8:z$=z$+","+MID$(STR$(w(i)),2,5):NEXT i
460 PRINT#2,z$
470 GOTO 470
480 REM **********************
490 REM ** Cursor ohne Zahl **
500 REM **********************
510 PEN#2,3:LOCATE#2,x,y:PRINT#2, CHR$(143)
520 FOR i=1 TO 100: NEXT i
530 PEN#2,2:LOCATE#2,x,y:PRINT#2, CHR$(32)
540 RETURN
550 REM **********************
560 REM ** Cursor mit Zahl **
570 REM **********************
580 REM
590 LOCATE#2,x-1,y:PEN#2,3:PRINT#2,m((x-3)/4,(y-2)/2)
600 FOR i=1 TO 100:NEXT i
610 LOCATE#2,x-1,y:PEN#2,2:PRINT#2,m((x-3)/4,(y-2)/2)
620 RETURN
```

Am Anfang finden wir die uns schon bekannten Initialisierungsbefehle für Farbe, WINDOWS, etc. Danach wird ein Feld von acht mal acht Nullen dimensioniert und mit einem LOCATE-Befehl positionsrichtig ausgegeben. In Zeile 140 beginnt dann die JOYSTICK-Abfrage-Schleife. Je nach Stellung des JOYSTICKS werden dann die Variablen X und Y, die die Position des Cursors in der X- und Y-Richtung enthalten, entsprechend verändert. Die nachfolgenden Zeilen stellen fest, ob an dieser Stelle ein Matrixzeichen oder ein Leerraum vorhanden ist. Davon abhängig wird ein Merker (= FLAG f) gesetzt oder nicht gesetzt. Er bestimmt, ob der Cursor als Cursorblock blinkt, oder nur die Matrixzahl die Farbe wechselt. Es wird in die entsprechende Subroutine gesprungen.

In den nachfolgenden Zeilen wird festgestellt, ob der Hochpfeil oder der Feuerknopf gedrückt wurden. Der Hochpfeil führt zum Programmabbruch und Ausdruck des zugehörigen SYMBOL-Kommandos. Die Wirkungweise des Feuerknopfes haben wir oben bereits erläutert. Wir können nun also mit dem Joystick beliebig über den Bildschirm

fahren und unser Wunschsymbol verändern. Geben wir schließlich den Hochpfeil ein, so wird uns das vollständige Kommando präsentiert. Dabei müssen wir jedoch beachten, daß der Computer zur Definition des neuen Zeichens immer den Character mit der Nummer 255 benutzt. Haben wir also ein neues großes A kreiert, so müssen wir diese Nummer gegen den entsprechenden Zeichen-CODE - in diesem Fall 65 - austauschen. Viel Spaß beim Basteln neuer Zeichen.

3.4 Hochauflösende Grafik

Bei der Arbeit mit Texten haben wir immer einen vorher bereits definierten Block von Bildpunkten gleichzeitig auf dem Bildschirm ausgegeben. Jedes Zeichen war dabei in einer Matrix aus Bildpunkten definiert, und diese brachten wir mit dem Ausdruck des Zeichens auf den Schirm. Im letzten Kapitel haben wir uns dann damit beschäftigt, wie wir unseren Zeichensatz umdefinieren können, um damit neue Zeichen zu schaffen. Trotz aller Änderungsmöglichkeiten waren wir bis jetzt immer auf die Acht-mal-acht-Matrix eines Zeichens beschränkt. Wir wollen diese Grenze nun sprengen und uns mit den Möglichkeiten einer Einzelpunktansteuerung beschäftigen.

Damit machen wir den Schritt von der niedrigauflösenden zur hochauflösenden Grafik. Schauen wir uns zunächst den Befehlssatz an, über den wir im Bereich der hochauflösenden Grafik verfügen.

3.4.1 Der Befehlssatz

Beim Arbeiten mit der hochauflösenden Grafik ist der Bildschirm in 400 Reihen zu je 640 Punkten unterteilt. Gezählt wird dabei von 0 bis 399 in der Senkrechten und von 0 bis 639 in der Waagerechten. Der Nullpunkt unseres Systems befindet sich dabei in der linken unteren Ecke des Bildschirms. Ähnlich unserem Text-Cursor, den wir mit dem LOCATE-Befehl über den gesamten Bildschirm bewegen konnten, verfügen wir auch hier über einen Cursor, den sogenannten Grafik-Cursor. Im Unterschied zu dem Kästchen des Text-Cursors ist dieser jedoch unsichtbar. Wir können ihn jedoch mit

```
PLOT XPOS,YPOS
```

jederzeit zum Vorschein bringen. Die Funktionen XPOS und YPOS enthalten nämlich immer die aktuelle Position unseres Grafik-Cursors in X- beziehungsweise Y-Richtung. Mit dem PLOT-Befehl setzen wir einen beliebigen Punkt auf unserem Grafikschirm. Geben wir zum Beispiel

```
PLOT 0,0
```

so sehen wir unseren Ursprungspunkt 0,0. Mit

```
PLOT 320,200
```

befinden wir uns im Mittelpunkt unseres Bildschirms. Mit dem PLOT-Befehl können wir also beliebige Punkte auf den Bildschirm setzen oder rücksetzen. Das Rücksetzen geschieht dabei dadurch, daß wir den Hintergrund mit der entsprechenden Hintergrundfarbe plotten. Dazu können wir das entsprechende Farbregister mit einem Komma an den PLOT-Befehl anhängen. So wird mit

`PLOT 320,200,0`	der Mittelpunkt gelöscht
`PLOT 320,200,1`	der Mittelpunkt auf Farbregister 1 gesetzt
`PLOT 320,200,2`	der Mittelpunkt auf Farbregister 2 gesetzt

Die letzte Möglichkeit funktioniert dabei natürlich nur in den MODES 0 und 1, da uns im MODE 2 auch in der hochauflösenden Grafik nur die Farbregister 0 und 1 zur Verfügung stehen. Für die Farbe gelten also dieselben Regeln, wie wir sie schon beim Arbeiten mit Texten kennengelernt haben. Ein großer Unterschied besteht jedoch in der Wirkung der Koordinaten. Mit

```
PLOT 320,200,1
```

wird unabhängig von den drei MODES immer genau der Mittelpunkt des Bildschirms gesetzt, allerdings mit unterschiedlicher Breite. Dies hat seine Ursache wieder in der Abspeicherung der Farben, die, wie wir ja schon gesehen haben, dazu führt, daß im

MODE 1 zwei Bildpunkte und im

MODE 0 vier Punkte

bei einem PLOT gleichzeitig nebeneinander gesetzt werden. Wir müssen also auch hier wieder eine Wahl zwischen höherer Auflösung und höheren farblichen Möglichkeiten treffen. Fragen wir nun die Position unseres Grafik-Cursors mit

```
PRINT XPOS,YPOS
```

ab, so erhalten wir die Werte 320 und 200. Unser Grafik-Cursor ist also mitgewandert. Der Grafik-Cursor befindet sich immer am Ende einer grafischen Aktivität, das heißt:

Zeichnen wir eine Linie von links oben nach rechts unten, so befindet sich der Grafik-Cursor danach am unteren Ende der Linie. Setzen wir einen Punkt an einer beliebigen Stelle auf den Bildschirm, wird auch der Grafik-Cursor an diese Stelle bewegt. Das Wissen um die aktuelle Position des Grafik-Cursors ist deswegen so wichtig, weil der CPC beim Zeichnen von Linien nicht den Anfangs- und den Endpunkt angibt, sondern nur den Endpunkt und als Anfangspunkt die Position des Grafik-Cursors voraussetzt. Der Endpunkt wird dabei wie bei unserem PLOT-Befehl definiert. Mit

```
DRAW 320,0
```

ziehen wir jetzt eine Linie vom Mittelpunkt des Bildschirms senkrecht nach unten. Der Grafik-Cursor befindet sich jetzt am Ende dieser Linie, ein nochmaliges DRAW 320,0 hat daher keinen Effekt. Auch beim DRAW-Befehl können wir wieder die Farbe entsprechend anhängen. Wollen wir also unsere Linie wieder löschen, so können wir das mit DRAW 320,200,0. Wir befinden uns damit auch wieder mit dem Grafik-Cursor im Bildschirmzentrum. Wir können aber auch den Grafik-Cursor direkt bewegen. Dazu dient der MOVE-Befehl. Er hat dieselbe Form wie das PLOT-Kommando, nur das hier kein Punkt gesetzt oder rückgesetzt wird, und damit auch das Anhängen einer Farbinformation entfällt. So liefert uns der folgende Einzeiler ein Koordinatenkreuz auf dem Bildschirm:

```
10 MOVE 320,0:DRAW 320,400:MOVE 0,200:DRAW 640,200
```

Ob ein bestimmter Bildpunkt gesetzt ist, und auf welche Farbe, können wir mit dem TEST-Befehl überprüfen. Das TEST-Kommando ist

wie XPOS und YPOS eine Funktion. Allerdings wird hier nicht die aktuelle Position abgefragt, sondern das Farbregister mit dem ein beliebiger Bildpunkt momentan belegt ist. Geben wir also

```
PRINT TEST(320,200)
```

ein, erhalten wir, nachdem wir das oben genannte Programm ausgeführt haben, den Wert Eins. Führen wir nun einen CLS-Befehl aus, so verschwindet unser Koordinatenkreuz, der Mittelpunkt wird gelöscht und ein nochmaliges

```
PRINT TEST(320,200)
```

ergibt den Wert Null.

Bis jetzt haben wir ausschließlich die absolute Adressierung benutzt, das heißt, wir haben jeden Punkt auf dem Bildschirm durch die Angabe seines X- und Y-Wertes bestimmt. Der CPC verfügt aber (wie auch bei der Textdarstellung) noch über eine andere Art der Adressierung: die relative Adressierung. Bei der relativen Adressierung wird die Position des neuen Bildpunktes oder der Endpunkt der zu zeichnenden Linie relativ zur Position des Grafik-Cursors bestimmt. Ein kurzes Beispiel macht den Unterschied schnell klar:

```
MOVE 320,200:DRAW 100,100
```

zieht eine Linie vom Mittelpunkt zum Punkt mit der Koordinate (100,100), das heißt nach links unten.

```
MOVE 320,200:DRAWR 100,100
```

zieht dagegen eine Linie, die vom Ausgangspunkt des Grafik-Cursors 100 Bildpunkte nach rechts und 100 Bildpunkte nach oben reicht. In diesem Fall wird also eine Linie nach rechts oben gezogen. Auch die anderen Befehle können wir durch eingefügtes R in relativ adressierte Kommandos umwandeln. TESTR stellt dann die Bildschirmfarbe in Abhängigkeit von der letzten Cursor-Position fest. PLOTR setzt oder löscht einen Punkt relativ zur Position des Grafik-Cursors. Mit MOVER bewegen wir den Grafik-Cursor relativ zu seiner alten Position. Das Anhängen der Farbinformation wirkt hierbei wie bei der absoluten Adressierung. Jede Adressierungsart hat je nach dem verfolgten Ziel bestimmte Vor- und gegebenfalls auch Nachteile. So ist typischerweise die grafische Darstellung einer Funktion einfacher mit den absolut adressierten

Kommandos zu realisieren, denn hier werden meist die Funktionswerte berechnet und diese brauchen dann nur noch geplottet zu werden oder bilden den Endpunkt einer mit DRAW zu zeichnenden Linie. Sollen dagegen komplexere Figuren, wie vergrößerte Zahlen oder Buchstaben, auf dem Bildschirm dargestellt werden, so ist es meist einfacher, diese mittels relativer DRAWs zu definieren. Dies hat darüber hinaus den Vorteil, daß die ganze Figur durch Änderung des Anfangspunktes verschoben werden kann, ohne daß die entsprechenden DRAW-Befehle geändert werden müssen. Teilweise kann es auch optimal sein, die eine Adressierungsart mit der anderen zu kombinieren.

Wir kommen nun zu dem wohl stärksten und auch längsten Kommando im Bereich der hochauflösenden Grafik, dem

ORIGIN-Befehl

Bis jetzt sind wir immer davon ausgegangen, daß sich der Nullpunkt unseres ganzen grafischen Systems in der linken unteren Ekke befindet. Dies ist beim Einschalten zwar der Fall, kann aber vom Benutzer jederzeit geändert werden. Nach

ORIGIN 320,200

befindet sich der Nullpunkt unseres grafischen Systems in der Bildschirmmitte. Dadurch ändert sich nun die Bedeutung aller Koordinaten. Hatte der Befehl DRAW 100,100 weiter oben noch das Ziehen einer Linie vom Mittelpunkt nach links unten zur Folge, so bringt uns dasselbe Kommando jetzt einen Strich nach rechts oben. Durch das neue Zentrum werden nun plötzlich auch negative, relative und absolute Adressen möglich. Dies hat sehr viele Vorteile. Zum einen können wir nun Funktionen, die durch mehrere Quadranten gehen, ohne großes Umrechnen direkt darstellen. Durch mehrmaliges Verschieben des Zentrums mit dem ORIGIN-Befehl ist es möglich, dieselbe Figur mehrmals nebeneinander auf dem Bildschirm abzubilden, ohne sie wieder neu definieren zu müssen.

Schließlich wird das einfache Darstellen von Kreisen erst durch eine Verschiebung des Zentrums in die Mitte des Bildschirms möglich. Mit der Verschiebung des Ursprungspunktes sind aber die Möglichkeiten des ORIGIN-Befehls noch nicht erschöpft. Zusätzlich dazu können wir nämlich noch die Eckkoordinaten eines Grafik-WINDOWS angeben. Alle grafischen Befehle bleiben dann auf diesen Teil des Bildschirms beschränkt. Die Definition des Grafik-WIN-

DOWS geht dabei wie bei einer normalen WINDOW-Definition vor sich, nur daß wir die 400 mal 640 Punkte der hochauflösenden Grafik für die Definition der Eckwerte benutzen. Damit ist es uns nun möglich, in dem einen Teil des Bildschirms eine Grafik zu zeichnen, die im anderen Teil des Bildschirms erklärt wird. Eine andere Möglichkeit wäre in Analogie zu unserem DREI-WINDOW-System die Ausgabe von Titel und Instruktionen oberhalb und unterhalb eines Grafik-Bildschirms, in dem wiederum die Hauptausgabe erfolgt. Ein besonderer Vorteil dabei ist, daß wir den Grafik-Bildschirm, wie auch die Text-WINDOWS separat mit dem CLS-Kommando löschen können, ohne daß die anderen WINDOWS beeinflußt werden. Wenn Sie das Ganze in Gedanken einmal durchspielen, werden Ihnen sicherlich noch viele weitere Anwendungsfälle einfallen. Wenden wir uns nun den praktischen Anwendungsmöglichkeiten zu.

3.4.2 Grafische Statistik

Die grafische Statistik ist wohl das Paradebeispiel für die Anwendung der hochauflösenden Grafik. Wir haben Sie bei der Beschäftigung mit den grafischen Möglichkeiten von WINDOWS bereits kennengelernt. Infolge der mangelhaften Auflösung von WINDOWS waren wir auf die Darstellung von Balkendiagrammen beschränkt. Mit den erweiterten Möglichkeiten der hochauflösenden Grafik haben wir nun ein ganzes Bündel von Darstellungsmöglichkeiten zur Verfügung. Im einzelnen sind dies:

- Balkendiagramme
- Stabdiagramme
- Kreis- oder Kuchendiagramme
- grafische Darstellung als Funktionsgraf

Bevor wir uns mit den eigentlichen Programmen auseinandersetzen, wollen wir uns mit diesen Darstellungsmitteln und ihren Eigenschaften etwas näher befassen. Fangen wir von hinten an. Beim Zeichnen von Funktionsgrafen geht es darum, eine Vielzahl von Meßwerten in Abhängigkeit von einer Variablen, typischerweise der Zeit, grafisch darzustellen. Es interessiert also die Änderung und gegebenenfalls auch die absolute Höhe dieser Werte im Zeitablauf. Dies kann grafisch sehr gut verdeutlicht werden, indem man

die einzelnen Meßwerte zu einer Funktion verbindet. Einen praktischen Anwendungsfall würde hier zum Beispiel die Darstellung des täglichen Auftrageingangs einer Firma über ein Jahr bilden.

Verfügen wir nur über wenige Werte, bietet sich das Darstellungsmittel des Stabdiagramms an. Dabei werden die entsprechenden Funktionswerte mit der X-Achse verbunden. Die dadurch entstehenden Stäbe oder Linien geben in ihrer Länge die Größe des Wertes wieder, das Verhältnis der Linien zueinander den relativen Anteil. Ein Beispiel hierfür wären Stichprobenmessungen jeder Art, zum Beispiel der Schadstoffgehalt der Luft im Wochenrhythmus. Aber auch die Lernfortschritte von Schülern oder Sportlern, festgestellt anhand monatlicher Tests, lassen sich mit dieser Darstellungsmethode gut wiedergeben.

Kreisdiagramme eignen sich gut, um das Verhältnis von bestimmten Zahlen zu veranschaulichen. Die Kreis- oder Kuchenstücke entsprechen dabei den einzelnen Zahlenwerten, die addiert als Summe den ganzen Kreis oder Kuchen bilden. Dieses Verfahren wird oft bei Bundestagswahlen etc. angewandt, um die Sitzverteilung im Parlament zu veranschaulichen. Aber auch im kommerziellen Bereich bieten sich gute Anwendungsmöglichkeiten, wenn zum Beispiel Quartalsergebnisse miteinander verglichen werden sollen.

Das letzte Darstellungsmittel, das Balkendiagramm, bietet sich an, wenn man über kumulierte Werte verfügt, oder Werte, die über einen längeren Zeitraum konstant bleiben, und sich dann plötzlich ändern. Summieren wir zum Beispiel unsere obengenannten Werte für die Auftragseingänge und teilen diese durch die Anzahl der Tage im Monat, so erhalten wir einen Wert für den mittleren Auftragseingang in diesem Monat. Dieser läßt sich nun durch die Höhe eines farbigen Balkens darstellen. Einen weiteren Anwendungsfall bildet die Darstellung des Sparzinses über den Zeitablauf. Da dieser Wert von den Banken relativ selten geändert wird, läßt er sich mittels des Balkendiagramms gut grafisch wiedergeben.

Wenden wir uns nun den Programmen zu. Zuerst wollen wir uns damit beschäftigen, wie wir das schon im Kapitel über WINDOWS kennengelernte Balkendiagramm mit der hochauflösenden Grafik erzeugen können. Als Beispiel benutzen wir hier die Darstellung der monatlichen Stromkosten im Jahresverlauf. Die entsprechenden Spielwerte oder auch Zahlen aus der Realität können Sie selbst eingeben.

```
10 REM ****************************
20 REM ** Saeulengrafik mit DRAW **
30 REM ****************************
40 INK 0,27:INK 1,21:PAPER 0:PEN 1:BORDER 0
50 DATA Januar,Februar,Maerz,April,Mai,Juni
60 DATA Juli,August,September,Oktober,November,Dezember
70 DIM m$(12),m(12)
80 FOR i=1 TO 12:READ m$(i):NEXT i
90 CLS
100 FOR i= 1 TO 12
110 PRINT"Bitte geben Sie die Stromkosten fuer    den Monat ";
120 PRINT m$(i);" ein";:INPUT m(i)
130 NEXT i
134 REM *****************
135 REM ** Maximalwert **
136 REM *****************
140 ma=0:mi=10000000
150 FOR i= 1 TO 12:ma=MAX(ma,m(i)):mi=MIN(mi,m(i)):NEXT i
160 CLS
170 LOCATE 7,2:PRINT"B A L K E N D I A G R A M M"
180 PRINT"Bitte geben Sie die Breite der Bloecke  an (5-50)";:INPUT b
190 IF b<5 OR b>50 THEN PRINT"Fehler":GOTO 180
200 CLS:LOCATE 7,2:PRINT"B A L K E N D I A G R A M M"
210 INK 0,0:INK 1,2:INK 2,15:INK 3,21:BORDER 0
220 ORIGIN 0,0
230 FOR i= 1 TO 12
240 f=(i MOD 3)+1
250 h=m(i)/ma*250
260 FOR j=1 TO b:MOVE b*i+j,80:DRAWR 0,h,f
270 NEXT j,i
280 PEN 2
290 GOTO 290
```

Das Hauptproblem einer jeden grafischen Darstellung ist die Wahl des Maßstabs. Wählt man einen relativ niedrigen Maßstab und gibt diesen fest vor, so werden zwar die Unterschiede in der Höhe der Balken sehr gut sichtbar, aber es besteht die Gefahr von Bereichsüberschreitungen. Setzen wir den entsprechenden Maßstab dagegen zu hoch an, so haben wir die Gewißheit, daß alle Blöcke richtig abgebildet werden, aber gegebenenfalls sind die Unterschiede nur noch schwach erkennbar. Das ganze stellt also ein Op-

timierungsproblem dar und ist speziell dann, wenn die Werte relativ weit schwanken, nur durch eine automatische Maßstabswahl lösbar.

Nach den üblichen Vorbereitungsarbeiten, wie Einlesen der Texte, Ausgabe des Titels Balkendiagramm, der Farbgebung und der Abfrage der entsprechenden Stromkosten in einer Schleife, ist dies die erste Tätigkeit des Computers zu Beginn des Ausdrucks. In einer Schleife über alle zwölf Werte wird mittels der Funktion MAX und MIN das Maximum beziehungsweise Minimum der Funktion ermittelt. Als nächstes erfolgt die Abfrage der Breite der Blöcke. Da der CPC über keinen Befehl in der hochauflösenden Grafik verfügt, der das Erzeugen eines Rechtecks ermöglicht, müssen wir unsere Blöcke aus einzelnen Linien mittels DRAW zusammensetzen. Mit dieser Angabe geben Sie an, wie viele Linien nebeneinander gezeichnet werden sollen und bestimmen damit, wie breit jeder PLOT werden soll. Der Bereich sinnvoller Eingaben ist dabei auf die Werte zwischen fünf und fünfzig beschränkt.

Die nächsten Zeilen bilden die eigentliche Darstellungsroutine. Zunächst setzen wir das Zentrum unseres Systems mit ORIGIN 0,0 in die linke untere Ecke und löschen den Bildschirm. Die eigentliche Darstellung geschieht dann auf den nächsten drei Zeilen in der geschachtelten FOR-TO-Schleife. Die äußere Schleife läuft dabei über alle Monate, die innere über alle Linien eines Blocks. Für jeden Monat wird zunächst die Farbe festgelegt. Danach wird die Höhe des entsprechenden Blocks aus dem von Ihnen eingegebenen Werten und dem vorher festgestellten Maßstab errechnet. Dann geht es in die innere Schleife. Mittels der Laufvariablen I und J und der vorher eingegebenen Breite wird der Fußpunkt der zu zeichnenden Linie auf der X-Achse errechnet und danach mit dem MOVE-Kommando der Grafik-Cursor auf diese Position bewegt. Der nachfolgende relative DRAWR-Befehl zieht dann eine Linie von entsprechender Höhe in der Farbe F. Das ganze wiederholt sich solange, bis ein Block mit entsprechender Breite fertiggestellt wurde. Danach wird die Monatsschleife für den nächsten Monat abgearbeitet und die Prozedur wiederholt sich wieder. Den Abschluß des Programms bildet dann ein sogenanntes "ewiges GOTO". Diese Zeile hat zur Folge, daß der Computer, so bald er auf sie stößt, in ihr hängen bleibt, da er den GOTO-Befehl laufend ausführt. Dies hat den Zweck, das störende READY am Programmende zu unterdrücken. Wollen Sie das Programm also beenden, so müssen Sie auf die ESC-Taste drücken.

Wenden wir uns nun den anderen Darstellungsarten zu. Das Stabdiagramm basiert auf demselben Prinzip. Auch hier repräsentiert die Höhe die Größe der entsprechenden Werte. Der einzige Unterschied besteht darin, daß sich nun nicht die verschiedenen Blöcke untereinander abdecken, die insgesamt die volle Breite einnehmen, sondern, daß sich Stäbe und Hintergrundfarbe einander abwechseln.

```
10 REM ***************************
20 REM ** Stabdiagramm mit DRAW **
30 REM ***************************
40 INK 0,27:INK 1,21:PAPER 0:PEN 1:BORDER 0
50 DATA Januar,Februar,Maerz,April,Mai,Juni
60 DATA Juli,August,September,Oktober,November,Dezember
70 DIM m$(12),m(12)
80 FOR i=1 TO 12:READ m$(i):NEXT i
90 CLS
100 FOR i= 1 TO 12
110 PRINT"Bitte geben Sie die Stromkosten fuer    den Monat ";
120 PRINT m$(i);" ein";:INPUT m(i)
130 NEXT i
140 REM *****************
150 REM ** Maximalwert **
160 REM *****************
170 ma=0:mi=10000000
180 FOR i= 1 TO 12:ma=MAX(ma,m(i)):mi=MIN(mi,m(i)):NEXT i
190 INK 0,0:INK 1,2:INK 2,15:INK 3,21:BORDER 0
200 CLS:LOCATE 10,2:PRINT"S T A B D I A G R A M M"
210 INK 0,0:INK 1,2:INK 2,15:INK 3,21:BORDER 0
220 REM ******************
230 REM ** Staebe ziehen **
240 REM ******************
250 ORIGIN 0,0
260 FOR i= 1 TO 12
270 f=(i MOD 3)+1
280 h=m(i)/ma*250
290 MOVE 25*i+166,80:DRAWR 0,h,f
300 MOVE 25*i+168,80:DRAWR 0,h,f
310 MOVE 160,80:DRAWR 330,0,2
320 NEXT I
330 PEN 2
340 GOTO 340
```

Was auf den ersten Blick schwieriger aussieht, ist in Wirklichkeit jedoch eine Vereinfachung des eben betrachteten Programms. Löschen wir den Bildschirm, und zeichnen wir statt eines ganzen Blockes nur eine Linie, so erhalten wir genau diesen Effekt. Nachfolgend das geänderte Listing, auch hier am Beispiel unserer schon bekannten Stromrechnung.

Etwas schwieriger wird die Sache, wenn wir unsere Daten in Form des Kreisdiagramms ausgeben wollen. Das Problem liegt darin, daß wir im Endeffekt eine Kreisfläche verschiedenfarbig unterteilen, was mittels Linien relativ schwer realisiert werden kann, besonders wenn man ein Abdecken der Fläche fordert. Je größer wir dabei den Radius unseres Kreises wählen, desto mehr Linien müssen wir zeichnen, und desto enger müssen diese Linien aneinander liegen, um noch eine gute Flächendeckung zu erreichen. Dies macht aber das Entwerfen der Grafik zum Geduldsspiel. Es heißt also, Kompromisse zu schließen. Wir haben deshalb eine mittlere Kreisgröße gewählt, die eine vernünftige Auflösung bietet, und Sie deswegen nicht gleich eine Kaffeepause einlegen müssen.

Der Anfang des Programms ähnelt den schon betrachteten. Sollte Ihnen die Stromrechnung langsam auf die Nerven gehen, können Sie ja auf Telefonkosten oder ähnliches ausweichen. Neu dagegen ist der Teil ab Zeile 150. Bei der Kreisdarstellung können wir nicht wie bei den vorgenannten Darstellungsarten nur mit den Funktionswerten operieren. Da am Ende ein Kreissegment das Verhältnis der entsprechenden Zahl zum Gesamten darstellen soll, müssen wir zunächst den relativen Anteil jeder Zahl am Gesamten berechnen. Dazu bilden wir zunächst die Summe über alle Werte und berechnen mit Hilfe dieser dann in einer zweiten Schleife den relativen Anteil jedes Werts. Da eine Darstellung aller zwölf Werte aufgrund des geringen Umfangs unseres Kreises eine unbefriedigende Auflösung bringt, fassen wir die relativen Anteile zu Quartalsanteilen zusammen. Nun müssen wir noch dafür sorgen, daß sich die einzelnen Kuchenstücke nicht überdecken, sondern schön nebeneinander auf der Kreisfläche abgebildet werden. Dies erreichen wir, indem wir die Werte der früher abzubildenden Stücke zu dem entsprechenden relativen Teil addieren.

Die Grafik

```
10 REM ********************
20 REM ** Kreisdiagramm **
30 REM ********************
40 INK 0,27:INK 1,21:PAPER 0:PEN 1:BORDER 0
50 DATA Januar,Februar,Maerz,April,Mai,Juni
60 DATA Juli,August,September,Oktober,November,Dezember
70 DIM m$(12),m(12),re(12)
80 FOR i=1 TO 12:READ m$(i):NEXT i
90 CLS
100 FOR i= 1 TO 12
110 PRINT"Bitte geben Sie die Stromkosten fuer denMonat ";
120 PRINT m$(i);" ein";:INPUT m(i)
130 NEXT i
140 REM ********************************
150 REM ** relativen Anteil berechnen **
160 REM ********************************
170 sum=0:CLS
180 FOR i=1 TO 12:sum=sum+m(i):NEXT i
190 FOR i=1 TO 12:re(i)=m(i)/sum:NEXT i
200 re(1)=re(1)+re(2)+re(3):re(2)=re(1)+re(4)+re(5)+re(6)
210 re(3)=re(2)+re(7)+re(8)+re(9):re(4)=re(3)+re(10)+re(11)+re(12)
220 REM ********************
230 REM ** Kreis zeichnen **
240 REM ********************
250 CLS
260 INK 0,0:INK 1,24:INK 2,6:INK 3,2
270 PEN 3
280 LOCATE 7,3:PRINT"K r e i s d i a g r a m m"
290 ORIGIN 300,200
300 re(1)=re(1)*2*PI:re(2)=re(2)*2*PI
310 re(3)=re(3)*2*PI:re(4)=re(4)*2*PI
320 FOR i=0 TO re(1) STEP 0.0051:a=SIN(i)*50
330 b=COS(i)*50:MOVE 0,0:DRAWR a,b,1:NEXT i
340 FOR i=re(1) TO re(2) STEP 0.0052:a=SIN(i)*50
350 b=COS(i)*50:MOVE 0,0:DRAWR a,b,2:NEXT i
360 FOR i=re(2) TO re(3) STEP 0.0052:a=SIN(i)*50
370 b=COS(i)*50:MOVE 0,0:DRAWR a,b,1:NEXT i
380 FOR i=re(3) TO re(4) STEP 0.0052:a=SIN(i)*50
390 b=COS(i)*50:MOVE 0,0:DRAWR a,b,3:NEXT i
400 GOTO 400
```

Danach werden alle Werte noch mit zwei mal PI multipliziert, um den ganzen Kreisumfang zu überdecken. Nun folgen vier FOR-TO-Schleifen, die den eigentlichen DRAW-Algorithmus enthalten. Jede Schleife läuft über ein Segment des Kreises, und zwar in relativ kleinen Schritten. Zur Berechnung der Punkte auf der Kreislinie wenden wir die Sinus- und Kosinusfunktion an, die wir mit 50 multiplizieren, um den richtigen Abstand zu erhalten. Danach wird der Grafik-Cursor in den Nullpunkt unseres Kreises gesetzt und die entsprechende Linie mit der Farbe Eins gezogen. Dasselbe wiederholt sich für die anderen Kreissegmente. Den Abschluß bildet wiederum die schon bekannte Dauerschleife mit GOTO.

Wenden wir uns nun dem letzten grafischen Darstellungsmittel zu, dem Funktionsgraf. Wir werden auf diesen nur kurz schematisch eingehen, da auch dieser wieder nur eine Variante desselben Themas darstellt. In unserem Programm Stabdiagramm brauchen wir nämlich nur die Zeile mit DRAWR durch

```
DRAW 50 * i,h:NEXT i
```

zu ersetzen, und statt der Stäbe verbindet das Programm nun die einzelnen Funktionspunkte durch Linien. Dies ist das Prinzip aller Programme zur Funktionsdarstellung. Was noch fehlt, ist ein Achsenkreuz. Dies läßt sich jedoch relativ leicht mit zwei DRAWS erreichen, die wir schon in der Einleitung zur hochauflösenden Grafik gesehen hatten. Die dazu nötigen Einteilungen sind relativ einfach und schnell mit DRAWS zu erzeugen. Wir wollen deswegen den Bereich der grafischen Statistik verlassen, und uns mit einem ganz anderen Bereich beschäftigen: dem Malen und Zeichnen mit der hochauflösenden Grafik.

3.4.3 Malen und Zeichnen mit der hochauflösenden Grafik

Spätestens seit der Erfindung der hochauflösenden Grafik ist klar, daß Computer nicht nur dazu dienen, aus Zahlen immer neue andere Zahlen zu produzieren. Heute werden Computer in vielen - auch kreativen - Berufen eingesetzt. Die Spannweite reicht hier vom Modeschöpfer bis zum Designer in der Autoindustrie. Sie alle profitieren von der Möglichkeit, am Bildschirm schnell Zeichnungen entwerfen, löschen und auch umändern zu können. Mit dem Prinzip solcher Zeichenprogramme, die wir PAINTING-Programme nennen

wollen, wollen wir uns nun beschäftigen.

Sehen wir uns zunächst einmal an, was wir brauchen:

Bei einem PAINTING-Programm bewegt man meistens einen kleinen, blinkenden Cursor über den Bildschirm. An den entsprechenden Bildschirmpositionen ist es dann möglich, Linien zu ziehen, Kreise, Rechtecke und andere Figuren zu setzen. Die meisten derartigen Programme besitzen darüber hinaus die Option, bestimmte Bildschirmbereiche zu löschen, zu kopieren, oder zu verschieben. Beim Zeichnen von Linien oder Figuren müssen dabei sowohl die Strichstärke als auch die Farbe ausgewählt werden können. Sofern die Programme mehr dem Erstellen von Titelbildern dienen, verfügen sie darüberhinaus meist über eine Abspeicherroutine, um das entsprechende Titelbild in spätere Programmentwicklungen einzubinden.

Eine andere oft angebotene Funktion, ist das automatische Ausfüllen von Figuren, deren Konturen man vorher eingegeben hat, mit einer vorher ausgewählten Farbe. Bei der Ansteuerung existieren im wesentlichen zwei Versionen, die gegebenenfalls miteinander kombiniert werden können. Erfolgt die Bedienung über die Tastatur, so wird der Grafik-Cursor meistens mit den entsprechenden Cursorsteuertasten auf den entsprechenden Punkt des Bildschirms dirigiert. Auf den Funktionstasten sind dabei meist die oben angesprochenen Funktionen wie Linien zeichnen, ausmalen etc. verfügbar, während die Wahl von Farb- und Strichstärke auf die Zahlen und den Rest der Tastatur beschränkt bleibt. Die andere Art der Ansteuerung ist Joystick-orientiert. Der Grafik-Cursor wird durch Bewegen des Joysticks über den Bildschirm dirigiert. Durch Drücken des Feuerknopfs werden die Anfangs- und die Endpunkte einer zu zeichnenden Linie, beziehungsweise die Eckpunkte der Figuren festgelegt. Die Auswahl der Funktion, mit der als nächstes gearbeitet werden soll, erfolgt dabei meist auf einem sogenannten Funktions-Tableau. Bei diesem Tableau handelt es sich um einen abgetrennten Teil des Bildschirms, oder um eine reservierte Bildschirmseite, die kurzfristig mit dem Hauptbildschirm ausgetauscht wird. Auf dieser sind die einzelnen Funktionen bildlich wiedergegeben, das heißt, ein Kreis steht für die Funktion Kreis zeichnen, ein ausgemaltes Rechteck dafür, ein ausgemaltes Rechteck auf den Bildschirm zu setzen usw. Die Auswahl der entsprechenden Funktionen erfolgt dann, indem man den Grafik-Cursor auf das entsprechende Funktionsbild setzt und wiederum den Feuerknopf betätigt.

Außer diesen beiden Bedienvarianten sind natürlich noch viele andere denkbar. Die sinnvollste dabei ist, eine Kombination aus den beiden erstgenannten. Dabei wird die Bewegung des Grafik-Cursors auf dem Schirm durch Setzen der entsprechenden Figuren weiterhin durch den Joystick gesteuert; die Auswahl der Funktion erfolgt parallel dazu durch die Tastatur. Schauen wir uns nun an, wie wir ein derartiges Programm mit dem CPC realisieren können. Wir müssen jedoch noch eine kleine Warnung vorausschicken:

Die nachfolgenden Programme dienen dazu, die wesentlichen Funktionen eines PAINTING-Programms zu demonstrieren. Aus Platzgründen können wir kein vollständiges PAINTING-Programm entwickeln. Wer also ein professionelles Zeichenprogramm erwartet, wird enttäuscht sein. Allerdings ist es mit den gegebenen Anregungen für jeden einzelnen möglich, sich sein individuelles Zeichenprogramm zusammenzustellen. Wer sich näher mit der Materie auseinandersetzen möchte, sei hier auf die Neuerscheinung des Verlags verwiesen.

```
10 REM **************
20 REM ** Joypaint **
30 REM **************
40 INK 0,0:PAPER 0:BORDER 0:INK 1,2:PEN 1
50 MODE 1
60 FOR i=0 TO 3
70 PRINT"Bitte geben Sie den Farbcode fuer Farb- register";i;" ein";
80 INPUT k:INK i,k:NEXT i
90 x=0:y=0:ORIGIN 0,0:CLS:f=1
100 IF(JOY(0) AND 8)=8 THEN x=x+1
110 IF(JOY(0) AND 4)=4 THEN x=x-1
120 IF(JOY(0) AND 2)=2 THEN y=y-1
130 IF(JOY(0) AND 1)=1 THEN y=y+1
140 IF x>639 THEN x=0:y=y+1
150 IF x<0 THEN x=639:y=y-1
160 IF y<0 THEN y=0
170 IF y>399 THEN y=399
180 IF INKEY(15)=0 THEN f=0
190 IF INKEY(13)=0 THEN f=1
200 IF INKEY(14)=0 THEN f=2
210 IF INKEY(5)=0 THEN f=3
220 PLOT x,y,f:GOTO 100
```

Bei dem hier abgedruckten Programm JOYPAINT finden wir die zuletzt angesprochene Eingabeart. Das Bewegen des Grafik-Cursors erfolgt Joystick-gesteuert und die Farben werden durch die Tastatur vorgegeben.

Das Programm arbeitet im MODE 1. Somit stehen uns die Farbregister 0 bis 3 für die Farbdefinition zur Verfügung. In diese werden in eine Abfrageschleife die vom Benutzer gewünschten Farbcodes eingelesen. In Zeile 90 setzen wir dann die Ausgangsdaten für unser Programm. Die Register X und Y enthalten die Position des Cursors in X- beziehungsweise Y-Richtung. In F ist die jeweils aktuelle Schreibfarbe des Grafik-Cursors gespeichert. Sie wird am Anfang auf eins gesetzt. Die nun folgenden Zeilen bis zum Programmende bilden eine Dauerschleife. Am Anfang wird die der Joystick und danach die Tastatur abgefragt, ob eien Farbänderung gewünscht wird. In Zeile 220 schließlich wird der entsprechende Bildpunkt mit der gewünschten Schreibfarbe gesetzt und es geht wieder zurück nach 100.

Basis der Joystick-Abfrage ist dabei das JOY-Kommando. Diese Funktion fragt den momentanen Zustand des Joysticks ab, und zwar JOY(0) von Joystick 0 und JOY(1) von Joystick 1. Drücken wir den Joystick nach oben, nimmt die JOY-Abfrage den Wert 1 an. Eine Bewegung nach unten gibt den Wert 2. Schieben wir den Joystick seitlich nach links, so erhalten wir eine 4 und bei Verschiebung nach rechts eine 8 als Funktionswert unserer JOY-Abfrage.

Wir sind nicht auf diese vier Richtungen beschränkt. Drücken wir den Joystick zum Beispiel nach rechts oben, so erhalten wir als Funktionswert 9. Eine kombinierte Bewegung wird also immer als Summe der Einzelbewegungen ausgegeben. Für unsere Bestimmung der aktuellen Cursorposition bringt dies einige Probleme mit sich. Da wir den Momentanstand des Cursors in den Registern X und Y gespeichert haben, müssen wir bei einer schräggerichteten Bewegung beide Register gleichzeitig bewegen. Die JOY-Funktion gibt uns aber leider nicht die einzelnen Werte, sondern nur ihre Summe an. Setzen wir also nur bei JOY (0) = 8 unser X-Register um eins hoch, so werden schräge Bewegungen von oben nach unten nicht erfaßt. Mit dieser einfachen Art der Abfrage können wir daher nur gradlinige Bewegungen feststellen. Allerdings ist es möglich, neben den vier Abfragen auf gradlinige Bewegungen noch vier Abfragen auf die Schrägbewegungen nach rechts oben, rechts unten, links oben und links unten einzubauen. Die dadurch notwendigen neuen Zeilen würden allerdings unser Programm unnötig verlangsa-

men.

Wir benutzen deshalb ein wenig binäre Arithmetik, um den Wert unserer JOY-Funktionen auf die uns passenden Größen umzuformen. Dabei nutzen wir die Tatsache aus, daß die Werte eins, zwei, vier und acht Potenzen von zwei sind, nämlich zwei hoch null, zwei hoch eins, zwei hoch zwei und zwei hoch drei. In binärer Schreibweise, der internen Sprache unseres Computers, würden diese Zahlen wie folgt aussehen:

```
1 = 0001

2 = 0010

4 = 0100

8 = 1000
```

Unser Funktionswert 9 würde dann wie folgt geschrieben:

```
9 = 1001
```

Wendet man die AND-Verknüpfung auf zwei solche Binärzahlen an, so erhält das Ergebnis nur an den Stellen eine Eins, an denen beide Werte eine Eins haben. Bilden wir also

```
9 AND 8  =        1001
         AND      1000
                  ----
                  1000
```

wird nur die erste Stelle unseres Ergebnisses gesetzt. Umgewandelt in dezimale Schreibweise hat also die Verknüpfung 9 AND 8 den Wert 8. Mit dieser Art der AND-Verknüpfung können wir also im Endeffekt prüfen, ob eine entsprechende Stelle unserer Binärzahl gesetzt war oder nicht. Mit diesem kleinen Trick können wir somit sehr schnell eine Zerlegung unseres JOY-Werts in die benötigten Änderungen für die X- und Y-Koordinate erreichen.

Wir müssen nun noch prüfen, ob die Werte für X und Y sich weiterhin im erlaubten Bildschirmbereich befinden. Für X stellen dabei die Werte 639 und 0 die Grenzen dar, für Y die Werte 0 und 399. Mit einer direkten Abfrage der Tastatur über die INKEY-Funktion - diese Technik werden wir im Kapitel über die Tastatur noch näher

kennenlernen - wird dann geprüft, ob eine Farbänderung notwendig ist. Falls ja, wird das Farbregister F auf den neuen Wert gesetzt. Die Eingabe der Farben erfolgt dabei über die Tasten null bis drei der abgesetzten Zehnertastatur. Da bei der Abfrage des Tastencodes die gedrückte Taste bestimmt wird, und nicht der zugehörige Wert, sind diese Tasten nicht mit den gleichlautenden Zahlentasten auf der oberen Reihe der Haupttastatur gleichgeschaltet. Ein Drücken auf diese Tasten hat also keinen Effekt. In Zeile 220 wird dann der entsprechende Punkt gesetzt und es erfolgt die Rückkehr zum Schleifenanfang nach Zeile 100.

Wir können nun im Rahmen unserer farblichen Möglichkeiten beliebige Linien auf dem Bildschirm zeichnen. Aber auch Kurven, Kreise und asymetrische Figuren sind mit dieser Zeichenmethode möglich.

In dieser Form hat das Programm jedoch noch einige gravierende Nachteile. Zum einen kann der Grafik-Cursor nur beim Ziehen einer Linie als Endpunkt dieser Linie erkannt werden. Es wäre wünschenswert, die Cursor-Position jederzeit zum Beispiel durch einen blinkenden Punkt-Cursor feststellen zu können. Zum anderen müssen wir uns bis jetzt immer noch mit einer Breite unseres Malstiftes von einem Bildpunkt begnügen. Mit dem Programm ist es bereits möglich, durch mehrmaliges Ziehen einer Linie nebeneinander oder paralleles Abfahren der Konturen einer Figur einen dickeren Strich zu erhalten. Bei komplizierteren Figuren erfordert dies jedoch bereits einige Geschicklichkeit und führt häufig zu Fehlern. Wir sollten unser Programm daher um eine Option zum automatischen Zeichnen dickerer Linien erweitern. Um das erstere zu erreichen, fügen wir vor PLOT zwei neue Befehle ein. Diese dienen dazu, den aktuellen Stand des X- und Y-Registers in den Variablen xa und ya zwischenzuspeichern. Beim nächsten Durchlauf der Schleife können wir dann in der neueingefügten und durch RENUMBER hochgesetzten Zeile 280 prüfen, ob die X- und Y-Werte gleichgeblieben sind. Ist dies der Fall, erfolgt der Sprung in das Unterprogramm in Zeile 390 und 400. Dies setzt zunächst einen Punkt mit der Cursorfarbe plus eins. Danach geht es in eine Zeitverzögerungsschleife, die hier relativ kurz über I=1 bis 20 läuft. Nach dem Rücksetzen des Punkts geht es zurück in das Hauptprogramm. Dieses kurze Programm bewirkt nun ein relativ auffälliges Aufblitzen des Punktes, an dem sich der Grafik-Cursor momentan befindet.

Um unser zweites Ziel zu erreichen, müssen wir mehr Programmzeilen schreiben. Wir führen eine neue Variable mit der Bezeichnung m ein, die die Strichdicke widerspiegeln sollen. m=0 bedeutet dabei, daß jeweils nur ein Bildpunkt gezeichnet wird, bei m=1 werden zwei parallel gesetzt und bei m=2 werden drei Punkte gleichzeitig auf dem Bildschirm ausgegeben. Diese Variable legen wir auf die Taste 7 unserer abgesetzten Zehnertastatur. Bei Programmbeginn wurde m auf 0 gesetzt. Einmaliges Drücken auf diese Taste setzt m auf 1 hoch, und mit dem zweiten Drücken erhöht sich der Wert von m auf 2. Betätigen wir die 7 ein weiteres Mal, so setzt das Programm m wieder auf 0 zurück. Diese relativ kompliziert erscheindende Abfrage benötigt dabei nur zwei Zeilen, die Zeilen 260 und 270. Mit der Änderung von m haben wir jedoch noch keine Änderung des Ausdrucks erreicht. Wir müssen dazu noch unseren PLOT-Befehl etwas umwandeln. Dazu trennen wir zuerst die GOTO-Anweisung in der alten Zeile 220 ab und schieben sie in die neue Zeile 350.

In den dazwischen verbleibenden zwei Zeilen werden nun in Abhängigkeit von m 0, ein oder zwei Punkte parallel zu den bereits geplotteten auf den Bildschirm gebracht. Dabei wenden wir einen kleinen Trick an. Wir verschieben die in X und Y gespeicherte aktuelle Cursor-Position diagonal um eine Einheit und setzen dann auf diese Koordinaten einen weiteren Bildpunkt. Dies hat zwar zur Folge, daß der Fußpunkt einer senkrechten Linie, die wir mit doppelter Breite zeichnen, schräg statt glatt abgeschnitten ist, was jedoch wegen dem im MODE 1 relativ geringen Abstand der Bildpunkte zueinander kaum ins Gewicht fällt. Hier noch einmal das geänderte Listing.

```
10 REM ****************
20 REM ** Joypaint 2 **
30 REM ****************
40 INK 0,0:PAPER 0:BORDER 0:INK 1,2:PEN 1
50 MODE 1
60 FOR i=0 TO 3
70 PRINT"Bitte geben Sie den Farbcode fuer Farb- register";i;" ein";
80 INPUT k:INK i,k:NEXT i
90 x=0:y=0:ORIGIN 0,0:CLS:f=1
100 IF(JOY(0) AND 8)=8 THEN x=x+1
110 IF(JOY(0) AND 4)=4 THEN x=x-1
120 IF(JOY(0) AND 2)=2 THEN y=y-1
130 IF(JOY(0) AND 1)=1 THEN y=y+1
140 IF x>639 THEN x=0:y=y+1
150 IF x<0 THEN x=639:y=y-1
160 IF y<0 THEN y=0
170 IF y>399 THEN y=399
180 REM ***************
190 REM ** Farbwahl **
200 REM ***************
210 IF INKEY(15)=0 THEN f=0
220 IF INKEY(13)=0 THEN f=1
230 IF INKEY(14)=0 THEN f=2
240 IF INKEY(5)=0 THEN f=3
250 FOR i=1 TO 30:NEXT i
260 IF INKEY(10)=0 THEN m=m+1
270 IF m>2 THEN m=0
280 IF xa=x AND ya=y THEN GOSUB 390
290 PLOT x,y,f:xa=x:ya=y
300 REM ********************
310 REM ** Mehrfachstrich **
320 REM ********************
330 IF m>0 THEN PLOT x-2,y-2,f
340 IF m>1 THEN PLOT x+2,y+2,f
350 GOTO 100
360 REM ************************
370 REM ** blinkender Cursor **
380 REM ************************
390 PLOT x,y,f+1:FOR i=1 TO 20: NEXT i
400 PLOT x,y,f:RETURN
```

4

Der Sound

4 Der Sound

4.1 Wie entsteht ein Ton?

Das menschliche Ohr kann Töne nur dadurch wahrnehmen, daß Luftschwingungen das Trommelfell in unserem Ohr in Schwingung versetzen, die dann das Gehirn auswertet. Jeder Ton ist also nichts anderes als eine Schwingung der Luft. Die einzelnen Töne unterscheiden sich in ihrer Tonhöhe durch die Anzahl der Schwingungen in einem bestimmten Zeitraum. Diese Anzahl der Schwingungen in einem bestimmten Zeitraum nennt man Frequenz. Je höher diese Frequenz ist, umso höher ist der Ton, den unser menschliches Ohr wahrnimmt.

Da unser Computer nicht in der Lage ist, ohne weiteres Luft in Schwingung zu versetzen, müssen wir uns fragen, wie dies geschieht. Speziell dafür hat der Schneider CPC 464 einen eigenen SOUNDCHIP, das heißt im Prinzip noch einen kleinen Computer, der die Töne erzeugt. Dieser SOUNDCHIP erzeugt allerdings noch nicht direkt den Ton, sondern erst das elektrische Signal, das dem jeweiligen Luftdruck, der erforderlich ist, um die Schwingung zu erzeugen, entspricht.

Um den Ton hörbar zu machen, benötigen wir einen Verstärker und einen Lautsprecher. Beides ist in unserem Computer eingebaut. Da der Verstärker aber nur eine geringe Leistung hat und der Lautsprecher sehr klein ist, sind die Töne, die aus unserem Computer herauskommen, relativ leise. Die Klangqualität ist ebenfalls nicht besonders gut. Wesentlich verbessert werden kann die Klangqualität und auch die Lautstärke, indem wir unseren Computer an eine Stereoanlage anschließen. Dafür ist auf der Rückseite ein spezieller Anschluß vorgesehen, aus dem unser Signal in Stereo entnommen werden kann.

Der SOUNDCHIP des CPC 464 enthält 3 Tongeneratoren und einen Rauschgenerator. Es handelt sich dabei nicht um Generatoren, wie sie zum Beispiel von elektronischen Orgeln her bekannt sind, sondern um Oszillatoren, wie man sie von Synthezisern her kennt. Der große Unterschied zwischen Generator und Oszillator ist der, daß die Tonhöhe eines erzeugten Tones nicht fest eingestellt ist wie

beim Generator, sondern von der Spannung des Stromes abhängt, mit dem der Oszillator kontrolliert wird. Dadurch wird es möglich, bei einem Oszillator fast stufenlos Tonhöhen zu programmieren.

Zur Programmierung des SOUNDCHIPS im CPC 464 stehen uns drei Befehle zur Verfügung. Zur Erzeugung eines Tons dient der Befehl SOUND. Die beiden Befehle ENT und ENV ermöglichen es, gewisse Effekte zu programmieren. Da es besonders wichtig ist, die Syntax des SOUND-Befehls verstanden zu haben, wollen wir uns zunächst mit diesem Befehl etwas näher beschÄftigen.

4.2 Die Programmierung von Tönen

Wie schon erwähnt, hat unser SOUNDCHIP drei Oszillatoren zur Verfügung, mit denen jeweils ein Ton erzeugt werden kann. Mit dem Befehl SOUND kann jeder dieser Oszillatoren separat angesprochen werden. Jeder Ton, der mit einem Oszillator erzeugt wird, wird auf einen separaten Soundkanal gelegt. Wir haben also insgesamt 3 Kanäle zur Verfügung. Beim Stereoton würde das den Kanälen rechts, links und mitte entsprechen.

Damit der Computer weiß, auf welchen Kanal er den Ton ausgeben soll, wenn der SOUND-Befehl aufgerufen wird, muß die Kanalnummer zunächst angegeben werden. Das heißt, hinter dem Befehl SOUND ist die erste Nummer die Kanalnummer oder - besser gesagt - der Kanalstatus. Wenn wir nur einen Ton ausgeben wollen, geben wir hier die Kanalnummer ein. Als Kanalnummern stehen uns zur Verfügung 1, 2 und 4. Daß der 3. Kanal mit 4 bezeichnet ist, hat eine besondere Bewandtnis, auf die wir später noch einmal zurückkommen. Nehmen wir zunächst die Kanalnummern einmal so hin. Wenn wir also im 1. Kanal einen Ton ausgeben wollen, geben wir jetzt hinter dem SOUND-Befehl eine 1 ein.

Als nächstes müssen wir eingeben, mit welcher Frequenz der Ton gespielt werden soll, das heißt also, welche Tonhöhe wir wünschen. Wenn wir die genaue Frequenz kennen, die ausgegeben werden soll, muß man etwas Rechenarbeit investieren, um den Wert zu ermitteln, der an dieser Stelle eingegeben werden muß. Nehmen wir einmal an, wir wollten den Kammerton A ausgeben. Wie den meisten

sicherlich bekannt ist, benötigen wir eine Frequenz von 440 Hz, um den Ton A zu erhalten (Hz ist die Abkürzung von Hertz und bedeutet Schwingungen pro Sekunde).

Es ist aber nicht möglich, diese 440 dort sofort einzugeben, da der Computer mit anderen Frequenzbereichen rechnet. Um die Zahl, die wir eingeben müssen, zu ermitteln, müssen wir die Zahl 125000 durch die Zahl 440 dividieren. Wir erhalten dann die Zahl 284. Diese muß an dieser Stelle eingegeben werden, um den Ton A zu erhalten. Für jede andere Frequenz muß diese Rechnung in analoger Weise durchgeführt werden. Wir erhalten dann selbstverständlich für jede andere Frequenz auch einen anderen Wert, der eingegeben werden muß. Da aber in der Regel nicht die Frequenzen sondern die Töne bekannt sind, die der Computer spielen soll, müssen wir in einer Tabelle die möglichen Töne und deren Codeziffern zusammenfassen. Sie finden diese Tabelle nachfolgend.

Tabelle 4.1 Frequenzcodes

Note	Oktave -1	Oktave 0	Oktave 1	Oktave 2	Oktave 3	Oktave 4
C_b	956	478	239	119	60	30
D^b	902	451	225	113	56	28
D_b	851	426	213	106	63	27
E^b	804	402	201	100	50	25
E	758	379	190	95	47	24
F_b	716	358	179	89	45	22
b^b	677	338	169	84	42	21
b_b	638	319	159	80	40	20
A^b	602	301	150	75	38	19
A_b	568	284	142	71	36	18
B^b	536	268	134	67	34	17
H	506	253	127	63	32	16

Die nächste einzugebende Zahl bestimmt die Länge des Tons. Diese Zahl ist etwas leichter zu ermitteln als die Tonhöhe. Bei den eingegebenen Zahlen handelt es sich hier um die Dauer in 1/100 Sekunden. Im Klartext bedeutet dies, daß bei Eingabe der Zahl 100 der Ton exakt eine Sekunde lang, bei Eingabe von 50 der Ton 1/2 Sekunde, bei Eingabe von 200 der Ton 2 Sekunden usw. gespielt

wird. Da in der Regel aber die Töne nicht nach ihrer Länge in Hundertstel Sekunden eingegeben werden sollen, sondern nach Notenwerten, die meistens bekannt sind, empfiehlt es sich, den Wert für eine bestimmte Notenlänge, zum Beispiel für eine Viertelnote einmal festzulegen und dann die entsprechenden Werte für die anderen Noten danach zu berechnen und an den entsprechenden Stellen einzugeben. Diese Technik werden wir später noch einmal betrachten, wenn wir ein fertiges Lied eingeben wollen.

Wir können jetzt mit der nächsten Zahl bestimmen, welches der 15 möglichen ENV-Register für diesen Ton gelten soll. Mit ENV kann ein An- und Abschwellen eines Tons in der Lautstärke definiert werden. Dieser Befehl muß vor dem SOUND-Befehl kommen, am besten zu Beginn des Programms. Da insgesamt 16 verschiedene Lautstärken-Hüllkurven zur Verfügung stehen, dürfte dies eigentlich für jede Aufgabenstellung ausreichen. Bei ENV müssen nach der Kanalnummer 3 Werte eingegeben werden, die im ersten Wert die Anzahl der Wiederholungen festlegen, im zweiten Wert die Höhe der Lautstärkenänderung bei jeder Wiederholung - und zwar positiv oder negativ - und im dritten Wert den Abstand zwischen den einzelnen Wiederholungen, das heißt die Pause. Eine Lautstärkenänderung ist mit diesem Befehl nur durch dauerndes Nacheinanderspielen von einzelnen SOUND-Registern möglich. Welches ENV-Register, das von vornherein definiert werden muß, für einen SOUND-Befehl gewählt wird, kann an dieser Stelle durch Eingabe der Zahl des ENV-Registers definiert werden.

Das gleiche gilt auch für das ENT-Register, das an der nachfolgenden Stelle definiert werden kann. Hier stehen auch 15 Register zur Verfügung, die von vornherein festgelegt werden müssen. Die Syntax dieses Befehles ist analog zu ENT, nur handelt es sich hierbei nicht um eine Lautstärkenänderung sondern um eine Frequenzverschiebung. Bei ENT gibt der erste Wert wiederum das Register an, der zweite Wert die Höhe der Frequenzverschiebung, der dritten Wert die Anzahl der Wiederholungen und der 4. Wert gibt die Pausen zwischen den einzelnen Wiederholungen an, ganz analog zum ENV-Befehl.

Die letzte Zahl hinter dem SOUND-Befehl gibt an, ob, und wenn ja, wie ein Rauschen dazuaddiert werden soll. Steht an dieser Stelle eine Null, wird kein Rauschen zugefügt; befindet sich hier eine Zahl zwischen 1 und 15, wird ein Rauschen unterschiedlicher Art zugefügt. Es handelt sich dabei jedesmal um sogenanntes "weißes Rauschen", das je nach Eingabe einer Zahl dumpfer oder heller

klingt. Dieser Effekt ist schwer in einem Buch zu beschreiben. Sie sollten ihn einmal ausprobieren und alle Zahlen durchspielen. So finden Sie heraus, welche Wirkung jede dieser Zahlen hat. Sie können das auch mit einer FOR-NEXT-Schleife durchführen, indem Sie zum Beispiel folgendes eingeben:

```
10 FOR I=1 TO 15
20 SOUND 1,100,100,10,0,0,I
30 NEXT I
```

Nachdem Sie dieses kleine Programm gestartet haben, hören Sie einen Ton und dann nacheinander die verschiedenen Möglichkeiten des Rauschens. Wenn Sie nur das Rauschen hören wollen, geben Sie anstelle der ersten 100 eine Null ein, dann wird kein Ton erzeugt, und nur das Rauschen ist hörbar. Wenn Sie dann noch wissen wollen, welches Rauschen Sie gerade hören, können Sie vor dem NEXT-Befehl noch einen PRINT I-Befehl einfügen.

Nachdem wir nun die Syntax dieses Befehls verstanden haben und auch den ENT-und ENV-Befehl kennengelernt haben, sollten wir eigentlich in der Lage sein, ein kleines Programm mit diesen Befehlen zu schreiben. Wir wollen jetzt einen kleinen Gag programmieren, und zwar einen startenden und wegfliegenden Hubschrauber. Bei einem startenden Hubschrauber hört man zunächst einen langsam anlaufenden Motor, der dann immer schneller läuft, bis der Hubschrauber startet, wegfliegt und dann immer leiser klingt. Dieses Problem lösen wir am besten so, indem wir in den SOUND-Befehl Variablen hineinschreiben, wie wir es soeben bei unserem kleinen Beispielprogramm Rauschen gesehen haben. Wir nennen diese Variable L und verändern sie mit einer FOR-NEXT-Schleife. Aber gleichzeitig soll sich auch die Lautstärke ändern und die Tonhöhe, nicht nur das Rauschen. Deshalb müssen wir in diese beiden Stellen auch unser L integrieren. In Zeile 50 und 60 wird jeweils unsere Variable L erzeugt, in Zeile 70 finden Sie an drei Stellen die Variable im SOUND-Befehl.

In Zeile 40 wird mit dem ENV-Befehl eine Volumenkurve definiert, die sich besonders am Ende des Programms beim Wegfliegen noch bemerkbar machen wird. Übrigens sind die beiden Kommas in Zeile 70 richtig, da dazwischen eigentliche eine Null stehen müßte. Wir können diese Null aber weglassen, der Computer versteht das auch so.

In der ersten FOR-NEXT-Schleife wird das Anlaufen des Motors hörbar gemacht. Die nächste FOR-NEXT-Schleife dient dem Wegfliegen des Hubschraubers. Dabei brauchen wir eigentlich den Klang nicht mehr zu verändern, sondern lediglich die Lautstärke zurückzunehmen. Die Befehle in Zeile 80 und Zeile 120 innerhalb der FOR-NEXT-Schleife sollen sicherstellen, daß die Töne auch hörbar unterschieden werden können. Würden diese Befehle weggelassen, würden sich die einzelnen SOUND-Befehle, da es sich um jeweils fast gleiche Werte handelt, hörbar nicht bewußt voneinander unterscheiden lassen, sondern fließend ineinander übergehen. Dies ist aber bei der Rotorbewegung des Hubschraubers nicht erwünscht. Hier nun das Listing unseres kleinen Programms:

```
10 '======================
20 'STARTENDER HUBSCHRAUBER
30 '======================
40 ENV 1,1,-1,12
50 FOR J= 1000 TO 1 STEP -5
60 L=INT(J/100)+6
70 SOUND 1,100+1*2,L*2,15,1,,L
80 SOUND 1,0,2
90 NEXT
100 FOR i=14.9 TO 0 STEP -0.1
110 SOUND 1,100,10,INT(i)+1,1,,5
120 SOUND 1,0,2
130 NEXT
```

Versuchen Sie einmal, mit diesem Programm herumzuspielen und die Parameter der einzelnen Werte zu ändern. Der beste Weg, mit dem SOUNDCHIP zurechtzukommen, die einzelnen Werten zu variieren. Bei Funktionen, die lediglich akustisch wahrnehmbar sind, dürfte es fast unmöglich sein, diese verbal hinreichend zu beschreiben. Jede dieser Funktionen muß man einmal gehört haben, um sie überhaupt beurteilen und einsetzen zu können.

4.3 Mehrere Töne gleichzeitig

Da der SOUNDCHIP im CPC 464 drei Kanäle hat, ist es möglich, bis zu drei Töne gleichzeitig klingen zu lassen. Sie werden jetzt si-

cherlich sagen, wozu benötigen wir dafür ein Extrakapitel, das ist doch eigentlich klar. Dann geben wir einfach den Befehl, daß in Kanal 1, in Kanal 2 und in Kanal 3 jeweils ein Ton ausgegeben werden soll.

Das kann man sicherlich so machen, das hat aber verschiedene Nachteile. Zunächst einmal muß bemerkt werden, daß diese Töne nicht unbedingt direkt zusammenklingen. Dies hängt ganz davon ab, wie der Zustand der einzelnen SOUND-Register ist.

In jedem der SOUND-Register kann eine Schlange gebildet werden. Schlange bedeutet dabei, daß die nächsten 4 in diesem Register auszugebenden Töne zwischengespeichert werden. Diese können selbstverständlich hintereinander eingegeben werden. Dann kann mit dem normalen Programm fortgefahren werden. Wenn also jetzt noch ein Ton in der Schlange eines SOUND-Registers zwischengespeichert ist, dann ist es kaum möglich, über diese Art alle 3 Töne gleichzeitig auszugeben.

Stellen Sie sich einmal vor, Sie wollten ein längeres Lied vom Computer spielen lassen und wollten dieses Lied programmieren. Da kann es durchaus vorkommen, daß mehrere Töne gemeinsam kommen müssen, an anderen Stellen wiederum nur ein Ton oder kein Ton, da es sich um Pausen handelt. Wenn jetzt die Töne einfach nur in ihren SOUND-Registern einprogrammiert würden, könnte es dabei leicht vorkommen, daß die einzelnen Stimmen nicht synchron laufen würden und es zu Verschiebungen in unserem Lied käme.

Viel interessanter wäre es, wenn wir einen Befehl hätten, bei dem dem Computer mitgeteilt wird, daß er diesen Ton in einem bestimmten SOUND-Register ausgeben soll und gleichzeitig damit in einem anderen SOUND-Register einen anderen Ton. Oder es wäre auch wünschenswert, dem Computer mit einem Befehl mitzuteilen, daß ein Ton in mehreren SOUND-Registern ausgegeben werden soll.

Das letztere ist sehr leicht zu realisieren. Wir sprachen bereits davon, daß die Nummer 3 als SOUND-Register noch frei ist. Gibt man nun diese 3 ein, so wird man feststellen, daß der definierte Ton in SOUND-Register 1 und 2 gleichzeitig ausgegeben wird, allerdings nur dieser eine Ton und nicht mehrere Töne.

Wie kommt dies zustande? Wir sprachen oben einmal von der Kanalnummer bzw. dem Kanalstatus. Hierbei handelt es sich nicht um eine Nummer des Kanals, in dem ein Ton ausgegeben werden soll,

sondern um die Art, nach der eine Ausgabe erfolgen soll. Dabei bedeuten die Statusnummern 1, 2 und 4, daß die Kanäle 1, 2 und 3 angesprochen werden. Durch Addition dieser Zahlen ist auch eine Überschneidung möglich. So bedeutet Eingabe der Statuszahl 3, daß Kanal 1 und 2 angesprochen werden, da 1 + 2 = 3. Die Statuszahl 5 bedeutet, daß Kanal 1 und 3 angesprochen werden; Kanal 2 und 3 werden durch die Statuszahl 6 angesprochen , und die Statuszahl 7 bedeutet, daß alle 3 Kanäle zur Ausgabe herangezogen werden.

Betrachten wir nun etwas näher das Problem der Erzeugung von mehreren Tönen gleichzeitig. Wenn zwei Töne gleichzeitig erzeugt werden sollen, ist es sinnvoll, dem Computer über den Kanalstatus mitzuteilen, daß zwei Töne in zwei SOUND-Registern gleichzeitig gespielt werden sollen. Um dies dem Interpreter mitzuteilen, gibt es noch weitere Kanalstatuszahlen. So bedeutet beispielsweise die Zahl 8, daß der ausgegebene Ton gemeinsam mit einem Ton in Kanal 1 erzeugt werden soll. Die Zahl 16 bedeutet, daß dieser Ton gemeinsam mit einem Ton in Kanal 2, und die Zahl 32 bedeutet, daß dieser Ton gemeinsam mit einem Ton in Kanal 3 erzeugt werden soll. Wir müssen beim Kanalstatus zwei Zahlen berücksichtigen, nämlich zunächst die Zahl des Kanals, in dem die Ausgabe erfolgen soll, und dann eine weitere Zahl als Anweisung, mit welchem anderen Kanal gemeinsam diese Ausgabe erzeugt werden soll. Beispiel: Wollen wir einen Ton in Kanal 1 erzeugen, der gemeinsam mit einem Ton in Kanal 3 ausgegeben werden soll, müssen wir als Kanalnummer 33 eingeben. Dies entspricht der Zahl 1 für die Kanalnummer und der Zahl 32 zur Erkennung der gemeinsamen Ausgabe.

Wenn Sie damit aber einen Befehl eingeben, werden Sie noch nichts hören. Dies liegt daran, daß in Kanal 3 noch kein Ton definiert wurde, der gemeinsam mit Kanal 1 ausgegeben werden sollte. Es ist also immer erforderlich, beide Töne gemeinsam zu definieren. Um in Kanal 3 einen Ton zu definieren, der mit Kanal 1 gemeinsam ausgegeben werden soll, müssen wir eine 12 als Kanalstatus eingeben. Dies entspricht der 4 für den Kanal 3 zuzüglich der 8 als "Rendezvousziffer" für Kanal 1.

Werden also jetzt zwei SOUND-Befehle, einmal mit der Kanalstatusziffer 33 und einmal mit der Kanalstatusziffer 12 eingegeben, können Sie zwei Töne hören. So wird es dann möglich, ein zweistimmiges Lied, bei dem nicht immer beide Stimmen gleichzeitig spielen, zu programmieren. Wenn nur ein Ton ausgegeben werden soll, müssen wir dazu die einfache Kanalnummer eingeben. Sollen zwei Töne ausgegeben werden, muß die Rendezvousziffer dazuaddiert

werden. Schauen wir uns dies einmal an einem Beispiel an:

```
10 REM ==========
20 REM EUROVISION
30 REM ==========
40 REM ================
50 REM BILDSCHIRMAUFBAU
60 REM ================
70 BORDER 1 : INK 0,1
80 MODE 0
90 LOCATE 10,3 : PRINT"*"
100 LOCATE 6,6 : PRINT"*"
110 LOCATE 14,6: PRINT"*"
120 LOCATE 5,12: PRINT"*"
130 LOCATE 15,12:PRINT"*"
140 LOCATE 6,18: PRINT"*"
150 LOCATE 10,21:PRINT"*"
160 LOCATE 14,18:PRINT"*"
170 REM ====
180 REM LIED
190 REM ====
200 SOUND 33,106,40 : SOUND 12,0,40
210 SOUND 33,80,37 : SOUND 12,638,80
220 SOUND 1,0,5
230 SOUND 1,80,18
240 SOUND 1,71,20
250 SOUND 33,63,40 : SOUND 12,638,40
260 SOUND 33,80,40 : SOUND 12,506,40
270 SOUND 33,53,80 : SOUND 12,851,40
280 SOUND 4,676,40
290 SOUND 33,63,57 : SOUND 12,638,37
300 SOUND 4,0,5 : SOUND 4,638,38
310 SOUND 1,0,5
320 SOUND 1,63,18
330 SOUND 33,60,40 :SOUND 12,956,80
340 SOUND 1,53,20
350 SOUND 1,60,20
360 SOUND 33,63,20 : SOUND 12,638,40
370 SOUND 1,60,20
380 SOUND 33,53,40 : SOUND 12,6,20
390 SOUND 4,478,20
400 SOUND 33,71,20 : SOUND 12,851,75
410 SOUND 1,80,20
```

```
420 SOUND 1,71,20
430 SOUND 1,63,20
440 SOUND 1,71,40
450 SOUND 4,0,5
460 SOUND 33,106,40 : SOUND 12,851,40
470 SOUND 33,80,37 : SOUND 12,638,75
480 SOUND 1,0,5
490 SOUND 1,80,18
500 SOUND 1,71,20
510 SOUND 33,63,40 : SOUND 12,638,40
520 SOUND 33,80,40 : SOUND 12,506,40
530 SOUND 33,53,80 : SOUND 12,851,40
540 SOUND 4,676,40
550 SOUND 33,63,57 :SOUND 12,638,38
560 SOUND 4,0,2 :SOUND 4,638,40
570 SOUND 1,0,5
580 SOUND 1,63,18
590 SOUND 33,60,20 : SOUND 12,956,80
600 SOUND 1,53,20
610 SOUND 1,63,20
620 SOUND 1,60,20
630 SOUND 33,71,60 : SOUND 12,851,38
640 SOUND 4,0,2 : SOUND 4,851,40
650 SOUND 1,80,18
660 SOUND 33,0,5 : SOUND 12,638,40
670 SOUND 1,80,57 : SOUND 4,851,40
680 SOUND 4,638,40
690 IF st<>1 THEN st=1: GOTO 200
700 FOR I=1 TO 2000 : NEXT : CLS
```

Dieses Programm spielt Ihnen das Thema der Eurovision mit einer Baßbegleitung vor. Dabei wurde die Melodie in SOUND-Register 1 abgelegt, die Begleitung in Kanal 3. Bei gemeinsamen Tönen müssen wir also, wie oben aufgerechnet, als Kanalstatus die Zahlen 33 und 12 eingeben, bei Einzeltönen die Statuszahlen 1 und 4. Wenn Sie sich das Programmlisting einmal anschauen, werden Sie feststellen, daß nur diese vier Kanalstatuszahlen verwendet worden sind.

Für die Musikexperten unter Ihnen sei noch ein kurzer Hinweis zum Tempo angebracht. Bei einer Viertelnote wurde hier das Tempo auf die Länge 40 normiert. Das bedeutet, daß für eine Halbenote die

Zahl 80 eingegeben werden müßte und für eine Achtelnote die Zahl 20.

Jetzt werden Sie allerdings in dem Programmlisting auch andere Zahlen finden, zum Beispiel 37 und 18. In Zeile 210 finden Sie die Zahl 37, in Zeile 230 die Zahl 18. Sie werden dabei feststellen, daß es sich beide Male um den Ton mit der Frequenz 80 handelt. Das entspricht der Note G. Da diese Note zweimal hintereinander, einmal als Viertelnote und einmal als Achtelnote ausgegeben werden mußte, wäre dies problematisch geworden, wenn in Zeile 210 der Wert 40 und in Zeile 230 der Wert 20 eingegeben worden wäre. Der SOUNDCHIP hätte dies nämlich zum Wert 60 addiert und nur einmal diese Note erklingen lassen. Um dies zu verhindern, wurde mit dem Zeitwert 5 einmal ein Ton, bestehend aus der Frequenz Null ausgegeben, das heißt also, der Ton wurde für kurze Zeit unterbrochen. Damit aber keine Taktverschiebung zustandekommt, müsse die beiden anderen Töne um den Wer 5 gekürzt werden. Deshalb wurde der erste Ton auf 37 gekürzt und der zweite Ton auf 18. Das bedeutet, daß der erste Ton etwas früher aufhört und der zweite Ton etwas später beginnt. Da diese Zeitspanne allerdings sehr klein ist, ist diese kurze Zeitverschiebung nicht hörbar.

Damit Sie über das Programm einen besseren Überblick haben, wurde es so gestaltet, daß in jeder Zeile jeweils nur eine Tonausgabe berücksichtigt wurde. Stehen also zwei Befehle in einer Zeile, handelt es sich um zwei Töne, die gleichzeitig ausgegeben werden, steht nur ein Befehl in einer Zeile, handelt es sich nur um einen Ton. Das bedeutet natürlich, daß in jeder Zeile, in der zwei Befehle stehen, die SOUND-Register 33 und 12 verwendet werden müssen, in den anderen Zeilen die SOUND-Register 1 oder 4. Dies wurde bewußt so gestaltet, damit Sie den Aufbau des Programms besser verstehen können.

Insgesamt soll das Thema mit der Begleitung zweimal gespielt werden. Deshalb mußte eine Wiederholung eingebaut werden, da wir nicht das gleiche Programm noch einmal eingeben wollten. Allerdings mußte nach der Wiederholung das Programm erkennen, daß die Wiederholung bereits ausgeführt worden ist. Um dies zu erreichen, haben wir eine zusätzliche Variable eingeführt, die Variable st als Abkürzung für Status. Sobald wir eine Wiederholung durchführen, ordnen wir dieser Variablen den Wert 1 zu. Da beim Starten des Programms alle Variablen auf Null gesetzt werden, bedeutet dies, daß jedesmal eine Wiederholung durchgeführt werden soll, wenn die Statusvariable nicht 1 ist. Wir fragen also in Zeile 690

ab, ob die Statusvariable ungleich 1 ist. Wenn das der Fall ist, wird dieser Variablen die Zahl 1 zugeordnet, und das Programm springt zur Zeile 200 zurück, das heißt die Melodie erklingt von neuem. Wenn wir dann wieder zur Zeile 690 kommen, ist die Statusvariable gleich 1, es wird also keine Übereinstimmung festgestellt, und das Programm ist zu Ende.

Dieses Programm soll die Möglichkeit des SOUNDCHIPS demonstrieren. Damit es nicht so langweilig auf dem Bildschirm aussieht, haben wir in einem kleinen Vorprogramm das Symbol der europäischen Flagge, den Kranz der Sterne auf blauem Hintergrund, dargestellt. Vielleicht fällt Ihnen noch ein schöneres Bild zu diesem Lied ein. Sie können das Programm jederzeit erweitern.

Jetzt haben wir es ermöglicht, zwei Töne auszugeben, und zwar in den Registern 1 und 3. Es ist natürlich auch möglich, jede andere Kombination, zum Beispiel 1 und 2 oder 2 und 3 zu wählen. Dann müssen selbstverständlich andere Kanalstatuszahlen gewählt werden. Selbstverständlich ist es auch möglich, alle drei Register zusammen erklingen zu lassen. Auch dann gelten wieder andere Kanalstatuszahlen. Die Werte dieser Kombinationen können Sie aus der folgenden Tabelle entnehmen.

Tabelle 4.2 Kanalstatus für mehrere Töne gleichzeitig

Kanal	jeder allein	1 und 2 zusammen	1 und 3 zusammen	2 und 3 zusammen	alle zusammen
1	1	17	33	-	49
2	2	10	-	34	42
3	4	-	12	20	28

Aus dieser Tabelle gehen alle Kombinationen hervor. In der ersten Spalte steht der anzusprechende Kanal, in der zweiten Spalte die Kanalstatusziffer, wenn jeder Kanal für sich alleine betrieben werden soll, danach kommen die Kombinationen.

Bedenken Sie bitte, daß bei der Auswahl einer Kombination jeweils auch die anderen erforderlichen Soundregister gesetzt werden müssen. Wenn Sie zum Beispiel eine Kombination von zwei Tönen auswählen und nur einen Soundchip setzen, kann der Computer Ihren Befehl nicht ausführen. Bei der Wahl von drei Tönen müssen Sie auch alle drei Register setzen. Werden nur zwei von den entsprechenden Soundregistern gesetzt, ist der Computer nicht in der Lage, diesen Befehl auszuführen und ignoriert ihn. Sie erhalten dann auch keine Fehlermeldung vom Computer, sondern der Lautsprecher bleibt stumm.

Die Striche in dieser Tabelle bedeuten, daß hierfür keine Kanalstatusziffer existiert. Dies ist ja auch verständlich, da bei einer Kombination von zwei Tönen nur zwei Register gesetzt zu werden brauchen. Das dritte ist dann überflüssig. Nachdem wir nun wissen, wie drei Töne gleichzeitig programmiert werden, können wir uns an ein größeres Beispiel wagen. Wir wählen dazu ein Lied aus, bei dem die Melodie zweistimmig mit einer Baßbegleitung gespielt werden soll. Dieses Lied ist der Titelsong aus dem Musical "Hello Dolly".

Auch bei diesem Programm wurde wieder so verfahren, daß zusammengehörende Befehle in einer Zeile abgedruckt werden. Stehen also in einer Zeile drei Befehle, bedeutet das, daß an dieser Stelle drei Töne gleichzeitig ausgegeben werden. Bei zwei Befehlen werden zwei Töne gleichzeitig ausgegeben, bei einem Befehl lediglich ein Ton. Bei diesem Programm mußten alle Kombinationen, die oben in der Tabelle angegeben sind, verwendet werden. Da unser Lied dreistimmig ist, das heißt eine Melodie, eine Begleitmelodie und eine Baßbegleitung gespielt werden sollen, war es nicht immer klar, welche Stimme gerade mit welcher Stimme zusammenkommen mußte. Dies variierte so stark, daß alle drei Kombinationen benötigt wurden. Daneben wurde natürlich auch die Möglichkeit, daß jede Stimme für sich spielen mußte, und auch die, daß alle drei Stimmen gemeinsam spielen mußten, gebraucht.

Da das Stück schneller sein sollte als unser vorhergehendes Beispiel, wurde hier eine Viertelnote auf den Wert 20 normiert. Das bedeutet, eine halbe Note erhält den Wert 40, eine ganze Note den Wert 80 und eine Achtelnote den Wert 10. Da es auch hier vorkam, daß gleiche Töne mehrfach hintereinander gespielt werden mußten, wurden auch hier kleine Zwischenräume eingefügt, die dann von der Länge der anderen Töne abgezogen werden mußten. Dadurch kommen auch andere Zahlen, als die eben angegebenen, zustande.

Der Sound

```
10 REM ===========
20 REM HELLO DOLLY
30 REM ===========
40 CLS
50 SOUND 1,95,20
60 SOUND 49,80,80 : SOUND 42,95,80 : SOUND 28,478,40,8
70 SOUND 4, 638,40
80 SOUND 49,95,20 : SOUND 42,119,20 : SOUND 28,478,80,8
90 SOUND 1,119,20
100 SOUND 1,95,20
110 SOUND 1,80
120 SOUND 49,71,80 : SOUND 42,119,80 : SOUND 28,568,40
130 SOUND 4, 478,40
140 SOUND 49,95,20 : SOUND 42,119,20: SOUND 28,568,80
150 SOUND 1,119,20
160 SOUND 1,95,20
170 SOUND 1,80,20
180 SOUND 49,63,20 : SOUND 42,80,80 : SOUND 28, 758,80
190 SOUND 1,60,20
200 SOUND 1,63,20
210 SOUND 1,60,20
220 SOUND 49,63,20 : SOUND 42,84,80 : SOUND 28,804,80
230 SOUND 1,60,20
240 SOUND 1,53,30
250 SOUND 1,60,10
260 SOUND 49,80,100 : SOUND 42,89,40 : SOUND 28,851,80
270 SOUND 2,119,40
280 SOUND 34,127,20 : SOUND 20,638,80
290 SOUND 1,84,20
300 SOUND 1,80,20
310 SOUND 1,75,20
320 SOUND 49,71,80 : SOUND 42,89,80: SOUND 28,716,40
330 SOUND 4,638,40
340 SOUND 49,89,20 : SOUND 42,106,20: SOUND 28,851,80
350 SOUND 1,106,20
360 SOUND 1,89,20
370 SOUND 1,71,20
380 SOUND 49,67,80  : SOUND 42,106,80: SOUND 28,851,40
390 SOUND 4,716,40
400 SOUND 49,89,20  : SOUND 42,106,20: SOUND 28,851,80
410 SOUND 1,106,20
```

```
420 SOUND 1,89,20
430 SOUND 1,71,20
440 SOUND 49,47,20: SOUND 42,89,80: SOUND 28,638,40
450 SOUND 1,53,20
460 SOUND 33,47,20 : SOUND 12,851,40
470 SOUND 1,53,20
480 SOUND 49,47,20 : SOUND 42,89,80 : SOUND 28,638,40
490 SOUND 1,53,20
500 SOUND 33,63,20 : SOUND 12,851,40
510 SOUND 1,80,20
520 SOUND 49,0,20 : SOUND 42,0,20 : SOUND 28,758,40
530 SOUND 17,71,40: SOUND 10,80,20
540 SOUND 34,84,40: SOUND 20,804,40
550 SOUND 1,63,20
560 SOUND 49,80,20 : SOUND 42,89,20: SOUND 28,851,40
570 SOUND 1,95,20
580 SOUND 33,89,20 : SOUND 12,638,40
590 SOUND 1,84,20
600 SOUND 49,80,80: SOUND 42,95,80: SOUND 28,478,40
610 SOUND 4,638,40
620 SOUND 49,95,20: SOUND 42,119,20: SOUND 28,478,80
630 SOUND 1,119,20
640 SOUND 1,95,20
650 SOUND 1,80,20
660 SOUND 49,71,80: SOUND 42,119,80: SOUND 28,568,40
670 SOUND 4,379,40
680 SOUND 49,95,20: SOUND 42,119,20: SOUND 28,568,80
690 SOUND 1,119,20
700 SOUND 1,95,20
710 SOUND 1,80,20
720 SOUND 49,53,20 :SOUND 42,89,80 : SOUND 28,638,80
730 SOUND 1,60,20
740 SOUND 1,53,20
750 SOUND 1,60,20
760 SOUND 49,53,20 : SOUND 42,95,80 : SOUND 28,478,80
770 SOUND 1,60,20
780 SOUND 1,53,20
790 SOUND 1,60,20
800 SOUND 49,71,80 : SOUND 42,89,80 : SOUND 28,716,35
810 SOUND 4,0,5
820 SOUND 4,716,40
```

```
830 SOUND 49,63,80 : SOUND 42,75,80 :SOUND 28,758,35
840 SOUND 4,0,5
850 SOUND 4,758,40
860 SOUND 49,0,20 :SOUND 42,0,20: SOUND 28,71,40
870 SOUND 17,60,40: SOUND 10,95,40
880 SOUND 4,478,40
890 SOUND 17,71,20: SOUND 10,119,20
900 SOUND 33,80,40: SOUND 42,127,80: SOUND 28,379,80
910 SOUND 1,95,10
920 SOUND 1,80,30
930 SOUND 49,60,8: SOUND 42,95,80: SOUND 28,568,40
940 SOUND 1,0,2
950 SOUND 1,60,8
960 SOUND 1,0,2
970 SOUND 1,60,18
980 SOUND 1,0,2
990 SOUND 33,60,20: SOUND 12,478,40
1000 SOUND 1,71,20
1010 SOUND 49,80,40 : SOUND 42,127,80 : SOUND 28,379,80
1020 SOUND 1,95,10
1030 SOUND 1,80,30
1040 SOUND 49,47,8: SOUND 42,63,40 : SOUND 28,426,80
1050 SOUND 1,0,2
1060 SOUND 1,47,8
1070 SOUND 1,0,2
1080 SOUND 1,47,18
1090 SOUND 1,0,2
1100 SOUND 17,47,18 : SOUND 10,67,40
1110 SOUND 1,0,2
1120 SOUND 1,47,18
1130 IF W=1 THEN 1240
1140 SOUND 49,47,20: SOUND 42,71,40 : SOUND 28,358,40
1150 SOUND 1,53,20
1160 SOUND 49,45,20 : SOUND 42,80,40 : SOUND 28,426,40
1170 SOUND 1,47,20
1180 SOUND 49,60,140 : SOUND 42,95,140: SOUND 28,478,40
1190 SOUND 4,638,40
1200 SOUND 4,568,40
1210 SOUND 4,506,40
1220 SOUND 1,95,20
1230 W=1 : GOTO 60
```

```
1240 SOUND 49,47,20: SOUND 42,71,40: SOUND 28,358,40
1250 SOUND 1,53,20
1260 SOUND 49,45,40: SOUND 42,63,40: SOUND 28,319,40
1270 SOUND 49,47,8: SOUND 42,63,40: SOUND 28,426,80
1280 SOUND 1,0,2
1290 SOUND 1,47,8
1300 SOUND 1,0,2
1310 SOUND 1,47,18
1320 SOUND 1,0,2
1330 SOUND 17,47,18: SOUND 10,67,40
1340 SOUND 1,0,2
1350 SOUND 1,47,18
1360 SOUND 1,0,2
1370 SOUND 49,47,20 : SOUND 42,71,40: SOUND 28,358,40
1380 SOUND 1,53,20
1390 SOUND 49,45,40 : SOUND 42,63,40 : SOUND 28,319,40
1400 SOUND 49,47,8: SOUND 42,63,40: SOUND 28,426,80
1410 SOUND 1,0,2
1420 SOUND 1,47,8
1430 SOUND 1,0,2
1440 SOUND 1,47,18
1450 SOUND 1,0,2
1460 SOUND 17,47,18: SOUND 10,67,40
1470 SOUND 1,0,2
1480 SOUND 1,47,18
1490 SOUND 1,0,2
1500 SOUND 49,47,20 : SOUND 42,71,40: SOUND 28,358,40
1510 SOUND 1,53,20
1520 SOUND 49,45,20 : SOUND 42,80,40: SOUND 28,426,40
1530 SOUND 1,47,20
1540 SOUND 49,60,140 : SOUND 42,95,140 : SOUND 28,478,40
1550 SOUND 4,638,40
1560 SOUND 4,478,60
```

Auch bei diesem Programm sollte ein Teil wiederholt werden. Allerdings nicht, wie bei unserem vorhergehenden Programm, das gesamte Lied, sondern ab einer bestimmten Stelle sollte ein bestimmter Teil wiederholt und dann zu einer anderen Stelle gesprungen werden. Wir haben dies wiederum mit einer Statusvariablen, hier W genannt, erreicht. In Zeile 1130 wird abgefragt, wenn W den Wert 1 erhält, soll zu einer anderen Zeile 1240 gesprungen werden, an der das Programm fortgesetzt wird. Ist W ungleich 1, bedeutet das, daß nach dem Starten und dem ersten Durchlauf das Programm einfach weiter bis zur Zeile 1230 fortgesetzt wird; dort wird W auf den Wert 1 gesetzt und zur Zeile 60 zurückgesprungen.

In der Musiksprache nennt man so etwas Wiederholung mit Klammern, das heißt ein Stück soll wiederholt werden, und einige Takte vor dem Wiederholungszeichen soll zu einem anderen Teil gesprungen werden. Der Teil, der nicht mit wiederholt werden soll, heißt dann Klammer 1, der Teil, in den dann hineingesprungen werden soll, heißt Klammer 2. Um so etwas im Computer zu verwirklichen, müssen wir ein kleines Gedächtnis einrichten, das uns sagt, ob wir die Wiederholung bereits ausgeführt haben oder nicht. Dieses Gedächtnis ist dabei unsere Statusvariable. Im Prinzip verhält sich ein Musiker, der die Noten mit seinem Instrument spielt, ähnlich. Auch er muß sich merken, ob er die Wiederholung bereits durchgeführt hat oder nicht. Er muß wissen, ob er im zweiten Durchgang ist und folglich in die Klammer 2 springen muß. Dieses Merken geschieht beim Computer nicht automatisch, sondern muß gesteuert werden. Die Statusvariable hilft uns dabei.

Nachdem wir jetzt ein zweistimmiges und ein dreistimmiges Lied programmiert haben, wollen wir uns einmal einem kleinen Trick zuwenden, bei dem es auch möglich ist, scheinbar mehr als dreistimmige Klänge aus dem Computer zu holen.

Unter Musikern ist eine Begleitungsart eines Stücks bekannt, die man "Nachschlag" nennt. Dies funktioniert grob gesprochen so, daß während der ersten Taktzeit ein Baß gespielt wird, während der zweiten Taktzeit ein Akkord, während der dritten Taktzeit wiederum ein Baß und während der vierten Taktzeit wieder ein Akkord. Wenn wir jetzt vom Computer den Baß und den Akkord lediglich zweistimmig spielen lassen, haben wir insgesamt nur zwei Stimmen unserer drei möglichen Stimmen verbraucht. Wir haben also unser drittes Register zur Verfügung, um eine Melodie dazu spielen zu können. Mit Hilfe unseres Rauschgenerators können wir dem ganzen

noch eine kleine Schlagzeugbegleitung hinzufügen. Machen wir uns das an folgendem Beispiel klar:

```
1 REM ==============================
2 REM RHYTHMISCHE BEGLEITUNG 3/4 TAKT
3 REM ==============================
10 SOUND 1,638,20,10,0,0,1
20 SOUND 1,0,20
30 SOUND 33,159,40,4,0,0,12 : SOUND 12,127,40
40 SOUND 1,0,5 : SOUND 4,0,5
50 SOUND 33,159,40,4,0,0,12 : SOUND 12,127,40
60 GOTO 10
```

Dieses Programm soll nur ein Beispiel sein, wie man eine Begleitung programmieren kann. Wir haben hier nur den Akkord G-Dur gewählt und auch lediglich ein G im Baß. Sie sollten sich diese Technik klarmachen und einmal versuchen, ein eigenes Lied mit einer solchen Begleitung zu programmieren. Selbstverständlich ist dies auch im 3/4-Takt (Walzer) möglich. Wir möchten das an folgendem Beispiel demonstrieren:

```
10 REM ==============================
20 REM RHYTHMISCHE BEGLEITUNG 4/4 TAKT
30 REM ==============================
40 SOUND 1,638,20,10,0,0,1
50 SOUND 1,0,20
60 SOUND 33,159,40,4,0,0,12 : SOUND 12,127,40
70 GOTO 40
```

Auch hier sind wiederum Ihrer Phantasie keine Grenzen gesetzt. Versuchen Sie einmal, mit diesem Grundmuster ein Lied zu programmieren. Eigentlich müßten Sie dazu jetzt in der Lage sein. Haben Sie nur Mut, setzen Sie sich dran und versuchen Sie einmal, Ihre bisherigen Erkenntnisse in ein kleines Programm umzusetzen. Wir können hier nur den Denkanstoß geben, und diese kleinen Beispielprogramme sollen Ihnen lediglich demonstrieren, was mit Ihrem Computer und dessen SOUNDCHIP alles möglich ist.

4.4 SOUND in ein BASIC-Programm integriert

Bisher haben wir einige Programme entwickelt, die als eigenständiges Musikprogramm abliefen. Dadurch war unser Rechner für andere Anwendungen blockiert. Bei Programmen, bei denen ein Lied erklingen soll, ist dies ja auch nicht unerwünscht. Bei einem solchen Programm möchte man sich lediglich an der Melodie erfreuen und nicht gleichzeitig noch andere Dinge vom Computer erledigen lassen. Für diese Problemstellung reichte also unsere Art zu programmieren vollständig aus.

Oft ist es aber so, daß ein BASIC-Programm und gleichzeitig ein Musikprogramm ablaufen sollen. Es wäre zwar theoretisch möglich, beide Programme gleichzeitig laufen zu lassen, indem man sie ineinanderschreibt, das heißt jeweils dann, wenn ein SOUND-Befehl erfolgt ist, dann den nächsten Befehl aus unserem BASIC-Programm einfügt und dann wieder einen SOUND-Befehl usw., so daß die Programme ineinandergemixt werden. Problematisch wird dabei die Abarbeitungszeit. Wir sind bei dieser Art nicht sicher, daß unsere beiden Programme so synchron laufen, wie wir das gerne hätten. Doch hier bietet sich ein Ausweg an, der uns vom Computer eröffnet wird.

Der SOUNDCHIP ist in der Lage, bis zu 4 Töne in einer Schlange abzulegen und diese dann nach und nach abzuarbeiten. Wünschenswert wäre es, wenn wir diese Schlange auf Vorrat programmieren könnten und dann jeweils nur noch abzurufen brauchten. Das bedeutet, wir hätten in unserem SOUND-Register 4 Töne nacheinander abgespeichert und würden dann an der jeweiligen Programmstelle, an der dies erwünscht ist, den nächsten Ton der Schlange abrufen. So wären wir dann in der Lage, an jeder Stelle unseres Programms einen gewünschten Ton eines SOUND-Programms einzufügen.

Wir bräuchten uns dann über die Synchronisation zwischen unserem BASIC-Programm und unserem Musikprogramm keine Gedanken mehr zu machen, da dies automatisch der Fall wäre, da wir nur an der Stelle, an der auch ein Ton erwünscht ist, diesen Befehl einfügen, mit dem der nächste Ton abgerufen wird. Um dies zu erreichen, wurde die Kanalstatuszahl 64 eingeführt. Zu dieser Zahl müssen wir selbstverständlich noch unsere Kanalnummer addieren. Das bedeutet, daß dies mit der Kanalstatusziffer 65 geschehen muß, wenn wir unsere Registerschlange für den Kanal 1 vollschrei-

ben wollen.

Noch einmal im Klartext: Wenn wir also als Kanalstatusnummer die Zahl 65 eingeben, wird der nächste Ton in der Schlange gespeichert und noch nicht sofort ausgegeben. Selbstverständlich können wir auch andere Kombinationen wählen, so zum Beispiel einen Ton, der in Register 1 zwischengespeichert und mit Kanal 2 gemeinsam ausgegeben werden soll. Dafür müssen wir dann die Kanalstatusziffer 81 eingeben. Gleichzeitig müssen wir natürlich auch in Kanal 2 einen entsprechenden Ton definieren.

Ein Ton, der so zwischengespeichert wird, kann mit dem RELEASE-Befehl abgerufen werden. Wir müssen hinter diesem Befehl noch eine Ziffer angeben, und zwar die Ziffer unsers Kanals, der gespielt werden soll, also 1, 2 oder 4. Schauen wir uns das Ganze einmal an einem kleinen Beispielprogramm an:

```
10 '======================
20 'BEISPIEL FUER RELEASE
30 '======================
40 FOR i=1 TO 4 : SOUND 65,90,20,10,10 : NEXT
50 ON SQ(1) GOSUB 90
60 CALL &BB8A
70 a$=INKEY$:IF a$="" THEN 70
80 PRINT a$; : RELEASE 1 : GOTO 50
90 SOUND 65,90,20,10,10
100 RETURN
```

Mit diesem kleinen Programm wollen wir erreichen, daß beim Drükken einer Taste der entsprechende Buchstabe auf den Bildschirm geschrieben und gleichzeitig ein Piepton erzeugt wird. Dazu schreiben wir unser SOUND-Register in Zeile 40 zunächst einmal mit 4 Tönen voll. Wir haben die Kanalstatusziffer 65 gewählt, das bedeutet, wir registrieren in Kanal 1 diese 4 Töne, ohne daß sie direkt gespielt werden. In Zeile 70 wird dann die Tastatur abgefragt. In Zeile 80 wird der eingegebene Buchstabe auf dem Bildschirm ausgegeben, mit RELEASE 1 wird der Ton abgerufen und das ganze Programm neu gestartet.

Wir müssen lediglich darauf achten, daß immer genügend Töne in unserem VIDEOCHIP als Schlange zwischengespeichert sind. Es gibt

in BASIC einen Befehl, der überprüft, ob ein Element dieser Schlange frei ist. Dieser Befehl heißt SQ. In Zeile 50 wird überprüft, ob in unserem Kanal 1 ein Element frei ist. Wenn dies der Fall ist, wird es mit Zeile 90 beschrieben. Wir springen also in eine Subroutine, die mit RETURN beendet wird. Durch diesen Befehl wird immer dann, wenn in unserer SOUND-Schlange ein Element abgearbeitet worden ist, unser BASIC-Programm kurz unterbrochen, und in der Subroutine in Zeile 90 dieses Element neu beschrieben. Zur Vervollständigung noch Zeile 60: Dort wird mit einer Routine, die der Interpreter vorgibt, der Cursor auf den Punkt gesetzt, an dem die nächste Ausgabe erfolgt. Um dies aufzurufen, brauchen wir lediglich in die Maschinenroutine einzuspringen. Dies geschieht mit einem CALL-Befehl.

Wir sind nun in der Lage, in ein laufendes BASIC-Programm einen Ton genau an einer gewünschten Stelle einzufügen. Selbstverständlich ist dies auch mit mehreren Tönen möglich. Wir müssen dann nur jeweils die einzelnen Kanäle abrufen. Wenn wir geschickt mit dieser Art der Programmierung umgehen, können wir anspruchsvolle Programmlösungen verwirklichen. Es muß nicht immer nur ein Piepston bei Tastatureingabe sein; man könnte sich auch vorstellen, daß in einem Rechenprogramm die Ausgabe eines Ergebnisses mit einem Ton angezeigt wird. Diverse andere Anwendungsmöglichkeiten kann man sich ebenfalls vorstellen. Der Phantasie sind hier keine Grenzen gesetzt.

Wenn eine solche Schlange definiert wird, hat dies den Nachteil, daß unser SOUND-Register blockiert ist. Wenn Sie danach einen normalen SOUND-Befehl eingeben und in der Schlange ist kein Element mehr frei, werden Sie feststellen, daß mit dem Computer nichts mehr passiert. Er ist solange blockiert, bis ein Element in der Schlange frei wird und das neue Element nachrücken kann. Da aber das Programm zunächst zum Stillstand gekommen ist, wird es nicht mehr passieren, daß das neue Element nachrücken kann, da kein altes Element mehr abgerufen wird. Wenn man da nicht höllisch aufpaßt, kann man manch böse Überraschung erleben.

Jetzt mag es aber in verschiedenen anderen Anwendungen vorkommen, daß zum Beispiel in einem Teil eines Programms eine Synchronisation eines bestimmten SOUND-Programms mit einem BASIC-Programm erforderlich ist und dies dann mit Hilfe der Schlange und dem RELEASE-Befehl verwirklicht wird. In einem anderen Teil des Programms kann es aber sein, daß ein anderer SOUND benötigt wird. Jetzt ist aber durch unseren ersten Programmteil unser SOUND-Re-

gister blockiert. Bevor wir jetzt den neuen SOUND programmieren können, der in diesem Teil des Programms nötig ist, müssen wir dafür sorgen, daß in der SOUND-Schlange Platz geschaffen wird. Dies wäre möglich, indem wir über RELEASE die bisherigen Eingaben erst abarbeiten. Da dies aber in diesem Programmteil unter Umständen nicht erwünscht ist, muß es eine Möglichkeit geben, die gesamte Schlange zu löschen.

Dies geschieht mit Hilfe der Kanalstatusziffer 128. Wenn wir also einen Ton in Register 1 ausgeben wollen und die bisherige Schlange von Register 1 löschen wollen, so müssen wir als Kanalstatus 129 eingeben. Selbstverständlich erhalten wir auch dann einen Ton, wenn sich keine Schlange in den Registern befindet. Wenn wir also sicher sein wollen, daß in jedem Falle der Ton sofort ausgegeben wird und der CompÄuter nicht wartet, bis erst Platz in einer Schlange wird, müßten wir diese Kanalstatusziffer eingeben. Damit ist dann sichergestellt, daß die bisherigen Register gelöscht werden und in Kanal 1 ein Ton ausgegeben wird.

Da aber in der Regel ein Programm entweder im Direktmodus, das heißt durch direkten Aufruf mit dem SOUND-Befehl, oder über die Schlange programmiert ist, wissen wir natürlich von vornherein, ob wir als Kanalstatus Ziffer 1 oder Ziffer 129 eingeben müssen. Wir können die 1 dabei benutzen, wenn vorher keine Schlange gebildet wurde und müssen in jedem Fall die 129 benutzen, wenn vorher eine Schlange gebildet wurde.

Gleiches gilt natürlich auch für die anderen Kanäle. In Kanal 2 muß dann 130 als Kanalstatus und in Kanal 3 132 als Kanalstatusziffer eingegeben werden. Selbstverständlich ist auch eine Kombination von allen anderen Zahlen möglich, zum Beipsiel mit einer Rendezvousziffer oder mit der Zahl 64, die dann bedeutet, daß eine neue Schlange beschrieben wird. Mit Eingabe der Zahl 192 bedeutet das, daß eine bestehende Schlange gelöscht und eine neue Schlange geschaffen wird. Wir müssen dann noch den Kanal dazuaddieren. Bei Kanal 1 käme dann die Zahl 193 heraus.

Unsere Warteschlange können wir auch noch für eine andere Programmiertechnik nutzen: Es ist auch möglich, über einfache SOUND-Befehle bis zu 4 Töne in die Warteschlange hineinzuschreiben, die dann nacheinander abgearbeitet und nicht, wie oben beschrieben, abgerufen werden. Wir können jetzt mit dem ON SQ GOSUB-Befehl abfragen, ob diese Schlange leer ist oder nicht. Wenn Platz in der Schlange ist, springen wir in unsere Subroutine, die neue SOUND-

Befehle enthält. Dadurch wird es möglich, gleichzeitig Musik zu spielen und ein Programm laufen zu lassen und nicht, wie oben beschrieben, innerhalb eines Programms bestimmte Töne aufzurufen.

Wir könnten uns vorstellen, daß während des Programmlaufs noch gleichzeitig eine Melodie gespielt wird. In unserem nächsten Beispiel wurde diese Programmiertechnik genutzt. Wir wollen damit zeigen, daß es möglich ist, gleichzeitig und dauerhaft einen Klang zu erzeugen und auf dem Bildschirm etwas auszugeben. Wir haben das Motorengeräusch eines Propellerflugzeugs dafür gewählt und wollen zur Illustration gleichzeitig den Bildschirm beschreiben lassen.

```
10 '========
20 'FLUGZEUG
30 '========
40 CLS
50 RANDOMIZE 1
60 FOR T=1 TO 50
70 ON SQ(1) GOSUB 180
80 A=INT(RND(5)*20)
90 B=INT(RND(7)*20)
100 LOCATE A+1,B+1
110 PRINT"FLUGZEUG"
120 ON SQ(1) GOSUB 180
130 NEXT T
140 CLS : GOTO 60
150 '====================
160 'SOUND UNTERPROGRAMM
170 '====================
180 FOR i=1 TO 4
190 SOUND 33,900,2,,,,10 :SOUND 12,897,2,,,,10
200 SOUND 33,0,4 : SOUND 12,0,4
210 NEXT i
220 RETURN
```

Im SOUND-Unterprogramm ab Zeile 180 wird unser SOUND-Register 1 und 3 mit 2 verschiedenen Tönen beschrieben. Die Frequenzen der beiden Töne liegen ziemlich dicht zusammen. Im ersten Ton wird der Wert 900 und im zweiten der Wert 897 gewählt. Die hohen Zahlen geben uns an, daß es sich um tiefe Töne handelt, und der ge-

ringe Unterschied bedeutet, daß sie fast gleich sind. Dieses "fast" bedeutet hier, daß eine leichte Klangverzerrung erfolgt. Durch diese Klangverzerrung hat man das Gefühl, daß der Klang schwebt.

In unserer Subroutine werden die beiden Klangregister vollständig mit 4 Tönen beschrieben. In Zeile 70 und Zeile 120 wird dann abgefragt, ob leere Stellen in den SOUND-Registern sind. Sind die SOUND-Register leer, springt das Programm in die Subroutine und füllt die SOUND-Register wieder auf. Beide Zeilen der Abfrage sind deshalb notwendig, damit es nicht vorkommt, daß das Programm erst später in diese Subroutine hineinspringt als nötig wäre, um einen ununterbrochenen SOUND-Programmverlauf sicherzustellen. Würde eine dieser beiden Zeilen gestrichen, könnte es immer wieder vorkommen, daß eine kurze Zeit der SOUND unterbrochen wäre. Bei unserer Problemstellung ist das unerwünscht, da es unser Ziel ist, gleichzeitig ein Verarbeitungsprogramm und ein Musikprogramm so ablaufen zu lassen, daß permanent Musik zu hören ist .(In unserem Fall handelt es sich um das dauerhafte und permanente Brummen eines Propellerflugzeugs.) In diesem Brummen darf natürlich keine Lücke entstehen, da ein Motor bei einem Flugzeug (schon aus Gründen der Sicherheit) ständig laufen muß.

Bei der Benutzung dieser Technik sollten Sie allerdings wissen, daß die dauernde Abfrage des SQ-Kanals (SQ ist die Abkürzung von SOUND QUEUE und bedeutet übersetzt Soundschlange) zusätzliche Arbeit für den Computer darstellt. Jede zusätzliche Arbeit führt zu einem Geschwindigkeitsverlust bei der Ausführung des Programms. Außerdem muß der Computer noch öfter in eine Subroutine springen, die die SOUND-Register neu setzt. Dies erfordert auch Zeit und verlangsamt die Ausführung des Programms. Doch diesen kleinen Nachteil sollten wir in Kauf nehmen, denn wir kommen ja in den Genuß von gleichzeitiger Musikdarbietung und einem BASIC-Programm, das zum Beispiel den Bildschirm gestalten kann.

Unser obiges Programmbeispiel soll selbstverständlich nur eine Demonstration sein und gleichzeitig anregen, eigene Experimente in dieser Art durchzuführen. Sie könnten zum Beispiel damit anfangen, als erstes einen anderen Bildaufbau zu wählen. Da es sich bei diesem Programm lediglich um eine zufällige Beschreibung des Bildschirms mit dem Wort Flugzeug handelt, ist es sicherlich so nicht sehr befriedigend, jedoch zur Demonstration der Möglichkeiten absolut hinreichend. Dieses Programm erhält erst dann einen Sinn, wenn Sie versuchen, die Programmiertechnik nachzuvollziehen

und das Programm vielleicht für Ihre eigenen Anwendungen anzupassen. Das SOUND-Unterprogramm läßt sich selbstverständlich auch in andere Programme einbauen. Das Anspringen che müßte dann analog wie in obigem Programm über den Befehl ON SQ GOSUB erfolgen.

4.5. Speicherplatzsparende SOUND-Programmierung

Wie ein Musikprogramm geschrieben oder in ein bestehendes BASIC-Programm eingebaut wird, und die Möglichkeiten, die uns der Computer bietet, haben wir bisher besprochen. Da wir nun alle in der Lage sein müßten, ein eigenes Musikkprogramm zu schreiben, sollten wir uns nun mit einer etwas platzsparenderen Programmiertechnik beschäftigen.

In unseren obigen Beispielen wurde aufgrund der Übersichtlichkeit für jeden neuen Ton auch ein neuer SOUND-Befehl gewählt. Um in unseren Programmbeispielen besser verstehen zu können, was dort vor sich geht, wurde auch jede neue SOUND-Ansprache in einer neuen Zeile ausgedruckt. Wir konnten also so genau sehen, ob ein zwei oder drei Töne ausgegeben wurden und waren in der Lage, die Reihenfolge der Töne zu kontrollieren. Diese Technik erfordert bei der Programmierung sehr viel Zeit, da jedesmal der SOUND-Befehl neu eingetippt werden muß und belegt auch im Speicher viel Platz. Zum Abschluß unseres Kapitels zum SOUND des CPC 464 wollen wir nun aufzeigen, wie man das besser machen kann.

Wir haben bereits in unseren Beispielen gesehen, daß es möglich ist, im SOUND-Befehl Variablen zu benutzen. Wir müssen also nicht jedesmal die Absolutzahl schreiben, sondern können einen Buchstaben als Variable verwenden, der wir je nach Bedarf einen Wert zuordnen.

Wir müssen nur darauf achten, daß jedesmal bei Aufruf der Zeile in der der SOUND-Befehl mit der Variablenkette steht, den Variablen der richtige Wert zugeordnet ist. Anstelle unserer bisherigen Vielzahl von SOUND-Befehlen können wir also jetzt schreiben:

```
SOUND A,B,C
```

Da wir bei unserem SOUND-Befehl nur drei Zahlen, und zwar den Kanalstatus, die Frequenz und die Dauer des Tones benötigen, reicht es also auch aus, daß wir in unserem Befehl lediglich 3 Buchstaben schreiben. Eigentlich müßten wir auch noch die Lautstärke des Tones definieren. Wenn wir dort nichts eingeben, wird die Lautstärke automatisch auf den Wert 4 gesetzt. Wir können uns diesen Befehl sparen. Wenn wir jedoch mit unterschiedlichen Stimmen und unterschiedlichen Lautstärken arbeiten wollen, müssen wir eine vierte Variable hinzunehmen.

Aber irgendwo muß jetzt noch in unserem Programm unser Lied, das der Computer spielen soll, abgespeichert werden. Es nützt ja nichts, einfach nur einen Befehl mit Variablen in das Programm zu schreiben, wir müssen auch in der Lage sein, den Variablen die benötigten Werte zuzuordnen. Dazu müssen diese Werte aber im Programm enthalten sein. Zur Speicherung von Daten in einem Programm bietet sich selbstverständlich der Befehl DATA an. Hinter diesem Befehl kann eine Kette von Daten folgen, die in einem Programm verwendet werden. Da wir in unseren obigen Programmbeispielen für jeden SOUND-Befehl immer drei Zahlen eingegeben haben, könnten wir jetzt bei der Programmierung eines neuen Lieds auch diese drei Zahlen hintereinander eingeben. Diesmal schreiben wir sie aber nicht hinter einen SOUND-Befehl, sondern in eine DATA-Zeile.

Der Vorteil bei der Programmierung liegt dann darin, daß wir einmal den Befehl DATA in die Zeile schreiben und dann in einer Zeile eine Vielzahl von Tönen ablegen können. Mit dem SOUND-Befehl konnten wir nur einen Ton programmieren. Wir mußten für jeden Ton einen neuen BASIC-Befehl (SOUND) anfügen. Jetzt können wir mehrere Töne unter einem BASIC-Befehl (DATA) zusammenfassen und zwar theoretisch so viele, wie eine BASIC-Zeile an Text fassen kann. Da in einer BASIC-Zeile 255 Zeichen abgespeichert werden können, können Sie sich ausrechnen, wieviel Töne in einer einzigen BASIC-Zeile Platz finden.

Dies vereinfacht natürlich nicht nur die Programmierung, da dadurch bei jedem Ton der SOUND-Befehl eingespart werden kann, sondern hat auch große Auswirkungen auf den Speicherplatzbedarf unseres Programms. Da wir in unserem Programm weniger BASIC-Befehle und auch weniger Zeilen verwenden, wird unser Programm im Speicher des Computers kürzer.

Allerdings muß dazugesagt werden, daß ein solches Programm auch unübersichtlicher wird als Programme in der Art, wie wir sie in

den vorhergehenden Abschnitten geschrieben haben. In den DATA-Zeilen stehen jetzt eine Vielzahl von Zahlen hintereinander, und man kann nicht sofort jeder Zahl ihre Funktion zuordnen. Es wird natürlich schwierig, festzustellen, welcher Ton gerade an welcher Stelle steht. Wir können das dann nur noch durch Abzählen der Zahlen in den DATA-Zeilen feststellen.

Wir haben jetzt unser Lied in DATA-Zeilen in unserem Programm untergebracht. Wir müssen lediglich noch den Computer dazu überreden, daß er diese DATA-Zeilen nacheinander liest und die dort gespeicherten Werte den Variablen A, B, C zuordnet und danach den SOUND-Befehl mit diesen Variablen durchführt. Zum Lesen der Daten aus den DATA-Zeilen steht uns der READ-Befehl zur Verfügung. Wenn in unserem Programm READ A, B,C steht, liest der Computer die ersten drei Zahlen aus unseren DATA-Zeilen und ordnet sie den Variablen A, B, C zu. Der Computer merkt sich jetzt, daß er die ersten drei Zahlen bereits gelesen hat und würde bei nochmaliger Durchführung dieses Befehles die nächsten drei Zahlen lesen und diese Zahlen wiederum den Variablen A, B und C zuordnen. Mit Hilfe einer FOR-NEXT-Schleife können wir jetzt alle Zahlen aus den DATA-Zeilen lesen, den Variablen A, B, C zuordnen und dann den SOUND-Befehl ausführen lassen. Eine solche Programmzeile sähe beispielsweise folgendermaßen aus:

```
FOR I = 1 TO 300: READ A,B,C:SOUND A,B,C:NEXT
```

Hier werden jetzt 300 Töne eingelesen, die mit jeweils 3 Zahlen definiert sind. Sollte in einem Programm die Anzahl der Töne nicht 300 sein, muß natürlich an dieser Stelle die tatsächliche Anzahl Töne stehen. Wenn Sie eine Zahl eingeben, die zu groß ist, erhalten Sie bei der Abarbeitung des Programms in dem Moment, wo der Computer versucht, aus einer DATA-Zeile ein Element zu lesen, das nicht mehr da ist, eine Fehlermeldung. Wird die Zahl zu niedrig angegeben, wird das Programm beendet, bevor das Lied zu Ende gespielt worden ist. Deshalb ist es also nötig, die genaue Anzahl der Töne in dieser FOR-NEXT-Schleife anzugeben.

Die oben beschriebene Programmiertechnik haben wir bei dem folgenden Beispielprogramm angewendet, mit dem das Kapitel über den SOUND des CPC 464 endet.

```
10 '==========
20 'WUNDERBAR
30 '==========
40 MODE 0 : LOCATE 5,13 : PRINT"WUNDERBAR"
50 FOR I=1 TO 186 : READ A,B,C : SOUND A,B,C : NEXT
60 '==========
70 'SOUNDDATAS
80 '==========
90 DATA 1,119,58,1,0,2,1,119,18,1,0,2,33,119,160,12,716,40,4,568,40,4,
956,40,4,716,120
100 DATA 17,71,58,10,89,58, 17,0,2,10,0,2,17,71,18,10, 89,18,17,0,2,10
,0,2,49,71,160,42,89,160,28,716,40,4,568,40,4,956,40
110 DATA 4,568,118,1,142,58,1,0,2,1,142,18,1,0,2,33,142,80,12,568,120
120 DATA 17,134,40,10,159,40,49,106,80,42,134,80,28,956,120,17,119,40,
10,142,20
130 DATA 49,119,160,42,142,160,28,716,40,4,758,40,4,851,40,4,956,120
140 DATA 17,71,58,10,89,58,17,0,2,10,0,2,17,71,18,10,89,20,1,0,2
150 DATA 49,71,120,42,95,120,28,638,75,4,0,5,4,638,20
160 DATA 49,80,80,42,95,80,28,956,120,17,71,40,10,95,40
170 DATA 49,67,160,42,95,120,28,638,78,4,0,2,4,638,40
180 DATA 4,956,120,17,106,40,10,134,40,17,95,40,10,119,40
190 DATA 49,80,40,42,142,40,28,716,120,17,0,40,10,0,40,17,89,18,10,142
,18,17,0,2,10,0,2
200 DATA 49,89,78,42,142,78,28,956,120,17,0,2,10,0,2,17,89,18,10,142,1
8,17,0,2,10,0,2
210 DATA 49,89,160,42,142,160,28,716,40,4,956,40,4,851,40,4,716,40
220 DATA 1,119,58,1,0,2,1,119,18,1,0,2,33,119,160,12,716,40,4,568,40,4
,956,40,4,716,120
230 DATA 17,71,58,10,89,58,17,0,2,10,0,2,17,71,18,10,89,18,17,0,2,10,0
,2,49,71,160,42,89,160,28,716,40,4,568,40,4,956,40
240 DATA 4,568,118,1,142,58,1,0,2,1,142,18,1,0,2,33,142,80,12,568,120
250 DATA 17,134,40,10,159,40,49,106,80,42,134,80,28,956,120,17,119,40,
10,142,20
260 DATA 49,119,160,42,142,160,28,716,40,4,758,40,4,851,40,4,956,120
270 DATA 17,71,58,10,89,58,17,0,2,10,0,2,17,71,18,10,89,20,1,0,2
280 DATA 49,71,120,42,95,120,28,638,75,4,0,5,4,638,20
290 DATA 49,80,80,42,95,80,28,956,120,17,71,40,10,95,40
300 DATA 49,67,160,42,95,120,28,638,78,4,0,2,4,638,40
310 DATA 4,956,120,17,106,40,10,134,40,17,95,40,10,119,40
320 DATA 49,80,40,42,142,40,28,716,120,17,0,40,10,0,40,17,89,18,10,142
,18,17,0,2,10,0,2
330 DATA 49,89,78,42,142,78,28,956,120,17,0,2,10,0,2,17,89,18,10,142,1
8,17,0,2,10,0,2
340 DATA 49,89,160,42,142,160,28,716,40,4,956,40,4,851,40,4,716,40
```

5

Der CPC und die Außenwelt

5 Der CPC und die Außenwelt

In den bisherigen Kapiteln haben wir uns im wesentlichen mit dem Innenleben unseres Computers beschäftigt. Nachdem wir seine Sprache, das BASIC mit den Anwendungsmöglichkeiten in der Daten- und Textverarbeitung etwas näher kennengelernt hatten, waren wir zu den Ausgaben des Computers übergegangen. Wir hatten uns mit seinen grafischen Möglichkeiten und auch mit seinen klanglichen Qualitäten etwas näher befaßt. Bisher war der CPC ein abgeschlossenes System, das arbeitete und in sich selbst Texte, Grafiken und Klänge erzeugte. Wir wollen uns nun im folgenden damit auseinandersetzen, wie wir den Kontakt mit dem Computer herstellen können. Zunächst wollen wir uns mit unserem Hauptkommunikationsmittel in der Verbindung zum Computer beschäftigen, der Tastatur. In diesem Kapitel wird es darum gehen, die Funktion der Tastatur etwas näher zu untersuchen. Desweiteren werden wir uns überlegen, wie wir andere Zeichensätze auf die Tastatur legen und sogar mit einem Tastendruck ganze Funktionen aufrufen, beziehungsweise automatisch ablaufen lassen können.

Im Anschluß daran geht es dann um die Kassette: um das Speichern, Laden und das Beheben von Ladefehlern. Wir werden uns in diesem Kapitel auch damit beschäftigen, wie man Programme mittels der Kassette überlagern kann, und sich automatisch ändernde Programme erzeugt. Im letzten Kapitel werden wir uns dann mit einer außergewöhnlichen Eigenschaft des Schneider auseinandersetzen: seinen Möglichkeiten bei der Echtzeitverarbeitung. Es wird darum gehen, wie wir dem CPC unsere externe Zeit begreiflich machen können, obwohl er selber kein Zeitgefühl hat. Der automatische Ablauf mehrerer Programme gleichzeitig, das Fachwort dafür ist MULTITASKING ist hierdurch möglich. Doch davon später mehr.

5.1 Die Tastatur

Auf den ersten Blick handelt es sich bei unserer Tastatur um eine Ansammlung von hell- und dunkelgrauen Plastiktasten. Auf jeder Taste ist ein Symbol, ein Buchstabe oder ein Zeichen aufgedruckt,

das man durch Drücken der jeweiligen Taste auf den Bildschirm bringen kann. Mit SHIFT erreichen wir dabei die oberhalb der Tasten angegebenen Zeichen beziehungswiese die Großbuchstaben, die nicht extra aufgedruckt wurden. Mittels der CONTROL-Taste erreichen wir die Systemsteuerzeichen auf die wir schon mehrfach in diesem Buch eingegangen sind und die nun eigentlich nichts Außergewöhnliches mehr darstellen. Wir könnten somit dieses Kapitel schließen. Doch ganz so schnell und ganz so einfach ist die Sache nicht. Drücken wir nämlich die CONTROL-Taste und die kleine ENTER-Taste im abgesetzten Zahlenfeld, so erscheint plötzlich ein

```
RUN"
```

und wir werden aufgefordert, die PLAY-Taste unseres Kassettenrecorders zu betätigen. Hat unser wohlgeordnetes Tastaturweltbild diesen Anschlag noch verkraftet, so gerät es spätestens nach

```
KEY DEF 67,1,50,50,50
```

ins Wanken. Drücken wir nun auf die Q-Taste, so erscheint in allen Ebenen, das heißt im Normal-, im SHIFT- und auch im CONTROL-Modus die Zahl 2 auf dem Schirm und dieser Zustand läßt sich auch durch mehrmaliges Drücken der ENTER-Taste oder der ESCAPE-Taste nicht mehr beheben. Wir haben mit den letzten beiden Eingaben zwei besondere Eigenschaften der CPC-Tastatur kennengelernt: den automatischen Funktionsaufruf und das umdefinierbare Tastenfeld. Um diese Fähigkeiten anwenden zu können, müssen wir uns zunächst ein wenig mit der Theorie der Tastaturabfrage auseinandersetzen.

Jede Taste hat eine Nummer. Q hat zum Beispiel die Nummer 67. Im Betriebssystem des Rechners existiert nur ein Unterprogramm, das nacheinander sämtliche Tasten abfragt und deren Status feststellt. Eine Taste kann dabei vier Zustände annehmen: Sie ist nicht gedrückt, sie ist gedrückt, SHIFT und die Taste sind gedrückt und CONTROL und die Taste sind gedrückt. Den aktuellen Wert jeder Taste können wir mit der INKEY-Funktion abfragen. Der folgende Einzeiler gibt ständig den Zustand der Q-Taste wieder.

```
10 PRINT INKEY(67): GOTO 10
```

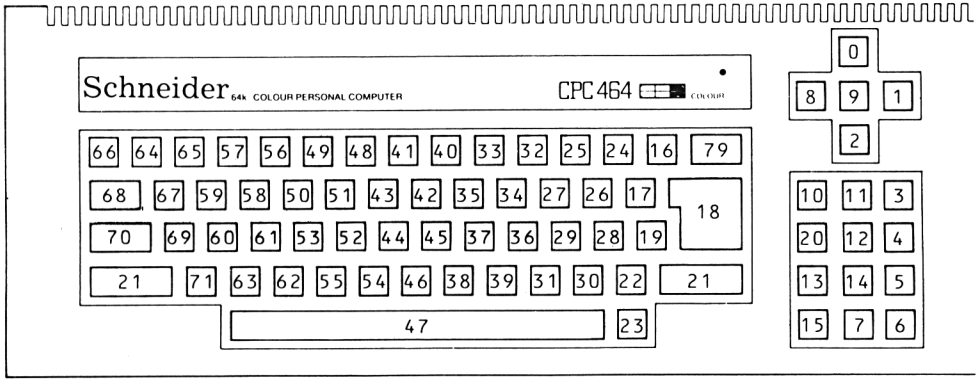
Solange wir die Q-Taste nicht betätigt haben, erscheint hier eine -1 auf dem Bildschirm. Drücken wir Q, so erhalten wir eine 0 als Wert dieser Funktion. Betätigen wir nun noch zusätzlich die SHIFT-Taste oder die CONTROL-Taste, so erhöht sich dieser Wert

auf 32 beziehungsweise 128. Betätigen wir beide Tasten gleichzeitig, das heißt SHIFT plus CONTROL plus Q, so ergibt sich die Summe aus den letzten beiden Tasten, das heißt 160.

Das Arbeiten mit der INKEY-Funktion ist eine sehr schnelle Art der Abfrage, wenn festgestellt werden soll, ob eine Taste und gegebenenfalls eine Zusatztaste gefrückt wurde. Sie bietet sich zum Beispiel für einfache Ja/Nein-Abfragen an. Hier müßte dann geprüft werden, ob die J- oder die N-Taste gedrückt wurden. Daneben kann sie immer dann Anwendung finden, wenn sehr schnell hintereinander mehrere Tasten bearbeitet werden müssen, wie das zum Beispiel in einer komfortablen Textverarbeitung oder bei einem Musikprogramm der Fall ist, bei denen die Tastatur als "KEYBOARD" benutzt wird.

```
INKEY(TN)
```

gibt also den jeweiligen Zustand der durch TN bezeichneten Taste wieder. TN ist die Tastaturnummer. Welche Taste mit welcher Tastaturnummer belegt ist, entnehmen Sie bitte dem nachfolgenden Schaubild.



Quelle: Bedienungshandbuch CPC 464

Bild 5.1 Die Tastaturnummern

So hat zum Beispiel die Leertaste die Nummer 47 und das Q, wie wir schon gesehen hatten, die Nummer 67. Die Zahlen in dem kleineren abgesetzten Zahlenfeld haben Nummern von 3 bis 20. Die Cursorsteuertasten liegen im Bereich zwischen null und neun. Bei der Betrachtung dieser Zahlen wird bereits eines deutlich: Was auf den ersten Blick gleich aussieht, nämlich die Tasten für die Zahlen in der Zehnertastatur, beziehungsweise in der oberen Reihe unserer Haupttastatur, ist nicht identisch. So fragen wir zum Beispiel mit

```
INKEY(13)
```

ab, ob die 1 auf unserer Zehnertastatur gedrückt wurde. Wir müssen jedoch

```
INKEY(64)
```

benutzen, wenn wir dieselbe Abfrage auf der Haupttastatur vornehmen wollen. Die Abfrage mit INKEY verläuft also in mehreren Punkten anders, als bei den anderen beiden Eingabemöglichkeiten mit INPUT oder INKEY$. Es wird nicht die Eingabe von Zeichen überprüft, sondern das Drücken auf eine Taste. Das Arbeiten mit INKEY erfordert eine Umformung, wenn zwischen Groß- und Kleinbuchstaben unterschieden werden soll. Bei der Eingabe eines a auf der Tastatur wird INKEY(69) = 0. Geben wir dagegen ein A ein, so erhalten wir 32 als Wert unserer Abfrage. Testen wir also den Tastendruck mit

```
IF INKEY(69) =  0 THEN ...
```

so tritt die nach dem THEN angefügte Handlung nur bei Drücken der A-Taste ohne SHIFT in Kraft. Wollen wir auch die Großbuchstaben erfassen, so müssen wir eine zweite Abfrage einführen, oder unsere Abfrage durch Einfügen einer zweiten Bedingung zu

```
IF INKEY(69) = 0 OR INKEY(69) = 32 THEN ....
```

ändern. Auch das Einlesen langer Texte wird mittels INKEY mehr als schwierig. Hier müßte nämlich für jeden Buchstaben der Tastatur eine separate Abfrage durchgeführt werden, wobei dann noch in Groß- und Kleinbuchstaben zu trennen wäre. Der INKEY-Befehl verfügt dafür aber über eine Reihe anderer Vorzüge. Zum ersten ist er sehr schnell. Eine Trennung nach der Schriftgröße ist anhand des Funktionswertes direkt möglich. Das Ergebnis der Abfrage ist

eine Zahl und kein STRING, was weniger Speicherplatz kostet. Außerdem können Zahlen mittels arithmetischer Operationen umgeformt werden, so daß aus dem Drücken einer Taste direkt eine Cursorbewegung oder ähnliches abgeleitet werden kann. Neben der Schnelligkeit der Abfrage hat das Arbeiten mit diesem Kommando noch einen wesentlichen Vorteil. Bei der INKEY-Abfrage können nämlich mehrere Tasten parallel abgefragt werden. Wir können also mehrere Tasten gleichzeitig drücken und erhalten dann bei der Abfrage dieser Tastennummern jedesmal den Wert 0 beziehungsweise, wenn wir gleichzeitig SHIFT oder CONTROL gedrückt hatten entsprechend höhere Werte. Dies ist notwendig für das mehrstimmige Musizieren auf der Computertastatur und ähnliche Anwendungen.

5.1.1 Umdefinieren der Tastatur

Nachdem wir festgestellt haben, in welchem Zustand sich unsere Taste befindet, stellt sich die Frage, wie die Tastatur mit der Bildschirmausgabe zusammenhängt, oder um es etwas einfacher auszudrücken:

Wer legt fest, welches Zeichen bei einem Tastendruck auf dem Bildschirm erscheint?

Dazu gibt es im Betriebssystem des Rechners ein Unterprogramm, den sogenannten KEY-MANAGER. Dieser ordnet jeder Taste in Abhängigkeit von ihrem Zustand verschiedene Zeichen, beziehungsweise eine Reihe von Zeichen zu. Diese werden danach auf dem Schirm von Bildschirmprogrammen dargestellt. An dieser Schnittstelle zwischen Tastatur und Bildschirm greifen wir nun ein, wenn wir die Belegung unserer Tasten ändern oder den Aufruf einer Funktion auf Tastendruck erreichen wollen. Dazu stehen uns im BASIC die Kommandos KEY und KEY DEF zur Verfügung. Mit KEY DEF stellen wir dabei den Zusammenhang zwischen einer Tastaturnummer und einem Bildschirmzeichen, das heißt einem CHARACTER dar. Der KEY DEF-Befehl hat dabei das folgende Format:

```
KEY DEF TN,REPEAT,NORMAL,SHIFT,CONTROL
```

TN ist die Tastaturnummer. Sie ist dem oben abgedruckten Schaubild zu entnehmen. An der nächsten Stelle finden wir nun REPEAT. Schreiben wir an diese Stelle eine Null, wird die REPEAT-Funktion für diese Taste ausgesetzt, das heißt, drücken wir diese Taste über eine längere Zeit, so wird das entsprechende Zeichen nicht mehr nach einer gewissen Verzögerung ständig wiederholt, sondern nur einmal ausgegeben. Steht an dieser Stelle eine eins, wird das entsprechende Zeichen wie gewohnt nach einiger Zeit wiederholt. Die Schnelligkeit der Wiederholung und die anfängliche Verzögerung können wir mit dem Kommando

```
SPEED KEY
```

für alle Tasten gemeinsam festlegen. SPEED KEY hat zwei Attribute. Am Anfang steht die Startverzögerung und nach dem Komma die Zeit bis zur nächsten Wiederholung. Die Angaben werden in fünfzigstel Sekunden gemacht.

```
SPEED KEY 10,2
```

bedeutet also, daß nach einer Verzögerung von einer fünftel Sekunde jede fünfundzwanzigstel Sekunde ein Zeichen ausgegeben wird. Mit dieser Geschwindigkeitsdefinition ist es allerdings schon sehr problematisch, bei der Wiederholung nur noch ein einzelnes Zeichen auf dem Bildschirm zu erhalten. Die letzten drei Argumente unseres KEY DEF-Befehls enthalten dann die Zeichen, die auf Tastendruck in den drei verschiedenen Ebenen erscheinen sollen. Da die Großbuchstaben A, B und C die Codes 65, 66 und 67 haben, können wir zum Beispiel diese mit

```
KEY DEF 67,1,65,66,67
```

auf die drei Ebenen der Q-Taste definieren. Drücken wir dann einfach auf die Taste, so erscheint ein A. Betätigen wir zusätzlich dazu die SHIFT-Taste, erscheint ein B, zusammen mit der CONTROL-Taste ein C. Bei dieser Auswahl ist die CONTROL-Taste vorherrschend, das heißt gleichzeitiges Drücken von SHIFT und CONTROL führt trotzdem zur Ausgabe eines C. Für die Belegung unserer Tastatur stehen uns die Nummern von 0 bis 255 zur Verfügung. Mit den Nummern 0 bis 31 erhalten wir die schon des öfteren erwähnten Systemsteuerzeichen, wie Cursor rechts, Bildschirm löschen etc. Ab 32 bis 125 einschließlich befindet sich ein Großteil des Buchstabenzeichensatzes des CPC 464: Groß- und Kleinbuchstaben, die Zahlen und die auf der Tastatur verfügbaren Sonderzeichen. Mit

Werten zwischen 128 und 159 erreichen wir die sogenannten EXPANSION CHARACTERS, auf die wir später noch eingehen werden.

Mit den Zahlenwerten zwischen 160 und 236 greifen wir wieder auf den normalen Zeichensatz des CPC 464 zurück. Hier finden sich neben griechischen Buchstaben auch eine Reihe interessanter Grafiksymbole, wie schräge Linien, schraffierte Felder, Blockgrafiken und Spielsymbole. Der Bereich von 237 an aufwärts ist den Spezialtastenfunktionen vorbehalten. Die Werte 240 bis 247 dienen zur Cursorsteuerung. Die Nummern 240 bis 243 bewegen den normalen Cursor. Die Nummern 244 bis 247 leisten dasselbe für den COPY-Cursor. Der Code 240 entspricht dabei dem Druck auf die Cursor-Taste. Definieren wir also

```
KEY DEF 67,1,240
```

läßt ein Druck auf die Q-Taste unseren Cursor nach oben wandern. Wenn man von diesen Spezialfunktionen einmal absieht, dürfte der Hauptanwendungsbereich für den KEY-Befehl im Umdefinieren der Tastatur zur Erreichung neuer Zeichen liegen. Hier bietet sich besonders der Bereich des griechischen Alphabets, sowie eine Auswahl aus der Blockgrafik an, um durch einfachen Tastendruck Grafiken, wenn auch mit verminderter Auflösung, auf dem Bildschirm erhalten zu können. Das Programm Key, das Sie auf der nächsten Seite abgedruckt finden, hat genau diesen Gedanken als Hintergrund. Es ermöglicht es, die Buchstaben auf unserer Haupttastatur mit neuen Zeichen zu belegen.

Zur Auswahl stehen zwei Varianten von Linien- und Blockgrafik, das griechische Alphabet, sowie natürlich die Groß-und die Kleinbuchstaben. Die ersten beiden DATA-Zeilen enthalten die Tastaturnummern, beginnend mit der uns ja schon bekannten 67 für Q, es folgt die 69 für W, die 58 für E und so weiter. Diese werden in das Feld nr hineingeladen. Als nächstes folgt dann die Abfrage der Belegung. Es wird festgestellt, welcher Zeichensatz für welche Ebene der Tastatur verwendet werden soll. Die nachfolgenden DATA-Zeilen enthalten die Zeichensätze mit den entsprechenden Codes. Sie werden in das Feld z(I,J) hineingeladen. Nachdem die fünf Zeichensätze eingelesen wurden, ist es ein leichtes, mittels 26 nacheinander ausgeführten KEY DEF-Kommandos die Buchstaben unserer Tastatur neu zu definieren. Dies geschieht in der Schleife in den letzten beiden Zeilen des Programms.

```
10 REM *********
20 REM ** KEY **
30 REM *********
40 CLS
50 DIM nr(26),z(5,26)
60 PRINT:PRINT "             Key-Definition":PRINT
70 PRINT "BG1 = Linien / Blockgrafik          (1)"
80 PRINT"BG2 = Linien / Blockgrafik           (2)"
90 PRINT"BG3 = griechisches Alphabet          (3)"
100 PRINT"BG5 = Grossbuchstaben                (4)"
110 PRINT"BG6 = Kleinbuchstaben                (5)"
120 DATA 67,59,58,50,51,43,42,35,34,27,69,60,61
130 DATA 53,52,44,45,37,36,71,63,62,55,54,46,38
140 RESTORE 120
150 FOR i=1 TO 26:READ nr(i):NEXT i
160 PRINT "Tastaturbelegung in 'NORMAL'-Ebene ";
170 INPUT n
180 IF n>0 AND n<6 THEN 210
190 PRINT "Falsche Eingabe !!!"
200 FOR i=1 TO 1500:NEXT i:GOTO 170
210 PRINT "Tastaturbelegung in 'SHIFT'-Ebene ";
220 INPUT s
230 IF s<1 OR s>5 THEN 240 ELSE 260
240 PRINT"Falsche Eingabe !!!"
250 FOR i=1 TO 1500:NEXT i:GOTO 220
260 PRINT"Tastaturbelegung in 'CONTROL'-Ebene ";
270 INPUT c
280 IF c<1 OR c>5 THEN 290 ELSE 310
290 PRINT"Falsche Eingabe !!!"
300 FOR i=1 TO 1500:NEXT i:GOTO 260
310 DATA 192,193,194,195,196,197,198,199,200,201,202,203,204
320 DATA 229,230,231,232,233,234,235,160,161,162,163,164,165
330 DATA 179,180,181,182,183,184,185,160,161,162,163,164,165
340 DATA 166,167,168,169,170,171,172,173,174,175,224,225,226
350 DATA 176,177,178,179,180,181,182,183,184,185,188,189,190
360 DATA 191,203,158,159,160,161,162,163,164,165,166,167,168
370 DATA 81,87,69,82,84,89,85,73,79,80,65,83,68
380 DATA 70,71,72,74,75,76,90,88,67,86,66,78,77
390 DATA 113,119,101,114,116,121,117,105,111,112,97,115,100
400 DATA 102,103,104,106,107,108,122,120,99,118,98,110,109
410 FOR i=1 TO 5:FOR j=1 TO 26:READ z(i,j):NEXT j,i
420 FOR i=1 TO 26:KEY DEF nr(i),1,z(n,i),z(s,i),z(c,i)
430 NEXT i
```

Drücken wir nun bei der Abfrage der Tastaturbelegung in der Normalebene auf 1, in der SHIFT-Ebene auf 3 und in der CONTROL-Ebene auf 5, erhalten wir beim Druck auf die Q-Taste einen Schrägstrich, bei gleichzeitigem Drücken von SHIFT ein griechisches Alpha und wenn wir die CONTROL-Taste betätigen, erscheint ein kleines q auf dem Bildschirm.

Die in den DATA-Zeilen abgelegten Zeichen sind natürlich nur ein Vorschlag. Sie können diese anhand des Zeichensatzes des CPC anders definieren. So fehlen zum Beispiel in unseren Zeichensätzen die Zeichen mit den Nummern von 205 bis 223. Dabei handelt es sich um einen Teil der interessanten Blockgrafik, der im wesentlichen voll ausgefüllte Linien und Schrägen enthält. Um zum Beispiel diesen Zeichensatz zusätzlich auf die Tastatur legen zu können, gehen wir wie folgt vor.

Zuerst nehmen wir den neuen Zeichensatz als BG 7 oben in unserer Aufstellung auf. Um die dazugehörige Ziffer 6 auch eingeben zu können, müssen wir die Fehlerabfragen, in denen geprüft wird, ob N, S und C sich im erlaubten Bereich von 1 bis 5 befinden, entsprechend um eins erhöhen. In der drittletzten Zeile unseres Programms lassen wir dann I von 1 bis 6 laufen, wobei wir allerdings beachten müssen, daß zuvor die Matrix (das ARRAY) z am Anfang unseres Programms auf 6,26 dimensioniert werden sollte. In die neuen Zeilen zwischen 460 und 470 können wir nun unseren neuen Datensatz hineinschreiben. Nach DATA geben wir zunächst das Zeichen an, das auf Drücken der Q-Taste erscheinen soll, danach erfolgt dasselbe für W und so weiter, Zeile für Zeile, von links nach rechts. Sie sollten dabei beachten, daß wie schon gesagt, die Zeichen zwischen 128 und 159 und größer als 237 von der Benutzung ausgeschlossen sind. Auf die Bedeutung der Zeichen von 237 an aufwärts sind wir schon eingegangen. Wir wollen uns deshalb nun mit dem anderen Bereich beschäftigen.

In diesem liegen die sogenannten EXPANSION CHARACTERS. Trifft der CPC bei der Übersetzung der Tastatur auf eine Zahl zwischen 128 und 159, so stellt er nun nicht das entsprechende Bildschirmzeichen dar, sondern assoziiert die Taste mit einem STRING, das heißt es werden mehrere Zeichen auf dem Bildschirm ausgegeben. Das ganze hört sich ziemlich kompliziert an, ist es aber nicht. Ein kleines Beispiel macht dies deutlich. Mit KEY definieren wir einen solchen EXPANSION CHARACTER, mit KEY DEF legen wir ihn auf die Tastatur. Geben wir also

```
KEY 140,"Alles klar!?" <ENTER>
KEY DEF 67,1,140 <ENTER>
```

ein, erscheint nun nach Drücken auf Q der Text "Alles klar!?" auf unserem Bildschirm. Wir können jetzt mit einem Tastendruck eine ganze Serie von Buchstaben und Zahlen gleichzeitig auf dem Bildschirm erscheinen lassen. Hängen wir an unseren STRING noch ein

```
CHR$(13)
```

an, wird er als Befehlszeile interpretiert und sofort ausgeführt. Definieren wir

```
KEY 139,"LIST-250"+CHR$(13)
KEY DEF 67,1,139
```

wird jetzt das Drücken der Q-Taste mit dem EXPANSION CHARACTER 139 verbunden, LIST -250 auf dem Bildschirm ausgegeben und dieses durch das angehängte CHR$(13) sofort ausgeführt. Damit ist es nun möglich, Programme ganz oder teilweise durch einfachen Tastendruck zu listen. Wir können auch häufig benutzte und etwas längere Wörter wie PAPER, PEN, INK, WINDOW# auf die Tastatur legen und so unsere Schnelligkeit beim Schreiben von Programmen erheblich vergrößern.

Bei der Belegung der Tastatur sind wir dabei natürlich nicht auf die Q-Taste beschränkt, sondern können jede Taste mit Ausnahme von ESCAPE für unsere Umdefinition benutzen. Für die Definition unseres EXPANSION STRINGS stehen uns, wie schon gesagt, die Nummern von 128 bis 159 zur Verfügung. Hierbei ist zu beachten, daß ein EXPANSION CHARACTER maximal 32 Zeichen lang sein kann. Es ist nicht möglich, einen ganzen Roman auf Tastendruck auf den Bildschirm zu bringen.

Auch bei der Gesamtzahl der EXPANSION CHARACTERS sind uns Grenzen gesetzt. Insgesamt können wir nur 120 Zeichen speichern. Dies sind also ungefähr vier Zeichen pro EXPANSION CHARACTER, wenn wir alle Zahlen belegen wollen. Eine Anwendung dieser Technik stellt das nachfolgend abgedruckte Programm KEY DEF dar. Es definiert die abgesetzte Zehnertastatur neu. Wir haben nur die Normalebene und die SHIFT-Ebene benutzt. In der Normalebene liegen in den oberen drei Reihen die häufigsten verwendeten Cursorsteuerzeichen, das Zeichen für Bildschirm löschen und für Home, das heißt die Bewegung des Cursors in die linke obere Bildschirmecke. In

der SHIFT-Ebene befinden sich die am häufigsten benutzten Befehle, wie die Farbbefehle PEN, PAPER und INK, die Umschaltung zwischen den beiden MODES 1 und 0 und RENUM. Auch längere Befehlsworte wie IF INKEY, WINDOW und SOUND finden sich hier.

Die letzte Zeile schließlich ist dem Editieren vorbehalten. Wir haben hier häufig benutzte LIST-Befehle auf die Tastatur gelegt. Durch Drücken auf ENTER werden die Zeilennummern bis 250 gelistet. Der Punkt bringt uns den Bereich zwischen 250 und 500 ins Bild und die Programmzeilen von mehr als 500 erhalten wir durch Drücken auf die 0. Dabei ist zu beachten, daß die LIST-Befehle und auch RENUM durch das angehängte CHR$(13) sofort ausgeführt werden. Wir brauchen uns noch nicht einmal um das RETURN zu kümmern.

Bei der Definition der Tastatur nutzen wir den Trick aus, daß die Tasten unser Zehnertastatur schon vom Einschalten an mit den EXPANSION CHARACTERS von 128 bis 140 belegt werden. Wir brauchen deshalb nur noch diese EXPANSION STRINGS zu definieren. Dies geschieht in den Zeilen 230 bis 350. Die letzten drei Zeilen dienen der Übertragung der Systemsteuerzeichen. Wir legen diese mit dem KEY DEF-Kommando auf die entsprechenden Tasten unserer Tastatur. An der ersten Stelle des KEY DEF-Kommandos steht dabei wieder die Tastennummer, danach die 1 für eingeschaltetes REPEAT und dann die Nummern der entsprechenden Systemsteuerzeichen. Die ersten vier steuern den Cursor, danach folgt die 30 für Cursor Home und die 12 für Bildschirm löschen. Die Belegung der einzelnen Tasten braucht man sich nicht herauszuschreiben, da diese durch die PRINT-Befehle am Anfang des Programms schön geordnet auf dem Bildschirm dargestellt werden.

```
10 REM *************
20 REM ** KEY DEF **
30 REM *************
40 MODE 1:INK 0,0:INK 1,24:BORDER 0:PAPER 0:PEN 1
50 PRINT"Belegung des Zahlenfeldes in der :"
60 PRINT"Normalebene        Shiftebene"
70 PRINT"--------------------------------"
80 PRINT"7: CURSOR hoch   , paper "
90 PRINT"8: CURSOR links  , pen"
100 PRINT"9: CURSOR rechts , pen "
110 PRINT"4: CURSOR runter , mode 2"
120 PRINT"5: CURSOR CLS    , mode 1"
130 PRINT"6: RENUM         , RENUM"
140 PRINT"1: CURSOR HOME   , if inkey("
150 PRINT"2: window#       , window#"
160 PRINT"3: sound         , sound "
170 PRINT"0: list 500-     , list 500-"
180 PRINT".: list 250-500  , list 250-500"
190 PRINT"ENTER : list -250, RUN"
200 REM ********************
210 REM ** Tastenbelegung **
220 REM ********************
230 KEY 139,"list-250"+CHR$(13)
240 KEY 140,"run"+CHR$(13)
250 KEY 138,"list 250-500"+CHR$(13)
260 KEY 128,"list 500-"+CHR$(13)
270 KEY 132,"mode 2"+CHR$(13)
280 KEY 129,"if inkey("
290 KEY 130,"window#"
300 KEY 131,"sound "
310 KEY 135,"paper "
320 KEY 136,"pen "
330 KEY 137,"ink "
340 KEY 133,"mode 1"+CHR$(13)
350 KEY 134,"renum"+CHR$(13)
360 KEY DEF 20,1,10:KEY DEF 10,1,11
370 KEY DEF 11,1,8:KEY DEF 3,1,9
380 KEY DEF 13,1,30:KEY DEF 12,1,12
```

Ein Vorteil der Tastaturbelegung mit KEY und KEY DEF ist auch, daß sie mit NEW nicht aus dem Programmspeicher gelöscht werden. Wir können unser Programm also nach einmaligem Durchlauf getrost löschen, und uns ein neues Programm zur Bearbeitung in den Speicher holen, beziehungsweise schreiben. Die einmal gesetzten KEY-Werte bleiben bis zu einer erneuten Definition oder bis zum Ausschalten des Rechners erhalten.

Mit diesen Fähigkeiten sind wir noch nicht am Ende angelangt. Weitere Möglichkeiten ergeben sich nämlich bei der Kombination von KEY und KEY DEF mit anderen Befehlen, speziell dem SYMBOL-Kommando. Waren wir bis jetzt auf den vorgegebenen Zeichensatz des CPC für unsere Tastatur beschränkt, so können wir diesen mit SYMBOL umdefinieren und so neue von uns entworfene Zeichen definieren. Diese können wir wieder mit KEY DEF auf die Tastatur übernehmen. Als kleine Aufgabe können Sie ja einmal mit SYMBOL, beziehungsweise dem Programm Zeichendefinition, die deutschen Umlaute entwerfen und diese dann mit KEY DEF auf die entsprechenden Tasten legen. Danach brauchen wir nur noch die Y- und die Z-Taste zu vertauschen und aus unserer CPC Rechnertastatur wird eine deutsche Schreibmaschinentastatur. Dies erreichen wir mit zwei kurzen Befehlen:

```
KEY DEF 43,1,122,90
KEY DEF 71,1,121,89
```

43 und 71 sind die Tastaturnummern für Y und Z. Die Bildschirmcodes für Z lauten 122 für ein kleines z und 90 für ein großes Z (siehe Handbuch). 121 und 89 bilden die analogen Werte für Y.

5.2 Tips und Tricks für das Arbeiten mit der Kassette

Mit der Anwendung der Kassette bei der Datenspeicherung haben wir uns in den Kapiteln über die Text- und Datenverwaltung schon ausgiebig beschäftigt. Wir wollen uns nun mit den Anwendungsmöglichkeiten der Kassette bei der Programmspeicherung und hier speziell beim automatischen Ändern von Programmen und dem Einfügen von Programmodulen beschäftigen. Am Ende dieses Kapitels geben wir dann noch einige Praxistips für den Umgang mit der Kassette be-

ziehungsweise dem Speichermedium Band.

Die Kommandos SAVE und LOAD werden Sie sicherlich schon beim Speichern und Laden von Programmen benutzt haben. Beim Laden eines Programms mit LOAD wurde dabei immer das alte Programm im Arbeitsspeicher überschrieben. Der CPC verfügt über einen MERGE-Befehl. Mit diesem ist es möglich, Programme in andere Programme einzufügen. Enthält das neue Programm Zeilennummern, die auch im alten Programm vorhanden sind, so werden die des alten Programms überschrieben. Befindet man sich daher im Unklaren darüber, ob Zeilen überschrieben werden, sollte man vor dem "MERGEn" das alte Programm auf hohe Zeilennummern umnumerieren. Dies geht zum Beispiel mit RENUM 10000. Dabei wird natürlich vorausgesetzt, daß das zweite Programm keine Zeilennummern höher als 10000 hat.

Der MERGE-Befehl bietet sich besonders an, um häufig benutzte Programm-Module schnell in ein zu entwickelndes Programm einbauen zu können. Ein Beispiel dafür wäre eine Routine zum Abspeichern und Lesen von Daten. Das Grundprinzip einer solchen Routine ist immer identisch. Zunächst muß ein FILE auf den Kassettenrecorder geöffnet werden, und zwar beim Einlesen mit dem Befehl OPENIN, sonst mit dem Befehl OPENOUT. Danach erfolgt das Einlesen der Daten in einer FOR-TO-Schleife, wonach am Ende das Programm mit CLOSEIN beziehungsweise CLOSEOUT abgeschlossen wird. Noch interessanter wird es, falls man die englischen Fehlermeldungen unterdrücken will. Dies geht relativ einfach, indem man dem Dateinamen ein Ausrufezeichen voranstellt. So werden bei

```
OPENIN "ABCDE"
```

die Aufforderung zum Drücken der PLAY-Taste und auch Fehlermeldungen ausgegeben. Bei

```
OPENIN "!ABCDE"
```

werden diese unterdrückt. Dies kann man dazu benutzen, deutsche Aufforderungen und Fehlermeldungen auszugeben. Der Ablauf ist dabei folgendermaßen (beim Einladen von Daten):

```
.
.
.
1000 OPENIN "!Dateiname"
1010 PRINT "Bitte druecken Sie die PLAY-Taste und
danach eine andere Taste"
1020 IF INKEY$=""THEN 1020
1030 FOR I = 1 TO 12:INPUT#9,M(I):NEXT I
1040 PRINT "Dateneingabe beendet !"
.
.
.
```

Mit MERGE können wir dieses Programmodul nun in unser Programm einfügen. Besonders bei längeren Dateneingaben, wenn mehrere Datenblöcke eingelesen werden, ist dies von Vorteil. Andere Anwendungsbereiche für das Arbeiten mit Programmodulen sind grafische Unterprogramme und mathematische Routinen, wie die oft benutzte Berechnung der Extremwerte (Min/Max-Berechnung) einer Menge von Zahlen. Mit dem MERGE-Befehl können wir beim Editieren, das heißt dem Verändern von Programmen, Programm-Module beliebig einladen und gegebenenfalls dann an unser Programm anpassen.

Das CPC BASIC verfügt noch über eine weitere sehr interessante Variante des MERGE-Befehls: den Befehl CHAIN MERGE. Dieser funktioniert genauso wie der MERGE-Befehl, allerdings läuft jetzt das Programm automatisch ab. Laden wir ein Programm mit

```
CHAIN MERGE "Programmname",100
```

wird dieses Programm in das alte Programm eingeladen und dann ab Zeile 100 des neuen zusammengefügten Programms mit der Ausführung begonnen. Mit diesem Befehl haben wir die Möglichkeit, sich selbst ändernde Programme zu schaffen. Das wohl bekannteste Anwendungsfeld für diesen Befehl ist die Erzeugung von OVERLAY- und Nachladeprogrammen. Bei einem Nachladeprogramm wird am Anfang ein kleines Programm in den Speicher geholt, welches dann gegebenenfalls über ein Menü andere Programme nachlädt. Dies könnte zum Beispiel ein Verwaltungsmenü sein, was auf die Programme Textverarbeitung, Adreßkartei, beziehungsweise Haushaltsbudgetberechnung zurückgreift. Am Anfang wird ein kleines Programm in den Speicher geladen, das zum Beispiel wie das Nachstehende nur aus elf Zeilen besteht.

```
10 REM ***********
20 REM ** Menue **
30 REM ***********
40 PRINT "TEXTVERARBEITUNG   (1)"
50 PRINT "ADRESSKARTEI       (2)"
60 PRINT "BUDGETBERECHNUNG   (3)"
70 z$=INKEY$:IF z$="" THEN 70
80 IF z$="1" THEN CHAIN MERGE"TEXT",200
90 IF z$="2" THEN CHAIN MERGE"ADRESS",500
100 IF z$="3" THEN CHAIN MERGE"BUDGET",1000
110 GOTO 70
```

Am Anfang gibt das Programm die Auswahlmöglichkeiten aus. Die Auswahl erfolgt dabei Eintippen der zugehörigen Zahl. In Zeile 70 wird die Tastatur mit der INKEY$-Funktion abgefragt. Erfolgt hier keine Eingabe, ist der STRING z$ leer, geht es zurück nach Zeile 70. Ansonsten werden die einzelnen Programme geladen. Dabei wurde vorausgesetzt, daß diese mit den Zeilen 200, 500 und 1000 beginnen sollen. An dieser Stelle nimmt das neu eingeladene Programm dann seine Arbeit auf. Zeile 110 stellt eine Fehlerkorrekturroutine dar. Bei einer Fehlbedienung - wenn keine der Zahlen 1 bis 3 eingegeben wurde - erfolgt der Rücksprung nach 70.

Dieses Programm ist natürlich noch erweiterungsfähig. Zum einen könnte das gewählte Programm noch einmal zur Kontrolle ausgegeben werden. Andererseits wäre es auch noch möglich, die deutschen Aufforderungen zum Drücken der PLAY-Taste wie oben einzubauen.

Eine andere interessante Anwendung des MERGE-Befehls findet sich im Spielebereich. Oft zeichnen sich Spiele durch eine extrem aufwendige und deswegen auch meist schöne Titelgrafik aus. Der Benutzer sollte ruhig etwas Zeit zum Betrachten haben. Andererseits wäre es unpraktisch, erst das gesamte Programm inklusive der Titelgrafik zu laden, wodurch eine ziemliche Wartezeit entsteht und dann eine künstliche Zeitverzögerung nach dem Aufbau des Titels einzubauen, um ein ungestörtes Betrachten des Titelbilds zu ermöglichen. Sinnvoller wäre es, erst den Titel einzuladen und die dann benötigte Zeit zum Einladen des Restprogramms als Zeitverzögerungsschleife zum Betrachten des Titelbilds zu benutzen. Dies ist natürlich mit der obengenannten Technik kein Problem. Man lädt zuerst das Titelprogramm, das den Titel ausgibt und das danach automatisch mit CHAIN MERGE das Hauptprogramm nachlädt, welches dann wiederum mit der hintenangegebenen Zeilennummer automatisch gestartet wird. Mit dem CHAIN MERGE-Befehl kann man übri-

gens auch aus jedem Programm einen "Auto-Starter" machen, also ein Programm, das sich selbst nach dem Einladen startet. Statt des normalen

```
LOAD"Programmname"
```

verwenden wir

```
CHAIN MERGE "Programmname",10
```

wobei davon ausgegangen wurde, daß das Programm auf Zeile 10 starten soll. Natürlich ist es auch möglich, mit dieser Befehlsvariante ein Programm in der Mitte beginnen zu lassen, zum Beispiel bei Zeile 1000. In diesem Fall wird der CHAIN MERGE-Befehl als Direktkommando zu einem erweiterten RUN"-Befehl, mit dem ja auch das automatische Laden und Anlaufen eines Programms, allerdings nur von der jeweiligen ersten Zeile an, möglich ist. Als Abschluß zu diesem Kapitel jetzt noch einige praktische Tricks für das Arbeiten mit der Kassette.

5.2.1 Das Beheben von Ladefehlern

Der CPC benutzt wie viele andere Homecomputer einen umgebauten Kassettenrecorder als Hauptspeichermedium. Dies hat Vor- und Nachteile. Zum einen sind die Kassetten fast an jedem Ort verfügbar und die Verwendung eines in Massenproduktion gefertigten Kassettenrecorders ist relativ billig. Das Problem dabei ist, daß Kassettenrecorder natürlich ursprünglich nicht für die Datenspeicherung entwickelt wurden. Daher ergeben sich bei der Benutzung als Datenrecorder manchmal Probleme. Wir werden auf diese im folgenden etwas näher eingehen und Lösungsmöglichkeiten aufzeigen.

Beim Laden von Programmen treten im wesentlichen zwei Fehler auf. Der erste äußert sich darin, daß nach der Aufforderung PLAY zu drücken trotz gedrückter PLAY-Taste nichts passiert. Dies kann zwei Ursachen haben. Zum einen kann es sein, daß Sie die PLAY-Taste schon gedrückt hatten und dann schnell zweimal hintereinander die ENTER-Taste betätigten, mit dem Wissen, die PLAY-Taste ist ja gedrückt. Da der CPC 464 nach dem ersten Drücken der ENTER-Taste erst eine relativ komplizierte Umformung der Daten vornimmt, die einige Zeit braucht, ist er aus dieser Routine gegebenenfalls

noch nicht zurück, bevor Sie Ihr zweites ENTER eingegeben haben. Er hat dieses dann nicht gespeichert und wartet folglich immer noch auf den angegebenen zweiten Tastendruck.

Die andere Möglichkeit kann darin bestehen, daß die eingeschobene Kassette klemmt, das heißt sich nicht drehen läßt. Dies kann nun entweder an Ihrer Kassette liegen oder an der Mechanik des Kassettenrecorders. Meistens liegt es an beidem, aber auch hier gibt es eine relativ einfache Lösung. Oft reicht schon das einfache Herausziehen und Wiedereinschieben der Kassette. In besonders hartnäckigen Fällen muß man die Kassette einmal kurz mit der Unterkante auf den Tisch schlagen. Dadurch werden die Bandwicklungen enger zusammengeschoben und das Band läuft dann wieder besser. Ein derartiger Fehler läßt sich relativ einfach beheben.

Schwieriger wird es, wenn das Band zwar läuft, aber durch Fehler beim Aufsprechen oder beim Lesen permanent Lesefehler (READ-ERRORS) auftreten. Dies kann mehrere Gründe haben. Zum einen setzen sich im Laufe der Zeit auf dem Tonkopf und auch auf dem Band kleine Staubteilchen und Abrieb vom Band ab. Diese halten das Band vom Tonkopf ab und dadurch nimmt die Lautstärke und auch die Deutlichkeit der als Töne gespeicherten Daten ab.

Lösungsstrategie:

Man lege ein Reinigungsband ein, oder reinige den Tonkopf mit Alkohol. Dazu taucht man ein Wattestäbchen in Alkohol und säubert damit die Oberfläche des Tonkopfs. Dabei ist unbedingt darauf zu achten, daß Ihr Wattestäbchen nicht fusselt. Denn dies würde gerade dem Zweck dieser Aktion zuwiderlaufen. Im Zweifelsfall sollten Sie hier lieber einen sauberen Lappen benutzen.

Ein anderer Grund kann darin bestehen, daß der Tonkopf nicht mehr richtig justiert ist. Dies ist meistens der Fall und rührt daher, daß der Tonkopf nur durch eine einzige Schraube auf Position gehalten wird und diese sich im Laufe der Zeit verdrehen kann, wodurch der Tonkopf abgesenkt oder angehoben wird. Diese Verschiebung können wir durch nachträgliches Drehen an der Schraube wieder aufheben.

Wir gehen dabei wie folgt vor: Zunächst benötigen wir einen Uhrmacherschraubenzieher No. 2 oder No. 3. Diesen erhalten wir in fast jedem Elektronikfachgeschäft und auch manche Werkzeuganbieter führen sie. Entfernen Sie nun zunächst die Kassette aus dem

Kassettenlaufwerk und lassen Sie bitte den Deckel geöffnet. Wenn Sie nun auf die PLAY-Taste drücken, schiebt sich aus dem Bereich der Tastatur ein Mechanikteil nach vorne. In der Mitte sehen Sie einen silbernen Würfel, das ist der Tonkopf. Links daneben befindet sich ein weißlicher kleiner Würfel, das ist der Löschkopf, den wir nicht beachten brauchen. Auf der linken Seite des Tonkopfs befindet sich eine kleine Schraube. Wir können sie durch den Schlitz in der Abdeckung des Kassettenrecorders erreichen.

Zunächst drehen wir diese Schraube ganz hinein und zwar im Uhrzeigersinn. Nun legen wir ein bespieltes Band ein, und spulen auf die Stelle vor, bei der gerade der Anfangspiepser ertönt. Der VOLUME-Regler an der rechten Seite des Gehäuses sollte dabei auf maximaler Lautstärke stehen. Während dieses Piepsers drehen wir nun die Schraube entgegen dem Uhrzeigersinn solange, bis die Lautstärke maximal wird. Bei richtig eingestellten Geräten müßten das zwei bis drei Umdrehungen entgegen dem Uhrzeigersinn sein. Nachdem wir die Qualität unserer Einstellung mit einigen Kassetten, auf denen es vorher Ladefehler (LOAD ERRORS) hagelte, überprüft haben, sollten wir die Schraube mit Lagellack oder ähnlichem gegen Verdrehen sichern. Die Einstellung ist beendet.

Diese Methode nützt allerdings nur etwas gegen Fehler in der Mechanik des Kassettenrecorders. Eine andere Ursache für LOAD ERRORS können sogenannte "drop outs" auf Ihrem Band sein. Dies sind Stellen, die nicht entsprechend magnetisiert werden können, so daß der Ton an dieser Stelle eine Unterbrechung hat. Diese ist so kurz, daß wir sie nicht hören können. Da der Computer bis zu 2000 Datenelemente pro Sekunde abspeichert, führt dies dazu, daß bestimmte Daten nicht mehr lesbar sind, was zum Abbruch des Ladevorgangs führt. Hier hilft nur der Wechsel auf ein anderes LOAD-ERROR-freies Band. Das Band, bei dem dies aufgetreten ist, sollte gekennzeichnet werden, um ein späteres nochmaliges Auftreten desselben Problems zu vermeiden.

Trotz all dieser Maßnahmen sollte man dennoch jedesmal nach dem Speichern speziell längerer Programme diese mit dem CAT-Befehl überprüfen. Dazu gibt man CAT und RETURN ein. Der Computer überprüft dann die auf dem Band gespeicherten Programme und gibt aus, um welchen Typ es sich handelt. Ein Dollarzeichen bedeutet ein Programm. Am Ende des Vergleichs gibt der CPC noch aus, ob das Programm, wenn es geladen worden wäre, störungsfrei in den Speicher gekommen wäre. Normalerweise erscheint an dieser Stelle ein freundliches "ok", womit wir sicher sein können, daß unser wert-

volles Programm in gutem, das heißt ladebereitem Zustand ist. Außerdem kann es nie schaden, besonders bei längeren Programmen eine Sicherungskopie auf einer speziell dafür vorgesehenen Kassette abzulegen.

Ein einfaches Rechenexempel mag den Sinn dieses Vorgehen erläutern. Ein Programm, an dem Sie vier Stunden gesessen haben, und das durch einen Ladefehler nicht mehr ladbar ist, kostet Sie vier Stunden Zeit. Das Abspeichern eines mittellangen Programms mit SPEED RIGHT 1, das heißt im Schnell-MODE, kostet Sie ungefähr eine Minute. Dabei ist noch zu bedenken, daß in dieser Minute natürlich nur der CPC arbeitet, während Sie sich bereits wieder neue Gedanken über die Veränderung Ihres Programms machen können. Es empfiehlt sich in jedem Fall, nach einiger Zeit eine Kopie aufs Band zu schreiben. Außerdem hat dieses Vorgehen den Vorteil, daß wir bei falschen Änderungen an unserem Programm einfach die letzte Version noch einmal einladen können.

5.3 Die Echtzeitverarbeitung

Zum Abschluß dieses Kapitels wollen wir noch kurz auf die Möglichkeiten des CPC bei der Echtzeitverarbeitung eingehen. Als Echtzeitverarbeitung bezeichnet man ein Verhalten des Computers, bei dem sich dieser unserer gewohnten externen Zeit anpaßt. Normalerweise ist dies nicht der Fall, denn jeder Computer arbeitet vom Einschalten an bis zum Ausschalten ständig. Er legt Daten ab, bedient den Bildschirm oder verharrt, während er auf unsere Eingabe wartet, in einer Tastaturabfrageschleife. Je nach Komplexität eines auszuführenden Befehls braucht er für diesen eine unterschiedlich lange Zeit. Es ist daher relativ schwer zu kalkulieren, nach wieviel Sekunden oder hundertstel Sekunden der CPC einen bestimmten Programmteil durchlaufen hat und dann wieder zu einem bestimmten Punkt zurückkehrt.

Dies schafft in den Fällen, in denen wir die interne Arbeitszeit des CPC mit unserer externen Zeit koordinieren wollen, größere Probleme. Dies ist bei allen zeitabhängigen Programmen der Fall, zum Beispiel bei einem Programm, das die Uhrzeit darstellen soll oder auch bei Routinen, die nach einer bestimmten Zeit einen Vor-

gang auslösen sollen. Im Gegensatz zu den meisten anderen Computern verfügt der CPC über zwei Befehle, mit denen sich diese Probleme exzellent lösen lassen: die Kommandos EVERY und AFTER.

Beide produzieren einen sogenannten INTERRUPT, das ist eine Unterbrechung eines Programms nach einer bestimmten vorher definierten Zeit. Das Programm wird nicht mit einem Ready beendet, sondern es erfolgt der Sprung in einen anderen Programmteil. Aus diesem können wir dann bei geeigneter Gestaltung der Befehle wieder in das Hauptprogramm zurückkehren. EVERY sorgt dabei für eine fortwährende Unterbrechung, AFTER erzeugt nur einen einzelnen INTERRUPT.

Basis beider Befehle sind vier Zeitgeber, sogenannte TIMER. Sie haben die Nummern von 0 bis 3 und sind im Endeffekt setzbare elektronische Zählwerke. Auf diese können wir mit den Kommandos EVERY und AFTER zurückgreifen. Die Berechnung der Zeiteinheiten geschieht in fünfzigstel Sekunden. Geben wir

```
AFTER 50,1 GOSUB 100
```

ein, geht der CPC eine Sekunde, nachdem er diesen Befehl ausgeführt hat, zur Zeile 100. Die erste Stelle des Kommandos gibt den Zeitwert in fünfzigstel Sekunden an, die zweite Stelle den benutzten TIMER. Er führt die auf Zeile 100 folgenden Befehle aus, und kehrt dann in das Hauptprogramm zurück. Das nachfolgende Programm demonstriert das Prinzip:

```
10 AFTER 50,1 GOSUB 100
20 PRINT "A"
30 GOTO 20
100 PRINT "Eine Sekunde ist vorueber !":RETURN
```

Am Beginn des Programms wird der TIMER Nummer 1 auf 50 gesetzt. Danach läuft der CPC in einer Dauerschleife in den Zeilen 20 und 30, und gibt laufend den Buchstaben A aus. Nach Ablauf einer Sekunde tritt dann der INTERRUPT ein. Der Computer springt zur Zeile 100 und druckt den dort angegebenen Text aus.

Diese drei Zeilen bilden natürlich noch keine sinnvolle Anwendung. Das Prinzip ist dafür aber gut erkennbar. Aus einem Hauptprogramm erfolgt, nachdem die vorgegebene Zeit abgelaufen ist, der Sprung in das Unterprogramm. Diese kann dann zusätzliche oder ergänzende Aufgaben wahrnehmen. Danach wird zu der Stelle zurück-

gesprungen, an der die Bearbeitung des Hauptprogramms abgebrochen wurde. AFTER sorgt für das einmalige Ablaufen dieses Vorgangs. Mit EVERY wiederholt sich das Ganze jede Minute. Um eine mit EVERY eingeleitete Unterbrechungroutine zu beenden oder einen mit AFTER gesetzten TIMER vor dem Zeitablauf zu stoppen, besitzt das CPC-BASIC den Befehl REMAIN. Mit

```
REMAIN(0)
```

wird der TIMER 0 zurückgesetzt und der Unterbrechungsvorgang damit gelöscht. REMAIN(1) bis REMAIN(3) setzten die anderen drei TIMER zurück.

Diese Befehle eröffnen uns ungeahnte Möglichkeiten. Das häufigste Anwendungsbeispiel für den Befehl EVERY ist der fortlaufende Sprung aus einem Hauptprogramm in ein Unterprogramm, aus dem nach Abarbeitung einer bestimmten Befehlskette wieder in das Hauptprogramm zurückgesprungen wird. Das typischste Beispiel dafür sind Uhrenprogramme. Es gibt zwar eine Reihe von Programmen, die nichts anderes tun, als eine Uhr auf dem Bildschirm des Computers anzuzeigen, die Verfasser halten derartige Programme nicht gerade für sinnvoll. Schön wäre es, wenn zum Beispiel bei Simulationen oder bei Spielen, die in Realzeit ablaufen sollen, die Uhrzeit permanent in das Spiel eingeblendet werden könnte. Dies leistet das nachfolgend abgedruckte Programm EVERY DEMO. Es arbeitet mit drei WINDOWS, wovon eigentlich nur das dritte zwingend ist. Die anderen wurden nur definiert, um zu einer schönen Bildschirmgestaltung kommen zu können.

Nach dem üblichen Setzen der Farben und Löschen der einzelnen WINDOWS beginnt ab Zeile 80 das eigentliche Programm. Als Hauptprogramm wurde hier eine Sinusberechnung gewählt. In einer Dauerschleife, die von X=0 bis unendlich läuft, werden die X- und nachfolgend die Sinus-X-Werte ausgegeben. In Abhängigkeit vom zweiten TIMER wird jede Sekunde in das Zeitunterprogramm ab Zeile 120 gesprungen. Dieses erhöht den Sekundenwert t1 um 1 , prüft dann, ob bereits 60 Sekunden verstrichen sind und setzt gegebenenfalls den Minutenwert höher. Dieselbe Kontrolle erfolgt danach für die Minuten und Stunden. Die Zeilen 150 bis 170 bauen dann aus diesen drei Variablen die Uhrzeit als STRING zusammen. In Zeile 180 wird dieser schließlich an der richtigen Stelle ausgegeben.

```
10 REM ****************
20 REM * every - demo *
30 REM ****************
40 INK 0,0:INK 1,2:INK 2,6:INK 3,21
50 WINDOW#0,1,40,4,25:WINDOW#3,31,40,1,3:WINDOW#2,1,30,1,3
60 PAPER#0,0:PEN#0,2:PAPER#2,0:PEN #2,3:PAPER#3,1:PEN#3,2
70 CLS#0:CLS#2:CLS#3
80 LOCATE#2,9,2:PRINT#2,"Uhrzeit - Demo"
90 EVERY 50,2 GOSUB 120
100 x=x+1 :PRINT x,SIN(x)
110 GOTO 100
120 t1 = t1+1:IF t1=60 THEN t2=t2+1:t1=0
130 IF t2=60 THEN t3=t3+1:t2=0
140 IF t3=24 THEN t3=0
150 t$=RIGHT$("0"+MID$(STR$(t3),2),2)+":"
160 t$=t$+RIGHT$("0"+MID$(STR$(t2),2),2)
170 t$=t$+":"+RIGHT$("0"+MID$(STR$(t1),2),2)
180 LOCATE#3,2,2:PRINT#3,t$
190 RETURN
```

Die Anwendung der Befehle EVERY und AFTER ist nicht auf reine Uhrzeitprogramme beschränkt. Eine weitere, sehr interessante Anwendung liegt darin, Programme parallel ablaufen lassen zu können. Dies wird als MULTI-TASKING bezeichnet, das heißt der Computer nimmt mehrere Aufgaben quasiparallel wahr. Zum Beispiel könnte in einer langwierigen Berechnung parallel jede fünftel Sekunde einmal die Tastatur abgefragt werden, um zu prüfen, ob bereits neue Anweisungen des Bedieners vorliegen. Durch diese Art der Programmgestaltung werden besonders rechenintensivere Programme sehr viel schneller. Im Spielebereich ist es möglich, Ereignisse durch den AFTER-Befehl erst mit einer gewissen Verzögerung auftreten zu lassen, während man an anderer Stelle bereits den Bildschirmaufbau für das nächste Bild vornimmt. Die normalerweise durch eine Zeitschleife nach dem Schema

```
FOR I=1 TO 1000:NEXT I
```

verbrauchte Zeit kann somit sinnvoll für weitere Aufgaben benutzt werden. Ein typisches Anwendungsbeispiel hierfür sind auch Erklärungstexte. Da wir Menschen langsamer lesen als der Computer Text ausgibt, wird normalerweise in solchen Programmen nach der Ausga-

be einer Textseite in eine Zeit-, das heißt Warteschleife gesprungen. Viel sinnvoller ist es, bereits jetzt die Texte für die nächste Erläuterungsseite zusammenzusetzen oder Variable vorzubereiten und nur von Zeit zu Zeit in eine Tastaturabfrageroutine zu springen, die dann prüft, ob der Benutzer bereits fertig gelesen hat. Die Echtzeitbefehle haben ein breites Anwendungsfeld bei der Koordination von Programmen und Programmteilen. Auch hier gilt wie für den ganzen Rest des Buches: Man wird erst damit vertraut, wenn man damit arbeitet. Spielen Sie einmal ein wenig mit diesen Befehlen, und Sie werden sehen, was für fantastische Möglichkeiten sich Ihnen bieten.

6

Anhang

6 Anhang

6.1 Farben beim CPC 464

6.1 Tabelle der Farben beim CPC 464

Code	Farbe
0	Schwarz
1	Dunkelblau
2	Mittelblau
3	Dunkelrot
4	Violett
5	Hellblau
6	Hellrot
7	Purpur
8	Hellviolett
9	Grasgrün
10	Cyan
11	Himmelblau
12	Heugelb
13	Weiß/Hellgrau
14	Pastellblau
15	Orange
16	Rosa
17	Oastellmagenta
18	Hellgrün
19	Seegrün
20	Hellcyan
21	Mittelgrün
22	Pastellgrün
23	Pastellcyan (weiß)
24	Hellgelb
25	Sehr helles Gelb
26	Volles Weiß
27	Mittelgrau
28	7
29	25
30	1
31	19

Grautöne
0 = Schwarz
27 = Mittelbraun
13 = Weiß/Hellgrau
26 = Volles Weiß

Rottöne
3 = Dunkelrot
6 = Hellrot
15 = Orange
16 = Rosa
7 = Purpur

Blautöne
1 = Dunkelblau
2 = Mittelblau
5 = Hellblau
11 = Himmelblau
14 = Pastellblau

Gelbtöne
12 = Heugelb dunkel
24 = Volles Gelb
25 = Pastellgelb

Grüntöne
9 = Grasgrün
18 = Helleres Grün
21 = Volles Grün
22 = Pastellgrün
19 = Seegrün

Cyan	violett
10 = Cyan	4 = Dunkelviolett
20 = Hellcyan	8 = Hellviolett
23 = Pastellcyan	17 = Pastellviolett

6.2 Nützliche Routinen des Betriebssystems

Die nachfolgenden Routinen werden alle mit dem CALL-Befehl und nachstehender Einsprungadresse aufgerufen. Dabei ist auf das vorangestellte "&" zu achten.

CALL &BB8A schaltet den Cursor ein. Anwendungszweck: zum Beispiel Blinken des Cursors in einer INKEY$-Abfrage in Zusammenhang mit &BB8D.

CALL &BB8D schaltet den Cursor wieder aus. Gegenteil von &BB8A.

CALL &BB9C TEXT-INVERSE-Routine. Die aktuellen Farben von PEN und PAPER werden vertauscht. Nochmaliges Ausführen der Routine führt wieder in den Ausgangszustand.

CALL &BD19 SOFT-SCROLL-Routine. Wartet auf nächsten Bildschirmrücklauf. Damit wird lästiges Flimmern des Textes vermieden.

6.3 Das Umschreiben von C64-Programmen

Das Problem jedes neuen Computers ist der Mangel an Software. Mit diesem Kapitel wollen wir versuchen, hier ein bißchen Abhilfe zu schaffen. Wir werden uns dabei auf den C64 von Commodore konzentrieren, da dieser als meistverkaufter Homecomputer der Welt auch die größte Software-Palette zur Verfügung hat. So sind Programme für den Commodore in fast jeder Fachzeitschrift zu finden.

Zur besseren Übersicht wollen wir vorab einige Informationen geben. Aufgrund des sehr großen Wortschatzes des CPC-BASICs können mit ihm fast alle auf dem Markt befindlichen Dialekte nachgebildet werden. Deshalb ist das Umschreiben von Commodore Programmen auf den CPC 464 problemlos, solange es sich nur um BASIC-Programme handelt. Die Schwierigkeiten beim Umschreiben von Programmen zwischen Computern fangen immer dort an, wo die Hardware, das heißt die festen Teile des Rechners, den programmtechnischen Möglichkeiten Grenzen setzen. Da die Hardware von Computer zu Computer teilweise sehr unterschiedlich ist, führt dies meist schon in den Bereichen Datenspeicherung, Bildschirmansteuerung und -aufbau und bei der Tastatur-Abfrage zu größerem Kopfzerbrechen. Der Transfer von Systemprogrammen ist meist unmöglich, wenn man von sehr artverwandten Computern derselben Herstellerfirma einmal absieht. Systemprogramme arbeiten mit speziellen Routinen des Betriebssystems oder produzieren durch Verändern von Speicherbereichen, Zeichengeneratoren etc. spezielle Effekte, die allerdings nur auf diesem Rechnermodell laufen.

Diese Abstufung in der Übertragbarkeit von Programmen ist auch beim Umschreiben von C64-Programmen auf den CPC gegeben. Aufgrund der verschiedenartigen CPUs ist es unmöglich, Maschinenroutinen von einem System auf ein anderes zu transferieren. Der Standard-BASIC-Befehlssatz ist bei beiden Rechnern identisch und auch viele zusätzliche Funktionen, die der C64 anderen Rechnern voraus hat, sind beim CPC implementiert. So können alle STRING-Operationen, mit Ausnahme des CHR$-Befehls - näheres dazu weiter unten - übernommen werden. Auch die Möglichkeiten bei der Arbeit mit numerischen Variablen, die der 64er bietet, sind beim CPC voll verfügbar. Damit kristallisieren sich die ersten drei Bereiche heraus, in denen ein relativ einfaches Umschreiben von Programmen möglich ist. Es sind dies fast alle mathematischen Programme, eine Vielzahl der in BASIC geschriebenen Datenverwal-

tungs- und Kartei-Programme und zusätzlich einige im BASIC geschriebene Textverarbeitungsprogrammen. Probleme treten hier hauptsächlich in den Bereichen Datenspeicherung und Bildschirmsteuerung auf. Wir werden uns mit diesen Schwierigkeiten jetzt nacheinander beschäftigen.

Zunächst zum Bereich der Datenspeicherung. Bei der Datenspeicherung ist zunächst zu beachten, daß der C64 bereits ohne Änderung des Betriebssystems eine Floppy, das heißt eine Diskettenstation, ansteuern kann. Es findet also nicht, wie beim Schneider, ein Wechsel des Betriebssystems durch das Einstecken der Floppy statt. Die Möglichkeiten der Diskettenoperationen sind daher, wenn auch mangelhaft von BASIC aus, so doch bereits voll verfügbar. Bei der Ein- und Ausgabe wurde deshalb ein etwas anderer Weg, als beim CPC, gegangen. Wichtig für das Ansprechen verschiedener Geräte ist zuerst einmal die Gerätenummer. Dabei bedeuten:

```
0 = die Tastatur
1 = Kassettenrecorder 1
2 = Kassettenrecorder 2
3 = Bildschirm
4 = Drucker
8 = Floppy
```

Das Ansprechen der verschiedenen Teile des Computers ist nicht wie beim CPC durch ein einfaches

```
PRINT# <Gerätenummer>
```

möglich. Beim C64 muß zuvor ein FILE für das anzusprechende Gerät eröffnet worden sein. Dies geschieht mit dem OPEN-Befehl. Der OPEN-Befehl hat drei Parameter. An erster Stelle steht die Nummer des zu eröffnenden FILEs. An zweiter Stelle die Gerätenummer, auf die dieses FILE arbeiten soll. Die dritte Zahl beinhaltet schließlich die sogenannte Sekundäradresse. Diese ist für das Umschreiben von Programmen nur in manchen Fällen wichtig. Sie dient dazu, Fehlerkanäle zu öffnen oder Direktkommandos an die einzelnen Geräte zu senden. Daneben kann sie dazu dienen, verschiedene FILES voneinander zu trennen. Der C64 verfügt somit über eine viel größere Zahl von Datenkanälen, als dies beim CPC der Fall ist. Wollen wir also einen Text auf dem Drucker beim C64 ausge-

ben, so müssen wir zuerst mit

```
OPEN 1,4,1
```

ein FILE auf den Drucker eröffnen, auf dem wir dann mit

```
PRINT#4,"ABCDEF"
```

den Text ABCDEF ausgeben können. Dasselbe gilt natürlich auch für numerische Variable oder Zeichenketten. Beim CPC wäre diese Befehlskette durch ein einfacheres

```
PRINT#8,"ABCDEF"
```

möglich. Eröffnen wir mit

```
OPEN 2,8,1,"<Dateiname>"
```

eine Datei auf die Floppy, oder mit

```
OPEN 3,2,1,"<Dateiname>"
```

ein FILE auf den Kassettenrecorder, erfolgt das Ansprechen immer mittels einfachem PRINT#3, oder PRINT#2.

Die Unterscheidung für das Eröffnen einer Datei zum Lesen oder zum Schreiben, wie das beim CPC mit den Befehlen

```
OPENIN  und  OPENOUT
```

der Fall ist. Dies ist beim C64 zwar möglich, aber nicht unbedingt notwendig. Beim Umschreiben von Programmen müssen wir zunächst nach dem zugehörigen OPEN-Befehl suchen und aus diesem die Gerätenummer entnehmen, bevor wir unseren PRINT-Befehl entsprechend einsetzen können. Zum Abschluß des Transfers noch einmal ein Beispiel einer Datenabspeicherungs-Routine für das Feld A, welches zehn Elemente enthalten soll. In beiden Fällen wurde der Kassettenrecorder als Abspeichermedium gewählt.

```
10 OPEN 1,2,1,"Datei"        10 OPENOUT"Datei"
 .                            .
 .                            .
 .                            .
50 FOR I=1 TO 10             50 FOR I=1 TO 10
60 PRINT#1,a(I)              60 PRINT#9,a(I)
70 NEXT I                    70 NEXT I
80 CLOSE 1                   80 CLOSEOUT
```

Mit diesen Vorkenntnissen dürften sich bei Programmen auf Kassette keinerlei Probleme beim Umschreiben ergeben. Etwas schwieriger wird es bei der Floppy. Diese ist nicht gezwungen, Daten hintereinander abzuspeichern, wie das beim Kassettenrekorder nicht anders möglich ist. Bei der Floppy existieren beim C64 noch drei weitere Abspeicherarten, die, falls sie in einem Programm auftauchen sollten, nicht übertragbar sind. Dies liegt darin, daß bei diesen Abspeicherungsarten an mehreren Punkten der Floppy gleichzeitig gearbeitet wird, was mit einem gradlinig laufenden Kassettenband natürlich nicht möglich ist. Glücklicherweise sind diese Speicherarten nicht besonders verbreitet und tauchen meist nur in professionellen Programmen auf, die wegen der Benutzung von Maschinen-Routinen und ihrer Größe sowieso jegliche Lust am Umschreiben nehmen. In Programmen, die die Floppy benutzen, müssen wir

```
PRINT#9
```

für das Ansteuern des Kassettenrecorders benutzen. In manchen Programmen, die mit der Floppy arbeiten, wird darüberhinaus manchmal ein Fehlerkanal eröffnet. Dies geschieht, indem ein FILE mit der Sekundäradresse fünfzehn auf die Floppy eröffnet wird, zum Beispiel

```
OPEN 2,8,15
```

und über dieses FILE mittels

```
INPUT# 2,ER,EM$,ES,ET
```

die Fehlercodes eingelesen werden. Eine solche Zeile ist zu streichen. Die Bildschirmansteuerung beim Commodore 64 kann man in drei Bereiche unterteilen:

- die Farbgebung
- den Zeichensatz
- die Ansteuerung des Bildschirms

Der Commodore 64 verfügt nicht über WINDOWS und besitzt auch nicht die Möglichkeit mittels direkter Adressierung über den LOCATE-Befehl eine Ausgabe auf den Bildschirm an eine bestimmte Stelle zu dirigieren. Beim Bildaufbau ist man auf die Cursorsteuerzeichen angewiesen. Sie werden in einem Listing als REVERS-gedruckte Zeichen ausgegeben. Diese Zeichen sind auch auf dem CPC verfügbar. Wir benutzen hierfür die Systemsteuerzeichen, auf die wir in dem Kapitel über die Grafik bereits näher eingegangen sind. Hier noch einmal kurz die wichtigsten Bedeutungen:

CTRL I = Cursor rechts
CTRL H = Cursor links
CTRL J = Cursor nach unten
CTRL K = Cursor nach oben

Auch das Löschen des Bildschirms und das Bewegen des Cursors in die linke obere Bildschirmecke sind wie beim C64 als Systemsteuerzeichen verfügbar. Die entsprechenden Zeichen erreichen wir mit

CTRL-L für das Löschen des Bildschirms und

CTRL-^ für HOME

HOME bringt den Cursor in die linke obere Bildschirmecke. Oft wird beim C64 der Direktzugriff auf den Speicher benutzt, um den Bildschirm zu verändern. Dazu verwendet man das POKE-Kommando. In den Speicherstellen 1024 bis 2024 werden die Codes für das entsprechende Zeichen "gePOKEt". Da eins der Bildschirmcode für A ist, würde ein POKE 1024,1 das Erscheinen eines A in der linken oberen Bildschirmecke, das heißt am Bildschirmanfang, bewirken. Geben wir POKE 1025,2 ein, so erhalten wir rechts davon ein B. Der Textspeicher des C64 wird fortlaufend beschrieben. Das heißt: nach dem vierzigsten Zeichen wird in der nächsten Zeile ausgegeben. Ein POKE 1064,3 würde also die Ausgabe eines C am Beginn der zweiten Zeile veranlassen. Diese Art der Bildschirmschirmansteue-

rung wird beim C64 meist in Situationen verwandt, wo nur wenige Zeichen an einer schwer zugänglichen Stelle des Bildschirms ausgegeben werden müssen. Schwer zugänglich heißt dabei, daß es einer ganzen Armada von Cursorsteuerzeichen bedarf, um diesen Punkt zu erreichen (zum Beispiel im Bildschirmmittelpunkt).

Beim Schneider-Computer können wir diese Art der Bildschirmansteuerung viel einfacher mit dem LOCATE-Kommando erreichen. Die Koordinaten für den LOCATE-Befehl erhalten wir durch einfache Umformung aus der angegebenen Speicherstelle. Zunächst ziehen wir 1024 ab und teilen das Ergebnis durch vierzig. Dies liefert uns die Zeilenzahl, die zweite Zahl in unserem LOCATE-Kommando. Der Rest unserer Division gibt uns dann die X-Koordinate an. Ein Beispiel mag dies verdeutlichen. Das Poken eines A in die Bildschirmmitte beim C64 mit

```
POKE 1524,1
```

würde wie folgt umgerechnet:

```
1524-1024 = 500

500 : 40 = 12 Rest 20

X-Koordinate = 20

Y-Koordinate = 12
```

CPC-Befehl:

```
LOCATE 20,12:PRINT "A"
```

Auch die Arbeit mit der hochauflösenden Grafik ist beim C64 nur durch direktes "POKEn" der Speicherzellen und damit der einzelnen Bildpunkte möglich. Programme, die dies nutzen, sind generell nicht auf den CPC 464 umschreibbar. Der CPC verfügt über eine vollkommen andere Art der Farb- und Bildpunktspeicherung als der C64 und außerdem wenden diese Programme meist aus Geschwindigkeitsgründen Maschinensubroutinen an, die Punkte zeichnen, Linien ziehen oder Kreise auf den Bildschirm bringen. Aufgrund der unterschiedlichen CPUs ist ein Transfer hier nicht möglich. Die einzige Möglichkeit besteht darin, die entsprechenden Programme logisch aufzubereiten. Wir müssen feststellen, wo eine Linie ge-

zogen wird und von welchen Eckkoordinaten aus. Ist diese Arbeit einmal getan, ist es dann ein Leichtes, dieselben Effekte auch mit den hochauflösenden Grafikbefehlen der CPC-Grafik zu realisieren. Eine Zeitersparnis, wie sie ja beim Umschreiben von Programmen eigentlich Sinn hat, kann man damit höchstwahrscheinlich nicht mehr erzielen.

Kommen wir nun zur Farbwahl. Auch hier sind die Definitionsmethoden von CPC und C64 unterschiedlich; sie können allerdings relativ problemlos ineinander übergeführt werden. Der C64 legt die Hintergrund- und die Rahmenfarbe direkt durch Eingabe des entsprechenden Farbcodes fest. Dies geschieht wieder mit POKE, und zwar wird mit

```
POKE 53280,X
```

die Rahmenfarbe festgelegt. Mit

```
POKE 53281,X
```

legen wir die Hintergrundfarbe fest. X ist der Farbcode. Der nachstehenden Tabelle können Sie entnehmen, wie die verschiedenen Zahlenwerte den Farben zugeordnet sind.

Tabelle 6.2 Farbcodes für den C64

0 schwarz	8 orange
1 weiß	9 braun
2 rot	10 hellrot
3 türkis	11 grau 1
4 violett	12 grau 2
5 grün	13 hellgrün
6 blau	14 hellblau
7 gelb	15 grau 3

Mit POKE 53281,2 setzen wir die Hintergrundfarbe auf rot. POKE 53280,6 legt die Rahmenfarbe auf blau fest. Die Festlegung der Rahmenfarbe können wir relativ einfach beim CPC mit dem BORDER-Kommando erreichen, das entsprechende Kommando wäre hier

```
BORDER 2
```

Für die Festlegung der Hintergrundfarbe müssen wir uns erst mit INK ein neues Farbregister definieren. Dazu benutzen wir die Befehle INK und PAPER:

```
INK 1,6:PAPER 1
```

Etwas schwieriger wird der Transfer der Schriftfarbe. Auch die Schriftfarbe wird beim C64 mittels spezieller Steuerzeichen festgelegt. Der Commodore verfügt dabei über sechzehn verschiedene Steuerzeichen, für jede Farbe eines. Damit ist es mit dem C64 möglich, sechzehn verschiedene Farben auch im Vierzig-Zeichen-Modus auf dem Bildschirm zu erhalten, während wir beim CPC 464 auf vier Farben inklusive der Hintergrundfarbe beschränkt sind. Bei Commodoreprogrammen, die mit sehr vielen Farbumschaltungen arbeiten, müssen wir deshalb gegebenenfalls die Zahl der benutzten Farben auf vier Stück reduzieren. Die Farbumschaltung nehmen wir einfach mit dem PEN-Befehl vor. Begann der Text auf dem Commodore zum Beispiel mit der Farbe rot und wechselt er nach einiger Zeit auf grün, definieren wir mit INK ein rotes und ein grünes Farbregister und greifen dann mit

```
PEN 1 beziehungsweise  PEN 2
```

auf diese beiden Farbregister zu. Zum besseren Verständnis dieser Technik und der damit gegebenen Möglichkeiten sollten Sie gegebenenfalls noch einmal das Kapitel über die Farbgebung zu Beginn der Grafik durchlesen. Neben den Farbsteuerzeichen besteht beim Commodore jedoch immer noch die Möglichkeit, zwischen Hinter- und Vordergrundfarbe umzuschalten, das heißt, ein Zeichen normal oder REVERS darzustellen. Glücklicherweise findet sich auch dieses Systemzeichen im Befehlssatz des CPC 464. Während der Commodore über zwei verschiedene Zeichen für REVERS EIN und REVERS AUS verfügt, brauchen wir uns beim Schneider mit derartigen Problemen nicht zu belasten.

```
CTRL-X
```

ist das entsprechende Steuerzeichen. Wird es zum erstenmal durchlaufen, schaltet es den REVERS-Modus ein. Wird es nocheinmal eingeben, wird der Modus wieder ausgeschaltet. Die Ausgabe eines REVERS-gedruckten Textes, wobei der voran- und der nachstehende Text normal ausgegeben werden, sieht bei beiden Computern folgendermaßen aus:

C64	CPC 464
PRINT "<RVS on>ABCDEF<RVS off>"	PRINT"<CTRL-X>ABCDE><CTRL-X>"

Mit diesen Informationen über die Farbe ist es möglich, fast alle Farbbefehle des Commodore entsprechend umzuschreiben. Widmen wir uns nun dem heikelsten Thema beim Transfer von Programmen vom 64er auf den CPC im Grafikbereich, dem Zeichensatz:

Der Commodore und der 464er haben einen in vielen Bereichen unterschiedlichen Zeichensatz. Beide verfügen natürlich über die entsprechenden Zeichen zur Darstellung von Zahlen und Buchstaben. Die Unterschiede ergeben sich im Bereich der Blockgrafik. Hier gibt es beim Commodore einige Zeichen, die im Zeichensatz des CPC nicht verfügbar sind. Dies ist jedoch kein Manko, da wir ja fast jedes beliebiges Zeichen mit dem SYMBOL-Kommando definieren können. Wir brauchen uns also bloß eine Kopie des Zeichensatzes des Commodore zu besorgen, um die nicht vorhandenen Zeichen mit dem SYMBOL-Kommando definieren zu können. Diese Liste finden wir zum Beispiel im Handbuch zum C64. Das Problem beim Transfer der Zeichensätze liegt nicht so sehr darin, daß bestimmmte Zeichen nicht übereinstimmen, sondern darin, daß bestimmte Zeichen beim Commodore nicht immer dasselbe bedeuten.

Die 1, die wir weiter oben in die linke Bildschirmecke "gePOKEt" hatten, kann nämlich je nach eingeschaltetem Grafikmodus ein großes oder ein kleines a auf den Bildschirm bringen. Das hängt davon ab, auf welchen Zeichensatz der Character-Generator zurückgreift, um den entsprechenden Code zu interpretieren. Dies wird durch "POKEn" in die Speicherstelle 53272 festgelegt. Ist der Inhalt dieser Zelle gleich 21, greift der Commodore auf den ersten Zeichensatz zurück. Dieser enthält die Großbuchstaben und die Grafiksymbole. Wurde dagegen der Wert dieser Zelle auf 23 gesetzt, so verfügt der Commodore jetzt über Groß/Kleinschreibung, wobei allerdings bestimmte Grafikzeichen nicht mehr verfügbar sind. Wir müssen uns also vor der Übertragung von Zeichen vom C64 auf den CPC vergewissern, auf welchen Zeichensatz wir zurückgreifen. So kann es sein, daß die scheinbar wirre Folge von Buchstaben im anderen MODE, das heißt bei Verwendung der Grafikzeichen, eine durchaus sinnvolle Blockgrafik auf den Bildschirm bringt. Hier ist also Vorsicht geboten.

Noch ein anderer Punkt macht uns in diesem Zusammenhang Schwierigkeiten: Der C64 hat fast alle Grafikzeichen auf der Tastatur

verfügbar. Sie liegen in der SHIFT- beziehungsweise in der CONTROL-Ebene. Wollen wir dies auch mit dem CPC 464 erreichen, müssen wir zuvor die Tastatur umdefinieren. Wie dies möglich ist, und welche Einschränkungen wir dabei zu beachten haben, haben wir bereits im fünften Kapitel bei der Beschäftigung mit der Tastatur gezeigt. Der einfachste Weg dürfte hier sein, mit dem SYMBOL-Kommando neue Zeichen mit relativ hohen Speichernummern - das heißt größer 200 - zu definieren, und diese dann mit KEY DEF auf die Tastatur zu legen.

Diese kleine Problemzusammenstellung der Schwierigkeiten, die beim Umschreiben von Commodoreprogrammen auf den 464er auftauchen können, erhebt natürlich keinen Anspruch auf Vollständigkeit. Sie sollte eine Übersicht über die am häufigsten vorkommenden Probleme und ihrer Lösung bieten. Das Eingehen auf jedes einzelne Spezialproblem würde wahrscheinlich ein neues Buch erfordern. Abschließend zu dieser Thematik sei nur noch das folgende gesagt:

Der C64 ist ein typischer POKE-Rechner, das heißt, ohne PEEK und POKE läßt sich der Rechner nur zu gut 25% seiner Möglichkeiten nutzen. Der CPC ist das genaue Gegenteil davon. Hier ist fast alles von BASIC aus mittels teilweise hochkomplexer Befehle möglich. Die Verwendung des POKE-Befehls beim CPC 464 ist deswegen nur in seltenen Ausnahmefällen überhaupt sinnvoll. Als generelle Regelung kann man daher festhalten:

Jedes POKE, das in einem Commodoreprogramm vorkommt, muß entweder umgeändert oder weggelassen werden.

Stichwortverzeichnis

Aus dem Markt & Technik-Buchverlag

W. Pest: **Computerchinesisch für Einsteiger**
Juli 1984, 107 Seiten
Ein praxisnahes Lexikon, das Personal Computer-Benutzern und solchen, die es werden wollen, das Lesen von Fachzeitschriften, Büchern, Bedienungsanleitungen und Datenblättern erleichtert · über 1000 häufig benötigte Fachbegriffe klar und verständlich erläutert · mit zahlreichen Abbildungen.
Best.-Nr. MT 690, DM 28,— (Sfr. 25,90/öS 218,40)

Personal Computer Lexikon
1982, 136 Seiten
Über 1000 Suchbegriffe aus Hard- und Software · deutsch/englisch · ausführlicher Artikel zu jedem Suchbegriff · englisch/deutsch Register im Anhang · der ideale Einstieg ins Homecomputing · das unentbehrliche Nachschlagewerk für den Profi.
Best.-Nr. MT 390, DM 19,80 (Sfr. 18,50/öS 154,40)

M.J.Winter: **Lehrspielzeug Computer: Atari**
Juli 1984, 139 Seiten
Das neue Computer-Kinderbuch für den Atari 400, 800 und 1200 · Spielprogramme und grafische Darstellungen für Kinder ab 8 Jahren · viele Rechenaufgaben für den kleinen Einstein · so macht Lernen Freude!
Best.-Nr. MT 696, DM 24,80 (Sfr. 23,—/öS 193,40)

H. Glicksman/K. Simon: **Spiele für den Atari**
September 1984, 216 Seiten
Eine unterhaltsame Einweisung in die Atari-BASIC-Programmierung anhand von bereits bewährten sowie raffinierten neuen Computerspielen · Wie man ein Programm strukturiert · Einsatz von Unterprogrammen · Tabellenverarbeitung · Bewegte Grafiken · Testen von Programmen · Noch nie hat Home-Computing so viel Spaß gemacht!
Best.-Nr. MT 678, DM 32,— (Sfr. 29,50/öS 249,60)

H. L. Schneider/R. Bichler: **Das Atari-Buch, Bd. 1**
1984, 158 Seiten
Die grundlegenden Programmiermöglichkeiten für Ihren Atari · mit einem Spiel zum Eingewöhnen · Erstellung von Text und Grafik · Player Missiles · Basic-Besonderheiten · ausführliche Assemblerlistings im Anhang.
Best.-Nr. MT 703, DM 32,— (Sfr. 29,50/öS 249,60)

T. Bridge: **Atari-Abenteuerspiele**
1984, 148 Seiten
Alles über die Anfänge der Abenteuerspiele · Textabenteuer mit vielen Rätseln · Schatzsuche · Kampf mit Monstern · Das Auge des Sternenkriegers · mit hilfreichen Anregungen zum Schreiben Ihrer eigenen Spieleprogramme.
Best.-Nr. MT 727, DM 29,80 (Sfr. 27,50/öS 232,40)

J. White
Strategische Computerspiele für Ihren Atari
1984, 148 Seiten
Aufbau eines Spielfeldes · der Bewegungsablauf · Mustereröffnungen · das Endspiel · Dame, Schach, Warp Trog als Beispiele strategischer Spiele · Anleitung zur systematischen Fehlersuche · Grundkenntnisse in Atari-Basic erforderlich.
Best.-Nr. MT 681, DM 32,— (Sfr. 29,50/öS 249,60)

H. Kohl/T. Kahn et al.: **Spiel und Spaß mit dem Atari**
1984, 338 Seiten
Einfache Programme in Basic · wie man ein Spiel entwickelt · Lernstoff trainieren · Zahlen und Logik · Grafik · Farben · Töne und Musik · den Atari-Computer spielend erforschen.
Best.-Nr. MT 672, DM 42,— (Sfr. 38,60/öS 327,60)

H.L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 1
1984, 270 Seiten
Der Commodore 64 und seine Handhabung · Einführung in die Grafik · Balkendiagramme · Einführung in die Spritetechnik · Basic-Erweiterungen in Assembler · Ein Leitfaden für Erstanwender.
Best.-Nr. MT 591 (Buch)
DM 48,— (Sfr. 44,20/öS 374,40)
Best.-Nr. MT 592 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H.L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 2
1984, 181 Seiten
Spiele nicht nur zum Abtippen · Programmlisting · Programmbeschreibung · Variablenübersicht · Programme nach Anleitung frei ergänzbar · das ideale Buch, um Programmieren spielend zu lernen.
Best.-Nr. MT 593 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 594 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 3
1984, 206 Seiten
Alles über Sprites · Wissenswertes über Multi-Color-Grafik · Assembler/Disassembler · jede Menge Basic-Erweiterungen · Umgang mit dem Soundgenerator · ein Leitfaden für Fortgeschrittene.
Best.-Nr. MT 595 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 596 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 4
1984, 261 Seiten
Einführung in Maschinenprogrammierung · Verknüpfung von Maschinenprogrammen mit Basic-Programmen · alles über Assembler/Disassembler · der Leitfaden für Systemprogrammierer.
Best.-Nr. MT 597 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 598 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 5
Juli 1984, 322 Seiten
Ein Leitfaden durch Simon's Basic · ausführliche Besprechung aller Befehle · viele erklärende Beispiele · mit kommentierter Assembler-Listing · das richtige Nachschlagewerk für den geübten Commodore 64-Benutzer.
Best.-Nr. MT 599 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 600 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar, Telefon: 089/4613-220

Aus dem Markt & Technik-Buchverlag

H.L. Schneider/W. Eberl
Das Commodore 64-Buch, Bd. 6: Spiele
1984, 190 Seiten
Programmieren auf dem Commodore 64 spielend gelernt · leicht verständliche Spielanleitungen · Programmlisting mit anschließender Programmbeschreibung · Variablenübersicht · Tips zum Ändern und Ergänzen des Programms.
Best.-Nr. MT 619 (Buch)
DM 38,— (Sfr. 35,—/öS 296,40)
Best.-Nr. MT 620 (Beispiele auf Diskette)
DM 58,— (Sfr. 58,—/öS 522,—)

H. L. Schneider: **Das Commodore 64-Buch, Bd. 7**
August 1984, 110 Seiten
Der Commodore 64 als Klaviatur · Noten schreiben mit hochauflösender Grafik · relative Dateien am Beispiel einer kleinen Adreßverwaltung · Benutzung des Joysticks und der Paddles · Grafikspeicher unter Kernal · Interrupt-Manager · eigene Zeichen definieren · Bildschirmrollen · für Profis.
Best.-Nr. MT 731, DM 38,— (Sfr. 35,—/öS 296,40)

Computerspiele & Wissenswertes — Commodore 64
1984, 156 Seiten
Eine Sammlung von interessanten und nützlichen Maschinenprogrammen · schnelle binäre Artithmetik · Basic-Erweiterungen · mit unterstützendem Assembler-Listing · für den fortgeschrittenen Programmierer.
Best.-Nr. MT 601 (Buch)
DM 29,80 (Sfr. 27,50/öS 232,40)
Best.-Nr. MT 602 (Beispiele auf Diskette)
DM 38,— (Sfr. 38,—/öS 342,—)

F. Ende
Das große Spielebuch — Commodore 64
1984, 141 Seiten
46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.
Best.-Nr. MT 603 (Buch)
DM 29,80 (Sfr. 27,50/öS 232,40)
Best.-Nr. MT 604 (Beispiele auf Diskette)
DM 38,— (Sfr. 38,—/öS 342,—)

Dr. P. Albrecht: **Commodore 64 — Multiplan**
1984, 230 Seiten
Multiplan jetzt auch für den Commodore 64 · der volle Leistungsumfang der 16-Bit-Version · Einführung in die Arbeitsweise von Tabellenkalkulationsprogrammen · praxisnahe Beispiele · Beschreibung alle Befehle und Funktionen · nicht nur für Anfänger.
Best.-Nr. MT 655, DM 48,— (Sfr. 44,20/öS 374,40)

E. H. Carlson: **Basic mit dem Commodore 64**
1984, 320 Seiten
Ein Basic-Lehrbuch für den jugendlichen Anfänger · übersichtlich gegliederte Lernprogramme · Alles über INPUT-GOTO · Let-Befehle · Editorfunktionen · POKE-Befehle für die Grafik · geeignet auch als Leitfaden für Lehrer und Eltern.
Best.-Nr. MT 657, DM 48,— (Sfr. 44,20/öS 374,40)

R. E. Williams: **CalcResult richtig eingesetzt**
1984, 236 Seiten
Ein Übungsbuch speziell für Anwender des CalcResult-Computerprogramms · zahlreiche Einsatzmöglichkeiten im täglichen Leben · Kreditrückzahlung · Rabattberechnung · Kostendeckung · Inventur · Finanzierung und Ankauf eines Hauses und vieles andere mehr.
Best.-Nr. MT 671, DM 48,— (Sfr. 44,20/öS 374,40)

W. B. Sanders: **Einführungskurs: Commodore 64**
1984, 276 Seiten
Die Programmiersprache Basic · Einsatzgebiete des Commodore 64-Basic: Grafik, Musik, Dateiverwaltung · mit vielen Beispielprogrammen, häufig benötigten Tabellen und nützlichen Tips · für Einsteiger und Fortgeschrittene.
Best.-Nr. MT 685, DM 38,— (Sfr. 35,—/öS 296,40)

J.W. Willis/D. Willis
Commodore 64 — leicht verständlich
1984, 154 Seiten
Informationen für den Computer-Neuling · Installation und Inbetriebnahme · Programmieren in Basic · Grafik und Töne · Auswahl von Hardware und Zubehör · Software für Ihren Computer · die ideale Einführung in das Arbeiten mit Ihrem Commodore 64.
Best.-Nr. MT 700, DM 29,80 (Sfr. 27,50/öS 232,40)

G. Beekman: **Ihr Heimcomputer Commodore 64**
August 1984, 296 Seiten
Alles Wissenswerte im Umgang mit dem Commodore 64 · Planung, Kauf und Inbetriebnahme der Anlage · Einsatz fertig gekaufter oder selbst erstellter Programme · Schwächen und Stärken der altbewährten und neuesten Programmiersprachen · die gängigsten Software-Angebote für jeden Einsteiger ·
Best.-Nr. MT 701, DM 38,— (Sfr. 35,—/öS 296,40)

M. J. Winter
Das Commodore 64-LOGO-Arbeitsbuch
September 1984, 225 Seiten
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
Best.-Nr. MT 720, DM 34,— (Sfr. 31,30/öS 265,20)

J. Mihalik
35 ausgesuchte Spiele für Ihren Commodore 64
September 1984, 141 Seiten
Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore 64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die Ihre Programmierkenntnisse vertiefen wollen!
Best.-Nr. MT 774, DM 24,80 (Sfr. 23,—/öS 193,40)

M.J.Winter: **Lehrspielzeug Computer: C 64/VC-20**
Juli 1984, 139 Seiten
Speziell für Kinder entwickelt führt dieses Buch spielerisch in die Basic-Welt des C 64/VC-20 ein · mit vielen lehrreichen Spielprogrammen und Grafikmöglichkeiten · kleinere Kinder benötigen die Hilfe ihrer sachkundigen Eltern.
Best.-Nr. MT 695, DM 24,80 (Sfr. 23,—/öS 193,40)

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar, Telefon: 089/4613-220

Aus dem Markt & Technik-Buchverlag

S. Urute
Grafik & Musik auf dem Commodore 64
Oktober 1984, 336 Seiten
68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.
Best.-Nr. MT 743, DM 38,— (Sfr. 35,—/öS 296,40)

H. L. Schneider
Commodore 64 Listings — Band 1: Spiele
Oktober 1984, 199 Seiten
Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise
Best.-Nr. MT 748, DM 24,80 (Sfr. 23,—/öS 193,40)

E. H. Carlson: **Lerne Basic auf dem VC-20**
August 1984, 320 Seiten
Das neue Basic-Lehrbuch für den Commodore VC-20 · einfach erklärte Basic-Befehle mit Übungen · viele heiße Actionspiele · nützliche Programmiertricks · mit ausführlichem Begriffslexikon · der Renner für junge Computer-Freaks!
Best.-Nr. MT 691, DM 38,— (Sfr. 35,—/öS 296,40)

P. Rädsch:
Programme und Tips für VC-20
1983, 152 Seiten
Nützliche Hilfsprogramme für die Arbeit mit dem VC-20 · kommerzielle Anwendung in der Textverarbeitung, Fakturierung und Lagerverwaltung · Möglichkeiten hochauflösender Grafik über eine Assemblerroutine · unterhaltsame Spielprogramme.
Best.-Nr. MT 513, DM 38,— (Sfr. 35,—/öS 296,40)

M. Hegenbarth/M. Schäfer: **Das VC-20-Buch**
1983, 351 Seiten
Eine Sammlung gut erklärter Programme · viele Spielebeispiele · einfache kommerzielle Anwendungen
Best.-Nr. MT 516 (Buch)
DM 49,— (Sfr. 45,10/öS382,20)
Best.-Nr. MT 581 (Kassette)
DM 19,90 (Sfr. 19,90/öS 179,10)
Best.-Nr. MT 582 (Diskette)
DM 29,90 (Sfr. 29,90/öS 269,10)

N. Hampshire: **Grafik mit dem VC-20**
1984, 202 Seiten
38 vollständige Programme · zahlreiche grafische Darstellungen · alles über hochauflösende Grafik und Multicolor-Modus · praktische Anwendungen und Simulationen von Kunst über Videospiele, Mathematik, Naturwissenschaften bis hin zum kaufmännischen Bereich.
Best.-Nr. MT 644, DM 32,— (Sfr. 29,50/öS 249,60)

R. Zamora/D. Inman et al.: **Basic mit dem VC-20**
1984, 364 Seiten
Eine schrittweise Einführung in das Gebiet von VC-20-Basic · Geräusch- und Musikerzeugung · Drucken von grafischen Schriftzeichen · Erstellen eines lauffähigen VC-20-Programms · Arbeiten mit Zeichenvariablen, einfachen Federvariablen, READ- und DATA-Befehlen · Zeichentricks.
Best.-Nr. MT 649, DM 38,— (Sfr. 35,—/öS 296,40)

D. Laine
Maschinencode-Programme für den ZX Spectrum
1984, 204 Seiten
Nützliche Maschinencode-Programme mit Ihrem ZX Spectrum · Sortierung von Fließkommazahlen · Übernahme von Parametern direkt von einem Basic-Programm · Flußdiagramme · für Profis und solche, die es werden wollen.
Best.-Nr. MT 702, DM 32,— (Sfr. 29,50/öS 249,60)

M. Gavin
Astronomie-Programme für den ZX-Spectrum
September 1984, 268 Seiten
Eine phantastische Reise in die Welt des Kosmos mit Ihrem ZX-Spectrum: Der Julianische Kalender · Die Mondphasen · Eigene Satelliten starten · Kepler's Umlaufbahnen · Die Umlaufbahn Plutos · Interessant nicht nur für Hobby-Astronome.
Best.-Nr. MT 732, DM 32,80 (Sfr. 27,50/öS 232,40)

T. Bridge/R. Carnell: **ZX-Spectrum Abenteuerspiele**
September 1984, 208 Seiten
Die Entstehungsgeschichte der Abenteuerspiele mit repräsentativen Beispielen für jede »Epoche«. Ein Programm speziell für Ihren ZX-Spectrum: »Das Auge des Sternenkriegers«, ein Grafik-Abenteuerspiel, das Sie in Atem hält!
Best.-Nr. MT 712, DM 29,80 (Sfr. 27,50/öS 232,40)

J. Cassidy/P. Katz et al.: **Im Land der Abenteuer**
1984, 146 Seiten
Eine Hilfestellung für zahlreiche Computerspiele · Tod in der Karibik · Transsylvanien · Unternehmen Asteroid · Das geheimnisvolle Haus · Zauberer und Prinzessin · Das goldene Vlies · Zeitzone · Der dunkle Kristall · mit Lösungen.
Best.-Nr. MT 699, DM 29,80 (Sfr. 27,50/öS 232,40)

J. R. Brown: **Basic für Einsteiger**
1984, 239 Seiten
Ein Arbeitsbuch für den absoluten Anfänger · Basic-Anweisungen Schritt für Schritt erklärt und anhand von einfachen Beispielen erläutert · das beliebte Arbeitsmittel für Lehrkräfte und für den interessierten Computerfan.
Best.-Nr. MT 680, DM 32,— (Sfr. 29,50/öS 249,60)

Ch. Langfelder: **Basic ohne Probleme, Bd. 1**
1983, 226 Seiten
Eine Unterweisung in Basic mit CBM-Rechnern (CMB 8032) · Grundlagen des Betriebssystems · Funktionsweise des Interpreters · mathematische Programme · Verarbeitung von Texten und Zeichen · Glossar der wichtigsten Fachbegriffe.
Best.-Nr. MT 480, DM 36,— (Sfr. 33,10/öS 280,80)

Ch. Langfelder: **Basic ohne Probleme, Bd. 2**
1982, 119 Seiten
Für alle CBM 8032-Rechner · ausgewählte Routinen und Programme · drei allgemeine Routinen · fünf kommerziell-technische Anwendungen · zwei Statistikprogramme · zwei Mathematikprogramme · drei Lehr- und Spielprogramme.
Best.-Nr. MT 490, DM 26,— (Sfr. 24,10/öS 202,80)

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar, Telefon: 089/46 13-220

Aus dem Markt & Technik-Buchverlag

H. L. Schneider: **Basic ohne Probleme, Bd. 3**
1983, 256 Seiten
Von der Problemanalyse über Programmentwurf zur Programmiertechnik · Beschreibung allgemeiner, immer wiederkehrender, wichtiger Programmsequenzen · einiges zum Thema Datenverwaltung · mögliche Dateiformen · Zugriffsverfahren auf Dateien, z.B. Binärbäume.
Best.-Nr. MT 500, DM 44,— (Sfr. 40,50/öS 343,20)

H. L. Schneider: **Basic ohne Probleme, Bd. 4**
1983, 428 Seiten
Eine komplette Dateiverwaltung · dateibeschreibende Variablen · Index-sequentielle Schlüsselverwaltung · verkettete Listen · variable Drucklisten · variabler Etikettendruck.
Best.-Nr. MT 514, DM 53,— (Sfr. 48,80/öS 413,40)

D.A. Brain: **Basic-Dialekte im Vergleich**
1984, 105 Seiten
Konvertierung von Apple-, Commodore-und TRS-80-Programmen · Grundlagen der jeweiligen Betriebssysteme · Untersuchung verschiedener Basic-Dialekte · alphabetische Auflistung aller Befehle für die verschiedenen Anpassungsrichtungen.
Best.-Nr. MT 564, DM 32,— (Sfr. 29,50/öS 249,60)

M. Waite/M. Pardee: **Basic-Programmier-Handbuch**
1984, 506 Seiten
Grundlagen · Basic und seine Dialekte geschäftliche- und wissenschaftliche Anwendungen · Spiele · Lernprogramme · alles über Programmsteuerung · Schleifen und Verzweigungen · Amortisationsprogramm · numerische Funktionen · Stringfunktionen · Variationen mit PEEK und POKE · der Zauberwürfel.
Best.-Nr. MT 658, DM 78,— (Sfr. 71,80/öS 608,40)

J.J. Purdum: **Basic-80 und CP/M**
1983, 296 Seiten
Wo anfangen? · Daten innerhalb eines Programms · Schleifenprogrammierung und Sprünge · Zahlen · sequentielle Datenzugriffsverfahren · Sortieren und Suchen · CP/M und Basic-80 · Fehlersuche · Beispiele.
Best.-Nr. MT 525, DM 48,— (Sfr. 44,20/öS 374,40)

K. Knecht: **Microsoft-Basic**
1984, 204 Seiten
Eine Übersicht der Version 5.0 von Microsoft-Basic · umfangreiche Beispiele für CP/M-Systeme und TRS-80 · Programmieren mit Sprüngen und Schleifen · Umgang mit Zeichenketten und Matrizen · die Arbeitsweise des Editors · Aufbau verschiedener Dateitypen.
Best.-Nr. MT 650, DM 48,— (Sfr. 44,20/öS 374,40)

Otmar Feger: **Ein-Chip-Mikrocomputer-Handbuch**
1983, 389 Seiten
Grundlagen · technisch Entwicklung · Trends · Marktübersicht und Eigenschaften der wesentlichen 4-, 8- und 16-Bit-Typen und ihrer Peripherie · Funktion und Arbeitsweise der wichtigsten mitintegrierten Schaltungskomponenten · Phasen und Durchführung eines Mikrocomputerprojektes.
Best.-Nr. MT 517, DM 58,— (Sfr. 53,40/öS 452,40)

K.H. Heß
Basic-Programme für CBM/VC-20-Computer
1984, 150 Seiten
Programmanwendungen für die Serien CBM 2000, 3000, 4000, 8000 und VC-20 · Analyse verschiedenster Aufgabenstellungen · allgemeingültige Lösungswege in CBM-Basic konvertiert · von Programmanpassungen für VC-20 · für Laien und Profis.
Best.-Nr. MT 501, DM 32,— (Sfr. 29,50/öS 249,60)

H. Stein: **Logo — Grafik, Sprache, Mathematik**
1984, 257 Seiten
Eine Einführung in Logo als Lehr- und Lernsprache · Grafikprozeduren · Zeichenkettenmanipulationen · Probleme der Rekursivität · Sprachbildung und Sprachforschung · Grundlagen der Arithmetik · mit umfassendem Glossar.
Best.-Nr. MT 648, DM 42,— (Sfr. 38,60/öS 327,60)

K. Knecht: **Einführung in Forth**
1984, 218 Seiten
Ausführliche Informationen über die MMS Forth-Version der Computersprache Forth · syntaktische Grundlagen · zahlreiche Programmierbeispiele · der richtige Einstieg in das Programmieren mit Forth.
Best.-Nr. MT 635, DM 58,— (Sfr. 53,40/öS 452,40)

J. Purdum: **Einführung in C**
1984, 304 Seiten
Die grundlegende Charakteristik von C · Operatoren, Variablen und Schleifen · Erstellung eigener Funktionen · Ein-und Ausgabeoperationen in C · Anlegen einer Adreßkartei · Einsatzmöglichkeiten in nahezu allen Bereichen · für Einsteiger und Fortgeschrittene.
Best.-Nr. MT 561, DM 69,— (Sfr. 63,50/öS 538,20)

W. Maaß: **Software-Schnellkurs: CP/M**
1984, 85 Seiten
Was man von CP/M unbedingt kennenlernen muß · die wichtigsten Befehle des 8-Bit-Standard-Betriebssystems und ihre Handhabung · die wichtigsten Befehle für den täglichen Umgang.
Best.-Nr. MT 605, DM 37,— (Sfr. 34,—/öS 288,60)

W. Maaß: **Software-Schnellkurs: CP/M-86**
1984, 93 Seiten
Der tägliche Umgang mit dem Betriebssystem · wie man Dateien anlegt, kopiert, sichert, löscht.
Best.-Nr. MT 615, DM 37,— (Sfr. 34,—/öS 288,60)

W. Pest: **Hardware-Auswahl leicht gemacht**
3. völlig überarbeitete und aktualisierte Ausgabe
1984/85, 485 Seiten
Die wichtigsten Daten von über 200 Personal-Computersystemen sowie die wichtigsten Peripheriegeräte · ausführliche Begriffserläuterungen · Checklisten für den Geräteeinkauf · Trendberichte und Bezugsquellen.
Best.-Nr. MT 350, DM 58,— (Sfr. 53,40/öS 452,40)

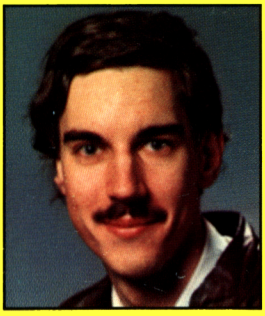
Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher in guten Buchhandlungen, Computershops und Fachabteilungen der Kaufhäuser.

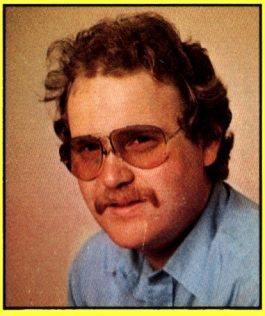
Sollten Sie diese Bücher ausnahmsweise im Handel nicht beziehen können, so bestellen Sie bitte bei:
Markt & Technik Verlag Aktiengesellschaft, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar, Telefon: 089/4613-220

CPC 464
FÜR EIN- UND UMSTEIGER

Dieses Buch ist eine praxisorientierte Spiel- und Arbeitshilfe für den Schneider CPC 464.
In einer Rundreise durch die Bereiche BASIC, Grafik, Sound, Tastaturanwendung und Kassettenrecordereinsatz, werden die meisten Befehle des CPC in kompakter, systematischer Form dargestellt. Schwerpunkte sind dabei die im Standard-BASIC nicht enthaltenen Kommandos und ihre Anwendung in einer Reihe nützlicher Programme zur Textverarbeitung, Datenverwaltung, in der Fehlerbehandlung, bei der grafischen Darstellung und im Musikbereich. Die weitgehend modular aufgebauten Beispielprogramme bilden den Grundstock für eine CPC 464 Programmbibliothek.
Das Buch ist für Anfänger und Fortgeschrittene, die sich die Möglichkeiten des Schneider CPC 464 über das Handbuch hinaus erschließen wollen, gleichermaßen gut geeignet.



CARSTEN STRAUSH geboren am 12.2. 1963, kam über die ersten Taschenrechner und die legendären 8080-Experimentier-Kits zur Computerei. Hier arbeitete er auf vielen der bekannteren Mikrocomputertypen, wie TRS-80, Sinclair, Commodore und gelangte schließlich auch zum CPC 464. Neben Veröffentlichungen und der Softwareentwicklung beschäftigt er sich auch mit dem Software-Marketing für alle gebräuchlichen Homecomputer und PC's.



HARTMUT PICK Diplom-Volkswirt, geboren am 25.4.1957, beschäftigt sich seit geraumer Zeit mit Microcomputer. Er hat bereits auf so bekannte Rechner wie TI99 und Commodore 64 gearbeitet. Beim Schneider CPC 464 hat er sich bisher ausführlich mit der Anwendung des Rechners auf dem wirtschaftlichen Gebiet sowie mit den musikalischen Fähigkeiten des Rechners beschäftigt.

Markt&Technik Verlag Aktiengesellschaft
ISBN 3-89090-090-9

DM 46,—
Sfr. 42,30 / öS 358,60

Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/