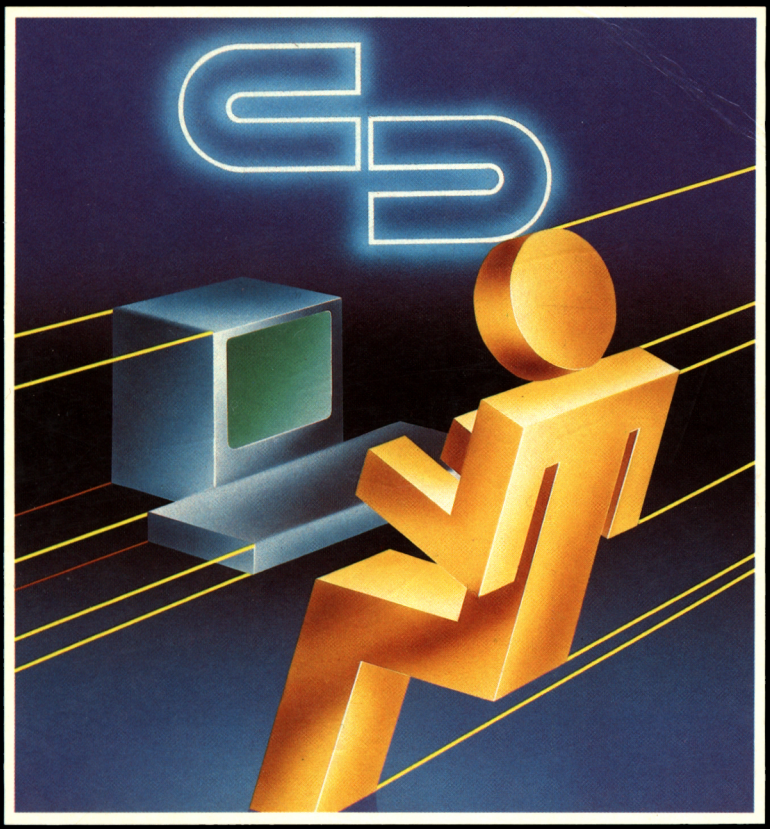
SYBEX

Arbeiten mit dem Schneider CPC



Hans Lorenz Schneider

Arbeiten
mit dem Schneider CPC

Arbeiten mit dem Schneider CPC

Hans Lorenz Schneider

BERKELEY · PARIS · DÜSSELDORF

Umschlagentwurf: Daniel Boucherie/tgr
Satz: Satzzentrum Oldenburg GmbH, Oldenburg
Gesamtherstellung: Druckhaus Neue Stalling GmbH & Co KG, Oldenburg

ISBN 3-88745-603-3
1. Auflage 1985

Inhaltsverzeichnis

Vorwort

Das vorliegende Buch soll Sie - wie der Titel schon sagt - beim Arbeiten mit Ihrem Schneider CPC unterstützen. Deshalb wurde weniger Wert auf die Erklärung und Interna gelegt, sondern vielmehr haben wir versucht, die Möglichkeiten Ihres Computers so darzustellen, daß Sie das Buch als Werkzeug zur Erstellung Ihrer individuellen Programme heranziehen können.

Wo es möglich war, haben wir typische Beispiele gewählt, die Sie nur auf Ihre Erfordernisse abzuändern brauchen (z. B. Diagramme). Alle Themen, die für den Gebrauch Ihres Computers notwendig sind, wurden entsprechend ihrer Wichtigkeit behandelt.

In der Hauptsache wird in diesem Buch auf das „vernünftige" Arbeiten mit dem Computer eingegangen, aber auch ein kleines Spielebeispiel ist vorhanden. Obwohl alle Gegebenheiten möglichst genau formuliert wurden, ist dieses Buch nicht nur für Anfänger geeignet, sondern auch für Fortgeschrittene, aber auch der Profi wird sicherlich den einen oder anderen Hinweis finden.

Besonderer Dank gilt Herrn Reinhard Damm, der mich bei den Themen Zeichengenerator und Sound unterstützte, sowie Herrn Andreas Kißlinger, der für die Themen Sprites, Hardcopy, den größten Teil der Tips und Tricks, die Betriebssystem-Routinen und die BASIC-Erweiterungen verantwortlich zeichnet.

Und nun viel Spaß beim Programmieren.

München, im April 1985 Hans Lorenz Schneider

Einleitung

Wie bereits im Vorwort erwähnt, soll dieses Buch eine Hilfestellung für diejenigen bieten, die Ihren Schneider CPC mit eigenen Programmen betreiben wollen. Aus diesem Grunde sind im ersten Kapitel einige grundlegende Programme vorgestellt, wie sie nicht nur auf dem CPC laufen.

Sobald Daten verarbeitet werden, sind auf jeden Fall Sortier- und Selektierverfahren unerläßlich. Da es für Kalenderdaten keinen eigenen Variablentyp gibt, werden sie meistens als Zeichenreihen dargestellt, erfordern jedoch eine eigene Behandlung. Den Abschluß von Kapitel 1 bilden noch einige wesentliche Bemerkungen über besonders herausragende Eigenschaften Ihres Rechners.

Im zweiten Kapitel beschäftigen wir uns mit dem Zeichensatz, wobei wir auch näher auf den Zeichengenerator eingehen und hierfür einen Zeichensatz-Editor zur Verfügung stellen.

Das dritte Kapitel ist der hochauflösenden Grafik gewidmet, wo wir zu Beginn auch noch das Thema Joystick ansprechen. Neben einer Übersicht über verschiedene Diagramme werden auch Sprites besprochen, obwohl sie von Hause aus beim CPC nicht vorgesehen sind. Den Abschluß von Kapitel 3 bildet ein Hardcopy-Programm, was sicherlich für Diagramme interessant ist, da man nicht immer einen Computer zum Vorzeigen mitnehmen kann.

Der größte Einsatzbereich von Home-Computern (neben Spielen) ist wohl die Dateiverwaltung. Diesem Thema wird in Kapitel 4 Rechnung getragen, wobei eine kleine Dateiverwaltung vollständig entwickelt wird.

Eine speziellere Möglichkeit bietet der CPC mit der Musikprogrammierung. Sofern Sie Interesse an diesem Thema haben, schlagen Sie in Kapitel 5 nach.

Die letzten drei Kapitel sind mehr oder weniger für Fortgeschrittene und Profis gedacht, da hier einige besondere Kniffe und wichtige Teile des Betriebssystems (insbesondere die in diesem Buch verwendeten) vorgestellt werden. Obwohl der CPC ein umfangreiches BASIC hat, sind sicherlich manchmal noch Erweiterungen wünschenswert. Die Vorgehensweise ist relativ unkompliziert und wird im letzten Kapitel besprochen.

Der Anhang bietet neben einem umfangreichen Stichwortverzeichnis noch zwei Übersichten über die BASIC-Befehle (Parameter-Bedeutung/Tokens) und eine Erklärung zu den im Buch verwendeten Variablen-Übersichten.

Kapitel 1

Allgemeine Programmroutinen

Um den Ein- und Umsteigern die Eingewöhnung an den Schneider CPC etwas zu erleichtern, haben wir im ersten Kapitel einige Routinen zusammengefaßt, die noch nichts speziell mit dem CPC zu tun haben. Neben der allgemeinen Verwaltung von Daten (Erfassen, Ändern, Löschen und Ausdrucken) sind in vielen Fällen Unterprogramme zum Sortieren, Selektieren und Mischen von Vorteil. Auch das Thema Kalenderdaten soll in diesem Kapitel erörtert werden, sowie einige andere natürliche Programmsequenzen.

SORTIEREN

Besonders in der Datenverwaltung ist es von Vorteil, wenn manche Daten sortiert vorliegen, z. B. eine nach Namen sortierte Adreßdatei oder eine Schallplattenliste, nach Songs oder Interpreten sortiert. Bei den Sortierverfahren kann man generell zwischen zwei Möglichkeiten wählen. Einerseits kann man die Originalstruktur der Daten sortieren, andererseits ein Hilfsfeld aus Zeigern aufbauen, die auf die Daten zeigen, wobei dann die Zeiger sortiert werden müssen. Im ersten Teil werden wir jeweils die originäre Datenstruktur sortieren und zum Schluß noch das Verfahren Quicksort so abändern, daß die originäre Datenstruktur beibehalten bleibt.

Das Verfahren mit Zeigern ist besonders dort sinnvoll, wo Datensätze auf einem externen Datenträger (Floppy) sortiert werden sollen. Die Ein-/Ausgabeoperationen von/auf die Floppy sind sehr zeitaufwendig, so daß beim Umsortieren der Datensätze die meiste Zeit für Schreib-/Leseoperationen verwendet wird. Da meist nur nach einem Teil der Daten sortiert wird (z. B. Namen bei einer Adreßdatei), genügt es, diese Namen in den Rechner zu holen und dort zu sortieren, wobei jeweils die Nummern der Datensätze mitsortiert werden, um die Daten später sortiert ausdrucken zu können.

Abb. 1.1 veranschaulicht die beiden unterschiedlichen Schemata.

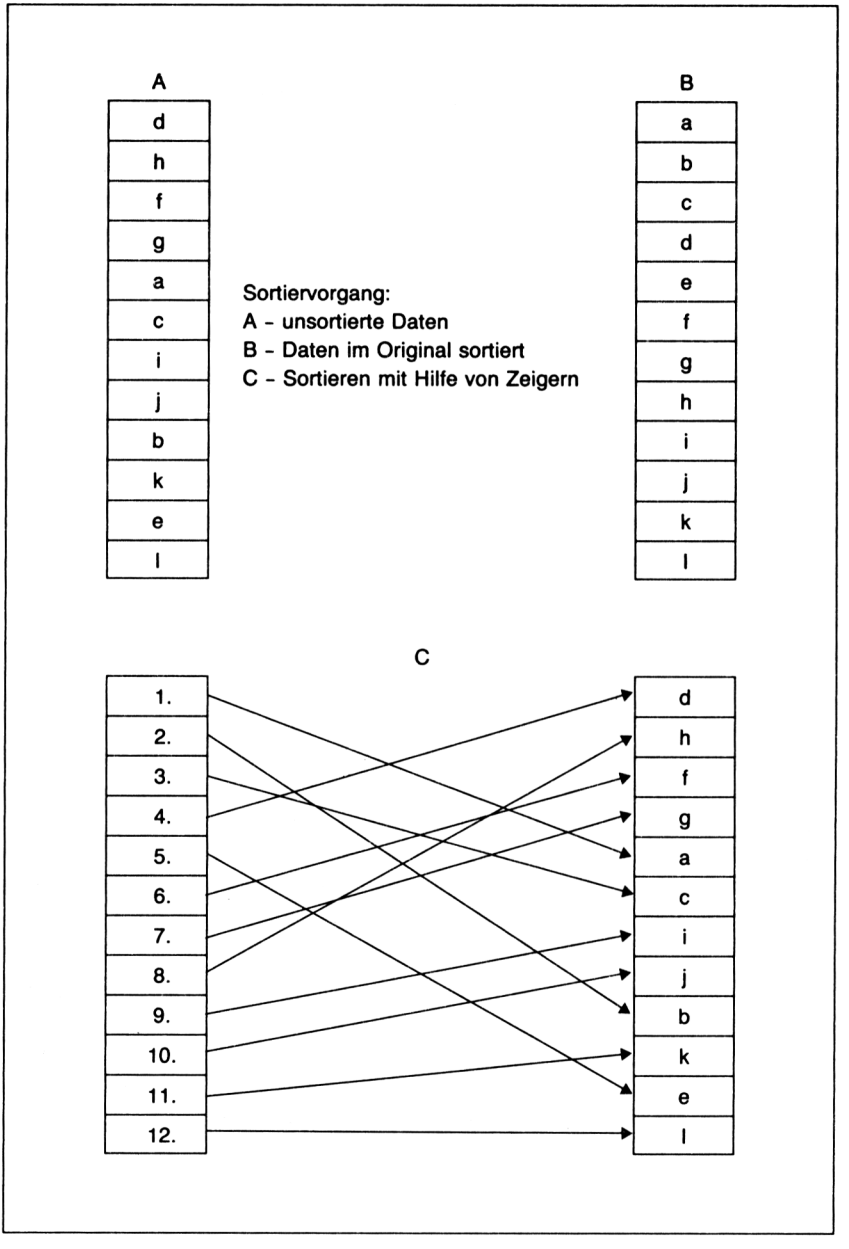


Abb. 1.1: Sortiervorgang

Ein einfaches Sortierverfahren

Ein sehr primitives Sortierverfahren besteht darin, jeweils zwei aufeinanderfolgende Elemente zu vertauschen, wenn das zweite Element kleiner als das vorhergehende ist und man aufsteigend sortieren will. Für sortierte Felder dürfte dies das schnellste Verfahren sein, da nur ein Durchlauf benötigt wird. Bei hochgradig unsortierten Daten sind jedoch erhebliche Rechenzeiten in Kauf zu nehmen.

```
1000 REM ------------------------------
1010 REM ---      RAHMENPROGRAMM      ---
1020 REM ------------------------------
1030 ANZAHL=100
1040 DIM FELD(ANZAHL)
1050 RANDOMIZE 464
1060 FOR I=1 TO ANZAHL
1070    FELD(I)=INT(RND(6)*1000)+1
1080    REM FELD(I)=I
1090    REM PRINT I,FELD(I)
1100 NEXT
1110 ZEITANFANG=TIME
1120 GOSUB 2000
1130 PRINT "SEKUNDEN";(TIME-ZEITANFANG)/
          300
1140 FOR I=1 TO ANZAHL
1150    PRINT I,FELD(I)
1160 NEXT
1170 END
2000 REM ------------------------------
2010 REM ---   EINFACHES SORTIEREN   ---
2020 REM ------------------------------
2030    FERTIG=0
2040    WHILE (FERTIG=0)
2050       FERTIG=1
2060       FOR I=1 TO ANZAHL-1
2070          IF FELD(I+1) < FELD(I)
                THEN HILF = FELD(I) :
                   FELD(I) = FELD(I+1) :
                   FELD(I+1) = HILF :
                   FERTIG=0
2080       NEXT
2090    WEND
2100 RETURN
```

Abb. 1.2: Listing für das Sortierprogramm

Programmbeschreibung

Um das Sortierprogramm testen zu können, haben wir ein kurzes Rahmenprogramm geschrieben, in dem ein zu sortierendes Feld erstellt wird, wobei die Anzahl der Elemente in Zeile 1030 vorgegeben wird. Nach dem Vorbesetzen des Zufallsgenerators in Zeile 1050 werden in der Schleife von Zeile 1060 bis 1100 die Feldelemente per Zufall besetzt. Um ein bereits sortiertes Feld für Testzwecke zu erhalten, ist das REM in Zeile 1080 zu entfernen und in Zeile 1070 einzubringen. Wer sich die unsortierten Daten zunächst ansehen möchte, kann in Zeile 1090 das REM entfernen.

Um die Sortierzeit zu ermitteln, wird in Zeile 1110 die aktuelle Zeit des Computers (seit dem Einschalten) festgehalten, bevor in der nächsten Zeile das Sortierunterprogramm aufgerufen wird. Da im CPC die Zeit in Schritten von 1/300 Sekunden fortgezählt wird, muß in Zeile 1130 noch eine Umrechnung in Sekunden erfolgen. In der folgenden Schleife wird das sortierte Feld ausgegeben.

Da für das vorliegende Sortierverfahren keine feste Vorgabe der Anzahl der Durchläufe gemacht werden kann (Minimum ein Durchlauf; Maximum von Vorsortierung und Anzahl der zu sortierenden Elemente anhängig), definieren wir zunächst eine Variable FERTIG, die auf 1 gesetzt ist, wenn ein Durchlauf ohne Vertauschen erfolgte.

Die Anweisung in Zeile 2040 mit dem zugehörigen WEND in Zeile 2090 stellt eine prinzipielle Endlosschleife dar, die so lange durchlaufen wird, bis das Feld sortiert ist. Aus ablauftechnischen Gründen wird als erstes in der Endlosschleife das Endekriterium auf 1 gesetzt, da es einfacher festzustellen ist, ob eine Vertauschung erfolgte, um dann gegebenenfalls diese Variable wieder zurückzusetzen (letzte Anweisung in Zeile 2070).

Die Schleife von Zeile 2060 bis 2080 durchläuft alle Elemente des zu sortierenden Feldes, wobei nur bis zum vorletzten Element überprüft werden muß. Andernfalls würde in der vorletzten Anweisung in Zeile 2070 ein Indexüberlauf entstehen.

Das Innere der Schleife besteht nur aus der Abfrage, ob das nächste Element im Feld kleiner als das aktuelle Element ist. Nur in diesem Fall muß eine Vertauschung erfolgen, die mit Hilfe der Variablen HILF durchgeführt wird.

Variablenübersicht

Bereits an dieser Stelle wurde mit der Aufstellung der Variablen, Unterprogramme und Programmblöcke begonnen, um Ihnen diese Form der Programmdokumentation näherzubringen, obwohl die wenigen Variablen auch ohne diese Übersicht zu überblicken wären. Bei allen kompletten Programmen (also nicht für Übungsbeispiele) in diesem Buch werden Sie diese Übersicht finden. Eine Zusammenstellung der hier verwendeten Abkürzungen finden Sie im Anhang C.

Variablenübersicht für SHELLSORT Zweck: Einfaches Sortieren			1000-2100
Variablen			
Name	Art	Bereich	Bedeutung
ANZAHL	G	Integer	Anzahl der zu sortierenden Elemente = Länge des Feldes
FELD( )	G/E/A	Dezimalzahlen	Zu sortierende Daten
FERTIG	L	0,1	Merker für Ende Sortieren
HILF	H	Dezimalzahlen	Hv. zum Vertauschen
I	H	1. . . ANZAHL	Laufvariable
TIME	S	Zeit	Systemvariable: Einschaltzeit
ZEITANFANG	L	Zeit	Zeit beim Start des Sortiervorgangs
Unterprogramme			
Zeile	Bedeutung		wird aufgerufen in Zeile
2000	SHELLSORT		1120
Programmaufteilung in Blöcke			
1. Zeile	Bedeutung		
1000 2000	Testrahmen SHELLSORT		

Abb. 1.3: Variablenübersicht für SHELLSORT

Minsort/Maxsort

Etwas besser als das vorher beschriebene Verfahren verhält sich Minsort/Maxsort bei hochgradig unsortierten Daten. Wenn man aufsteigend sortieren will (Minsort), wird jeweils das Feld nach dem kleinsten Element durchsucht. Dieses kleinste Element wird dann mit dem ersten Element vertauscht, so daß das erste Element danach irgendwo anders im Rest des Feldes zu finden ist. Der erste Eintrag des Feldes enthält somit das kleinste Element des gesamten Feldes. Nun wird das Verfahren ab dem zweiten Element erneut durchgeführt, und das zweite Element des Feldes enthält dann das zweitniedrigste Element. Dies wird bis zum vorletzten Element durchgeführt, da die Sortierung der letzten beiden Felder gegebenenfalls durch Vertauschen durchgeführt wird.

```
1000 REM ------------------------------
1010 REM ---      RAHMENPROGRAMM      ---
1020 REM ------------------------------
1030 ANZAHL=1000
1040 DIM FELD(ANZAHL)
1050 RANDOMIZE 464
1060 FOR I=1 TO ANZAHL
1070    FELD(I)=INT(RND(6)*1000)+1
1080    REM PRINT I,FELD(I)
1090 NEXT
1100 ZEITANFANG=TIME
1110 GOSUB 2000
1120 PRINT "SEKUNDEN";(TIME-ZEITANFANG)/
        300
1130 END
1140 FOR I=1 TO ANZAHL
1150    PRINT I,FELD(I)
1160 NEXT
1170 END
2000 REM ------------------------------
2010 REM ---      MINSORT             ---
2020 REM ------------------------------
2030    FOR I=1 TO ANZAHL-1
2040       MINIMUM=1000000
2050       FOR J=I TO ANZAHL
2060          IF FELD(J) < MINIMUM
              THEN MINIMUM=FELD(J) :
                   FELDNR=J
2070          REM PRINT I;J;MINIMUM;
                   FELDNR
2080       NEXT
2090       HILF          = FELD(I)
2100       FELD(I)       = FELD(FELDNR)
2110       FELD(FELDNR)  = HILF
2120    NEXT
2130 RETURN
```

Abb. 1.4: Listing für das MINSORT-Programm

Programmbeschreibung

Der Vorspann zum Testen des Programms MINSORT ist Ihnen ja bereits aus dem letzten Kapitel bekannt, so daß wir gleich zum Sortierunterprogramm übergehen können.

Auch hier wird das ganze Feld von vorne bis hinten (vorletztes Element) durchlaufen, jedoch nicht beliebig oft, sondern nur einmal. Eine Hilfsvariable MINIMUM wird zunächst so hoch angesetzt, daß keines der Elemente des zu sortierenden Feldes größer ist. Dann wird jeweils der Rest des Feldes durchlaufen und nach dem Minimum gesucht. Der Befehl MIN kann hier nicht verwendet werden, da er nur das Minimum aus einer vorgegebenen Liste heraussucht und nicht aus einem ganzen Feld. Wer sich den Ablauf des Programms verdeutlichen möchte, kann in Zeile 2070 das REM wieder entfernen.

Nachdem der Rest des Feldes durchlaufen wurde, steht in MINIMUM das kleinste gefundene Element. Dies ist jedoch nicht so wichtig wie die Nummer (Index) des Feldelementes, in dem sich dieses Element befindet (FELDNR).

In den Zeilen 2090 bis 2110 wird jetzt dieses minimale Element mit dem ersten noch nicht sortierten Element vertauscht, und die Minimumsuche wird, vom nächsten (noch nicht sortierten) Element ausgehend, neu gestartet.

Variablenübersicht

Variablenübersicht für MINSORT Zweck: Mittelschnelles Sortieren			1000-2130
Variablen			
Name	Art	Bereich	Bedeutung
ANZAHL	G	Integer	Anzahl der zu sortierenden Elemente = Länge des Feldes
FELD( )	G/E/A	Dezimalzahlen	Zu sortierende Daten
FELDNR	H	1. . . ANZAHL	Index des aktuell kleinsten Elementes
FERTIG	H	0,1	Merker für Ende des Sortierens
HILF	H	Dezimalzahlen	Hv. zum Vertauschen
I	H	1. . . ANZAHL	Laufvariable
MINIMUM	H	Dezimalzahlen	kleinstes gefundenes Element in einem Durchgang
TIME	S	Zeit	Systemvariable: Einschaltzeit
ZEITANFANG	L	Zeit	Zeit beim Start des Sortiervorgangs
Unterprogramme			
Zeile	Bedeutung		wird aufgerufen in Zeile
2000	MINSORT		1110
Programmaufteilung in Blöcke			
1. Zeile	Bedeutung		
1000	Testrahmen		
2000	MINSORT		

Abb. 1.5: Variablenübersicht für MINSORT

Quicksort

Das schnellste bisher bekannte Verfahren ist Quicksort. Das Verfahren ist etwas komplizierter, aber nicht undurchschaubar. Im Prinzip wird mit jedem Durchlauf das Feld in zwei Abschnitte teilsortiert, wobei in dem einen Abschnitt alle Elemente gesammelt werden, die kleiner als ein Vergleichselement sind, und in dem anderen Teil alle Elemente, die größer als das Vergleichsele-

ment sind. Auf diese beiden Abschnitte kann man nun wieder ein beliebiges Sortierverfahren anwenden, z. B. Quicksort. Bei sehr kleinen Feldern ist allerdings der Rahmenaufwand zum Sortieren (Kellerspeicher füllen und entleeren, Vertauschen der Elemente) relativ groß, so daß hier ein anderes Sortierverfahren herangezogen werden sollte, um Rechenzeit zu sparen. Im Beispiel dazu haben wir das im letzten Kapitel beschriebene Unterprogramm MINSORT genommen.

Da jeweils ein Feld (oder später die einzelnen Abschnitte) in zwei (weitere) Abschnitte unterteilt wird und dann Quicksort wieder auf diese beiden Abschnitte angewendet wird, spricht man auch von einer rekursiven Prozedur: Das Unterprogramm ruft sich selbst wieder auf.

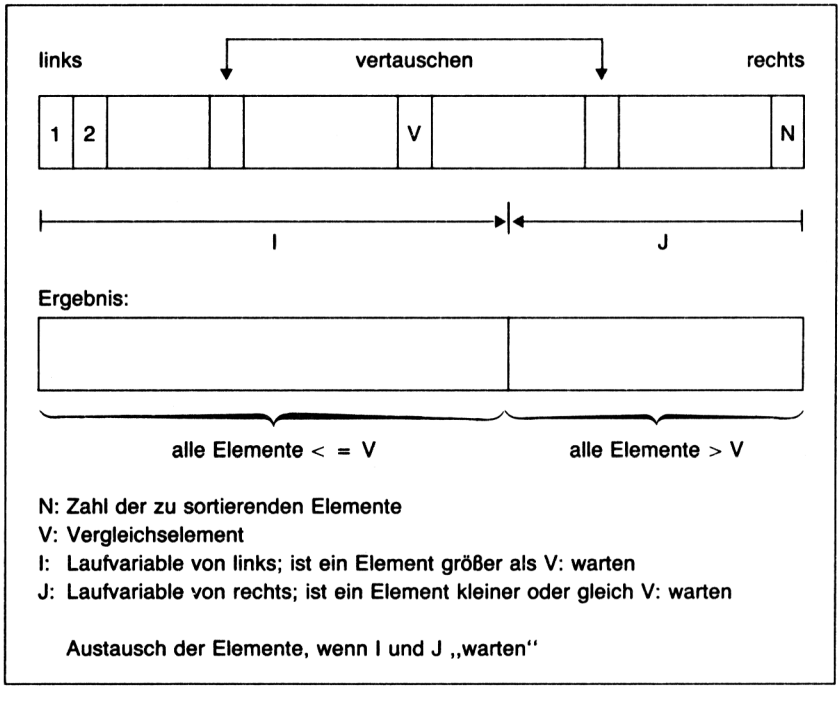


Abb. 1.6: Das Prinzip des Quicksort-Algorithmus

Da der CPC keine rekursiven Prozeduren kennt, müssen diese simuliert werden. Dazu werden zwei Felder LINKS( ) und RECHTS( ) dimensioniert, die die Grenzen für die verschiedenen Durchläufe aufnehmen sollen. Die prinzipielle Vorgehensweise ist folgende: Ein Zeiger (hier I) läuft von links nach rechts und vergleicht alle Elemente mit dem Vergleichselement. Findet er ein

Element, das größer als das Vergleichselement ist, so bleibt er stehen. Dann startet ein Zeiger von rechts (hier J), der alle Elemente wiederum vergleicht, aber anhält, wenn das betreffende Element des Datensatzes kleiner als das Vergleichselement ist. Beachten Sie: Links von dem Element mit der Nummer I sind nun alle Zahlen kleiner als das Vergleichselement, und rechts vom Feldelement mit der Nummer J sind alle Elemente größer. Wenn Sie nun diese Elemente I und J vertauschen, trifft dies auch für diese beiden Elemente zu, und die beiden Laufvariablen I und J können ihre Arbeit - wie eben - wieder aufnehmen. Sobald I und J aufeinandertreffen, ist das Ziel der ersten Teilsortierung erreicht. Die so erhaltenen Grenzen der beiden Abschnitte werden nun in die Felder LINKS( ) und RECHTS( ) abgelegt und der Vorgang startet erneut.

Mit etwas Probieren findet man heraus, daß ab zehn Elementen ein anderes Sortierverfahren günstiger ist. Wird nun von unserem provisorischen Stapel - LINKS( ) und RECHTS( ) - das nächste Grenzenpaar herangezogen, dann wird zunächst geprüft, ob diese Grenzen mehr als neun Elemente auseinanderliegen. Wenn ja, geht Quicksort in die nächste Runde, ansonsten werden die Elemente mit Minsort sortiert.

Abb. 1.7: Listing für das QUICKSORT-Programm

```
1000 REM ------------------------------
1010 REM ---      RAHMENPROGRAMM      ---
1020 REM ------------------------------
1030 ANZAHL=10
1040 DIM FELD(ANZAHL)
1050 DIM LINKS(1000)
1060 DIM RECHTS(1000)
1070 RANDOMIZE 464
1080 FOR I=1 TO ANZAHL
1090    FELD(I)=INT(RND(6)*1000)+1
1100    REM FELD(I)=I
1110    REM PRINT I,FELD(I)
1120 NEXT
1130 ZEITANFANG=TIME
1140 GOSUB 3000
1150 PRINT "SEKUNDEN";(TIME-ZEITANFANG)/
          300
1160 FOR I=1 TO ANZAHL
1170    PRINT I,FELD(I)
1180 NEXT
1190 END
2000 REM ------------------------------
2010 REM ---      MINSORT             ---
2020 REM ------------------------------
2030    REM PRINT "MINSORT","LINKS";
                LINKS;"RECHTS";RECHTS
2040    FOR I=LINKS TO RECHTS-1
2050       MINIMUM=1000000
2060       FOR J=I TO ANZAHL
2070          IF FELD(J) < MINIMUM
                THEN MINIMUM=FELD(J) :
                     FELDNR=J
2080       NEXT
2090       HILF          = FELD(I)
2100       FELD(I)       = FELD(FELDNR)
2110       FELD(FELDNR)  = HILF
2120    NEXT
2130 RETURN
```

```
3000 REM ------------------------------
3010 REM ---         QUICKSORT        ---
3020 REM ------------------------------
3030    ZEIGER       = 1
3040    LINKS(1)     = 1
3050    RECHTS(1)    = ANZAHL
3060    LINKS        = LINKS(ZEIGER)
3070    RECHTS       = RECHTS(ZEIGER)
3080    ZEIGER       = ZEIGER-1
3090    IF RECHTS-LINKS < 10
          THEN GOSUB 2000 : GOTO 3260
3100    VERGLEICH=FELD((LINKS+RECHTS)/2)
3110    I = LINKS
3120    J = RECHTS
3130    REM PRINT "QUICKSORT","LINKS";
                I;"RECHTS";J
3140    WHILE (1=1)
3150       IF FELD(I) < VERGLEICH
             THEN I=I+1 : GOTO 3150
3160       IF FELD(J) > VERGLEICH
             THEN J=J-1 : GOTO 3160
3170       IF I>J THEN 3240
3180       HILF     = FELD(I)
3190       FELD(I)  = FELD(J)
3200       FELD(J)  = HILF
3210       I=I+1
3220       J=J-1
3230    IF I<J THEN
        WEND
3240    IF LINKS<J
          THEN ZEIGER=ZEIGER+1 :
               LINKS(ZEIGER)=LINKS :
               RECHTS(ZEIGER)=J
3250    IF I<RECHTS
          THEN ZEIGER=ZEIGER+1 :
               LINKS(ZEIGER)=I :
               RECHTS(ZEIGER)=RECHTS
3260    IF ZEIGER>0 THEN GOTO 3060
3270 RETURN
```

Abb. 1.7: Listing für das QUICKSORT-Programm (Fortsetzung)

Programmbeschreibung

Das Rahmenprogramm ist Ihnen ja aus den beiden vorhergehenden Abschnitten schon bekannt, achten Sie jedoch auf die Zeilen 1050 und 1060, in denen die für die Umsetzung des rekursiven Aufrufs benötigten Felder LINKS( ) und RECHTS( ) definiert werden. In dem Unterprogramm MINSORT wurde auch nur Zeile 2030 geändert, damit Sie sich auf dem Bildschirm anschauen können, wie das Sortieren fortschreitet.

In dem Unterprogramm QUICKSORT ist durch die Felder LINKS( ) und RECHTS( ) ein rekursiver Aufruf simuliert worden. Diese beiden Felder stellen den Stack (Kellerspeicher, Stapel) dar, und die Variable ZEIGER zeigt auf das nächste Element im Stack. Dieser Zeiger wird zunächst auf das erste Element gesetzt, und in den Stack werden die beiden Grenzen des Feldes (1 und ANZAHL) eingetragen. In Zeile 3060 beginnt das Programmstück, das bis zur Fertigstellung des Sortierens immer wieder durchlaufen wird, daher müssen in die Hilfsvariablen LINKS und RECHTS die aktuellen Werte aus den entsprechenden Feldern übernommen werden. Da diese vorgegebenen Grenzen im folgenden bearbeitet werden, kann der Zeiger um eins ver-

mindert werden, um auf die nächsten zu bearbeitenden Sortiergrenzen zu zeigen.

In Zeile 3090 erfolgt die Abfrage, ob weniger als zehn Elemente zu sortieren sind; in diesem Falle wird das Unterprogramm MINSORT aufgerufen und zur Überprüfung in Zeile 3260 gesprungen, wo abgefragt wird, ob bereits alles sortiert wurde.

Wie bereits erwähnt, ist es durch geschickte Wahl eines Vergleichselementes möglich, Rechenzeit zu sparen. Wir wählen also aus den zu sortierenden Teilfeldern das mittlere Element aus. Die beiden Laufvariablen I und J werden in den Zeilen 3110 und 3120 mit aktuellen Grenzen besetzt, und wer will, kann sich in Zeile 3130 den aktuellen Vorgang am Bildschirm ausgeben lassen.

Eine prinzipielle Endlosschleife mittels der Anweisungen WHILE und WEND haben Sie bereits im Abschnitt „Ein einfaches Sortierverfahren" kennengelernt. Es wurde jedoch in der Bedingung ein Abbruchkriterium angegeben. In Zeile 3140 gehen wir noch einen Schritt weiter. Da wir nicht wissen, wie oft die folgenden Anweisungen durchlaufen werden müssen, und auch kein Endekriterium festlegen können, haben wir hier eine Endlosschleife konstruiert, die nie abbricht, da die Bedingung „1 = 1" immer erfüllt ist. Die Schleife wird aufgrund einer IF-Abfrage in Zeile 3230 verlassen.

In den Zeilen 3150 und 3160 wird, jeweils von beiden Enden des Feldes ausgehend, mit dem Vergleichselement überprüft, wie es im Text vorher beschrieben ist. Das Abbruchkriterium für das „Gegeneinanderlaufen" von I und J finden Sie in Zeile 3170, und ab Zeile 3180 werden die Elemente vertauscht, sowie, ab Zeile 3210, einerseits um eins erhöht und andererseits um eins vermindert, um sie dann weiter gegeneinanderlaufen lassen zu können.

Den Zeilen 3240 und 3250 ist besondere Beachtung zu schenken, da sie Spezialfälle beinhalten. Haben sich (bei sehr kleinen Grenzen) I oder J nicht verändert, so braucht ein entsprechendes Teilfeld nicht sortiert zu werden, weil es nicht vorhanden ist. Der THEN-Teil wird im Normalfall ausgeführt, d. h. das zu sortierende Teilfeld wird an dem Treffpunkt der Variablen I und J aufgeteilt. Beide Teilfelder werden mit ihren Grenzen auf den Stack gelegt.

Man sollte sich hier verdeutlichen, daß nicht immer der Stack um zwei Elemente erweitert wird, da auch das Sortierverfahren MINSORT Teilbereiche aufarbeitet, aber keine neuen Werte auf den Stack legt. Abbruchkriterium des Sortierverfahrens ist eine 0 in der Variablen ZEIGER.

Variablenübersicht

Variablenübersicht für QUICKSORT 1000-3270
Zweck: Schnelles Sortieren

Variablen

Name	Art	Bereich	Bedeutung
ANZAHL	G	Integer	Anzahl der zu sortierenden Elemente = Länge des Feldes
FELD( )	G/E/A	Dezimalzahlen	Zu sortierende Daten
FELDNR	H	1. . . ANZAHL	Index des aktuell kleinsten Elementes
HILF	H	Dezimalzahlen	Hv. zum Vertauschen
I	H	1. . . ANZAHL	Laufvariable
J	H	1. . . ANZAHL	Laufvariable
LINKS	H/E	1. . . ANZAHL	Linke Grenze des aktuellen Teilfeldes
LINKS( )	G	1. . . ANZAHL	Hilfsstack für linke Grenze von noch zu bearbeitenden Teilfeldern
MINIMUM	H	Dezimalzahlen	kleinstes gefundenes Element in einem Durchgang
RECHTS	H/E	1. . . ANZAHL	Rechte Grenze des aktuellen Teilfeldes
RECHTS( )	G	1. . . ANZAHL	Hilfsstack für rechte Grenze von noch zu bearbeitenden Teilfeldern
TIME	S	Zeit	Systemvariable: Einschaltzeit
VERGLEICH	H	Dezimalzahlen	Vergleichselement als Vertauschkriterium
ZEIGER	H	1. . . ANZAHL	Zeiger auf aktuelle Elemente in LINKS( ) und RECHTS( )
ZEITANFANG	L	Zeit	Zeit beim Start des Sortiervorgangs

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
2000	MINSORT	3090
3000	QUICKSORT	1140

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Testrahmen
2000	MINSORT
3000	QUICKSORT

Abb. 1.8: Variablenübersicht für QUICKSORT

Zusammenfassung

Es gibt noch einige Sortierverfahren mehr, wie z. B. das Sortieren mit Hilfe von Binärbäumen (oder allgemein Suchbäume), das wir jedoch nicht an dieser Stelle erörtern wollen, da die vorgestellten Verfahren für die Verwendung mit dem CPC sicherlich ausreichen. Vergleicht man die Geschwindigkeit der drei vorgestellten Verfahren, so ergibt sich folgendes Bild:

Zahl der Elemente	Zeit in Sekunden					
	unsortiert			sortiert		
	Bubble-Sort	Minsort/ Maxsort	Quicksort	Bubble-Sort	Minsort/ Maxsort	Quicksort
10	1,08	0,66	1,17	0,22	0,63	1,13
100	92,4	38,2	37,8	1,03	37,59	30,31
1000	10560,4 ca. 3 Std.	3560,5 ca. 1 Std.	3085	9,04	3555	2706,75
5000	268454,6 ca. 74,5 Std.	88052 ca. 24,5 Std.	76260 ca. 21 Std.	44,62	87980 ca. 24,5 Std.	70250 ca. 19,5 Std.

Abb. 1.9: Programmlaufzeiten für die vorgestellten Sortierverfahren

Ist N die Anzahl der zu sortierenden Elemente, so ist unmittelbar einsichtig, daß der Aufwand jeweils mit N wächst, aber bei den verschiedenen Verfahren unterschiedlich schnell: Bei dem ersten vorgestellten Verfahren ist der Aufwand - im schlechtesten Fall - proportional N hoch 2, bei Minsort/Maxsort (N hoch 2)/2 und bei Quicksort N*ln(N). Wie wir bereits bei Quicksort gesehen haben, ist jedoch der Aufwand nicht alleine abhängig von der Zahl der zu sortierenden Elemente, sondern auch von dem Programm selbst und natürlich seinem Aufwand. Bezieht man den Programmaufwand in die Kalkulation mit ein, so ergibt sich nach ,,Knuth, Arts of Computerprogramming, Art 3, Sorting and Searching" folgendes Bild:

Primitives Verfahren : 2*N hoch 2 + 9*N
Heap-Sort (Suchbäume) : 23*ln(N) + 0,2*N
Quicksort : 12N*ln(N) + 2*N

Damit beträgt der Faktor zwischen Minsort/Maxsort und Quicksort immerhin noch 100. Aber man muß auch den Zustand der zu sortierenden Daten berücksichtigen, wie obige Abbildung zeigt. Bei dem einfachen Verfahren ist dies besonders wichtig; bei Minsort/Maxsort ist es jedoch unerheblich. Bei

Quicksort wiederum ist der Aufwand bei vorsortierten Daten sehr stark von dem gewählten Vergleichselement abhängig. Daher wurde auch das mittlere Element gewählt. Haben Sie z. B. eine Adreßdatei, die bereits vorsortiert ist und bei der nur einige Ergänzungen gemacht werden, so ergibt sich bei der vorliegenden Wahl des Vergleichselementes mit dem Verfahren Quicksort eine fast optimale Rechenzeit.

Daß die Zeitunterschiede in der Abb. 1.9 nicht so gravierend sind, liegt an der Verwendung eines BASIC-Interpreters. Da in Quicksort mehr Programmaufwand steckt, ist die Schnelligkeit des verwendeten Programms (und im besonderen der Unterschied zwischen Interpretern und Compilern) von großer Bedeutung. Die von Knuth angegebenen Daten beziehen sich auf Großrechner mit entsprechend optimierten Compilern.

Quicksort mit Hilfszeigern

Da bei Zahlen selten mit Verwendung von Hilfszeigern sortiert wird, haben wir Quicksort im folgenden auf Sortieren von Zeichenketten umgestellt. Im Prinzip läuft das Verfahren genauso wie im Abschnitt „Quicksort" geschildert ab, nur daß die Variablen auf Zeichenketten ausgelegt sind und beim Vertauschen jeweils der Zeiger auf das Element (Feldnummer) mit vertauscht werden muß, um in dem Hilfsfeld die Zeiger so zu sortieren, das später die zu dem zu sortierenden Element zugehörigen Daten vom Rechner ausgelesen werden können.

Abb. 1.10: Listing für das QUICKSORT-Programm mit Hilfszeiger

```
1000 REM ------------------------------
1010 REM ---      RAHMENPROGRAMM      ---
1020 REM ------------------------------
1030 ANZAHL=10
1040 DIM FELD(ANZAHL)
1050 DIM FELD$(ANZAHL)
1060 DATA Schneider,Sybex,Floppy,Drucker
1070 DATA Rechner,Recorder,Tastatur
1080 DATA Bildschirm,Interface,Papier
1090 FOR I=1 TO ANZAHL
1100    READ FELD$(I)
1110    PRINT I,FELD$(I)
1120 NEXT
1130 DIM LINK%(1000)
1140 DIM RECHTS(1000)
1150 RANDOMIZE 464
1160 ZEITANFANG=TIME
1170 GOSUB 3000
1180 PRINT "SEKUNDEN";(TIME-ZEITANFANG)/
          300
1190 FOR I=1 TO ANZAHL
1200    PRINT I,FELD$(I)
1210 NEXT
1220 END
```

```
2000 REM ------------------------------
2010 REM ---      MINSORT             ---
2020 REM ------------------------------
2030     REM PRINT "MINSORT","LINKS";
                 LINKS;"RECHTS";RECHTS
2040     FOR I=LINKS TO RECHTS-1
2050        MINIMUM$="ZZZZZZZZZZ"
2060        FOR J=I TO ANZAHL
2070           IF FELD$(J) < MINIMUM$
                  THEN MINIMUM$=FELD$(J)
                       :FELDNR=J
2080        NEXT
2090        HILF$          = FELD$(I)
2100        FELD$(I)       = FELD$(FELDNR)
2110        FELD$(FELDNR)  = HILF$
2120        FELD(I)        = FELDNR
2130     NEXT
2140 RETURN
3000 REM ------------------------------
3010 REM ---         QUICKSORT        ---
3020 REM ------------------------------
3030     ZEIGER       = 1
3040     LINKS(1)     = 1
3050     RECHTS(1)    = ANZAHL
3060     LINKS        = LINKS(ZEIGER)
3070     RECHTS       = RECHTS(ZEIGER)
3080     ZEIGER       = ZEIGER-1
3090     IF RECHTS-LINKS < 3
           THEN GOSUB 2000 : GOTO 3330
3100     VERGLEICH$ =
              FELD$((LINKS+RECHTS)/2)
3110     I = LINKS
3120     J = RECHTS
3130     REM PRINT "QUICKSORT","LINKS";
                 I;"RECHTS";J
3140     WHILE (1=1)
3150        IF FELD$(I) < VERGLEICH$
              THEN I=I+1 : GOTO 3150
3160        IF FELD$(J) > VERGLEICH$
              THEN J=J-1 : GOTO 3160
3170        IF I>J THEN 3310
3180 :
3190        HILF$     = FELD$(I)
3200        HILF      = FELD(I)
3210 :
3220        FELD$(I) = FELD$(J)
3230        FELD(I) = FELD(J)
3240 :
3250        FELD$(J) = HILF$
3260        FELD(J) = HILF
3270 :
3280        I=I+1
3290        J=J-1
3300     IF I<J THEN
         WEND
3310     IF LINKS<J
           THEN ZEIGER=ZEIGER+1 :
                LINKS(ZEIGER)=LINKS :
                RECHTS(ZEIGER)=J
3320     IF I<RECHTS
           THEN ZEIGER=ZEIGER+1 :
                LINKS(ZEIGER)=I :
                RECHTS(ZEIGER)=RECHTS
3330     IF ZEIGER>0 THEN GOTO 3060
3340 RETURN
```

Abb. 1.10: Listing für das QUICKSORT-Programm mit Hilfszeiger (Fortsetzung)

Programmbeschreibung

Da die allgemeinen Abläufe vorher schon besprochen wurden, werden wir im folgenden nur auf die Änderungen eingehen. Die erste Änderung befindet

Variablenübersicht für QUICKSORT (Zeiger) 1000-3340
Zweck: Schnelles Sortieren mit Hilfsfeld

Variablen

Name	Art	Bereich	Bedeutung
ANZAHL	G	Integer	Anzahl der zu sortierenden Elemente = Länge des Feldes
FELD( )	G/E/A	Integer	Zeiger auf die zu sortierenden Daten
FELD$( )	G/E/A	1. . . ANZAHL/ Strings	Zu sortierende Daten
FELDNR	H	1. . . ANZAHL	Index des aktuell kleinsten Elementes
HILF	H	Integer	Hv. zum Vertauschen
HILF$	H	Strings	Hv. zum Vertauschen
I	H	1. . . ANZAHL	Laufvariable
J	H	I. . . ANZAHL	Laufvariable
LINKS	H/E	1. . . ANZAHL	Linke Grenze des aktuellen Teilfeldes
LINKS( )	G	1. . . ANZAHL	Hilfsstack für linke Grenze von noch zu bearbeitenden Teilfeldern
MINIMUM$	H	Strings	kleinstes gefundenes Element in einem Durchgang
RECHTS	H/E	1. . . ANZAHL	Rechte Grenze des aktuellen Teilfeldes
RECHTS( )	G	1. . . ANZAHL	Hilfsstack für rechte Grenze von noch zu bearbeitenden Teilfeldern
TIME	S	Zeit	Systemvariable: Einschaltzeit
VERGLEICH$	H	Strings	Vergleichselement als Vertauschkriterium
ZEIGER	H	1. . . ANZAHL	Zeiger auf aktuelle Elemente in LINKS( ) und RECHTS( )
ZEITANFANG	L	Zeit	Zeit beim Start des Sortiervorgangs

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
2000	MINSORT	3090
3000	QUICKSORT	1170

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Testrahmen
2000	MINSORT
3000	QUICKSORT

sich bereits in Zeile 1050, wo neben dem Feld FELD( ) ein weiteres Feld mit dem Namen FELD$( ) dimensioniert wird. Letzteres soll die eigentlichen Daten aufnehmen, wobei die Variable FELD( ) hier den Zeiger auf das entsprechende Element beinhaltet. Da aus algorithmischen Gründen sowohl bei Minsort als auch bei Quicksort die zu sortierenden Daten auch vertauscht werden müssen, ist dieses Verfahren im vorliegenden Beispiel eigentlich unnötig. Wenn man jedoch die Zeichenketten als Teil eines Feldes auffaßt, wobei beim Ausdruck das gesamte Feld nach der sortierten Ordnung ausgegeben werden soll, kann man die Zeiger als Feldindizes interpretieren.

Die nächste wichtige Änderung ist in Zeile 2050 durchgeführt, wo die Variable MINIMUM$ natürlich mit der größtmöglichen Zeichenkette besetzt wird, wenn nur Buchstaben verwendet werden ist dies Z. Zusätzlich zum Vertauschen der Feldelemente des zu sortierenden Feldes (dies muß gemacht werden, damit der Algorithmus korrekt arbeitet) muß in Zeile 2120 dem entsprechenden Element in FELD( ) die ermittelte Nummer des kleinsten Feldelementes zugewiesen werden.

Bei Quicksort reicht die einfache Vertauschung der zu sortierenden Daten und die ausschließliche Zuweisung an die Variable FELD( ) nicht aus. Hier müssen auch die als Zeiger fungierenden Variablen mit vertauscht werden.

SELEKTIEREN

Neben dem Sortieren ist das Selektieren in der Datenverarbeitung - hier besonders bei kommerziellem Einsatz - von entscheidender Bedeutung. Durch Selektierverfahren ist es möglich, z. B. alle Kunden mit negativen Umsätzen herauszufinden oder alle Kunden, die einen gewissen Umsatz übersteigen, oder alle Kunden aus dem Postleitzahlengebiet 8, oder alle Kunden, die zu einem bestimmten Vertreter gehören, oder alle Kunden, die an einem bestimmten Tag Geburtstag haben, oder . . . herauszufiltern. Aber auch im privaten Bereich gibt es viele Möglichkeiten, bei denen das Selektieren eingesetzt werden kann: Heraussuchen aller Songs eines Interpreten aus einer Schallplatten-/Tonbandsammlung, Heraussuchen eines bestimmten Dias mit einem festgelegten Thema bzw. vorgegebenen Datum, Fehlteillisten von Briefmarken (Bestand = 0), alle Einzelteile für eine Modelleisenbahn von einem bestimmten Hersteller usw.

⬅ *Abb. 1.11: Variablenübersicht für QUICKSORT (Zeiger)*

Da in sehr vielen Fällen Adressen verwaltet werden - dies sowohl im kommerziellen als auch im privaten Bereich -, haben wir als Beispiel ein paar Adressen erfaßt, die der Vorspann zu unseren Selektierprogrammen in Abb. 1.12 zeigt.

```
1000 REM ------------------------------
1010 REM ---    Vorspann und Daten   ---
1020 REM ------------------------------
1030 DATA Huber,Alex,Alleestr.15
1040 DATA 8000 Muenchen,-100.52,1548.84
1050 DATA Meier,Sepp,Gartenstr.58
1060 DATA 5000 Koeln 51,1458.45,451.48
1070 DATA Schuba,Franz,Ringstr.67
1080 DATA 8521 Adorf/Cteil,-12.47,79.48
1090 DATA Meyer,Franz,Friedastr.56
1100 DATA 8475 Cstadt 23,142.48,474.48
1110 DATA Straus,Marga,Himmelstr.48
1120 DATA 8547 Video-City,-55.55,555.55
1130 DATA Mayer,Jupp,Meierstr.45
1140 DATA 4856 Seppstadt,145.58,8569.85
1150 DATA Maier,Stefan,CPC-Str.464
1160 DATA 8012 Ottobrunn,178.54,4789.59
1170 DATA Mayer-Mueller,H.,Floppystr.15
1180 DATA 7896 Diskhausen,-78.45,905
1190 :
1200 ANZAHL=8
1210 :
1220 DIM NAME$    (ANZAHL)
1230 DIM VORNAME$ (ANZAHL)
1240 DIM STRASSE$ (ANZAHL)
1250 DIM PLZORT$  (ANZAHL)
1260 DIM KONTO    (ANZAHL)
1270 DIM UMSATZ   (ANZAHL)
1280 :
1290 FOR I=1 TO ANZAHL
1300     READ NAME$    (I)
1310     READ VORNAME$ (I)
1320     READ STRASSE$ (I)
1330     READ PLZORT$  (I)
1340     READ KONTO    (I)
1350     READ UMSATZ   (I)
1360 NEXT
1370 :
1380 MODE 2
1390 PRINT "Kontrollausdruck"
1400 PRINT
1410 FOR I=1 TO ANZAHL
1420     PRINT NAME$(I);" ";VORNAME$(I),
               STRASSE$(I),PLZORT$(I),
               KONTO(I),UMSATZ(I)
1430 NEXT
1440 A$=INKEY$ : IF A$="" THEN 1440
1450 :
```

Abb. 1.12: Vorspann zu den Selektierprogrammen

Der Aufbau wurde so konzipiert, daß Sie auch Ihre eigenen Daten eintragen können, da z. B. in Zeile 1200 die Anzahl der eingegebenen Datensätze vorgegeben werden kann. In den Zeilen 1220 bis 1270 werden die entsprechenden Felder dimensioniert. Die Bedeutung der Felder ergibt sich bereits aus den

Variablennamen. Das Einlesen erfolgt in den Zeilen ab 1290, und ab Zeile 1380 wird ein Kontrollausdruck durchgeführt. Jedes der weiteren Segmente enthält auch die Warteschleife, die auf das Drücken einer Taste wartet und die in Zeile 1440 dargestellt ist.

Einfaches Selektieren nach Namen

In den folgenden Abschnitten wollen wir - außer beim Selektieren nach Zahlen - lediglich nach dem Namen selektieren. Zur Übung ist es Ihnen freigestellt, die Programmsegmente entsprechend zu ändern, wobei die Postleitzahl ein ähnlich reiches Betätigungsfeld darstellt.

```
2000 REM ------------------------------
2010 REM --- Einfaches Selektieren  ---
2020 REM ---        nach Namen        ---
2030 REM ------------------------------
2040 MODE 1
2050 INPUT"gesuchter Name";NAME$
2060 FOR I=1 TO ANZAHL
2070    IF NAME$(I)=NAME$
          THEN GOSUB 3030 : GEFUNDEN=1
2080 NEXT
2090 IF GEFUNDEN=0
       THEN PRINT"Nichts gefunden"
2100 A$=INKEY$ : IF A$="" THEN 2100
2110 GOTO 4000
2120 :
```

Abb. 1.13: Selektieren nach Namen (Teilprogramm)

Da die Ausgabe weniger als 40 Zeichen beträgt, haben wir wegen der besseren Lesbarkeit den Bildschirmmodus entsprechend umgeschaltet. Zunächst wird der gesuchte Name erfaßt, und in den Zeilen 2060 bis 2080 wird eine Schleife durchlaufen, die auf Gleichheit des eingegebenen Suchnamens mit den erfaßten Namen prüft. Ist eine Gleichheit vorhanden, wird ein Unterprogramm zur Ausgabe der Daten aufgerufen. Durch die programmierte Struktur mit dem Unterprogramm werden also alle Datensätze angezeigt, die mit dem gesuchten Namen übereinstimmen, und nicht, wie in vielen anderen Programmen, nur der erste.

Weiterhin wird der Merker GEFUNDEN auf 1 gesetzt, um zu dokumentieren, daß mindestens ein Name ausgegeben wurde. Wer sich die Trefferquote ausgeben lassen möchte, kann diese Anweisung ersetzen durch GEFUNDEN-=GEFUNDEN+1.

Ist der Wert von GEFUNDEN in Zeile 2090 gleich 0, so wird eine entsprechende Meldung ausgegeben. Dies ist wichtig, um anzuzeigen, daß der Rechner mit der Suche fertig ist. Es folgt die bekannte Warteschleife und ein Sprung auf das nächste Selektiersegment. Vorweg zeigen wir zur Vollständigkeit das Unterprogramm zur Ausgabe der Daten, das auch von den folgenden Programmteilen aufgerufen wird (Abb. 1.14).

```
3000 REM ------------------------------
3010 REM ---      Ausgabe der Daten  ---
3020 REM ------------------------------
3030     PRINT "-----------------------"
3040     PRINT "Name:",NAME$(I)
3050     PRINT "Vorname",VORNAME$(I)
3060     PRINT "Strasse",STRASSE$(I)
3070     PRINT "PLZ/Ort",PLZORT$(I)
3080     PRINT "Konto",KONTO(I)
3090     PRINT "Umsatz",UMSATZ(I)
3100     PRINT "-----------------------"
3110 RETURN
3120 :
```

Abb. 1.14: Das Unterprogramm zur Datenausgabe

Bei Eingabe von „Meier" als Suchkriterium ergibt sich die Ausgabe in Abb. 1.15.

```
-----------------------
Name:          Meier
Vorname        Sepp
Strasse        Gartenstr.58
PLZ/Ort        5000 Koeln 51
Konto           1458.45
Umsatz           451.48
-----------------------
```

Abb. 1.15: Ausgabe eines gesuchten Namens

Einfaches Selektieren nach Zahlenbereichen

Da nur in speziellen Fällen nach einer bestimmten Zahl selektiert wird, haben wir diesen Teil von vornherein für Zahlenbereiche ausgelegt. Wird nur nach einer bestimmten Zahl selektiert (Spezialfall des dargestellten Programms), so ist bei beiden Eingaben jeweils die gleiche Zahl einzugeben.

```
4000 REM ------------------------------
4010 REM --- Einfaches Selektieren  ---
4020 REM --- nach Zahlenbereichen   ---
4030 REM ------------------------------
4040 CLS
4050 GEFUNDEN=0
4060 INPUT"Von Kontostand";VON
4070 INPUT"Bis Kontostand";BIS
4080 FOR I=1 TO ANZAHL
4090    IF KONTO(I)>=VON AND KONTO(I)<=B
IS          THEN GOSUB 3030:GEFUNDEN=1
4100 NEXT
4110 IF GEFUNDEN=0
       THEN PRINT"Nichts gefunden"
4120 A$=INKEY$ : IF A$="" THEN 4120
4130 :
```

Abb. 1.16: Selektieren nach Zahlenbereichen (Teilprogramm)

Das Programmstück ist analog zum einfachen Selektieren nach Namen aufgebaut und unterscheidet sich wesentlich nur in Zeile 4090, wo in der bedingten Anweisung die beiden Kontostände mit AND verknüpft werden. Nach Eingabe von 100 (Untergrenze) und 3000 (Obergrenze) erfolgt die Ausgabe, wie in Abb. 1.17 gezeigt.

```
-----------------------
Name:         Meier
Vorname       Sepp
Strasse       Gartenstr.58
PLZ/Ort       5000 Koeln 51
Konto          1458.45
Umsatz          451.48
-----------------------
-----------------------
Name:         Meyer
Vorname       Franz
Strasse       Friedastr.56
PLZ/Ort       8475 Cstadt 23
Konto          142.48
Umsatz          474.48
-----------------------
-----------------------
Name:         Mayer
Vorname       Jupp
Strasse       Meierstr.45
PLZ/Ort       4856 Seppstadt
Konto          145.58
Umsatz          8569.85
-----------------------
-----------------------
Name:         Maier
Vorname       Stefan
Strasse       CPC-Str.464
PLZ/Ort       8012 Ottobrunn
Konto          178.54
Umsatz          4789.59
-----------------------
```

Abb. 1.17: Ausgabe der Personen mit Kontostand innerhalb eines bestimmten Bereichs

Selektieren mit Wild Cards

Die Wild Cards sind Ihnen vielleicht schon aus der Bedienung der Floppy geläufig oder von anderen Rechnern. Allgemein wird diese Handhabung auch als Präfixsuche bezeichnet, weil ein bekannter Anfangsteil (Präfix) eingegeben wird und die restlichen Zeichen irrelevant sind.

Verwendet werden kann diese Selektierweise, wenn mit den eingegebenen Zeichen vor dem * bereits eine eindeutige Zuordnung gegeben ist oder mehrere gleiche Elemente mit gleichem Anfang gefunden werden sollen. Ein Beispiel: Wird nach Namen gesucht und Sie wollen den Datensatz eines gewissen Herrn Janzcyk und sind sich nicht sicher über die genaue Schreibweise, so genügt die Eingabe „Jan*", um mindestens diesen Namen aus den vorhandenen Daten herauszufinden. Das zweite Beispiel hat die Postleitzahl zur Grundlage: Sie wollen alle Kunden aufgelistet haben, die im Postleitzahlenbereich 8 wohnhaft sind. Hier genügt die Eingabe 8*.

```
5000 REM ------------------------------
5010 REM --- Selektieren mit Wild   ---
5020 REM ---        Cards           ---
5030 REM ------------------------------
5040 CLS
5050 GEFUNDEN=0
5060 INPUT"Name (Abk. mit '*')";SUCH$
5070 STERNPOS=INSTR(1,SUCH$,"*")
5080 IF STERNPOS=0 THEN GOTO 6000
5090 FOR I=1 TO ANZAHL
5100    IF LEFT$(NAME$(I),STERNPOS-1)=
           LEFT$(SUCH$,STERNPOS-1)
        THEN GOSUB 3030 : GEFUNDEN=1
5110 NEXT
5120 IF GEFUNDEN=0
       THEN PRINT"Nichts gefunden"
5130 A$=INKEY$ : IF A$="" THEN 5130
5140 :
```

Abb. 1.18: Selektieren mit Wild Cards (Teilprogramm)

Nachdem der Bildschirm gelöscht und unser Merker GEFUNDEN zurückgesetzt wurde, wird das Suchkürzel in Zeile 5060 erfaßt. Um festzustellen, an welcher Stelle sich der * innerhalb dieses Suchstrings befindet - die Position wird zum späteren Vergleich benötigt -, ziehen wir den Befehl INSTR( ) zu Hilfe. Das Ergebnis wird in der Variablen STERNPOS abgelegt. In der nächsten Zeile muß noch überprüft werden, ob überhaupt ein * in dieser Zeichenkette enthalten war, um einen Fehler in Zeile 5100 zu vermeiden. In unserem Beispiel springen wir einfach zum nächsten Segment; in der Regel ist hier jedoch noch eine Fehlermeldung anzubringen oder eventuell ein Sprung zur

Zeile 2060 (einfaches Selektieren nach Namen), um dem Anwender auch eine Eingabe ohne * zu ermöglichen.

Zur Überprüfung in Zeile 5100 sind jetzt nur noch die Zeichen heranzuziehen, die vor dem * stehen, daher die Anwendung des Befehls LEFT$ und als entsprechende Länge STERNPOS-1.

Der Rest des Listings ist Ihnen in ähnlicher Form schon bekannt. In Abb. 1.19 sehen Sie die Ausgabe aufgrund der Suchabfrage „S*".

```
------------------------
Name:         Schuba
Vorname       Franz
Strasse       Ringstr.67
PLZ/Ort       8521 Adorf/Cteil
Konto         -12.47
Umsatz         79.48
------------------------
------------------------
Name:         Straus
Vorname       Marga
Strasse       Himmelstr.48
PLZ/Ort       8547 Video-City
Konto         -55.55
Umsatz         555.55
------------------------
```

Abb. 1.19: Programmausgabe nach der Suchabfrage „S"*

Selektieren mit Platzhaltern

Etwas anders ist die Vorgehensweise beim Selektieren mit sogenannten Platzhaltern, für die in der Regel ein ? verwendet wird. Im Gegensatz zu den Wild Cards wird nicht ab einem bestimmten Zeichen der ganze Rest der Zeichenkette vernachlässigt, sondern es werden nur einzelne Zeichen durch diesen Platzhalter ersetzt, mit der Bedeutung, daß Unterschiede in diesem einzelnen Zeichen unwichtig sind. Ein Paradebeispiel dieses Selektiervorgehens ist das Suchen eines Meier, Maier, Mayer oder Meyer. Durch die Eingabe von „M??er" mit unseren Beispieldaten ergäbe sich der Ausdruck aus Abb. 1.20.

Abb. 1.20: Programmausgabe nach der Suchabfrage „M??er"

```
------------------------
Name:         Meier
Vorname       Sepp
Strasse       Gartenstr.58
PLZ/Ort       5000 Koeln 51
Konto          1458.45
Umsatz         451.48
------------------------
```

```
------------------------
Name:          Meyer
Vorname        Franz
Strasse        Friedastr.56
PLZ/Ort        8475 Cstadt 23
Konto           142.48
Umsatz          474.48
------------------------
------------------------
Name:          Mayer
Vorname        Jupp
Strasse        Meierstr.45
PLZ/Ort        4856 Seppstadt
Konto           145.58
Umsatz          8569.85
------------------------
------------------------
Name:          Maier
Vorname        Stefan
Strasse        CPC-Str.464
PLZ/Ort        8012 Ottobrunn
Konto           178.54
Umsatz          4789.59
------------------------
```

Abb. 1.20: Programmausgabe nach der Suchabfrage „M??er" (Fortsetzung)

Die Programmierung ist etwas aufwendiger als im letzten Fall und könnte wie in Abb. 1.21 gezeigt aussehen.

```
6000 REM ------------------------------
6010 REM --- Selektieren mit Platz- ---
6020 REM ---       haltern (?)       ---
6030 REM ------------------------------
6040 CLS
6050 GEFUNDEN=0
6060 INPUT"Name mit '??'";SUCH$
6070 FOR I=1 TO ANZAHL
6075    IF LEN(SUCH$) <> LEN(NAME$(i))
          THEN 6140
6080    FOR J=1 TO LEN(SUCH$)
6090       IF MID$(SUCH$,J,1)="?"
             THEN GOTO 6110
6100       IF MID$(NAME$(I),J,1) <>
             MID$(SUCH$,J,1)
             THEN GOTO 6140
6110    NEXT J
6120 GOSUB 3030
6130 GEFUNDEN=1
6140 NEXT I
6150 A$=INKEY$ : IF A$="" THEN 6150
6160 :
```

Abb. 1.21: Selektieren mit Platzhaltern (Teilprogramm)

Innerhalb der Suchschleife wird nun eine zweite Schleife angebracht, die jeweils die einzelnen Zeichen überprüft. Ist in der Suchanfrage ein ?, so kann eine Überprüfung unterbleiben. Im anderen Fall werden die einzelnen Buchstaben der aktuell zu überprüfenden Zeichenkette und der Such-Zeichenkette

miteinander verglichen, wobei beim ersten Unterschied bereits zur Überprüfung des nächsten Datensatzes übergegangen werden kann. Aus Rechenzeitgründen ist die Überprüfung in dieser Weise (und nicht auf Gleichheit mit anderer Sprungverteilung) sehr wichtig, wenn viele Datensätze zu durchsuchen sind.

Selektieren mit Platzhaltern und Wild Cards

Wie Sie sicherlich schon festgestellt haben, ist der Doppelname „Mayer-Mueller" bisher beim Selektieren noch nicht aufgetaucht. Einen bestimmten Maier mit Wild Cards herauszufinden durch Eingabe von „M*" wäre bei großen Datenmengen relativ unsinnig, da alle mit M beginnenden Namen ausgegeben würden. Das Selektieren mittels Platzhalter für den zweiten und dritten Buchstaben brachte aber nicht den Doppelnamen, da nur auf die Länge von fünf Zeichen überprüft wurde. Wie löst man nun eine Suchanfrage, wobei folgendes bekannt ist: Ein „M??er" wird gesucht, wobei das dritte Zeichen wahrscheinlich ein y ist und ein Doppelname in Erwägung gezogen wird?

```
7000 REM ------------------------------
7010 REM --- Selektieren mit Platz- ---
7020 REM --- haltern und Wild Cards ---
7030 REM ------------------------------
7040 CLS
7050 GEFUNDEN=0
7060 INPUT"Name mit '?' und '*';SUCH$
7070 STERNPOS=INSTR(1,SUCH$,"*")
7080 IF STERNPOS=0 THEN END
7090 FOR I=1 TO ANZAHL
7100     FOR J=1 TO STERNPOS-1
7110         IF MID$(SUCH$,J,1)="?"
               THEN GOTO 7130
7120         IF MID$(NAME$(I),J,1) <>
                MID$(SUCH$,J,1)
               THEN GOTO 7160
7130     NEXT J
7140 GOSUB 3030
7150 GEFUNDEN=1
7160 NEXT I
```

Abb. 1.22: Selektieren mit Platzhaltern und Wild Cards (Teilprogramm)

Das Programmstück ist eine Zusammenfassung der letzten beiden Abschnitte, wobei besonders Zeile 7100 zu beachten ist, da das Endekriterium der Schleife jetzt die um eins verminderte Position des * ist.

Die Ausgabe sehen Sie in Abb. 1.23.

```
------------------------
Name:          Meyer
Vorname        Franz
Strasse        Friedastr.56
PLZ/Ort        8475 Cstadt 23
Konto           142.48
Umsatz          474.48
------------------------
------------------------
Name:          Mayer
Vorname        Jupp
Strasse        Meierstr.45
PLZ/Ort        4856 Seppstadt
Konto           145.58
Umsatz          8569.85
------------------------
------------------------
Name:          Mayer-Mueller
Vorname        H.
Strasse        Floppystr.15
PLZ/Ort        7896 Diskhausen
Konto          -78.45
Umsatz          905
------------------------
```

Abb. 1.23: Programmausgabe nach Suchen mit Platzhaltern und Wild Cards

Wahlweises Selektieren nach verschiedenen Datensatzelementen

Wenn Sie beim Selektieren nach allen Kriterien Speicherplatz, Rechenzeit und Arbeit sparen wollen, dann hilft Ihnen folgendes Rezept weiter: Um einheitliche Variablen zu erhalten, werden Zahlen auch als Zeichenketten dargestellt - die Sie notfalls mittels der VAL( )-Funktion umwandeln können. Sie benötigen dann nicht nur für jedes Datensatzelement ein eigenes Feld, sondern können eine Matrix heranziehen, die in unserem Beispiel INHALT$(,) heißt. Wie sieht nun das andere Programm aus? Die Daten und den Vorspann sehen Sie in Abb. 1.24.

Natürlich muß die Anzahl der Elemente des Datensatzes auch definiert wer-

Abb. 1.24: Daten und Vorspann (Teilprogramm)

```
1000 REM ------------------------------
1010 REM ---    Vorspann und Daten  ---
1020 REM ------------------------------
1030 DATA Huber,Alex,Alleestr.15
1040 DATA 8000 Muenchen,-100.52,1548.84
1050 DATA Meier,Sepp,Gartenstr.58
1060 DATA 5000 Koeln 51,1458.45,451.48
1070 DATA Schuba,Franz,Ringstr.67
1080 DATA 8521 Adorf/Cteil,-12.47,79.48
1090 DATA Meyer,Franz,Friedastr.56
1100 DATA 8475 Cstadt 23,142.48,474.48
1110 DATA Straus,Marga,Himmelstr.48
1120 DATA 8547 Video-City,-55.55,555.55
1130 DATA Mayer,Jupp,Meierstr.45
1140 DATA 4856 Seppstadt,145.58,8569.85
1150 DATA Maier,Stefan,CPC-Str.464
1160 DATA 8012 Ottobrunn,178.54,4789.59
```

```
1170 DATA Mayer-Mueller,H.,Floppystr.15
1180 DATA 7896 Diskhausen,-78.45,905
1190 :
1200 ELEMENTE=6
1210 ANZAHL  =8
1220 :
1230 DIM INHALT$(ANZAHL,ELEMENTE)
1240 :
1250 FOR I=1 TO ANZAHL
1260    FOR J=1 TO ELEMENTE
1270       READ INHALT$(I,J)
1280    NEXT
1290 NEXT
1300 :
1310 PRINT "Kontrollausdruck"
1320 PRINT
1330 FOR I=1 TO ANZAHL
1340    FOR J=1 TO ELEMENTE
1350       PRINT INHALT$(I,J)
1360    NEXT
1370    PRINT "-----------------------"
1380 NEXT
1390 :
1400 A$=INKEY$ : IF A$="" THEN 1400
1410 :
```

Abb. 1.24: Daten und Vorspann (Fortsetzung)

den, und um dies allgemein zu halten, haben wir in Zeile 1200 eine Vorbesetzung vorgenommen. Die Dimensionierung beschränkt sich nun nur noch auf eine einzige Matrix (Zeile 1230). Zum Einlesen werden zwei ineinandergeschachtelte Schleifen herangezogen (Zeile 1250 bis 1290), die auch für den Kontrollausdruck verwendet werden (Zeile 1310 bis 1380).

Wie müssen nun die einzelnen Programmzeilen zum Selektieren geändert werden? Im wesentlichen sollte aufgrund der Benutzerfreundlichkeit die Bezeichnung der einzelnen Elemente vorangestellt werden. Zusätzlich zu dem Vergleichskriterium muß natürlich erfaßt werden, welches Element zum Selektieren herangezogen werden soll, wozu die entsprechende Nummer genommen wird.

Die letzte Änderung ist die Bedingung innerhalb der Schleife, wo das jeweilige zu durchsuchende Feld durch INHALT$( ) ersetzt wird, natürlich mit der entsprechenden Kennzeichnung.

Abb. 1.25: Selektieren nach Namen und Zahlenbereichen (Teilprogramm)

```
2000 REM ------------------------------
2010 REM --- Selektieren nach Namen ---
2020 REM ------------------------------
2030 CLS
2040 PRINT"1 - Name
2050 PRINT"2 - Vorname
2060 PRINT"3 - Strasse
2070 PRINT"4 - PLZ/Ort
2080 PRINT"5 - Kontostand
2090 PRINT"6 - Umsatz
2100 :
2110 INPUT"Welche Rubrik (1-6)";WO
```

```
2120 INPUT"Was suchen";SUCH$
2130 :
2140 FOR I=1 TO ANZAHL
2150    IF INHALT$(I,WO)=SUCH$
          THEN GOSUB 3030 : GEFUNDEN=1
2160 NEXT
2170 :
2180 IF GEFUNDEN=0
       THEN PRINT"Nichts gefunden"
2190 A$=INKEY$ : IF A$="" THEN 2190
2200 GOTO 4000
2210 :
3000 REM ------------------------------
3010 REM ---      Ausgabe der Daten  ---
3020 REM ------------------------------
3030    PRINT "-----------------------"
3040    PRINT "Name",INHALT$(I,1)
3050    PRINT "Vorname",INHALT$(I,2)
3060    PRINT "Strasse",INHALT$(I,3)
3070    PRINT "PLZ/Ort",INHALT$(I,4)
3080    PRINT "Konto",INHALT$(I,5)
3090    PRINT "Umsatz",INHALT$(I,6)
3100    PRINT "-----------------------"
3110 RETURN
3120 :
4000 REM ------------------------------
4010 REM --- Selektieren nach Zahl- ---
4020 REM ---        bereichen        ---
4030 REM ------------------------------
4040 CLS
4050 GEFUNDEN=0
4060 PRINT"1 - Name
4070 PRINT"2 - Vorname
4080 PRINT"3 - Strasse
4090 PRINT"4 - PLZ/Ort
4100 PRINT"5 - Kontostand
4110 PRINT"6 - Umsatz
4120 :
4130 INPUT"Welche Rubrik (5/6)";WO
4140 INPUT"von";VON
4150 INPUT"bis";BIS
4160 :
4170 FOR I=1 TO ANZAHL
4180 HILF=VAL(INHALT$(I,WO))
4190    IF HILF>=VON AND HILF<=BIS
          THEN GOSUB 3030:GEFUNDEN=1
4200 NEXT
4210 :
4220 IF GEFUNDEN=0
       THEN PRINT"Nichts gefunden"
4230 A$=INKEY$ : IF A$="" THEN 4230
4240 :
```

Abb. 1.25: Selektieren nach Namen und Zahlenbereichen (Fortsetzung)

Auch die Ausgabe in den Zeilen 3040 bis 3090 muß entsprechend der Matrix angepaßt werden. Beim Selektieren nach Zahlenbereichen ist besonders Zeile 4180 von Bedeutung, wo die Umwandlung der als Zeichenketten dargestellten Zahlen in ihre Werte vorgenommen wird.

Auch für das Selektieren mit Wild Cards und/oder Platzhaltern sind diese wenigen Änderungen durchzuführen. Als Beispiel haben wir die Programmzeilen ab 7000 geändert und abgedruckt, wobei Sie die Teilbereiche in den Zeilen ab 5000 und 6000 als Übung selbst ändern sollten. Eine Übung für Fortgeschrittene wäre es, wenn Sie das Selektieren mit Platzhaltern und Wild Cards sowohl für Zahlen als auch Zeichenketten zulassen würden.

```
7000 REM ------------------------------
7010 REM --- Selektieren mit Platz- ---
7020 REM --- haltern und Wild Cards ---
7030 REM ------------------------------
7040 CLS
7050 GEFUNDEN=0
7060 :
7070 PRINT"1 - Name
7080 PRINT"2 - Vorname
7090 PRINT"3 - Strasse
7100 PRINT"4 - PLZ/Ort
7110 PRINT"5 - Kontostand
7120 PRINT"6 - Umsatz
7130 :
7140 INPUT"Welche Rubrik (1-6)";WO
7150 INPUT"Name mit '?' und '*'";SUCH$
7160 :
7170 STERNPOS=INSTR(1,SUCH$,"*")
7180 IF STERNPOS=0 THEN END
7190 :
7200 FOR I=1 TO ANZAHL
7210    FOR J=1 TO STERNPOS-1
7220       IF MID$(SUCH$,J,1)="?"
             THEN GOTO 7240
7230       IF MID$(INHALT$(I,WO),J,1) <>
             MID$(SUCH$,J,1)
             THEN GOTO 7270
7240    NEXT J
7250 GOSUB 3030
7260 GEFUNDEN=1
7270 NEXT I
```

Abb. 1.26: Selektieren mit Platzhaltern und Wild Cards (geändertes Teilprogramm)

Wie Sie an diesem kleinen Beispiel sehen, spart geschickte Programmierung sowohl Rechenzeit als auch Speicherplatz und vor allen Dingen Programmierarbeit. Das hier dargestellte Beispiel zur Verallgemeinerung werden wir im Kapitel „Dateiverwaltung" noch genauer erläutern.

MISCHVERFAHREN

Besonders für Besitzer des CPC in seiner Grundversion (mit Kassettenrecorder) dürfte dieses Kapitel interessant sein, wenn Sie verschiedene Teile einer logischen Datei in diversen physikalischen Dateien auf Band abgelegt haben. Neben weitaus komplizierteren Verfahren wollen wir hier nur das Ein-Phasen-Mischsortieren beschreiben.

Ziel dieses Verfahrens ist es, mehrere einzelne Dateien (die in sich sortiert sind) zu einer gemeinsamen Datei zusammenzufassen. Wenn in diesem Zusammenhang von Eingabebändern gesprochen wird, muß dies nicht immer eine Kassette sein. Sie können auch mehrere kleinere Dateien in drei verschiedenen Feldern im Hauptspeicher Ihres Rechners zum Mischen halten.

Außerdem kann das Verfahren nicht nur angewendet werden, wenn die Dateien auf mehreren Kassetten verstreut sind. Haben Sie zum Beispiel bereits

ein sortiertes Feld und wollen ein unsortiertes einfügen, so können Sie beim Sortieren Rechenzeit sparen, wenn Sie anstatt der Zusammenfassung beider Felder nur das unsortierte Feld sortieren und anschließend dieses Mischverfahren verwenden. Als Lauf gilt jeweils eine Einheit von einem in sich sortierten Datenblock, wobei auch mehrere Läufe hintereinander auf einem Band stehen können.

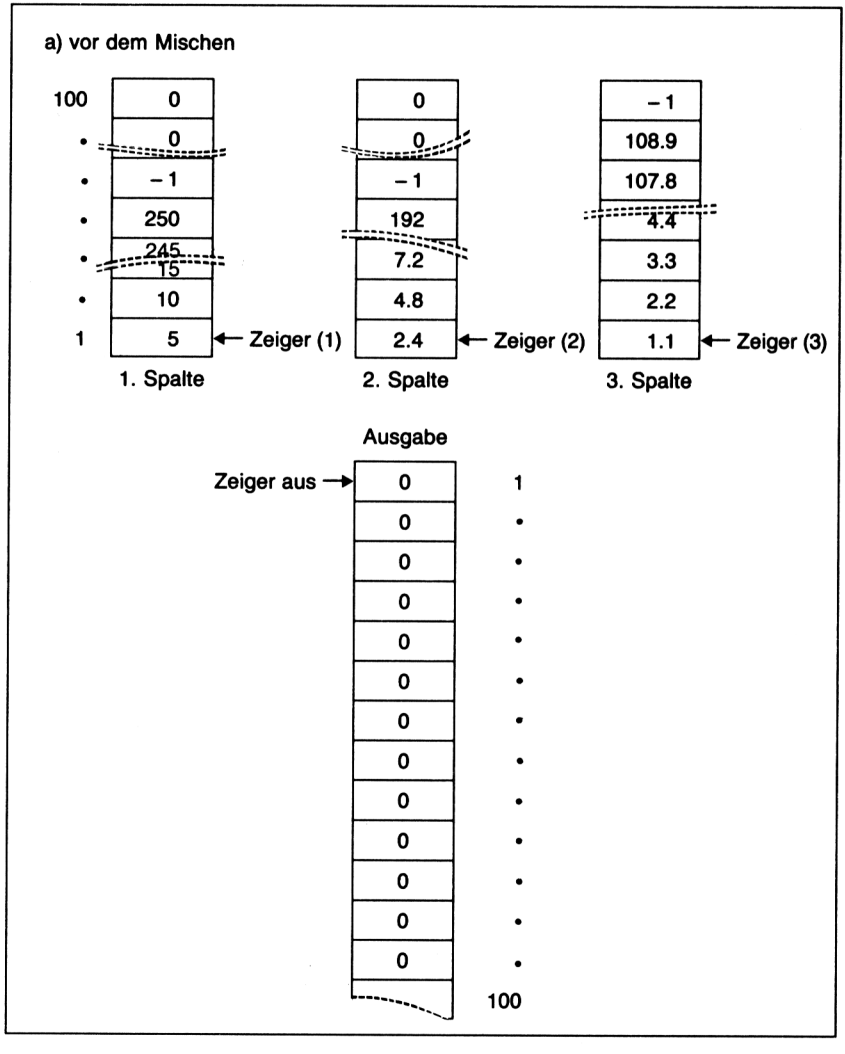


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm

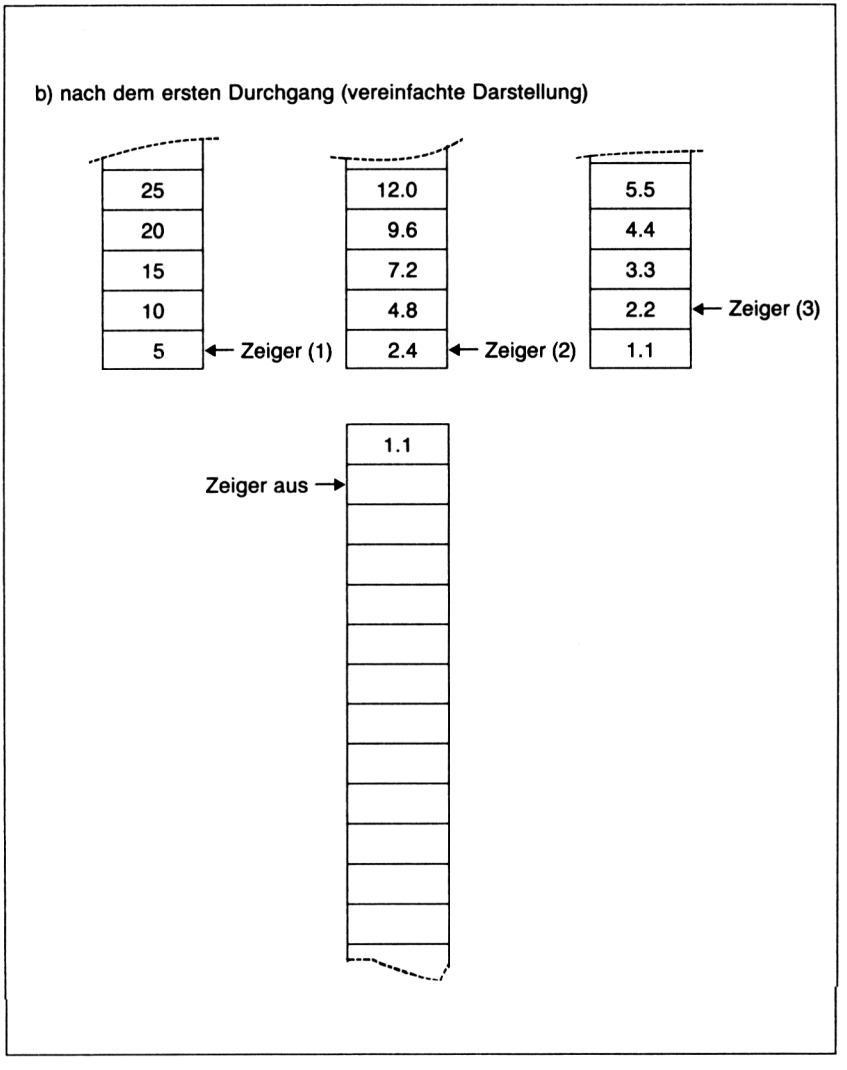


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm (Fortsetzung)

Der prinzipielle Ablauf ist folgender: Man liest von allen Eingabemedien jeweils den ersten Datensatz und vergleicht diese untereinander. Dann wählt man den kleinsten aus und schreibt diesen auf das Ausgabeband. Von dem Band des ausgegebenen Elementes liest man nun den nächsten Datensatz ein.

Dann verfährt man mit den eingelesenen Datensätzen wie vorher und sucht sich das kleinste Element heraus. Ist ein Lauf zu Ende, so wird dieses Band übergangen, bis alle Läufe zu Ende sind. Auf diese Weise entsteht ein Ausgabeband mit einem wesentlich längeren Lauf, der gegebenenfalls wieder ge-

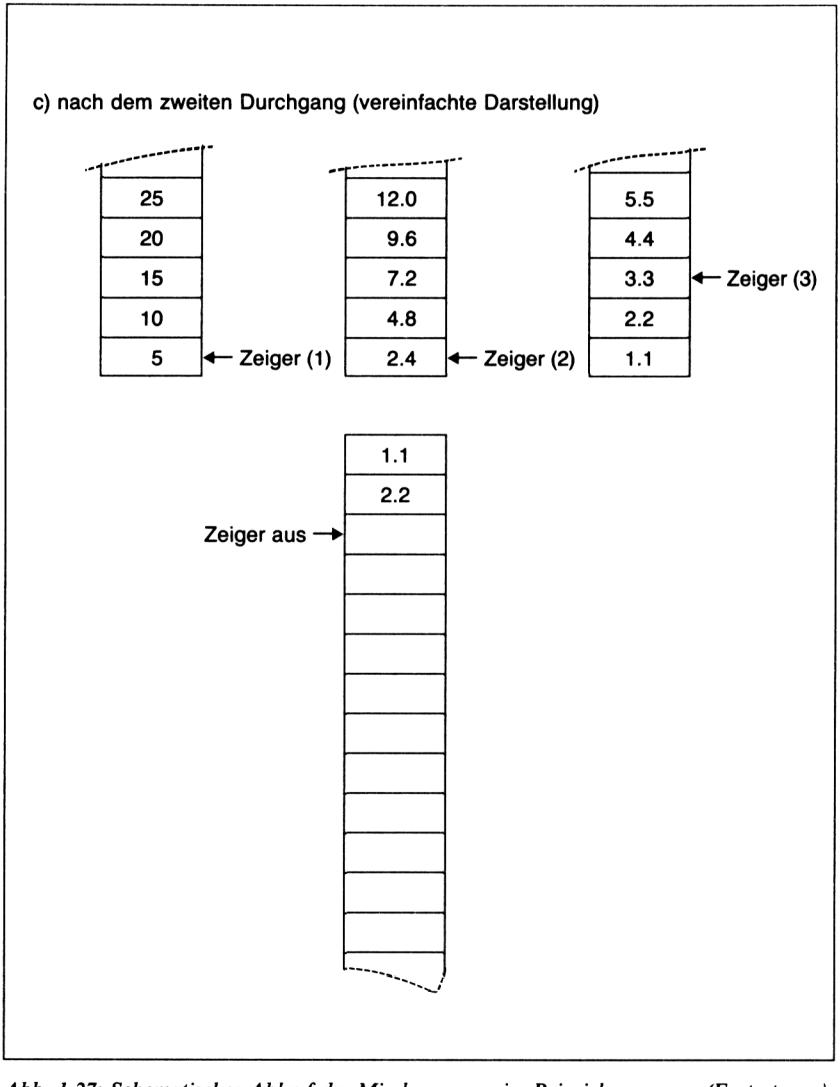


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm (Fortsetzung)

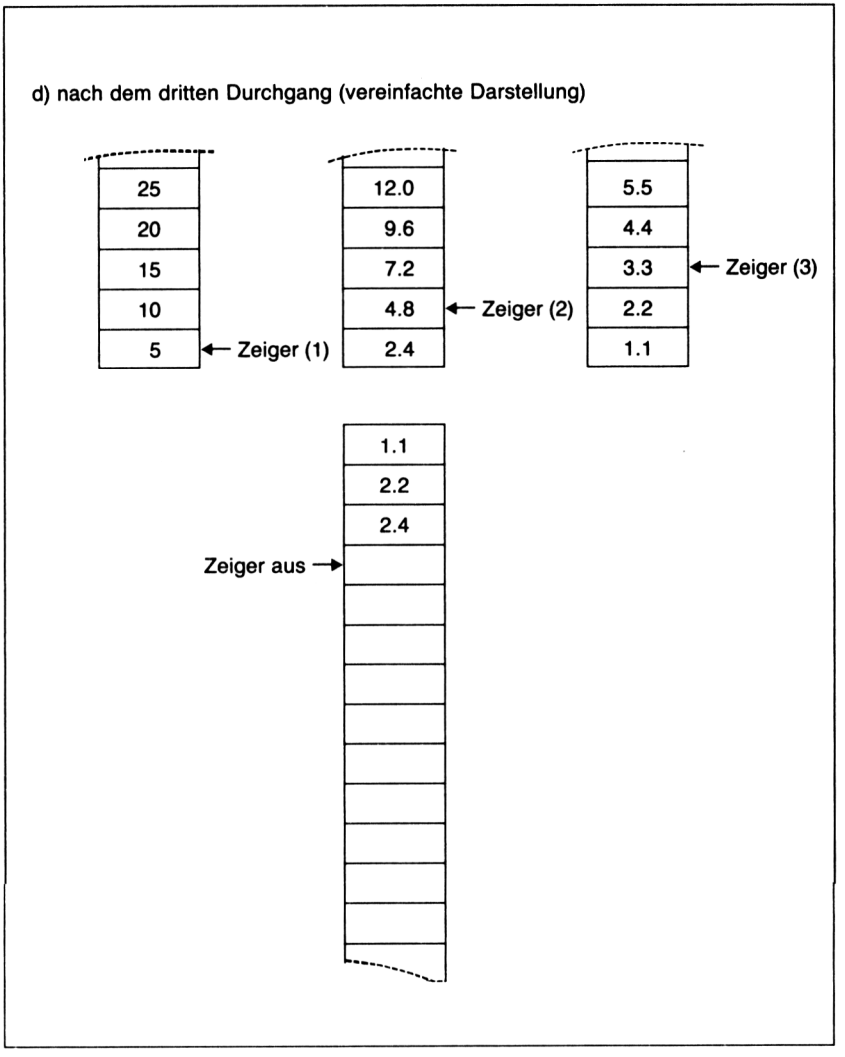


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm (Fortsetzung)

stückelt werden muß. Dieses Verfahren wird sofort wiederholt, bis alle einzulesenden Bänder verarbeitet sind.

Sind alle Eingabebänder abgearbeitet, so vertausche man die Ausgabe- mit den Eingabebändern. Auf den neuen Eingabebändern entstehen dann Läufe

größerer Länge. Mit jedem Tausch Ausgabebänder gegen Eingabebänder werden die Läufer größer und die Anzahl der Läufe geringer. Dies wird so lange fortgesetzt, bis nur noch ein Lauf existiert, der dann die komplett sortierten Datenfolgen beinhaltet. Abb. 1.27 veranschaulicht das Verfahren.

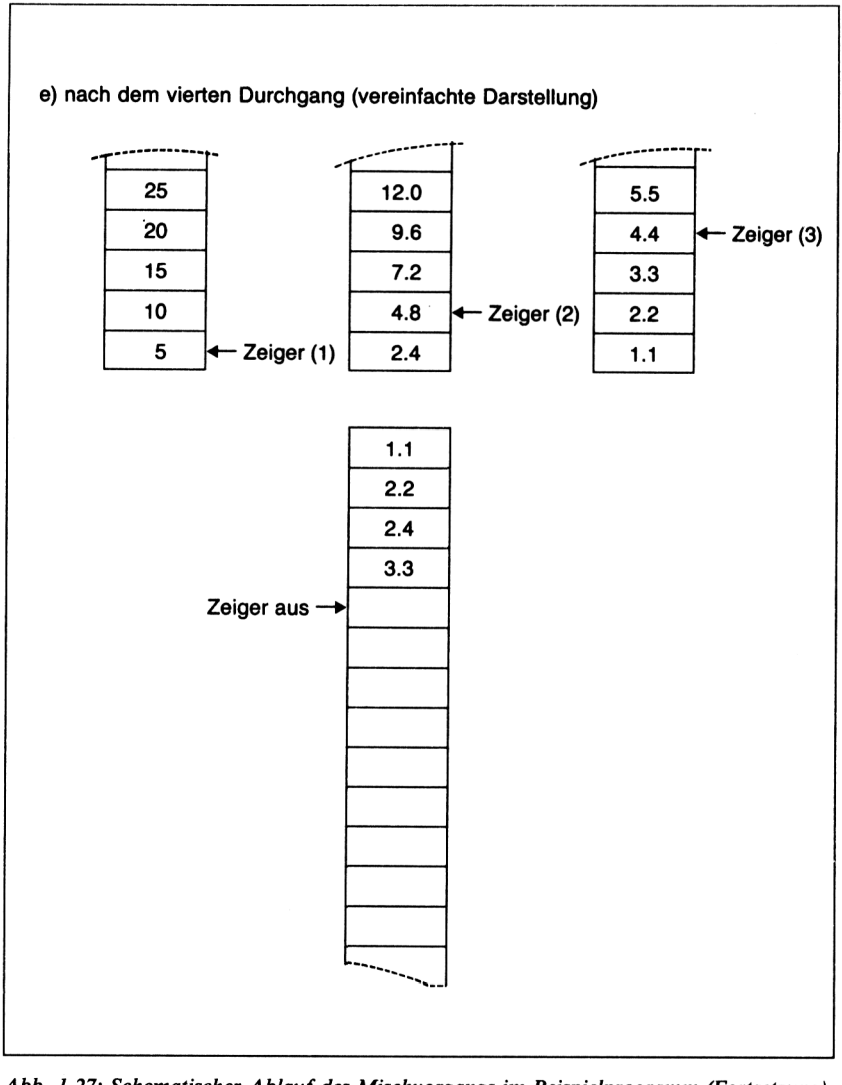


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm (Fortsetzung)

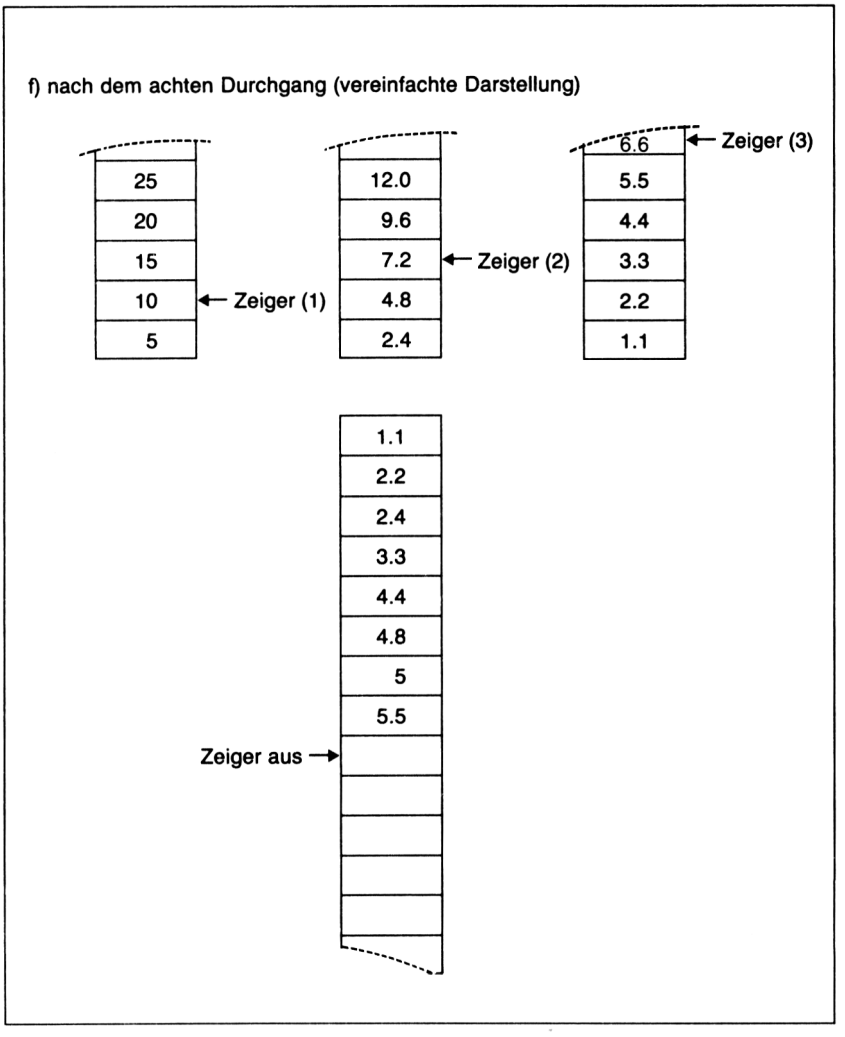


Abb. 1.27: Schematischer Ablauf des Mischvorgangs im Beispielprogramm (Fortsetzung)

Hier ist ein Beispiel-Programmstück für drei Eingabemedien und ein Ausgabemedium, wobei zur Vereinfachung drei Spalten einer Matrix herangezogen werden, die bereits durch die Vorgabe sortiert sind. Außerdem befindet sich in jeder Spalte nur ein Lauf. Die Abb. 1.27 macht in einer Serie von Zeichnungen den Ablauf deutlich.

Abb. 1.28: Mischprogramm

```
1000 REM ------------------------------
1010 REM ---      Mischen             ---
1020 REM ------------------------------
1030 :
1040 REM - Im folgenden werden die     -
1050 REM - Spalten der Matrix WERTE() -
1060 REM - gemischt. Der Vorgang ist  -
1070 REM - fuer Ihre Anwendung auf     -
1080 REM - Dateien (von Diskette und  -
1090 REM - Recorder, sowie Felder zu  -
1100 REM - uebertragen.                -
1110 :
1120 DIM WERTE(100,3)
1130 :
1140 FOR I=1 TO 50
1150    WERTE(I,1)=i*5
1160 NEXT
1170 WERTE(51,1)=-1
1180 :
1190 FOR I=1 TO 80
1200    WERTE(I,2)=i*2.4
1210 NEXT
1220 WERTE(81,2)=-1
1230 :
1240 FOR I=1 TO 99
1250    WERTE(I,3)=i*1.1
1260 NEXT
1270 WERTE(100,3)=-1
1280 :
1290 DIM AUSGABE(300)
1300 DIM AKTUELL(3)
1310 DIM LAUFENDE(3)
1320 DIM ZEIGER(3)
1330 :
1340 ZEIGER(1)=1
1350 ZEIGER(2)=1
1360 ZEIGER(3)=1
1370 :
1380 FOR I=1 TO 100
1390    PRINT I;
1400    FOR J=1 TO 3
1410       PRINT WERTE(I,J);
1420    NEXT
1430    PRINT
1440 NEXT
1450 :
2000 REM ------------------------------
2010 REM --- eigentl. Mischvorgang  ---
2020 REM ------------------------------
2030 :
2040 REM --- Vorbesetzen der akt.   ---
2050 REM --- Vergleichselemente     ---
2060 :
2070 FOR I=1 TO 3
2080    AKTUELL(I)=WERTE(ZEIGER(I),I)
2090 NEXT
2100 :
2110 ZEIGERAUS=0
2120 VERGLEICH=1000
2130 LAEUFE=0
2140 :
2150 REM ---       MISCHEN            ---
2160 :
2170 FOR I=1 TO 3
2180    IF LAUFENDE(I)>0 THEN GOTO 2240
2190    LAEUFE=1
2200    IF AKTUELL(I)<VERGLEICH
          THEN VERGLEICH=AKTUELL(I) :
               WO=I
2210 NEXT
2220 :
2230 ZEIGERAUS          = ZEIGERAUS+1
2240 AUSGABE(ZEIGERAUS) = VERGLEICH
2250 IF LAEUFE=0
       THEN END
```

```
2260 ZEIGER(WO)          = ZEIGER(WO)+1
2270 AKTUELL(WO)   = WERTE(ZEIGER(WO),WO)
2280 IF AKTUELL(WO)=-1 THEN LAUFENDE(WO)
=1
2290 PRINT"KONTROLLE:";VERGLEICH;WO
2300 PRINT,,ZEIGER(1);ZEIGER(2);
          ZEIGER(3)
2310 GOTO 2120
```

Abb. 1.28: Mischprogramm (Fortsetzung)

Programmbeschreibung

Wie wir bereits erwähnt haben, zeigen wir den Mischvorgang an drei Spalten des Feldes WERTE(,). Im Regelfall werden hier entweder drei Dateien auf einer Diskette geöffnet oder ein Feld, eine Diskettendatei und eine Datei vom Kassettenrecorder benutzt oder ähnliche Zusammenstellungen herangezogen.

Nach der Dimensionierung des Feldes WERTE(,) werden die einzelnen Spalten mit Zahlen besetzt (Zeilen ab 1140, 1190 und 1240). Durch die Vorgabe sind die Felder in sich schon sortiert, aber die Zahlen sind so gewählt, daß nicht fortlaufend jeweils ein Element herausgenommen wird, sondern wirklich abgeprüft werden muß, welches das jeweils kleinste Element ist. Sofern Ihre Daten noch nicht sortiert sind, ziehen Sie bitte eines der Sortier-Unterprogramme zu Hilfe.

Weiterhin wird in dem Beispiel deutlich, daß die einzelnen Spalten in verschiedener Länge (50, 80, 99) gültige Werte haben. Das Ende wird jeweils durch eine −1 markiert. Ab Zeile 1290 werden weitere Variablenfelder besetzt, deren Bedeutung Sie aus der Variablenübersicht entnehmen können. Ein Kontrollausdruck der nicht gemischten Daten erfolgt ab Zeile 1380.

Ab Zeile 2070 beginnt das eigentliche Mischen, indem zunächst die ersten Elemente der drei Spalten von WERTE(,) in das Feld AKTUELL( ) übernommen werden. Dann wird der Zeiger in das Ausgabefeld vorbesetzt, ein Vergleichswert hochgesetzt und der Merker für den Abschluß aller Läufe auf 0 gebracht.

Das Mischen besteht aus einer Schleife (Beginn in Zeile 2170), wo zunächst für jede Spalte (jedes Eingabeband) geprüft wird, ob dies schon zu Ende ist. Dann wird der Merker für das Ende aller Läufe umgesetzt, da mindestens noch ein Element zum Mischen anstand. Sofern dies kleiner als ein Vergleichselement ist (ähnlich zu MINSORT wird hier wieder das kleinste Element herausgesucht), wird es in VERGLEICH zwischengespeichert, und natürlich wird die Nummer des Eingabebandes (Spaltennummer des Feldes WERTE(,)) festgehalten.

Variablenübersicht für MISCHEN			1000-2310
Variablen			
Name	Art	Bereich	Bedeutung
AKTUELL( )	G	1. . .3/ Dezimalzahlen	aktuelle Werte der drei Eingabebänder
AUSGABE( )	G	1. . .300/ Dezimalzahlen	Ausgabeband
I	H	1. . .100	Laufvariable
J	H	1. . .3	Laufvariable
LAEUFE	H	0,1	Merker für Ende aller Läufe
LAUFENDE( )	H	1. . .3/0,1	Merker für Ende eines Eingabebandes aktuell auszugebendes Element
VERGLEICH	H	Dezimalzahl	zu mischende Daten
WERTE( )	G	1. . .300; 1. . .3/ Dezimalzahlen	
WO	H	1. . .3	Nummer des Eingabebandes für das aktuell auszugebende Element
ZEIGER( )	H	1. . .3/Integer	Zeiger auf aktuelle Elemente in den Eingabebändern
ZEIGERAUS	H	Integer	Zeiger auf aktuelles Element im Ausgabeband
Unterprogramme			
Zeile	Bedeutung		wird aufgerufen in Zeile
keine			
Programmaufteilung in Blöcke			
1. Zeile	Bedeutung		
1000 2000	Vorspann und Testrahmen Mischvorgang		

Abb. 1.29: Variablenübersicht für MISCHEN

Nachdem von den anstehenden Eingabebändern das kleinste Element gefunden ist, wird zunächst der Zeiger im Ausgabefeld erhöht, was beim sequentiellen Fortschreiben in eine Datei unterbleiben kann. Dann wird das gefundene Element in die Ausgabedatei geschrieben und schließlich abgefragt, ob alle Läufe beendet sind, womit das Programmende erreicht wäre.

Ist das Programmende nicht erreicht, so wird der Zeiger auch in die Spalte, aus der das zuletzt auf das Ausgabeband geschriebene Element stammt, um eins erhöht und der nächste Wert aus dieser Spalte eingelesen. Dann wird noch gefragt, ob eine – 1 vorliegt, womit das Laufende in der betreffenden Spalte gekennzeichnet ist. Bevor die drei aktuell anstehenden Elemente wieder abgefragt werden, wird noch eine Kontrollausgabe ausgeführt.

ETWAS ZUM THEMA KALENDERDATEN

Obwohl Kalenderdaten in der Regel als Zeichenketten gespeichert werden, kann man sie - neben Integer-, Fließkomma- und Zeichenketten-Variablen - als „vierten Variablentyp" bezeichnen, da sie in der Datenverwaltung sehr häufig vorkommen und eine spezielle Handhabung erfordern. Im folgenden sollen Kalenderdaten immer in der Form „TT.MM.JJ" vorliegen, wobei auch Tage und Monate zwischen eins und neun mit einer führenden Null versehen werden. Um nun Kalenderdaten ordnungsgemäß verarbeiten zu können, ist ihre Plausibilität bereits bei der Erfassung zweckmäßigerweise zu überprüfen. Da man das Eingabejahr nur im Einzelfall prüfen kann, wollen wir uns im folgenden auf die Prüfung von Tag und Monat beschränken.

Plausibilitätsprüfung

Die Prüfung auf einen gültigen Monat ist relativ einfach, da hier nur Zahlen zwischen 1 und 12 vorkommen können. Etwas schwieriger wird es bei den Tagen, da hier die Werte zwischen 29 und 31 nur bei verschiedenen Monaten (gegebenenfalls unter Berücksichtigung eines Schaltjahres) möglich sind. Bei der Berechnung des Schaltjahres wollen wir uns auf die übliche Regel (die letzten beiden Ziffern des Jahres sind durch vier teilbar) beschränken, ohne die Sonderregelung für volle Jahrhunderte (kein Schalttag, außer wenn Jahrhundert durch vier teilbar) zu berücksichtigen. Zur besseren Verarbeitung wird das Kalenderdatum zunächst in seine verschiedenen Teile aufgespalten.

Programmbeschreibung

Im Testrahmen wird zunächst die maximale Anzahl von Tagen in jedem Monat in dem Feld MAXMONAT( ) festgehalten. Dann wird das Unterprogramm zur Plausibilitätsprüfung nach einer entsprechenden Eingabe durch den Anwender aufgerufen und anschließend die Fehlermeldung ausgewertet. Das Programm brechen Sie ab, indem Sie zweimal die ESC-Taste drücken.

```
1000 REM ------------------------------
1010 REM ---      Testrahmen          ---
1020 REM ------------------------------
1030 :
1040 DATA 31,28,31,30,31,30,31,31,30,
       31,30,31
1050 DIM MAXMONAT(12)
1060 :
1070 FOR I=1 TO 12
1080    READ MAXMONAT(I)
1090 NEXT
1100 :
1110 INPUT"DATUM";DA$
1120 GOSUB 2000
1130 :
1140 IF RICHTIG=1
       THEN PRINT"RICHTIG" : GOTO 1110
1150 IF RICHTIG=0
       THEN PRINT"Fehler bei Punkten":.
            GOTO 1110
1160 IF RICHTIG=-1
       THEN PRINT"falsches Datum" :
            GOTO 1110
1170 IF RICHTIG=-2
       THEN PRINT"falsche Laenge" :
            GOTO 1110
1180 :
2000 REM ------------------------------
2010 REM --- Plausibilitaetspruef.  ---
2020 REM ------------------------------
2030 :
2040 REM ---     auf Laenge          ---
2050 :
2060    IF LEN(DA$) <> 8
          THEN RICHTIG=-2 : RETURN
2070 :
2080 REM ---     auf Punkte          ---
2090 :
2100    RICHTIG=0
2110 :
2120    SCHALTJAHR=0
2130 :
2140    IF MID$(DA$,3,1) <> "."
          THEN RETURN
2150    IF MID$(DA$,6,1) <> "."
          THEN RETURN
2160 :
2170 REM ---     auf Tage            ---
2180 :
2190    DAY   = VAL(LEFT$(DA$,2))
2200    MONAT = VAL(MID$(DA$,4,2))
2210    JAHR  = VAL(RIGHT$(DA$,2))
2220 :
2230    RICHTIG=-1
2240 :
2250    IF JAHR/4 - INT(JAHR/4) = 0
          THEN SCHALTJAHR=1
2260    IF MONAT=2 AND DAY=29 AND
           SCHALTJAHR=1
          THEN RICHTIG=1 : RETURN
2270    IF MONAT=2 AND DAY=29 AND
           SCHALTJAHR=0
          THEN RETURN
2280 :
2290    IF MONAT < 1 OR MONAT > 12
          THEN RETURN
2300    IF DAY<1 OR DAY>MAXMONAT(MONAT)
          THEN RETURN
2310 :
2320    RICHTIG=1
2330 RETURN
```

Abb. 1.30: Kalenderdaten - Plausibilitätsprüfung

Bei der Plausibilitätsprüfung wird zunächst einmal auf die Länge des Feldes hin überprüft. Ist diese verschieden von 8, so ist die Länge nicht korrekt, und das Unterprogramm wird sofort verlassen. Vielleicht noch etwas zur Programmphilosophie: Die häufigsten Fehler werden vornangestellt, um möglichst kurze Rechenzeiten zu erzielen.

Variablenübersicht für DATPLAUS 1000-2330
Zweck: Plausibilitätsprüfung des Datums

Variablen

Name	Art	Bereich	Bedeutung
DA$	G	Kalenderdatum	eingegebenes Datum
DAY	H	1. . .31	aktuelle Tageszahl
I	H	1. . .12	Laufvariable
JAHR	H	00. . .99	aktuelle Jahreszahl
MAXMONAT( )	G	1. . .12/28. . .31	maximale Tageszahlen der verschiedenen Monate
MONAT	H	1. . .12	aktuelle Monatszahl
RICHTIG	A	−2. . .1	Merker für Art des Fehlers: −2: DA$ nicht 8 Zeichen lang −1: Datum nicht gültig 0: Punkte falsch gesetzt 1: o. k.
SCHALTJAHR	H	0,1	Merker, ob Schaltjahr (1 = ja)

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
2000	Plausibilitätsprüfung	1120

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Testrahmen und Auswertung
2000	UP Prüfung
2040	- auf Länge (= 8)
2080	- Punkte an 3. und 6. Stelle
2170	- korrekte Tageszahl/Monatszahl

Abb. 1.31: Variablenübersicht für DATPLAUS

Der nächsthäufige Fehler ist eine falsche Position der Punkte innerhalb des Datums, was in den Zeilen 2140 und 2150 abgefragt wird. Zuvor wird jedoch

noch die Variable RICHTIG auf 0 gesetzt, um dies in den Zeilen 2140/2150 zu sparen. Ebenso wird die Variable SCHALTJAHR auf 0 gesetzt, da sie eventuell von einer vorgehenden Eingabe noch auf 1 stehen könnte.

Als letztes wird die Korrektheit der Tage und Monate geprüft, wobei eine Variable TAG nicht möglich ist, da ein BASIC-Befehl gleichen Namens existiert. Hier wurde auf das englische DAY ausgewichen. Auch hier wird die Variable RICHTIG wieder auf die entsprechende Fehlernummer gesetzt, die beim Abbruch im folgenden gültig ist. In Zeile 2250 wird geprüft, ob ein Schaltjahr vorliegt. Entsprechend wird die Variable SCHALTJAHR auf 1 gesetzt, und in den nächsten beiden Zeilen wird geprüft, ob der 29. Februar eingegeben wurde. In Abhängigkeit von der Variablen SCHALTJAHR wird das Unterprogramm mit der entsprechenden Fehlermeldung verlassen. In Zeile 2290 erfolgt die Plausibilitätsprüfung auf den Monat und in Zeile 2300 auf alle anderen Tage außer dem Schalttag. Wurde das Unterprogramm nicht verlassen, so ist das Datum bei Erreichen der Zeile 2320 korrekt, und die Variable RICHTIG wird entsprechend gesetzt.

Datum platzsparend speichern

Bei Mikrocomputern ist in der Regel der externe Speicherplatz (Disketten) ein großes Problem. Wenn viele Kalenderdaten gespeichert werden müssen, so kann man mit einem kleinen Trick den Platzbedarf pro Datum auf ca. 37% verringern. Der Rechner speichert sich alle Zeichen im sogenannten ASCII-Code, d. h. als Binärzahl. Ein a wird z. B. als 65 (01000001) gespeichert. Da auch Steuerzeichen im ASCII-Code gespeichert werden (z. B. Bildschirm löschen), ist hier jedoch Vorsicht geboten. Wir wollen den Buchstabenbereich 65 (a) bis 90 (z) und die angrenzenden Sonderzeichen benutzen, was auch vollkommen ausreicht.

Gegeben sei ein Datum der Form TT.MM.JJ in der Variablen DA$.

T = VAL(MID$(DA$,1,2)) - Tageszahl
M = VAL(MID$(DA$,4,2)) - Monatszahl
J = VAL(MID$(DA$,7,2)) - Jahreszahl (ohne 19)

T kann einen Wert zwischen 1 und 31 annehmen. Wenn wir nun 65 zu T addieren, liegt T im gewählten Bereich. M kann einen Wert zwischen 1 und 12 annehmen, auch hier addieren wir 65. J kann zwar alle Werte zwischen 0 und 99 annehmen, jedoch kann man davon ausgehen, daß J im Bereich 70 bis 95 liegt.

Das gekürzte Datum wollen wir in DK$ ablegen und erhalten hierfür:

DK$ = CHR$(T + 65) + CHR$(M + 65) + CHR$(J)

wobei die Funktion CHR$(X) zu einer Zahl das zugehörige ASCII-Zeichen ergibt.

Beispiel: 15.06.85 wird gespeichert als PGU mit

ASCII-Code von P = 80 (65 + 15)
ASCII-Code von G = 71 (65 + 6)
ASCII-Code von U = 85

Entschlüsselt wird das Ganze mit:

T = ASC(LEFT$(DK$,1)) - 65
M = ASC(MID$(DK$,2,1)) - 65
J = ASC(RIGHT$(DK$,1))

DA$ = RIGHT$(STR$(T),2) + '.' + RIGHT$(STR$(M),2) + '.'
+ RIGHT$(STR$(J),2)

In einer Auftragsbearbeitung, in der Bestelldatum, Lieferscheindatum, Rechnungsdatum und Datum des Zahlungseingangs gespeichert werden, reduziert sich der Platzbedarf von 32 Byte je Auftrag auf 12 Byte je Auftrag. Bei 1000 zu speichernden Aufträgen sind dies 20 KByte.

KLEINERE HILFSROUTINEN

Im folgenden wollen wir Ihnen zwei kurze Routinen vorstellen, die hauptsächlich im kommerziellen Bereich immer wieder benötigt werden.

Führende Nullen anfügen

Besonders bei einer Auftragsbearbeitung, aber auch in anderen Fällen ist es wichtig, bei Zahlen führende Nullen anzufügen, ohne daß zwischen den Nullen und der Zahl ein Zwischenraum - der durch das positive Vorzeichen entsteht - oder ein Minus-Zeichen dazwischenliegt. Folgendes kleine Programm erledigt das für Sie:

```
100 A=2589
110 N=7
120 HILFS=STR$(A)
130 HILFS=RIGHT$(HILFS,LEN(HILFS)-1)
140 AUS$=RIGHT$("0000000"+HILFS,N)
150 PRINT AUS$
199 END
```

Abb. 1.32: Anfügen führender Nullen

Das Programmstück macht die Vorgehensweise deutlich. Zunächst wird die gewünschte Zahl in eine Zeichenkette umgewandelt, und anschließend wird das erste Zeichen abgeschnitten, indem die Zeichen von rechts bis auf das erste Zeichen in HILF$ abgelegt werden. Dann wird in Zeile 140 an die Variable HILF$ noch eine entsprechende Anzahl von Nullen vorne angefügt, und schließlich wird die Zahl auf die gewünschte Länge (N) abgeschnitten.

Man hätte das Ganze noch ausführlicher gestalten können, jedoch ist die Darstellung übersichtlich genug. Kürzer geht es jedoch mit folgender Zeile:

```
900 AUS$=RIGHT$("0000000"+RIGHT$(STR$(A),LEN(STR$(A))-1),N) : PRINT AU
S$
```

Abb. 1.33: Anfügen führender Nullen in einer Programmzeile

Hier sind alle Arbeitsschritte in einer einzigen Zeile zusammengefaßt. Sie können sich die Gleichheit der Ergebnisse veranschaulichen, indem Sie zunächst das Programm mit RUN starten und anschließend GOTO 900 eingeben.

Eliminieren anhängender Leerzeichen

Ein weiteres Problem ist das Eliminieren anhängender Leerzeichen in einer Zeichenkette. Normalerweise haben Datensatzelemente in Datensätzen eine feste Länge, so daß z. B. bei einer vorgegebenen Länge von 25 Zeichen der Name „Huber" mit 20 Leerzeichen aufgefüllt wird. Für die Routine aus Abb. 1.34 gibt es jedoch auch noch andere Anwendungsbeispiele.

```
100 NAME$="HUBER          "
110 VORNAME$="PETER             "
120 REM --- Ausgabe ohne Elimination
130 PRINT NAME$;", ";VORNAME$
140 REM --- ELIMINIEREN
150 X$=NAME$ : GOSUB 200
160 PRINT X$;", ";
170 X$=VORNAME$ : GOSUB 200
180 PRINT X$
190 END
200 IF RIGHT$(X$,1)=" "
      THEN X$=LEFT$(X$,LEN(X$)-1) :
           GOTO 200
210 RETURN
```

Abb. 1.34: Eliminieren von anhängenden Leerzeichen

Die wichtigste Zeile ist die Nummer 200, in der so lange von rechts her ein Zeichen weggenommen wird, wie dort noch Leerzeichen stehen. Das Absuchen mittels des Befehls INSTR auf das erste Vorkommen eines Leerzeichens

ist nicht sinnvoll, wenn Sie z. B. Doppelnamen verwenden, aber auch in anderen Fällen können Leerzeichen inmitten des Textes auftauchen, so daß bei der Vorgehensweise mit INSTR wichtige Teile der Daten abgeschnitten würden.

WICHTIGE BASIC-BEFEHLE

Obwohl die BASIC-Befehle ausführlich im Handbuch besprochen sind, wollen wir noch etwas näher auf einige wichtige Befehle eingehen, besonders auf das Zusammenspiel zwischen verschiedenen Befehlen.

PRINT USING

Besonders im kaufmännischen Bereich ist der Befehl PRINT USING wichtig, da er die kaufmännische Zahlendarstellung (wenn auch nicht ganz) unterstützt. Leider bleibt es bei der englischen Schreibweise, d. h. Dezimalpunkt und Tausenderkomma gegenüber der deutschen Schreibweise Dezimalkomma und Tausenderpunkt.

Abb. 1.35: Beispiele für PRINT USING

```
1000 REM ------------------------------
1010 REM ---      Beispiele fuer      ---
1020 REM ---       PRINT USING        ---
1030 REM ------------------------------
1040 RANDOMIZE 464
1050 DIM ZAHLEN(100)
1060 :
1070 FOR I=1 TO 100
1080     ZAHLEN(I)=INT(1000000*RND(TIME))/100
1090     IF RND(45) < 0.5 THEN ZAHLEN(I)=ZAHLEN(I)*-1
1100 NEXT
1110 :
1120 REM ---  TABELLENAUSGABE         ---
1130 :
1140 FOR I=1 TO 90 STEP 3
1150     PRINT USING "####.##";ZAHLEN(I);
1160     PRINT"      ";
1170     PRINT USING "####.##";ZAHLEN(I+1);
1180     PRINT"      ";
1190     PRINT USING "####.##";ZAHLEN(I+2)
1200 NEXT
1210 A$=INKEY$ : IF A$="" THEN 1210
1220 :
1230 REM ---  TABELLENAUSGABE         ---
1240 REM ---  VORZEICHEN HINTEN       ---
1250 :
1260 FOR I=1 TO 90 STEP 3
1270     PRINT USING "####.##-";ZAHLEN(I);
1280     PRINT USING "####.##-";ZAHLEN(I+1);
1290     PRINT USING "####.##-";ZAHLEN(I+2)
1300 NEXT
1310 :
1320 A$=INKEY$ : IF A$="" THEN 1320
1330 :
1340 REM ---  TABELLENAUSGABE         ---
1350 REM ---  MIT '*' VORAUS          ---
1360 :
```

```
1370 FOR I=1 TO 90 STEP 3
1380    PRINT USING "***####.##-";ZAHLEN(I);
1390    PRINT USING "***####.##-";ZAHLEN(I+1);
1400    PRINT USING "***####.##-";ZAHLEN(I+2)
1410 NEXT
1420 :
1430 A$=INKEY$ : IF A$="" THEN 1430
1440 :
1450 REM ---  TABELLENAUSGABE        ---
1460 REM ---  MIT '$' VORAUS         ---
1470 :
1480 FOR I=1 TO 90 STEP 3
1490    PRINT USING "$$####.##-";ZAHLEN(I);
1500    PRINT USING "$$####.##-";ZAHLEN(I+1);
1510    PRINT USING "$$####.##-";ZAHLEN(I+2)
1520 NEXT
1530 :
1540 A$=INKEY$ : IF A$="" THEN 1540
1550 :
1560 REM ---  TABELLENAUSGABE        ---
1570 REM ---  MIT '$' UND '*' VORAUS---
1580 :
1590 FOR I=1 TO 90 STEP 3
1600    PRINT USING "**$####.##-";ZAHLEN(I);
1610    PRINT USING "**$####.##-";ZAHLEN(I+1);
1620    PRINT USING "**$####.##-";ZAHLEN(I+2)
1630 NEXT
1640 :
1650 A$=INKEY$ : IF A$="" THEN 1650
1660 :
1670 REM ---  TABELLENAUSGABE        ---
1680 REM ---  MIT '$', '*' UND ','   ---
1690 :
1700 FOR I=1 TO 90 STEP 3
1710    PRINT USING "**$#,###.##-";ZAHLEN(I);
1720    PRINT USING "**$#,###.##-";ZAHLEN(I+1);
1730    PRINT USING "**$#,###.##-";ZAHLEN(I+2)
1740 NEXT
1750 :
1760 A$=INKEY$ : IF A$="" THEN 1760
1770 :
1780 REM ------------------------------
1790 REM --- EINZELNE AUSGABE        ---
1800 REM ------------------------------
1810 PRINT USING "#,###.##-";ZAHLEN(1);ZAHLEN(2);ZAHLEN
(3)
1820 PRINT
1830 PRINT USING "###,###.##-";ZAHLEN(1);ZAHLEN(2);ZAHL
EN(3)
1840 PRINT
1850 PRINT USING "###,###.##^^^^";ZAHLEN(4);ZAHLEN(5)
1860 PRINT
1870 PRINT USING "######.##^^^^";ZAHLEN(6);ZAHLEN(7)
1880 PRINT
1890 PRINT USING "#.##^^^^";ZAHLEN(8);ZAHLEN(9)
1900 PRINT
1910 GOTO 1930
1920 PRINT USING "#.##^^^^";ZAHLEN(8);"      ";ZAHLEN(9)

1930 PRINT
1940 PRINT USING "#.##^^^^";ZAHLEN(10); : PRINT"     ";
: PRINT USING "#.##^^^^";ZAHLEN(11)
1950 PRINT
1960 PRINT USING "##.##^^^^";ZAHLEN(10); : PRINT"     ";
 : PRINT USING "##.##^^^^";ZAHLEN(11)
```

Abb. 1.35: Beispiele für PRINT USING (Fortsetzung)

Zunächst wird im Listing aus Abb. 1.35 ein Feld ZAHLEN( ) definiert und mit Werten zwischen −9999.99 und +9999.99 gefüllt.

Ab Zeile 1150 wird eine ganz normale Tabelle mit unseren Zufallszahlen ausgegeben. Man sieht an der Ausgabe, daß Werte größer als 1000 im negativen Bereich mit einem vorangestellten % ausgegeben sind. Dies weist darauf hin, daß für die Ausgabe nicht genügend Platz zur Verfügung stand, obwohl die Zahlen richtig ausgegeben sind. Achten Sie also bei der Darstellung von Tabellen auf eine genügend große Anzahl von #-Zeichen im PRINT USING-Befehl, da sonst Ihre schöne Tabelle zerstört wird. Die Leerzeichen in den Zeilen 1160 und 1180 haben wir im nächsten Bereich ab Zeile 1210 ausgelassen, so daß die Zahlen direkt hintereinander ausgegeben werden. Die tabellarische Form ist immer noch gegeben, wobei hier das Minuszeichen hinten angebracht ist.

Ab Zeile 1380 werden vor der Zahl Sterne ausgegeben. Die Angabe im Handbuch mit „maximal 2 Zeichen" vor der Zahl ist etwas mißverständlich, da auch bei dreistelligen Zahlen drei Sterne ausgegeben werden. Als nächstes wird der Stern durch ein $-Zeichen ersetzt, wobei jedoch nicht alle führenden Stellen mit einem $ aufgefüllt werden, sondern nur die letzte Stelle vor der Zahl. Alle anderen Positionen werden mit einem Leerzeichen besetzt. Bei der Ausgabe von * und $ wird wieder ein Zeichen vorne mehr ausgegeben, allerdings keine Leerzeichen mehr, sondern Sterne.

Das nächste Beispiel (ab Zeile 1710) zeigt Ihnen die Verwendung des Tausenderkommas in Verbindung mit * und $. Sie sehen daraus, daß die Möglichkeiten mit PRINT USING beliebig kombinierbar sind.

Kommen wir nun noch zu einigen einzelnen Beispielen. Zunächst die normale Ausgabe mit Tausenderkomma und Dezimalpunkt und hintenangesetztem Vorzeichen in Zeile 1810. Größere Abstände erhalten Sie, wenn Sie mehr Ziffernzeichen vor die Zahl stellen, auch wenn diese nicht benötigt werden. Damit erübrigt sich das Einfügen von Leerzeichen bei der Ausgabe, was einige Probleme macht, wie wir weiter unten noch sehen werden. An den Zeilen ab 1810 sehen Sie auch, daß mit einer Angabe hinter PRINT USING mehrere Zahlen ausgeben werden können. Die Ausgabe in Zeile 1850 zeigt die Möglichkeit der Exponentialdarstellung auf, wenn auch sehr unübersichtlich, was in Zeile 1870 auch nicht viel besser wird. Bei der Verwendung einer Exponentialdarstellung sollten Sie die Form, wie sie in Zeile 1890 dargestellt ist, heranziehen, da Zahlen bei Exponentialdarstellung üblicherweise nur mit einer Vorkommastelle abgebildet werden. Die Zeile 1920 wird absichtlich übersprungen, da Sie hier auf Type Mismatch-Fehler laufen würden: Es ist nicht gestattet, hinter einer PRINT USING-Angabe für Zahlen Zeichenketten einzufügen. Sofern Sie dieses wünschen, müssen Sie die Darstellungen aus Zeile 1940 oder 1960 übernehmen.

WINDOW, CLS, CLG, ORIGIN

In diesem Abschnitt wollen wir uns nun mit der Fenstertechnik und damit zusammenhängenden Problemen beschäftigen.

Abb. 1.36: Beispiele zur Fenstertechnik

```
1000 REM ------------------------------
1010 REM ---  WINDOW, CLS, CLG, ... ---
1020 REM ------------------------------
1030 MODE 1
1040 WINDOW #1,3,19,3,10
1050 PAPER #1,1
1060 CLS #1
1070 :
1080 WINDOW #2,21,37,3,10
1090 PAPER #2,2
1100 CLS #2
1110 :
1120 WINDOW #3,3,19,15,22
1130 PAPER #3,3
1140 CLS #3
1150 :
1160 WINDOW #4,21,37,15,22
1170 PAPER #4,4
1180 CLS #4
1190 :
1200 BORDER 0
1210 :
1220 A$=INKEY$ : IF A$="" THEN 1220
1230 :
1240 PAPER #4,1
1250 INK 1,7,11
1260 :
1270 A$=INKEY$ : IF A$="" THEN 1270
1280 :
1290 SPEED INK 250,50
1300 CLS #4
1310 :
1320 FOR I=1 TO 4
1330    FOR J=0 TO 3
1340       PEN #I,J
1350       PRINT #I,"Fenster";i;" Pen";j
1360    NEXT
1370 NEXT
1380 :
1390 A$=INKEY$ : IF A$="" THEN 1390
1400 :
1410 WINDOW SWAP 1,2
1420 CLS #1
1430 A$=INKEY$ : IF A$="" THEN 1430
1440 CLS #2
1450 PRINT #1,"Diese Ausgabe"
1460 PRINT #1,"erfolgt in "
1470 PRINT #1,"Fenster 1, wird"
1480 PRINT #1,"aber in Fenster"
1490 PRINT #1,"2 dargestellt."
1500 :
1510 INK 1,8
1520 PRINT #2,"Hier wurde Fen-"
1530 PRINT #2,"ster 2 ange-"
1540 PRINT #2,"sprochen"
1550 :
1560 A$=INKEY$ : IF A$="" THEN 1560
1570 :
1580 WINDOW #5,15,25,6,19
1590 PAPER #5,2
1600 PEN #5,1
1610 CLS #5
1620 PRINT #5," Fenster    werden"
1630 PRINT #5," ueber-     schrieben"
1640 :
```

```
1650 A$=INKEY$ : IF A$="" THEN 1650
1660 :
1670 PAPER 3
1680 CLS
1690 LOCATE 3,24
1700 PRINT"CLS loescht deshalb auch die Fenster
1710 :
1720 A$=INKEY$ : IF A$="" THEN 1720
1730 :
1740 CLS #1
1750 CLS #2
1760 CLS #5
1770 PAPER #3,0
1780 CLS #3
1790 CLS #4
1800 PRINT #2,"auf die Reihen-  folge kommt es an
1810 A$=INKEY$ : IF A$="" THEN 1810
1820 PRINT #2,"AAAAAAAAAAAAAAAA"
1830 PRINT #2,"AAAAAAAAAAAAAAAA"
1840 A$=INKEY$ : IF A$="" THEN 1840
1850 PRINT #2,"AAAAAAAAAAAAAAAA"
1860 PRINT #2,"AAAAAAAAAAAAAAAA"
1870 :
1880 ORIGIN 100,100,50,550,50,150
1890 CLS
1900 PAPER 2
1910 CLG
1920 TAG
1930 PRINT"Grafik-Fenster"
1940 MOVE 0,0
1950 DRAW 600,200,1
1960 MOVE 10,20
1970 PRINT"Grafik-Fenster";
1980 A$=INKEY$ : IF A$="" THEN 1980
```

Abb. 1.36: Beispiele zur Fenstertechnik (Fortsetzung)

Zunächst definieren wir vier Fenster, die neben- und übereinander angeordnet sind. Zu jedem Fenster wird der Hintergrund (PAPER) mit der gleichlautenden Farbauswahlnummer (INK) besetzt. Indem wir dann den Bildschirm im Bereich dieses Fensters löschen, wird der Hintergrund automatisch mit der entsprechenden Farbe eingefärbt.

Nach Starten des Programms sehen wir, daß am Bildschirm das Fenster #4 keine andere Farbe enthält. Dies liegt jedoch nicht daran, daß hier gleiche Farben vorliegen, sondern daß die Farbauswahlnummer 4 im Modus 1 nicht verwendet werden kann. Wir schalten also die Farbauswahlnummer 1 auf das Fenster #4. In Zeile 1250 wird das Fenster #1 auf Blinken zwischen den Farben Purpur und Himmelblau geschaltet. Es ist ein gleichmäßiges, relativ schnelles Blinken. Das Blinken würde jetzt auch einen Schreibstift mit der zugeordneten Farbauswahlnummer 1 betreffen. In Zeile 1290 wird die Geschwindigkeit des Blinkens umgestellt, wobei die Purpur-Phase jetzt fünfmal so lange dauert wie die Himmelblau-Phase. Dann werden in den beiden ineinandergeschachtelten Schleifen ab Zeile 1320 alle Kombinationsmöglichkeiten von Schreibstift- und Hintergrund-Farbe durchgespielt.

In Zeile 1410 werden die Zuordnungen der Fenster #1 und #2 vertauscht, d. h. daß jetzt alle Ausgaben auf Fenster #2 im BASIC-Befehl tatsächlich in

Fenster #1 erscheinen und umgekehrt. Wie die beiden CLS-Befehle zur Prüfung deutlich machen, bleibt jedoch die Farbzuordnung erhalten. Beachten Sie bitte, daß in allen Fenstern der Schreibstift jetzt die Farbauswahlnummer #3 zugewiesen bekommen hat. Im weiteren noch eine kurze Ausgabe, die die Möglichkeiten des WINDOW SWAP deutlich macht. Ab Zeile 1580 wird ein neues Fenster mit der Nummer #5 definiert, das allen anderen Fenstern teilweise überlagert ist. Hier wird deutlich, daß das zuletzt geöffnete und angesprochene Fenster alle anderen Fenster überschreibt, wenn eine Ausgabe in diesem Bereich vorgenommen wird.

Der gesamte Bildschirm ist auch ein Fenster (#0), was man nicht vergessen sollte. Die Zeilen ab 1670 machen dies deutlich. Ab Zeile 1740 werden die Fenster durch die CLS-Befehle auf dem Bildschirm wieder sichtbar gemacht, und ein erneutes Überschreiben erfolgt.

Auch ein Grafik-Fenster ist vorhanden, das mit dem ORIGIN-Befehl gesetzt werden kann. Durch dieses gesetzte Fenster werden zwar Koordinaten - man sieht es an der Linie - in den Befehlen noch angenommen, ohne daß ein Fehler erzeugt wird, aber Koordinaten außerhalb des Fensters werden nicht mehr auf dem Bildschirm ausgegeben. Mit dem TAG-Befehl lassen sich Zeichen des normalen Zeichensatzes auch außerhalb ihrer normalen Spalten und Zeilen positionieren und ausgeben. Dies wird in Zeile 1930 deutlich (untere Ausgabe am Bildschirm). Hinter der Ausgabe „Grafik-Fenster" erscheint ein geknickter Pfeil nach links und ein Pfeil nach unten. Wie wir in Kapitel 2 näher erläutern werden, sind dies Zeichen des Zeichensatzes aus dem Bereich 0 bis 31, genauer die Zeichennummer 13 und 10. Der abknickende Pfeil nach links dürfte Ihnen vielleicht von anderen Computern oder einem Fernschreiber bekannt sein, er bedeutet in der Regel - abgebildet auf einer Taste - einen Wagenrücklauf (Carriage Return). Mit CHR$(13) lösen Sie ja normalerweise beim CPC auch ein Carriage Return aus. Der Pfeil nach unten ist ein Line Feed (LF), das Sie auch mit CHR$(10) erzeugen können. Wie Sie an diese Zeichendarstellung gelangen, wird in Kapitel 2 näher erläutert.

Wenn einer Zeichenausgabe mittels PRINT nach dem TAG-Befehl ein Semikolon folgt, so werden diese Sonderzeichen unterdrückt, was durch die Ausgabe in Zeile 1970 deutlich gemacht wird. Aber hier sehen wir auch noch etwas anderes: Wenn Buchstaben in eine Grafik eingetragen werden, wird die Grafik in der gesamten 8×8-Matrix des Textzeichens überschrieben. Die vorher gezeichnete Linie ist nun nicht mehr durchgehend.

INT, CINT, FIX, ROUND, UNT

Die letzten BASIC-Befehle, die wir hier besprechen wollen, dienen der Zahlendarstellung und -umwandlung.

```
1000 REM ------------------------------
1010 REM --- Zahlenumwandlung/Runden---
1020 REM ------------------------------
1030 :
1040 FOR I=1 TO 10
1050    A=100*RND(TIME)
1060    PRINT A,INT(A)
1070 NEXT
1080 :
1090 A$=INKEY$ : IF A$="" THEN 1090
1100 PRINT"----------------------------
1110 :
1120 FOR I=1 TO 10
1130    A=100*RND(TIME)
1140    PRINT A,CINT(A)
1150 NEXT
1160 :
1170 A$=INKEY$ : IF A$="" THEN 1170
1180 PRINT"----------------------------
1190 :
1200 A=1258.4569
1210 PRINT A,CINT(A*100)/100
1220 :
1230 A$=INKEY$ : IF A$="" THEN 1230
1240 PRINT"----------------------------
1250 :
1260 PRINT A,INT(A*100+0.5)/100
1270 A=123.25896
1280 PRINT A,CINT(A*100)/100
1290 :
1300 A$=INKEY$ : IF A$="" THEN 1300
1310 PRINT"----------------------------
1320 :
1330 FOR I=1 TO 10
1340    A=100*RND(TIME)
1350    PRINT A,FIX(A)
1360 NEXT
1370 :
1380 A$=INKEY$ : IF A$="" THEN 1380
1390 PRINT"----------------------------
1400 :
1410 FOR I=1 TO 10
1420    A=-100*RND(TIME)
1430    PRINT A,FIX(A)
1440    PRINT A,INT(A)
1450 NEXT
1460 :
1470 A$=INKEY$ : IF A$="" THEN 1470
1480 PRINT"----------------------------
1490 :
1500 FOR I=1 TO 10
1510    A=-100*RND(TIME)
1520    PRINT A,ROUND(A);ABS(A);ROUND(AB
S(A),2)
1530 NEXT
1540 :
1550 A$=INKEY$ : IF A$="" THEN 1550
1560 PRINT"----------------------------
1570 :
1580 PRINT UNT(20000)
1590 PRINT UNT(-20000)
1600 PRINT UNT(-1),UNT(65535)
1610 PRINT UNT(40000)
1620 PRINT UNT(-40000)
```

Abb. 1.37: Zahlenumwandlungen

Es werden jeweils zehn Zahlen ausgegeben, um die Befehle auch für unterschiedliche Zahlendarstellungen deutlich zu machen. Der Einfachheit halber werden Zufallszahlen herangezogen.

Der INTEGER-Befehl (INT) bewirkt eine Ausgabe der nächstniedrigeren ganzen Zahl, d. h. es wird keine Rundung vorgenommen. Für positive Zahlen entspricht dies dem Abschneiden der Dezimalstellen, bei negativen Zahlen kann man sich dies veranschaulichen, indem die Dezimalstellen abgeschnitten werden und noch eine 1 subtrahiert wird.

Das korrekte Runden übernimmt der CINT-Befehl. Wie bei der Ausgabe ab Zeile 1200 deutlich wird, ist dies jedoch nur für Zahlen im Bereich −32768 bis +32767 möglich. Starten Sie das Programm zur Fortführung mit GOTO 1260. Zeile 1260 zeigt Ihnen eine andere Möglichkeit für eine korrekte $^5/_4$-Rundung. Und ab Zeile 1270 wird nochmals die Möglichkeit für kleinere Zahlen mit CINT vorgeführt.

Der Befehl, der nur die Dezimalstellen abschneidet, heißt FIX, wie das Beispiel ab Zeile 1330 zeigt. Ab Zeile 1410 wird der Unterschied zwischen FIX und INT deutlich (der natürlich nur bei negativen Zahlen besteht).

Eine andere Möglichkeit des Rundens ist die Verwendung des Befehls ROUND. Zeile 1520 zeigt - nach der Ausgabe der Originalzahl - zunächst das Runden ohne Nachkommastellen, dann den Absolutwert der Zahl (ABS) und zum Abschluß noch den positiven Wert der Zahl auf zwei Nachkommastellen gerundet, u. a. auch als Beispiel für eine verschachtelte Eingabe.

Etwas Besonderes ist der UNT-Befehl, der Adreßausdrücke (0 bis 65535) in ihre ganzzahlige Darstellung umwandelt. Hierbei ist die interne Darstellung des Rechners - das sogenannte Zweierkomplement - zu beachten. Negative Werte sind nur bis zu einem Betrag von 32768 zugelassen, wie der Abschluß des Programms in Zeile 1620 zeigt.

Bei Eingabe von −1 und 65535 ergibt sich jeweils die gleiche Ausgabe. Die Werte 20000 und −20000 werden nicht verändert. Bei der Ausgabe in Zeile 1610 sieht man deutlich die Art der Umrechnung: Größere Zahlen als 32767 werden von 65535 abgezogen und mit einem negativen Vorzeichen versehen. Dies hat seine Ursache - wie bereits erwähnt - in der internen Darstellungsart von Zahlen im Rechner. Normalerweise werden Zahlen mit einem Vorzeichen-Bit und 15 „Zahl-Bits" dargestellt, so daß auf diese Art nur Zahlen im Bereich zwischen −32768 und +32767 darstellbar sind.

Dies können Sie sich verdeutlichen, indem Sie nacheinander auf dem Bildschirm eingeben:

```
PRINT BIN$(-25536)
PRINT BIN$(40000)
```

Beide Bitmuster sind gleich.

DIE FARBWAHL BEIM CPC

Der CPC bietet vielfältige Farbmöglichkeiten, die auch für die benutzerfreundliche Programmierung genutzt werden sollten. Da die Farbwahl im Handbuch etwas mißverständlich beschrieben ist, wollen wir hier noch einmal kurz darauf eingehen.

Anschaulich ausgedrückt, stellt Ihnen der CPC 27 verschiedene Tintenfarben und die gleiche Anzahl von Papierfarben zur Verfügung. Den Farben sind jeweils Nummern von 0 bis 26 zugeordnet. Um Ihnen das Nachschlagen im Handbuch zu ersparen, während Sie mit diesem Buch arbeiten, hier noch einmal die den Farbnummern zugeordneten Farben:

Tinte (INK)	Farbe
0	Schwarz
1	Blau
2	Hellblau
3	Rot
4	Magenta
5	Hellviolett
6	Hellrot
7	Purpur
8	helles Magenta
9	Grün
10	Blaugrün
11	Himmelblau
12	Gelb
13	Weiß
14	Pastellblau
15	Orange
16	Rosa
17	Pastellmagenta
18	Hellgrün
19	Seegrün

20	helles Blaugrün
21	Limonengrün
22	Pastellgrün
23	Pastellblaugrün
24	Hellgelb
25	Pastellgelb
26	Leuchtendweiß

Diese Farben können Sie jedoch nicht alle gleichzeitig nutzen. Die Anzahl der Farben, die Sie nutzen können, ist vom verwendeten Bildschirmmodus abhängig:

Mode 0 : 16 Farben
Mode 1 : 4 Farben
Mode 2 : 2 Farben

Den Rahmen (Border) lassen wir für diese Betrachtungen außer acht, da er gesondert eingefärbt werden kann und somit unabhängig vom verwendeten Bildschirmmodus ist. Mit dem INK-Befehl können Sie nun 2, 4 oder 16 der 27 Farben auswählen (siehe Abb. 1.38).

Die im Handbuch angegebene Beschreibung „Farbstift" für INK ist deshalb mißverständlich, weil mit dem INK-Befehl nicht der Farbstift direkt ausgewählt wird, sondern nur die in dem jeweiligen Modus zur Verfügung stehenden Farben.

Aus den ausgewählten Farben können Sie nun Ihrem Schreibstift (PEN) wiederum eine Farbe zuordnen, indem Sie die Nummer der vorher ausgewählten Tinte - wir wollen im weiteren von Farbauswahlnummer sprechen - diesem Schreibstift zuordnen. Genauso geht es mit der Hintergrundfarbe (dem Papier - PAPER). Abb. 1.39 S. 66 veranschaulicht diesen Sachverhalt: Mittels der INK-Befehle wird der Nummer 0 die Farbe 26 (Leuchtendweiß) zugeordnet und der Nummer 1 die Farbe 15 (Orange). Die Auswahl für Ihre Schreibstifte und Papiere beschränkt sich auf vier Möglichkeiten, wovon die Möglichkeiten 2 und 3 jedoch für normale Anwendungen nicht in Betracht kommen. Sie sind aber sinnvoll, wenn Sie zunächst den Bildschirm füllen wollen, ohne daß dies dem Anwender auffällt. Bei der Möglichkeit 1 schreiben Sie Leuchtendweiß auf orangem Hintergrund und bei der Möglichkeit 4 Orange auf weißem Hintergrund.

Bei Verwendung der Bildschirmmodi 0 und 1 ist der Aufbau analog, nur daß Sie hier zwischen 255 (Mode 0) und 16 Möglichkeiten (Mode 1) in der Farb-

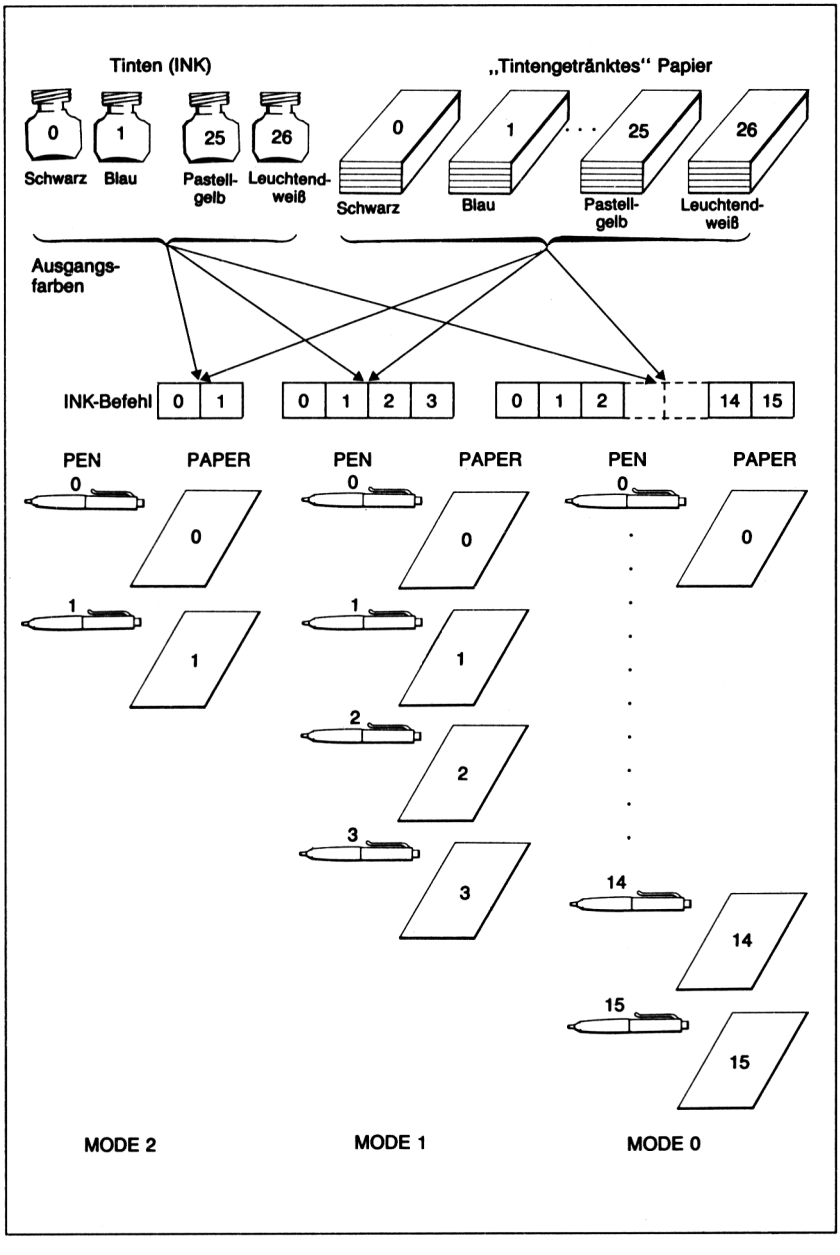


Abb. 1.38: „Tinten"- und „Papier"-Farben beim Schneider CPC

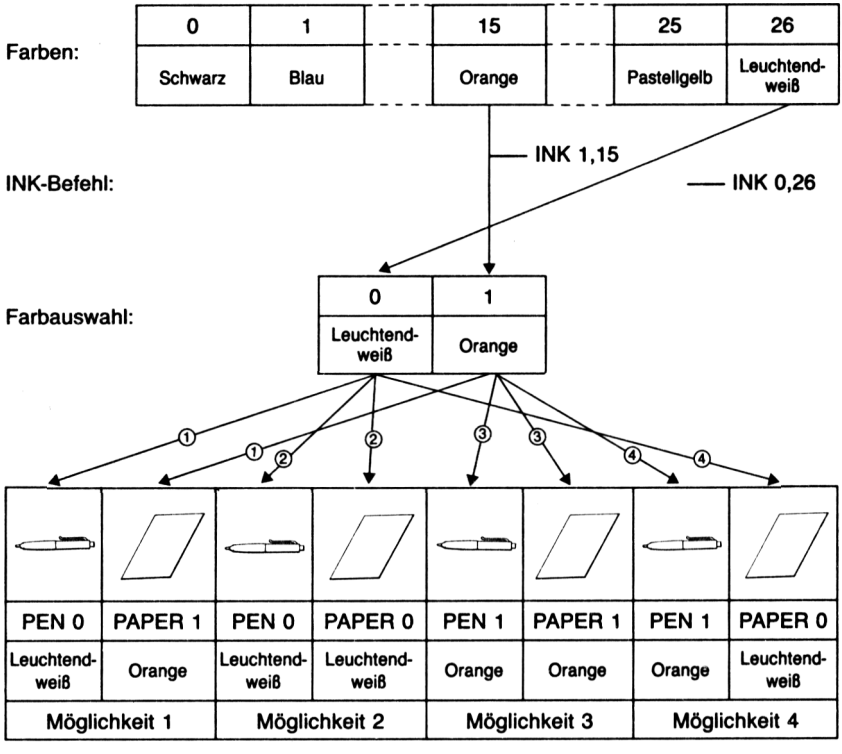


Abb. 1.39: Farbauswahlschema für Mode 2 (Beispiel)

kontrastierung wählen können. Beispiele dazu werden im Verlauf des Buchs noch genügend gegeben, z. B. das Kartenspiel in Kapitel 2.

Während eines Programmlaufs, aber auch im Direktmodus, haben Sie weiterhin die Möglichkeit, neue Farben mit dem INK-Befehl auszuwählen. Ersetzen Sie z. B. die Nummer 0 (Leuchtendweiß) durch die Farbe 1 (Blau), so werden auch die Farbstifte und Papierarten, die der INK-Nummer 0 zugeordnet sind, umgefärbt. Dies gilt auch für bereits ausgegebene Texte, Bildschirmhintergrund und Grafiken.

Damit Sie nicht gleich nach dem Einschalten eine Farbe einstellen müssen, sind bereits Farben vorgegeben, wie Sie im Handbuch auf Seite G3/Seite 4 nachlesen können. Wie man dort auch leicht erkennen kann, dürfen zwei verschiedenen Papier-/Schreibstiftnummern auch gleiche Farben zugeordnet werden. Durch die Zuordnung verschiedener Papierarten/Schreibstifte zu den verschiedenen Fenstern eröffnen sich weitere Möglichkeiten. Am besten

probiert man mit den drei Farbbefehlen, aber auch mit den Befehlen CLS (Bildschirm löschen), wenn die Papierfarbe umgesetzt wurde, CLG (Grafik löschen) und WINDOW sowie WINDOW SWAP etwas herum. Als kleiner Anreiz soll Ihnen das Programm aus Abb. 1.40 dienen.

Abb. 1.40: Beispiele zur Farbwahl

```
100 INK 0,1
110 INK 1,24
120 PAPER 0
140 PEN 1
1000 REM ------------------------------
1010 REM --     Farbwahl im Modus 2  ---
1020 REM ------------------------------
1030 :
1040 REM --- FAN = Farbauswahlnummer---
1050 :
1060 MODE 2
1070 INK 0,0    :REM FAN 0 auf Schwarz
1080 FOR I=1 TO 1000 : NEXT
1090 INK 1,26   :REM FAN 1 auf L.weiss
1100 LOCATE 10,10
1110 PRINT"Testausdruck INK 0,0 / INK 1,26
1120 GOSUB 10000
1130 REM ------------------------------
1140 CLS
1150 LOCATE 10,10
1160 INK 2,12
1170 PRINT"INK 2,12 bringt keine Wirkung"
1180 GOSUB 10000
1190 REM PEN 0
1200 CLS
1210 FOR I=1 TO 100
1220    PRINT"AAAAAAAAAAAAAAAAAAAA";
1230 NEXT
1240 FOR I=1 TO 1000
1250 NEXT
1260 INK 1,11
1270 PEN 1
1280 FOR I=1 TO 3000
1290 NEXT
1300 CLS
1310 LOCATE 10,10
1320 PRINT"Umschalten: PEN=PAPER, dann
1330 LOCATE 10,12
1340 PRINT"Farbwechsel und
1350 LOCATE 10,14
1360 PRINT"Umschalten PEN
1370 FOR I=1 TO 1000
1380 NEXT
1390 GOSUB 10000
2000 REM ------------------------------
2010 REM --     Farbwahl im Modus 1  ---
2020 REM ------------------------------
2030 MODE 1
2040 LOCATE 10,10
2050 PRINT"Umschalten in Modus 1
2055 CLS
2060 PAPER 0 : PRINT"PAPER 0"
2070 PAPER 1 : PRINT"PAPER 1"
2080 PAPER 2 : PRINT"PAPER 2"
2090 PAPER 3 : PRINT"PAPER 3"
2100 GOSUB 2900
2110 PEN 2
2120 GOSUB 10000
2130 GOTO 3000
2889 END
2899 REM ---      Unterprogramm       ---
2900    FOR I=0 TO 3
2910       FOR J=0 TO 3
2920          PAPER i
```

```
2930          PEN j
2940          PRINT"PAPER";i;" PEN";j
2950       NEXT
2960    NEXT
2970 RETURN
3000 REM ------------------------------
3010 REM --      Farbwahl im Mode 0  ---
3020 REM ------------------------------
3025 MODE 0
3030 CLS
3040 FOR I=0 TO 15
3050    FOR J=0 TO 15
3060       PAPER i
3070       PEN j
3080       PRINT"PAPER";i;" PEN";j
3090    NEXT
3100 NEXT
4000 REM ------------------------------
4010 REM ---     Farbmoeglichkeiten  ---
4020 REM ---          insgesamt      ---
4030 REM ------------------------------
4035 CLS
4036 INK 0,0 : PAPER 0 : CLS
4040 FOR I=0 TO 26
4050    INK 0,I
4060    FOR J=0 TO 26
4070       INK 1,J
4080       LOCATE 1,5
4090       PRINT"INK 0:";i;" INK 1:";j
4100       LOCATE 1,10
4110       PAPER 0 : PEN 1
4120       PRINT"PAPER 0 / PEN 1"
4130       PAPER 1 : PEN 0
4140       LOCATE 1,15
4150       PRINT"PAPER 1 / PEN 0"
4160       LOCATE 1,20
4170       PRINT"Taste druecken
4180       a$=INKEY$ : IF A$="" THEN GOTO 4180
4190    NEXT
4200 NEXT
5000 REM ------------------------------
5010 REM ---     Farbmoeglichkeiten  ---
5020 REM ---     mit Blinken         ---
5030 REM ------------------------------
5035 MODE 1
5040 INK 0,0 : PAPER 0 : PEN 1 : CLS
5050 FOR I=0 TO 26
5060    FOR J=0 TO 26
5065       IF I=J THEN GOTO 5240
5070       INK 0,I,J
5080       FOR K=0 TO 26
5090          FOR L=0 TO 26
5095             IF K=L THEN GOTO 5220
5100             INK 1,K,L
5110             LOCATE 1,5
5120             PRINT"INK 0:";i;",";j;" INK 1:";k;",";
l
5130             LOCATE 1,10
5140             PAPER 0 : PEN 1
5150             PRINT"PAPER 0 / PEN 1"
5160             PAPER 1 : PEN 0
5170             LOCATE 1,15
5180             PRINT"PAPER 1 / PEN 0"
5190             LOCATE 1,20
5200             PRINT"Taste druecken
5210             A$=INKEY$ : IF A$="" THEN GOTO 5210
5220          NEXT L
5230       NEXT K
5240    NEXT J
5250 NEXT I
9999 END
10000    LOCATE 10,15
10010    PRINT"bitte Taste druecken
10020    A$=INKEY$ : IF A$="" THEN GOTO 10020
10030 RETURN
```

Abb. 1.40: Beispiele zur Farbwahl (Fortsetzung)

Da PAPER 0 und PEN 1 als Standard eingestellt sind, führt das Umfärben der Farbauswahlnummern 0 und 1 sofort zu einer Änderung auf dem Bildschirm (1070 und 1090).

Ab Zeile 1190 wird ein weiteres Experiment gestartet, das zwei verschiedene Auswirkungen zeigt. Wird in Zeile 1190 der Schreibstift nicht auf die Farbauswahl Nummer 0 gesetzt, so wird der ganze Bildschirm mit A vollgeschrieben und anschließend die Farbe von A in Himmelblau umgewandelt. Geben Sie jedoch in Zeile 1190 dem Schreibstift die gleiche Farbe wie dem Papier, so erfolgt im weiteren keine Umwandlung von A in Himmelblau.

Beim Umschalten in Mode 1 wird zwar der Bildschirm gelöscht, aber die Farbgebung ändert sich nicht. Im weiteren sieht man die nächsten beiden vordefinierten Farben: Für PAPER 2 (INK 2) ist es helles Blaugrün und für PAPER 3 (INK 3) ist es Hellrot. Anschließend wird im Text am Bildschirm jeweils angegeben, welche Modifikationen vorgenommen wurden. Das Unterprogramm ab Zeile 2900 zeigt, daß auch Farben als Variablen verwendet werden können. Die nichtlesbaren Bereiche ergeben sich aus dem Zusammenhang.

Ab Zeile 3000 finden Sie die Aufstellung der Farbkombinationen für den Modus 0, wobei Sie bitte zum Anhalten jeweils zwischen der ESC-Taste und einer anderen Taste auswählen. Ab Zeile 4000 schließlich befindet sich ein kurzes Programmstück, das Ihnen zur Auswahl Ihrer Lieblingskombinationen dienen soll. Es wurde absichtlich so ausführlich gestaltet, da bei Verwendung zweier gleicher Farben ihre Wirkung unterschiedlich sein kann, wenn sie als Schrift- oder Hintergrundfarbe wechselweise verwendet werden.

Wesentlich mehr Möglichkeiten ergeben sich durch das Blinken von Vorder- und Hintergrund, wie es ab Zeile 5000 realisiert ist. Durch die Programmierung wird auch deutlich, daß blinkende Farben nicht nur bei den Farbauswahlnummern 14 und 15 dargestellt werden können. Am Anfang des Programms ist die Grundeinstellung aufgeführt (Mode 2), so daß Sie durch Abbrechen und Neustart des Programms jederzeit das Blinken wieder abschalten können.

Kapitel 2

Der Zeichensatz des CPC

Der Schneider CPC hat einen gut ausgearbeiteten Zeichensatz, in dem neben den normalen Sonderzeichen, Ziffern und Buchstaben (groß und klein) auch etliche Grafikzeichen und sogar griechische Buchstaben definiert sind. Auch die vier Kartensymbole Kreuz, Pik, Herz und Karo sind im Zeichensatz enthalten, sowie Noten, Pfeile, Zeichen für maskulin und feminin und sogar verschiedene Darstellungsweisen eines Männchens. Zunächst wollen wir in diesem Kapitel Grundsätzliches über den Zeichensatz sagen, um dann anhand der Zeichen für Spielkarten ein kleines Beispiel für die Anwendung von Zeichen zu geben, die nicht direkt über die Tastatur angesprochen werden können. Zur Auflockerung ist auch ein ganzes Kartenspiel gezeigt.

Der letzte große Teil beschäftigt sich mit Änderungen des Zeichensatzes für diejenigen, die Anwendungsprobleme haben, die mit dem vorhandenen Zeichensatz nicht zu lösen sind. Auch wird z. B. für die Beschriftung von Diagrammen die Herstellung einer senkrechten Schrift aufgezeigt.

ALLGEMEINES

Um Ihnen die Zeichensuche im Handbuch etwas zu erleichtern, geben wir hier zunächst eine kurze Übersicht über die einzelnen Bereiche des Zeichensatzes:

```
 32       : Leerzeichen
 33 -  47 : Sonderzeichen
 48 -  57 : Ziffern
 58 -  64 : Sonderzeichen
 65 -  90 : Großbuchstaben
 91 -  96 : Sonderzeichen
 97 - 122 : Kleinbuchstaben
123 - 125 : Sonderzeichen
126 - 127 : Grafikzeichen
128       : Leerzeichen
```

129 - 143 : Grafikzeichen (4*4 Aufteilung)
144 - 159 : Grafikzeichen zur Rahmenbildung
160 - 163 : Sonderzeichen anderer Sprachen
164 : Copyright-Zeichen
165 - 175 : Spezielle Sonderzeichen (z. B. Indizes, Frage- und Ausrufezeichen, Spiegelbild)
176 - 191 : Kleines griechisches Alphabet
192 - 205 : Grafikzeichen mit „Schrägstrichen"
206 - 223 : Grafikzeichen mit Schachbrettmuster und Dreiecken
224 - 225 : Spezielle Grafikzeichen
226 - 229 : Grafikzeichen für Spielkarten
230 - 233 : Kringel rund/eckig; geschlossen/offen
234 - 235 : maskulin, feminin
236 - 237 : Viertelnote/Achtelnote
238 - 239 : Explosionszeichen/Rakete
240 - 247 : Diverse Zeichen
248 - 251 : Männchen
252 - 255 : Spezielle Grafikzeichen

Mit diesen Zeichen ist man für fast alle Anwendungsfälle gut gerüstet. Trotzdem gibt es mit den BASIC-Befehlen SYMBOL und SYMBOL AFTER die Möglichkeit, Zeichen selbst zu definieren, worauf wir später noch näher eingehen.

Eine inverse Darstellung der Zeichen ist im Zeichensatz nicht vorgesehen und auch nicht nötig, da durch Austausch der Farbzuordnung mit den Befehlen PEN und PAPER eine umgekehrte Darstellung gegenüber der Umgebung möglich ist.

Der Zugriff auf Zeichen, die nicht von der Tastatur direkt angesprochen werden, erfolgt am besten über den BASIC-Befehl CHR$( ), wobei die Nummer des Zeichens (siehe Tabelle oben) als Parameter angegeben werden muß. Häufig verwendete Sonderzeichen können auch mittels des KEY-Befehls auf Tasten gelegt werden, so wird z. B. mit folgendem Befehl das Copyright-Zeichen auf die ENTER-Taste in der Zehnertastatur gelegt:

```
KEY 139, CHR$(164)
```

Auch in diesem Fall ist es anzuraten, sich die Problematik durch Probieren näherzubringen.

Die im Handbuch nicht abgebildeten Zeichennummern 0 bis 31 sind - neben ihrer Funktion als Steuercodes - als Zeichen benutzbar. Der Code 1 ist näm-

lich der Steuercode, um diese Zeichen sichtbar zu machen. Das Programm aus Abb. 2.1 bringt Ihnen die Darstellung der Zeichen mit der Nummer 0 bis 31.

```
100 FOR I=0 TO 31
110     PRINT I,CHR$(1);CHR$(I)
120     PRINT
130 NEXT
```

Abb. 2.1: Programm zur Darstellung der Steuercodes

Speziell ergeben sich folgende Zeichen:

0 : Quadrat
1 : linker und oberer Strich
2 : kopfstehendes T
3 : rechter und unterer Strich
4 : Blitz/Hochspannung
5 : Quadrat mit X in der Mitte
6 : Abhakzeichen
7 : Glockenzeichen (entspricht dem vom Fernschreiber). Der Steuercode von 7 ist ein kurzer Piepston
8 : spitzer Pfeil nach links
9 : spitzer Pfeil nach rechts
10 : spitzer Pfeil nach unten
11 : spitzer Pfeil nach oben
12 : spitzer Doppelpfeil nach unten
13 : abgewinkelter Pfeil nach links
14 : Kreis mit X in der Mitte
15 : Kreis mit Kringel in der Mitte
16 : Quadrat mit Querstrich
17 : Vollkreis mit rechts offenem U
18 : Vollkreis mit kopfstehendem, rechts offenem U
19 : Vollkreis mit kopfstehendem, links offenem U
20 : Vollkreis mit links offenem U
21 : X mit Haken unten links/durchgestrichenes Abhakzeichen
22 : Symbol für Rechteckschwingung
23 : 90 Grad nach rechts gedrehtes T
24 : X mit Strich oben und unten
25 : senkrechter Strich mit Punkt in der Mitte
26 : an der Längsachse gespiegeltes Fragezeichen
27 : Kreis mit Querstrich waagerecht

28 : Vollquadrat mit links offenem U
29 : Vollquadrat mit kopfstehendem, links offenem U
30 : Vollquadrat mit kopfstehendem, rechts offenem U
31 : Vollquadrat mit rechts offenem U

Da die meisten Steuerzeichen die Bildschirmausgabe beeinflussen, wollen wir an dieser Stelle die Wirkung der Steuerzeichen besprechen:

Code	Name	Wirkung
0	NUL	Keine.
1	SOH	ermöglicht den Ausdruck der Zeichen 0 bis 31 (siehe Beispiel oben).
2	STX	Cursor ausschalten.
3	ETX	Cursor einschalten.
4	EOT	Dieser Steuercode hat einen Parameter, der den Bildschirmmodus angibt. Dieser Steuercode kann aber in Zusammenhang mit dem Parameter anstatt des MODE-Befehls verwendet werden. Es ergibt sich die übliche Zuordnung mit: CHR$(4) ; CHR$(0) : Modus 0 CHR$(4) ; CHR$(1) : Modus 1 CHR$(4) ; CHR$(2) : Modus 2
5	ENQ	Hinter diesem Steuercode muß noch ein Parameter folgen, der die Nummer eines Zeichens angibt, das an der jeweiligen Cursorposition (linke obere Ecke des Zeichens entspricht der Cursorposition) ausgegeben wird. Dies entspricht dem Zeichenausdruck unter Verwendung des TAG-Befehls.
6	ACK	Umkehrung von Code 21; läßt die Ausgabe von Zeichen auf dem Bildschirm wieder zu.
7	BEL	Piepston.
8	BS	Cursor ein Zeichen nach links.
9	TAB	Cursor ein Zeichen nach rechts.
10	LF	Cursor eine Zeile nach unten.
11	VT	Cursor eine Zeile nach oben.

Code	Name	Wirkung
12	FF	Aktuelles Bildschirmfenster löschen und Cursor in obere linke Ecke positionieren.
13	TR	Cursor auf den Anfang der Zeile (innerhalb des aktuellen Fensters) setzen.
14	SO	Im Parameter wird eine INK-Nummer angegeben, worauf die Papierfarbe umgeschaltet wird (Modulo 16).
15	SE	Entsprechend 14 für die PEN-Farbe.
16	DEL	Zeichen unter Cursor löschen (PEN-Farbe in PAPER-Farbe umwandeln).
17	DC1	Alle Zeichen links vom Cursor in der aktuellen Zeile und im Fenster werden (einschließlich) gelöscht.
18	DC2	Alle Zeichen rechts vom Cursor (einschließlich) innerhalb der aktuellen Zeile und dem aktuellen Fenster werden gelöscht.
19	DC3	Alle Zeichen vom Fensteranfang bis zur aktuellen Cursorposition werden (einschließlich) gelöscht.
20	DC4	Alle Zeichen ab dem Cursor (einschließlich) bis zum Fensterende werden gelöscht.
21	NAK	Zeichenausgabe unterdrücken.
22	YN	In einem weiteren Parameter wird der Überschreibmodus (0) oder der Transparentmodus (1) gesetzt.
23	ETB	In einem zusätzlichen Parameter wird der Grafik-Schreibmodus gesetzt, wobei folgende Parameter möglich sind: 0 : WRITE-Modus 1 : XOR-Modus 2 : AND-Modus 3 : OR-Modus
24	CRN	Invers-Modus einschalten durch Vertauschen von PEN- und PAPER-INKs. Durch Vertauschen der INKs arbeitet dieses Steuerkommando ähnlich wie ein Flip-Flop. Wenn Sie den Invers-Modus eingeschaltet haben und CHR$(24) eingeben, erscheint wieder der Normalmodus.

Code	Name	Wirkung
25	IM	Dieser Steuercode hat neun Parameter, die jeweil mit CHR$( ) anzugeben sind. Mit diesem Steuercode können neue Zeichen definiert werden, wobei die letzten acht Parameter jeweils die Zeilen von oben nach unten für das Zeichen im Binärmuster definieren.
26	SOB	Bei diesem Steuercode sind mit Hilfe von CHR$( ) vier Parameter einzugeben, die jeweils das Textfenster begrenzen. Dabei spezifizieren die ersten beiden Parameter die linke und die rechte Grenze (die größte ist jeweils die rechte Spalte) und die hinteren beiden Parameter die obere und untere Begrenzung (größere Angabe ist jeweils die untere Begrenzung). D. h. die Reihenfolge der Parameter 1 und 2 sowie 3 und 4 ist irrelevant.
27	ESC	Escape-Steuerzeichen für den Anwender.
28	FS	Dieser Steuercode entspricht dem INK-Befehl. Drei Parameter sind einzugeben, wobei der erste die INK-Nummer (Modulo 16) angibt und der zweite und dritte Parameter die beiden Farben.
29	GS	Dieser Steuercode entspricht dem BORDER-Befehl, wobei die Parameter die Farben (Modulo 32) angeben.
30	RS	Setzt den Cursor in die linke obere Ecke des aktuellen Fensters.
31		Positionieren des Cursors mittels zweier Parameter: Der erste gibt die Spalte, der zweite die Zeile an. Die linke obere Ecke hat die Parameter 1,1. Wobei zur Positionierung in Zeile 10, Spalte 20 einzugeben ist: CHR$(31); CHR$(10); CHR$(20) Dieser Steuercode entspricht dem LOCATE-Befehl.

Sofern Sie kein Programm im Rechner haben, sollten Sie mit den Steuercodes etwas herumprobieren. Wenn etwas schief geht, ist dies nicht tragisch, da Sie durch gleichzeitiges Drücken von SHIFT/CTRL/ESC immer in den Einschaltmodus Ihres Rechners zurückkommen.

GRAFIKZEICHEN DES ZEICHENSATZES

Wie bereits erwähnt, werden die über die Tastatur nicht erreichbaren Zeichen des Zeichensatzes am besten mit dem CHR$( )-Befehl verarbeitet. Am Beispiel von Spielkarten wollen wir Ihnen dies im folgenden zeigen. Dabei werden im Rahmen der Beispielprogramme auch noch einige andere BASIC-Befehle näher besprochen.

Spielkarten gestalten

Die Benutzung von Zeichen des Zeichensatzes, die nicht über die Tastatur angesprochen werden können, nehmen wir zum Anlaß, einmal verschiedene BASIC-Befehle im Zusammenhang darzustellen. Zunächst der Vorspann des Programms mit dem Testrahmen:

```
1000 REM ------------------------------
1010 REM --- Vorspann und Testrahmen---
1020 REM ------------------------------
1030 :
1040 REM ---      Kartenhoehen        ---
1050 :
1060 DATA A,2,3,4,5,6,7,8,9,Z,B,D,K
1070 DIM HOEHE$(13)
1080 FOR I=1 TO 13
1090    READ HOEHE$(I)
1100 NEXT
1110 :
1120 REM --- Bildsch./Farben einst. ---
1130 :
1140 BORDER 0
1150 MODE 1
1160 CLS
1170 :
1180 INK 0,26
1190 INK 1,4
1200 INK 2,7
1210 INK 3,0
1220 :
1230 REM ---     Testlauf            ---
1240 :
1250 FOR J=1 TO 4
1260    IF J=1 THEN FARBE=3 : ART=226
1270    IF J=2 THEN ART=229
1280    IF J=3 THEN FARBE=2 : ART=228
1290    IF J=4 THEN ART=227
1300    FOR HOEHE=1 TO 13
1310       X=16 : Y=10
1320       GOSUB 2000
1330       A$=INKEY$
1340       IF A$="" THEN 1330
1350    NEXT
1360 NEXT
1370 END
```

Abb. 2.2: Programmvorspann und Testrahmen

Als erstes wird in den Zeilen 1060 bis 1100 ein Feld HOEHE$( ) mit 13 Elementen so vorbesetzt, daß die Kartenhöhen (As = 1, . . ., König = 13) in auf-

steigender Reihenfolge entsprechend dem Index eingetragen sind. Anschließend wird die Rahmenfarbe auf Schwarz gesetzt und der 40-Zeichen-Modus eingeschaltet sowie der Bildschirm gelöscht. Für unsere vier zur Verfügung stehenden Farben wählen wir aus der Palette die Farben Leuchendweiß (0), Magenta (1), Purpur (2) und Rot (3) aus, wobei die im INK-Befehl angegebenen Farbauswahlnummern in Klammern gesetzt sind.

Ab Zeile 1250 befindet sich der Testrahmen, in dem alle Karten, von As bis König und in den Kartenfarben Kreuz, Pik, Herz und Karo, ausgegeben werden. Die Schleife mit der Laufvariablen J (Zeilen 1250 bis 1360) ist dabei für die Kartenfarbe zuständig. In der Variablen ART wird jeweils die Zeichennummer für Kreuz (226), Pik (229), Herz (228) und Karo (227) angegeben. Für die beiden schwarzen Farben wird der Paramater FARBE auf 3 gesetzt und entsprechend für die beiden roten Farben die Variable auf 2, was der Bildschirmfarbe Purpur entspricht. Beachten Sie, daß der Bildschirmhintergrund bereits in Zeile 1180 auf Leuchtendweiß umgesetzt wurde, da PAPER nach Einschalten des Systems auf 0 gesetzt ist.

Für die verschiedenen Kartenhöhen ist die Schleife von Zeile 1300 bis 1350 verantwortlich. Die X- und Y-Koordinate wird vorbesetzt, und in der Zahlvariablen HOEHE werden alle Kartenwerte durchlaufen. Damit Sie jetzt zur Kontrolle jede einzelne Karte anschauen können, ist in den Zeilen 1330 und 1340 noch eine Warteschleife eingebaut, die auf einen Tastendruck hin abgebrochen wird. Dies ist wohl der Hauptanwendungsfall für den INKEY$-Befehl.

Abb. 2.3: Darstellung der Karten (Unterprogramm)

```
2000 REM ------------------------------
2010 REM ---      KARTEN AUSGEBEN     ---
2020 REM ------------------------------
2030 :
2040 H$=HOEHE$(HOEHE)
2050 :
2060 REM --- ERSTE ZEILE --------------
2070 :
2080 PEN 1
2090 LOCATE X,Y
2100 PRINT CHR$(150);CHR$(154);CHR$(154)
;          CHR$(154);CHR$(154);CHR$(154)
;          CHR$(156)
2110 :
2120 REM --- ZWEITE ZEILE --------------
2130 :
2140 LOCATE X,Y+1
2150 PRINT CHR$(149);
2160 PEN FARBE
2170 PRINT H$;"    ";H$;
2180 PEN 1
2190 PRINT CHR$(149)
2200 :
2210 REM --- DRITTE ZEILE -------------
2220 :
2230 LOCATE X,Y+2
2240 PRINT CHR$(149);
2250 PEN FARBE
```

```
2260 IF HOEHE=3
       THEN PRINT"  ";CHR$(ART);"  "; :
            GOTO 2290
2270 IF HOEHE>5
       THEN PRINT" ";CHR$(ART);" ";
            CHR$(ART);" "; :
            GOTO 2290
2280 PRINT"      ";
2290 PEN 1
2300 PRINT CHR$(149)
2310 :
2320 REM --- VIERTE ZEILE -------------
2330 :
2340 LOCATE X,Y+3
2350 PRINT CHR$(149);
2360 PEN FARBE
2370 IF HOEHE=2
       THEN PRINT"  ";CHR$(ART);"  "; :
            GOTO 2400
2380 IF HOEHE=4 OR HOEHE=5 OR HOEHE>7
       THEN PRINT" ";CHR$(ART);" ";
            CHR$(ART);" "; :
            GOTO 2400
2390 PRINT"      ";
2400 PEN 1
2410 PRINT CHR$(149)
2420 :
2430 REM --- FUENFTE ZEILE ------------
2440 :
2450 LOCATE X,Y+4
2460 PRINT CHR$(149);
2470 PEN FARBE
2480 IF HOEHE=1 OR HOEHE=3 OR
        HOEHE=5 OR HOEHE=9
       THEN PRINT"  ";CHR$(ART);"  "; :
             GOTO 2520
2490 IF HOEHE=6 OR HOEHE>9
       THEN PRINT" ";CHR$(ART);" ";
            CHR$(ART);" "; :
             GOTO 2520
2500 IF HOEHE=7
       THEN PRINT " ";CHR$(ART);
            CHR$(ART);CHR$(ART);" "; :
            GOTO 2520
2510 PRINT"      ";
2520 PEN 1
2530 PRINT CHR$(149)
2540 :
2550 REM --- SECHSTE ZEILE ------------
2560 :
2570 LOCATE X,Y+5
2580 PRINT CHR$(149);
2590 PEN FARBE
2600 IF HOEHE=2
       THEN PRINT"  ";CHR$(ART);"  "; :
             GOTO 2630
2610 IF HOEHE=4 OR HOEHE=5 OR HOEHE>7
       THEN PRINT" ";CHR$(ART);" ";
            CHR$(ART);" "; :
            GOTO 2630
2620 PRINT"      ";
2630 PEN 1
2640 PRINT CHR$(149)
2650 :
2660 REM --- SIEBTE ZEILE -------------
2670 :
2680 LOCATE X,Y+6
2690 PRINT CHR$(149);
2700 PEN FARBE
2710 IF HOEHE=3
       THEN PRINT"  ";CHR$(ART);"  "; :
            GOTO 2740
2720 IF HOEHE>5
       THEN PRINT" ";CHR$(ART);" ";
            CHR$(ART);" "; :
            GOTO 2740
2730 PRINT"      ";
2740 PEN 1
2750 PRINT CHR$(149)
```

```
2760 :
2770 REM --- ACHTE ZEILE --------------
2780 :
2790 LOCATE X,Y+7
2800 PRINT CHR$(149);
2810 PEN FARBE
2820 PRINT H$;"    ";H$;
2830 PEN 1
2840 PRINT CHR$(149)
2850 :
2860 REM --- NEUNTE ZEILE -------------
2870 :
2880 LOCATE X,Y+8
2890 PRINT CHR$(147);CHR$(154);CHR$(154)
;          CHR$(154);CHR$(154);CHR$(154)
;          CHR$(153)
2900 RETURN
```

Abb. 2.3: Darstellung der Karten (Fortsetzung)

In den Zeilen ab 2000 befindet sich das eigentliche Unterprogramm zur Darstellung der Karten, wobei wir hier aus Demonstrationsgründen einen ausführlichen Weg gegangen sind. Um die Zeilen nicht allzu kompliziert werden zu lassen, haben wir das Kartenzeichen entsprechend der gewünschten Kartenhöhe in der Variablen H$ abgelegt. Die Karte wird nun zeilenweise aufgebaut, wobei für eine Karte mindestens neun Zeilen benötigt werden, um einen einigermaßen ansprechenden Aufbau zu erhalten. Im Verhältnis dazu, und auch wegen des optischen Aufbaus der einzelnen Zeilen, muß eine Karte sieben Zeichen breit sein. Damit können auch noch Spiele wie z. B. Poker oder Black Jack auf dem Bildschirm dargestellt werden, da maximal fünf Karten nebeneinander passen. Als Übung können Sie auch ausprobieren, wie Karten zu überlagern sind, was mit den Möglichkeiten des CPC recht einfach ist.

Der Aufbau jeder einzelnen Zeile ist in etwa gleich, wobei die erste und die letzte Zeile relativ einfach gestaltet sind. Der Rahmen einer Karte wird in Magenta gezeichnet, dementsprechend wird für die erste Kartenzeile in Zeile 2080 der Farbstift umbesetzt. Die durch die Koordination X und Y angegebene Position ist die linke obere Ecke der Karte; dorthin wird der Cursor positioniert. Die Zeile 2100 gibt die obere Zeile aus, die aus den Grafikzeichen „Ecke unten rechts" (150), 5mal „Strich in der Mitte" (154) und „Ecke unten links" (156) besteht.

In der zweiten Zeile muß natürlich der Cursor eine Spalte tiefer positioniert werden. Durch die freie Positionierung einer Karte auf dem Bildschirm kann übrigens dieses Unterprogramm für alle Bildschirmpositionen bei entsprechender Angabe von X und Y ohne weitere Positionsangaben aufgerufen werden. Im weiteren wird ein „senkrechter Strich in der Mitte" (149) als linke Kartenbegrenzung ausgegeben. Anschließend wird der Schreibstift auf die gewünschte Kartenfarbe gesetzt. Die zweite Kartenzeile gibt in den beiden oberen Ecken immer die Kartenhöhe aus (Programmzeile 2170). Anschließend

wird der Schreibstift wieder auf die Farbe Magenta zurückgesetzt, um den rechten Begrenzungsstrich auszugeben.

Etwas komplizierter wird es in der dritten Bildschirmzeile. Nachdem der Cursor positioniert, der linke Begrenzungsstrich gedruckt und die Kartenfarbe eingestellt ist, muß die weitere Abfolge abhängig von der Kartenhöhe geschehen. Sofern die Kartenhöhe 3 ist, muß in die Mitte der Karte das Kartenfarbensymbol eingetragen werden. Bei Karten höheren Wertes als 5 ist jeweils ein Zeichen rechts und links von der Mitte die Kartenfarbe auszugeben. In allen anderen Fällen sind nur Leerzeichen auszugeben (Zeile 2280). Anschließend muß natürlich der Farbstift wieder auf die Rahmenfarbe umgestellt und der rechte Begrenzungsrand ausgegeben werden.

Ab der vierten Zeile wollen wir nun noch die einzelnen Einträge in der Zeile besprechen, die Sie durch Vergleich mit Spielkarten nachvollziehen können. In der vierten Zeile muß im mittleren Zeichen die Kartenfarbe eingetragen werden, wenn eine 2 vorliegt. Zwei Kartenzeichen rechts und links von der Mitte müssen bei den Karten 4, 5 und 8 bis K erfolgen. Wir haben versucht, die Problematik durch Strukturierung des Programms so einfach wie möglich darzustellen.

Am kompliziertesten ist die fünfte (mittlere) Kartenzeile. Hier müssen sowohl ein als auch zwei oder sogar drei Kartenzeichen richtig positioniert werden, dies natürlich wieder in Abhängigkeit von der Kartenhöhe. Hier werden die Möglichkeiten der Programmstrukturierung mittels Einfügen von Leerzeichen sehr deutlich.

Die Kartenzeilen 6, 7 und 8 sind wegen der spiegelbildlichen Darstellung von Spielkarten auf die Zeilen 4, 3 und 2 (in dieser Reihenfolge) zurückzuführen. Aus Gründen der Übersichtlichkeit haben wir diese jedoch nicht als Unterprogramm gestaltet. Die einzige Änderung befindet sich jeweils bei dem LOCATE-Befehl, der den Cursor natürlich anders positionieren muß. Das Gegenstück zur ersten Zeile ist die letzte (neunte) Zeile, die ebenfalls wie die erste Zeile aufgebaut ist und sich nur in dem äußerst linken und äußerst rechten Element unterscheidet (Ecke oben links/Ecke oben rechts).

Dieses Unterprogramm wollen wir im folgenden in ein kurzes Kartenspiel einbauen.

Ein kleines Kartenspiel

Wir wollen als Beispiel für den Einsatz des eben erklärten Programmteils ein kleines Kartenspiel beschreiben, bei dem auch andere wichtige Befehle wie z. B. WINDOW benutzt werden.

Das Kartenspiel selbst hat recht einfache Regeln: Jeder Spieler bekommt zwei Karten offen ausgelegt und eine verdeckte Karte daZWISCHEN. Daher ist das Spiel auch unter dem Namen ZWISCHEN bekannt, aber es gibt noch einen weiteren Grund für den Namen: Der Spieler muß nun gegen einen Pott wetten, ob die verdeckte Karte in ihrer Höhe ZWISCHEN die beiden offenen Karten paßt. Minimal-Einsatz ist eine Einheit, Maximum so viele Einheiten, wie sich im Pott befinden. Damit auch immer etwas im Pott ist, wird er durch eine Einheit von jedem Mitspieler aufgefüllt, sofern sein Inhalt geringer ist als die Anzahl der Mitspieler. Soviel zum Kartenspiel selbst; kommen wir nun zu dem Listing, wobei wir mit dem Rahmenprogramm und dem Bildschirmaufbau beginnen (Abb. 2.4).

Abb. 2.4: Rahmenprogramm und Bildschirmaufbau

```
1000 REM ------------------------------
1010 REM ---        VORSPANN          ---
1020 REM ------------------------------
1030 :
1040 :
1050 REM ---      SPIELERDATEN        ---
1060 :
1070 CLS
1080 LOCATE 10,10
1090 INPUT"Wie viele Mitspieler";mi
1100 :
1110 DIM KAPITAL(MI)
1120 DIM KAPHILF(MI)
1130 DIM NAME$(MI)
1140 DIM KARTEN(MI*3,2)
1150 :
1160 FOR I=1 TO MI
1170    KAPITAL(I)=99
1180    CLS
1190    LOCATE 10,10
1200    PRINT"Name des";i;"-ten Spielers
1210    LOCATE 10,15
1220    INPUT NAME$(I)
1230 NEXT
1240 :
1250 POTT=MI
1260 :
1270 REM --- KARTENHOEHEN -------------
1280 :
1290 DATA A,2,3,4,5,6,7,8,9,Z,B,D,K
1300 DIM HOEHE$(13)
1310 FOR I=1 TO 13
1320    READ HOEHE$(I)
1330 NEXT
1340 :
1350 REM ---    BILDSCHIRMAUFBAU     ---
1360 :
1370 BORDER 17
1380 :
1390 WINDOW #1,5,13,2,13
1400 WINDOW #2,16,26,2,13
1410 WINDOW #3,28,36,2,13
1420 WINDOW #4,5,36,16,23
1430 WINDOW #5,18,24,3,12
1440 :
1450 INK 0,0
1460 INK 1,7
1470 INK 2,15
1480 INK 3,1
1490 :
1500 PAPER 0
1510 PAPER #1,2
1520 PAPER #2,3
```

```
1530 PAPER #3,2
1540 PAPER #4,2
1550 PAPER #5,3
1560 :
1570 PEN 2
1580 PEN #4,0
1590 PEN #5,1
1600 :
1610 CLS
1620 CLS #1
1630 CLS #2
1640 CLS #3
1650 CLS #4
1660 CLS #5
1670 :
1680 X=2 : Y=2
1690 :
1700 REM ---  AUSSEHEN VERDECKTE K. ---
1710 :
1720 VERDECKT$=CHR$(207)
1730 :
1740 FOR I=1 TO 7
1750    VERDECKT$=VERDECKT$+VERDECKT$
1760 NEXT
1770 :
1780 VERDECKT$ = LEFT$(VERDECKT$,70)
1790 :
```

Abb. 2.4: Rahmenprogramm und Bildschirmaufbau (Fortsetzung)

Nachdem der Bildschirm gelöscht wurde, wird der Cursor in dem Textfeld, das jeweils 10 Felder vom oberen und linken Rand entfernt ist, positioniert und anschließend die Anzahl der Mitspieler erfragt. Dann werden vier Felder dimensioniert, die folgende Bedeutung haben:

KAPITAL - Eigenkapital jedes Mitspielers
KAPHILF - Hilfsgeld für das Kapital zum Sortieren für Rangliste
NAME$ - Namen der Mitspieler
KARTEN - Ausgeteilte Karten aller Mitspieler:
jeweils alle drei Karten, wobei die beiden Spalten der Matrix folgende Bedeutung haben:

1. Eintrag - Kartenfarbe
2. Eintrag - Kartenhöhe

Dann wird für alle Mitspieler in der Schleife von Zeile 1160 bis 1230 das Spielkapital bereitgestellt und der Name erfragt, wobei wieder der LOCATE-Befehl herangezogen wird. In Zeile 1250 wird der Pott aufgefüllt, so daß jeder Mitspieler zu Beginn ein Kapital von 100 Einheiten hat.

In den Zeilen ab 1290 werden die Zeichen für die Kartenhöhen, von As bis König, abgekürzt in dem Feld HOEHE$( ) abgelegt.

Beim Bildschirmaufbau wird zunächst der Rand auf Pastellmagenta gesetzt, dann werden die erforderlichen fünf Bildschirmfenster definiert. Die Fenster haben folgende Bedeutung:

WINDOW #1 : Fenster oben links für erste offene Karte
WINDOW #2 : Fenster oben Mitte für verdeckte Karte
WINDOW #3 : Fenster oben rechts für zweite offene Karte
WINDOW #4 : Feld unten über fast die gesamte Bildschirmbreite für Anweisungen und Anzeigen
WINDOW #5 : Fenster innerhalb von WINDOW 2 für die Ausgabe der verdeckten Karte

Gerade am letzten Beispiel sieht man auch, daß mehrere Fenster ineinandergeschachtelt werden können. Die Definition eines fünften Fensters ist erforderlich, um eine entsprechende Farbgestaltung durchführen zu können. Der Hintergrund für die Karten sollte immer hell sein, um die Kartenfarben Rot und Schwarz besser zur Geltung zu bringen. Da im Mode 1 nur vier Farben zur Verfügung stehen, wurde teilweise davon abgewichen, aber auch mit vier Farben läßt sich eine unterschiedliche Thematik am Bildschirm kontrastreich darstellen.

Ab Zeile 1450 werden die einzelnen Farben wie folgt ausgewählt:

Farbauswahlnummer 0 : Schwarz
Farbauswahlnummer 1 : Purpur
Farbauswahlnummer 2 : Orange
Farbauswahlnummer 3 : Blau

Anschließend werden die entsprechenden Papierfarben vergeben, die für die einzelnen Fenster wie folgt vorbesetzt sind:

Papierfarbe Gesamtbildschirm : Schwarz
Papierfarbe Fenster 1 : Orange
Papierfarbe Fenster 2 : Blau
Papierfarbe Fenster 3 : Orange
Papierfarbe Fenster 4 : Orange
Papierfarbe Fenster 5 : Blau

Im weiteren werden die Farbstiftfarben gesetzt, wobei die Farbstifte für die drei Fenster zur Darstellung der Karten später bei deren Darstellung gesetzt werden und eine Vorbesetzung somit nicht erforderlich ist. Für den Gesamtbildschirm und die Fenster 4 und 5 ergibt sich folgende Zuordnung:

Farbstift Gesamtbildschirm : Orange
Farbstift Fenster 4 : Schwarz
Farbstift Fenster 5 : Purpur

Ab Zeile 1610 werden alle Fenster gelöscht, damit die eingestellten Farben auch als Hintergrund erscheinen. In Zeile 1680 werden zwei Koordinaten vorbesetzt, die später bei der Ausgabe der Karten Verwendung finden.

Zur Gestaltung der verdeckten Karte wird das Zeichen mit der Nummer 207 herangezogen (Schachbrett). Die verdeckte Karte wird später in Fenster fünf ausgegeben. Damit nicht 70mal CHR$(207) geschrieben werden muß, wird die Zeichenkette VERDECKT$ siebenmal verdoppelt und anschließend auf 70 Zeichen abgeschnitten.

```
2000 REM ------------------------------
2010 REM ---      KARTEN VERTEILEN    ---
2020 REM ------------------------------
2030 :
2040 LOCATE #4,2,2
2050 PRINT #4,"Die Karten werden gemischt"
2060 :
2070 FOR I=1 TO MI*3
2080    KARTEN(I,1)=INT(RND(4)*13)+1
2090    KARTEN(I,2)=INT(RND(5)*4)+1
2100    FOR J=I-1 TO 1 STEP -1
2110       IF KARTEN(I,1)=KARTEN(J,1)
             THEN
             IF KARTEN(I,2)=KARTEN(J,2)
               THEN  GOTO 2080
2120    NEXT
2130 NEXT
2140 :
2150 GOSUB 6060
```

Abb. 2.5: Karten verteilen (Teilprogramm)

Nach dem Benutzerhinweis, daß die Karten gemischt werden (Fenster 4), werden alle drei Karten für jeden Mitspieler per Zufallsgenerator mit Höhe und Farbe ermittelt. Dazu werden nicht die Karten direkt gemischt, sondern es wird zunächst jeweils eine Kartenhöhe und -farbe festgelegt, wobei die Werte anschließend mit bereits ermittelten Kartenwerten verglichen werden. Tritt hier eine Übereinstimmung auf, so wird neu ausgelost. Durch den Unterprogrammaufruf in Zeile 2150 werden die Karten für jeden Spieler sortiert, so daß die niedrige Karte links und die höhere Karte rechts auf dem Bildschirm erscheint.

Abb. 2.6: Spielerschleife (Teilprogramm)

```
3000 REM ------------------------------
3010 REM ---     Spielerschleife      ---
3020 REM ------------------------------
3030 FOR SPIELER=1 TO MI
3040 :
3050 REM ---     ERSTE KARTE          ---
3060 :
3070 H2=1
3080 GOSUB 9000
3090 :
```

```
3100 REM ---    VERDECKTE KARTE       ---
3110 :
3120 PRINT #5,VERDECKT$
3130 :
3140 REM ---     DRITTE KARTE         ---
3150 :
3160 H2=3
3170 GOSUB 9000
3180 :
3190 REM ---    SPIELERHANDLUNG       ---
3200 :
3210 CLS #4
3220 PRINT #4
3230 PRINT #4," Im Pott sind:";POTT
3240 PRINT #4," Dein Kapital:";KAPITAL(SPIELER)
3250 PRINT #4
3260 PRINT #4," ";NAME$(SPIELER);", Was setzt Du";
3270 INPUT #4,EINSATZ
3280 IF EINSATZ > POTT THEN PRINT #4 :
        PRINT #4," Maximal Pott" :
        PRINT CHR$(7) :
        FOR WARTE=1 TO 2000 : NEXT :
        GOTO 3210
3290 REM ---       AUFDECKEN          ---
3300 H2=2
3310 PAPER #2,2
3320 CLS #2
3330 X=3
3340 GOSUB 9000
3350 X=2
```

Abb. 2.6: Spielerschleife (Fortsetzung)

Der Hauptteil des Programms besteht aus der Spielerschleife ab Zeile 3000. Die Schleife beginnt in Zeile 3030 und endet in Zeile 4320. Auf ein Einrücken wurde hier wegen der Größe der Schleife verzichtet.

In der Hilfsvariablen H2 wird die Nummer der Karte an das Unterprogramm ab Zeile 9000 übergeben. Nachdem die beiden offenen Karten am Bildschirm angezeigt sind, wird dem Spieler eine Information über den Inhalt des Potts und sein Kapital übermittelt, und er wird anschließend nach seinem Einsatz gefragt. Ist der Einsatz höher als der Inhalt des Potts, so wird in Zeile 3280 eine Fehlermeldung gebracht und die Frage nach dem Einsatz wiederholt. Sobald der Einsatz korrekt getätigt worden ist, wird die verdeckte Karte aufgedeckt, wobei der Hintergrund den bereits aufgedeckten Karten angepaßt wird. Da die linke obere Ecke der mittleren Karte drei Bildschirmzeichen vom linken Fensterrand entfernt liegt, wird die X-Koordinate vorher entsprechend umbesetzt und nachher wieder korrigiert.

Abb. 2.7: Auswertung (Teilprogramm)

```
4000 REM ------------------------------
4010 REM ---        AUSWERTUNG         ---
4020 REM ------------------------------
4030 H1=(SPIELER-1)*3
4040 IF KARTEN(H1+1,1)<KARTEN(H1+2,1)
      AND KARTEN(H1+2,1)<KARTEN(H1+3,1)
      THEN GOTO 4200
```

```
4050 :
4060 REM ---        VERLOREN          ---
4070 :
4080 PAPER #4,1
4090 CLS #4
4100 LOCATE #4,5,3
4110 PEN #4,2
4120 PRINT #4," Du hast leider VERLOREN"
4130 FOR WARTE=1 TO 3000 : NEXT
4140 KAPITAL(SPIELER)=KAPITAL(SPIELER)-EINSATZ
4150 POTT=POTT+EINSATZ
4160 PAPER #4,2
4170 PEN #4,0
4180 GOTO 4310
4190 :
4200 REM ---        GEWONNEN          ---
4210 :
4220 CLS #4
4230 LOCATE #4,5,3
4240 PRINT #4," ZWISCHEN - ZWISCHEN - ZWISCHEN"
4250 FOR WARTE=1 TO 3000 : NEXT
4260 KAPITAL(SPIELER)=KAPITAL(SPIELER)+EINSATZ
4270 POTT=POTT-EINSATZ
4280 IF POTT < MI THEN GOSUB 7000
4290 :
4300 REM ---   NAECHSTER SPIELER    ---
4310 :
4320 NEXT
```

Abb. 2.7: Auswertung (Fortsetzung)

Auswertung

Um unübersichtliche Anweisungen zu vermeiden, wird in der Hilfsvariablen H1 eine Grundzahl festgelegt, die die letzte Karte des letzten Spielers angibt, so daß die Karten des Spielers die Nummern H1 + 1, H1 + 2 und H1 + 3 haben. Die wichtigste Zeile für die Auswertung ist die Zeile 4040, wo die Überprüfung auf „Dazwischenliegen" erfolgt. Da durch das Sortieren jeweils links die kleinere Karte (H1 + 1) und rechts die höhere Karte (H1 + 3) liegt, kann die Auswertung auf diese einfache Weise erfolgen. Ist die Höhe der mittleren Karte zwischen den beiden anderen, so wird zur Zeile 4200 übergegangen, wo der Abschnitt für „gewonnen" abgehandelt wird.

Zunächst jedoch die Behandlung für „verloren": Nach dem Umsetzen einiger Farben für Fenster 4 wird die Verlustmeldung ausgegeben, das Kapital des Spielers um seinen Einsatz verringert sowie der Inhalt des Potts um diesen Einsatz erhöht. Anschließend werden die Papierfarbe und die Farbstiftfarbe auf den ursprünglichen Wert zurückgesetzt, und es wird zum Ende der Schleife gesprungen, womit der nächste Spieler an der Reihe ist.

Ähnliches passiert auch, falls die Wette gewonnen wurde, natürlich mit umgekehrtem Vorzeichen. Hierbei ist jedoch auch noch darauf zu achten, ob das Minimum im Pott (eine Einheit je Spieler) unterschritten ist. In diesem Falle muß der Pott aufgefüllt werden, was durch ein Unterprogramm realisiert ist.

Ende einer Runde

```
5000 REM ------------------------------
5010 REM ---       RANGLISTE          ---
5020 REM ------------------------------
5030 GOSUB 8000
5040 CLS #4
5050 LOCATE #4,5,3
5060 PRINT #4,"Noch'ne Runde (j/n)";
5070 A$=INKEY$
5080 IF A$="J" OR A$="j"
       THEN CLS #1 : CLS #2 : CLS #3 :
            GOTO 2040
5090 IF A$="N" OR A$="n"
       THEN LOCATE #4,5,3 :
         PRINT #4," OK! Tschuess !!" :
         END
5100 GOTO 5070
```

Abb. 2.8: Rangliste (Teilprogramm)

Nachdem alle Spieler einmal gewettet haben, wird zunächst eine Rangliste ausgegeben, was in dem Unterprogramm ab Zeile 8000 geschieht. Anschließend wird gefragt, ob noch eine Runde gespielt werden soll, was durch Eingabe von J, j, N und n beantwortet werden kann. Das Drücken der ENTER-Taste wird mittels des Befehls INKEY$ umgangen.

Sortieren der Karten je Spieler

```
6000 REM ******************************
6010 REM ******************************
6020 REM ***     Unterprogramme     ***
6030 REM ******************************
6040 REM ******************************
6050 :
6060 REM ------------------------------
6070 REM --- KARTEN JE SPIELER SORT.---
6080 REM ------------------------------
6090    FOR I=0 TO MI-1
6100       IF KARTEN(I*3+1,1)>KARTEN(I*3+3,1)
             THEN GOTO 6140
6110    NEXT
6120 RETURN
6130 REM --- VERTAUSCHEN DER KARTEN ---
6140    HI              =KARTEN(I*3+1,1)
6150    KARTEN(I*3+1,1)=KARTEN(I*3+3,1)
6160    KARTEN(I*3+3,1)=HI
6170    GOTO 6110
```

Abb. 2.9: Karten sortieren und vertauschen (Unterprogramm)

Das Sortieren der beiden offenen Karten für jeden Spieler geschieht in den Zeilen ab 6090. Von Sortieren kann hier im eigentlichen Sinne nicht die Rede sein, da die erste und dritte Karte eines Spielers nur vertauscht werden, sofern die dritte Karte höher als die erste ist.

Pott auffüllen

```
7000 REM ------------------------------
7010 REM ---     POTT AUFFUELLEN      ---
7020 REM ------------------------------
7030     POTT = POTT+MI
7040     FOR FUELLEN=1 TO MI
7050        KAPITAL(FUELLEN)=KAPITAL(FUELLEN)-1
7060     NEXT
7070 RETURN
```

Abb. 2.10: Pott auffüllen (Unterprogramm)

Das Auffüllen des Potts geschieht, indem zunächst dem Pott die entsprechenden Einheiten hinzugefügt werden und anschließend jedem Spieler eine Einheit abgezogen wird. Als Laufvariable ist die Verwendung der Variablen SPIELER nicht anzuraten, da das Unterprogramm innerhalb der Spielerschleife aufgerufen wird.

Rangliste sortieren

```
8000 REM ------------------------------
8010 REM ---  RANGLISTE ERSTELLEN   ---
8020 REM ------------------------------
8030     FOR I=1 TO MI
8040        KAPHILF(I)=KAPITAL(I)
8050     NEXT
8060     CLS #1
8070     CLS #2
8080     CLS #3
8090     LOCATE #1,2,5
8100     PRINT #1,"RANGLISTE"
8110     FOR I=1 TO MI
8120        MAXIMUM = -10000
8130        FOR J=1 TO MI
8140           IF KAPHILF(J) > MAXIMUM
                 THEN MAXIMUM=KAPHILF(J):
                      WO=J
8150        NEXT J
8160     PRINT #2,NAME$(WO)
8170     PRINT #3,KAPHILF(WO)
8180     KAPHILF(WO)=-11111
8190     NEXT I
8200 RETURN
8210 END
```

Abb. 2.11: Rangliste erstellen (Unterprogramm)

Zur Erstellung der Rangliste wird das Kapital der einzelnen Spieler zunächst in die Variable KAPHILF( ) transferiert. Die Fenster 1 bis 3 werden gelöscht, weil die einzelnen Daten der Rangliste hier später ausgegeben werden sollen. Das Sortieren der Rangliste geschieht mit einem ähnlichen Verfahren wie in MINSORT. Allerdings wird innerhalb des Feldes KAPHILF( ) nicht umsortiert, sondern ermittelte Maxima (hier wird absteigend sortiert) werden sofort

ausgegeben. Damit ein bereits ausgegebener Wert zum Sortieren nicht mehr herangezogen wird, wird in Zeile 8180 der Wert −11111 eingetragen, weil die Vorgabe für das Maximum in Zeile 8120 −10000 war.

Karten ausgeben

Das Unterprogramm wurde bereits besprochen, so daß wir hier nur noch auf die Änderungen eingehen wollen. Zunächst wurde das Auswerten der Parameter noch in das Unterprogramm übernommen (Farbe und Art vorbesetzen). Außerdem wurden die LOCATE-Befehle, PRINT-Befehle und PEN-Befehle den jeweiligen Fenstern angepaßt.

Wir hoffen, Sie haben viel Spaß mit dem Spiel, zumal die Mitspielerzahl beliebig ist, das Spiel also auch auf Parties eingesetzt werden kann, und die

Abb. 2.12: Karten auf dem Bildschirm ausgeben (Unterprogramm)

```
9000 REM ------------------------------
9010 REM ---      KARTEN AUSGEBEN     ---
9020 REM ------------------------------
9030 :
9040 REM ------------------------------
9050 REM ---        PARAMETER         ---
9060 REM ---                          ---
9070 REM --- FARBE     : ROT/SCHWARZ ---
9080 REM --- HOEHE     : KARTENHOEHE ---
9090 REM --- ART       : CODE FARBE  ---
9100 REM --- N         : FENSTERNR.  ---
9110 REM --- X: X-POS. LINKS OBEN    ---
9120 REM --- Y: Y-POS. LINKS OBEN    ---
9130 REM ------------------------------
9140 :
9150 REM ---    PARAMETER AUSWAEHLEN ---
9160 :
9170 H3=(SPIELER-1)*3+H2
9180 HOEHE=KARTEN(H3,1)
9190 IF KARTEN (H3,2)=1 THEN FARBE=0 : ART=226
9200 IF KARTEN (H3,2)=2 THEN FARBE=0 : ART=229
9210 IF KARTEN (H3,2)=3 THEN FARBE=1 : ART=228
9220 IF KARTEN (H3,2)=4 THEN FARBE=1 : ART=227
9230 N=H2
9240 H$=HOEHE$(HOEHE)
9250 :
9260 REM --- ERSTE ZEILE --------------
9270 :
9280 PEN #N,3
9290 LOCATE #N,X,Y
9300 PRINT #N,CHR$(150);CHR$(154);CHR$(154);CHR$(154);CHR$(154);CHR$(1
54);CHR$(156)
9310 :
9320 REM --- ZWEITE ZEILE -------------
9330 :
9340 LOCATE #N,X,Y+1
9350 PRINT #N,CHR$(149);
9360 PEN #N,FARBE
9370 PRINT #N,H$;"    ";H$;
9380 PEN #N,3
9390 PRINT #N,CHR$(149)
9400 :
9410 REM --- DRITTE ZEILE -------------
9420 :
9430 LOCATE #N,X,Y+2
9440 PRINT #N,CHR$(149);
```

```
9450 PEN #N,FARBE
9460 IF HOEHE=3 THEN PRINT #N,"  ";CHR$(ART);"  "; : GOTO 9490
9470 IF HOEHE>5 THEN PRINT #N," ";CHR$(ART);" ";CHR$(ART);" "; : GOTO
9490
9480 PRINT #N,"      ";
9490 PEN #N,3
9500 PRINT #N,CHR$(149)
9510 :
9520 REM --- VIERTE ZEILE -------------
9530 :
9540 LOCATE #N,X,Y+3
9550 PRINT #N,CHR$(149);
9560 PEN #N,FARBE
9570 IF HOEHE=2 THEN PRINT #N,"  ";CHR$(ART);"  "; : GOTO 9600
9580 IF HOEHE=4 OR HOEHE=5 OR HOEHE>7 THEN PRINT #N," ";CHR$(ART);" ";
CHR$(ART);" "; : GOTO 9600
9590 PRINT #N,"      ";
9600 PEN #N,3
9610 PRINT #N,CHR$(149)
9620 :
9630 REM --- FUENFTE ZEILE ------------
9640 :
9650 LOCATE #N,X,Y+4
9660 PRINT #N,CHR$(149);
9670 PEN #N,FARBE
9680 IF HOEHE=1 OR HOEHE=3 OR HOEHE=5 OR HOEHE=9 THEN PRINT #N,"  ";CH
R$(ART);"  "; : GOTO 9720
9690 IF HOEHE=6 OR HOEHE>9 THEN PRINT #N," ";CHR$(ART);" ";CHR$(ART);"
 "; : GOTO 9720
9700 IF HOEHE=7 THEN PRINT #N," ";CHR$(ART);CHR$(ART);CHR$(ART);" "; :
 GOTO 9720
9710 PRINT #N,"      ";
9720 PEN #N,3
9730 PRINT #N,CHR$(149)
9740 :
9750 REM --- SECHSTE ZEILE ------------
9760 :
9770 LOCATE #N,X,Y+5
9780 PRINT #N,CHR$(149);
9790 PEN #N,FARBE
9800 IF HOEHE=2 THEN PRINT #N,"  ";CHR$(ART);"  "; : GOTO 9830
9810 IF HOEHE=4 OR HOEHE=5 OR HOEHE>7 THEN PRINT #N," ";CHR$(ART);" ";
CHR$(ART);" "; : GOTO 9830
9820 PRINT #N,"      ";
9830 PEN #N,3
9840 PRINT #N,CHR$(149)
9850 :
9860 REM --- SIEBTE ZEILE -------------
9870 :
9880 LOCATE #N,X,Y+6
9890 PRINT #N,CHR$(149);
9900 PEN #N,FARBE
9910 IF HOEHE=3 THEN PRINT #N,"  ";CHR$(ART);"  "; : GOTO 9940
9920 IF HOEHE>5 THEN PRINT #N," ";CHR$(ART);" ";CHR$(ART);" "; : GOTO
9940
9930 PRINT #N,"      ";
9940 PEN #N,3
9950 PRINT #N,CHR$(149)
9960 :
9970 REM --- ACHTE ZEILE --------------
9980 :
9990 LOCATE #N,X,Y+7
10000 PRINT #N,CHR$(149);
10010 PEN #N,FARBE
10020 PRINT #N,H$;"    ";H$;
10030 PEN #N,3
10040 PRINT #N,CHR$(149)
10050 :
10060 REM --- NEUNTE ZEILE -------------
10070 :
10080 LOCATE #N,X,Y+8
10090 PRINT #N,CHR$(147);CHR$(154);CHR$(154);CHR$(154);CHR$(154);CHR$(
154);CHR$(153)
10100 RETURN
10110     RETURN
```

Abb. 2.12: Karten auf dem Bildschirm ausgeben (Fortsetzung)

Schadenfreude sehr groß geschrieben wird. Selbst wenn Sie links ein As und rechts einen König haben, kann immer noch etwas schiefgehen, wobei die letzten Spieler - gutes Gedächtnis vorausgesetzt - wegen der bereits gefallenen Karten einen kleinen Vorteil haben.

Als Übung könnten Sie vielleicht diesen Umstand beheben, indem Sie nach jedem Durchlauf den ersten Spieler an die letzte Position rücken (Vertauschen von Name und Kapital, indem Sie eine Schleife von 0 bis MI-2 laufen lassen und anschließend die Werte aus den Elementen 0 der Felder in diejenigen mit dem Index MI transferieren).

Variablenübersicht

Variablenübersicht für ZWISCHEN 1000 – 10110
Zweck: Kartenspiel

Variablen

Name	Art	Bereich	Bedeutung
A$	H	J, j, N, n	Abfrage Spielfortsetzung
ART	H/L	226,. . .,229	Kartensymbol
EINSATZ	G	1. . .POTT	Einsatz eines Spielers
FARBE	H/L	0,1 (Schw./Rot)	Kartenfarbe
FUELLEN	H	1. . .MI	Laufvariable zum Füllen des Potts
H$	H/L	A,. . .,K	aktuelle Kartenhöhe
H1	H	1. . .MI*3 – 2	Hv. für Position der Karten eines Spielers in KARTEN( )
H2	H/E	Integer	Hv. zur Kartenausgabe
H3	H/L	1. . .MI*3 – 2	Hv. für Position der Karten eines Spielers in KARTEN( )
HI	H	1. . .MI*3 – 2	Hv. für Position der Karten
HOEHE	G	1. . .13/A,. . .,K	aktuelle Kartenhöhen
HOEHE$( )	G	1. . .13/A,. . .,K	Kennzeichnung der Kartenhöhen
I	H	1. . .MI/1. . .13 1. . .7/1. . .MI*3	Laufvariable
I	H	1. . .I – 1	Laufvariable
KAPHILF( )	H	1. . .MI/1. . .MI*100	Hilfsfeld zum Sortieren
KAPITAL( )	G	1. . .MI/1. . .MI*100	Kapital der einzelnen Spieler
KARTEN( )	G/E	1. . .3*MI/1. . .13	Karten aller Spieler
MAXIMUM	H	– 10000. . .MI*100	Hv. zum Sortieren (Rangliste)
MI	G	Integer	Anzahl der Mitspieler
N	H/E	Integer	Nummer der auszugebenden Kar.
NAME$( )	G	1. . .MI/Strings	Namen der Spieler

Variablenübersicht für ZWISCHEN 1000 – 10110
Zweck: Kartenspiel

Variablen

Name	Art	Bereich	Bedeutung
POTT	G	1. . .100*MI	Pott/Maximaler Einsatz
SPIELER	G	1. . .MI	Laufvariable für eine Runde
VERDECKT$	G	70* CHR$(207)	Darstellung verdeckte Karte
WARTE	H	1. . .2000/3000	Laufvariable für Warteschl.
WO	H	1. . .MI	Hv. zum Sortieren (Rangliste)
X	E	2,3	X-Koordinate für Kartenpos.
Y	E	2	Y-Koordinate für Kartenpos.

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
6060	Karten sortieren	2150
7000	Pott füllen	4280
8000	Rangliste erstellen	5030
9000	Karten ausgeben	3080, 3170, 3340

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Vorspann/Spielerdaten/Bildschirm/Kartenhöhen
2000	Karten verteilen
3000	Spielerschleife/Ausgabe der Karten/Einsatz
4000	Auswertung
5000	Rangliste/Abfrage für weitere Runde
6000	Karten sortieren
7000	Pott füllen
8000	Rangliste sortieren und ausgeben
9000	Karten ausgeben

Abb. 2.13: Variablenübersicht für ZWISCHEN

DER ZEICHENGENERATOR

Die Zeichen, die der Schneider CPC darstellen kann, bestehen alle aus Punkten, die in einer 8×8-Matrix angeordnet sind. Einzelne Zeilen dieser Matrix werden als achtstellige Binärzahlen codiert und können so im Speicher als ein Byte abgelegt werden. Mit acht aufeinanderfolgenden Bytes kann man also ein ganzes Zeichen darstellen. Ein Beispiel:

1. Zeile	00011000 =	24
2. Zeile	00111100 =	60
3. Zeile	01100110 =	102
4. Zeile	01100110 =	102
5. Zeile	01111110 =	126
6. Zeile	01100110 =	102
7. Zeile	01100110 =	102
8. Zeile	00000000 =	0

Das oben dargestellte Zeichen ist, wie Sie sicher schon erkannt haben, ein A. Wenn Sie mit dem PEEK-Befehl die Speicherstellen, in denen das Muster für das A gespeichert ist, auslesen, erhalten Sie die rechts daneben stehenden Zahlen. Die Reihenfolge, in der die Zeichenmuster gespeichert sind, entspricht bis zum 126sten Zeichen dem ASCII-Code (American Standard Code for Information Interchange). Die Zeichen 127 bis 255, die der Schneider CPC besitzt, sind nicht im ASCII-Code enthalten. Über die Reihenfolge können Sie sich im Handbuch auf den Seiten 1 bis 13 im Anhang III informieren.

Die gesamten Zeichenmuster sind im ROM von &3800 (dezimal 14592; & steht wie im Handbuch für hexadezimal) bis &3FFF (dezimal 16383) gespeichert. Diesen Speicherbereich nennt man auch Zeichengenerator. Das erste Zeichen im ASCII-Code, das normal (siehe Kapitel 2) auf dem Bildschirm dargestellt werden kann, ist das Leerzeichen (Space). Im Speicher stehen somit acht Nullen. Der Code für das Leerzeichen ist dezimal 32 oder &20. Wenn der Computer die Anweisung erhält, einen bestimmten Buchstaben oder sonst ein Zeichen auf dem Bildschirm auszugeben, nimmt er den Code des Zeichens, multipliziert das Ergebnis mit acht und addiert die erste Adresse des Zeichengenerator-ROMs dazu.

Als Ergebnis enthält der Rechner die erste Adresse des Musters des Zeichens, das auf dem Bildschirm ausgegeben werden soll. Der Inhalt dieser und der folgenden sieben Adressen werden so in den Bildschirmspeicher übertragen, daß sie auf dem Bildschirm untereinander erscheinen und zusammen das Zeichen bilden.

Die Funktionsweise kann man sich in einem kleinen Programm veranschaulichen. Dazu liest man die Speicherstellen, die das Zeichenmuster enthalten, mit PEEK aus und wandelt mit der BIN$-Funktion die ausgelesenen Bytes in Binärzahlen um, die man untereinander auf den Bildschirm bringt. Allerdings ist es nicht möglich, mit dem PEEK-Befehl Speicherstellen im ROM zu lesen. Dieses Problem läßt sich jedoch mit dem Befehl SYMBOL AFTER 32 lösen (der SYMBOL AFTER-Befehl funktioniert nicht mehr, wenn vorher mit MEMORY das Ende des BASIC-RAMs verschoben wurde). Mit diesem

Befehl überträgt man den gesamten Zeichensatz aus dem ROM ins RAM ab Adresse &A500 (dezimal 42240). Das Betriebssystem holt sich dann die Zeichenaus dem RAM. Das Programm könnte jetzt etwa folgendermaßen aussehen:

```
10 SYMBOL AFTER 32
20 CLS
30 INPUT"Zeichennummer ";n
40 adr = (n - 32)*8 + 42240
50 FOR i = 0 to 7
60       x = PEEK(adr + i)
70       PRINT BIN$(x,8)
80 NEXT
90 PRINT
100 GOTO 30
```

Zur besseren Lesbarkeit der Zeichen kann man das Programm noch etwas abändern:

```
60 x$ = BIN$(PEEK(adr + i),8)
70 FOR ii = 1 to 8
71       IF MID$(x$,ii,1) = "1" THEN PRINT "*";
                                ELSE PRINT " " ;
72 NEXT
73 PRINT
```

Wenn Sie das Programm gestartet haben, können Sie einen Code für ein Zeichen eingeben (nur Werte von 32 bis 255 sind empfehlenswert). Es erscheint das Bitmuster des Zeichens.

Die Zeichen, die jetzt im RAM stehen, können natürlich verändert werden. Entweder benutzt man dazu den SYMBOL-Befehl, oder man setzt einfach neue Werte mit POKE in die entsprechenden Speicherzellen, z. B.:

```
SYMBOL AFTER 32
POKE 42240,255
```

Wenn Sie jetzt die Leertaste drücken, erscheint ein waagerechter Strich auf dem Bildschirm. Um die richtige Speicherzelle zu finden, kann man wieder die obige Formel verwenden. Das Schneider-BASIC hat aber auch einen speziellen Befehl, mit dem man die Zeichen verändern kann, das ist der schon erwähnte SYMBOL-Befehl. Wie Sie vielleicht schon aus dem Handbuch wissen, verlangt der SYMBOL-Befehl neun Parameter, die Zeichennummer und acht Byte, die das Muster des Zeichens ergeben. Der Befehl macht nichts anderes, als mit der Zeichennummer die erste Adresse des Zeichens zu berechnen und die acht folgenden Zahlen ab dieser Adresse abzuspeichern. Für die Anwendungsmöglichkeiten eines geänderten Zeichensatzes folgt nun ein kleines Beispiel.

Senkrechte Schrift

Die Zeichen 128 bis 255 des Schneider-Zeichensatzes sind Sonderzeichen, die für die normale Textausgabe nicht gebraucht werden und auch gar nicht mit einem Tastendruck zu erreichen sind. Diese Zeichen bieten sich zum Ändern geradezu an. Das folgende Programm belegt den „hinteren" Teil des Zeichengenerators mit Zeichen, die um 90 Grad gegen den Uhrzeigersinn gedreht sind. Mit diesen Zeichen kann man senkrecht von unten nach oben schreiben. Das kann ganz nützlich sein, um etwa Koordinatenachsen zu beschriften.

Die Druckposition muß dabei für jedes Zeichen einzeln berechnet und der Cursor jedesmal erst mit LOCATE an die richtige Stelle gebracht werden. Die gedrehten Zeichen erhalten Nummern, die um 128 höher sind als die Nummern der Originalzeichen. Das Programm können Sie in ein eigenes Programm einbauen. Die Zeilennummern können mit dem RENUM-Befehl leicht angepaßt werden. An das eigentliche Programm, das die Zeilen 1000 bis 1190 umfaßt, ist ein Beispielprogramm angehängt, das den Text, der der Variablen A$ zugeordnet ist, senkrecht auf dem Bildschirm ausgibt.

```
1000 REM ------------------------------
1010 REM -Unterprg.senkrecht schreiben-
1020 REM ------------------------------
1030 :
1040 basis=41984
1050 SYMBOL AFTER 32
1060 :
1070 FOR c=32 TO 125
1080    :
1090    FOR byte=0 TO 7
1100       b=0
1110       :
1120       FOR bit=7 TO 0 STEP -1
1130          IF PEEK(basis+8*c+7-bit) AND 2^byte THEN b=b+2^bit
1140       NEXT bit
1150       :
1160       POKE basis+8*(c+128)+byte,b
1170    NEXT byte
1180    :
1190 NEXT c
1200 :
1210 REM ------- Ende Unterprg. -------
1220 :
2000 REM -------    Beispiel    -------
2010 :
2020 a$="Das ist ein Beispiel!"
2030 l=LEN(a$)
2040 :
2050 CLS
2060 :
2070 FOR i=1 TO l
2080    c$=CHR$(ASC(MID$(a$,i,1))+128)
2090    LOCATE 10,25-i
2100    PRINT c$
2110 NEXT
2120 :
2130 END
```

Abb. 2.14: Senkrechte Schrift (Unterprogramm)

Programmbeschreibung

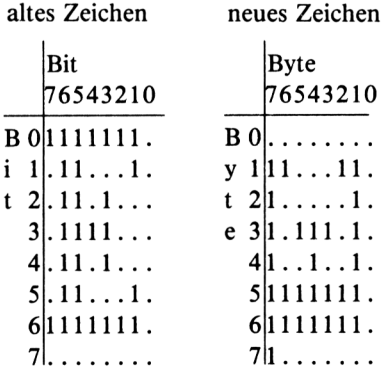
Zeile 1040: Die Variable basis enthält die Startadresse des Zeichengenerators minus 32×8. Dadurch vereinfacht sich die Berechnung der Startadresse für ein Zeichen etwas. Diese Berechnung wird im Programm knapp 6000mal ausgeführt, so daß sich diese Vereinfachung schon bemerkbar macht. Noch etwas schneller geht es, wenn man den Ausdruck „basis + 8*c" einer Variablen zuordnet (in Zeile 1060 einfügen) und im weiteren Programm dann anstatt dieses Ausdrucks die Variable verwendet.

Zeile 1050: Der gesamte Zeichensatz wird ins RAM kopiert.

Zeile 1070: Hier beginnt die Schleife, die die Zeichencodes durchzählt. Es wird bei Zeichen 32 begonnen, und bis Zeichen 125 werden alle Zeichen um 90 Grad gedreht. Wenn im Programm der Zeichensatz noch nicht verändert worden ist, so wie hier, kann man die Schleife auch erst mit 33 beginnen lassen.

Zeile 1090 -1170: Die Variable byte bezieht sich auf die neuen Zeichen. Wenn im alten Zeichen z. B. im dritten Byte von oben das fünfte Bit von links gesetzt ist, muß im neuen Zeichen im (9-5)ten Byte das dritte Bit von links gesetzt werden. Wenn ein Byte des neuen Zeichens gefüllt werden soll, geht das Programm in der nächsten Schleife die Bits im Byte des neuen Zeichens von links nach rechts durch und stellt fest, ob im Originalzeichen das Bit an der Stelle, die byte angibt, in dem Byte, das der Ausdruck „7-bit" angibt, gesetzt ist. Ist dies der Fall, dann wird im neuen Zeichen im Byte „byte" das Bit „bit" gesetzt.

Das hört sich komplizierter an, als es ist. Die untenstehende Grafik verdeutlich den Zusammenhang etwas besser.



Bevor die nächste Schleife, in der das neue Bit zusammengesetzt wird, begonnen werden kann, muß die Variable b, die das neue Byte enthalten soll, auf Null gesetzt werden.

Zeile 1130: Dann wird in der Schleife geprüft, ob im Byte an der Stelle „7-bit" das Bit, das dem dezimalen Wert (2^{byte}) entspricht, gesetzt ist, indem das zu prüfende Byte mit dem Wert (2^{byte}) durch AND verknüpft wird. Kommt bei der Verknüpfung etwas heraus, wird zum neuen Byte b der Wert 2^{bit} addiert.

Zeile 1140 - 1190: Mit „NEXT bit" wird die innere Schleife geschlossen. Wenn diese einmal durchlaufen ist, wird das fertige neue Byte in den Zeichengenerator eingebaut. Das geschieht in Zeile 1160. Soll das Programm als Unterprogramm in ein anderes Programm eingefügt werden, muß nach „NEXT c" in Zeile 1190 ein RETURN folgen. Das eigentliche Programm ist hier beendet.

Auf die Zeilen 2000 bis 2130 folgt ein Beispiel, das zeigt, wie der umgeformte Zeichensatz zu benutzen ist.

Zeile 2020 - 2030: Die Variable a$ enthält den Text, der senkrecht auf den Bildschirm geschrieben werden soll. Für die Schleife, die die Zeichen ausgibt, wird noch die Länge des Textes benötigt.

Zeile 2070 - 2110: Aus dem Text wird Zeichen für Zeichen herausgeholt, und von diesem Zeichenwird mit der ASC-Funktion der Code ermittelt. Zu diesem Code wird 128 addiert, das ergibt die Nummer des gedrehten Zeichens. Mit der CHR$-Funktion wird der Variablen c$ das Zeichen, das zu der ermittelten Nummer gehört, zugeordnet. Für jedes Zeichen, das gedruckt werden soll, muß der Cursor neu positioniert werden.

127 Zeichen je Bildschirmzeile

Wer mit dem Computer oft große Tabellen erstellt und diese auf dem Bildschirm darstellen will, der wird feststellen, daß selbst achtzig Zeichen, die der CPC im MODE 2 auf dem Schirm anzeigen kann, für derartige Anwendungen etwas wenig sind. Auch wenn man Grafiken beschriften will, wünscht man sich vielleicht eine kleinere Schrift, um z. B. Koordinatenachsen ausführlicher beschriften zu können.

Deshalb stellen wir hier ein Programm vor, das „das Letzte" aus dem Rechner herausholt. Den Besitzern eines CPC mit Farbmonitor empfehlen wir,

dieses Kapitel zu überblättern, da schon die achtzig Zeichen pro Zeile im MODE 2 auf dem Farbmonitor nicht mehr ohne weiteres klar zu lesen sind. Bei 127 Zeichen in einer Zeile ist dann mit ziemlicher Sicherheit nichts mehr zu erkennen.

Der Trick beim neuen Bildschirmmodus (Sie können ihn „MODE 4" nennen) besteht darin, einfach neue Zeichen, die nur vier Bit breit sind, zu definieren. Die Ausgabe der Zeichen geschieht im Grafikmodus (mit TAG). Dabei überlappen sich benachbarte Zeichen.

Da wir von links nach rechts schreiben, müssen die Zeichen ganz links in der Punktmatrix des Zeichenmusters stehen. Andernfalls würde ein Zeichen das vorherige zum Teil wieder löschen. Wir packen also immer ein Zeichen in die ersten vier Spalten der Punktmatrix und lassen die restlichen vier Spalten frei. Den ganzen Zeichensatz von Hand zu ändern, ist etwas mühselig. Für solch monotone Arbeiten ist schließlich der Computer da. Wir brauchen also erst einmal ein Programm, das aus den alten (Original-)Zeichen vier Spalten herausnimmt und den Rest im Zeichenmuster ganz nach vorne schiebt.

Das beste Ergebnis erhält man in den meisten Fällen bei den Spalten 2, 4, 5 und 7 (von links) eines Zeichens. Das sind die Bits mit den Werten 64, 16, 8 und 2. Das Programm hat also immer eine Byte aus dem Zeichengenerator zu holen und zu prüfen, ob die obengenannten Bits gesetzt sind (mit AND - wie gehabt).

Entsprechend diesen Bits sind dann in einer Variablen die Bits mit den Werten 128, 64, 32 und 16 zu summieren und das Ergebnis, also das neue Byte, im Zeichengenerator abzulegen. Hier ist eine Tabelle für die zusammengehörenden Bits:

	Bitwert			
alt	2	8	16	64
neu	16	32	64	128

Diese Tabelle ist im folgenden Programm in zwei Variablenfeldern gespeichert. In zwei verschachtelten Schleifen wird der Speicherbereich von 42240 bis 43000 (von Zeichen 32 bis 126) nach der obenstehenden Tabelle verändert. Das Ganze dauert etwa 25 Sekunden. Danach wird dieser Programmteil nicht mehr gebraucht.

Was die Ausgabe der Zeichen auf dem Bildschirm betrifft, ändert sich einiges. Nach Ausgabe eines Zeichens muß der Computer um 5 Pixels (Bildpunkte) nach rechts rücken, und der Zeilenvorschub ist erst nach 127 Zeichen fällig. Die Ausgabe samt Zeilenvorschub und Scrolling lassen wir von einem

Unterprogramm erledigen. Der PRINT-Befehl entfällt dann also; statt dessen heißt es GOSUB. Das Programm in Abb. 2.15 enthält das Unterprogramm für die Ausgabe und ein Beispiel, wie das Unterprogramm anzuwenden ist.

Das Ausgabeprogramm übernimmt den auszugebenden Text von der Variablen a$. Ein PRINT-Befehl im 127-Zeichenmodus sieht dann so aus:

```
a$ = " " : GOSUB . . .
```

Das entspricht PRINT "text", wobei der „text" direkt an den zuletzt geschriebenen Text angehängt wird. Die Cursorposition ist in den Variablen xcurs und ycurs gespeichert. Um einen Zeilenvorschub mit „Wagenrücklauf" zu erzwingen, muß man ycurs um eins erhöhen und xcurs auf Null setzen, bevor man in das Ausgabeprogramm springt.

Man kann den Cursor natürlich auch an eine beliebige Stelle auf den Bildschirm setzen, indem man den beiden Variablen die entsprechenden Werte zuweist (LOCATE). Zahlen werden wie folgt ausgedruckt:

```
a$ = STR$( x ) : GOSUB . . .
```

Abb. 2.15: Programm für Zeilenerweiterung

```
1000 REM ------------------------------
1010 REM ---  Darstellung von       ---
1020 REM ---  127  Zeichen / Zeile  ---
1030 REM ------------------------------
1040 :
1050 REM ------------------------------
1060 REM ---  Zeichensatz  aendern  ---
1070 REM ------------------------------
1080 :
1090 SYMBOL AFTER 32
1100 w(1)=2:w(2)=8:w(3)=16:w(4)=64
1110 v(1)=16:v(2)=32:v(3)=64:v(4)=128
1120 :
1130 FOR i= 42240 TO 43000
1140    byte=0:x=PEEK(i)
1150    :
1160    FOR k=1 TO 4
1170       IF (x AND w(k)) THEN byte=byte+v(k)
1180    NEXT k
1190    :
1200    POKE i,byte
1210 NEXT i
1220 :
1230 MODE 2
1240 ORIGIN 0,398
1250 :
2000 REM ------------------------------
2010 REM ---    Anwendungsbeispiel  ---
2020 REM ------------------------------
2030 :
2040 xcurs=10:ycurs=15:REM = LOCATE 10,15
2050 a$="Das ist ein Test! Mit diesem Programm kann man
 127 Zeichen in einer einzigen Zeile darstellen. Mit Fa
rbmonitor geht da nix mehr!          S T I M M T ´ S
???
2060 GOSUB 9000
2070 xcurs=0:ycurs=ycurs+1
2080 GOTO 2050
2090 END
2100 :
```

```
9000 REM ------------------------------
9010 REM --- Unterprg. fuer Ausgabe ---
9020 REM ------------------------------
9030    :
9040    l=LEN (a$)
9050    ORIGIN 0,398
9060    :
9070    FOR i=1 TO l
9080       IF ycurs>24 THEN ycurs=24:LOCATE 1,25:PRINT:
PRINT
9090       c$=MID$(a$,i,1)
9100       TAG
9110       MOVE xcurs*5,-16*ycurs
9120       PRINT c$;
9130       TAGOFF
9140       xcurs =xcurs+1:IF xcurs=128 THEN xcurs=0:ycu
rs=ycurs+1
9150       c$=MID$(a$,i,1)
9160    NEXT i
9170    :
9180    a$=""
9190 RETURN
9200 :
9210 REM ------------------------------
```

Abb. 2.15: Programm für Zeilenerweiterung (Fortsetzung)

Programmbeschreibung

Zeile 1090: Auf RAM-Zeichensatz schalten.

Zeile 1100 -1110: Den Variablen werden die oben genannten Tabellenwerte für die Änderung der Zeichen zugewiesen.

Zeile 1130 - 1210: Der Bereich von 42240 bis 43000 im Zeichengenerator wird geändert. Das sind Zeichen 32 bis 126. Bei Bedarf kann man den Bereich natürlich auch vergrößern.

Die Variable byte wird auf Null gesetzt, da sie die Summe der Bitwerte für das Zeichenmuster speichern soll. Die Variable x erhält ein Byte des Zeichengenerators, das in der folgenden FOR. . .NEXT-Schleife daraufhin untersucht werden soll, ob es die Bits mit den Werten 2, 8, 16 und 64 enthält.

Das neue Byte wird anstelle des alten im Speicher abgelegt (Zeile 1200). Wenn der Zeichensatz geändert ist, wird auf Bildschirmmodus 2 geschaltet. Um die Ausgabe etwas zu vereinfachen, wird der Ursprung der Bildschirmkoordinaten auf (0,398) (links oben) gelegt.

Zeile 2000 - 2100: Hier finden Sie ein Beispiel, das zeigt, wie das Ausgabeunterprogramm anzuwenden ist. Die Funktionsweise wurde vorher schon erläutert.

Zeile 9000 - 9210: Diese Zeilen enthalten das Unterprogramm, das die Textausgabe durchführt. Zuerst wird die Länge des Textes ermittelt, der dann in einer FOR . . .NEXT-Schleife ausgegeben wird. In dieser Schleife wird erst geprüft, ob gescrollt werden muß, um eine beschreibbare Zeile zu erhalten. Das ist immer dann der Fall, wenn die Zeilennummer größer als 24 ist, was entweder durch das Erzwingen eines Zeilenvorschubs im Hauptprogramm eintreten kann oder durch einen Zeilenvorschub, der im Unterprogramm selbst ausgelöst wurde, weil die vorherige Zeile voll war.

Dann wird aus dem Text ein einzelnes Zeichen mit MID$ herausgegriffen. Dieses Zeichen wird an der Stelle des Grafikcursors ausgegeben. Da im TAG-Modus gearbeitet wird, muß der PRINT-Befehl mit einem Semikolon abgeschlossen werden. Die Position des Grafikcursors wird aus den Koordinaten des (gedachten) Textcursors, die die Variablen xcurs und ycurs enthalten, im MOVE-Befehl ausgerechnet.

Nach Ausgabe des Zeichens wird wieder in den normalen PRINT-Modus zurückgeschaltet.

Zeichensatzeditor

Der frei definierbare Zeichensatz bietet natürlich noch weitere Möglichkeiten, z. B. die Ausgabe der deutschen Umlaute auf dem Bildschirm oder das Anlegen eines griechischen Zeichensatzes mit Großbuchstaben. An dieser Stelle sei jedoch ausdrücklich darauf hingewiesen, daß ein Drucker, auch der Schneider NLQ, seinen eigenen Zeichensatz besitzt, der mit dem Zeichensatzeditor natürlich nicht geändert werden kann. Als kleine Hilfe kann man das Text-Hardcopy-Programm aus Kapitel 3 verwenden, natürlich nur zur Ausgabe des Bildschirminhaltes.

Die deutschen Umlaute kann man etwa wie folgt programmieren:

```
10 SYMBOL AFTER 32
20 SYMBOL 123,102,0,120,12,124,204,118,0        REM ä
30 SYMBOL 124,102,0,60,102,102,102,60,0         REM ö
40 SYMBOL 125,102,0,102,102,102,102,62,0        REM ü
50 SYMBOL 91,102,24,60,102,126,102,102,0        REM Ä
60 SYMBOL 92,198,56,108,198,198,108,56,0        REM Ö
70 SYMBOL 93,102,0,102,102,102,102,60,0         REM Ü
80 SYMBOL 126,56,102,102,120,102,102,108,96     REM ß
```

Einen griechischen Zeichensatz auf diese Weise zu programmieren kann allerdings leicht in Arbeit ausarten. Erst muß man die einzelnen Zeichen entwer-

fen, dann muß man alle Bytes, also für jedes Zeichen acht (ca. 450 Berechnungen), ausrechnen, und zum Schluß muß das alles noch in den Computer getippt werden.

Da ist es sicher einfacher, ein Programm zu schreiben, mit dessen Hilfe man die Zeichen auf dem Bildschirm entwirft, das dann gleich die Bytes berechnet und den Zeichensatz auf Band/Diskette speichert. Der Zeichensatz läßt sich einfach auf dem Band ablegen, indem man den Speicherbereich von 42240 bis 44032 auslagert. Der Befehl dazu lautet:

```
SAVE "name",B,42240,1792
```

Das Programm, das den Zeichensatz verwendet, wird zuerst auf Band gespeichert und danach der Zeichensatz. Zu Anfang schreibt man:

```
SYMBOL AFTER 32
LOAD "!name"
```

Danach verwendet das Programm den vom Band geladenen Zeichensatz. Ein solches Programm, mit dem man einen Zeichensatz bequem entwerfen kann, möchten wir Ihnen hier vorstellen. Dieser Zeichensatzeditor enthält einige recht komfortable Befehle.

Funktionsbeschreibung

Zunächst soll allerdings darauf hingewiesen werden, daß das Programm möglicherweise nicht mehr läuft, wenn man die Farben ändert. Warum, läßt sich später besser erklären.

Wenn das Programm gestartet ist, werden dem Anwender drei Funktionen zur Auswahl angeboten. Man wählt durch Drücken der Tasten 1, 2 oder 3:

1. Laden eines Zeichensatzes von Band
2. Speichern des Zeichensatzes
3. Ändern des Zeichensatzes

Beim Laden des Zeichensatzes wird nach dem Namen des zu ladenden Zeichensatzes gefragt. Wenn kein Name eingegeben wird, lädt der Computer das erstbeste Programm. Ist dieses kein Zeichensatz, kann der Editor im Speicher gelöscht werden.

Das Speichern ist natürlich erst dann sinnvoll, wenn man schon etwas am Zeichensatz geändert hat.

Aus dem Programmteil, in die Zeichen editiert werden, kommt man an den Anfang zurück, indem man, wenn gerade ein Zeichen zum Editieren angezeigt wird, die Tasten CTRL und X (gleichzeitig) drückt.

Im Editierteil wird zuerst nach einem Zeichen gefragt, das verändert werden soll. Hier kann man ein beliebiges Zeichen eingeben, das über die Tastatur erreichbar ist, oder man drückt nur die ENTER-Taste. In diesem Fall erfragt der Rechner eine Zeichennummer. Es wird eine Zahl zwischen 31 und 256 erwartet.

Man kann auch ein Zeichen ganz neu entwerfen und dann an irgendeiner Stelle im Zeichensatz einfügen, indem man ein beliebiges Zeichen eingibt, dieses dann mit CTRL-L löscht und mit CTRL-N die Zeichennummer ändert. Ist das Zeichen ausgewählt, wird es in dem Feld links oben auf dem Bildschirm groß angezeigt. Rechts daneben stehen die Codes (die acht Bytes) des Zeichenmusters, noch weiter rechts werden das Zeichen in Originalgröße und die Nummer, ggf. auch die geänderte Nummer, des Zeichens angezeigt.

In der Bildschirmmitte werden in einer Zeile die schon geänderten Zeichen angezeigt, als Gedächtnisstütze bei längeren Editiersitzungen. Darunter sieht man, welche Funktionen wie aufgerufen werden können (Funktionenmenü).

Tippt man auf eine der Cursortasten, erscheint im Zeichenfeld ein Cursor, der mit den entsprechenden Tasten in alle vier Richtungen bewegt werden kann, ohne daß das Zeichen dabei verändert wird.

Die COPY-Taste hat zweierlei Funktion: Drückt man die COPY-Taste, wenn sich der Cursor über einem Punkt befindet, so wird der Punkt gelöscht. Befindet sich an der Cursorposition kein Punkt, so wird dort ein Punkt gesetzt. Der Cursor verschwindet dabei und erscheint erst wieder, wenn er weiterbewegt wird.

Die Taste CTRL-L löscht das ganze Feld. Mit CTRL-L und danach CTRL-I wird das ganze Feld mit Punkten gefüllt. CTRL-I allein invertiert das angezeigte Zeichen. Wenn Sie in Spiegelschrift schreiben wollen, können Sie mit CTRL-S das Zeichen an seiner senkrechten Achse spiegeln. CTRL-D dreht das Zeichen um 90 Grad im Uhrzeigersinn. Mit diesen zwei Funktionen kann man alle Verdrehungen, die zu einem Zeichen führen, das in die 8 × 8-Matrix paßt (also nicht etwa eine 45-Grad-Drehung), bewerkstelligen. Ein Beispiel sehen Sie in Abb. 2.16.

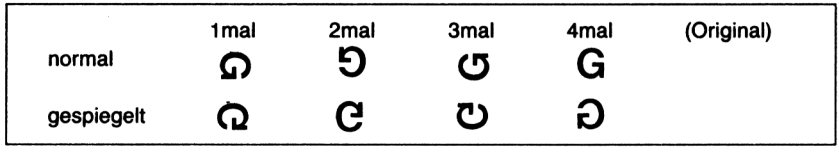


Abb. 2.16: Drehen eines Zeichens

Mit CTRL-C kann man sich die Bytes für das Muster (Bitmuster) rechts neben dem Editierfeld ausgeben lassen. Dies ist praktisch, wenn man in einem eigenen Programm nur wenige selbst definierte Zeichen verwenden will. Da wäre es zu umständlich, den ganzen Zeichensatz erst vom Band zu laden. Besser man notiert sich die Codes und gibt sie in einem SYMBOL-Befehl, wie im obenstehenden Beispiel die deutschen Umlaute, an.

Wenn man ein Zeichen geändert hat und es an einer anderen Stelle im Zeichensatz einfügen will, wie es beim senkrechten Zeichensatz gemacht wird, drückt man CRTL-N. Der Rechner fragt dann nach der neuen Nummer, unter der das Zeichen abgelegt werden soll. Mit ENTER wird das Zeichen (erst) im Zeichensatz gespeichert, und dann wird wieder nach einem Zeichen gefragt, das geändert werden soll. Mit CTRL-X gelangt man, wie schon beschrieben, in das Anfangsmenü und kann den Zeichensatz auslagern. Diese Taste kann man aber auch benutzen, wenn man es sich anders überlegt hat und das Zeichen doch nicht in den Zeichensatz übernehmen will. Auf alle anderen Tasten reagiert das Programm mit einem Piepston; ebenso wenn man mit dem Cursor aus dem Editierfeld heraus will.

Hier noch einmal die Funktionen in tabellarischer Form:

CTRL-D	: Zeichen drehen
CTRL-I	: Zeichen invertieren
CTRL-L	: Zeichen löschen
CTRL-N	: Zeichennummer ändern
CTRL-S	: Zeichen spiegeln
CTRL-X	: Zurück zum Menü
COPY	: Punkt im Zeichen invertieren

Abb. 2.17: Zeichensatzeditor

```
1000 REM ------------------------------
1010 REM ---     Zeichensatzeditor    ---
1020 REM ------------------------------
1030 :
1040 MODE 1
1050 :
1060 REM ------------------------------
1070 REM ---      Voreinstellung      ---
1080 REM ------------------------------
1090 :
1100 WINDOW #2,1,40,18,25
1110 WINDOW #3,1,40,15,15
1120 WINDOW #4,2,9,12,12
1130 WINDOW #1,2,9,4,12
1140 :
1150 INK 0,18:INK 1,0:INK 2,1:INK 3,24
1160 BORDER 0
1170 DIM b$(8),bl$(8)
1180 DIM b(8)
1190 SYMBOL AFTER 32
```

```
1200 cbasis=57505
1210 basis=42240-32*8
1220 PAPER #1,2:PEN #1,3
1230 PAPER #0,0:PEN #0,1
1240 cursor$=CHR$(202)
1250 punkt$=CHR$(233)
1260 hl$="   Z E I C H E N S A T Z E D I T O R    "
1270 :
1280 GOTO 3040
1290 :
2000 REM ------------------------------
2010 REM ---      Unterprogramme      ---
2020 REM ------------------------------
2030 :
2040 REM ------------------------------
2050 REM ---   inverse Darstellung   ---
2060 REM ------------------------------
2070 :
2080    PEN 0
2090    PAPER 1
2100 RETURN
2110 :
2120 REM ------------------------------
2130 REM ---  positive Darstellung   ---
2140 REM ------------------------------
2150 :
2160    PEN 1
2170    PAPER 0
2180 RETURN
2190 :
3000 REM ------------------------------
3010 REM ---       HAUPTPROGRAMM      ---
3020 REM ------------------------------
3030 :
3040 CLS
3050 GOSUB 2080
3060 PRINT hl$
3070 GOSUB 2160
3080 PRINT:PRINT "1 Zeichensatz laden"
3090 PRINT:PRINT "2 Zeichensatz speichern"
3100 PRINT:PRINT "3 Zeichensatz aendern"
3110 PRINT:PRINT "  ( mit  CTRL-X zurueck zum Menue )"

3120 a$=INKEY$
3130 IF a$<"1" OR a$>"3" THEN 3120
3140 ON VAL(a$) GOTO 8000,9000,4040
3150 :
4000 REM ------------------------------
4010 REM ---     Zeichen aendern     ---
4020 REM ------------------------------
4030 :
4040 CLS
4050 :
4060 GOSUB 2080
4070 PRINT hl$
4080 GOSUB 2160
4090 PRINT #3,zeichensatz$
4100 INPUT #2,"Zeichen ";zeichen$
4110 IF zeichen$="" THEN INPUT #2,"Zeichennummer";zeich
en$ ELSE GOTO 4140
4120 zeichen=VAL(zeichen$+CHR$(0)):zeichen$=""
4130 IF zeichen<32 OR zeichen>255 THEN 4040
4140 IF zeichen$="" THEN zeichen$=CHR$(zeichen) ELSE ze
ichen=ASC(zeichen$)
4150 :
4160 CLS #1
4170 :
4180 LOCATE 19,5
4190 PRINT"Zeichen: ";zeichen$
4200 LOCATE 19,6
4210 PRINT"Nummer :";zeichen
4220 :
4230 REM ------------------------------
4240 REM ---   Bitmuster anzeigen    ---
4250 REM ------------------------------
```

Abb. 2.17: Zeichensatzeditor (Fortsetzung)

```
4260 :
4270 FOR i=0 TO 7
4280    b(i)=PEEK(basis+zeichen*8+i)
4290    b$(i)=""
4300    :
4310    FOR ii=7 TO 0 STEP-1
4320       IF b(i) AND 2^ii THEN b$(i)=b$(i)+punkt$ ELS
E b$(i)=b$(i)+" "
4330    NEXT
4340    :
4350    PRINT #1,b$(i);
4360    LOCATE 12,4+i
4370    PRINT"      "
4380    LOCATE 12,4+i
4390    PRINT b(i)
4400    bl$(i)=b$(i)
4410 NEXT
4420 :
4430 CLS #4
4440 :
4450 IF PEEK(cbasis+80+2)=239 THEN punkt=1 ELSE punkt=0
4460 :
4470 REM ------------------------------
4480 REM ---      Menue anzeigen      ---
4490 REM ------------------------------
4500 CLS #2
4510 :
4520 PRINT #2," Cursor: ";CHR$(240);"  ";CHR$(241);"
";CHR$(242);"  ";CHR$(243);"         Punkt: COPY"
4530 PRINT #2," Menue:    CTRL-X     Drehen:   CTRL-D"

4540 PRINT #2," Spiegeln: CTRL-S     Loeschen: CTRL-L"

4550 PRINT #2," Invertieren:                   CTRL-I"
4560 PRINT #2," Bitmuster codieren:            CTRL-C"

4570 PRINT #2," Zeichennummer aendern:         CTRL-N"

4580 PRINT #2," Zeichen im Zeichensatz ablegen: ENTER"
4590 :
5000 REM ------------------------------
5010 REM ---      Editierroutine     ---
5020 REM ------------------------------
5030 :
5040 x=1:y=1
5050 :
5060 REM ---      Tastaturabfrage    ---
5070 :
5080 a$=INKEY$
5090 IF a$="" THEN 5080
5100 IF a$=CHR$(240) THEN 6020
     :REM auf
5110 IF a$=CHR$(241) THEN 6130
     :REM ab
5120 IF a$=CHR$(242) THEN 6240
     :REM links
5130 IF a$=CHR$(243) THEN 6350
     :REM rechts
5140 IF a$=CHR$(224) THEN 7380
     :REM copy
5150 IF a$=CHR$(13) THEN 7560
     :REM enter
5160 IF a$="D" THEN 6440
     :REM drehen
5170 IF a$="S" THEN 6710
     :REM spiegeln
5180 IF a$="I" THEN 6980
     :REM invertieren
5190 IF a$="C" THEN 7160
     :REM codieren
5200 IF a$="N" THEN 7450
     :REM Nummer aendern
5210 IF a$="L" THEN CLS #1:punkt=0:GOTO 5080
     :REM loeschen
5220 IF a$="X" THEN 3040
```

Abb. 2.17: Zeichensatzeditor (Fortsetzung)

```
     :REM menue
5230 PRINT CHR$(7);:GOTO 5080
5250 :
5260 :
6000 REM --------- Cursor auf ---------
6010 :
6020 IF y=1 THEN PRINT CHR$(7);:GOTO 5080
6030 LOCATE #1,x,y
6040 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
6050 y=y-1
6060 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6070 LOCATE #1,x,y
6080 PRINT #1,cursor$
6090 GOTO 5080
6100 :
6110 REM -------- Cursor ab -----------
6120 :
6130 IF y=8 THEN PRINT CHR$(7);:GOTO 5080
6140 LOCATE #1,x,y
6150 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "

6160 y=y+1
6170 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6180 LOCATE #1,x,y
6190 PRINT #1,cursor$
6200 GOTO 5080
6210 :
6220 REM -------- Cursor links --------
6230 :
6240 IF x=1 THEN PRINT CHR$(7);:GOTO 5080
6250 LOCATE #1,x,y
6260 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "

6270 x=x-1
6280 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6290 LOCATE #1,x,y
6300 PRINT #1,cursor$
6310 GOTO 5080
6320 :
6330 REM ------- Cursor rechts --------
6340 :
6350 IF x=8 THEN PRINT CHR$(7);:GOTO 5080
6360 LOCATE #1,x,y
6370 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "

6380 x=x+1
6390 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6400 LOCATE #1,x,y
6410 PRINT #1,cursor$
6420 GOTO 5080
6430 :
6440 REM ----------- Drehen -----------
6450 :
6460 LOCATE #1,x,y
6470 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
6480 :
6490 FOR i=1 TO 8
6500    bl$(i)=""
6510    :
6520    FOR ii=1 TO 8
6530       IF PEEK(cbasis+80*ii+2*i)=239 THEN bl$(i)=pu
nkt$+bl$(i) ELSE bl$(i)=" "+bl$(i)
6540    NEXT ii
6550    :
6560 NEXT i
6570 :
6580 CLS #1
6590 :
6600 FOR i=1 TO 8
6610    PRINT #1,bl$(i);
6620 NEXT i
```

Abb. 2.17: Zeichensatzeditor (Fortsetzung)

```
6630 :
6640 CLS #4
6650 :
6660 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6670 LOCATE #1,x,y
6680 PRINT #1,cursor$
6690 GOTO 5080
6700 :
6710 REM ---------- Spiegeln ----------
6720 :
6730 LOCATE #1,x,y
6740 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
6750 :
6760 FOR i=1 TO 8
6770    bl$(i)=""
6780    :
6790    FOR ii=1 TO 8
6800       IF PEEK(cbasis+80*i+2*ii)=239 THEN bl$(i)=pu
nkt$+bl$(i) ELSE bl$(i)=" "+bl$(i)
6810    NEXT ii
6820    :
6830 NEXT i
6840 :
6850 CLS #1
6860 :
6870 FOR i=1 TO 8
6880    PRINT #1,bl$(i);
6890 NEXT i
6900 :
6910 CLS #4
6920 :
6930 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
6940 LOCATE #1,x,y
6950 PRINT #1,cursor$
6960 GOTO 5080
6970 :
6980 REM -------- Invertieren  --------
6990 :
7000 LOCATE #1,x,y
7010 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
7020 :
7030 FOR i=1 TO 8
7040    FOR ii=1 TO 8
7050       LOCATE #1,ii,i
7060       IF PEEK(cbasis+80*i+2*ii)=239 THEN PRINT #1,
" " ELSE PRINT #1,punkt$
7070    NEXT ii
7080    :
7090 NEXT i
7100 :
7110 IF PEEK(cbasis+80*y+2*x)=239 THEN punkt=1 ELSE pun
kt=0
7120 LOCATE #1,x,y
7130 PRINT #1,cursor$
7140 GOTO 5080
7150 :
7160 REM ---------- Codieren ----------
7170 :
7180 LOCATE #1,x,y
7190 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
7200 :
7210 FOR i=1 TO 8
7220    b(i)=0
7230    :
7240    FOR ii=1 TO 8
7250       IF PEEK(cbasis+80*i+2*ii)=239 THEN b(i)=b(i)
+2^(8-ii)
7260    NEXT ii
7270    :
7280    LOCATE 12,3+i
7290    PRINT"      "
7300    LOCATE 12,3+i
```

Abb. 2.17: Zeichensatzeditor (Fortsetzung)

```
7310    PRINT b(i)
7320 NEXT i
7330 :
7340 LOCATE #1,x,y
7350 PRINT #1,cursor$
7360 GOTO 5080
7370 :
7380 REM --- Punkt setzen/loeschen  ---
7390 :
7400 LOCATE #1,x,y
7410 IF punkt=1 THEN punkt=0:PRINT #1," ";:GOTO 5080
7420 LOCATE #1,x,y
7430 IF punkt=0 THEN punkt=1:PRINT #1,punkt$;:GOTO 5080
7440 :
7450 REM ------- Nummer aendern -------
7460 :
7470 CLS #2
7480 :
7490 INPUT #2,"neue Nummer ";zeichen$
7500 zeichen=VAL(zeichen$)
7510 IF zeichen<32 OR zeichen>255 THEN 7470
7520 LOCATE 19,7
7530 PRINT"neue Nummer:";zeichen
7540 GOTO 4500
7550 :
7560 REM ------ Zeichen einfuegen -----
7570 :
7580 LOCATE #1,x,y
7590 IF punkt=1 THEN PRINT #1,punkt$ ELSE PRINT #1," "
7600 :
7610 FOR i=1 TO 8
7620    b(i)=0
7630    :
7640    FOR ii=1 TO 8
7650       IF PEEK(cbasis+80*i+2*ii)=239 THEN b(i)=b(i)
+2^(8-ii)
7660    NEXT ii
7670    :
7680    POKE basis+8*zeichen+i-1,b(i)
7690 NEXT i
7700 :
7710 zeichensatz$=RIGHT$(zeichensatz$+CHR$(zeichen),39)
7720 GOTO 4040
7730 :
8000 REM ------------------------------
8010 REM ---          Laden           ---
8020 REM ------------------------------
8030 :
8040 CLS
8050 :
8060 PRINT"Laden:"
8070 f$="":INPUT"Filename ";f$
8080 PRINT"Cassette einlegen !"
8090 WHILE INKEY$="":WEND
8100 a$=INKEY$
8110 LOAD f$
8120 GOTO 3040
8130 :
9000 REM ------------------------------
9010 REM ---        Speichern         ---
9020 REM ------------------------------
9030 :
9040 CLS
9050 :
9060 PRINT"Speichern:"
9070 f$="":INPUT"Filename ";f$
9080 PRINT"Kassette einlegen !"
9090 WHILE INKEY$="":WEND
9100 a$=INKEY$
9110 SAVE f$,b,basis,1792
9120 GOTO 3040
```

Abb. 2.17: Zeichensatzeditor (Fortsetzung)

Programmbeschreibung

Zeile 1100 -1130: Die Fenster werden festgelegt. In Fenster #2 wird das Menü angezeigt. Außerdem gehen alle Eingaben über Fenster #2. In Fenster #3 werden die geänderten Zeichen angezeigt. Ist die Zeile voll, wird nach links gescrollt. Window #1 ist das Feld, in dem die Zeichen editiert werden. Es ist acht Spalten breit und neun Zeilen lang, obwohl nur acht Zeilen gebraucht werden. Dies hat folgende Bewandtnis:

Wird in der letzten Spalte der letzten Zeile eines Fensters ein Zeichen gedruckt, scrollt das Betriebssystem den Inhalt des Fensters eine Zeile nach oben, wodurch das ganze Zeichen verschoben wird. Um das zu verhindern, werden neun Zeilen verwendet, wobei in der letzten Zeile garantiert nichts ausgegeben wird. Die Sache hat allerdings den Haken, daß das Zeichenfeld zu groß ist, was leicht irritiert und auch nicht schön aussieht. Das wiederum wird durch Fenster #4 behoben, das das Fenster #1 am unteren Rand verdeckt, so daß es so aussieht, als habe Fenster #1 acht Zeilen.

Zeile 1150 -1160: Die Tinten dürften zwar andere Farben erhalten, doch darf man dem Hintergrund (Paper) und der Schrift (Pen) im Window #1 keine anderen Tinten zuordnen, da sonst das Bitmuster, das über den Bildschirmspeicher abgetastet wird, vom Programm nicht mehr richtig erkannt wird: Andere Tinten ergeben im Bildschirmspeicher andere Bitkombinationen (bei PEEK erhaltene Werte). Die Rahmenfarbe kann man dagegen beliebig verändern.

Zeile 1170 -1180: Die Variablen, die später bei manchen Operationen die einzelnen Zeilen des Zeichenmusters enthalten, und die Variablen, in denen die codierten Bytes zusammengesetzt werden, werden dimensioniert.

Zeile 1190: Zeichensatz ins RAM kopieren und auf RAM-Zeichensatz umschalten.

Zeile 1200 -1210: Die Variable basis enthält die Basisadresse für Zugriffe auf den Zeichensatzgenerator. cbasis enthält die Basisadresse für die Erkennung des Bitmusters im Bildschirmspeicher. Zählt man zu dieser Adresse 80 dazu, ist man in der ersten Zeile in Fenster #1; zählt man nochmal zwei dazu, ist man in der ersten Spalte der ersten Zeile in Window #1. Mit der Formel

```
adresse = cbasis + 80*y + 2*x
```

erhält man die Adresse eines Bytes, das Auskunft darüber gibt, ob an der Position (x,y) (gleiche Koordinaten wie bei LOCATE #1) in Fenster #1 ein Punkt ist oder nicht. Ergibt PEEK(adresse) 239, dann ist an der Stelle (x,y) ein Punkt. Der Wert, den man bei PEEK(adresse) erhält, ändert sich, wenn andere Tinten verwendet werden.

Zeile 1240 -1260: Die Zeichen, die als Cursor und Punkt verwendet werden, sind nicht auf der Tastatur vorhanden und werden deshalb in Variablen abgelegt.

Zeile 2000 -2180: Zwei Unterprogramme für Umschaltung inverse/normale Darstellung.

Zeile 3000 -3110: Beginn des Hauptprogramms. Der Bildschirm wird gelöscht, und in die erste Zeile wird der Programmname invers geschrieben. Die verschiedenen Menüpunkte werden angezeigt.

Zeile 4040 -4210: Hier beginnt Menüpunkt 3. Die Anweisung in Zeile 4090 zeigt nach weiteren Programmdurchläufen die bereits editierten Zeichen in Fenster #3. Nachdem der Bildschirm wieder eingerichtet ist, fragt der Rechner nach dem Zeichen, das geändert werden soll. Nach der Eingabe enthält die Variable zeichen$ das Zeichen und die Variable zeichen dessen Nummer. In Zeile 4110 wird geprüft, ob ein Zeichen eingegeben wurde. Wenn nicht, wird nach einer Zeichennummer gefragt. Ist die eingegebene Zeichennummer nicht zulässig, beginnt die Abfrage von neuem. Wenn die Abfrage erfolgreich war, wird, je nachdem, ob ein Zeichen oder eine Nummer eingegeben wurde, die fehlende Variable ergänzt. Danach wird das Editierfeld mit CLS #1 gelöscht, und das Zeichen und die Zeichennummer werden daneben angezeigt.

Zeile 4230 -4450: In dem vorbereiteten Editierfeld muß das Bitmuster des Zeichens angezeigt werden. Dazu werden in einer Schleife die acht Bytes aus dem Zeichengenerator geholt und in einer weiteren Schleife „zerlegt". Die Adressen der Bytes werden, wie vorher schon beschrieben, aus der Basisadresse des Zeichengenerators und der Zeichennummer berechnet. In den Zeilen 4310 bis 4330 wird jedes Byte von links nach rechts auf gesetzte Bits untersucht. Mit 2^{ii} werden der Reihe nach die Binärzahlen 10000000, 1000000, 100000 usw. erzeugt, die dann durch AND mit dem Byte, das gerade untersucht wird, verknüpft werden. Ist das Ergebnis dieser Verknüpfung nicht Null, so ist das Bit an der ii-ten Stelle von

rechts gesetzt. In diesem Fall bekommt die Variable b$(i), die gewissermaßen ein Bild des untersuchten Bytes darstellt, einen Punkt angehängt.

Nach Beendigung dieser Schleife wird das Byte im Fenster #1 ausgegeben. Daneben soll der dezimale Wert des Bytes angezeigt werden. Da an dieser Stelle schon etwas stehen kann, wenn bereits andere Zeichen geändert worden sind, muß die alte Zahl gelöscht werden (mit PRINT" "), denn möglicherweise hat die neue Zahl weniger Stellen und überdeckt die alte Zahl nicht ganz, so daß von dieser ein Rest stehenbleibt.

Wenn die Schleife abgearbeitet ist und das ganze Zeichen im Fenster steht, wird noch die untere (die neunte) Zeile, die ja überflüssig ist, von FENSTER #4 verdeckt (CLS #4). Für den weiteren Programmablauf wird in Zeile 4450 geprüft, ob an der Cursorposition ein Punkt oder eine Leerstelle steht. Ist dort ein Punkt, erhält die Variable punkt den Wert Eins, sonst den Wert Null.

Zeile 4470 -4580: In der unteren Bildschirmhälfte (Fenster #2) werden die aufrufbaren Funktionen angezeigt.

Zeile 5000 -5230: Bevor die Tastaturabfrage beginnt, wird die Cursorposition wieder auf 1,1 (links oben) gesetzt. In Zeile 5080 beginnt die Tastaturabfrage.

Wenn ein Zeichen eingegeben wurde, durchläuft der Rechner den IF-Block in den Zeilen 5100 bis 5220, bis der Befehl erkannt wurde. Wenn eine falsche Eingabe gemacht wurde, gelangt der Rechner in Zeile 5230, wo ein Piepston ausgelöst wird. Nach CHR$(7) muß unbedingt ein Semikolon stehen, da sonst nach jedem CHR$(7) ein Zeilenvorschub erfolgt und der Bildschirminhalt irgendwann nach oben verschwindet.

In den Zeilen 5100 bis 5150 wird auf die Cursor-, COPY- und ENTER-Tasten geprüft, wie im Listing vermerkt ist. In den Zeilen 5160 bis 5220 wird auf Sonderfunktionen geprüft.

Wichtig: In den Zeilen 5160 bis 5220 stehen stehen im Listing in den Anführungszeichen normale Buchstaben. Diese Buchstaben müssen mit der CTRL-Taste eingegeben werden. Also bei D müssen Sie CRTL und D gleichzeitig drücken! Auf dem Bildschirm erscheinen dann ganz andere Zeichen. Bei diesen Zeichen steigt aber der Drucker aus. Deshalb stehen hier im Listing normale Buchstaben.

Zeile 6000 -6420: Diese vier Programmteile führen die Cursorbewegungen aus. Bei jedem Druck auf die Cursortasten muß die in den Variablen x und y gespeicherte Cursorposition verändert und dann der Cursor entsprechend versetzt werden. Zuerst muß aber geprüft werden, ob ein Versetzen des Cursors in die gewünschte Richtung überhaupt noch möglich ist, d. h. ob sich der Cursor nicht schon am Rand befindet.

Wenn z. B. y gleich 1 ist (Zeile 6020), dann ist der Cursor am oberen Rand. In diesem Fall wird ein CHR$(7) ausgegeben und zur Tastaturabfrage zurückgesprungen. Wenn eine Versetzung des Cursors möglich ist, dann wird an der augenblicklichen Cursorposition wieder das Zeichen eingesetzt, das dort war, bevor der Cursor an diese Stelle gesetzt wurde. Das kann entweder ein Punkt sein, wenn punkt gleich 1 ist, oder eine Leerstelle, wenn punkt 0 gleich ist. Dann wird die Cursorposition (x,y) verändert und geprüft, ob dort ein Punkt ist. Das geschieht, indem im Bildschirmspeicher nachgeschaut wird, ob bei der Adresse, die die Formel

cbasis + 80*y + 2*x

angibt, das für einen gesetzten Punkt typische Byte mit dem Wert 239 zugeordnet ist. Dann wird mit „LOCATE #1,x,y“ die Druckausgabe auf die Stelle (x,y) gesetzt und das im Vorspann gewählte Cursorzeichen ausgegeben.

Zeile 6440 -6690: Drehen des Zeichens um 90 Grad im Uhrzeigersinn. Zuerst muß der Cursor von der Bildfläche verschwinden, damit das Zeichen vollständig aus dem Bildschirmspeicher geholt werden kann. Dann wird jeweils eine Spalte des bestehenden Zeichens in eine Zeile b1$ des gedrehten Zeichens verwandelt. Diesmal wird das Bitmuster sozusagen von hinten aufgerollt. Die Spalten werden von oben nach unten überprüft. Wenn ein Punkt gefunden wird, dann wird an den Inhalt von b1$(i) links ein Punkt angefügt, andernfalls ein Leerzeichen. Wenn auf diese Weise alle Bytes „im Kasten“ sind, wird Fenster #1 gelöscht und in der Schleife von Zeile 6600 bis 6620 das gedrehte Zeichen ausgegeben. Zum Schluß wird wieder die neunte Zeile von Fenster #1 durch CLS #4 wegretuschiert und der Cursor, nachdem sich das Programm das Zeichen an der Cursorposition gemerkt hat, an die alte Stelle gesetzt.

Zeile 6710 -6960: Das Spiegeln läuft vom Programmtechnischen her im großen und ganzen genauso ab wie das Drehen. Nur das Kriterium dafür, ob im neuen Byte ein Punkt gesetzt wird oder nicht, ist anders. Das Programm prüft der Reihe nach die Bytes des angezeigten Zeichens von oben nach unten. Die Bits werden dabei von links nach rechts überprüft. Wird ein Punkt gefunden, dann wird an b1$(i) links ein Punkt angefügt, sonst eine Leerstelle.

Zeile 6980 -7160: Das Invertieren eines Zeichens ist etwas einfacher. Das Programm geht in zwei verschachtelten Schleifen das ganze Feld durch und gibt an der Stelle (i,ii) einen Punkt aus, wenn dort eine Leerstelle ist, und eine Leerstelle, wenn dort ein Punkt ist. Auch hier müssen natürlich wieder die Routinearbeiten wie das Verschwindenlassen des Cursors und das Merken des Zeichens an der Cursorposition durchgeführt werden.

Zeile 7160 -7360: Bei diesem Programmteil kann auf die eben erwähnten Routinearbeiten weitgehend verzichtet werden, da am Zeichen nichts verändert wird. Die einzelnen Bits werden genau wie beim Invertieren überprüft. Mit 2^i wird der Wert des überprüften Bits ermittelt. Die einzelnen Bits einer Zeile werden in b(i) aufaddiert. Wenn ein Byte fertig ist, wird der Wert des Bytes neben dem Bitmuster angezeigt.

Zeile 7380 -7430: Mit der COPY-Taste können Punkte in das Muster eingefügt oder gelöscht werden, je nachdem, ob an der Cursorposition ein Punkt ist oder nicht. Man nennt das auch „flippen".

Zeile 7450 -7540: Die Nummer des Zeichens wird geändert. Die Eingabe erfolgt über Fenster #2, das dafür gelöscht wird. Die Variable zeichen$, deren Inhalt sowieso nicht mehr gebraucht wird, wird hier benutzt, um Eingabefehler abzufangen. Wenn aus Versehen keine Zahl eingegeben wurde, ist VAL(zeichen$) gleich Null, und die Eingabe wird nicht angenommen. Ansonsten würde der Computer „Redo from Start" ausgeben. Da das Fenster #2 gelöscht wurde, wird nicht direkt zur Tastatur gesprungen, sondern über den Umweg der Menüanzeige.

Zeile 7560 -7720: Das Zeichen wird in den RAM-Zeichensatz eingefügt. Der Programmteil arbeitet genauso wie der Programmteil „codieren", jedoch werden die ermittelten Bytewerte nicht ausgedruckt, sondern in den Zeichensatz gepoked. Das neue Zeichen wird in die Variable zeichensatz$ aufgenommen und im Fenster #3 ange-

zeigt, wenn wieder nach Zeile 4040 zur Zeicheneingabe gesprungen wird. Die Länge der Variablen ist auf 39 Zeichen beschränkt, um ein Scrollen des Fensterinhaltes zu vermeiden. Wenn mehr als 39 Zeichen geändert wurden, werden nur die letzten 39 angezeigt.

Zeile 8000 -8130: Laden eines auf Kassette gespeicherten Zeichensatzes, danach Sprung zum Anfang des Hauptprogramms.

Zeile 9000 -9120: Der Zeichensatz wird unter dem Namen, den F$ enthält, abgespeichert, indem ab der Adresse basis einfach 1800 Bytes auf Band gespeichert werden.

Variablenübersicht

Variablenübersicht für ZEICHENSATZEDITOR 1000-9120

Variablen

Name	Art	Bereich	Bedeutung
A$	H	1 Zeichen	von Tastatur
B( )	G	0. . .255/1. . .8	Bytewert eines Bitmusters
B$( )	G	Bitmuster/ 1. . .8	je eine Zeile für diverse Zwecke
B1$( )	G	Bitmuster/ 1. . .8	je eine Zeile für diverse Zwecke
BASIS	G	41319	Basisadresse Zeichengenerator
CBASIS	G	57505	Basisadresse für Bitmustererkennung
CURSOR$	G	CHR$(202)	Cursorzeichen
F$	H	String	Dateiname
HL$	H	siehe Listing	Überschrift
PUNKT$	G	CHR$(233)	Zeichen für gesetzten Punkt in Bitmuster
X	H	1. . .8	aktuelle X-Position des Cursors im Bitmuster
Y	H	1. . .8	aktuelle Y-Position des Cursors im Bitmuster
ZEICHEN	G	0. . .255	Nummer des aktuellen Zeichens
ZEICHEN$	G	1 Zeichen	aktuelles Zeichen

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
2080	Inverse Darstellung	3050 (auch CHR$(24) möglich, vergleiche Kap. 2)

Zeile	Bedeutung	wird aufgerufen in Zeile
2160	Normale Darstellung	4060 3070 (siehe 2080), 4080

Programmaufteilung in Blöcke	
1. Zeile	Bedeutung
1000	Vorspann
2000	Unterprogramme für inverse und normale Darstellung
3000	Beginn Hauptprogramm (Menü)
4000	Zeichen ändern
4230	Bitmuster anzeigen
4470	Menü anzeigen
5000	Zeichen editieren
5060	Tastaturabfrage
6000	Cursor nach oben
6110	Cursor nach unten
6220	Cursor nach links
6330	Cursor nach rechts
6440	Zeichen drehen
6710	Zeichen spiegeln
6980	Zeichen invertieren
7160	Zeichen codieren
7380	Punkt setzen/löschen
7450	Nummer ändern
7560	Zeichen einfügen
8000	Zeichensatz laden
9000	Zeichensatz speichern

Abb. 2.18: Variablenübersicht für ZEICHENSATZEDITOR

Beispiel „Mondlandung"

Zum Schluß dieses Teils möchte ich noch zu einer ganz anderen Anwendung des frei definierbaren Zeichensatzes kommen, und zwar in einem Spiel. Im Originalzeichensatz sind ja schon ein paar Strichmännchen enthalten. Genau so gut kann man kleine Raumschiffe, andere Spielkartensymbole usw. entwerfen. Dabei kann man - zur größeren Darstellung - auch mehrere neu definierte Zeichen zusammenfassen. Die Bewegung allzu großer Zeichen erinnert allerdings stark an eine Raupe oder einen Tausendfüßler.

Wie man solche Zeichen einigermaßen flüssig über den Bildschirm bewegt, zeigt das folgende Programm. Es handelt sich um eine Version des altbekannten Mondlandespiels.

Zunächst die Spielregeln: Am unteren Bildschirmrand sieht man eine Mondlandschaft, die einige flache Stellen aufweist. Diese Stellen sind geeignete Landeplätze. Mit den Cursortasten muß man die Mondlandefähre zu einem Landeplatz steuern und vor dem Aufsetzen genügend abbremsen. Das Bremsen will aber gut eingeteilt sein, da der Treibstoff begrenzt ist. Alle Brems- und Lenkmanöver kosten Treibstoff. Sobald die Landschaft fertig gezeichnet ist, wartet der Computer auf einen Tastendruck als Startschuß. In der Zwischenzeit malt er Sternchen in den Mondhimmel. Nach dem ersten Tastendruck erscheint die Mondlandefähre am oberen Bildrand, und das Spiel geht los.

Es ist darauf zu achten, daß die Steuerung über Düsen erfolgt und eine Bewegung nach links oder rechts, die durch den Gebrauch einer Düse entstanden ist, so lange anhält, bis sie durch einen Strahl aus der gegenüberliegenden Düse wieder gestoppt wird. Nach links steuert man mit der Taste mit Pfeil nach links, nach rechts steuert man mit der gegenüberliegenden Taste im Cursorblock, und gebremst wird mit der Taste mit Pfeil nach oben.

Das Programm soll kein perfektes Telespiel sein, sondern ist als einfaches Beispiel für die Möglichkeiten gedacht, die die BASIC-Programmierung in Verbindung mit umdefinierten Zeichen bietet. Es bleibt dem Leser überlassen, das Programm nach eigenen Vorstellungen zu verbessern und neue Schikanen in das Spiel einzubauen.

Abb. 2.19: Mondlandespiel - Beispiel für geänderten Zeichensatz

```
1000 REM ------------------------------
1010 REM ---       MONDLANDUNG       ---
1020 REM ------------------------------
1030 :
1040 SPEED KEY 1,3
1050 KEY 128,"speed key 20,3"+CHR$(13)
1060 :
1070 MODE 1
1080 INK 0,0
1090 INK 1,26
1100 INK 2,15
1110 BORDER 0
1120 PAPER 0
1130 :
1140 SYMBOL 253,48,60,60,126,126,126,60,192
1150 :
1160 REM ----     Mondlandefaehre    ---
1170 SYMBOL 255,60,126,90,126,36,126,66,231
1180 mf$=CHR$(255)
1190 :
1200 REM ---      Raketenstrahl      ---
1210 SYMBOL 254,60,126,126,60,60,60,24,24
1220 rs$=CHR$(254)
1230 :
1240 REM ------------------------------
1250 REM --- Variablen vorbesetzen ---
1260 REM ------------------------------
1270 :
1280 DIM x0(26)
```

```
1290 dv=0.1
1300 h=390
1310 t=200
1320 x=320
1330 g=0.05
1340 a0$=CHR$(240)
1350 a1$=CHR$(242)
1360 a2$=CHR$(243)
1370 :
2000 REM ------------------------------
2010 REM ---  Landschaft zeichnen  ---
2020 REM ------------------------------
2030 :
2040 CLS
2050 :
2060 i=0
2070 n=0
2080 i=i+2
2090 a=INT(RND(8)*200+0.5)
2100 IF a>1 THEN 2190
2110 :
2120 REM ---  Landeplatz zeichnen  ---
2130 :
2140 MOVE i,0
2150 DRAW i+25,0,3
2160 x0(n)=i
2170 i=i+25
2180 n=n+1
2190 MOVE i,0
2200 DRAW i,a/25,3
2210 IF i<640 THEN 2080
2220 IF n=0 THEN 2040
2230 :
3000 REM ------------------------------
3010 REM ---  Warten auf Prg.start  ---
3020 REM ------------------------------
3030 :
3040 WINDOW #2,15,25,10,13
3050 PRINT #2,"MONDLANDUNG"
3060 PRINT #2,"Start mit S"
3070 :
3080 WHILE INKEY$=""
3090    x1=640*RND(8)
3100    y1=370*RND(8)+30
3110    PLOT x1,y1,1
3120    FOR i=1 TO 50:NEXT
3130 WEND
3140 :
3150 TAG
3160 :
3170 CLS #2
3180 :
4000 REM ------------------------------
4010 REM ---       Hauptprogramm      ---
4020 REM ------------------------------
4030 :
4040 a$=INKEY$
4050 IF dv=0 THEN 4090
4060 IF a$=a0$ THEN vy=vy-dv:t=t-1:SOUND 1,0,10,7,,,15
4070 IF a$=a1$ THEN vx=vx-dv:t=t-1:SOUND 1,0,10,7,,,15
4080 IF a$=a2$ THEN vx=vx+dv:t=t-1:SOUND 1,0,10,7,,,15
4090 vy=vy+0.02
4100 MOVE x,h
4110 PRINT" ";
4120 MOVE x,h-16
4130 PRINT" ";
4140 h=h-vy
4150 x=x+vx
4160 MOVE x,h
4170 PEN 1
4180 PRINT mf$;
4190 MOVE x,h-16
4200 PEN 2
4210 IF a$=a0$ AND dv>0 THEN PRINT rs$;
4220 IF t<0 THEN dv=0
4230 :
```

Abb. 2.19: Mondlandespiel - Beispiel für geänderten Zeichensatz (Fortsetzung)

```
4240 REM -- pruefen auf Bodenkontakt --
4250 :
4260 IF h>17 THEN 4040
4270 :
5000 REM ------------------------------
5010 REM --- Landung/Daten auswerten---
5020 REM ------------------------------
5030 :
5040 REM --- Geschwindigkeit pruefen---
5050 :
5060 IF vy>0.3 THEN 5410
5070 :
5080 REM ------------------------------
5090 REM ---      weiche Landung      ---
5100 REM ------------------------------
5110 :
5120 REM --- auf Landeplatz gel.?   ---
5130 :
5140 w=0
5150 :
5160 FOR i=1 TO n
5170    IF x>=x0(i) AND x<=x0(i)+9 THEN w=1
5180 NEXT
5190 :
5200 IF w=0 THEN 5410:REM Landepl verfehlt
5210 TAGOFF
5220 LOCATE 1,25
5230 PRINT
5240 SOUND 1,125000/400,50,7
5250 SOUND 1,125000/800,50,7
5260 SOUND 1,125000/1600,50,7
5270 :
5280 PRINT"            Gut  gelandet!"
5290 PRINT:PRINT"Geschwindigkeit beim Aufsetzen:";INT(v
y*100)/10;"m/s"
5300 PRINT"Verbrauchter Treibstoff:";200-t;"l"
5310  :
5320 SPEED KEY 20,2
5330 END
5340 :
5350 ********************************
5360 :
5370 REM ------------------------------
5380 REM ---      Bruchlandung       ---
5390 REM ------------------------------
5400 :
5410 FOR i=1 TO 200
5420    INK 0,26
5430    INK 0,0
5440 NEXT
5450 :
5460 ENV 1,1,15,1,15,-1,10
5470 ENT 1,1,-5,1,10,1,15
5480 SOUND 1,1000,0,15,1,1,15
5490 MOVE x,y:PRINT" ";
5500 :
5510 FOR i=1 TO 314 STEP 8
5520    y=90*SIN(i/100)
5530    MOVE x-i,y
5540    PRINT CHR$(253);
5550    MOVE x+i,y
5560    PRINT CHR$(253);
5570    MOVE x-i,y
5580    PRINT" ";
5590    MOVE x+i,y
5600    PRINT" ";
5610 NEXT
5620 :
5630 TAGOFF
5640 :
5650 PRINT"           BRUCHLANDUNG!!"
5660 PRINT
5670 :
```

Abb. 2.19: Mondlandespiel - Beispiel für geänderten Zeichensatz (Fortsetzung)

```
6000 REM ------------------------------
6010 REM ---      Unfallursache      ---
6020 REM ------------------------------
6030 :
6040 PRINT"Aufsetzgeschwindigkeit:";INT(100*vy)/10;"m/s
"
6050 IF vy>0.3 THEN PRINT"zu schnell !"
6060 PRINT"restlicher Treibstoff:";t;"l"
6070 IF w=0 AND vy<0.3 THEN PRINT"Ausserhalb von einem
Landeplatz auf ge-":PRINT "setzt !"
6080 :
6090 SPEED KEY 20,2
6100 END
6110 :
6120 ********************************
```

Abb. 2.19: Mondlandespiel - Beispiel für geänderten Zeichensatz (Fortsetzung)

Programmbeschreibung

Das Programm arbeitet mit der TAG-Funktion. Zur Erinnerung: Ist der TAG-Modus eingeschaltet, werden alle PRINT-Anweisungen an der Stelle, wo sich der Grafikcursor befindet, ausgeführt. Der Cursor wird also mit MOVE nach Grafikkoordinaten positioniert.

Noch ein wichtiger Hinweis: Eine Sonderfunktionstaste muß unbedingt mit dem Text „speed key 20,3" belegt sein! Die Belegung ist in Zeile 1050 schon vorgesehen. Am besten Sie geben die Zeile zuerst ein und machen mit der einzelnen Zeile einen Probelauf. Im Programm wird damit die Wiederholverzögerung für die Tasten abgeschaltet. Sollte beim Probelauf des Programms ein Fehler auftreten, wonach das BASIC in den Direktmodus schaltet, ist es nicht mehr möglich, etwas Vernünftiges einzugeben, da jeder auch noch so kurze Tastendruck gleich mehrere Zeichen produziert. In einem solchen Fall hilft nur noch die Sonderfunktionstaste.

Zeile 1040: Abschalten der Verzögerung für die Repeat-Funktion. Die Verzögerung würde sich beim Abbremsen des Raumschiffes unangenehm bemerkbar machen.

Zeile 1050: Belegen der Taste 0 im Ziffernblock als „Rettungsring", wie oben beschrieben.

Zeile 1070 -1120: Das Programm benutzt den MODE 1 mit 320 Punkten pro Zeile. Die Farben können beliebig gewählt werden.

Zeile 1140: Das hier verschlüsselte Zeichen stellt einen Teil der Mondlandefähre nach einer Bruchlandung dar.

Zeile 1170 -1180: Die Mondlandefähre wird definiert und in der Variablen mit mf$ abgelegt.

Zeile 1210 -1220: Wenn die Bremsdüse gezündet wird, sieht man unter der Mondlandefähre den Raketenstrahl aufleuchten (fast wie bei einer richtigen Rakete). Hier wird die Form des Strahls festgelegt und in rs$ abgelegt.

Zeile 1280 -1360: Dimensionieren und Vorbesetzen der Variablen.
x0$(1) enthält die Koordinaten des linken Randes der Landeplätze, die später durch Zufallszahlen ermittelt werden.

dv gibt die Schubkraft der Triebwerke an.

h ist die Höhe Oberkante der Mondlandefähre über dem Grund.

t ist der Treibstoffvorrat. Die Werte können je nach Erfahrung des Astronauten verändert werden.

x ist die horizontale Koordinate der Mondlandefähre.

g ist die Gravitation.

Die Variablen a0$, a1$ und a2$ dienen zum Vergleichen, welche Taste gedrückt wurde. Sie enthalten die Zeichen, die den Tasten für Cursor nach links, rechts und oben entsprechen.

Zeile 2040 -2220: Am unteren Rand des Bildschirms wird die Mondlandschaft gezeichnet. Sie besteht aus lauter eng beeinander stehenden, senkrechten Strichen unterschiedlicher Länge. Die Länge der Striche wird in Zeile 2090 ermittelt. Ist der Strich kleiner oder gleich eins, was nicht so oft vorkommt, dann wird ein Landeplatz in die Landschaft gezeichnet. Wenn nicht, dann wird der Strich gezeichnet und die Koordinaten um zwei weitergezählt; da im Modus 1 jeder Punkt zwei Bits breit ist, stehen die Striche ohne Zwischenraum nebeneinander.

Die Landeplätze sind jeweils 25 Bits breit, die Mondlandefähre im Modus 1 16 Bits. Es bleiben also 9 Bits Spielraum zum Landen. Jedesmal, wenn ein Landeplatz angelegt wird, merkt sich das Programm in x0Z, wo der Landeplatz ist, und zählt mit, wieviel Landeplätze es schon sind (n).

Stellt sich dann zum Schluß heraus, daß es null Landeplätze sind, wird einfach nochmal eine Landschaft gezeichnet, damit der Pilot auch wirklich eine Chance hat, den Adler zu landen.

Zeile 3040 -3060: Fenster #2 enthält sozusagen das Titelbild. Nach dem Start wird es gelöscht.

Zeile 3080 -3130: Die WHILE-Schleife wartet auf einen Tastendruck, um das Spiel zu starten. Solange keine Taste gedrückt wird, werden Sternchen an zufällig gewählten Punkten gezeichnet. Die FOR ... NEXT-Schleife in Zeile 3120 verzögert die Sache etwas.

Zeile 3150 -3170: Das Spiel geht jetzt los. Es wird in den TAG-Modus geschaltet, damit die Mondfähre in kleineren Schritten bewegt werden kann, und die Programmüberschrift wird gelöscht.

Zeile 4040 -4080: Tastaturabfrage. Wenn die Schubkraft dv der Triebwerke gleich Null ist, wird die Auswertung des Tastendrucks übersprungen. Normalerweise wird aber der Eingabe entsprechend die Geschwindigkeit in horizontaler und vertikaler Richtung verändert. Das Bremsen und Steuern verursacht ein Geräusch im Innern der Fähre, das mit dem SOUND-Befehl wiedergegeben wird. Bei jeder Geschwindigkeitsänderung bleibt ein Liter Treibstoff auf der Strecke.

Zeile 4090 -4220: Nach Durchlaufen der Tastaturabfrage wird die Sinkgeschwindigkeit um 0.02 erhöht. Das ist die Mondanziehungskraft. Das Raumschiff muß jetzt an die neue Position gebracht werden. Erst wird es an der alten Position gelöscht; sowohl das Raumschiff selbst, als auch der eventuell vorhandene, 16 Bits tieferliegende Raketenstrahl. Dann wird die neue Position berechnet und die Mondlandefähre dort wieder ausgedruckt. Wenn die Bremstaste gedrückt ist, also wenn (in Zeile 4210) a$ gleich a0$ ist und der Tank noch nicht leer ist, dann wird auch der Raketenstrahl unter dem Raumschiff angebracht. Es muß auch geprüft werden, ob noch Sprit im Tank ist. Ist das nicht der Fall, dann wird in Zeile 4220 die Schubkraft auf Null gesetzt.

Zeile 4260: Wenn Bodenkontakt besteht, wird zur Auswertung der Landedaten verzweigt. Bodenkontakt besteht dann, wenn die Oberkante der Mondfähre 17 Bits hoch liegt. Die Fähre ist 16 Bits hoch, 2 Bits ist der Landeplatz hoch, und 1 Bit sinkt die Fähre im Boden ein.

Zeile 5060: Hier wird überprüft, ob es eine weiche Landung war. Bei Aufsetzgeschwindigkeiten von mehr als 3 Meter pro Sekunde geht die Fähre zu Bruch. Das Programm verzweigt dann zum Programmteil „Bruchlandung".

Zeile 5140 -5330: Wenn die Landegeschwindigkeit ausreichend klein war, kann es dennoch vorkommen, daß der Pilot außerhalb eines Landeplatzes

landen wollte. Das ist nicht zulässig und gilt deshalb als Bruchlandung. Zunächst wird angenommen, der Pilot sei nicht vorschriftsmäßig gelandet und die Variable w, die nachher Aufschluß über die Zulässigkeit des Landegebietes geben soll, auf Null gesetzt.

In der Schleife von Zeile 5160 bis 5180 wird für jeden Landeplatz geprüft, ob die x-Koordinate des Raumschiffs innerhalb der Toleranz für den betreffenden Landeplatz liegt. Wenn ja, dann hat der Pilot den Landeplatz nicht verfehlt, und w wird auf 1 gesetzt.

Stellt sich bei der Überprüfung in Zeile 5200 heraus, daß kein Landeplatz richtig getroffen wurde, wird die Landung als Unfall gewertet und entsprechend verzweigt. Ist w gleich 1, dann wird der TAG-Modus ausgeschaltet, und ein Tonsignal zeigt die erfolgreiche Landung an.

Der Cursor wird in die letzte Zeile gesetzt. Damit scrollt der ganze Bildschirminhalt bei der Ausgabe des Kommentars zur Landung nach oben.

Zeile 5410 -5660: Die Bruchlandung wird optisch und akustisch angezeigt. In der Schleife in den Zeilen 5410 bis 5540 wird die Hintergrundfarbe 200mal hintereinander schnell auf 26, dann auf 0 geschaltet. Das geschieht schneller, als es auf dem Bildschirm angezeigt werden kann, wodurch ein scheinbar unkontrolliertes Flackern entsteht, das das Explodieren der Mondlandefähre anzeigen soll.

In den Zeilen 5460 bis 5480 wird ein Explosionsgeräusch simuliert (was allerdings nicht ganz den Tatsachen entspricht). In den Zeilen 5490 wird die Mondfähre gelöscht. Die folgende Schleife zeigt die davonfliegenden Bruchstücke der Fähre, die in Zeile 1140 definiert wurden. Diese fliegen auf einer Sinuskurve.

Das ist zwar falsch, aber erstens ist es einfach zu programmieren, und zweitens ist die Kurve hier kaum von einer richtigen Flugbahn zu unterscheiden (die Bildschirmkrümmung hilft auch etwas mit). Nachdem die Bruchstücke aufgeschlagen sind, wird mit TAGOFF in den normalen Druckmodus geschaltet und die Unfallursache genannt.

Zeile 6040 -6100: Auswertung der Flugdaten zur Bestimmung der Unfallursache.

Für Joystickbesitzer ist nachfolgend eine Programmänderung dargestellt, die die Verwendung eines (normalen) Joysticks zur Steuerung des Raumschiffs ermöglicht.

```
4040 a=JOY(0)
4050 IF dv=0 THEN 4090
4060 IF a AND 16 THEN vy=vy-dv:t=t-1:SOUND 1,0,10,7,,,1
5
4070 IF a AND 4 THEN vx=vx-dv:t=t-1:SOUND 1,0,10,7,,,15

4080 IF a AND 8 THEN vx=vx+dv:t=t-1:SOUND 1,0,10,7,,,15




4210 IF (a AND 16) AND dv>0 THEN PRINT rs$;
```

Abb. 2.20: Mondlandespiel - Änderungen für Joystick

Variablenübersicht

Variablenübersicht für MONDLANDUNG 1000-6120			
Variablen			
Name	Art	Bereich	Bedeutung
a	H	Dezimal	allgemeine Hilfsvariable
a$	H	1 Zeichen	von Tastatur
a0$	H	CHR$(240)	Vergleichszeichen
a1$	H	CHR$(242)	Vergleichszeichen
a2$	H	CHR$(243)	Vergleichszeichen
dv	G	Dezimal	Schubkraft
g	G	0.5	Gravitation
h	G	Integer	Höhere Mondlandefähre
i	H	Integer	Hilfsvariable/Laufvariable
mf$	G	CHR$(255) geändert	Landefähre
n	H	Integer	Anzahl der Landeplätze
rs$	G	CHR$(254) geändert	Raketenstrahl
t	G	Dezimal	Treibstoff
vx	G	Dezimal	Geschwindigkeit in X-Richtung
vy	G	Dezimal	Geschwindigkeit in Y-Richtung
w	H	Integer	Merker für „Landegüte“
x	G	0...320	horizontale Koordinate der Landeplätze

Name	Art	Bereich	Bedeutung
x0( )	G	1...320	linke Koordinaten der Landeplätze
x1	H	1...640	Hilfsvariable zum Zeichnen von Sternchen (X-Koordinate)
y1	H	0...400	Hilfsvaribale zum Zeichnen von Sternchen (Y-Koordinate)

Unterprogramme		
Zeile	Bedeutung	wird aufgerufen in Zeile
keine		

Programmaufteilung in Blöcke	
1. Zeile	Bedeutung
1000	Vorspann; Zeichen definieren
1240	Variablen besetzen
2000	Landschaft ausgeben
2120	Landeplatz ausgeben
3000	Programmstart erfragen
4000	Hauptprogramm
4040	Tastaturabfrage
4090	Landefähre neu positionieren
4260	Bodenkontakt?
5000	Landung
5060	Geschwindigkeit?
5080	weiche Landung
5140	auf Landeplatz
5410	Bruchlandung
6000	Unfallursache ausgeben

Abb. 2.21: Variablenübersicht für MONDLANDUNG (Fortsetzung)

Kapitel 3

Hochauflösende Grafik

Wie alle neueren Heimcomputer hat auch der Schneider CPC eine hochauflösende Grafik, dies jedoch mit besonders hervortretenden Merkmalen: Auflösung bis 640 × 200 Bildpunkte (Pixel) und bis zu sechzehn Farben gleichzeitig auf dem Bildschirm (dann allerdings nur noch mit 160 × 200 Bildpunkten).

Die Bedeutung von Bildpunkt und Bildschirmpunkt ist beim CPC nicht die gleiche. Beim Modus 0 können zwar bis zu sechzehn Farben dargestellt werden, jedoch werden vier Bildschirmpunkte zu einem Bildpunkt (Pixel/picture element) zusammengefaßt. Eine genauere Untersuchung über die Arbeitsweise des Rechners bei der Verwendung von Grafiken soll nicht Gegenstand dieses Buchs sein.

An dieser Stelle wollen wir uns vielmehr um die Anwendung der Grafik kümmern. Eingebettet ist auch eine kurze Übersicht über die Möglichkeit des Joysticks, der dann in Verbindung mit hochauflösender Grafik in einem Programm besprochen wird. Des weiteren werden verschiedene Formen von Diagrammen besprochen und auch eine Grafikeigenschaft, die zwar vielleicht von anderen Rechnern her bekannt ist, beim Schneider CPC jedoch nicht vorgesehen ist: Sprites. Den Abschluß bilden zwei Utilities zur Ausgabe von Hardcopies (Text und hochauflösende Grafik) vom Bildschirm auf den Drukker.

ALLGEMEINES

Da Text- und Grafikbildschirm beim CPC nicht getrennt sind, gilt der jeweils eingestellte Bildschirmmodus sowohl für Text als auch für Grafik. Wie schon beim Text sind folgende Modi möglich:

Modus 0: 160 × 200 Pixel mit maximal 16 Farben
Modus 1: 320 × 200 Pixel mit maximal 4 Farben
Modus 2: 640 × 200 Pixel mit maximal 2 Farben

Der CPC stellt eine Reihe von Grafikbefehlen zur Verfügung, die wir im folgenden nochmal kurz auflisten wollen:

DRAW, DRAWR:	Ziehen einer Linie zur angegebenen Koordinate von der aktuellen Cursorposition.
PLOT, PLOTR:	Ausgabe eines Punktes an der angegebenen Position.
MOVE, MOVER:	Grafikcursor an die angegebene Position bewegen.
ORIGIN:	Koordinatenkreuz positionieren und gegebenenfalls das Grafikfenster festlegen.
TEST, TESTR:	Prüfen, ob Punkt an der angegebenen Koordinate gesetzt ist.
XPOS, YPOS:	Aktuelle Koordinaten des Grafikcursors abfragen.

Die Befehle mit einem R am Ende geben die relative Position (von der aktuellen Cursorposition gesehen) für den jeweiligen Befehl an. So ist es beim CPC möglich, einerseits eine feste Koordinate in einem Befehl vorzugeben, aber andererseits auch eine Entfernung von der aktuellen Cursorposition aus anzugeben. Nach der Ausführung eines jeden Grafikbefehls befindet sich der Grafikcursor an der zuletzt ausgegebenen Position. Dies vereinfacht besonders die Arbeit z. B. bei Liniendiagrammen, wie wir später noch sehen werden.

Die Farben für Hintergrund und Grafikstift gelten analog den eingestellten Farben für den Textmodus.

Bei den Befehlen können zwar ungültige Koordinaten (z. B. PLOT 800,800,1) angegeben werden, es erfolgt jedoch weder eine Ausgabe noch eine Fehlermeldung. Achten Sie auf ungültige Koordinaten besonders beim Testen von Routinen.

Nun kommen wir jedoch zu einigen Anwendungsbeispielen für die Grafik.

GRAFIK UND JOYSTICK

Zunächst noch etwas zum Thema Joystick. Wenn Sie im Ready-Modus den Joystick bewegen, so erscheinen auf dem Bildschirm für die vier Hauptrichtungen oben, unten, links und rechts jeweils die spitzen Pfeile, wie wir sie in Kapitel 2 als Zeichen des Zeichensatzes mit den Nummern 8 bis 11 erklärt haben. Dies läßt sich jedoch für Spiele und ähnliches sehr schlecht verwenden.

Die Werte des Joysticks können mit den Befehlen JOY(0) und JOY(1) für die beiden Joysticks erfragt werden. Für diejenigen, die einen Joystick besitzen, bei dem auch Diagonalen zugelassen sind, ergeben sich die Werte aus Abb.

3.1. Dabei ist den vier Hauptrichtungen jeweils eines der Bits im zweiten Halbbyte zugeordnet und den Feuer-Tasten die letzten beiden Bit im ersten Halbbyte. Die Diagonalen ergeben sich aus den betreffenden Bitkombinationen, wobei logischerweise nicht alle Bitkombinationen sinnvoll sind.

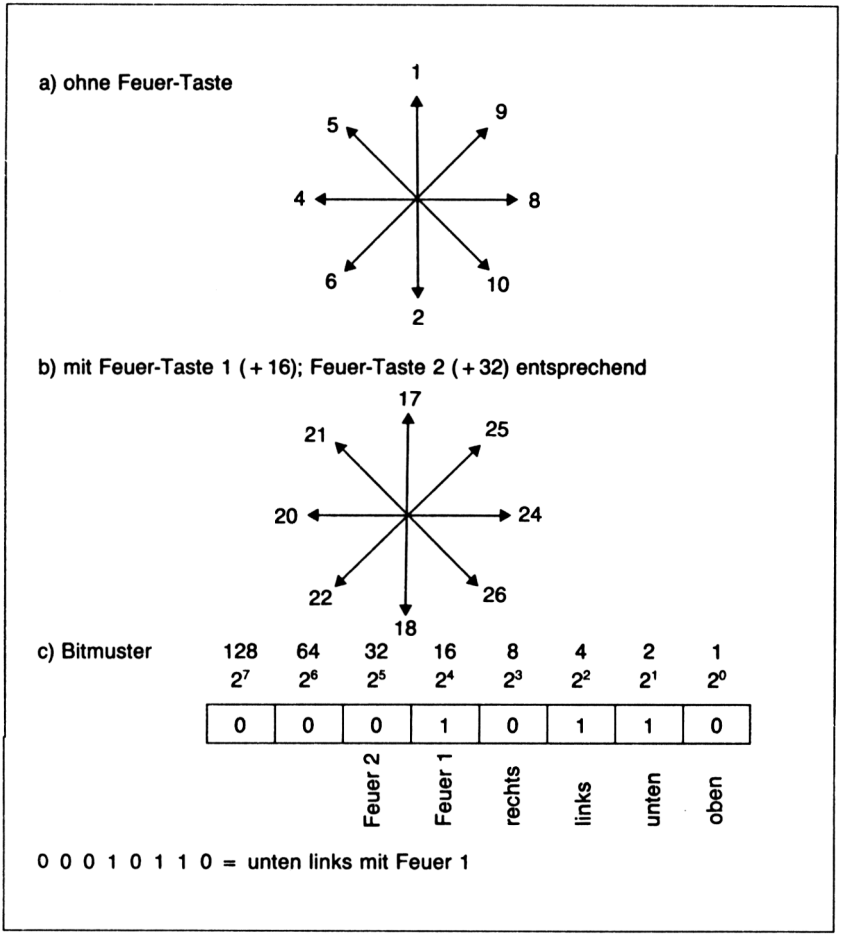


Abb. 3.1: Joystick-Werte in JOY(0) und JOY(1)

Mit folgendem kurzen Programm können Sie sich die Zuordnung auf dem Bildschirm sichtbar machen:

```
1000 DIM RICHTUNG$(26)
1010 :
1020 DATA oben,unten,-,links,links oben
1030 DATA links unten,-,rechts,rechts oben
1040 DATA rechts unten,-,-,-,-,-,Feuer
1050 DATA oben mit Feuer,unten mit Feuer
1060 DATA -,links mit Feuer,links oben mit Feuer
1070 DATA links unten mit Feuer,-,rechts mit Feuer
1080 DATA rechts oben mit Feuer,rechts unten mit Feuer
1090 :
1100 FOR I=1 TO 26
1110    READ RICHTUNG$(I)
1120 NEXT
1130 :
1140 JO=JOY(0)
1150 PRINT JO,RICHTUNG$(JO)
1160 GOTO 1140
```

Abb. 3.2: Programm für Joystick-Zuordnungswerte

Der Index in dem Feld RICHTUNG$( ) entspricht dabei dem betreffenden Joystickwert. Aus programmtechnischen Gründen wollen wir in Zukunft auch immer den Joystickwert vor seiner weiteren Verarbeitung einer Variablen (generell J0) zuordnen, um zu verhindern, daß sich bei größeren Überprüfungen innerhalb eines Programms dieser Wert plötzlich ändert und das Programm undefiniert reagiert.

Im Handbuch sind als Ersatz für die Joystickführung auch Tastenkombinationen angegeben, wobei für den Joystick 0 die Tasten nicht auf der Tastatur liegen. Die Werte von Joystick 1 können aber über die Tastatur simuliert werden, wobei die Zuordnung aus Abb. 3.3 deutlich wird. Bemerkenswert ist die Tatsache, daß zwei Tasten gleichzeitig gedrückt werden können - bei der Abbildung in Klammern gesetzt - und so auch die Diagonale erfragt werden kann.

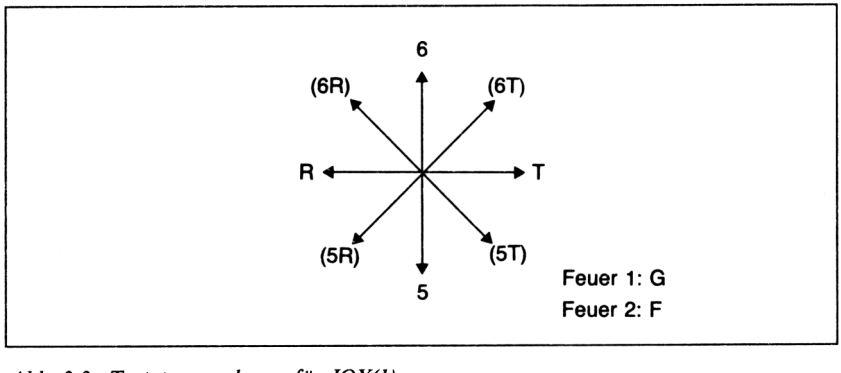


Abb. 3.3: Tastaturzuordnung für JOY(1)

Die Eigenschaften der Steuerung durch Joystick wollen wir uns nun in einem Programm zunutze machen, wo mit dem Joystick auf dem Bildschirm gezeichnet werden soll. Das Programm soll für Sie einen Rahmen darstellen, den Sie beliebig erweitern können, z. B. durch Ausgabe von Kreisen, Rechtecken und Blöcken auf entsprechenden Tastendruck.

```
1000 REM ------------------------------
1010 REM ---  Zeichnen mit Joystick ---
1020 REM ------------------------------
1030 :
1040 REM --- Bildschirm einstellen  ---
1050 :
1060 MODE 1
1070 BORDER 0
1080 PAPER 2
1090 INK 2,17
1100 PEN 0
1110 CLS
1120 :
1130 WINDOW #1,1,40,22,25
1140 PEN #1,2
1150 PAPER #1,3
1160 CLS #1
1170 :
1180 WINDOW #2,17,26,23,25
1190 PEN #2,3
1200 PAPER #2,2
1210 :
1220 X=320: Y=200
1230 ORIGIN 0,0
1240 GOTO 3080
1250 :
2000 REM --- Schleife zur Joystick- ---
2010 REM ---         abfrage          ---
2020 :
2030 JO=JOY(0)
2040 TASTE = 0
2050 IF JO>15 THEN JO=JO-16 : TASTE=1
2060 IF JO=9 THEN X=X+1 : Y=Y+1 : GOTO 2140
2070 IF JO=10 THEN X=X+1 : Y=Y-1 : GOTO 2140
2080 IF JO=6 THEN X=X-1 : Y=Y-1 : GOTO 2140
2090 IF JO=5 THEN X=X-1 : Y=Y+1 : GOTO 2140
2100 IF JO=1 THEN Y=Y+1 : GOTO 2140
2110 IF JO=8 THEN X=X+1 : GOTO 2140
2120 IF JO=2 THEN Y=Y-1 : GOTO 2140
2130 IF JO=4 THEN X=X-1 : GOTO 2140
2140 IF TASTE THEN PLOT X,Y,0
2150 :
3000 REM ---     Tastaturabfrage      ---
3010 :
3020 A$=INKEY$
3030 IF A$="" THEN 2030 '
3040 IF ASC(A$)>96 OR ASC(A$)<123 THEN B$=A$ ELSE GOTO 2030

3050 IF B$="z" THEN GOTO 3080
3060 IF B$="l" THEN RUN
3070 GOTO 2030
3080 :
4000 REM ---          Zeichnen        ---
4010 :
4020 CLS #2
4030 LOCATE #2,2,2
4040 PRINT #2,"Zeichnen
4050 PEN #1,2
4060 PAPER #1,3
4070 GOTO 2030
```

Abb. 3.4: Programm zum Zeichnen mit Joystick

Um einen Mittelweg zwischen verschiedenen Farbmöglichkeiten und einer hohen Auflösung einzuschlagen, haben wir den Bildschirmmodus 1 verwendet.

Zunächst wird eine Voreinstellung des Randes (Schwarz), des Papiers und des Schreibstifts vorgenommen und dann der Bildschirm gelöscht. Anschließend werden zwei Bildschirmfenster definiert, wobei Fenster #1 am unteren Bildschirmrand einen Bereich von vier Zeilen abdeckt und Fenster #1 innerhalb dieses Bereichs in der Mitte einen kleinen Ausschnitt. Der größere Rahmen kann von Ihnen zur Darstellung eines Menüs oder sonstiger Informationen herangezogen werden, der kleinere Rahmen ist für die momentane Aktivität vorgesehen.

Im weiteren wird der Grafikcursor in der Bildschirmmitte positioniert und der Ursprung des Koordinatenkreuzes auf den linken unteren Rand. Dann wird das Programmsegment zum Zeichnen aufgerufen.

In Zeile 2030 wird der Variablen J0 der Wert des Joysticks 0 zugewiesen und in der nächsten Zeile die Hilfsvariable TASTE zurückgesetzt. In Zeile 2050 wird sie wieder auf 1 gesetzt, wenn die Feuer-Taste gedrückt ist. Durch diese Vorgehensweise braucht im folgenden nur noch der Bereich zwischen 1 und 10 für die Joystickwerte überprüft zu werden, was einiges an Rechenzeit spart.

In den Zeilen 2060 bis 2130 wird aufgrund des jeweiligen Joystickwertes der Grafikcursor entsprechend bewegt. Durch den Befehl „GOTO 2140" wird wieder eine Rechenzeitverminderung erzielt, die besonders bei Diagonalen auftritt.

In Zeile 2140 wird deutlich, daß ein Punkt nur dann ausgegeben wird, wenn die Feuer-Taste gedrückt ist. Sie können also durch normale Bewegungen des Joysticks (ohne Feuer-Taste) den Cursor über den Bildschirm bewegen und durch Drücken der Feuer-Taste diesen Weg auch zeichnen.

Ab Zeile 3000 befindet sich ein Programmstück, das die Tastatur auf einen Tastendruck hin abfragt. Wenn kein Tastendruck erfolgte, wird sofort wieder zur Übernahme des Joystickwertes übergegangen. Die Plausibilitätsprüfung in Zeile 3040 läßt alle Buchstaben zu und verweigert die Eingabe aller anderen Tasten. Im folgenden Verteiler wurden bereits die Möglichkeiten z für zeichnen und l für löschen für Sie vorgesehen.

Die einfachste und schnellste Methode zum Löschen des Bildschirms ist ein Programmneustart, da in diesem Fall keine Variablen zurückgesetzt werden müssen. In dieser einfachen Form des Programms bietet es auch keine weiteren Schwierigkeiten.

Zur Darstellung eines Menüpunktes auf dem Bildschirm können Ihnen die Zeilen ab 4000 zur Vorlage dienen. Dem Ausbau des Programms sind nur durch Ihre Phantasie Grenzen gesetzt. Von den bereits erwähnten Kreisen, Rechtecken und Blöcken (Unterprogramme dazu finden Sie in den nächsten Kapiteln) ausgehend, können Sie auch Menüpunkte wie z. B. Bildschirmbereiche kopieren, Teilbereiche löschen, Umfärben oder - für Schaltpläne - festdefinierte Symbole realisieren.

In Abb. 3.5 sehen Sie ein kleines Beispiel.

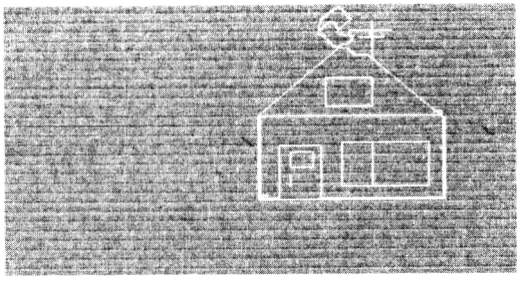


Abb. 3.5: Beispiel für das Zeichnen mit Joystick

Variablenübersicht

Variablenübersicht für JOYGRAF			1000-4070
Variablen			
Name	Art	Bereich	Bedeutung
A$	H	1 Buchstabe	Einlesen eines Zeichens von der Tastatur
B$	H	1 Buchstabe	Auswertung von A$
J0	G	0. . .42	Joystickwert bei letzter Abfrage
TASTE	G	0,1	Merker für Feuer-Taste
X	G	0. . .639	Aktuelle X-Koordinate des Grafik-Cursors
Y	G	0. . .199	Aktuelle Y-Koordinate des Grafik-Cursors

Unterprogramme		
Zeile	Bedeutung	wird aufgerufen in Zeile
keine		

Programmaufteilung in Blöcke	
1. Zeile	Bedeutung
1000	Vorspann/Bildschirmaufbau
2000	Joystickbearbeitung
3000	Tastaturbearbeitung (Menü)
4000	Behandlung Zeichnen
5000	Behandlung einer Anwendererweiterung
:	:

Abb. 3.6: Variablenübersicht für JOYGRAF

DIAGRAMME

Durch die Fähigkeit der hochauflösenden Grafik sind Computer ein geeignetes Hilfsmittel, um endlose - wenig aussagekräftige - Zahlenkolonnen anschaulich darzustellen. Bei den Diagrammen gibt es verschiedene Typen, von denen wir die drei wichtigsten im folgenden als Beispiel darstellen wollen.

Liniendiagramme

Die am häufigsten verwendete Form, z. B. für Umsätze oder andere zahlenmäßige Entwicklungen, ist das Liniendiagramm. Die Größe einer Zahl ist das Maß für den Abstand eines Linienpunktes von der Nullachse (unterer Bildschirmrand bei ORIGIN 0,0). Liniendiagramme werden durch eine besondere Eigenschaft des CPC einfach realisierbar, da eine Linie immer von der aktuellen Position des Grafikcursors zu einer angegebenen Position gezogen wird und sich der Grafikcursor dann dort befindet. Dadurch brauchen nur die jeweiligen Zahlenwerte entsprechend umgerechnet zu werden.

Im folgenden Beispiel haben wir auch die Möglichkeit, verschiedene Zahlengruppen auszuwählen, aufgezeigt. Zunächst sehen wir uns das Programm an (Abb. 3.7).

Als erstes zeigt das Programm ein kleines Menü, bei dem Sie zwischen den drei Liniengrafiken „Umsatz in Stück", „Umsatz in DM" und „Rohertrag"

Abb. 3.7: Programm zum Zeichnen von Liniendiagrammen

```
1000 REM ------------------------------
1010 REM --- Vorspann und Daten      ---
1020 REM ------------------------------
1030 :
1040 REM ---        Menue              ---
1050 :
1060 CLS
1070 :
1080 LOCATE 5,5
1090 PRINT "1 - UMSATZ IN STUECK"
1100 :
1110 LOCATE 5,7
1120 PRINT "2 - UMSATZ IN DM"
1130 :
1140 LOCATE 5,9
1150 PRINT "3 - ROHERTRAG"
1160 :
1170 A$=INKEY$
1180 IF A$="" THEN GOTO 1170
1190 IF A$="1" THEN RESTORE 2060
1200 IF A$="2" THEN RESTORE 2200
1210 IF A$="3" THEN RESTORE 2340
1220 :
2000 REM ------------------------------
2010 REM ---         Daten           ---
2020 REM ------------------------------
2030 :
2040 REM ---    Stueckumsatz         ---
2050 :
2060 DATA 5,6
2070 :
2080 DATA 41.6666667,90
2090 :
2100 DATA 800,2300,3500,4800,5200,5300
2110 DATA 0,900,2000,3500,4000,4200
2120 DATA 0,100,3000,6000,9000,12000
2130 DATA 0,0,0,500,1000,1400
2140 DATA 0,0,900,3400,2800,4500
2150 :
2160 DATA 14,24
2170 :
2180 REM ---     DM-Umsatz           ---
2190 :
2200 DATA 5,6
2210 :
2220 DATA 1250,90
2230 :
2240 DATA 17000,50000,90000,120000,130000,140000
2250 DATA 0,26000,70000,105000,120000,130000
2260 DATA 0,3000,135000,235000,300000,380000
2270 DATA 0,0,40000,100000,135000,170000
2280 DATA 0,0,20000,85000,60000,95000
2290 :
2300 DATA 22,16
2310 :
2320 REM ---        Rohertrag        ---
2330 :
2340 DATA 5,6
2350 :
2360 DATA 156.25,90
2370 :
2380 DATA 2000,8000,9000,12000,12500,13400
2390 DATA 0,2000,12000,12500,13000,14000
2400 DATA 0,0,5000,6000,8000,9000
2410 DATA 0,0,2000,11000,14000,15000
2420 DATA 0,0,2000,10000,12000,15000
2430 :
2440 DATA 10,32
2450 :
2460 REM --- Einlesen der Daten     ---
2470 :
2480 READ N,M
2490 READ VERT,HOR
2500 :
```

```
2510 DIM WERTE(N,M)
2520 :
2530 FOR I=1 TO N
2540    FOR J=1 TO M
2550       READ WERTE(I,J)
2560       WERTE(I,J)=WERTE(I,J) / VERT
2570    NEXT
2580 NEXT
2590 :
3000 REM ------------------------------
3010 REM ---       Hauptprogramm      ---
3020 REM ------------------------------
3030 :
3040 CLS
3050 ORIGIN 0,0
3060 FARBE=1
3070 :
3080 REM ---    Kurven zeichnen       ---
3090 :
3100 FOR I=1 TO N
3110    FOR J=1 TO M
3120       DRAW POSX+HOR,WERTE(I,J),FARBE
3130       POSX=XPOS
3140    NEXT
3150    PLOT 0,0
3160    POSX=0
3170    IF I=2 THEN INK 2,26 : FARBE=2
3180    IF I=4 THEN INK 3,15 : FARBE=3
3190 NEXT
3200 :
3210 PEN 4
3220 :
3230 REM ---     Vertikale Linien     ---
3240 :
3250 FOR I=0 TO M
3260    PLOT I*HOR,0
3270    DRAW I*HOR,350
3280    PLOT 0,0
3290    DRAW M*HOR,0
3300 NEXT
3310 :
3320 REM ---    Horizontale Linien    ---
3330 :
3340 READ ABSCHNITTE,HOEHE
3350 :
3360 FOR I=1 TO ABSCHNITTE
3370    PLOT 0,I*HOEHE
3380    DRAW 540,I*HOEHE
3390 NEXT
3400 :
3410 REM ---       Beschriftung       ---
3420 :
3430 PEN 1
3440 LOCATE 1,3
3450 PRINT" 3/83 4/83  1/84  2/84  3/84  4/84"
3460 :
3470 A=VAL(A$)
3480 ON A GOTO 5020,4020,6020
3490 :
4000 REM -   Beschr.  DM-Umsatz      ---
4010 :
4020 LOCATE 36,25 : PRINT"  0T"
4030 LOCATE 36,24 : PRINT" 20T"
4040 LOCATE 36,23 : PRINT" 40T"
4050 LOCATE 36,22 : PRINT" 60T"
4060 LOCATE 36,21 : PRINT" 80T"
4070 LOCATE 36,20 : PRINT"100T"
4080 LOCATE 36,19 : PRINT"120T"
4090 LOCATE 36,18 : PRINT"140T"
4100 LOCATE 36,17 : PRINT"160T"
4110 LOCATE 36,16 : PRINT"180T"
4120 LOCATE 36,15 : PRINT"200T"
4130 LOCATE 36,14 : PRINT"220T"
4140 LOCATE 36,13 : PRINT"240T"
4150 LOCATE 36,12 : PRINT"260T"
```

Abb. 3.7: Programm zum Zeichnen von Liniendiagrammen (Fortsetzung)

```
4160 LOCATE 36,11 : PRINT"280T"
4170 LOCATE 36,10 : PRINT"300T"
4180 LOCATE 36,9 : PRINT"320T"
4190 LOCATE 36,8 : PRINT"340T"
4200 LOCATE 36,7 : PRINT"360T"
4210 LOCATE 36,6 : PRINT"380T"
4220 LOCATE 36,5 : PRINT"400T"
4230 GOTO 7020
4240 :
5000 REM -   Beschr. Stueck-Umsatz  ---
5010 :
5020 LOCATE 36,25 : PRINT" 0 T"
5030 LOCATE 36,22 : PRINT" 2 T"
5040 LOCATE 36,19 : PRINT" 4 T"
5050 LOCATE 36,16 : PRINT" 6 T"
5060 LOCATE 36,13 : PRINT" 8 T"
5070 LOCATE 36,10 : PRINT"10 T"
5080 LOCATE 36,7 : PRINT"12 T"
5090 LOCATE 36,4 : PRINT"14 T"
5100 GOTO 7020
5110 :
6000 REM -   Beschr. Rohertrag      ---
6010 :
6020 LOCATE 36,25 : PRINT"  0T"
6030 LOCATE 36,23 : PRINT"  5T"
6040 LOCATE 36,21 : PRINT" 10T"
6050 LOCATE 36,19 : PRINT" 15T"
6060 LOCATE 36,17 : PRINT" 20T"
6070 LOCATE 36,15 : PRINT" 25T"
6080 LOCATE 36,13 : PRINT" 30T"
6090 LOCATE 36,11 : PRINT" 35T"
6100 LOCATE 36,9 : PRINT" 40T"
6110 LOCATE 36,7 : PRINT" 45T"
6120 LOCATE 36,5 : PRINT" 50T"
6130 LOCATE 36,3 : PRINT" 55T"
6140 :
7000 REM ---      Warteschleife      ---
7010 :
7020 A$=INKEY$ : IF A$="" THEN 7020
7030 MODE 1
7040 INK 1,24 : PEN 1
7050 RUN
```

Abb. 3.7: Programm zum Zeichnen von Liniendiagrammen (Fortsetzung)

wählen können. Durch die RESTORE-Befehle in den Zeilen 1190 bis 1210 wird der Zeiger auf die DATA-Zeilen entsprechend gesetzt. Die drei Datenbereiche ab Zeile 2000 sind wie folgt aufgeteilt:

Anzahl der Objekte/Anzahl der Elemente je Objekt
Umrechnungsfaktor für die Höhe/Abstand der Linienendpunkte voneinander
Daten zeilenweise je Objekt
Anzahl der Abschnitte/Höhe der Abschnitte in Pixel

Durch die Umrechnung mittels des Faktors VERT können die Originaldaten eingegeben werden (was nicht unbedingt in DATA-Zeilen geschehen muß, sondern auch sequentiell vom Datenträger erfolgen kann), da sie entsprechend ihrer Größe heruntergestuft werden, so daß die Grafik ein proportionales Abbild der wirklichen Zahlen ist. Dabei ist zu berücksichtigen, daß die

Bildschirmauflösung in der Höhe 200 Punkte beträgt und am oberen Rand gegebenenfalls noch eine Spaltenbeschriftung vorzunehmen ist. Direkt davon abhängig sind die Variablen ABSCHNITTE und HOEHE, die für eine entsprechende Rasterung der ausgegebenen Daten sorgen.

Ab Zeile 2480 werden die Daten eingelesen und gleich mit dem Faktor VERT umgerechnet. Ab Zeile 3040 erfolgt dann die Ausgabe, wobei durch den Parameter FARBE für einzelne Objektgruppen auch eine unterschiedliche Farbgebung durchgeführt werden kann, wie es z. B. in den Zeilen 3170 und 3180 dargestellt ist. Bei der Ausgabe in den Zeilen 3120 und 3130 ist zu berücksichtigen, daß der Grafikcursor jeweils am Ende der soeben gezeichneten Linie steht. Über die Hilfsvariable POSX wird die X-Koordinate „weitergeschaltet". Für jedes neue Objekt müssen der Grafikcursor und die Hilfsvarable POSX wieder zurückgesetzt werden.

In Zeile 3210 wird die Farbe zur Ausgabe umgeschaltet, und in den Zeilen ab 3250 erfolgt die Ausgabe der senkrechten Linien, die natürlich nur von der Variablen HOR abhängig sind.

Zum Zeichnen der waagerechten Linien werden die beiden Parameter ABSCHNITTE und HOEHE herangezogen. Durch Verwendung dieser Variablen kann dieser Programmabschnitt für alle möglichen Auflösungen herangezogen werden. Ab Zeile 3430 erfolgt die Beschriftung, die sich sicherlich selbst erklärt. Eine Übung für Sie wäre es, die Beschriftung nicht durch einzelne Positionsangaben mit anschließender Ausgabe zu realisieren, sondern durch eine Schleife (für die auszugebenden Werte verwenden Sie am besten die STR$( ) -Funktion). In Abb. 3.8, 3.9 und 3.10 ist die Ausgabe des Liniendiagramms mit dem Hardcopy-Programm vom Ende des Kapitels dargestellt.

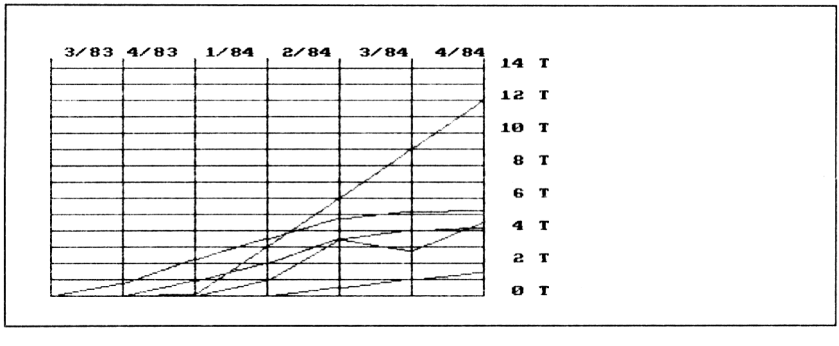


Abb. 3.8: Beispiel für ein Liniendiagramm

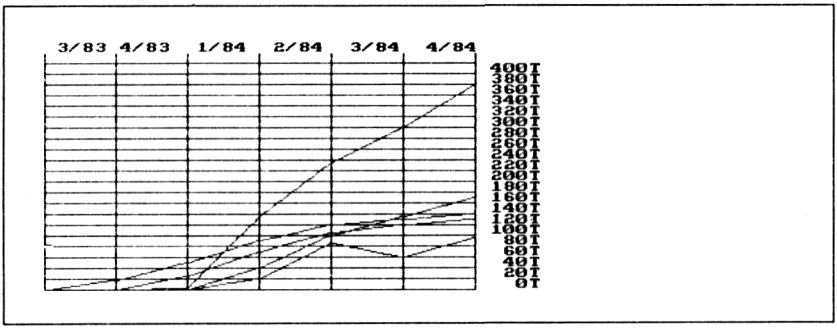


Abb. 3.9: Beispiel für ein Liniendiagramm

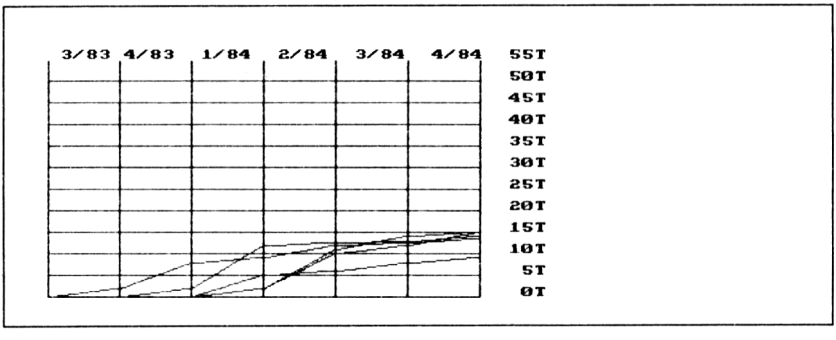


Abb. 3.10: Beispiel für ein Liniendiagramm

Variablenübersicht

Variablenübersicht für LINIE Zweck: Ausgabe eines Liniendiagramms			1000-7050
Variablen			
Name	Art	Bereich	Bedeutung
A	H	0. . .9	Zahlenwert von A$
A$	H	1 Zeichen	Von Tastatur
ABSCHNITTE	H	Integer	Anzahl der vertikalen Abschnitte
FARBE	G	1. . .16	Aktuelle Farbauswahlnummer für Linien
HOEHE	G	1. . .200	Höhe der einzelnen Abschnitte in Bildschirmpunkten

Variablenübersicht für LINIE 1000-7050
Zweck: Ausgabe eines Liniendiagramms

Variablen

Name	Art	Bereich	Bedeutung
HOR	G	Integer	Abstand (horizontal) der Werte
I	H	1. . . N/O. . .M 1 . . . ABSCHNITTE	Laufvariable
J	H	1. . .M	Laufvariable
N	G	Integer	Anzahl der Objekte
M	G	Integer	Anzahl der Werte je Objekt
POSX	H	Integer	Alte X-Position des Grafikcursors
VERT	G	Dezimal	Umrechnungsfaktor für vertikale Skalierung
WERTE(,)	G	Dezimal	Auszugebende Werte im Original (ohne Faktor)

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
keine		

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Menü
2000	Daten
3000	Hauptprogramm mit Ausgabe
4000	Beschriftung für DM-Umsatz
5000	Beschriftung für Stück-Umsatz
6000	Beschriftung für Rohertrag

Abb. 3.11: Variablenübersicht für LINIE

Balkendiagramme

Ähnlich wie bei den Liniendiagrammen funktioniert auch die Ausgabe bei den Balkendiagrammen, jedoch ist es hier schwieriger, mehrere Objekte gleichzeitig darzustellen (Balken überlappen lassen).

Da Sie nun schon einige Übung im Umgang mit der Grafik haben, werden wir

Ihnen im folgenden nur ein Programm zur Ausgabe einzelner Balken (z. B. mehrere Werte eines Objektes oder mehrere Objekte mit entsprechenden Werten) vorstellen. Die Darstellung mehrerer überlappender Balken empfehlen wir als Übung.

Abb. 3.12: Programm zum Zeichnen von Balkendiagrammen

```
1000 REM ------------------------------
1010 REM ---     Vorspann und Daten  ---
1020 REM ------------------------------
1030 :
1040 BORDER 0
1050 CLS
1060 ORIGIN 0,0
1070 LOCATE 5,5
1080 INPUT"Anzahl der Balken";ANZAHL
1090 :
1100 DIM WERT(ANZAHL)
1110 DIM HOEHE(ANZAHL)
1120 DIM PROZENT(ANZAHL)
1130 :
1140 CLS
1150 :
1160 REM ---     Erfassen der Daten  ---
1170 :
1180 FOR I=1 TO ANZAHL
1190     LOCATE 5,2+I
1200     PRINT "Wert fuer Balken";I;
1210     INPUT WERT(I)
1220     SUMME=SUMME+WERT(I)
1230     IF WERT(I) > MAXIMUM
           THEN MAXIMUM=WERT(I)
1240 NEXT
1250 :
1260 CLS
1270 LOCATE 5,2
1280 PRINT"Summe              ";SUMME
1290 :
1300 LOCATE 5,4
1310 INPUT"Abstand der Balken";ABSTAND
1320 LOCATE 5,6
1330 INPUT"Breite             ";BREITE
1340 LOCATE 5,8
1350 INPUT"Grundzeile         ";GRUND
1360 LOCATE 5,10
1370 INPUT"Obergrenze         ";OBEN
1380 LOCATE 5,12
1390 INPUT"Stufenhoehe        ";STUFEN
1400 LOCATE 5,14
1410 INPUT"Beginn in Spalte   ";SPALTE
1420 LOCATE 1,16
1430 :
1440 PRINT"      1 - Rechteck
1450 PRINT"      2 - Block
1460 PRINT"      3 - Quader
1470 PRINT
1480 INPUT"      1/2/3        ";ART
1490 :
1500 IF ART < 1 OR ART > 3
       THEN GOTO 1480
1510 IF ART = 3
       THEN LOCATE 5,20 :
       INPUT"Tiefe             ";TIEFE
1520 :
1530 REM ---        Umrechnung        ---
1540 :
1550 FOR I=1 TO ANZAHL
1560     PROZENT(I) =
             INT(WERT(I)*100/SUMME+0.5)
1570     HOEHE(I) = INT(WERT(I) *
             (GRUND+OBEN+TIEFE+STUFEN *
             (ANZAHL-1))/MAXIMUM+0.5)
```

```
1580 NEXT
1590 :
1600 CLS
1610 :
2000 REM ------------------------------
2010 REM ---        Hauptprogramm     ---
2020 REM ------------------------------
2030 :
2040 UNTENX = SPALTE - ABSTAND
2050 :
2060 FOR I=1 TO ANZAHL
2070    OBENX  = UNTENX+ABSTAND
2080    UNTENX = OBENX+BREITE
2090    OBENY  = GRUND+HOEHE(I)
2100    UNTENY = GRUND+STUFEN*(I-1)
2110    IF ART = 3
          THEN HINTENX = OBENX+TIEFE :
               HINTENY = OBENY+TIEFE
2120    ON ART GOSUB 3000,4000,5000
2130 NEXT
2140 :
2150 A$=INKEY$
2160 IF A$="" THEN 2150
2165 IF A$="h" THEN CALL &A080
2170 END
3000 REM ------------------------------
3010 REM ---         Rechteck         ---
3020 REM ------------------------------
3030 :
3040    MOVE OBENX,OBENY
3050    DRAW OBENX,UNTENY,1
3060    DRAW UNTENX,UNTENY,1
3070    DRAW UNTENX,OBENY,1
3080    DRAW OBENX,OBENY,1
3090 RETURN
3100 :
4000 REM ------------------------------
4010 REM ---            Block         ---
4020 REM ------------------------------
4030 :
4040    FOR LINIE=UNTENY TO OBENY
4050       MOVE OBENX,LINIE
4060       DRAWR BREITE,0,1
4070    NEXT
4080 RETURN
4090 :
5000 REM ------------------------------
5010 REM ---          Quader          ---
5020 REM ------------------------------
5030 :
5040    MOVE OBENX,OBENY
5050    GOSUB 3000
5060    DRAW HINTENX,HINTENY,1
5070    MOVE UNTENX,OBENY
5080    DRAW UNTENX+TIEFE,HINTENY,1
5090    REM DRAW UNTENX+HINTENX-OBENX,
  HINTENY,1
5100    MOVE OBENX,UNTENY
5110    DRAW HINTENX,UNTENY+TIEFE,1
5120    REM DRAW HINTENX,UNTENY+HINTENY-
  OBENY,1
5130    MOVE UNTENX,UNTENY
5140    DRAW UNTENX+TIEFE,UNTENY+TIEFE,1
5150    REM  REM UNTENX+HINTENX-OBENX,
  UNTENY+HINTENY-OBENY,1
5160    OBENX  = HINTENX
5170    OBENY  = HINTENY
5180    UNTENX = UNTENX+TIEFE
5190    UNTENY = UNTENY+TIEFE
5200    GOSUB 3000
5210 RETURN
```

Abb. 3.12: Programm zum Zeichnen von Balkendiagrammen (Fortsetzung)

Anders als im vorigen Abschnitt haben wir hier den Weg der Eingabe über Tastatur beschritten. Auch dabei werden die Daten erfaßt und aufbereitet, wobei einige Daten mehr einzugeben sind. Durch die Vielzahl der Variablen (ABSTAND, BREITE, GRUND, OBEN, STUFEN und SPALTE) können Sie Ihre Grafik beliebig auf dem Bildschirm positionieren und haben so gegebenenfalls Platz für erläuternde Texte oder Beschriftungen. Der größte Balken orientiert sich proportional am größten eingegebenen Wert (MAXIMUM). Er füllt den gesamten Abstand zwischen den Pixel-Zeilen GRUND und OBEN aus. Dabei wird die Erhöhung (für fluchtpunktartige Darstellung) jeweils mit eingerechnet.

Den linken Rand können Sie durch die Variable SPALTE bestimmen. Des weiteren können Sie noch die BREITE in Bildschirmpunkten für alle Balken angeben sowie den ABSTAND zwischen den Balken, ebenfalls in Bildschirmpunkten.

Ab Zeile 1440 wählen Sie zwischen den möglichen Darstellungsarten, wie im Listing angegeben. Zeile 1500 ist lediglich eine Plausibilitätsprüfung, und für die Darstellung als Quader muß natürlich noch die TIEFE angegeben werden (Zeile 1510).

Ab Zeile 1550 befindet sich die Umrechnung, wobei auch eine Umrechnung in Prozent vom Gesamtanteil erfolgt, die Sie übungshalber neben, über oder unter dem Balken anbringen können.

Durch die Vorgabe von UNTENX im Hauptprogramm kann innerhalb der Schleife die linke X-Koordinate eines Balkens sehr leicht aus der rechten X-Koordinate des letzten ausgegebenen Balkens und des Abstandes zwischen den Balken errechnet werden. Ist der linke Rand festgelegt, so ergibt sich der rechte Rand aus dem linken Rand, vermehrt um die weiter oben erfaßte Breite. Bei der vorher errechneten Höhe (Zeile 1570) muß nun nur noch die Anzahl der Stufen mitverarbeitet werden. Außerdem muß für einen Quader noch ein weiterer Bezugspunkt angegeben werden.

Bewußt haben wir Ihnen drei Möglichkeiten für Balkendiagramme zur Auswahl gestellt, um Ihnen auf diese Weise auch BASIC-Unterprogramme für Rechteck, Block und Quader aufzeigen zu können, da der CPC von Hause aus diese Befehle nicht kennt. Die Vorgehensweise entnehmen Sie bitte den Abbildungen 3.13 bis 3.15. Durch die angegebenen Koordinaten sind die geometrischen Figuren eindeutig beschrieben.

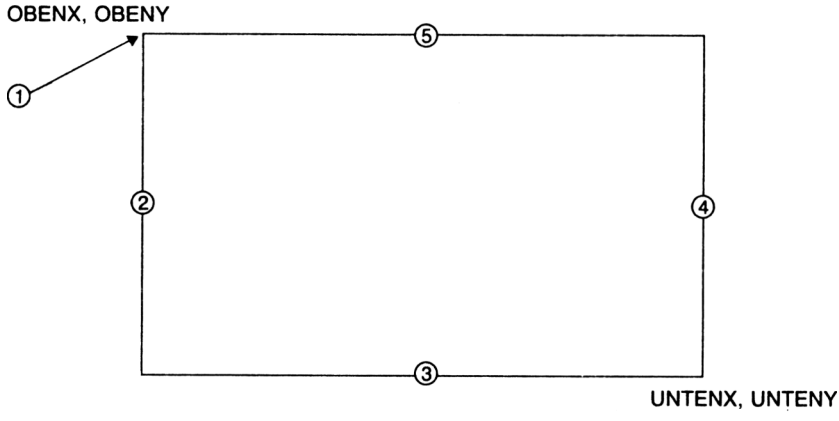


Abb. 3.13: Koordinaten und Reihenfolge der Darstellung beim Rechteck

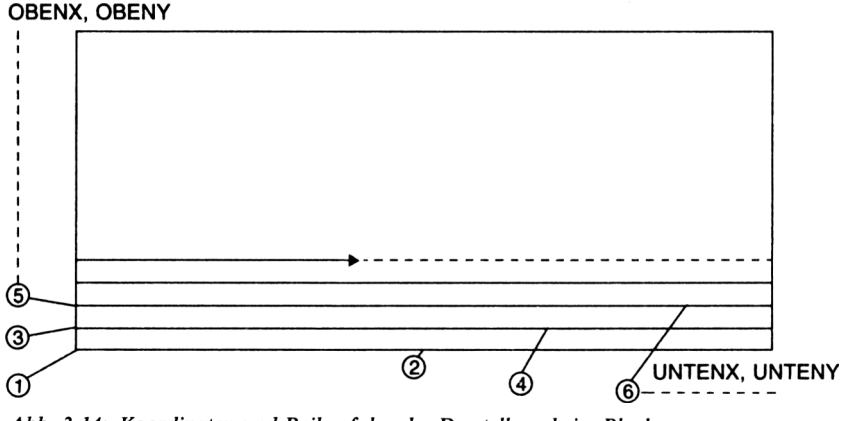


Abb. 3.14: Koordinaten und Reihenfolge der Darstellung beim Block

Das Unterprogramm zum Zeichnen eines Blocks kommt wegen der Grafikphilosophie des CPC (Grafikcursor am letzten gezeichneten Punkt) mit relativ wenig Befehlen aus. Im Gegensatz zum Rechteck wurde hier der DRAWR-Befehl verwendet, da die Y-Koordinate bekannt ist und die Breite die Länge jeder einzelnen Linie angibt.

Beim Quader wurde in REM-Zeilen jeweils die allgemeine Formel zur Berechnung der Punkte angegeben, wenn z. B. die Tiefe unterschiedlich oder überhaupt nicht bekannt ist. In der Abb. 3.15 ist dies für die X-Koordinaten anhand der Vektoren verdeutlicht.

Wir wollen auch von den Balkendiagrammen wieder eine Hardcopy zeigen.

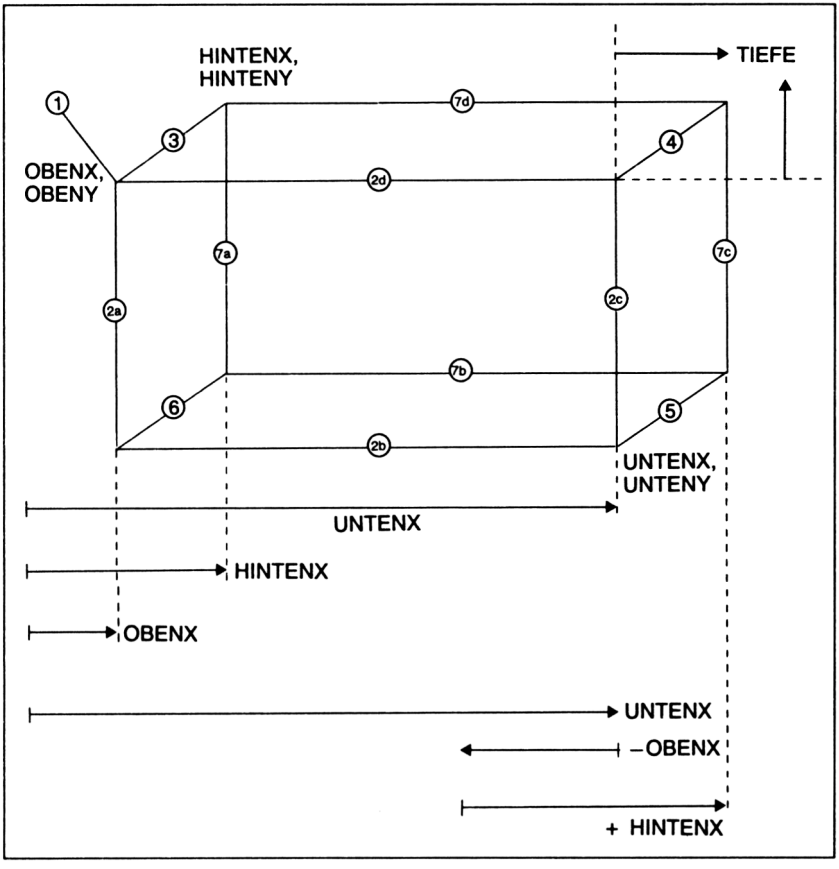


Abb. 3.15: Koordinaten und Reihenfolge der Darstellung beim Quader

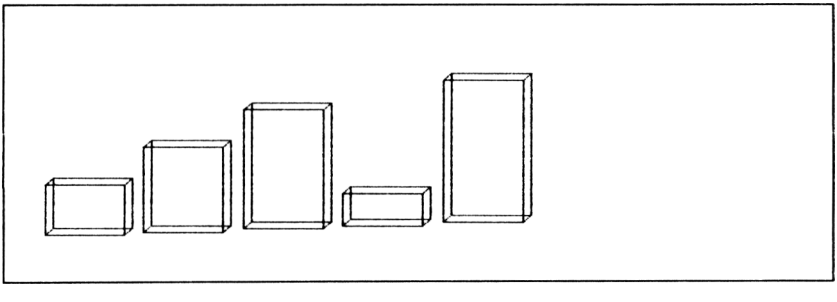


Abb. 3.16: Beispiel für ein Balkendiagramm (Quader)

Variablenübersicht

Variablenübersicht für BALKEN 1000-5210
Zweck: Ausgabe von Balkendiagrammen

Variablen

Name	Art	Bereich	Bedeutung
ABSTAND	G	Integer	Abstand der Balken in Pixel
ANZAHL	G	Integer	Anzahl der Werte (Balken)
ART	H	1,2,3	Art der Ausgabe: 1-Rechteck 2-Block 3-Quader
BREITE	G	Integer	Breite der Balken in Pixel
GRUND	G	Integer	Unterste Zeile der Ausgabe
HINTENX	H	1. . .640	X-Koordinate oben links hinten für Quader
HINTENY	H	1. . .200	Y-Koordinate oben links hinten für Quader
HOEHE( )	G	Integer	Höhe der Balken; aus WERT( ) umgerechnet
I	H	1. . .ANZAHL	Laufvariable
LINIE	H	UNTENY. . . OBENY	Laufvariable bei Blockausgabe
MAXIMUM	H/G	Dezimal	Größter Wert
PROZENT( )	G	Dezimal	Prozentanteil jedes einzelnen Wertes an der Summe zur weiteren Verarbeitung
OBEN	G	Integer	Oberste Zeile der Ausgabe
OBENX	H	1. . .640	X-Koordinate oben links für Block, Rechteck und Quader
OBENY	H	1. . .200	Y-Koordinate oben links für Block, Rechteck und Quader
SPALTE	G	Integer	Linke Grenze der Ausgabe
STUFEN	G	Integer	Erhöhung der Grundzeile für jeden weiteren Balken
SUMME	H/G	Dezimal	Aktuelle Summe/Gesamtsumme der Werte
TIEFE	G	Integer	„Seitliche Versetzung" beim Quader
UNTENX	H	1. . .640	X-Koordinate unten rechts für Block, Rechteck und Quader
UNTENY	H	1. . .200	Y-Koordinate unten rechts für Block, Rechteck und Quader
WERT( )	G	Dezimal	Auszugebende Werte je Balken

Unterprogramme		
Zeile	Bedeutung	wird aufgerufen in Zeile
3000	Rechteck ausgeben	2120
4000	Block ausgeben	2120
5000	Quader ausgeben	2120
Programmaufteilung in Blöcke		
1. Zeile	Bedeutung	
1000	Vorspann und Datenerfassung vom Bildschirm	
2000	Hauptprogramm	
3000	Ausgabe eines Rechtecks	
4000	Ausgabe eines Blocks	
5000	Ausgabe eines Quaders (hohl)	

Abb. 3.17: Variablenübersicht für BALKEN

Tortendiagramme

Als letzte Möglichkeit für Diagramme wollen wir Ihnen die Tortendiagramme vorstellen, unter anderem um Ihnen auch die Anwendung für zwei weitere Grafik-Routinen (Kreis, Ellipse; Radius) zu zeigen.

Abb. 3.18: Programm zum Zeichnen von Tortendiagrammen

```
1000 REM ------------------------------
1010 REM ---    Vorspann und Daten  ---
1020 REM ------------------------------
1030 :
1040 BORDER 0
1050 CLS
1060 ORIGIN 0,0
1070 LOCATE 5,5
1080 INPUT"Anzahl der Teile ";ANZAHL
1090 :
1100 DIM WERT(ANZAHL)
1110 DIM GRAD(ANZAHL)
1120 DIM PROZENT(ANZAHL)
1130 :
1140 CLS
1150 :
1160 REM ---    Erfassen der Daten  ---
1170 :
1180 FOR I=1 TO ANZAHL
1190    LOCATE 5,2+I
1200    PRINT "Wert fuer Teil";I;
1210    INPUT WERT(I)
1220    SUMME=SUMME+WERT(I)
1230 NEXT
1240 :
1250 CLS
1260 LOCATE 5,2
1270 PRINT"Summe = 100%      ";SUMME
1280 :
1290 REM ---       Umrechnung       ---
```

```
1300 :
1310 FOR I=1 TO ANZAHL
1320    PROZENT(I) =
           INT(WERT(I)*100/SUMME+0.5)
1330    GRAD (I) =
           INT(WERT(I)*360/SUMME+0.5)
1340 NEXT
1350 :
2000 REM ------------------------------
2010 REM ---        Hauptprogramm     ---
2020 REM ------------------------------
2030 :
2040 MITTELX  = 320
2050 MITTELY  = 200
2060 RADIUSX  = 200
2070 RADIUSY  = 100
2080 :
2090 GOSUB 3000
2100 :
2110 WINKEL = 0
2120 SUMME  = 0
2130 :
2140 GOSUB 4000
2150 :
2160 FOR I=1 TO ANZAHL-1
2170    SUMME=SUMME+GRAD(I)
2180    WINKEL=SUMME
2190    GOSUB 4000
2200 NEXT
2210 :
2220 A$=INKEY$
2230 IF A$="" THEN 2220
2240 :
2250 END
3000 REM ------------------------------
3010 REM ---          Kreis           ---
3020 REM ------------------------------
3030 :
3040    IF RADIUSX > RADIUSY
          THEN H=RADIUSX
3050    IF RADIUSX <= RADIUSY
          THEN H=RADIUSY
3060    FOR KREIS=0 TO 2*PI STEP 1/H
3070       PUNKTX=INT(MITTELX + RADIUSX
                 *SIN(KREIS)+0.5)
3080       PUNKTY=INT(MITTELY + RADIUSY
                 *COS(KREIS)+0.5)
3090       PLOT PUNKTX,PUNKTY,1
3100    NEXT
3110 RETURN
4000 REM ------------------------------
4010 REM ---          Radius          ---
4020 REM ------------------------------
4030 :
4040    MOVE MITTELX,MITTELY
4050    H=WINKEL * 2 * PI/360
4060    DRAWR INT(RADIUSX*SIN(H)+0.5),
              INT(RADIUSY*COS(H)+0.5),1
4070 RETURN
```

Abb. 3.18: Programm zum Zeichnen von Tortendiagrammen (Fortsetzung)

Der Beginn des Listings ist ähnlich wie im Programm in Abb. 3.12, jedoch statt der Höhe der Balken muß hier eine Gradeinteilung vorgenommen werden. Auch das Erfassen der Daten ist ähnlich, obwohl hier nicht so viele Parameter benötigt werden.

Die Umrechnung in Prozentwerte von der Gesamtsumme (Zeile 1320) und in eine Gradeinteilung (Zeile 1330) unterscheidet sich nur durch die Faktoren 100 bzw. 360.

Für die weiter unten vorgestellten Programme zum Zeichnen eines Kreises bzw. eines Radius werden ab Zeile 2040 die Parameter vorgestellt (Kreismittelpunkt ist auch Mittelpunkt des Bildschirms, und es wird eine Ellipse mit RADIUSX = 200 und RADIUSY = 100 ausgegeben). Wie Sie wissen, ist der Kreis ein Spezialfall der Ellipse, wobei RADIUSX und RADIUSY gleich sind.

In Zeile 2090 wird das Unterprogramm zum Ausgeben eines Kreises aufgerufen und in den Zeilen 2110 bis 2140 der erste Radius (0 Grad) ausgegeben. Die Variable SUMME wird in der Schleife von Zeile 2160 bis 2200 jeweils zum Summieren der bereits ausgegebenen Abschnitte verwendet, da sonst die einzelnen Radien immer von 0 Grad aus gerechnet würden.

Beim Unterprogramm zur Darstellung des Kreises wird zunächst eine Hilfsvariable so gesetzt, daß eine möglichst optimale Schrittweite erzielt wird. Dann werden in einer Schleife - aufgrund der eben errechneten Schrittweite - die einzelnen Punkte des Kreises ausgegeben, wobei noch eine Rundung auf ganze Zahlen erfolgt. Die Formel zur Berechnung der Kreispunkte finden Sie in jedem Schulbuch oder Lexikon.

Ähnlich ergibt sich auch die Ausgabe eines Radius, wobei für jede neue Ausgabe der Grafikcursor wieder auf den Mittelpunkt des Kreises gestellt werden muß. In diesem Fall dient die Hilfsvariable H zur Vereinfachung der Angaben im DRAWR-Befehl.

Ebenso wie bei den Balkendiagrammen können Sie hier auch noch Beschriftungen anbringen, wobei wir Sie besonders auf den TAG-Befehl hinweisen möchten. Fortgeschrittene können auch versuchen, einige Teilstücke aus dem Kreis herauszutrennen, bzw. die einzelnen Abschnitte verschieden einzufärben. Auch hier zeigen wir Abb. 3.19 wieder eine Hardcopy.

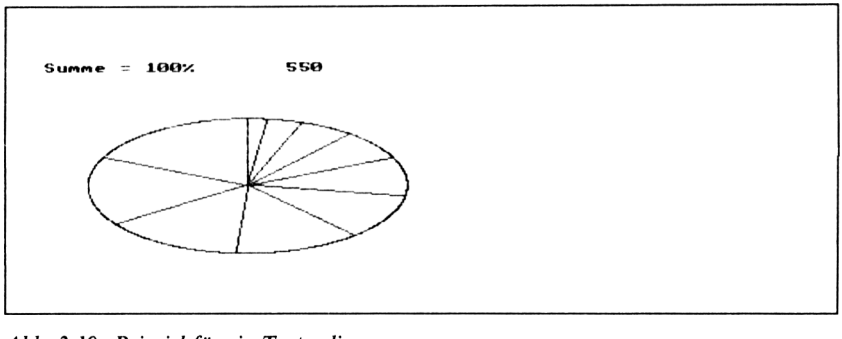


Abb. 3.19: Beispiel für ein Tortendiagramm

Variablenübersicht

Variablenübersicht für KREIS 1000-4070
Zweck: Ausgabe von Tortendiagrammen

Variablen

Name	Art	Bereich	Bedeutung
ANZAHL	G	Integer	Anzahl der Werte (Teile)
GRAD( )	G	Integer	eingegebene Werte in Winkel umgerechnet
H	H	Dezimal	Hv. zur Berechnung einer optimalen Schrittweite bei der Kreisausgabe (Ellipse)
I	H	1. . .ANZAHL	Laufvariable
MITTELX	G/E	1. . .640	absolute X-Koordinate des Kreismittelpunktes
MITTELY	G/E	1. . .200	absolute Y-Koordinate des Kreismittelpunktes
PI	S	3,1415. . .	Kreiszahl π
PROZENT( )	G	Dezimal	Prozentanteil jedes einzelnen Wertes an der Summe zur weiteren Verarbeitung
RADIUSX	G/E	0. . .320	Radius in X-Richtung
RADIUSY	G/E	0. . .100	Radius in Y-Richtung
SUMME	H/G	Dezimal	aktuelle Summe/Gesamtsumme der Werte
WERT( )	G	Dezimal	Auszugebende Werte je Balken
WINKEL	H/E	0. . .360	Ausgabewinkel für Radius

Unterprogramme

Zeile	Bedeutung	wird aufgerufen in Zeile
3000	Kreis (Ellipse) ausgeben	2090
4000	Radius zeichnen	2140

Programmaufteilung in Blöcke

1. Zeile	Bedeutung
1000	Vorspann und Datenerfassung vom Bildschirm
2000	Hauptprogramm
3000	Ausgabe eines Kreises
4000	Ausgabe eines Radius

Abb. 3.20: Variablenübersicht für KREIS

SPRITES

Das Thema Sprites hat schon viele Gemüter beschäftigt. Manche glauben, es gäbe nichts Überflüssigeres als diese, während andere wiederum nicht ohne sie auskommen möchten. Nun, worum geht's eigentlich? Was sind Sprites, und wozu werden sie gebraucht?

Sprites sind nichts anderes als, im Normalfall, frei definierbare Zeichen, oft auch mehrfarbig. Die Größe dieser Zeichen ist von Rechner zu Rechner verschieden. Ein wichtiges Merkmal von Sprites ist ihre freie Positionierbarkeit (es genügt die Angabe eines Koordinatenpunktes, um alle Pixels auf dem Bildschirm zu positionieren). Außerdem sollten sie an beliebigen Orten auf dem Bildschirm auftauchen können, ohne den Hintergrund zu zerstören, d. h. nach dem Löschen eines Sprite muß der Hintergrund so aussehen, als wäre nie eine Veränderung des Bildes aufgetreten.

Sprites werden bis auf sehr wenige Anwendungsfälle eigentlich nur in Spielen gebraucht. Die Funktion in Spielen ist sehr leicht erkennbar, da die meisten Spiele davon leben, daß irgendwelche Figuren auf dem Bildschirm herumschwirren, die eingefangen, abgeschossen oder nicht berührt werden dürfen. Die wenigen kommerziellen Anwendungen beschränken sich auf die Darstellung kleiner Symbole, die dem Anwender die Bedienung seines Systems erleichtern.

Nach dieser Einführung nun ein kleiner Ausblick auf die Möglichkeiten der Sprite-Darstellung auf dem Schneider CPC. Im Prinzip hat der CPC schon so etwas wie Sprites, nämlich die frei definierbaren und frei positionierbaren Zeichen seines Zeichensatzes. Der einzige Nachteil dieser Zeichen ist die geringe Ausgabegeschwindigkeit sowie die Tatsache, daß diese Zeichen einfarbig sind. Man müßte also zur Darstellung farbiger Symbole mehrere Zeichen in unterschiedlichen Farben übereinanderschreiben.

Die freie Positionierbarkeit kann auf dem CPC mit dem TAG-Befehl erfolgen. Wenn zusätzlich der Grafikmodus auf XOR-Modus (Exclusive Oder-Verknüpfung) eingestellt ist, so wird der Hintergrund nicht zerstört, ja das Zeichen kann durch zweimaliges Schreiben auf die gleiche Position auch wieder gelöscht werden. XOR-Modus bedeutet, daß sich die neue Farbe eines Grafikpixels durch die logische Verknüpfung mit XOR von alter Farbe und der neu zu schreibenden Farbe ergibt. Wir wollen uns dazu ein Beispiel ansehen. Dabei bedeutet % binär, und alle Verknüpfungen erfolgen mit XOR.

Allgemein gilt (bitweise) folgende Zuordnung bei der XOR-Verknüpfung:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Beispiel:

alte Farbe (Bitmuster: 9 dez = %1001):	9 dez = %1001
zu schreibende Farbe (12 dez):	12 dez = %1100
neue Farbe (5 dez):	5 dez = %0101
nochmal Farbe 12:	12 dez = %1100
Ergebnis = alte Farbe:	9 dez = %1001

So können auf relativ einfache Weise Sprites der Größe 8×8 Pixels realisiert werden. Doch wie gesagt, diese Ausgabeweise ist langsam, deshalb besteht bei manchen CPC-Besitzern der Wunsch nach schnelleren - und größeren - Sprites. Manchmal ist die „Sprite-Fähigkeit" ein wichtiges Kaufargument. Dieser Drang nach Sprites soll hier erst einmal mit einem einfachen Programm befriedigt werden.

Da der Videocontroller des CPC (ein 6845 von Motorola) nicht die Möglichkeit bietet, Sprites unabhängig vom Bildspeicher darzustellen, müssen die Sprites direkt in den Speicher eingeschrieben werden. Dadurch ergibt sich der Nachteil, daß der frühere Speicherinhalt zerstört wird, wenn man nicht auf Kosten der Schnelligkeit den Bildschirminhalt mit dem Sprite durch XOR verknüpfen will. Das vorgestellte Programm bietet nur die Möglichkeit, Sprites in den Speicher destruktiv einzuschreiben. Das Löschen der Sprites geschieht durch Überschreiben des Sprite mit einem anderen Sprite, das leer ist. Die Programme zur Darstellung der Sprites sind als Erweiterungsbefehle in das Betriebssystem eingebunden (siehe auch Kapitel über Befehlserweiterungen).

Unter BASIC werden sie mit ihrem Namen aufgerufen und somit wie ein normaler BASIC-Befehl behandelt. Um dem BASIC anzugeben, daß es sich hierbei um einen externen Befehl handelt, muß dem Befehl der senkrechte Strich mit SHIFT @ vorangestellt werden. Näheres hierzu finden Sie auch im Kapitel über BASIC-Erweiterungen.

Das Programm funktioniert bisher nur im MODE 0. Die Größe der darstellbaren Sprites beträgt 16×16 Pixels, und die untere linke Ecke des Sprite definiert seine Position. Die Position des Sprite kann in den üblichen Grafik-

Koordinaten angegeben werden, ist aber von einer möglichen ORIGIN-Definition unabhängig.

Abb. 3.21: Sprite-Programm

```
                    10 *H SPRITES für den CPC464
BCD1                20 ERWEIT: EQU  #BCD1
BC1D                30 DOTPOS: EQU  #BC1D
BC26                40 NEXTLN: EQU  #BC26
BC2C                50 INKCOD: EQU  #BC2C
                    60
9200                70 SPRTAB: EQU  #9200
                    80
9000                90        ENT   #9000              ; Startadresse
9000               100        ORG   #9000              ; Anfang des Codes
9000  010E90        110        LD    BC,TABELL          ; Register für Befehls-
9003  210A90        120        LD    HL,EXTTAB          ; erweiterung laden
9006  CDD1BC        130        CALL  ERWEIT             ; Befehle initialisieren
9009  C9            140        RET                      ; zurück zum BASIC
900A                150 EXTTAB: DEFS  4                 ; Platz für Betriebssys.
900E  1990          160 TABELL: DEFW  NAMEN             ; Sprungtabelle für
9010  C34790        170        JP    SPRON              ; externe Befehle
9013  C33490        180        JP    SPROFF             ; mit den Befehlsnamen
9016  C3E190        190        JP    SPRGEN
9019  53505249      200 NAMEN:  DEFB  "S","P","R","I","T","E","O","N"+#80
9021  53505249      210        DEFB  "S","P","R","I","T","E","O","F","F"+#80
902A  53505249      220        DEFB  "S","P","R","I","T","E","G","E","N"+#80
9033  00            230        DEFB  #00
                    240
9034  FE02          250 SPROFF: CP    2                 ; 2 Argumente?
9036  C0            260        RET   NZ                 ; zurück, wenn nicht
9037  3E10          270        LD    A,16               ; 16. Sprite
9039  DD4601        280        LD    B,(IX+1)           ; Y-Koordinate ins
903C  DD4E00        290        LD    C,(IX+0)           ; BC-Register
903F  DD5603        300        LD    D,(IX+3)           ; X-Koordinate ins
9042  DD5E02        310        LD    E,(IX+2)           ; DE-Register
9045  1806          320        JR    START
                    330
9047  FE03          340 SPRON:  CP    3                 ; 3 Argumente?
9049  C0            350        RET   NZ                 ; zurück, wenn nicht
904A  CDAF90        360        CALL  GT3ARG             ; 3 Argumente holen
904D  CDC190        370 START:  CALL  TSTARG             ; Argumente prüfen
9050  D8            380        RET   C                  ; ungültig --> Return
9051  3C            390        INC   A                  ; A=A+1 wegen Schleife
9052  E5            400        PUSH  HL                 ; Y-Koordinate retten
9053  218091        410        LD    HL,SPRTAB-#80      ; mittels HL und BC
9056  018000        420        LD    BC,#80             ; den Tabellenanfang
9059  09            430 TABANF: ADD   HL,BC              ; bestimmen
905A  3D            440        DEC   A                  ; HL := SPRTAB +
905B  20FC          450        JR    NZ,TABANF          ; Spritenr. * 128
905D  E3            460        EX    (SP),HL            ; Y <--> Tabellenanfang
905E  CD1DBC        470        CALL  DOTPOS             ; HL=Adresse des Punktes
9061  EB            480        EX    DE,HL              ; DE=Adresse des Punktes
9062  E1            490        POP   HL                 ; HL:=Spritematrix
9063  CB41          500        BIT   0,C                ; X-Koordinate ungerade?
9065  3E10          510        LD    A,16               ; Spritehöhe = 16 Zeilen
9067  2012          520        JR    NZ,UNGERA          ; wenn X ungerade
9069  F5            530 SPZEIL: PUSH  AF                 ; wenn X gerade
906A  D5            540        PUSH  DE                 ; AF u. DE retten
906B  010800        550        LD    BC,8               ; 8 Byte in X-Richtung
906E  EDB0          560        LDIR                     ; Zeile --> Bildspeicher
9070  D1            570        POP   DE                 ; Adresse holen
9071  EB            580        EX    DE,HL              ; Bildadr.<->Spriteadr.
9072  CD26BC        590        CALL  NEXTLN             ; Adresse für nächste
9075  EB            600        EX    DE,HL              ; Spritezeile holen
9076  F1            610        POP   AF                 ; Zähler holen
9077  3D            620        DEC   A                  ; um 1 erniedrigen
9078  20EF          630        JR    NZ,SPZEIL          ; 16 Zeilen schreiben
907A  C9            640        RET
907B                650 UNGERA:                         ; wenn ungerade,
907B  F5            660 SPRZEI: PUSH  AF                 ; Zähler und Bild-
907C  D5            670        PUSH  DE                 ; speicheradr. retten
907D  1A            680 PIXEL1: LD    A,(DE)             ; Bildpunkt holen und
907E  E6AA          690        AND   #AA                ; Bits 7,5,3,1 wählen
9080  47            700        LD    B,A                ; zwischenspeichern
```

```
9081  7E        710           LD   A,(HL)          ; Spritepunkt holen
9082  E6AA      720           AND  #AA             ; Bits 7,5,3,1 wählen
9084  0F        730           RRCA                 ; nach rechts schieben
9085  B0        740           OR   B               ; mit gesp. Wert verknü.
9086  12        750           LD   (DE),A          ; --> Bildadr.
9087  13        760           INC  DE              ; nächste Bildadr.
9088  0607      770           LD   B,7             ; 7 Bytes = 14 Pixels
908A  7E        780 LOOP:     LD   A,(HL)          ; Byte nochmal holen
908B  E655      790           AND  #55             ; Bits 6,4,2,0 wählen
908D  07        800           RLCA                 ; nach links schieben
908E  4F        810           LD   C,A             ; speichern
908F  23        820           INC  HL              ; nächstes Spritebyte
9090  7E        830           LD   A,(HL)          ; holen und Bits
9091  E6AA      840           AND  #AA             ; 7,5,3,1 wählen
9093  0F        850           RRCA                 ; nach rechts schieben
9094  B1        860           OR   C               ; mit gesp. Wert verknü.
9095  12        870           LD   (DE),A          ; --> Bildspeicher
9096  13        880           INC  DE              ; nächstes Bildbyte
9097  10F1      890           DJNZ LOOP            ; 7mal wdh.
9099  7E        900 PIX_16:   LD   A,(HL)          ; letztes Spritebyte
909A  E655      910           AND  #55             ; Bits 6,4,2,0 wählen
909C  07        920           RLCA                 ; nach links schieben
909D  47        930           LD   B,A             ; speichern
909E  1A        940           LD   A,(DE)          ; Bildbyte holen
909F  E655      950           AND  #55             ; Bits 6,4,2,0 wählen
90A1  B0        960           OR   B               ; mit gesp. Wert verknü.
90A2  12        970           LD   (DE),A          ; --> Bildspeicher
90A3  23        980           INC  HL              ; nächstes Spritebyte
90A4  D1        990           POP  DE              ; Bildadresse holen
90A5  EB       1000           EX   DE,HL           ; Bildadr.<->Spriteadr.
90A6  CD26BC   1010           CALL NEXTLN          ; nächste Bildz. best.
90A9  EB       1020           EX   DE,HL           ; Spriteadr.<->Bildadr.
90AA  F1       1030           POP  AF              ; Zähler holen
90AB  3D       1040           DEC  A               ; um 1 erniedrigen
90AC  20CD     1050           JR   NZ,SPRZEI       ; 16mal wdh.
90AE  C9       1060           RET
              1070
90AF  DD7E00   1080 GT3ARG:   LD   A,(IX+0)        ; Spritenummer holen
90B2  E60F     1090           AND  #0F             ; zw. 0 und 15
90B4  DD4603   1100           LD   B,(IX+3)        ; Y-Koordinate ins
90B7  DD4E02   1110           LD   C,(IX+2)        ; BC-Register
90BA  DD5605   1120           LD   D,(IX+5)        ; X-Koordinate ins
90BD  DD5E04   1130           LD   E,(IX+4)        ; DE-Register
90C0  C9       1140           RET                  ; zurück
              1150
90C1  B7       1160 TSTARG:   OR   A               ; Carry löschen
90C2  217001   1170           LD   HL,400-32       ; Ymax
90C5  ED42     1180           SBC  HL,BC           ; Y < Ymax?
90C7  D8       1190           RET  C               ; sonst RETURN
90C8  B7       1200           OR   A               ; Carry löschen
90C9  214002   1210           LD   HL,640-64       ; Xmax
90CC  ED52     1220           SBC  HL,DE           ; X < Xmax?
90CE  D8       1230           RET  C               ; sonst RETURN
90CF  CB28     1240           SRA  B               ; BC-Register auf
90D1  CB19     1250           RR   C               ; Hälfte reduzieren
90D3  CB2A     1260           SRA  D               ; DE-Register auf
90D5  CB1B     1270           RR   E               ; ein Viertel
90D7  CB2A     1280           SRA  D
90D9  CB1B     1290           RR   E               ; reduzieren
90DB  210F00   1300           LD   HL,15           ; Y-Koordinate
90DE  09       1310           ADD  HL,BC           ; korrigieren
90DF  B7       1320           OR   A               ; Carry löschen
90E0  C9       1330           RET                  ; zurück
              1340
90E1  FE02     1350 SPRGEN:   CP   2               ; 2 Argumente?
90E3  C0       1360           RET  NZ              ; zurück, wenn nicht
90E4  DD7E00   1370           LD   A,(IX+0)        ; Spritenummer
90E7  E60F     1380           AND  #0F             ; Nr. zw. 0 und 15
90E9  47       1390           LD   B,A             ; --> B-Register
90EA  04       1400           INC  B               ; mind. 1 Durchlauf
90EB  218091   1410           LD   HL,SPRTAB-#80   ; aus Spritetabellen-
90EE  118000   1420           LD   DE,#80          ; anfang und Spritegröße
90F1  19       1430 TABBEG:   ADD  HL,DE           ; den Anfang des Sprites
90F2  10FD     1440           DJNZ TABBEG          ; berechnen
90F4  DD5603   1450           LD   D,(IX+3)        ; Adresse der Matrix
```

Abb. 3.21: Sprite-Programm (Fortsetzung)

```
90F7  DD5E02    1460          LD   E,(IX+2)          ; ins DE-Register
90FA  EB        1470          EX   DE,HL            ; Matrix und Tabellen-
                1480                                  adresse vertauschen
90FB  0680      1490          LD   B,128            ; 128 BYTE pro Sprite
90FD  7E        1500 INKLOP:  LD   A,(HL)           ; Matrixink holen
90FE  CD2CBC    1510          CALL INKCOD           ; codieren und
9101  E6AA      1520          AND  #AA              ; Bits 7,5,3,1 wählen
9103  4F        1530          LD   C,A              ; zwischenspeichern
9104  23        1540          INC  HL               ; nächsten Matrixwert
9105  23        1550          INC  HL               ; wählen und
9106  7E        1560          LD   A,(HL)           ; Matrixink holen
9107  CD2CBC    1570          CALL INKCOD           ; Ink codieren und
910A  E655      1580          AND  #55              ; Bits 6,4,2,0 wählen
910C  B1        1590          OR   C                ; mit vorigem Wert ODER
910D  12        1600          LD   (DE),A           ; verknüpfen,abspeichern
910E  13        1610          INC  DE               ; Zieladresse + 1
910F  23        1620          INC  HL               ; Quelladresse + 2
9110  23        1630          INC  HL               ; bestimmen
9111  10EA      1640          DJNZ INKLOP           ; 128mal wiederholen
9113  C9        1650          RET                   ; zurück zum BASIC
```

Abb. 3.21: Sprite-Programm (Fortsetzung)

Programmbeschreibung

Wir wollen zunächst die SPRITEON-Routine beschreiben. Die SPRITEON-Routine benötigt drei Parameter, zuerst die X-Koordinate, dann die Y-Koordinate und die Sprite-Nummer. Beim Eintritt in das Programm werden die Koordinaten überprüft und, wenn sie nicht im zulässigen Bereich liegen, ein RETurn zum BASIC ausgeführt, also keine Operation durchgeführt. Das gleiche passiert, wenn weniger oder mehr als drei Parameter übergeben werden.

Die Sprite-Nummer kann jeden beliebigen Wert annehmen, wobei aber nur Werte zwischen 0 und 15 sinnvoll sind, da das Programm immer nur die untersten vier Bits selektiert, um eine Sprite-Nummer zwischen 0 und 15 zu gewährleisten. Aus dieser Sprite-Nummer wird dann der Tabellenanfang für dieses Sprite berechnet. Der Tabellenanfang der Sprite-Tabelle liegt bei &9200, und jedes Sprite belegt 128 Bytes in dieser Tabelle, d. h. Sprite 0 belegt &9200 bis &927F, Sprite 1 belegt &9280 bis &92FF und so fort. Das Lösch-Sprite belegt den Bereich von &9A00 bis &9A7F. Dieser Bereich sollte in einem BASIC-Programm mit Nullen initialisiert werden, damit ein Löschen gewährleistet ist.

Nach der Berechnung des Tabellenanfangs werden die korrigierten X- und Y-Koordinaten an eine Routine des CPC-Betriebssystems übergeben (&BC1D), welche die zu diesen Koordinaten gehörende Speicheradresse zurückliefert und zugleich angibt, wie viele Pixels in ein Byte passen, und die für diesen Punkt entsprechende Maske angibt. Diese Maske kann auf eine ungerade Punktposition untersucht werden, und damit können unterschiedliche Routinen zur Ausgabe angesprungen werden.

Diese unterschiedlichen Routinen sind notwendig, da bei einer geraden Speicheradresse des Sprite Byte für Byte und Zeile für Zeile in den Speicher geschrieben werden kann und man damit keine vorhandenen Bildpunkte zu berücksichtigen braucht. Bei ungeraden Speicheradressen muß jedoch bei dem ersten und dem letzten Byte nur ein Teil, nämlich vier Bits, in den Speicher geschrieben werden.

Die mittleren vierzehn Bildpunkte (7 Bytes) müssen allerdings ebenfalls behandelt werden. Der als Bildpunkt höherwertige Teil eines nachfolgenden Bytes muß zum niederwertigen Teil eines vorangegangenen Bytes verschoben und mit diesem zusammen in den Bildspeicher eingeschrieben werden, wenn nicht doppelt soviel Platz zur Ablage des Sprite verwendet werden soll.

Die erste Routine für das Einschreiben bei einer geraden Adresse kann auf dem Z80 sehr einfach mit dem Blockverschiebebefehl LDIR gelöst werden. Nach der Übertragung von je acht Bytes (16 Pixels) muß die vor Ausführung der Blockverschiebung abgespeicherte Bildschirmadresse wiedergeholt werden und mittels der CPC-Routine &BC26 auf die Adresse der nächsten darunterliegenden Zeile korrigiert werden. Dieser Vorgang muß mit der Blockverschiebung sechzehnmal durchlaufen werden, um die 16×16 Bildpunkte zu übertragen.

Etwas Ähnliches geschieht bei ungerade liegenden Pixels. Jedoch wird bei jeder Zeile zuerst der vorherige Wert der ersten Speicherposition ausgelesen, die Bits 7, 5, 3, 1 selektiert, zwischengespeichert, vom ersten Byte des Sprite die Bits 7, 5, 3, 1 selektiert, eine Position nach rechts verschoben und mit den zwischengespeicherten Bits des Bildschirmspeichers mit Hilfe der ODER-Funktion verknüpft und wieder in den Bildspeicher eingeschrieben. Die nächsten sieben Bytes werden gleich behandelt. Es wird immer ein Byte eingelesen, die Bits 6, 4, 2, 0 selektiert, eine Position nach links geschoben, zwischengespeichert, das nächste Byte des Sprite eingelesen, die Bits 7, 5, 3, 1 selektiert, nach rechts geschoben und mit den zuvor verschobenen Bits anhand der ODER-Funktion verknüpft.

Der entstandene Wert wird in den Bildspeicher kopiert und dieser Vorgang siebenmal wiederholt. Der letzte Bildpunkt wird wieder gesondert behandelt, und zwar in einer dem ersten entgegengesetzten Weise. Vom Sprite werden dazu die Bits 6, 4, 2, 0 selektiert, nach links verschoben, vom Bildspeicher die Bits 6, 4, 2, 0 geholt, mit ODER verknüpft und in den Bildspeicher zurückgeschrieben.

Dieser gesamte Vorgang wiederholt sich bei jeder Zeile des Sprite. Zum Löschen eines Sprite mit dem externen Befehl SPRITEOFF werden nur die

Koordinaten des zu löschenden Sprite überreicht. Beim Löschvorgang wird eigentlich nur ein Sprite mit keiner Belegung benutzt, und zwar das Sprite 16.

Zur Generierung eines Sprite müssen die Farbwerte in einer Integer-Matrix der Form (X, Y) abgelegt werden, wobei die maximalen Indizes für X und Y die Werte 15 annehmen dürfen. Dem externen Befehl SPRITEGEN wird die Adresse der Integer-Matrix und die Nummer des gewünschten Sprite angegeben. Sind z. B. in einer Matrix mit der Dimensionierung DIM sprite%(15,15,9) die Plätze sprite%(x,y,0) bis sprite %(x,y,4) belegt, so entsprechen die X- und Y-Indizes 0 der linken oberen Ecke. Dementsprechend müssen die Farbwerte darin abgelegt werden. Soll nun z. B. das Sprite 0 mit der Matrix sprite%(x,y,3) belegt werden, so ist SPRITEGEN folgendermaßen aufzurufen:

```
ISPRITEGEN, @ sprite%(0,0,3),0
```

(Den senkrechten Strich vor dem SPRITEGEN erreicht man durch Drücken von Shift @.)

In der Praxis ist es manchmal ratsam, dem SPRITEON- bzw. dem SPRITEOFF-Befehl ein CALL &BD19 voranzustellen, was eine Zeitverzögerung bis zum nächsten Bildrücklauf bewirkt.

Noch etwas ist zu beachten: Der Befehl SPRITEGEN arbeitet nur im MODE 0 exakt, da er die CPC-Routine zur Ink-Codierung verwendet.

Sollte der nachfolgende BASIC-Loader benutzt werden, so empfiehlt es sich, ihn zu eigenen Programmen mittels MERGE an das Programm zu binden und mit einem RETURN am Ende zu versehen. Bereits generierte Sprites sollte man mit

```
SAVE "SPRITES", B,&9200,&800
```

abspeichern, um sie bei Programmen einfach dazuladen zu können.

Abb. 3.22: BASIC-Lader für die Sprite-Programme

```
14500 ' Der nachstehende BASIC-Lader ist für 3 Spritepr
ogramme vorhanden.
14510 ' Diese Spriteprogramme werden, wenn das Programm
 mit CALL &9000
14520 ' aufgerufen wurde, als Befehlserweiterung des CP
C betrachtet.
14530 ' Die Befehle heißen SPRITEON, SPRITEOFF und SPRI
TEGEN und werden
14540 ' alle durch SHIFT 'Klammeraffe'"BEFEHLSNAME",Par
ameterliste aufgerufen.
14550 ' Die Parameter sind :
14560 ' bei SPRITEGEN,'Klammeraffe'Integermatrix(0,0),S
pritenr.  ;wobei die
14570 ' Integermatrix mit DIM (15,15) dimensioniert wur
de und die Farbnummer
```

```
14580 ' für die jeweilige Position enthält ( 0,0 ist li
nks oben ; erster
14590 ' Index ist X-Koordinate ; zweiter ist Y-Koordina
te ) .
14600 ' Die Sprites werden dann ab &9200 abgelegt und j
eder Sprite belegt
14610 ' 128 Bytes im Speicher. Die Spritetabelle geht b
is &9A7F
14620 ' SPRITEON,X,Y,Spritenr. ; wobei x < 576 und => 0
 sein muß , sowie
14630 ' y <368 und => 0 sein muß. Die Spritenr. muß zw.
 0 und 15 liegen
14640 ' SPRITEOFF,X,Y ; für die Koordinaten gilt diesel
be Einschränkung wie
14650 ' bei dem SPRITEON-Befehl.
14660 ' ----------------------------------------------
------------------
14670 ' Ein wichtiger Hinweis:
14680 ' Alle Befehle arbeiten nur im MODE 0 ordnungsgem
äß!!!!!!
14690 '
15000 MEMORY &8FFF
15010 FOR n%=&9000 TO &9113
15020     READ byte$
15030     byte=VAL("&"+byte$)
15040     POKE n%,byte
15050     summe=summe+byte
15060 NEXT
15070 IF summe<>&7A56 THEN PRINT" Spriteprogramme nicht
 korrekt geladen!":END
15080 DATA 01,0A,90,21,30,90,CD,D1,BC,C9,15,90,C3,47,90
,C3,34,90,C3,E1,90
15090 DATA 53,50,52,49,54,45,4F,CE,53,50,52,49,54,45,4F
,46,C6
15100 DATA 53,50,52,49,54,45,47,45,CE,00,00,00,00,00,FE
,02,C0,3E,10,DD,46,01
15110 DATA DD,4E,00,DD,56,03,DD,5E,02,18,06,FE,03,C0,CD
,AF,90,CD,C1,90
15120 DATA D8,3C,E5,21,80,91,01,80,00,09,3D,20,FC,E3,CD
,1D,BC,EB,E1,CB,41
15130 DATA 3E,10,20,12,F5,D5,01,08,00,ED,B0,D1,EB,CD,26
,BC,EB,F1,3D,20,EF
15140 DATA C9,F5,D5,1A,E6,AA,47,7E,E6,AA,0F,B0,12,13,06
,07,7E,E6,55,07,4F
15150 DATA 23,7E,E6,AA,0F,B1,12,13,10,F1,7E,E6,55,07,47
,1A,E6,55,B0,12,23
15160 DATA D1,EB,CD,26,BC,EB,F1,3D,20,CD,C9,DD,7E,00,E6
,0F,DD,46,03,DD,4E,02
15170 DATA DD,56,05,DD,5E,04,C9,B7,21,70,01,ED,42,D8,B7
,21,40,02,ED,52,D8
15180 DATA CB,28,CB,19,CB,2A,CB,1B,CB,2A,CB,1B,21,0F,00
,09,B7,C9,FE,02,C0
15190 DATA DD,7E,00,E6,0F,47,04,21,80,91,11,80,00,19,10
,FD,DD,56,03,DD,5E,02
15200 DATA EB,06,80,7E,CD,2C,BC,E6,AA,4F,23,23,7E,CD,2C
,BC,E6,55,B1
15210 DATA 12,13,23,23,10,EA,C9
```

Abb. 3.22: BASIC-Lader für die Sprite-Programme (Fortsetzung)

HARDCOPY

Bevor wir zur Beschreibung der Hardcopy-Programme kommen, soll der Begriff Hardcopy und der Nutzen von Hardcopy erklärt werden: Eine Hardcopy ist ein möglichst wirklichkeitsgetreuer Abzug des Bildschirminhalts. Mit einer Hardcopy ist es möglich, Grafiken oder Text direkt vom Bildschirm auf den Drucker auszugeben.

Es ergeben sich zwei Möglichkeiten von Hardcopies:
1. Text-Hardcopy
2. Grafik-Hardcopy

Sie müssen beim CPC und den meisten anderen Computern auch getrennt realisiert werden.

Bei der Textvariante muß nur immer ein Zeichen vom Bildschirm gelesen werden, während bei der Grafikroutine jedes Pixel des Bildschirms ausgelesen werden muß. Bei vielen Computern bereitet die erste Variante keine Schwierigkeiten, da oft ein Zeichen einem Byte im Bildspeicher entspricht. Beim Schneider hingegen werden die Zeichen immer in den Grafikspeicher eingetragen, d. h. im MODE 2 werden 8, im MODE 1 werden 16 und im MODE 0 gar 32 Bytes benötigt, um ein Zeichen darzustellen.

Es wäre vielleicht schwierig, eine Routine zu schreiben, die diese Ablage von Zeichen berücksichtigt und sie in Abhängigkeit vom Anfang des Bildschirmspeichers erkennt und in den ASCII-Code umwandelt, wenn uns der CPC in seinem Betriebssystem nicht eine solche Routine zur Verfügung stellen würde. Diese Routine ist vielen sicher von der Anwendung des COPY-Cursors her geläufig. Sie untersucht die aktuelle Cursorposition im aktuellen Fenster auf ein Zeichen und liefert, wenn ein solches erkennbar war, die Nummer dieses Zeichens im Akku, einem Register des Z80-Prozessors, zurück. War kein Zeichen erkennbar, eventuell wegen einem Grafikteil im Feld eines Zeichens oder wegen geänderter Hintergrund- oder Schreibstiftfarbe, so vermerkt die Routine dies im Carry-Flag (Bit zur Kennzeichnung eines Übertrags) des Prozessors. Doch jetzt wollen wir zur Programmbeschreibung der verschiedenen Hardcopy-Routinen kommen.

Text-Hardcopy

Zunächst möchten wir Ihnen das Listing vorstellen (Abb. 3.23).

Abb. 3.23: Hardcopy für Texte

```
                    10 *H Texthardcopyprogramm für den CPC464
A000                20          ORG   #A000
BB69                30 GETWIN:  EQU   #BB69
BB66                40 SETWIN:  EQU   #BB66
BB78                50 GETCUR:  EQU   #BB78
BB75                60 SETCUR:  EQU   #BB75
BB60                70 READXY:  EQU   #BB60
BC17                80 GETWID:  EQU   #BC17
                    90
BB1B               100 GETKEY:  EQU   #BB1B
00FC               110 BREAK:   EQU   #FC
                   120
BD2E               130 TESTPR:  EQU   #BD2E
BD2B               140 SENDPR:  EQU   #BD2B
000D               150 CR:      EQU   #0D
```

```
000A                160 LF:     EQU  #0A
                    170
A000  CD78BB        180         CALL GETCUR             ; Cursorpos. abfragen
A003  E5            190         PUSH HL                 ; und sichern
A004  CD69BB        200         CALL GETWIN             ; aktuelles Fenster
A007  E5            210         PUSH HL                 ; abfragen und
A008  D5            220         PUSH DE                 ; sichern
A009  ED7370A0      230         LD   (STACKPT),SP       ; Stackpointer sichern
A00D  CD17BC        240         CALL GETWID             ; Bildmaße abfragen
A010  C5            250         PUSH BC                 ; Bildbreite retten
A011  C5            260         PUSH BC                 ; BC nach DE kopieren
A012  D1            270         POP  DE                 ; Fenster auf volle
A013  210000        280         LD   HL,0000            ; Bildgröße
A016  CD66BB        290         CALL SETWIN             ; setzen
A019  C1            300         POP  BC                 ; Bildbreite holen
A01A  04            310         INC  B                  ; Spaltenzahl --> B
A01B  48            320         LD   C,B                ; Kopie --> C
A01C  2601          330         LD   H,1                ; mit linkem oberen
A01E  2E01          340         LD   L,1                ; Zeichen anfangen
A020  CD65A0        350         CALL PRCRLF             ; Zeilenvorschub drucken
A023  C5            360 NXTCHR: PUSH BC                 ; BC und HL auf
A024  E5            370         PUSH HL                 ; Stack retten
A025  CD75BB        380         CALL SETCUR             ; Cursor positionieren
A028  CD60BB        390         CALL READXY             ; Zeichen von H := X
                    400 ;                                 und L := Y einlesen
A02B  3802          410         JR   C,GUELTIG          ; Zeichen erkannt?
A02D  3E20          420         LD   A," "              ; sonst Leerzeichen
A02F  CD45A0        430 GUELTI: CALL PRCHR              ; Zeichen drucken
A032  E1            440         POP  HL                 ; HL und BC vom
A033  C1            450         POP  BC                 ; Stack holen
A034  24            460         INC  H                  ; nächste Spalte
A035  10EC          470         DJNZ NXTCHR             ; wenn nicht, Zeilenende
A037  CD65A0        480         CALL PRCRLF             ; sonst Zeilenvorschub
A03A  41            490         LD   B,C                ; Zeilenlänge neu laden
A03B  2601          500         LD   H,1                ; erste Spalte = 1
A03D  3E19          510         LD   A,25
A03F  BD            520         CP   L                  ; letzte Zeile?
A040  280B          530         JR   Z,QUIT             ; --> Hardcopy verlassen
A042  2C            540         INC  L                  ; sonst nächste Zeile
A043  18DE          550         JR   NXTCHR             ; und nächstes Zeichen
                    560
A045  57            570 PRCHR:  LD   D,A                ; Zeichen retten
A046  CD1BBB        580 WAITPR: CALL GETKEY
A049  FEFC          590         CP   BREAK              ; ESC-Taste testen
A04B  200E          600         JR   NZ,NOBRK
A04D  ED7B70A0      610 QUIT:   LD   SP,(STACKPT)       ; bei BREAK oder Ende,
                    620 ;                                 Stackpointer auf
                    630 ;                                 Eintrittswert setzen
A051  D1            640         POP  DE                 ; altes Fenster
A052  E1            650         POP  HL                 ; wenn gedrückt dann
A053  CD66BB        660         CALL SETWIN             ; herstellen und
A056  E1            670         POP  HL                 ; alte Cursorpos.
A057  CD75BB        680         CALL SETCUR             ; wieder setzen
A05A  C9            690         RET                     ; Hardcopy verlassen!
A05B  CD2EBD        700 NOBRK:  CALL TESTPR             ; sonst wieder Druck-
A05E  38E6          710         JR   C,WAITPR           ; bereitschaft testen
A060  7A            720         LD   A,D                ; Zeichen holen
A061  CD2BBD        730         CALL SENDPR             ; und ausgeben
A064  C9            740         RET
                    750
A065  3E0D          760 PRCRLF: LD   A,CR               ; CR und
A067  CD45A0        770         CALL PRCHR
A06A  3E0A          780         LD   A,LF               ; LF ausgeben
A06C  CD45A0        790         CALL PRCHR              ; = Zeilenvorschub
A06F  C9            800         RET                     ; Rückkehr
                    810
A070                820 STACKP: DEFS 2                  ; Speicher für Stack
```

Abb. 3.23: Hardcopy für Texte (Fortsetzung)

Programmbeschreibung

Zuerst werden die Maße des aktuellen Fensters und die Cursorposition auf den Stack gerettet und der Stackpointer in zwei Speicherzellen abgelegt. Danach wird das Fenster auf Maximalgröße gesetzt. Die maximale Spalten- und Zeilenzahl erhält man mit dem Aufruf von &BC17.

Nun kann mit der Cursorsetzroutine &BB75, äquivalent zum LOCATE-Befehl aus BASIC, der Cursor an alle Positionen des Bildschirms gesetzt werden, das Zeichen mittels der Routine BB60H ausgelesen und bei Erkennungsfehler durch ein Leerzeichen ersetzt werden. Dieses Zeichen wird dann jeweils durch eine Ausgaberoutine an den Drucker überreicht, wenn dieser bereit ist, ein Zeichen zu empfangen, und die ESC-Taste nicht betätigt wurde.

Nach dem Überprüfen von je einer Bildschirmzeile wird ein Zeilenvorschub mit Wagenrücklauf ausgedruckt und die nächste Zeile des Bildschirms untersucht, bis die letzte Zeile überprüft wurde. Dann erfolgt ein weiterer Zeilenvorschub auf dem Drucker, der Stackpointer wird auf den abgespeicherten Wert gesetzt, um an die Werte für die Cursorposition und die Fenstergröße heranzukommen. Das Fenster wird wieder auf Eintrittsgröße und der Cursor auf die Eintrittsposition gesetzt.

Danach kehrt das Programm zur aufrufenden Routine zurück, also in den meisten Fällen in ein BASIC-Programm. Sollte während des Programms die ESC-Taste gedrückt worden sein, so werden die Fenstergröße und die Cursorposition wieder hergestellt und die Routine verlassen.

Grafik-Hardcopy

Bei der Grafik-Hardcopy gibt es zwei Probleme, die von dem 7 Bit breiten Centronics-Port des CPC herrühren:

1. Es können nur 7 Bits für Grafik ausgegeben werden.
2. Die Breite der Hardcopy muß auf 640 Pixels eingestellt werden. 640 ist aber 2*256 + 128, und 128 kommt aus dem Port als 0 heraus.

Das erste Problem kann einfach bewältigt werden: Man muß nur den Zeilenvorschub des Druckers auf sieben horizontale Punkte beschränken. Dies geschieht durch die Ausgabe von ESC + "1". Durch diese Steuersequenz wird der Drucker auf 7/72 inch Zeilenvorschub (7 Dots) eingestellt.

Das zweite Problem läßt sich auf zwei verschiedene Weisen bewältigen, einmal durch die Beschränkung auf 639 Pixels, denn 639 = 2*256 + 127, und zweitens durch Angabe von mehr Pixeln. Geeignet sind z. B. 768 = 3*256 Pixels. Die weiteren Pixels müssen nur zusätzlich ausgegeben werden.

Abb. 3.24: Hardcopy für Grafik

```
                        10 *H Grafikhardcopyprogramm für den CPC464
A080                    20         ORG  #A080
BBCC                    30 GETORG: EQU  #BBCC
BBC9                    40 SETORG: EQU  #BBC9
BBCF                    50 SETX12: EQU  #BBCF
BBD2                    60 SETY12: EQU  #BBD2
BBD5                    70 GETX12: EQU  #BBD5
BBD8                    80 GETY12: EQU  #BBD8
BBC6                    90 GETPOS: EQU  #BBC6
BBC0                   100 MOVEXY: EQU  #BBC0
BBDE                   110 SETPEN: EQU  #BBDE
BBE1                   120 GETPEN: EQU  #BBE1
BBE4                   130 SETPAP: EQU  #BBE4
BBE7                   140 GETPAP: EQU  #BBE7
                       150
BBF0                   160 TESTXY: EQU  #BBF0
BBBA                   170 GRINIT: EQU  #BBBA
                       180
BB1B                   190 GETKEY: EQU  #BB1B
00FC                   200 BREAK:  EQU  #FC
                       210
BD2E                   220 TESTPR: EQU  #BD2E
BD2B                   230 SENDPR: EQU  #BD2B
001B                   240 ESC:    EQU  #1B
000D                   250 CR:     EQU  #0D
000A                   260 LF:     EQU  #0A
                       270
A080  CDC6BB           280         CALL GETPOS             ; Cursorpos. abfragen
A083  E5               290         PUSH HL                 ; X- und Y-Koordinate
A084  D5               300         PUSH DE                 ; sichern
A085  CDCCBB           310         CALL GETORG             ; Origin abfragen
A088  E5               320         PUSH HL                 ; X- und Y-Koordinate
A089  D5               330         PUSH DE                 ; sichern
A08A  CDD5BB           340         CALL GETX12             ; Grafikfensterbreite
A08D  E5               350         PUSH HL                 ; abfragen und X1 und
A08E  D5               360         PUSH DE                 ; X2 sichern
A08F  CDD8BB           370         CALL GETY12             ; Grafikfensterhöhe
A092  E5               380         PUSH HL                 ; abfragen und Y1 und
A093  D5               390         PUSH DE                 ; Y2 sichern
A094  CDE1BB           400         CALL GETPEN             ; Grafikpen holen
A097  47               410         LD   B,A                ; Pen --> B
A098  CDE7BB           420         CALL GETPAP             ; Grafikpaper holen
A09B  4F               430         LD   C,A                ; Paper --> C
A09C  C5               440         PUSH BC                 ; Paper u. Pen sichern
A09D  ED7360A1         450         LD   (STACKPT),SP       ; Stackpointer sichern
A0A1  CDBABB           460         CALL GRINIT             ; Reset des Grafiksys.
A0A4  3E1B             470         LD   A,ESC              ; Drucker auf 7/72"
A0A6  CD13A1           480         CALL PRCHR              ; Zeilenvorschub
A0A9  3E31             490         LD   A,"1"              ; einstellen
A0AB  CD13A1           500         CALL PRCHR
A0AE  218F01           510         LD   HL,399             ; oben anfangen (Y:=399)
A0B1                   520 NXTLIN:                         ; Zeile ausgeben
A0B1  CD47A1           530 GRAINI: CALL PRCRLF             ; Zeilenvorschub
A0B4  3E1B             540         LD   A,ESC              ; Drucker für die
A0B6  CD13A1           550         CALL PRCHR              ; Ausgabe von
A0B9  3E4C             560         LD   A,"L"              ; Grafik mit
A0BB  CD13A1           570         CALL PRCHR              ; 768 Punkten
A0BE  3E00             580         LD   A,0                ; = 0 Punkte +
A0C0  CD13A1           590         CALL PRCHR
A0C3  3E03             600         LD   A,3                ; + 3*256 Punkte
A0C5  CD13A1           610         CALL PRCHR              ; vorbereiten
A0C8  0680             620         LD   B,128              ; 128 * CHR$(0)
A0CA  3E00             630 LOOP:   LD   A,0                ; ausgeben
A0CC  CD13A1           640         CALL PRCHR              ; --> noch 640 Punkte
A0CF  10F9             650         DJNZ LOOP               ; übrig
A0D1  110000           660         LD   DE,0               ; links anfangen (X:=0)
A0D4  E5               670 XLOOP:  PUSH HL                 ; Y sichern
A0D5  0607             680         LD   B,7                ; Zähler für 7 Punkte
A0D7  0E00             690         LD   C,0                ; CHR$ ist erst 0
A0D9  E5               700 DTLOOP: PUSH HL                 ; Y retten
A0DA  CD52A1           710         CALL DOTTST             ; Punkt testen
A0DD  CB11             720         RL   C                  ; Carry ins unterste Bit
A0DF  E1               730         POP  HL                 ; v. C rotieren, Y holen
A0E0  2B               740         DEC  HL                 ; Y um einen Pixel
```

```
A0E1  2B        750          DEC  HL              ; = 2 Punkte erniedrigen
A0E2  10F5      760          DJNZ DTLOOP          ; nächster der 7 Punkte
A0E4  79        770          LD   A,C             ; CHR$ --> A
A0E5  E1        780          POP  HL              ; Y holen
A0E6  E5        790          PUSH HL              ; und wieder sichern
A0E7  010800    800          LD   BC,8            ; Sind es die
A0EA  ED42      810          SBC  HL,BC           ; 4 letzten Dotreihen?
A0EC  3002      820          JR   NC,NOLAST       ; ja? Dann die obersten
A0EE  E678      830          AND  %1111000        ; 4 Druckbits selekt.
A0F0  CD13A1    840 NOLAST:  CALL PRCHR           ; CHR$ ausgeben
A0F3  13        850          INC  DE              ; X := X + 1
A0F4  218102    860          LD   HL,641          ; hat X-Koordinate
A0F7  ED52      870          SBC  HL,DE           ; bereits 641 erreicht
A0F9  E1        880          POP  HL              ; (Y holen)
A0FA  20D8      890          JR   NZ,XLOOP        ; sonst nächster X-Wert
A0FC  B7        900          OR   A               ; Carry löschen
A0FD  010E00    910          LD   BC,14           ; BC:=14 (= 7 Pixel)
A100  ED42      920          SBC  HL,BC           ; Y:=Y-14, wenn Y>0
A102  30AD      930          JR   NC,NXTLIN       ; dann nächste Zeile
A104  3E1B      940 ENDE:    LD   A,ESC           ; sonst Zeilenvorschub
A106  CD13A1    950          CALL PRCHR           ; auf 1/6 inch
A109  3E32      960          LD   A,"2"           ; zurückstellen
A10B  CD13A1    970          CALL PRCHR
A10E  CD47A1    980          CALL PRCRLF          ; und Zeilenvorschub
A111  1808      990          JR   QUIT            ; Hardcopy verlassen
              1000
A113  4F        1010 PRCHR:  LD   C,A             ; Zeichen retten
A114  CD1BBB    1020 WAITPR: CALL GETKEY          ; Taste holen
A117  FEFC      1030         CP   BREAK           ; ESC-Taste testen
A119  2022      1040         JR   NZ,NOBRK
A11B  ED7B60A1  1050 QUIT:   LD   SP,(STACKPT)    ; Bei BREAK oder Ende
                1060 ;                             Stackpointer auf
                1070 ;                             Eintrittswert setzen
A11F  C1        1080         POP  BC              ; Paper und Pen holen
A120  79        1090         LD   A,C             ; Paper --> A
A121  CDE4BB    1100         CALL SETPAP          ; Grafikpaper setzen
A124  78        1110         LD   A,B             ; Pen --> A
A125  CDDEBB    1120         CALL SETPEN          ; Grafikpen setzen
A128  D1        1130         POP  DE              ; Y2 und Y1 holen u.
A129  E1        1140         POP  HL              ; die Grafikfensterhöhe
A12A  CDD2BB    1150         CALL SETY12          ; wieder herstellen
A12D  D1        1160         POP  DE              ; X2 und X1 holen u.
A12E  E1        1170         POP  HL              ; die Grafikfensterbr.
A12F  CDCFBB    1180         CALL SETX12          ; wieder herstellen
A132  D1        1190         POP  DE              ; alten Origin
A133  E1        1200         POP  HL              ; holen und
A134  CDC9BB    1210         CALL SETORG          ; herstellen
A137  D1        1220         POP  DE              ; alte Grafikpos.
A138  E1        1230         POP  HL              ; holen und
A139  CDC0BB    1240         CALL MOVEXY          ; wieder setzen
A13C  C9        1250         RET                  ; Hardcopy verlassen!
A13D  CD2EBD    1260 NOBRK:  CALL TESTPR          ; sonst wieder Druck-
A140  38D2      1270         JR   C,WAITPR        ; bereitschaft testen
A142  79        1280         LD   A,C             ; wenn bereit, Zeichen
A143  CD2BBD    1290         CALL SENDPR          ; holen und ausgeben
A146  C9        1300         RET                  ; und Rückkehr
              1310
A147  3E0D      1320 PRCRLF: LD   A,CR            ; CR und
A149  CD13A1    1330         CALL PRCHR
A14C  3E0A      1340         LD   A,LF            ; LF ausgeben
A14E  CD13A1    1350         CALL PRCHR           ; = Zeilenvorschub
A151  C9        1360         RET                  ; Rückkehr
              1370
A152  C5        1380 DOTTST: PUSH BC              ; BC, DE und HL
A153  D5        1390         PUSH DE              ; auf den Stack
A154  E5        1400         PUSH HL              ; retten
A155  CDF0BB    1410         CALL TESTXY          ; Punkt (DE,HL) testen
A158  B7        1420         OR   A               ; Carry löschen u. A=0?
A159  2801      1430         JR   Z,NOTSET        ; A=0, dann nichts tun
A15B  37        1440         SCF                  ; sonst Carry setzen
A15C  E1        1450 NOTSET: POP  HL              ; HL, DE und BC
A15D  D1        1460         POP  DE              ; wieder vom Stack
A15E  C1        1470         POP  BC              ; holen
A15F  C9        1480         RET                  ; und Rückkehr
              1490
A160            1500 STACKP: DEFS 2               ; Speicher für Stack
```

Abb. 3.24: Hardcopy für Grafik (Fortsetzung)

Programmbeschreibung

Im vorliegenden Hardcopy-Programm wurde der zweite Weg beschritten. Die 128 Pixels zuviel werden am Anfang jeder Zeile als 0 ausgegeben, wodurch die Grafik auf dem Papier ziemlich in der Mitte ausgegeben wird.

Wie bei der Texthardcopy werden beim Eintritt in das Programm erst einmal die Zustände der Grafikroutinen des CPC abgespeichert. Diese Zustände sind:

- die Grafikcursorposition
- die Länge des Ursprungs
- die Breitenkoordinaten des Grafikfensters
- die Höhenkoordinaten des Grafikfensters
- sowie Farbe von Grafikstift und Hintergrund

Danach wird wieder der Stackpointer gespeichert, um definierte Austrittsbedingungen zu gewährleisten, und schließlich wird ein RESET des Grafiksystems durchgeführt, um sämtliche Grafikfensterdefinitionen zu löschen. Danach wird der Drucker auf 7/72 inch Zeilenvorschub eingestellt und die eigentliche Ausgabeschleife betreten.

Für die Grafikausgabe wird nun erst ein Zeilenvorschub auf den Drucker ausgegeben und der Drucker dann auf Grafik mit 768 Punkten eingestellt, wovon sofort die ersten 128 überreicht werden, bevor mit der TESTROUTINE die Grafikpixels abgetastet werden. Bei der Ermittlung der jeweils gesetzten Punkte wird in der oberen linken Ecke des Bildschirms begonnen und mit der TEST ABSOLUT-Routine (&BBF0) jeweils sieben untereinander liegende Punkte abgetastet.

Ist der Punkt nicht mit NK 0 gesetzt, so wird das Carry-Flag gesetzt, in das Register C des Z80 nach links eingeschoben und der Y-Wert um zwei (1 Pixel) vermindert. Ist dies siebenmal durchgeführt worden, so wird überprüft, ob es sich um die letzten vier Y-Reihen handelt, wenn ja, so werden die unteren drei Bits auf 0 gesetzt. Dann wird der Wert von C an den Drucker ausgegeben. Diese Prozedur wird 640mal wiederholt, wobei jedesmal ein um eins erhöhter X-Wert benutzt wird.

Anschließend wird ein Zeilenvorschub ausgegeben, der Y-Wert um vierzehn (7 Pixels) vermindert und, wenn es nicht die letzten vier Y-Pixels waren, die nächsten sieben Bildschirmzeilen abgetastet und ausgegeben. Ansonsten wird der Drucker wieder auf einen Zeilenvorschub von 1/6 inch über ESC + "2" zurückgestellt, ein Zeilenvorschub ausgegeben und die Hardcopy über die Routine für Unterbrechungen verlassen.

Die Unterbrechungsroutine wird ebenfalls angesprungen, wenn während der Druckerausgabe die ESC-Taste gedrückt wurde. In diesem Fall wird der Stackpointer auf den abgespeicherten Wert geladen und folgende Zustände des Grafiksystems wiederhergestellt:

- Farbe des Grafikstifts und des Hintergrunds
- Höhe des Grafikfensters
- Breite des Grafikfensters
- Ursprung der Grafik
- Grafikcursorposition

Da eine Unterbrechung im Normalfall während der Ausgabe erfolgt, sollte der Drucker bei nicht ordnungsgemäßem Programmende kurz ausgeschaltet werden, da weder der Zeilenvorschub umgestellt, noch die Grafikoption des Druckers ausgeschaltet wurde.

Sofern Sie keinen Assembler besitzen, finden Sie im nächsten Kapitel einen BASIC-Lader für beide Hardcopy-Routinen.

BASIC-Lader

Abb. 3.25: BASIC-Lader

```
10000 ' Hardcopyprogramme für den CPC 464 mit Drucker N
LQ 401,EPSON FX/RX
10010 ' und kompatible (z.B.BMC BX-100). Die Hardcopypr
ogramme werden mit dem
10020 ' CALL-Befehl aufgerufen. Eventuell definierte Wi
ndows, sowie Text-
10030 ' und Grafikcursorposition bleiben ebenso wie die
 ORIGIN-Definitionen
10040 ' erhalten. Der Abbruch der Programme kann jederz
eit durch Drücken der
10050 ' ESC-Taste erfolgen. Das Programm kehrt dann zur
 aufrufenden Routine
10060 ' zurück. Es werden nur ASCII-Zeichen zw. 0 und 1
27 ausgegeben. Die
10070 ' Zeichen zw. 0 und 31 sind allerdings Steuerzeic
hen für die jeweiligen
10080 ' Drucker. Die Ausgabe der Grafikzeichen des CPC
464 kann nur durch
10090 ' Umbau des Centronics-Ports oder die Graphikhard
copy erfolgen.
10100 ' Die Texthardcopy wird mit CALL &A000
10110 ' und die Grafikhardcopy mit CALL &A080 aufgerufe
n.
10120 '
10150 MEMORY &9FFF
10190 ' Es folgt die Laderoutine für die Texthardcopy
10200 summe=0
10210 FOR n%=&A000 TO &A071
10220     READ byte$
10230     byte=VAL("&"+byte$)
10240     POKE n%,byte
10250     summe=summe+byte
10260 NEXT n%
10270 IF summe<>&39EC THEN PRINT"  Texthardcopy nicht k
orrekt geladen !":END
10280 DATA CD,78,BB,E5,CD,69,BB,E5,D5,ED,73,70,A0,CD,17
,BC,C5,C5,D1
```

```
10290 DATA 21,00,00,CD,66,BB,C1,04,48,26,01,2E,01,CD,65
,A0,C5,E5,CD,75,BB
10300 DATA CD,60,BB,38,02,3E,20,CD,45,A0,E1,C1,24,10,EC
,CD,65,A0,41,26,01
10310 DATA 3E,19,BD,28,0B,2C,18,DE,57,CD,1B,BB,FE,FC,20
,0E,ED,7B,70,A0
10320 DATA D1,E1,CD,66,BB,E1,CD,75,BB,C9,CD,2E,BD,38,E6
,7A,CD,2B,BD,C9
10330 DATA 3E,0D,CD,45,A0,3E,0A,CD,45,A0,C9,00,00
10340 '
10390 ' Es folgt die Laderoutine für die Grafikhardcopy
10400 summe=0
10410 FOR n%=&A080 TO &A161
10420     READ byte$
10430     byte=VAL("&"+byte$)
10440     POKE n%,byte
10450     summe=summe+byte
10460 NEXT n%
10470 IF summe<>&76EF THEN PRINT"  Grafikhardcopy nicht
 korrekt geladen!":END
10480 DATA CD,C6,BB,E5,D5,CD,CC,BB,E5,D5,CD,D5,BB,E5,D5
,CD,D8,BB,E5,D5
10490 DATA CD,E1,BB,47,CD,E7,BB,4F,C5,ED,73,60,A1,CD,BA
,BB,3E,1B
10500 DATA CD,13,A1,3E,31,CD,13,A1,21,8F,01,CD,47,A1,3E
,1B,CD,13,A1
10510 DATA 3E,4C,CD,13,A1,3E,00,CD,13,A1,3E,03,CD,13,A1
,06,80,3E,00
10520 DATA CD,13,A1,10,F9,11,00,00,E5,06,07,0E,00,E5,CD
,52,A1,CB,11
10530 DATA E1,2B,2B,10,F5,79,E1,E5,01,08,00,ED,42,30,02
,E6,78
10540 DATA CD,13,A1,13,21,81,02,ED,52,E1,20,D8,B7,01,0E
,00
10550 DATA ED,42,30,AD,3E,1B,CD,13,A1,3E,32,CD,13,A1,CD
,47,A1,18,08
10560 DATA 4F,CD,1B,BB,FE,FC,20,22,ED,7B,60,A1,C1,79,CD
,E4,BB
10570 DATA 78,CD,DE,BB,D1,E1,CD,D2,BB,D1,E1,CD,CF,BB,D1
,E1,CD,C9,BB
10580 DATA D1,E1,CD,C0,BB,C9,CD,2E,BD,38,D2,79,CD,2B,BD
,C9
10590 DATA 3E,0D,CD,13,A1,3E,0A,CD,13,A1,C9
10600 DATA C5,D5,E5,CD,F0,BB,B7,28,01,37,E1,D1,C1,C9,00
,00
```

Abb. 3.25: BASIC-Lader (Fortsetzung)

Ist der Lader einmal aufgerufen worden, so sind die Programme bis zum nächsten RESET oder bis zu einer Veränderung durch POKEs ab &A000 bis &A161 abgelegt und können durch CALL &A000 für die Text-Hardcopy und CALL &A080 für die Grafik-Hardcopy aufgerufen werden. Anstatt das Programm immer zuerst zu laden, kann man den BASIC-Lader auch mit MERGE zu einem Programm dazuladen. Allerdings sollte dann am Ende des Laders am besten ein RETURN stehen, damit dieser als Unterprogramm aufgerufen werden kann.

Eine andere Möglichkeit wäre, den Loader einmal mit RUN zu starten und dann die Maschinenprogramme mit

```
SAVE "HARDCOPY", B,&A000,&162
```

zu speichern. Wenn eine Hardcopy benötigt wird, so ist erst der Speicherbereich mit MEMORY &9FFF zu begrenzen und dann die Datei mit LOAD

"HARDCOPY" einzuladen. Die Programme können dann in der angegebenen Weise aufgerufen werden.

Zum Schluß beachten Sie noch folgendes:

- Die Text-Hardcopy von MODE 0- und Mode 1-Texten ist auf dem Drukker nur 20 bzw. 40 Zeichen breit. Zum Ausdruck solcher Texte sollte man besser die Grafik-Hardcopyroutine verwenden.
- Der Aufruf der Programme sollte am besten immer nur während eines laufenden Programms erfolgen, damit weder Ready-Meldungen noch der Programmaufruf mit CALL auf dem Drucker erscheinen. Notfalls hilft eine kurze Programmschleife.
- Sollte ein Abbruch nötig sein, so werden die Programme bereits durch den einmaligen Druck der ESC-Taste abgebrochen. Ein Stop und anschließender Start durch Druck einer anderen Taste ist nicht möglich. Der Abbruch ist endgültig.

Bedienungsanleitung für die Hardcopy-Programme

Die Hardcopy-Programme sind ausgelegt für den CPC mit Drucker NLQ 401, EPSON FX/RX und Kompatible (z. B. BMC BX-100). Sie werden mit dem CALL-Befehl aufgerufen; eventuell definierte Fenster, sowie Text- und Grafikcursorposition bleiben ebenso wie die ORIGIN-Definitionen erhalten.

Der Abbruch der Programme kann jederzeit durch Drücken der ESC-Taste erfolgen. Das Programm kehrt dann zur aufrufenden Routine zurück. Es werden nur ASCII-Zeichen zwischen 0 und 127 ausgegeben; die Zeichen zwischen 0 und 31 sind allerdings Steuerzeichen für die jeweiligen Drucker. Die Ausgabe von den Grafikzeichen des CPC kann nur durch Umbau des Centronics-Ports oder die Grafik-Hardcopy erfolgen. Die Text-Hardcopy wird mit

```
CALL &A000
```

und die Grafik-Hardcopy mit

```
CALL &A080
```

aufgerufen.

Kapitel 4

Dateiverwaltung

Eines der Haupteinsatzgebiete für Computer ist heute die Datenverwaltung. Jeder hat etwas mit seinem Computer zu verwalten: der Firmenchef seine Buchhaltung oder Fakturierung und der private Anwender seine Dias, Tonbänder, Schallplatten oder Videokassetten. Aus diesem Grunde wollen wir es auch nicht versäumen, Ihnen hier ein Grundkonzept für eine Dateiverwaltung anzubieten, bei dem natürlich die Fenstertechnik und andere Eigenschaften des CPC berücksichtigt werden.

Bevor wir uns mit dem Programm zur Dateiverwaltung beschäftigen, wollen wir noch ein paar Begriffe erläutern:

Datensatzelement, Item, Datenfeld

Das ist die kleinste Einheit einer Datei, wenn man von den einzelnen Zeichen absieht. Ein Datensatzelement ist z. B. ein Name, ein Vorname, eine Telefonnummer, aber auch ein Interpret, ein Song, eine Fundstelle auf dem Tonband, ein Spielfilmtitel, eine Videokassetten-Nummer oder. . .

Datensatz, Record

Ein Datensatz besteht aus mehreren Datensatzelementen, die hintereinandergereiht werden. Meistens sind sie durch ein Trennzeichen (z. B. CHR$(13)) getrennt.

Datei, File

Eine Datei - hier besser Datendatei im Gegensatz zur Programmdatei - ist eine Zusammenfassung gleichartiger Datensätze. D. h. allen Datensätzen einer Datei liegen die gleichen Datensatzelemente in der gleichen Reihenfolge (und meist in der gleichen Länge) zugrunde.

Sequentielle Datei

Wir verwenden für unsere Dateiverwaltung nur sequentielle Dateien, d. h. alle Datensatzelemente werden hintereinander auf ein externes Speicher-

medium geschrieben; die Zuordnung zu einzelnen Datensätzen oder zu einer Datei erfolgt durch die Reihenfolge beim Abspeichern bzw. Einlesen.

Direktzugriffsdatei

Bei Direktzugriffsdateien hat man die Möglichkeit, auf einzelne Datensätze gezielt zuzugreifen. Dies ist bei sequentiellen Dateien nicht möglich. Diese können nur ganz eingelesen bzw. gespeichert werden. Direktzugriffsdateien werden im folgenden nicht behandelt.

ALLGEMEINE VORGEHENSWEISE

Wie wir bereits in Kapitel 1 beim Selektieren erwähnt haben, kann man unter Zuhilfenahme von sogenannten dateibeschreibenden Variablen eine Dateiverwaltung relativ flexibel aufbauen. Hierbei sind dann zwei logische Variablenebenen zu unterscheiden: eine, die die Daten, die verwaltet werden sollen, beschreibt und eine andere Variablenebene, die sie beinhaltet. Am besten machen wir dies gleich am Beispiel der Dateiverwaltung deutlich.

Abb. 4.1: Vorspann der Dateiverwaltung (Teilprogramm)

```
100 REM ------------------------------
110 REM ---       Vorspann           ---
120 REM ------------------------------
130 :
140 MODE 1
150 :
160 REM ---  Daten der Datei        ---
170 :
180 ANZAHL = 7
190 SAETZE = 100
200 LETZTER = 0
210 DATEI$="Kunden"
220 :
230 DIM NAME$     (ANZAHL)
240 DIM INHALT$   (ANZAHL,SAETZE)
250 DIM LAENGE    (ANZAHL)
260 DIM ART$      (ANZAHL)
270 :
280 DATA Name,a,18
290 DATA Vorname,a,13
300 DATA Strasse,a,23
310 DATA PLZ/Ort,a,25
320 DATA Telefon,a,12
330 DATA Konto,f,5.2
340 DATA Umsatz,f,6.2
350 :
360 FOR I=1 TO ANZAHL
370    READ NAME$(I)
380    IF LEN(NAME$(I)) > MAXIMUM THEN MAXIMUM = LEN(NAME$(I))
390    READ ART$(I)
400    READ LAENGE(I)
410 NEXT
420 :
500 REM ---  Bildschirmaufbau      ---
510 :
520 CLS
```

```
530 :
540 GOSUB 12040
550 :
560 CLS #4
570 PRINT #4,"1 - ";DATEI$;" erfassen"
580 PRINT #4,"2 - ";DATEI$;" anzeigen"
590 PRINT #4,"3 - ";DATEI$;" aendern"
600 PRINT #4,"4 - ";DATEI$;" loeschen"
610 PRINT #4,"5 - ";DATEI$;" listen"
620 PRINT #4,"6 - ";DATEI$;" selektieren"
630 PRINT #4,"7 - ";DATEI$;" sortieren"
640 PRINT #4,"8 - ";DATEI$;" speichern"
650 PRINT #4,"9 - ";DATEI$;" laden"
800 LOCATE #3,10,2
810 PRINT #3," Bitte waehlen Sie:
820 :
830 :
840 A$=INKEY$
850 A=VAL(A$)
860 IF A < 1 OR A > 9 THEN GOTO 840
870 ON A GOTO 1000,2000,3000,4000,5000,6000,7000,8000,9000
880 :
```

Abb. 4.1: Vorspann der Dateiverwaltung (Fortsetzung)

Nehmen wir an, Sie wollen für einen kleinen Betrieb eine einfache Adreßdatei führen, in der außer dem Namen, Vornamen, Straße, Postleitzahl und Ort auch Telefon, Kontostand und Umsatz gespeichert sind. Ein jeder Datensatz in dieser Dateiverwaltung hätte also sieben verschiedene Einträge, was wir in Zeile 180 in der Variablen ANZAHL festhalten. Außerdem brauchen wir noch die Anzahl der zu verwaltenden Datensätze, die in unserem Beispielprogramm mit 100 vorgegeben ist. Die maximale Anzahl von Datensätzen, die speicherbar sind, ist vom restlichen Speicherplatz abhängig. Die Dokumentation zu diesem Programm wurde sehr ausführlich gestaltet, um Ihnen die Materie einfach darbieten zu können. Sie erhöhen natürlich die Anzahl der möglichen Datensätze erheblich, wenn Sie alle Bemerkungen und überflüssigen Leerzeichen weglassen. Zu einigen anderen Möglichkeiten, wie Sie Rechenzeit und Speicherplatz sparen können, kommen wir später.

Da wir z. B. bei 50 eingegebenen Datensätzen nicht alle 100 vorgegebenen Sätze durchsuchen wollen, setzen wir noch eine Variable LETZTER zunächst auf Null, die später angibt, der wievielte Datensatz innerhalb der Datei der letzte ist. Weiterhin geben wir der Datei noch einen Namen, der einerseits der Dokumentation dient, andererseits aber später zur Ausgabe auch noch gebraucht wird.

Daß wir die Daten alle im Hauptspeicher halten, bringt zwar Speicherplatzprobleme, aber es ergibt sich dadurch eine wesentlich leichtere Handhabung mit den Peripheriegeräten. Die Daten können sequentiell gespeichert werden, und die Dateiverwaltung ist somit sowohl mit Kassettenrecorder als auch mit Diskettenlaufwerk verwendbar. Sollten Sie mehr Daten haben, so teilen Sie

diese einfach nach Postleitzahlbereichen oder anderen Kriterien auf und erstellen eine Komplettliste mit dem in Kapitel 1 angegebenen Mischprogramm.

Übrigens können Sie die dateibeschreibenden Variablen aus den DATA-Zeilen auch in eine sequentielle Datei schreiben und zu Beginn der Dateiverwaltung einlesen. Sofern Sie die Dateiverwaltung - wie wir später sehen werden - für das Ausdrucken noch etwas in eine allgemeinere Form umschreiben, können Sie sie für die unterschiedlichsten Daten heranziehen.

Da die Daten im Hauptspeicher stehen, der leider nicht gegen Stromausfälle geschützt ist, sollten Sie bei längeren Sitzungen häufiger Ihre Daten speichern. Besonders mit Disketten geht das Speichern recht schnell, und sofern Sie einen Kassettenrecorder besitzen, sollten Sie die Arbeitszeiten (Speichern auf der einen Seite/Änderungen erneut durchführen bzw. Neueingabe andererseits) sorgfältig gegeneinander abwägen.

Doch nun wollen wir das Programm weiter betrachten. Ab Zeile 230 werden vier verschiedene Felder definiert, die später folgende Daten aufnehmen sollen:

NAME$( ) : Bezeichnungen der einzelnen Datensatzelemente

INHALT$( ) : Eigentliche Daten, wobei der erste Index die Nummer des Elementes und der zweite Index die Nummer des Datensatzes angibt

LAENGE( ) : Die Länge in Zeichen oder Ziffern der einzelnen Datensatzelemente

ART$( ) : Art des Datensatzelementes, wobei Sie folgende Zuordnung treffen können:

- a - Alphanumerische Zeichen
- i - Integerzahl
- f - Fließkommazahl
- d - Kalenderdatum

Die Längenangabe ist natürlich abhängig von der Art, wobei für alphanumerische Zeichen die Anzahl der Zeichen, für Integerzahlen die Anzahl der Ziffern (Achtung, Vorzeichen beachten!), für Dezimalzahlen die Anzahl der Vor- bzw. Nachkommastellen (Achtung, auch hier auf das Vorzeichen und den Dezimalpunkt achten!) und für Kalenderdaten acht Zeichen in der Form TT.MM.JJ. vorgesehen sind.

Ab Zeile 280 definieren wir unsere Beispieldaten, wobei für jedes Datenelement die Bezeichnung, die Art und die Länge in der DATA-Zeile angegeben ist.

Die den Werten zugeordneten Felder werden in der Schleife von Zeile 360 bis Zeile 410 besetzt, wobei wir noch die Bezeichnung mit der maximalen Anzahl von Buchstaben herausfinden müssen, damit wir später unseren Bildschirmaufbau auch variabel gestalten können.

Bildschirmaufbau

Da wir später von dem bisher gewählten Modus 1 auf Modus 2 umschalten werden, müssen wir die Definition der Fenster wiederholen. Deshalb wurde dieser Teil als Unterprogramm ausgelegt (Abb. 4.2).

```
12000 REM ------------------------------
12010 REM --- Anzeige in Fenster #1 ---
12020 REM ------------------------------
12030 :
12040    WINDOW #1,1,40,1,3
12050    CLS #1
12060    PRINT #1,CHR$(150);
12070 :
12080    FOR I=1 TO 38
12090       PRINT #1,CHR$(154);
12100    NEXT
12110 :
12120    PRINT #1,CHR$(156);
12130 :
12140    PRINT #1,CHR$(149);"      Dateiverwaltung fuer  ";
12150    PEN #1,0
12160    PAPER #1,1
12170    PRINT #1,DATEI$;
12180    PEN #1,1
12190    PAPER #1,0
12200    PRINT #1,"      ";CHR$(149);
12210    PRINT #1,CHR$(147);
12220 :
12230 :
12240    FOR I=1 TO 38
12250       PRINT #1,CHR$(154);
12260    NEXT
12270 :
12280    PRINT #1,CHR$(153);
12290 :
12300 REM ------------------------------
12310 REM --- Weitere Fenster defin.---
12320 REM ------------------------------
12330 :
12340    WINDOW #2,1,MAXIMUM+2,13-INT(ANZAHL/2)+1,13+INT(ANZAHL/2)+1
12350    PEN #2,0
12360 :
12370    WINDOW #3,1,40,23,25
12380    PEN #3,2
12390    PAPER #3,3
12400    CLS #3
12410 :
12420    WINDOW #4,9,32,9,20
12430    CLS #4
12440 :
12450   WINDOW #5,MAXIMUM+3,40,13-INT(ANZAHL/2)+1,13+INT(ANZAHL/2)+1
12460 PEN #5,3
12470 :
12480 RETURN
```

Abb. 4.2: Fensterdefinitionen (Teilprogramm)

Mit den bereits beim Einschalten eingestellten Farben werden im ganzen sieben neue Fenster definiert, womit wir die Kapazität an Fenstern des CPC voll ausschöpfen. Die einzelnen Fenster haben folgende Bedeutung:

Fenster #1 : Kopf

Fenster #2 : Anzeige der Datenelementbezeichnung am linken Bildschirmrand in der Höhe abhängig von der Anzahl der Datensatzelemente

Fenster #3 : Fenster zur Anzeige der aktuellen Tätigkeit, bzw. für Fehlermeldungen und Eingaben

Fenster #4 : Menüfenster

Fenster #5 : Datenfenster

Fenster #6 : Überschrift für Listen (80 Zeichen breit)

Fenster #7 : Datenfenster bei Ausgabe von Listen (80 Zeichen breit)

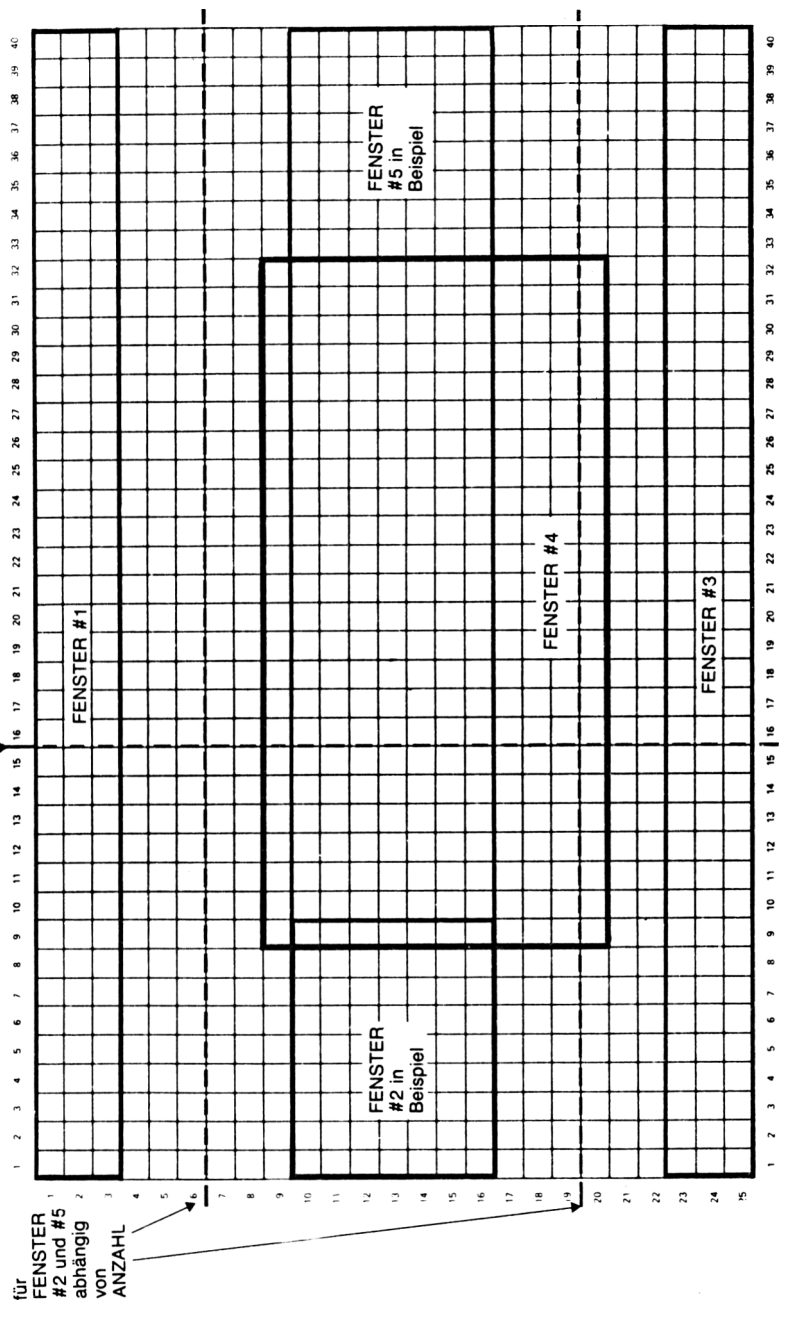
Folgende Farbzuordnung für die Fenster ist vorgesehen:

Fenster#	PAPER	PEN
0	0	1
1	0	1
2	1	0
3	3	2
4	0	1
5	2	3
6	1	0
7	0	1

Wie Sie daran sehen, sind die allgemeinen Angaben in den Originalfarben (gelb/blau) gehalten, wobei die Datenelementbezeichnungen invers erscheinen und die sich jeweils ändernden Teile (Daten/Meldungen) in den beiden anderen Farben. Die Farben bei den Daten und bei den Meldungen sind invers zueinander.

Als erstes wird in Zeile 12040 das Fenster #1 definiert, wobei die Größe aus Abb. 4.3 deutlich wird. Beachten Sie bitte, daß die Grenze zwischen Fenster #2 und Fenster #5 abhängig von MAXIMUM ist und die obere und untere Begrenzung abhängig von ANZAHL.

Abb. 4.3: Aufteilung des Bildschirms in Fenster #1 bis Fenster #5


für FENSTER #2 und #5 abhängig von MAXIMUM
für FENSTER #2 und #5 abhängig von ANZAHL
FENSTER #1
FENSTER #2 in Beispiel
FENSTER #5 in Beispiel
FENSTER #4
FENSTER #3

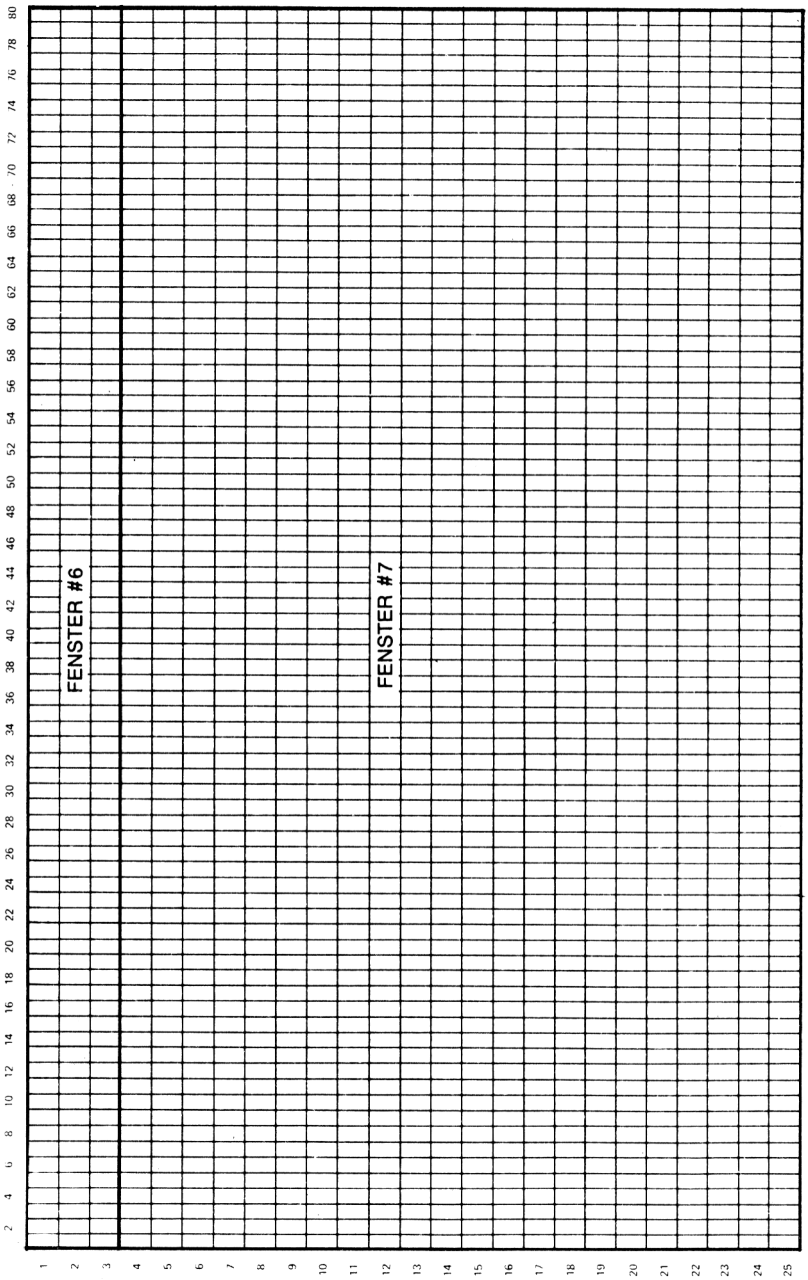


Abb. 4.4: Aufteilung des Bildschirms für Fenster #6 und Fenster #7

In Fenster #1 wird ein Rahmen gezogen, wie wir ihn schon von dem Rahmen der Karten her kennen. In die Mitte wird der Text „Dateiverwaltung fuer" eingetragen und, bedingt durch Zeile 12170 und die Farbumschaltungen vorher und nachher, der Dateiname invers ausgegeben.

Ab Zeile 12340 werden die Fenster #2 bis #5 definiert (siehe Abb. 4.3) und entsprechend die Farben für die Fenster gesetzt, sowie Fenster #3 und Fenster #4 gelöscht, um den entsprechenden Farbhintergrund zu erhalten.

Im Hauptprogramm wird zunächst das Menü in Fenster #4 ausgegeben und anschließend in Fenster #3 die Aufforderung „Bitte waehlen Sie:". In den Zeilen ab 840 erfolgt die Abfrage der Eingabe, wobei in Zeile 870 in die entsprechenden Programmteile verzweigt wird, die nicht als Unterprogramm, sondern nur als normale Programmverzweigung organisiert sind. Die Nummer des Menüpunktes entspricht dabei der Tausenderstelle der Zeilennummer.

ERFASSEN DER DATEN

Wir gehen hier das Menü in seiner Reihenfolge durch, wobei die am häufigsten vorkommenden Punkte im Menü oben angesetzt sind. Die Reihenfolge der Eingabe für die einzelnen Teilschritte kann von Ihnen beliebig vorgenommen werden.

```
1000 REM ------------------------------
1010 REM ---       Daten erfassen     ---
1020 REM ------------------------------
1030 :
1040 CLS #4
1050 CLS #3
1060 :
1070 GOSUB 10000
1080 :
1090 LETZTER = LETZTER+1
1100 :
1110 PAPER #5,2
1120 CLS #5
1130 :
1140 FOR I=1 TO ANZAHL
1150    LOCATE #5,1,I
1155    CLS #3
1160    LOCATE #3,10,2
1170    PRINT #3,"Erfassen ";Name$(i);"
1180    LINE INPUT #5,INHALT$(I,LETZTER)
1190 NEXT
1200 :
1210 CLS #3
1220 LOCATE #3,10,2
1230 PRINT #3,"Naechster (n) / Menue (m)
1240 :
1250 A$=INKEY$
1260 IF A$="n" THEN GOTO 1090
1270 IF A$="m" THEN GOTO 9500
1280 GOTO 1250
1290 :
```

Abb. 4.5: Daten erfassen (Teilprogramm)

Bevor wir Daten erfassen können, müssen noch einige Vorarbeiten geleistet werden. Zunächst müssen das Menü und die Anweisung in Fenster #3 gelöscht werden, was in den Zeilen 1040 und 1050 geschieht. Dann muß das Unterprogramm aus Abb. 4.6 aufgerufen werden, um die Bezeichnungen der einzelnen Datensatzelemente am Bildschirm auszugeben.

```
10000 REM ------------------------------
10010 REM ------------------------------
10020 REM ---                          ---
10030 REM ---     Unterprogramme       ---
10040 REM ---                          ---
10050 REM ------------------------------
10060 REM ------------------------------
10070 :
10080 :
11000 REM ------------------------------
11010 REM --- Anzeige NAME$() in #2 ---
11020 REM ------------------------------
11030    PEN #2,0
11040    PAPER #2,1
11050    CLS #2
11060 :
11070    FOR I=1 TO ANZAHL
11080      PRINT #2," ";NAME$(I)
11090      LOCATE #2,MAXIMUM+2,I
11100      PRINT #2,":";
11110    NEXT
11120 :
11130 RETURN
11140 :
```

Abb. 4.6: Anzeige NAME$ in Fenster #2 (Unterprogramm)

Zunächst werden die Farben gegenüber dem normalen Bildschirm umgedreht, und mit dem Löschbefehl in Zeile 11050 wird ein hellblauer Hintergrund ausgegeben. Die Schleife ab Zeile 11070 druckt für alle Datensatzelemente den Namen und an letzter Position im Fenster einen Doppelpunkt.

Im Hautprogramm wird jetzt der Hintergrund von Fenster #5 auf „helles Blaugrün" umgeschaltet und in der Schleife ab Zeile 1140 jedes einzelne Datensatzelement erfaßt, wobei zusätzlich in Fenster #3 noch ausgegeben wird, welches Element gerade erfaßt wird. Durch den LINE INPUT-Befehl wird ein Zeilenvorschub bei der Eingabe unterdrückt.

Ab Zeile 1210 wird im Kommandofenster eine Meldung ausgegeben und anschließend abgefragt, ob noch ein Datensatz erfaßt werden soll oder ins Menü zurückgesprungen werden soll.

Achtung: Nach dem Unterprogramm wird zunächst unser Zeiger LETZTER um 1 erhöht, wodurch wir Platz für einen neuen Datensatz schaffen. Wer meint, daß es mit dem vorgegebenen Platz knapp wird, kann in Zeile 1095 noch eine Plausibilitätsprüfung (LETZTER größer als SAETZE?) einfügen.

ANZEIGEN DER DATEN

Beim Anzeigen der Daten müssen wir natürlich zunächst eingeben, welchen Datensatz wir anzeigen wollen. Dabei soll uns die in Kapitel 1 beschriebene Präfixsuche bei der Eingabe unterstützen, so daß wir nicht immer alle Namen ausschreiben müssen.

In Abb. 4.7 betrachten wir zunächst das Listing.

```
2000 REM ------------------------------
2010 REM ---       Daten anzeigen     ---
2020 REM ------------------------------
2030 :
2040 CLS #4
2050 CLS #3
2060 :
2070 GOSUB 10000
2080 :
2090 PAPER #5,2
2100 CLS #5
2110 :
2120 CLS #3
2130 LOCATE #3,10,2
2140 PRINT #3,"Bitte ";NAME$(1);" eingeben"
2150 :
2160 LINE INPUT #5,SUCHNAME$
2170 GEFUNDEN=0
2180 :
2190 STERNPOS=INSTR(1,SUCHNAME$,"*")
2200 IF STERNPOS = 0 THEN STERNPOS=LAENGE(1)
2210 :
2220 FOR I=1 TO LETZTER
2230     IF LEFT$(INHALT$(1,I),STERNPOS-1) = LEFT$(SUCHN
AME$,STERNPOS-1) THEN GOTO 2500
2240 NEXT
2250 :
2260 CLS #3
2270 LOCATE #3,2,2
2280 PRINT #3,"Keine Daten mehr vorhanden/Taste
2290 :
2300 A$=INKEY$
2310 IF A$="" THEN 2300
2320 GOTO 9500
2330 :
2500 REM ---       AUSGABE            ---
2510 :
2520 GEFUNDEN=1
2530 CLS #5
2540 :
2550 FOR J=1 TO ANZAHL
2560     LOCATE #5,1,J
2570     PRINT #5,INHALT$(J,I)
2580 NEXT
2590 :
2600 CLS #3
2610 LOCATE #3,10,2
2620 PRINT #3,"Weiter (w); Abbruch (a)
2630 :
2640 A$=INKEY$
2650 IF A$="w" THEN GOTO 2240
2660 IF A$="a" THEN GOTO 9500
2670 GOTO 2640
2680 :
```

Abb. 4.7: Daten anzeigen (Teilprogramm)

Wie bereits im letzten Kapitel gezeigt, müssen hier erst die Fenster #3 und #4 gelöscht und das Unterprogramm ab Zeile 10000 aufgerufen werden; auch das Fenster #5 ist in der Farbe umzuschalten. Dann wird der Anwender ab Zeile 2120 um eine Eingabe gebeten, die im ersten Feld des Datensatzes stattfindet.

Der Cursor braucht nicht extra gesetzt zu werden, da durch das Löschen des Fensters #5 in Zeile 2100 die linke obere Ecke bereits erreicht ist. Dann wird die Variable GEFUNDEN auf Null zurückgesetzt, die uns später angeben soll, ob der gewünschte Datensatz (oder ein ähnlicher) gefunden wurde.

Wie bereits in Kapitel 1 beschrieben, wird in der Variablen STERNPOS die Position eines eventuellen Sternchens festgehalten. Mit einem einfachen Programmabbruch ist es nicht getan, wenn kein Stern in SUCHNAME$ vorkommt. Wir setzen die Variable auf die Länge des Datensatzelementes, womit alle Buchstaben untersucht werden. In der Schleife ab Zeile 2220 wird dann die Suche gestartet, wobei zur Programmzeile 2500 verzweigt wird, wenn eine Übereinstimmung erzielt ist.

Wenn diese Schleife durchlaufen wurde, wird eine Meldung an den Anwender herausgegeben und auf Tastendruck gewartet. Ist auch dieser erfolgt, so wird zur Programmzeile 9500 übergegangen, wo die Fenster für eine Menüausgabe zurückgesetzt werden (Abb. 4.8).

```
9500 REM ------------------------------
9510 REM ---  Vorlauf fuer Menue    ---
9520 REM ------------------------------
9530 :
9540 CLS #3
9550 PAPER #2,0
9560 CLS #2
9570 PAPER #5,0
9580 CLS #5
9590 GOTO 560
9600 :
```

Abb. 4.8: Vorlauf für das Menü (Teilprogramm)

Fenster #3 wird gelöscht und die Papierfarbe von Fenster #2 auf blau zurückgesetzt, um mit dem Löschen wieder einen blauen Hintergrund zu bekommen. Das gleiche geschieht mit Fenster #5, wonach zur Zeile 560 übergegangen wird (Ausgabe des Menüs).

Ab Zeile 2500 im Programmteil „Daten anzeigen" (Abb. 4.7) wird zunächst der Merker GEFUNDEN auf 1 gesetzt und anschließend der Bildschirm für die Datenausgabe gelöscht. Dann werden in einer Schleife alle Inhalte des ge-

fundenen Datensatzes (mit der Nummer I) ausgegeben. Ab Zeile 2600 wird noch abgefragt, ob weitergesucht werden soll oder nicht. Entsprechend der Eingabe (w oder a) wird verzweigt.

ÄNDERN DER DATEN

Auch zum Ändern der Daten muß zunächst nach dem Datensatz gesucht werden, der geändert werden soll. Deshalb sind im Listingausdruck in Abb. 4.9 die ersten Zeilen analog den bereits im letzten Abschnitt aufgeführten. Wir haben das hier der Übersichtlichkeit halber noch einmal dokumentiert. Sie können jedoch diesen Teil als Unterprogramm schreiben, das z. B. den Bereich von Zeile 3040 bis Zeile 3600 enthält, wobei der Ausgabetext in Zeile 3260 dann als Variable gestaltet werden sollte.

Abb. 4.9: Daten ändern (Teilprogramm)

```
3000 REM ------------------------------
3010 REM ---       Daten aendern      ---
3020 REM ------------------------------
3030 :
3040 CLS #4
3050 CLS #3
3060 :
3070 GOSUB 10000
3080 :
3090 PAPER #5,2
3100 CLS #5
3110 :
3120 CLS #3
3130 LOCATE #3,10,2
3140 PRINT #3,"Bitte ";NAME$(l);" eingeben"
3150 :
3160 LINE INPUT #5,SUCHNAME$
3170 STERNPOS=INSTR(1,SUCHNAME$,"*")
3180 IF STERNPOS = 0 THEN STERNPOS=LAENGE(1)
3190 :
3200 FOR I=1 TO LETZTER
3210    IF LEFT$(INHALT$(1,I),STERNPOS-1) = LEFT$(SUCHN
AME$,STERNPOS-1) THEN GOTO 3500
3220 NEXT
3230 :
3240 CLS #3
3250 LOCATE #3,2,2
3260 PRINT #3,"Datensatz nicht gefunden/Taste
3270 :
3280 A$=INKEY$
3290 IF A$="" THEN 3280
3300 GOTO 9500
3310 :
3500 REM ---       AUSGABE            ---
3510 :
3520 CLS #5
3530 :
3540 FOR J=1 TO ANZAHL
3550    LOCATE #5,1,J
3560    PRINT #5,INHALT$(J,I)
3570 NEXT
3580 :
3590 CLS #3
3600 LOCATE #3,10,2
3610 PRINT #3,"Diesen Satz aendern? (j/n)
3620 :
```

```
3630 A$=INKEY$
3640 IF A$="n" THEN GOTO 3220
3650 IF A$="j" THEN GOTO 3750
3660 GOTO 3630
3670 :
3750 REM ---  Aenderung durchfuehren---
3760 :
3770 CLS #3
3780 LOCATE #3,1,2
3790 PRINT #3,"Welches Element aendern (1-";ANZAHL;") /
 0-Ende"
3800 :
3810 A$=INKEY$
3820 IF A$="" THEN GOTO 3810
3830 A=VAL(A$)
3840 IF A=0 THEN GOTO 9500
3850 IF A < 1 OR A > ANZAHL THEN 3810
3860 :
3870 LOCATE #5,1,A
3880 PRINT #5,CHR$(18)
3890 LOCATE #5,1,A
3900 LINE INPUT #5,INHALT$(A,I)
3910 CLS #5
3920 :
3930 FOR J=1 TO ANZAHL
3940    LOCATE #5,1,J
3950    PRINT #5,INHALT$(J,I)
3960 NEXT
3970 :
3980 GOTO 3750
3990 :
```

Abb. 4.9: Daten ändern (Fortsetzung)

Bei der Änderung der Daten wird zunächst gefragt, welches Element geändert werden soll, wobei hier über die Nummer des Datensatzelementes zugegriffen wird. Sofern Sie mehr Datensatzelemente in Ihrem Datensatz haben, empfiehlt es sich, die Nummern voranzustellen, da es nicht sehr sinnvoll ist, jedesmal die Elementbezeichnung einzutippen.

Bei Eingabe von 0 erfolgt der übliche Abbruch zu dem Programmteil, der die Fenster zur Menüausgabe aufbereitet. In Zeile 3850 befindet sich noch eine Plausibilitätsprüfung, und ab Zeile 3870 wird das eigentliche Ändern durchgeführt.

In A ist die Nummer des zu ändernden Elementes vorgegeben, so daß durch den LOCATE-Befehl genau diese Zeile anvisiert wird. In Kapitel 2 haben wir bereits die Steuerzeichen erwähnt, so daß Ihnen der Code 18 keine Schwierigkeiten bereitet (löschen des Restes der Zeile bis zum Fensterende). Nachher muß natürlich erneut der Cursor wieder an den Anfang des Fensters in der betreffenden Zeile positioniert werden, und nun können die neuen Daten erfaßt werden.

Das Löschen haben wir deshalb vorgenommen, um Mißverständnisse zu vermeiden. Wie Sie es vom Programmeditor her kennen, arbeitet der CPC nicht bildschirmorientiert. Wurde die Zeile nicht gelöscht, kann man z. B. beim

Ändern eines einzelnen Buchstabens der Meinung sein, daß auch der Rest der Zeile wieder als Element in den Datensatz übernommen wird, was jedoch nicht der Fall ist.

Zur Kontrolle wird anschließend das Fenster #5 gelöscht, und die wirklich gespeicherten Daten werden angezeigt. Sofern Sie mehrere Elemente des Datensatzes ändern wollen, erhalten Sie dazu die Möglichkeit, indem in Zeile 3980 erneut ein Sprung zur Änderungsroutine durchgeführt wird.

LÖSCHEN DER DATEN

Auch beim Löschen der Daten muß zunächst wieder der Datensatz herausgesucht und angezeigt werden (als Kontrolle, ob der richtige Datensatz gelöscht wird). Wer sich bereits das Unterprogramm für den ersten Bereich der Zeilen geschrieben hat, kann dies aufrufen.

Abb. 4.10: Daten löschen (Teilprogramm)

```
4000 REM ------------------------------
4010 REM ---       Daten loeschen     ---
4020 REM ------------------------------
4030 :
4040 CLS #4
4050 CLS #3
4060 :
4070 GOSUB 10000
4080 :
4090 PAPER #5,2
4100 CLS #5
4110 :
4120 CLS #3
4130 LOCATE #3,10,2
4140 PRINT #3,"Bitte ";NAME$(1);" eingeben"
4150 :
4160 LINE INPUT #5,SUCHNAME$
4170 STERNPOS=INSTR(1,SUCHNAME$,"*")
4180 IF STERNPOS = 0 THEN STERNPOS=LAENGE(1)
4190 :
4200 FOR I=1 TO LETZTER
4210    IF LEFT$(INHALT$(1,I),STERNPOS-1) = LEFT$(SUCHN
AME$,STERNPOS-1) THEN GOTO 4520
4220 NEXT
4230 :
4240 CLS #3
4250 LOCATE #3,2,2
4260 PRINT #3,"(Weiteren) Datensatz nicht gefunden
                          -> Taste
4270 :
4280 A$=INKEY$
4290 IF A$="" THEN 3280
4300 :
4310 GOTO 9500
4320 :
4500 REM ---       AUSGABE            ---
4510 :
4520 CLS #5
4530 :
4540 FOR J=1 TO ANZAHL
4550    LOCATE #5,1,J
4560    PRINT #5,INHALT$(J,I)
```

```
4570 NEXT
4580 :
4590 CLS #3
4600 LOCATE #3,10,2
4610 PRINT #3,"Diesen Satz loeschen? (j/n)
4620 A$=INKEY$
4630 IF A$="n" THEN GOTO 4220
4640 IF A$="j" THEN GOTO 4750
4650 GOTO 4620
4660 :
4750 REM --- Loeschung durchfuehren ---
4760 :
4770 FOR J=1 TO ANZAHL
4780    INHALT$(J,I)=INHALT$(J,LETZTER)
4790 NEXT
4800 :
4810 LETZTER = LETZTER-1
4820 :
4830 GOTO 9500
4840 :
```

Abb. 4.10: Daten löschen (Fortsetzung)

Beim eigentlichen Löschen kann ein Element innerhalb eines Feldes nicht einfach gelöscht werden, man müßte es als gelöscht kennzeichnen. Aus Platzgründen ist dies jedoch nicht sinnvoll. Das einfachste ist das Umkopieren des letzten Datensatzes an die Stelle, wo der Datensatz gelöscht werden soll. Der Zeiger auf den letzten Datensatz ist natürlich die Variable LETZTER, und der aktuelle Datensatz hat durch den Suchvorgang die Nummer I. Dadurch vermindert sich die Länge der Datei natürlich um einen Datensatz, was in Zeile 4810 berücksichtigt wird.

DRUCKEN DER DATEN

Etwas komplizierter ist die tabellarische Ausgabe von Daten, die wir im folgenden besprechen wollen.

Abb. 4.11: Daten listen (Teilprogramm)

```
5000 REM ------------------------------
5010 REM ---       Daten ausgeben     ---
5020 REM ------------------------------
5030 :
5040 MODE 2
5050 :
5060 WINDOW #7,1,80,4,25
5070 WINDOW #6,1,80,1,3
5080 :
5090 PEN #6,0
5100 PAPER #6,1
5110 CLS #6
5120 LOCATE #6,3,2
5130 PRINT #6,"1 - Adressen ;  2 - Konten ;   3 - Telef
onliste
5140 :
```

```
5150 A$=INKEY$
5160 IF A$="1" THEN GOTO 5300
5170 IF A$="2" THEN 5300 : REM GOTO 5500
5180 IF A$="3" THEN 5300 : REM GOTO 5800
5190 GOTO 5150
5200 :
5300 CLS #6
5310 PRINT #6
5320 PRINT #6," Name              Vorname       Strasse
              PLZ/Ort"
5330 PRINT #6,"----------------------------------------
-------------------------------------";
5340 :
5350 FOR I=1 TO LETZTER
5360     NAME$=LEFT$(INHALT$(1,I)+SPACE$(LAENGE(1)),LAEN
GE(1))
5370     VORNAME$=LEFT$(INHALT$(2,I)+SPACE$(LAENGE(2)),L
AENGE(2))
5380     STRASSE$=LEFT$(INHALT$(3,I)+SPACE$(LAENGE(3)),L
AENGE(3))
5390     ORT$=LEFT$(INHALT$(4,I)+SPACE$(LAENGE(4)),LAENG
E(4))
5400     PRINT #7,NAME$;VORNAME$;STRASSE$;ORT$
5410 NEXT
5420 :
5430 A$=INKEY$
5440 IF A$="" THEN GOTO 5430
5450 :
5460 MODE 1
5470 :
5480 GOSUB 12040
5490 :
5500 REM ********** Uebung ************
5510 GOTO 9500
5520 :
5800 REM ********** Uebung ************
5810 GOTO 9500
5820 :
```

Abb. 4.11: Daten listen (Fortsetzung)

Zunächst schalten wir auf Modus 2 um, da die meisten Datensätze wohl nicht auf 40 Zeichen Breite unterzubringen sind, und auch bei 80 Zeichen pro Zeile gibt es noch einige Schwierigkeiten, so daß wir die Ausgabe in drei verschiedene Listen (Adressen; Konten; Telefonliste) gliedern. Dabei wollen wir hier die erste Liste vorstellen und haben die nächsten beiden Listen (Konten/Telefonliste) für Sie als Übung gedacht.

Zunächst wird in Fenster #6 abgefragt, welche Liste ausgegeben werden soll, wonach sich der Verteiler ab Zeile 5160 richtet.

Dann wird der Listenkopf in Fenster #6 ausgegeben und ab Zeile 5350 die Daten in Bildschirmfenster #7.

Die Ausgabe der Daten sieht komplizierter aus, als sie ist. Zunächst werden die auszugebenden Datensätze auf ihre - am Anfang - definierte Länge gebracht.

Jeder Inhalt eines Datensatzelementes wird zunächst mit einer Zeichenreihe aus Leerzeichen (SPACE$) verknüpft und dann auf die definierbare Länge

abgeschnitten. In Zeile 5400 werden alle Daten nacheinander in einer Zeile ausgegeben. Sofern ein Datensatzelement seine maximale Länge erreicht hat, steht allerdings kein Zwischenraum mehr, aber Sie können die trennenden Semikolons in Zeile 5400 durch ";" "j" ersetzen. Vergessen Sie dann auch nicht, Zeile 5320 entsprechend zu ändern, und natürlich muß die Länge der Datensatzelemente auf die Bildschirmbreite abgestimmt werden.

Sofern Sie mehr Datensätze erfaßt haben, als auf dem Bildschirm Platz haben (22), können Sie durch Drücken von ESC die Ausgabe anhalten, was bekanntermaßen zu den „normalen" Funktionen des CPC gehört. Mit einer beliebigen anderen Taste können Sie die Ausgabe weiterlaufen lassen. Da alle Daten im Hauptspeicher gehalten und nicht sequentiell von einem Datenträger eingelesen werden, dürfte es für den fortgeschrittenen Programmierer keine große Schwierigkeit bedeuten, auch ein Blättern vorwärts und rückwärts zu realisieren.

Nach Abschluß der Liste wird auf einen Tastendruck gewartet (Zeilen 5430/5440). Anschließend wird wieder auf Modus 1 geschaltet und das Unterprogramm zum Setzen der Fenster aufgerufen. Ab Zeile 5500 und 5800 ist Platz für die anderen Listen reserviert.

SPEICHERN DER DATEN

Kommen wir nun zu der Kommunikation mit den externen Datenträgern, wobei wir zuerst das Speichern berücksichtigen wollen.

```
8000 REM ------------------------------
8010 REM ---      Daten speichern     ---
8020 REM ------------------------------
8030 :
8040 CLS #3
8050 LOCATE #3,10,2
8060 PRINT #3,"Daten werden gespeichert
8070 :
8080 OPENOUT "!"+DATEI$+".dat"
8090 WRITE #9,LETZTER
8100 :
8110 FOR I=1 TO LETZTER
8120    FOR J=1 TO ANZAHL
8130       WRITE #9,INHALT$(J,I)
8140    NEXT
8150 NEXT
8160 :
8170 CLOSEOUT
8180 :
8190 GOTO 9500
8200 :
```

Abb. 4.12: Daten speichern (Teilprogramm)

Zunächst wird in Fenster #3 eine Meldung ausgegeben, daß die Daten gespeichert werden, damit der Anwender nicht versehentlich denkt, daß der Rechner abgestürzt ist. Dann wird eine Datendatei mit unserem Dateinamen und angehängtem „.DAT" geöffnet und zunächst die Anzahl der Datensätze (LETZTER) gespeichert. In zwei ineinandergeschachtelten Schleifen werden die Datensätze (alle Elemente eines Datensatzes zusammen) fortlaufend weggeschrieben. Anschließend wird nur noch die Datei geschlossen und die Fensterbehandlung vor dem Menüaufruf durchgeführt.

LADEN DER DATEN

Analog dem Speichern geschieht das Laden der Daten, wobei nur in den Zeilen 8080 und 8170 das „OUT" durch ein „IN" ausgetauscht und natürlich das WRITE in 8090 und 8130 in ein INPUT verwandelt wird. Die letzte Änderung ist der Austausch von „gespeichert" in Zeile 8060 in „geladen" in Zeile 9060, um den Unterschied auch dem Anwender kundzutun.

```
9000 REM ------------------------------
9010 REM ---      Daten laden         ---
9020 REM ------------------------------
9030 :
9040 CLS #3
9050 LOCATE #3,10,2
9060 PRINT #3,"Daten werden geladen
9070 :
9080 OPENIN "!"+DATEI$+".dat"
9090 INPUT #9,LETZTER
9100 :
9110 FOR I=1 TO LETZTER
9120    FOR J=1 TO ANZAHL
9130       INPUT #9,INHALT$(J,I)
9140    NEXT
9150 NEXT
9160 :
9170 CLOSEIN
9180 :
9190 GOTO 9500
```

Abb. 4.13: Daten laden (Teilprogramm)

TIPS FÜR DAS SORTIEREN UND SELEKTIEREN

Dem aufmerksamen Leser wird sicherlich nicht entgangen sein, daß die Menüpunkte 6-Selektieren und 7-Sortieren vorgesehen wurden. Da wir bereits in Kapitel 1 verschiedene Verfahren zum Sortieren und Selektieren aufgezeigt haben und außerdem beim Erfragen eines Datensatzes zum Anzeigen, Ändern und Löschen bereits Wild Cards zugelassen haben, dürfte Ihnen diese Programmierung keine Schwierigkeiten mehr bereiten.

Verwenden Sie ein Sortierverfahren Ihrer Wahl, wobei Minsort in den meisten Fällen ausreichend ist, da die Anzahl der Datensätze sowieso begrenzt ist. Um sich jedoch das zeitaufwendige Vertauschen aller Datensatzelemente eines Datensatzes beim Sortieren zu sparen, sollte mit Verwendung von Zeigern sortiert werden, was z. B. durch ein Hilfsfeld ZEIGER(SAETZE) realisiert werden kann. In diesem Fall ist in das Feld ZEIGER( ) zu jedem Index die entsprechende Nummer einzutragen (einfache FOR ... NEXT-Schleife) und über den Index auch die Datensatzelemente, die sortiert werden sollen, zurückzugreifen.

Beim Vertauschen werden dann allerdings nur die Indexelemente vertauscht und kein Element des Datensatzes, auch nicht das zu sortierende. Dies funktioniert, da durch das Austauschen der Indizes in dem Feld ZEIGER( ) bereits sortierte Datensätze ausgeklammert werden. Diese Vorgehensweise macht es anhand einer Abfrage, nach welchem Datensatzelement sortiert werden soll, möglich, nach allen Datensatzelementen zu sortieren.

Wenn wir das aktuell zu überprüfende Element SORT nennen und in der Variablen WO die Nummer des Datensatzelementes, nach dem sortiert werden soll, festgehalten haben, so könnte der Zugriff wie folgt aussehen:

```
SORT = INHALT$(WO,ZEIGER(I))
```

Das Selektieren beim Drucken von Listen könnte mit dem vorhandenen Wissen auch keine Schwierigkeiten bereiten, da mitten in dem Listprogrammteil nur die Selektierabfrage eingebaut werden muß, um bei Nicht-Übereinstimmung das Ausdrucken zu unterlassen.

Sofern Sie ein Diskettenlaufwerk besitzen und Ihre Daten trotzdem auf Kassettenrecorder speichern oder vom Kassettenrecorder laden wollen, so setzen Sie in die entsprechende Programmroutine nur „|TAPE.IN“ oder „|TAPE.OUT“. Dadurch haben Sie alle Möglichkeiten, zwischen verschiedenen Geräten die Daten auszutauschen.

Kapitel 5
Sound

Das folgende Kapitel über die Sound-Möglichkeiten des CPC sollte im Zusammenhang mit dem Kapitel 6 des Handbuchs gelesen werden. Das vorliegende Kapitel ist als zusätzliche Lektüre gedacht. Deshalb wurde auch darauf verzichtet, die Syntax der Befehle noch einmal in allen Einzelheiten zu wiederholen. Hier sollen die Funktionsweise und die Anwendungsmöglichkeiten der Sound-Befehle näher beschrieben werden, als dies im Handbuch durchgeführt wurde. Wir wünschen Ihnen viel Vergnügen beim Lesen und Ausprobieren.

Der Schneider CPC hat bekanntlich zur Erzeugung von Klangeffekten und Programmierung von Musik drei Tongeneratoren eingebaut. Die Tonsignale stehen an der Klinkenbuchse an der Rückseite des Gehäuses (wenn man das Gerät von hinten betrachtet ganz rechts) in Stereo zur Verfügung. Dort kann man, wenn man im Basteln einige Erfahrung hat, den CPC mit dem Mikrofoneingang (oder einem ähnlichen) der Stereo-Anlage verbinden.

Wem dies zu kompliziert ist, der kann sich die Musik - wenn man die erzeugten Töne so nennen darf - auch über den eingebauten Lautsprecher anhören. Doch in diesem Kapitel soll uns weniger die Hardware interessieren als vielmehr die Programmierung des Sounds. Zum Glück haben die Erbauer des CPC auch an diejenigen gedacht, die der Maschinensprache des Z80 nicht mächtig sind, und im BASIC ausreichende Möglichkeiten zur Sound-Programmierung vorgesehen.

MÖGLICHKEITEN DER SOUND-PROGRAMMIERUNG

In diesem Kapitel wollen wir die wichtigsten Sound-Befehle anhand von Beispielen eingehend besprechen.

Der SOUND-Befehl

Der wichtigste Befehl ist SOUND. Geben Sie einmal im Direktmodus die Befehlsfolge

```
FOR h=1 TO 500 : SOUND 1,h : NEXT
```

ein, und hören Sie sich das Ergebnis an. Dazu müssen Sie natürlich den Lautstärkeregler an der rechten Gehäuseseite aufdrehen. Aus dem Lautsprecher erklingt ein Ton, der zunächst noch gar nicht hörbar ist (weil er zu hoch ist) und der im Gegensatz zu dem im SOUND-Befehl angegebenen Parameter h immer tiefer wird. Lesen Sie im Handbuch auf Seite 2 im Kapitel 6 nach:

Frequenz (in Hz) = 125000/Tonperiode

Der Parameter h gibt die im Handbuch erwähnte Tonperiode an. Die Frequenz des in unserem Beispiel erzeugten Tons läuft also von

125000/1 = 125000 Hz

abwärts bis zu

125000/500 = 250 Hz

Der maximale für die Tonperiode zulässige Wert ist 4095. Demnach hat der tiefste Ton, den der CPC hervorbringt, eine Frequenz von

125000/4095 = 30.5 Hz

Der Frequenzbereich, den der Mensch wahrnehmen kann, beginnt bei etwa 15 Hz und endet bei 15000 bis 20000 Hz. Die Hörfähigkeit für hohe Frequenzen nimmt mit dem Alter ab. Daß jemand Frequenzen bis zu 20 Hz wahrnehmen kann, dürfte die Ausnahme sein.

Wenn man mit dem CPC sehr hohe Töne erzeugt, fällt auf, daß der erzeugte Ton in der Frequenz stark von der im SOUND-Befehl angegebenen Frequenz abweicht. So kann man scheinbar Töne um 50 kHz (Kilohertz) noch wahrnehmen. Im üblicherweise für Musik verwendeten Tonbereich wirkt sich dieser Fehler jedoch kaum aus.

Wie berechnet man nun die Frequenz für eine bestimmte Note? Im SOUND-Befehl gibt man sie am besten wie folgt an:

```
SOUND 1,125000/Freq
```

Für „Freq" gibt man die gewünschte Frequenz ein. Auf diese Weise spart man sich das Umrechnen auf die Tonperiode. Zum Beispiel

```
SOUND 1,125000/440
```

ergibt den Ton a mit 440 Hz (über Kanal A). Der Ton a wird oft als Basis zur Errechnung der anderen Töne herangezogen (z. B. Stimmen eines Instruments). Um die Frequenz des nächsthöheren Tons, des a#/ais (b), zu erreichen, multipliziert man die Frequenz 440 Hz beim a mit 1.05946 (zwölfte Wurzel aus 2, da eine Oktave 12 Töne umfaßt und die Frequenz sich je Oktave verdoppelt/halbiert).

440 * 1.05946 = 466.162 Hz

das ist genau die Frequenz von a#. Die Frequenz des nächsthöheren Tons h ergibt sich demnach aus

466.162 * 1.05946 = 493.88 Hz

Für die Programmierung von Musikstücken ist es übersichtlicher, den Ton c als Basis zu verwenden. Die einzelnen Töne der Tonleiter erhalten Nummern, beginnend bei 0 für c. c# erhält die Nummer 1, d die Nummer 2 usw.

Für die Noten, die tiefer liegen als das c, gibt es negative Nummern. H erhält −1, A# erhält −2 usw. Dabei muß man die Halbtonschritte zwischen H und C und zwischen E und F beachten. In Abb. 5.1 finden Sie eine kleine Tabelle, die das Schema veranschaulicht.

C	−12	c#	1
C#	−11	d	2
D	−10	d#	3
D#	− 9	e	4
E	− 8	f	5
F	− 7	f#	6
F#	− 6	g	7
G	− 5	g#	8
G#	− 4	a	9
A	− 3	a#	10
A#	− 2	h	11
H	− 1	c1	12
c	0		

Abb. 5.1: Jeder Ton erhält eine Nummer

Aus der Nummer der Note kann man jetzt mit einer einfachen Formel die Frequenz der Note ausrechnen:

Freq = 261.626*1.05946^n

Bei der Formel ist 261.626 die Frequenz der Note C, 1.05946 der oben schon erwähnte Faktor zur Berechnung der benachbarten Noten und n die Nummer der Note, deren Frequenz man berechnen will.

Durch diese Numerierung der Noten ist es leicht möglich, Akkorde zu programmieren, z. B.:

```
10 DEF FN p(t) = 125000/(261.626*1.05946 ↑ t)
20 t = 0
30 SOUND 1,FN p(t)
40 SOUND 2,FN p(t + 4)
50 SOUND 4,FN p(t + 7)
```

Das obige Programm erzeugt einen C-Dur-Dreiklang. Durch Verändern von t in Zeile 20 kann man Dreiklänge in anderen Tonarten erhalten. In Zeile 10 ist eine Funktion vordefiniert, die aus der angegebenen Note (nach der obenstehenden Tabelle) die Periode für den SOUND-Befehl berechnet. Will man einen Moll-Akkord haben, ändert man einfach Zeile 40 wie folgt:

```
40 SOUND 2,FN p(t + 3)
```

Probieren Sie das kleine Programm einmal aus. Weiter wollen wir hier nicht in die Musiktheorie einsteigen, schließlich geht es in diesem Buch um Computer.

Wir kommen noch einmal zurück zum SOUND-Befehl und dessen Parameter. Wie Sie im Beispiel zur Programmierung von Akkorden gesehen haben, werden die drei Tonkanäle mit 1, 2 und 4 bezeichnet. Das CPC-BASIC übergibt den ersten Parameter im SOUND-Befehl direkt an den SOUND-Chip, der den übergebenen Wert als Binärzahl interpretiert. Jedes Bit dieser Zahl löst eine bestimmte Funktion aus. Die Zahl %00000001 (dezimal 1) z. B. schaltet den Generator A oder, wie im Handbuch bezeichnet, den Kanal A ein, %00000010 (dezimal 2) den Generator B und %00000100 (dezimal 4) den Generator C.

Man kann auch mit einem SOUND-Befehl mehrere Generatoren gleichzeitig einschalten, indem man die Zahlen addiert. Mit SOUND 7,125000/440 werden alle drei (7 = 1 + 2 + 4 = %00000111) Generatoren gleichzeitig eingeschaltet (alle erzeugen dann den gleichen Ton). Das kann z. B. dann sinnvoll sein, wenn der CPC an eine Stereo-Anlage angeschlossen ist und man bestimmte Stereo-Effekte erzeugen will.

Interessante Effekte erzielt man durch Überlagerung nahe beieinanderliegender Frequenzen. Dadurch entstehen sogenannte Schwebungen, wie man sie von Propellerflugzeugen oder Kreissägen kennt, z. B.:

```
10 SOUND 1,100,1000
20 SOUND 2,101,1000
```

Deutlich hört man, wie die Tonhöhe auf und ab „schwingt". Das geht natürlich auch auf drei Kanälen:

```
10 SOUND 1,100,1000
20 SOUND 2,101,1000
30 SOUND 4,102,1000
```

Versuchen Sie durch Variation der Tonperioden neue Klänge zu finden. Wichtig ist dabei, daß die Werte für die Tonperioden der einzelnen Kanäle eng beieinanderliegen. Die besten Ergebnisse erzielt man mit Abweichungen, die weniger als 5 Prozent betragen. Betrachten wir noch ein Beispiel:

```
10 SOUND 1,4000,1000
20 SOUND 1,4008,1000
```

Das Geräusch, das der Computer dabei erzeugt, hat eine gewisse Ähnlichkeit mit dem Geräusch eines kreisenden Hubschraubers. Leichte Schwebungen lassen sich auch noch beim Zusammenklang von Tönen, die sich um fast eine Oktave unterscheiden, feststellen. Wie Sie weiter vorne lesen konnten, erhält man einen Ton, der um eine Oktave höher liegen soll, wenn man die Frequenz verdoppelt; bzw. man halbiert die Frequenz, wenn der Ton eine Oktave tiefer liegen soll.

```
10 SOUND 1,2002,1000
20 SOUND 2,4000,1000
```

Wie Sie sehen, kann man dem Rechner viele interessante Geräusche entlokken. Zum Schluß hören wir einen verbesserten Hubschrauber, dessen Funktionsweise etwas später vollständig geklärt werden wird:

```
10 SOUND 1,4004,1000,,,,15
20 SOUND 2,4000,1000
```

Die bisher erzeugten Töne waren alle gleich lang und gleich laut. Dauer und Lautstärke eines Tons können jedoch auch im SOUND-Befehl angegeben werden. Die Tondauer wird in 1/100stel Sekunden hinter der Periode angegeben. Mit dem folgenden Parameter wählt man die Lautstärke, z. B.:

```
SOUND 4, 125000/440,250,15
```

Dieser Befehl gibt über den Generator C für 2,5 Sekunden (250/100stel Sekunden) den Ton A (440 Hz) mit der Lautstärke 15 aus. Die Lautstärke kann zwischen 0 und 15 variiert werden.

Der ENV-Befehl

Für den Musikfan wird es allerdings wenig befriedigend sein, immer nur Töne zu spielen, die von Anfang bis Ende in der gleichen Lautstärke erklingen. Bei den meisten Musikinstrumenten ändert sich die Lautstärke des gespielten Tons mit der Zeit, z. B. bei allen (akustischen) Instrumenten, bei denen Sai-

ten angezupft oder angeschlagen werden. Wird bei solchen Instrumenten ein Ton angeschlagen, erklingt er zuerst relativ laut und wird dann immer leiser, da die schwingende Saite Energie in Form von Schall abgibt.

Um eine Folge von immer leiser werdenden Tönen, die als ein Ton erklingen sollen, in einem Musikstück zu programmieren, ist BASIC zu langsam. So eine schnelle und flüssig erscheinende Lautstärkeänderung ist nur in Maschinensprache zu bewerkstelligen. Deshalb ist im Schneider-BASIC ein Befehl enthalten, mit dem man den Lautstärkeverlauf für einen Ton, eine sogenannte Lautstärkehüllkurve, festlegen kann. Mit dem ENV-Befehl hat man die Möglichkeit, 15 Lautstärkehüllkurven festzulegen und eine davon im SOUND-Befehl aufzurufen.

Wie schon gesagt, kann man die Lautstärke von 0 bis 15 variieren. Der zeitliche Verlauf der Lautstärke setzt sich aus einzelnen Abschnitten zusammen, von denen man bis zu fünf mit einem ENV-Befehl festlegen kann.

Innerhalb eines Abschnittes steigt oder fällt die Lautstärke gleichmäßig. Der erste Abschnitt beginnt immer mit der Lautstärke 0. Von 0 kann man die Lautstärke zu einem bestimmten Wert ansteigen lassen, indem man angibt, wie oft die Lautstärke erhöht werden soll, um wieviel jeweils die Lautstärke erhöht werden soll und wie lange nach jeder Änderung die Laustärke anhalten soll, bevor sie wieder verändert wird.

Gibt man für die Dauer der einzelnen Schritte - im Handbuch als Pausenlänge bezeichnet – 1 ein und macht nur einen Schritt mit der Schrittweite 15, so kann man die Hüllkurve trotzdem bei 15 beginnen lassen, sozusagen mit dem zweiten Abschnitt. Bei weiteren Abschnitten geht man von der Endlautstärke des vorhergehenden Abschnittes aus. Man kann die Lautstärke wieder abfallen lassen, indem man für die Schrittgröße, also den Wert, um den die Lautstärke bei jedem Schritt erhöht wird, eine negative Zahl angibt, oder die Lautstärke noch weiter ansteigen lassen, sofern sie nicht schon auf 15 eingestellt ist.

Die dritte Möglichkeit ist die, daß man die Lautstärke konstant hält, was man erreicht, indem man als Schrittgröße 0 angibt. Es gibt also vier verschiedene Typen von Lautstärkeabschnitten:

1. Der erste Typ läßt die Lautstärke in gleichmäßigen Schritten ansteigen; die Endlautstärke erhält man, wenn man die Schrittzahl mit der Schrittgröße multipliziert und das Ergebnis zur Endlautstärke des vorherigen Abschnittes hinzuzählt. Die Dauer des gesamten Abschnittes erhält man, wenn man die Schrittzahl mit der Pausenlänge multipliziert.
2. Der zweite Typ verringert die Lautstärke gleichmäßig, indem bei jedem

Schritt eine negative Schrittgröße zur Lautstärke addiert wird. Die Schrittgröße muß als negative Zahl im ENV-Befehl angegeben werden. Bei beiden Abschnittstypen ist es ratsam, die Schrittgröße und die Pausenlänge möglichst klein zu wählen und so viele Einzelschritte zu machen, wie es die Abschnittsdauer zuläßt, um eine möglichst flüssige Änderung zu bekommen.

3. Der dritte Typ hält die Lautstärke über eine Zeit konstant. Die Schrittzahl ist 1, die Schrittgröße ist 0, und die Pausenlänge ist gleich der Dauer des ganzen Abschnittes.
4. Der vierte Typ schaltet die Lautstärke augenblicklich auf einen neuen Wert um. Als Schrittzahl wählt man wieder 1, als Pausenlänge auch 1, und die Schrittgröße ergibt sich als Differenz zwischen der alten und der neuen Lautstärke.

Alle Abschnitte müssen im Befehl mit drei Werten beschrieben werden, damit das BASIC die Werte der Hüllkurve richtig zuordnet. Die Reihenfolge der Eingabe ist dabei:

- Schrittzahl
- Schrittgröße
- Pausenlänge

Sehen wir uns ein kleines Beispiel an:

```
10 ENV 1,3,5,50
20 SOUND 1,100,0,0,1
```

Die 1 im ENV-Befehl ist die Nummer der Hüllkurve. Sie muß im SOUND-Befehl als fünfter Parameter (im Beispiel die letzte Angabe im SOUND-Befehl) angegeben werden. Danach steht die Schrittzahl: drei Schritte. Bei jedem Schritt wird die Lautstärke um 5 Einheiten erhöht. Die Lautstärkestufen sind dann im einzelnen 5, 10 und 15. Die letzte Zahl gibt die Pausenlänge an. Jeder Schritt dauert eine halbe (50/100) Sekunde.

Über den Tonkanal A erklingt durch den SOUND-Befehl der Ton mit der Tonperiode 100. Die Lautstärke und die Dauer sind mit 0 angegeben, da sie von der im letzten Parameter angegebenen Hüllkurve 1 gesteuert werden. Ist die Lautstärke nicht auf 0 gesetzt, so schaltet die Lautstärke nach Beendigung der Hüllkurve noch einmal kurz auf den im SOUND-Befehl angegebenen Wert.

Starten Sie jetzt das Programm mit RUN, und hören Sie sich das Ergebnis an. Deutlich sind die drei Schritte auszumachen. Ändern Sie jetzt die Zeile 10 ab:

```
10 ENV 1,15,1,5
```

Damit haben Sie 15 Schritte mit der Schrittgröße 1 und der Dauer 5 definiert. Das Ergebnis ist ein Ton, der schnell und ziemlich gleichmäßig lauter wird. Erweitern wir den Befehl noch um einen zweiten Schritt:

```
10 ENV 1,15,1,5,15,-1,5
```

Es handelt sich bei der Erweiterung um einen Abschnitt vom oben beschriebenen Typ 2. Die Schrittgröße ist −1. Damit nimmt die Lautstärke 15mal um 1 ab und geht so wieder auf 0 zurück. Bei solchen Abschnittsfolgen sollte man immer darauf achten, daß die Lautstärke nicht unter 0 fällt. Wenn man die Lautstärke noch weiter zu verringern versucht, wird sie wieder um 15 erhöht, und der Rechner arbeitet von diesem erhöhten Wert aus die Hüllkurve weiter ab. Den Effekt kann man sich anhören, wenn man im zweiten Abschnitt die Schrittzahl erst auf 30 und dann auf 45 vergrößert:

```
10 ENV 1,15,1,5,30,-1,5
RUN
10 ENV 1,15,1,5,45,-1,5
RUN
```

Das kann man allerdings auch ausnutzen, um Abschnitte zu sparen. Mit

```
10 ENV 1,15,-1,5
```

erhält man eine Hüllkurve, die bei 15 beginnt und dann auf 0 absinkt. Umgekehrt geht das natürlich auch. Wenn der Wert über 15 steigt, wird er automatisch um 15 vermindert. Der Befehl

```
10 ENV 1,127,1,5
```

erzeugt eine Hüllkurve, in der die Lautstärke langsam periodisch ansteigt und schlagartig auf 0 abfällt. Mehr als 127 Schritte verkraftet der Befehl allerdings nicht. Auch wenn man Abschnitte konstanter Lautstärke programmiert, kann man sich so manchen Schritt sparen:

```
10 ENV 1,1,15,10,1,-15,60,1,15,80,1,-15,60
20 SOUND 1,100,0,0,1
30 GOTO 20
```

Das Programm erzeugt ein „Besetztzeichen" wie beim Telefon. Die Hüllkurve enthält vier Abschnitte, die jeweils aus einem Schritt bestehen. Der erste Abschnitt hält eine Zehntelsekunde lang die Lautstärke auf 15, der zweite dauert 0,6 Sekunden und verringert die Lautstärke um 15 auf 0. Es entsteht nach dem ersten Schritt dadurch eine Pause von 0,6 Sekunden. Es folgt wieder für 0,8 Sekunden ein Ton mit Lautstärke 15 und danach eine Pause.

In Abb. 5.2 bis 5.5 sehen Sie noch einmal eine Zusammenstellung der vier Typen von Abschnitten:

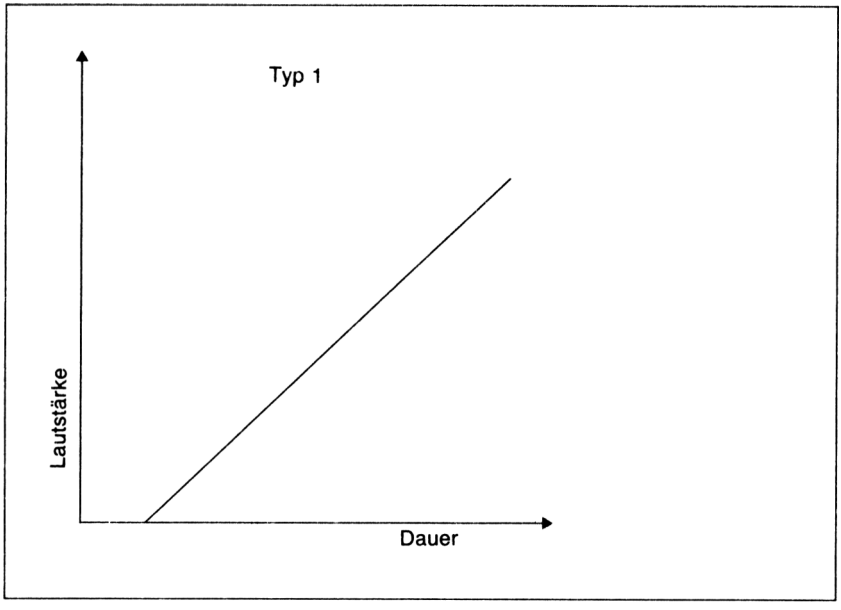


Abb. 5.2: Die Lautstärke steigt gleichmäßig an

Typ 1: Die Schrittgröße ist positiv, mit jedem Schritt steigt die Lautstärke an. Die Dauer errechnet man durch Multiplikation der Schrittzahl mit der Pausenlänge. Die Endlautstärke erhält man, indem man die Schrittgröße mit der Schrittzahl multipliziert und das Ergebnis zur Endlautstärke des vorhergehenden Abschnittes addiert.

Wird dabei der Wert 15 überschritten, so muß, sooft diese Grenze überschritten wird, von dem Ergebnis 15 abgezogen werden. Die Hüllkurve hat dann mehr Abschnitte als im ENV-Befehl angegeben.

Typ 2: Durch die negative Schrittgröße wird erreicht, daß die Lautstärke mit jedem Schritt abnimmt. Die Dauer des Abschnittes errechnet sich wie beim Typ 1. Beim Berechnen der Endlautstärke muß man allerdings, anders als beim Typ 1, beim Unterschreiten der Lautstärke 0 jedesmal 15 dazuzählen.

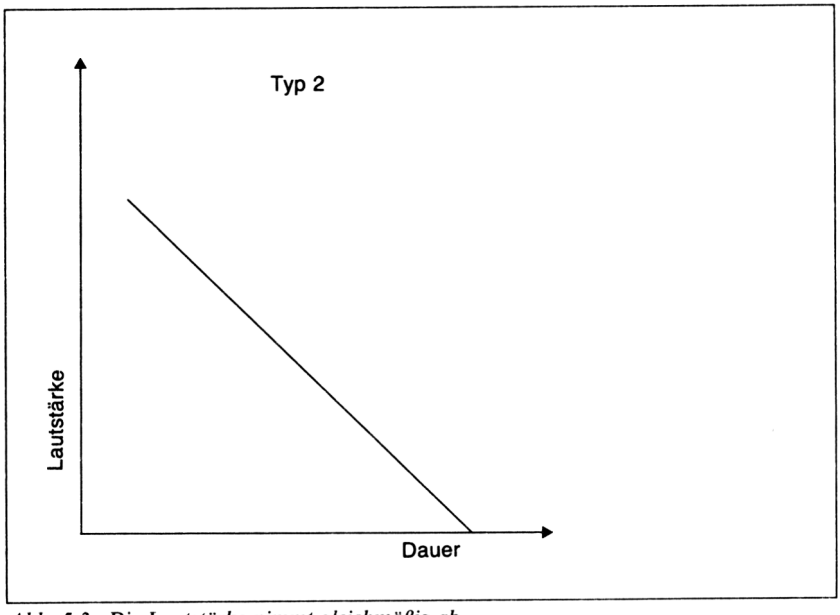


Abb. 5.3: Die Lautstärke nimmt gleichmäßig ab

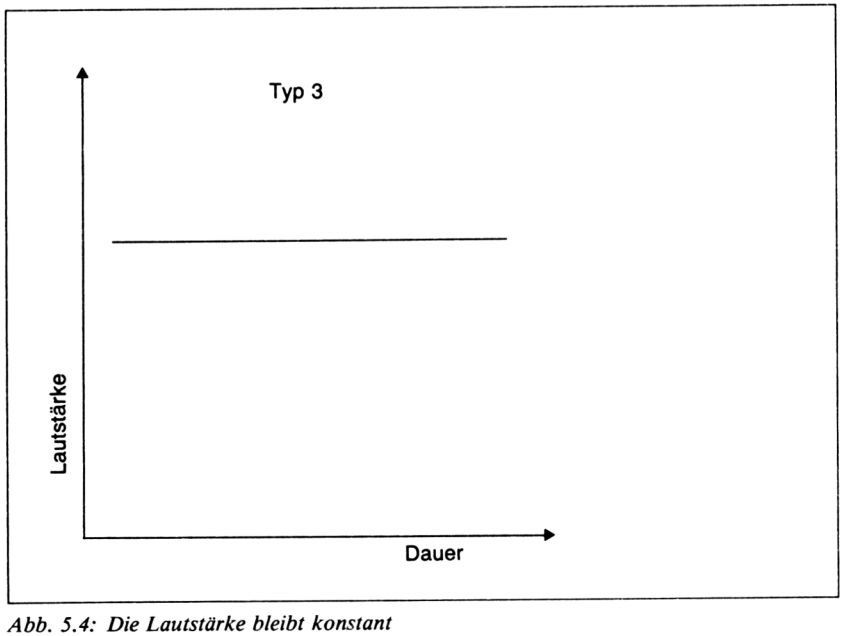


Abb. 5.4: Die Lautstärke bleibt konstant

Typ 3: Die Lautstärke bleibt für die angegebene Zeit konstant. Man definiert einen Schritt, wodurch der Abschnitt genau so lang ist wie die Pausenlänge. Soll die Endlautstärke des vorhergehenden Abschnittes beibehalten werden, so gibt man als Schrittgröße 0 an.

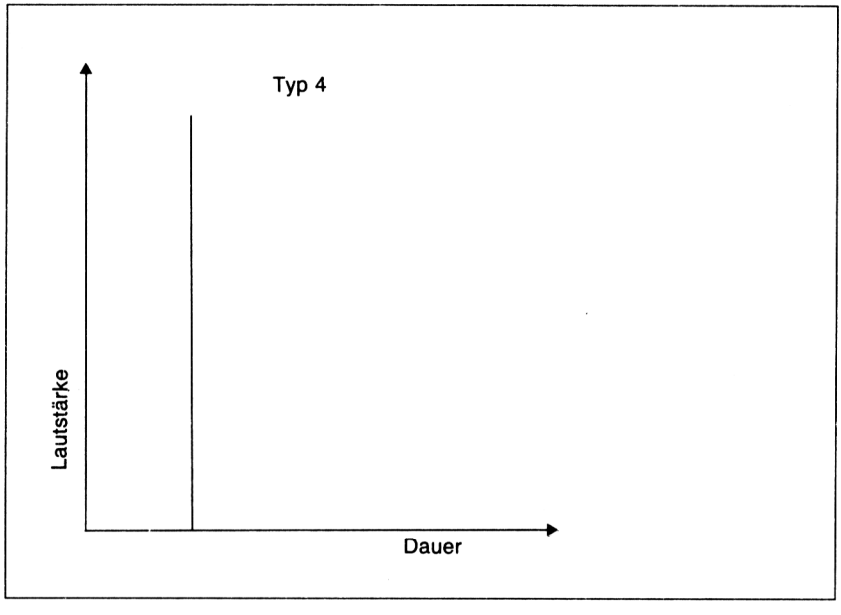


Abb. 5.5: Die Lautstärke wird übergangslos umgestellt

Typ 4: Die Lautstärke soll schlagartig einen bestimmten Wert annehmen. Die Schrittzahl und die Dauer werden auf 1 gesetzt. Die Schrittgröße ist die Differenz von der alten zur neuen Lautstärke.

Für alle Parameter des ENV-Befehls müssen ganze Zahlen angegeben werden.

Der ENT-Befehl

Der ENV-Befehl war eine Variante, um den Verlauf eines Tons, nämlich seine Lautstärke, zu beeinflussen. Eine andere Variante stellt der ENT-Befehl dar. Mit dem ENT-Befehl ist es möglich, eine Hüllkurve wie für die Lautstärke auch für die Tonhöhe anzugeben.

Auch hier kann man wieder 15 Hüllkurven vordefinieren. Im SOUND-Befehl wird die Nummer der Tonhüllkurve an der sechsten Stelle angegeben.

Die ENT-Kurve ist genau wie die ENV-Kurve in bis zu fünf Abschnitte unterteilbar. Die Programmierung ist ähnlich wie bei der ENV-Kurve. Es wird zur Definition eines Abschnittes die Schrittzahl, die Schrittgröße (in Prozent) und die Pausenlänge benötigt.

Beispiel:

```
10 ENT 1,15,1,10
20 SOUND 1,100,150,7,,1
```

Bei Verwendung der ENT-Kurve im SOUND-Befehl müssen Dauer und Lautstärke trotzdem angegeben werden, sofern man nicht noch zusätzlich eine ENV-Kurve verwendet!

Wenn Sie das obenstehende Beispielprogramm starken, hören Sie einen Ton, der niedriger wird. Die Veränderungen der Tonhüllkurve beziehen sich auf die Tonperiode. Genaugenommen haben wir es mit einer Tonperiodenhüllkurve zu tun.

Im Beispiel wird die Abweichung der Tonperiode 15mal, jeweils mit einer Zehntelsekunde (10/100) Abstand, um 1 erhöht. Dadurch, daß die Tonperiode erhöht wird, wird der Ton natürlich tiefer.

Die Dauer des SOUND-Befehls ist auf die Dauer der Tonhüllkurve von 15mal einer Zehntelsekunde abgestimmt. Dauert der SOUND-Befehl weniger lange, dann wird die Tonhüllkurve vorzeitig abgebrochen. Dauert der SOUND-Befehl länger, als es in der Definition der Tonhüllkurve vorgesehen ist, dann bleibt während der übrigen Zeit der Ton auf der Höhe, die die Hüllkurve zuletzt angegeben hat.

Soll die Hüllkurve, wenn sie bis zum Ende durchgespielt wurde, wiederholt werden, dann muß die Nummer der Hüllkurve ein Minuszeichen bekommen, und zwar im ENT-Befehl und nicht wie bei der ENV-Kurve im SOUND-Befehl:

```
10 ENT -15,10,5,5,10,-5,5
20 SOUND 1,200,1000,7,,15
```

15 ist die Hüllkurvennummer. Das Minuszeichen bedeutet, daß die Kurve wiederholt werden soll. Danach sind zwei Abschnitte definiert. Beim ersten steigt die Tonperiode um 50 (5×10) an, beim zweiten fällt sie wieder um 50 ab (-5×10).

Auch bei der ENT-Kurve kann man den für den SOUND-Befehl zulässigen Wertebereich - in diesem Fall handelt es sich um den Wertebereich für die Tonperiode - überschreiten. Verläßt man den Wertebereich über die eine

Grenze, kommt man über die andere Grenze in den Wertebereich hinein. Das bedeutet: Weicht die Tonperiode so weit von dem im SOUND-Befehl angegebenen Wert ab, daß sie kleiner als 1 wird, dann folgen bei weiterer Abänderung wieder tiefe Töne. Das heißt, daß bei Überschreiten dieser Grenze zum Wert der Tonperiode wieder 4096 hinzugezählt werden.

Weicht die Tonperiode so weit nach oben ab, daß sie 4096 überschreitet, dann folgen danach wieder hohe Töne; es werden also 4096 abgezogen (vgl. Kapitel 6 im Handbuch).

Beispiel:

```
10 ENT 1,239,-30,10
20 SOUND 1,100,2390,7,,1
```

Der Ton ist erst tief, wird dann langsam höher und wiederholt sich, wenn er eine bestimmte Frequenz überschritten hat. Das umgekehrte Beispiel sieht so aus:

```
10 ENT 1,239,30,10
20 SOUND 1,100,2390,7,,1
```

Die weiter vorne schon erwähnte Überlagerung von Frequenzen ist im Zusammenhang mit der Verwendung von Tonhüllkurven besonders interessant.

```
10 ENT 1,5,-1,20
20 SOUND 1,100,100,7,,1
30 SOUND 2,100,100
```

Mit der Tonfrequenz ändert sich auch die Geschwindigkeit, mit der die Schwebungen sich wiederholen.

Zusammenfassung: Der ENT-Befehl verändert die Tonperiode im Verlauf des durch SOUND ausgegebenen Tons. Die ENT-Kurve kann aus bis zu fünf Abschnitten bestehen, von denen jeder durch die drei Werte: Schrittzahl, Schrittgröße, Pausenlänge beschrieben wird.

Man kann die gleichen Typen von Abschnitten eingeben, wie sie beim ENV-Befehl beschrieben sind.

Die ENT-Kurve beginnt bei dem Wert der Tonperiode, der im SOUND-Befehl angegeben ist.

Die Tonperiode, die am Ende eines Abschnittes gerade gespielt wird, erhält man, wenn man zur Tonperiode, die vor Beginn des Abschnittes eingestellt war, die Schrittzahl des Abschnittes, multipliziert mit der Schrittgröße, addiert.

Die Länge der Abschnitte errechnet sich wie bei den Abschnitten der ENV-Kurve.

Im SOUND-Befehl müssen Dauer und Lautstärke angegeben werden.

Will man eine Tonhüllkurve, die weniger lange dauert als der Ton, der gespielt wird, wiederholen, so muß man - im Gegensatz zur ENV-Kurve - beim ENT-Befehl eine negative Hüllkurvennummer angeben.

Das Rauschen

Die letzte hier beschriebene Möglichkeit, den Klang zu beeinflussen, bietet die sogenannte Geräuschperiode. Als letzten Parameter im SOUND-Befehl kann man eine Zahl zwischen 1 und 15 angeben, wodurch dem Ton ein Rauschen beigemischt wird. Allerdings hat man dabei eher den Eindruck, als sei dem Rauschen ein Ton beigemischt, so penetrant erklingt dieses Rauschen, dessen Lautstärke leider nicht zu beeinflussen ist.

```
SOUND 1,100,500,7,,,1
```

Mit den Zahlen 1 bis 15 kann man sich aussuchen, welche Frequenzen im Rauschen vorherrschen sollen. Das Rauschen klingt bei 1 hell und bei 15 dumpf. Die Abstufungen dazwischen ergeben kaum wesentliche Unterschiede.

Das Rauschen kann auch allein erklingen, wenn der SOUND-Befehl keine Angabe für die Tonperiode enthält, d. h. wenn als zweiter Parameter im SOUND-Befehl eine 0 angegeben ist.

```
SOUND 1,0,500,7,,,1
```

Und hier ergibt sich jetzt doch noch eine Möglichkeit, die Lautstärke des Rauschens zu regulieren: Man läßt einfach in den einen Kanal einen Ton laufen und über den anderen Kanal ein Rauschen ohne Ton.

```
10 SOUND 1,0,500,3,,,1
20 SOUND 2,100,500,7
```

Probieren Sie jetzt einmal die verschiedenen Rauscharten von 1 bis 15 durch. Ein Beispiel für das Rauschen Nr. 15 haben Sie schon am Anfang des Kapitels gehört (wenn Sie die zwei Programmzeilen ausprobiert haben). Es war der Hubschrauber.

Jetzt folgt noch ein kleines Beispiel, in dem alle 15 Rauschperioden vorkommen:

```
10 ENV 1,1,15,1,15,-1,20
20 FOR i=1 T0 15
30       SOUND 1,0,10,7,,,i
40 NEXT i
50 SOUND 1,0,0,0,1,,15
```

Nachdem das Programm 15mal eine Rauschperiode in den Kanal A geschickt hat und bei der Rauschperiode 15 angekommen ist, klingt das Rauschen 15 mit der ENV-Kurve 1 langsam aus. Es entsteht der Eindruck, als entferne sich die Geräuschquelle langsam (oder auch schnell).

Noch eine Besonderheit ist zu beachten: Wenn über einen Kanal gerade ein Rauschen ausgegeben wird und auf einem anderen Kanal ein SOUND-Befehl an die Reihe kommt, der ein anderes Rauschen ausführen soll, dann ändert sich das Rauschen, das über den erstgenannten Kanal läuft. Über beide Kanäle hört man das gleiche Rauschen, und zwar das, welches zuletzt aufgerufen wurde. Folgendes Beispiel geben Sie dazu bitte im Direktmodus ein:

```
SOUND 1,0,9000,,,,15
SOUND 2,0,9000,,,,1
```

Hören Sie, was passiert? Wenn Ihnen das Rauschen zu bunt wird, gibt es verschiedene Möglichkeiten, die Tonausführung sofort abzubrechen: Die erste ist die Eingabe einer BASIC-Zeile, z. B. „1 : (ENTER)". Im Direktmodus unterbricht RUN sofort alle Töne, ganz gleich, ob sich im Speicher ein Programm befindet oder nicht, ebenso die ESC-Taste.

Im Programm erreicht man einen sofortigen Abbruch mit STOP. Bei END geht die Tonausführung über das Programmende hinaus weiter. Einen sofortigen Tonstop ohne Programmabbruch für einen Kanal erreicht man, indem man die Lautstärke des Kanals mit sofortiger Wirkung (!) auf Null setzt. Das geht so:

```
SOUND 1+128,0
```

Das Beispiel gilt für Kanal A. Für Kanal B müßte der erste Parameter „2+128" lauten. Sie können natürlich die einzelnen Werte auch zusammenzählen und gleich das Ergebnis angeben, z. B.:

```
SOUND 135,0
```

für alle Kanäle. Dadurch, daß man die Kanalnummer (1, 2, 4) um 128 erhöht, teilt man dem Rechner mit, daß der SOUND-Befehl sofort ausgeführt werden soll. Aber das ist schon das Thema des nächsten Abschnittes.

Das Timing

Nun kommen wir zu etwas Neuem. Im Handbuch ist von Haltepunkten (Holds), Rendezvous und Flushs die Rede. Allerdings ist es dem Handbuch kaum zu entnehmen, was es damit, vor allem mit den Rendezvous, auf sich hat. Da hilft dann nur noch ausprobieren.

Um den Begriff Rendezvous im Zusammenhang mit der Sound-Programmierung beim CPC zu verstehen, stellt man sich die Kanäle als Warteschlangen vor, in denen die SOUND-Befehle darauf warten, ausgeführt zu werden.

Jedem SOUND-Befehl, der in die Warteschlange geschickt wird, kann man eine Nachricht (Information) mitgeben, die der Kanalnummer, also dem ersten Parameter im Befehl, beigefügt wird. Jedem SOUND-Befehl können mehrere verschiedene Informationen mitgegeben werden, solange dies sinnvoll ist. Jede Nachricht kann natürlich nur einmal beigefügt werden.

Die einzelnen Informationen, die man den SOUND-Befehlen mitgeben kann, sind im Handbuch im Kapitel 6, Seite 6, in einer Tabelle aufgeführt. Darin steht z. B., daß man die Kanäle A, B und C mit den Zahlen 1, 2 und 4 aufruft. Die Nachrichten, daß ein Rendezvous mit den Kanälen A, B oder C stattfinden soll, erhalten die Nummern 8, 16 und 32.

Diese Zahlen sind so gewählt, daß eine beliebige Kombination der Zahlen wieder nur eine (!) bestimmte Zahl ergibt, so daß keine Mißverständnisse entstehen können. Wenn man einen Ton nach Kanal A (Zahl 1) schickt und ihm die Nachricht mitgeben will, er soll ein Rendezvous mit Kanal C (Zahl 32) ausführen (was immer das auch sein mag), so addiert man $1 + 32 = 33$. Als ersten Parameter gibt man demnach 33 an.

Was ist nun ein solches Rendezvous? Stellen wir uns wieder die drei Warteschlangen A, B und C vor. In allen drei warten SOUND-Befehle darauf, ausgeführt zu werden. In Kanal B ist nun ein SOUND-Befehl, der die Nachricht mitführt, er solle ein Rendezvous mit einem SOUND-Befehl in Kanal C ausführen. Der Befehl in der Schlange von Kanal B hat die Zahl 34 (32 + 2) als ersten Parameter mitbekommen. Das Schneider-BASIC hält diesen Befehl zurück, wodurch auch alle Befehle, die in der Schlange hinter ihm stehen, aufgehalten werden. In den Kanälen A und C geht die Abfertigung der SOUND-Befehle ohne Unterbrechung weiter.

In Kanal C steht bereits ein SOUND-Befehl in der Warteschlange, der die Nachricht für das BASIC hat, er solle ein Rendezvous mit einem Befehl in Kanal B ausführen. Er hat die Nummer 20 (4 + 16). Das ist der Befehl, auf den der Befehl mit der Zahl 34 in Kanal B gewartet hat. Sobald der Befehl 20 in Schlange C an der Reihe ist, darf auch der „34er" in Schlange B passieren.

Beide verlassen gemeinsam ihre Warteschlangen, während Kanal A von alledem unberührt bleibt. Schicken wir also einen Ton in Kanal B und geben ihm die Nachricht mit, er solle auf einen Ton im Kanal C mit einer entsprechenden Nachricht warten:

```
SOUND 2+32,100
```

Nichts passiert. Zur Kontrolle schicken wir einen Ton ohne Nachricht in den Kanal C:

```
SOUND 4,200
```

Der Befehl wird sofort ausgeführt. Jetzt schicken wir einen tiefen Ton (zur Unterscheidung) nach C und geben ihm die Nachricht mit, er solle ein Rendezvous mit dem Ton in Kanal B ausführen:

```
SOUND 4+16,1000
```

Beide Töne erklingen gleichzeitig. Natürlich kann man auch zwei Kanäle, etwa A und B, mit Tönen laden, die eine Rendezvous-Anweisung für den dritten Kanal erhalten. In den dritten Kanal schickt man einen Ton, der sowohl die Nachricht für ein Rendezvous mit dem ersten, als auch eine Nachricht für ein Rendezvous mit dem zweiten Kanal mitführt. Die einzelnen Nachrichten werden einfach addiert.

Der dritte Befehl startet dann alle drei Kanäle gleichzeitig. Das probieren wir in einem Beispiel aus:

```
SOUND 1+32,100
SOUND 1,200
```

Der erste Kanal wird mit zwei Tönen geladen, ebenso der zweite:

```
SOUND 2+32,400
SOUND 2,800
```

Jetzt schicken wir den Befehl in Kanal C, der die Rendezvous-Anweisung für A und B enthält:

```
SOUND 4+8+16,1000
```

Was passiert, hören Sie selbst.

Der Witz solcher Rendezvous besteht darin, daß man Musikstücke mit mehr als einer Stimme programmieren kann. Dabei geht man am besten folgendermaßen vor: Man lädt jeweils ein paar Töne der zweiten (und dritten) Stimme, mit Rendezvous-Anweisungen versehen, auf Vorrat in die Kanäle. Die erste Stimme erhält die Rolle der Führungsstimme, die die anderen Töne bei Bedarf aufruft. Dies soll ein kleines Beispiel im folgenden Abschnitt illustrieren.

EIN KLEINES MUSIKSTÜCK

Abb. 5.6: Der Entertainer

```
1000 REM ------------------------------
1010 REM --- Rendezvousbeispiel :   ---
1020 REM ---                       ---
1030 REM --- Auszu■aus             ---
1040 REM ---                       ---
1050 REM --- "The Entertainer"     ---
1060 REM ---                 von   ---
1070 REM --- Scott Joplin           ---
1080 REM ------------------------------
1090 :
1100 :
1110 REM ---  Felder fuer Noten     ---
1120 :
1130 DIM st(1000),n(1000),l(1000)
1140 DIM st2(500),n2(500),l2(500)
1150 :
2000 REM ------------------------------
2010 REM ---Einleseschleife f. Noten---
2020 REM ------------------------------
2030 :
2040 i=0
2050 j=0
2060 :
2070 REM ---  erste Stimme          ---
2080 :
2090 READ a,b,c
2100 st(i)=a
2110 IF b=99 THEN n(i)=0 ELSE n(i)=125000/(261.626*1.05
946^b)
2120 l(i)=200/c
2130 PRINT i;st(i),n(i),l(i)
2140 i=i+1
2150 IF a=-1 THEN 3040
2160 IF a=-2 THEN 2210
2170 GOTO 2090
2180 :
2190 REM ---  zweite Stimme         ---
2200 :
2210 READ a,b,c
2220 st2(j)=a
2230 IF b=99 THEN n2(j)=0 ELSE n2(j)=125000/(261.626*1.
05946^b)
2240 l2(j)=200/c
2250 PRINT TAB(40);j;st2(j),n2(j),l2(j)
2260 j=j+1
2270 IF a=-3 THEN 2090
2280 GOTO 2210
2290 :
3000 REM ------------------------------
3010 REM --- Hauptprogramm          ---
3020 REM ------------------------------
3030 :
3040 i=-1
3050 j=-1
3060 :
3070 REM --- zweite Stimme in Kanal ---
3080 REM --- B laden                ---
3090 :
3100 j=j+1
3110 IF st(i+1)=-1 THEN 4020
3120 IF st2(j)=-3 THEN 3190
3130 SOUND st2(j),n2(j),l2(j)
3140 GOTO 3100
3150 :
3160 REM --- erste Stimme in Kanal  ---
3170 REM --- A laden und ausfuehren ---
3180 :
3190 i=i+1
3200 IF st(i)=-2 THEN 3100
3210 SOUND st(i),n(i),l(i)
3220 GOTO 3190
3230 :
4000 REM --- Ende/Warten auf ENTER  ---
```

```
4010 :
4020 WHILE INKEY$<>CHR$(13):WEND
4030 GOTO 3040
4040 :
9000 REM ------------------------------
9010 REM --- Datas fuer Noten        ---
9020 REM ------------------------------
9030 :
9040 REM Takt 1:
9050 DATA 17,2,8,1,3,8,-2,1,1
9060 DATA 10,-1,5,-3,1,1
9070 REM Takt 2:
9080 DATA 17,4,8,1,12,4,1,4,8,17,12,4,17,4,8,1,12,8,-2,
1,1
9090 DATA 10,-5,4,2,0,5,10,-5,5,10,-2,5,-3,1,1
9100 REM Takt 3:
9110 DATA 17,12,2,17,99,8,1,24,8,17,26,8,1,27,8,-2,1,1

9120 DATA 10,-7,4,2,5,5,10,-8,5,10,17,8,-3,1,1
9130 REM Takt 4:
9140 DATA 17,28,8,1,24,8,17,26,8,17,28,4,17,23,8,1,26,4
,-2,1,1
9150 DATA 10,19,5,10,17,9,10,19,5,10,14,9,-3,1,1
9160 REM Takt 5:
9170 DATA 17,24,4,17,12,2,1,2,8,1,3,8,-2,1,1
9180 DATA 10,16,5,10,4,2,2,-1,5,-3,1,1
9190 REM Takt 6:
9200 DATA 17,4,8,1,12,4,1,4,8,17,12,4,17,4,8,1,12,8,-2,
1,1
9210 DATA 10,-5,4,2,0,5,10,-5,5,10,-2,5,-3,1,1
9220 REM Takt 7:
9230 DATA 17,12,2,17,99,4,17,24,8,1,19,8,-2,1,1
9240 DATA 10,-7,4,2,5,5,10,-8,5,10,3,5,-3,1,1
9250 REM Takt 8:
9260 DATA 17,18,8,1,21,8,17,24,8,1,28,8,17,28,8,1,26,8,
17,24,8,1,21,8,-2,1,1
9270 DATA 10,-3,5,10,2,5,10,-3,5,10,6,5,-3,1,1
9280 REM Takt 9:
9290 DATA 17,26,4,17,7,4,17,9,4,17,2,8,1,3,8,-2,1,1
9300 DATA 10,7,5,10,-5,5,10,-3,5,10,-1,5,-3,1,1
9310 REM Takt 10:
9320 DATA 17,4,8,1,12,4,1,4,8,17,12,4,17,4,8,1,12,8,-2,
1,1
9330 DATA 10,-5,4,2,0,5,10,-5,5,10,-2,5,-3,1,1
9340 REM Takt 11:
9350 DATA 17,12,2,17,99,8,1,24,8,17,26,8,1,27,8,-2,1,1
9360 DATA 10,-7,4,2,5,5,10,-8,5,10,17,8,-3,1,1
9370 REM Takt 12:
9380 DATA 17,28,8,1,24,8,17,26,8,17,28,4,17,23,8,1,26,4
,-2,1,1
9390 DATA 10,19,5,10,17,8,10,19,5,10,14,8,-3,1,1
9400 REM Takt 13:
9410 DATA 17,24,4,17,12,2,1,24,8,1,26,8,-2,1,1
9420 DATA 10,16,5,10,4,2,-3,1,1
9430 REM Takt 14:
9440 DATA 17,28,8,1,24,8,17,26,8,1,28,8,17,28,8,1,24,8,
17,26,8,1,24,8,-2,1,1
9450 DATA 10,12,5,10,12,5,10,10,5,10,11,5,-3,1,1
9460 REM Takt 15:
9470 DATA 17,28,8,1,24,8,17,26,8,1,28,8,17,28,8,1,24,8,
17,26,8,1,24,8,-2,1,1
9480 DATA 10,9,5,10,9,5,10,8,5,10,7,5,-3,1,1
9490 REM Takt 16:
9500 DATA 17,28,8,1,24,8,17,26,8,1,28,8,17,28,8,1,23,8,
17,26,4,-2,1,1
9510 DATA 10,7,5,10,17,9,10,19,9,10,17,5,-3,1,1
9520 REM Takt 17:
9530 DATA 17,24,4,17,12,2,17,16,8,-2,1,1
9540 DATA 10,16,5,10,4,2,10,7,8,-3,1,1
9550 DATA -1,1,1
```

Abb. 5.6: Der Entertainer (Fortsetzung)

Das Programm spielt ein zweistimmiges Musikstück, das Ihnen bekannt vorkommen dürfte. Die erste Stimme ist die Melodiestimme. Sie übernimmt auch das Timing.

Die zweite Stimme, die von der ersten gesteuert wird, ist die „Baßstimme". Zu Beginn eines jeden Taktes wird die Baßstimme, die über Kanal B läuft, neu geladen. Man hat dann sozusagen die zweite Stimme auf Vorrat im Kanal B. Die einzelnen Töne der zweiten Stimme sind, soweit sie gleichzeitig mit einer Melodienote (einem Ton der ersten Stimme) erklingen sollen, mit Rendezvous-Anweisungen versehen.

Die erste Note in jedem Takt erhält grundsätzlich eine Rendezvous-Anweisung, um zu verhindern, daß der Rechner schon zu spielen anfängt, bevor der ganze Takt geladen ist.

Ein Kanal faßt maximal vier SOUND-Befehle hintereinander. Das bedeutet für unser Programm, daß pro Takt nur vier Töne der Begleitstimme gespielt werden können. Außerdem ist darauf zu achten, daß im Haupt- und Begleittakt gleich viele Rendezvous-Anweisungen enthalten sind, da sonst ein heilloses Durcheinander entsteht.

Wenn nun der Takt für die Begleitstimme komplett ist, springt das Programm zu der Stelle, an der die Melodiestimme erzeugt wird. Es wird wieder der ganze Takt in den Kanal - diesmal ist es Kanal A - geladen.

Die Ausführung beginnt jedoch sofort, da die entsprechenden Rendezvous-Partner der Melodietöne in Kanal B schon bereitstehen. Der erste Ton eines jeden Taktes ruft gleichzeitig den ersten Ton der Begleitstimme auf.

Bis zu drei weitere Rendezvous werden nach Bedarf arrangiert. Die Daten für die einzelnen Noten sind in Variablenfeldern abgelegt. Zu jeder Note gehören drei Variablen, die den Kanalstatus (das ist die Kanalnummer, die bei Bedarf auch mit einer Rendezvous-Anweisung versehen sein kann), die Tonperiode und die Dauer enthalten.

Es sind die Variablen st(i), n(i), l(i) für die erste Stimme und die Variablen st2(j), n2(j), l2(j) für die zweite Stimme. Die Variablen beider Stimmen werden unabhängig voneinander indiziert, und zwar von der Variablen i für die erste Stimme und j für die zweite Stimme. Man kann sich die Variablenfelder für die Stimmen als parallelliegende Schienen vorstellen: Nachdem die Felder von den Einleseschleifen vollständig besetzt worden sind, beginnt das Programm bei der Schiene, die die zweite Stimme enthält - im weiteren Schiene 2 genannt -, die Töne in den Kanal B zu laden und zählt dabei die Töne in der Variablen j mit.

Da die Takte der Zweitstimme verschieden viele Töne enthalten können (maximal vier), muß das Programm entsprechende Informationen erhalten, wenn der Takt zu Ende ist. Das geschieht durch die Variable, die den Kanalstatus beinhaltet. In der Ladeschleife für Kanal B wird geprüft, ob st2(j) den Wert −3 (der sowieso als Kanalstatus nicht in Frage kommt) enthält. Ist dies der Fall, dann weiß das Programm, daß der Takt hier zu Ende ist, und verzweigt in die Ladeschleife (oder „Spielschleife", da die Töne hier sofort gespielt werden) für Kanal A.

Das Programm wechselt also zur Schiene 1 und zählt die Töne dieser Schiene mit der Variablen j durch. Auch hier enthält die Variable, die den Kanalstatus angibt (st(i)), die Information, wann der Takt der ersten Stimme zu Ende ist. st(i) enthält dann den Wert −2. Das Programm wechselt wieder nach Schiene 2 und fährt an der Stelle, an der es vorher aufgehört hat, fort, die Töne der zweiten Stimme in den Kanal B zu laden. Inzwischen werden die restlichen vier Töne, die sich im Kanal A befinden, abgespielt, so daß keine Pause entsteht.

Wir wollen die Programmzeilen im einzelnen erläutern:

Zeile 1130 -1140: Dimensionierung der Notenfelder. In Zeile 1130 sind die Variablen für die erste Stimme dimensioniert. st( ) enthält den Kanalstatus, n( ) die Tonperiode und l( ) die Dauer der Note. In Zeile 1140 werden analog die Variablen für die zweite Stimme dimensioniert. Die Variablen bieten genug Platz für Ihre eigenen Musikstücke, die Sie statt des „Entertainers" in die DATA-Zeilen setzen können.

Zeile 2040 -2050: Die Schleifenvariablen, mit denen die Variablen der Notenfelder indiziert werden, werden vor Beginn der Einleseschleife auf 0 gesetzt.

Zeile 2090 -2170: Einleseschleifen für die erste Stimme. Es werden immer drei Werte - der Kanalstatus, die Nummer der Note und der Notenwert - gleichzeitig gelesen. Die Noten sind, entsprechend der zu Beginn des Kapitels wiedergegebenen Tabelle, numeriert. Aus der Notennummer wird mit der Formel in Zeile 2110 die Tonperiode berechnet und der Variablen n(i) zugeordnet.

Die Unterscheidung zwischen 99 und anderen Notennummern dient dazu, im Text Pausen zu setzen. Ist die Notennummer gleich 99, dann handelt es sich um eine Pause, und als Tonperiode wird der Variablen n(i) der Wert 0 übergeben. Die Länge der Pause richtet sich nach der in l(i) angegebenen Tondauer. Die

Pausen sind für das Timing der Zweitstimme wichtig. Sie führen immer eine Rendezvous-Anweisung mit.

In Zeile 2120 wird dann aus dem dritten gelesenen Wert die Dauer der Note berechnet. Als Notenwert ist auf den DATA-Zeilen immer der Nenner des Bruchs, also bei einer halben Note die 2, bei einer Viertelnote die 4 usw. angegeben.

Die Dauer der Note wird in Zeile 2120 ausgerechnet und der Variablen l(i) zugewiesen. Hier kann man die Geschwindigkeit, mit der das Musikstück gespielt werden soll, ändern, indem man statt $^{200}/_{100}$ Sekunden als Dauer für eine ganze Note einen anderen Wert einsetzt.

In Zeile 2130 werden die Nummern des Tons, der Kanalstatus, die errechnete Tonperiode und die errechnete Dauer ausgedruckt. Dadurch hat man eine gewisse Kontrolle, ob die DATA-Zeilen stimmen. Wenn das Programm läuft, kann man diese Zeile wieder löschen. Die Werte für die Zweitstimme werden dann (wenn man im 80-Zeichen-Modus arbeitet) eingerückt.

Wenn die Variablen besetzt und deren Inhalte angezeigt worden sind, wird die Zählvariable i um 1 erhöht. Danach wird der eingelesene Kanalstatus überprüft, ob vielleicht der Takt zu Ende ist (-2) oder das Musikstück. Enthält die Variable -1, dann ist das Musikstück zu Ende, und der Einlesevorgang wird abgebrochen. Das Programm verzweigt zu dem Takt, der das Spielen der Töne übernimmt.

Zeile 2210 -2280: Die Zählvariablen werden wieder zurückgesetzt, damit die Tonausführung beginnen kann.

Zeile 3100 -3140: Die zweite Stimme wird in den Kanal B geladen. Zuerst wird die Zählvariable um 1 erhöht und deutet damit auf die nächste zu spielende Note hin. Danach wird, bevor der SOUND-Befehl ausgeführt wird, überprüft, ob st2(j) die Information für „Taktende" enthält. Dieser Wert darf nämlich den SOUND-Befehl nicht erreichen, sonst kommt es zu einer Fehlermeldung, abgesehen davon, daß dann das ganze Arrangement durcheinanderkommt.

Außerdem wird überprüft, ob im nächsten Haupttakt das Ende des Musikstücks angekündigt wird (Zeile 3110). In diesem Fall

enthalten die Variablen für die zweite Stimme nämlich keine Werte mehr, und das Programm muß sofort beendet werden.

Zeile 3190 -3220: Nachdem ein Takt von der Zweitstimme in Kanal B geladen worden ist, verzweigt das Programm durch die Abfrage in Zeile 3120 hierher. Es werden die Töne des zugehörigen Taktes der Hauptstimme in den Tonkanal geschickt, wodurch gleichzeitig das Abspielen der Begleitstimme beginnt. Wenn das Taktende erreicht ist, wird zum Laden der Begleitstimme verzweigt.

Zeile 4020 -4030: Wenn in Zeile 3110 festgestellt wurde, daß das Stück zu Ende ist, dann verzweigt das Programm hierher. Nach Drücken der ENTER-Taste wird das Stück von neuem gespielt.

Zeile 9000 -9540: In den DATA-Zeilen sind die Noten enthalten. Eine Dreiergruppe von Zahlen ergibt immer eine Note oder eine Information für Taktende.

In der ersten Zeile steht immer die Hauptstimme, in der zweiten Zeile die Begleitstimme.

Zeile 9550: Information für „Musikende". Mit Rücksicht auf das Urheberrecht möchten wir das Beispiel hier nicht weiter ausführen.

Anmerkung: Sicher haben Sie sich gewundert, daß die Begleitstimme immer aus Fünftel- statt Viertelnoten und Neuntel- statt Achtelnoten besteht. Damit soll verhindert werden, daß bei geringen Timing-Fehlern gleich das ganze Programm durcheinanderkommt. Zwischen den beiden Stimmen entsteht auf diese Weise ein Sicherheitsabstand.

Wenn Sie in den DATA-Zeilen für die Begleitstimme vor −3 noch eine Note einfügen, etwa 10, 17, 5, dann können Sie leicht feststellen was passiert, wenn man einem Kanal zuviel auf einmal zumutet. Vor solchen Überraschungen kann man sich allerdings schützen, indem man abfragt, wie viele Töne im Kanal Schlange stehen.

Die SQ-Funktion

Die SQ-Funktion liefert Ihnen eine vollständige Information über den Zustand eines Kanals. Die Anzahl der freien Plätze im Kanal erhält man mit

```
PRINT (SQ(x) AND 7)
```

Für x setzt man die Kanalnummer (1, 2 oder 4) ein. Wenn man statt

```
SOUND x,Periode
```

schreibt

```
IF (SQ(x) AND 7) THEN SOUND x,Periode
```

kann es nicht vorkommen, daß der Kanal x überlastet wird und der Rechner sich „aufhängt". Bei der Programmierung von Musikstücken sollte man allerdings auf solche Tricks verzichten. Da muß es auch so funktionieren. Schließlich kann man bei einem Musikstück nicht einfach irgendwelche Töne weglassen, wenn es einem zu lange dauert.

Die SQ-Funktion liefert noch andere Informationen, z. B. darüber, ob der Kanal durch ein Rendezvous blockiert ist und mit welchem Kanal das Rendezvous stattfinden soll, oder ob der Kanal durch ein Hold blockiert ist.

Der Ausdruck

```
(SQ(x) AND n)
```

liefert einen Wert, der verschieden von Null ist, wenn n in der Information über den Kanal x, die SQ(x) liefert, enthalten ist.

n ist dabei eine der Zahlen aus der Tabelle auf Seite 6 Kapitel 6, im Handbuch, die für verschiedene Anweisungen wie Rendezvous oder Hold stehen. In diesem Zusammenhang möchten wir noch etwas zum Befehl

```
ON SQ(x) GOSUB Zeilennummer
```

sagen. Die Syntax dieses Befehls kann leicht zu der irrtümlichen Annahme verleiten, er sei wie die ON ERROR GOTO-Anweisung zu verwenden, nämlich einmal am Programmanfang. Im Gegensatz zum ON ERROR GOTO-Befehl wird hier aber der Sprung nach „Zeilennummer" nur ausgeführt, wenn das Programm auf den ON SQ(x) GOSUB-Befehl stößt und im Kanal mit der Nummer x Platz für Töne frei ist.

Durch einen Programmiertrick kann man erreichen, daß der Befehl nur einmal im Programm stehen muß, indem dieser in regelmäßigen Abständen durch Interruptbefehle wie EVERY angesprungen wird.

Der HOLD-Befehl

Ein weiterer Begriff, der in diesem Zusammenhang im Handbuch auftaucht, ist der oben schon erwähnte HOLD-Befehl. Ein Hold ist etwas ähnliches wie ein Rendezvous. Hold hat in der Tabelle die Nummer 64. SOUND-Befehle, die mit einem Hold versehen sind, werden in der Warteschlange aufgehalten. Der Code 64 ist für alle drei Kanäle gleich. Der Unterschied zum Rendezvous besteht darin, daß der Ton dann nicht von einem anderen SOUND-Befehl ausgelöst wird, sondern durch den RELEASE-Befehl.

Bei RELEASE wird eine Kanal-Nummer angegeben. Ist ein Kanal durch einen SOUND-Befehl mit einem Hold blockiert, kann er nur wieder mit dem RELEASE-Befehl, der die richtige Kanalnummer enthält, in Gang gesetzt werden. Zum Beispiel:

```
SOUND 1+64,100
```

wird erst ausgeführt, wenn man

```
RELEASE (1)
```

eingibt. Zu beachten ist dabei, daß jeder Tonkanal nur vier Befehle gleichzeitig faßt. Bei jedem weiteren Befehl wartet das BASIC darauf, daß wieder Platz in der Schlange ist.

Nehmen wir an, es soll ein sechster Befehl in die Schlange geschickt werden. Der Rechner bleibt so lange im Programm beim SOUND-Befehl stehen, bis in der Schlange wieder ein Platz frei ist. Ist aber die Schlange durch ein Hold (oder ein Rendezvous) blockiert, so stoppt das Programm. Es ist vollständig lahmgelegt, da jetzt natürlich kein RELEASE-Befehl ausgeführt werden kann.

Der FLUSH-Befehl

Die letzte Anweisung, die im Kanalstatus verpackt werden kann und die wir hier beschreiben möchten, ist die FLUSH-Anweisung.

Wenn ein SOUND-Befehl mit einer FLUSH-Anweisung ausgeführt wird, werden sofort alle noch im angesprochenen Kanal stehenden Töne gelöscht (!), der gerade gespielte Ton wird abgebrochen, und der im SOUND-Befehl angegebene Ton wird gespielt.

SOUND-EDITOR

Zum Schluß des Kapitels soll noch ein Programm vorgestellt werden, das zum Experimentieren mit dem Sound verleiten soll.

Mit dem Programm „Soundmixer" können Sie alle Möglichkeiten, den Sound zu beeinflussen, die das Schneider-BASIC bietet, ausprobieren und die Sound-Parameter für eigene Programme festlegen.

Bedienungsanleitung

Auf dem Bildschirm werden für jeden Tongenerator alle aktuellen Parameter wie Frequenz, Tondauer usw. angezeigt. Die Parameter für die Tonkurve

(ENT) und die Lautstärkekurve (ENV) sind in Grafiken dargestellt. Mit den Tasten 1, 2 und 3 werden die einzelnen Generatoren in Gang gesetzt, d. h. der eingestellte Ton erklingt für die angegebene Dauer.

Durch Drücken mehrerer Tasten gleichzeitig kann man auch zwei oder alle drei Generatoren zugleich spielen lassen. Um die einzelnen Parameter zu verändern, wählt man ein bestimmtes Stichwort wie z. B. Freq.: mit den Cursortasten an, was dann invers (dunkel auf hellem Grund) angezeigt wird.

Die Tasten für links und rechts wählen den Generator, mit den Tasten für auf und ab wählt man zwischen ENV, ENT, Freq., Dauer, Noise. Wenn die Bezeichnung des Parameters, den man ändern will, invers erscheint, drückt man die ENTER-Taste, worauf in den untersten beiden Zeilen des Bildschirms das Menü verschwindet und der Rechner nach den neuen Werten fragt. Zu diesen Werten wollen wir hier einige Erläuterungen geben:

Freq. (Frequenz) — Hier gibt man einfach die gewünschte Frequenz in Hertz ein. Die voreingestellten Werte ergeben jeweils den Ton a in verschiedenen Oktaven.

Dauer — Die Tondauer wird zwar in Millisekunden (ms) angezeigt, muß aber, wie im SOUND-Befehl, in Hundertstelsekunden eingegeben werden.

Bei der Eingabe von 0 oder einer negativen Zahl richtet sich die Tondauer nach der Tonhüllkurve. Ist die eingegebene Zahl kleiner als -1 (also -2, -3 usw.), wird der Ton so oft wiederholt, wie es der Betrag der Zahl angibt.

Noise — Dem Ton kann ein Rauschen beigemischt werden, um z. B. Motoren oder Wind nachzuahmen. 0 bedeutet kein Rauschen, 1 ist ein helles Rauschen und 15 ein tiefes Rauschen.

Werte sind zwischen 0 und 15 möglich.

ENV — Bei ENV (ENvelope Volume) und ENT (ENvelope Tone) werden die auf dem Bildschirm gezeigten Kurven modifiziert. Zuerst fragt der Rechner, wie viele Abschnitte Sie definieren wollen. Jedes gerade Stück der Kurve entspricht einem Abschnitt. In der angezeigten Grundeinstellung (übrigens die gleiche wie im Kapitel 6 im Handbuch) sind es z. B. drei.

Danach wird für jeden Abschnitt die Schrittzahl (Stufenzahl), die Lautstärke, die nach Ausführen des Schrittes erreicht werden soll, und die Pausenlänge erfragt.

Die Pausenlänge ist die Zeit, in der die Lautstärke auf einem bestimmten Wert bleibt, der sich durch Multiplikation der Nummer des gerade auszuführenden Schrittes mit der Schrittgröße ergibt, und in Zehntelsekunden angegeben wird.

Die Gesamtdauer des Tons erhält man, wenn man die Anzahl aller Schritte mit der jeweiligen Pausenlänge multipliziert und die Ergebnisse addiert. Die Schrittgröße wird automatisch ermittelt.

Zum Schluß muß man noch die Gesamtlautstärke eingeben. Wenn alle Werte festgelegt sind, wird der fertige SOUND-Befehl, so wie er im Programm stehen müßte, angezeigt. Mit ENTER kommt man wieder zurück ins Hauptmenü.

ENT

Die Eingabe erfolgt analog der Eingabe beim ENT-Befehl. Die Kurve gibt die Abweichung der tatsächlich gespielten Periode (125000/Freq.) zu der im SOUND-Befehl angegebenen Periode an.

Die Abweichung kann beliebig groß gewählt werden. Beim Überschreiten der Grenzen des zulässigen Wertebereichs wird automatisch (vom BASIC) die gespielte Tonperiode um 4096 wieder in den zulässigen Wertebereich hineinverschoben.

Mit diesen Befehlen sind, bis auf Haltepunkte und Rendezvous, alle Möglichkeiten, die das Schneider-BASIC bietet, erschöpft.

Abb. 5.7: Soundmixer

```
1000 REM ------------------------------
1010 REM ---        SOUNDMIXER        ---
1020 REM ------------------------------
1030 :
1040 DIM y$(5)
1050 DIM xv(3,5),yv(3,5),y(5)
1060 DIM xt(3,5),yt(3,5)
1070 DIM p(5),q(5),r(5)
1080 DIM t(5),v(5),w(5)
1090 DIM vol(5),diff(5)
1100 :
1110 REM ---      Voreinstellung      --
1120 :
```

```
1130 REM --- Huellkurven einstellen ---
1140 :
1150 FOR i=1 TO 3
1160     xv(i,1)=5 :yv(i,1)=15
1170     xv(i,2)=21:yv(i,2)=15
1180     xv(i,3)=31:yv(i,3)=0
1190     xv(i,4)=31:yv(i,4)=0
1200     xv(i,5)=31:yv(i,5)=0
1210     xt(i,1)=5 :yt(i,1)=5
1220     xt(i,2)=15:yt(i,2)=-5
1230     xt(i,3)=25:yt(i,3)=5
1240     xt(i,4)=30:yt(i,4)=0
1250     xt(i,5)=30:yt(i,5)=0
1260 NEXT
1270 :
1280 REM -Variablen  f. Menuesteuerung-
1290 :
1300 y(1)=7
1310 y$(1)="ENV:"
1320 y(2)=14
1330 y$(2)="ENT:"
1340 y(3)=21
1350 y$(3)="Freq.:"
1360 y(4)=22
1370 y$(4)="Dauer:"
1380 y(5)=23
1390 y$(5)="Noise:"
1400 :
1410 REM ------------------------------
1420 REM ---        Maske aufbauen     -
1430 REM ------------------------------
1440 :
1450 REM ---          Farben          ---
1460 :
1470 MODE 2:INK 0,1:INK 1,26
1480 PEN 0:PAPER 1:BORDER 1
1490 :
1500 PRINT "                            S  O  U  N  D  M
I  X  E  R                              "
1510 PEN 1:PAPER 0
1520 PRINT "       L i n k e r    K a n a l
   R e c h t e r    K a n a l       "
1530 PRINT "         Generator 1                  Generato
r 2                 Generator 3"
1540 PRINT:PRINT " ENV:                         ENV:
                   ENV:"
1550 LOCATE 1,14
1560 PRINT " ENT:                               ENT:
             ENT:"
1570 LOCATE 1,21
1580 PRINT " Freq.:  440 Hz                     Freq.:  880 Hz
             Freq.:  1760 Hz"
1590 PRINT " Dauer:  310 ms                     Dauer:  310 ms
             Dauer:  310 ms"
1600 PRINT " Noise:  0                          Noise:  0
             Noise:  0
1610 :
1620 REM ---   Koordinatensysteme   ---
1630 :
1640 MOVE 320,384:DRAW 320,344
1650 MOVE 0,344:DRAW 640,344
1660 MOVE 427,344:DRAW 427,0
1670 MOVE 213,344:DRAW 213,0
1680 MOVE 0,312:DRAW 640,312
1690 :
1700 WINDOW #1,1,26,8,13
1710 WINDOW #2,28,53,8,13
1720 WINDOW #3,55,80,8,13
1730 WINDOW #4,1,26,15,20
1740 WINDOW #5,28,53,15,20
1750 WINDOW #6,55,80,15,20
1760 WINDOW #7,1,80,24,25
1770 :
1780 REM ---   Farben fuer Windows  ---
1790 :
1800 PEN #7,0:PAPER #7,1
```

Abb. 5.7: Soundmixer (Fortsetzung)

```
1810 :
1820 FOR i=1 TO 6
1830    PEN #i,1:PAPER #i,0
1840 NEXT
1850 :
1860 REM --- aktuellen Bef. invert. ---
1870 :
1880 GOSUB 8360
1890 LOCATE 2,y(3)
1900 PRINT y$(3)
1910 LOCATE 9,5
1920 PRINT"Generator 1"
1930 GOSUB 8410
1940 :
1950 REM ------------------------------
1960 REM ---  Voreinstellung Sound  ---
1970 REM ------------------------------
1980 :
1990 FOR generator=1 TO 3
2000    GOSUB 8040
2010    diff=8:GOSUB 8200
2020    ENV generator,5,3,1,1,0,16,5,-3,2
2030    ENT generator,5,1,1,10,-1,1,10,1,1,5,-1,1
2040    freq(generator)=220*2^generator
2050    laut(generator)=12
2060    dauer(generator)=31
2070    krach(generator)=0
2080 NEXT generator
2090 :
2100 generator=1
2110 befehl=3
2120 GOSUB 8450:REM Menue anzeigen
2130 :
2140 REM ---   Vorprg. ausgefuehrt  ---
2150 :
2160 SOUND 1,125000/freq(1)
2170 SOUND 1,125000/freq(2)
2180 SOUND 1,125000/freq(3)
2190 :
3000 REM --------------------------
3010 REM ---   Hauptprogramm    ---
3020 REM --------------------------
3030 :
3040 REM ---    Tastaturabfrage     ---
3050 :
3060 i$=INKEY$
3070 IF i$="" THEN 3060
3080 IF i$="1" THEN SOUND 1,125000/freq(1),dauer(1),lau
t(1),1,1,krach(1)
3090 IF i$="2" THEN SOUND 2,125000/freq(2),dauer(2),lau
t(2),2,2,krach(2)
3100 IF i$="3" THEN SOUND 4,125000/freq(3),dauer(3),lau
t(3),3,3,krach(3)
3110 :
3120 IF i$=CHR$(13) THEN 3210                : REM ente
r
3130 IF i$=CHR$(242) THEN 3230               : REM curs
or links
3140 IF i$=CHR$(243) THEN 3390               : REM curs
or rechts
3150 IF i$=CHR$(240) THEN 3550               : REM curs
or auf
3160 IF i$=CHR$(241) THEN 3670               : REM curs
or ab
3170 GOTO 3060
3180 :
3190 REM ---    Befehl aufrufen     ---
3200 :
3210 ON befehl GOTO 4270,4830,3790,3950,4110
3220 :
3230 REM ---      Cursor Links      ---
3240 :
3250 IF generator=1 THEN 3060
3260 LOCATE 9+(generator-1)*27,5
3270 PRINT"Generator";generator
3280 LOCATE 2+(generator-1)*27,y(befehl)
```

Abb. 5.7: Soundmixer (Fortsetzung)

```
3290 PRINT y$(befehl)
3300 generator=generator-1
3310 LOCATE 9+(generator-1)*27,5
3320 GOSUB 8360
3330 PRINT"Generator";generator
3340 LOCATE 2+(generator-1)*27,y(befehl)
3350 PRINT y$(befehl)
3360 GOSUB 8410
3370 GOTO 3060
3380 :
3390 REM ---       Cursor rechts      ---
3400 :
3410 IF generator=3 THEN 3060
3420 LOCATE 9+(generator-1)*27,5
3430 PRINT"Generator";generator
3440 LOCATE 2+(generator-1)*27,y(befehl)
3450 PRINT y$(befehl)
3460 generator=generator+1
3470 LOCATE 9+(generator-1)*27,5
3480 GOSUB 8360
3490 PRINT"Generator";generator
3500 LOCATE 2+(generator-1)*27,y(befehl)
3510 PRINT y$(befehl)
3520 GOSUB 8410
3530 GOTO 3060
3540 :
3550 REM ---         Cursor auf       ---
3560 :
3570 IF befehl=1 THEN 3060
3580 LOCATE 2+(generator-1)*27,y(befehl)
3590 PRINT y$(befehl)
3600 befehl=befehl-1
3610 LOCATE 2+(generator-1)*27,y(befehl)
3620 GOSUB 8360
3630 PRINT y$(befehl)
3640 GOSUB 8410
3650 GOTO 3060
3660 :
3670 REM ---          Cursor ab       ---
3680 :
3690 IF befehl=5 THEN 3060
3700 LOCATE 2+(generator-1)*27,y(befehl)
3710 PRINT y$(befehl)
3720 befehl=befehl+1
3730 LOCATE 2+(generator-1)*27,y(befehl)
3740 GOSUB 8360
3750 PRINT y$(befehl)
3760 GOSUB 8410
3770 GOTO 3060
3780 :
3790 REM ------------------------------
3800 REM ---  Frequenz einstellen   ---
3810 REM ------------------------------
3820 :
3830 CLS #7
3840 :
3850 PRINT #7,"Frequenz fuer Generator";generator;
3860 INPUT #7,freq(generator)
3870 IF freq(generator)<31 OR freq(generator)>125000 TH
EN 3830
3880 LOCATE 10+(generator-1)*27,y(3)
3890 PRINT"          "
3900 LOCATE 10+(generator-1)*27,y(3)
3910 PRINT freq(generator);"Hz"
3920 GOSUB 8450
3930 GOTO 3060
3940 :
3950 REM ------------------------------
3960 REM ---  Tondauer einstellen   ---
3970 REM ------------------------------
3980 :
3990 CLS #7
4000 :
4010 PRINT #7,"Tondauer fuer Generator";generator;
4020 INPUT #7,dauer(generator)
4030 IF dauer(generator)<-32768 OR dauer(generator)>327
```

Abb. 5.7: Soundmixer (Fortsetzung)

```
67 THEN 3990
4040 LOCATE 10+(generator-1)*27,y(4)
4050 PRINT"          "
4060 LOCATE 10+(generator-1)*27,y(4)
4070 IF dauer(generator)>0 THEN PRINT 10*dauer(generato
r);"ms" ELSE PRINT "repeat"
4080 GOSUB 8450
4090 GOTO 3060
4100 :
4110 REM ------------------------------
4120 REM ---  Geraeusch einstellen  ---
4130 REM ------------------------------
4140 :
4150 CLS #7
4160 :
4170 PRINT #7,"Geraeuschperiode fuer Generator";generat
or;
4180 INPUT #7,krach(generator)
4190 IF krach(generator)<0 OR krach(generator)>15 THEN
4150
4200 LOCATE 10+(generator-1)*27,y(5)
4210 PRINT"          "
4220 LOCATE 10+(generator-1)*27,y(5)
4230 PRINT krach(generator)
4240 GOSUB 8450
4250 GOTO 3060
4260 :
4270 REM ------------------------------
4280 REM ---  ENV-Kurve festlegen   ---
4290 REM ------------------------------
4300 :
4310 CLS #7
4320 :
4330 INPUT #7,"Wieviele Abschnitte moechten sie eingebe
n ";anzahl
4340 IF anzahl<1 OR anzahl>5 THEN 4330
4350 FOR i=1 TO anzahl
4360    PRINT #7,"Abschnitt";i;": Schrittzahl";
4370    INPUT #7,p(i)
4380    IF p(i)<0 OR p(i)>127 THEN 4360
4390    PRINT #7,"            bis zu welcher Lautstaerk
e ";
4400    INPUT #7,vol(i)
4410    IF vol(i)<0 OR vol(i)>15 THEN 4390
4420    q(i)=(vol(i)-vol(i-1))/p(i)
4430    PRINT #7,"            Schrittgroesse =";INT(q(i
))
4440    PRINT #7,"            Pausenlaenge ";
4450    INPUT #7,r(i)
4460    IF r(i)<0 OR r(i)>127 THEN 4440
4470    xv(generator,i)=xv(generator,i-1)+p(i)*r(i)
4480    yv(generator,i)=vol(i)
4490 NEXT
4500 :
4510 FOR i=anzahl+1 TO 5
4520    p(i)=0
4530    q(i)=0
4540    r(i)=0
4550    xv(generator,i)=xv(generator,anzahl)
4560    yv(generator,i)=yv(generator,anzahl)
4570 NEXT
4580 :
4590 INPUT #7,"Gesamtlautstaerke ";laut(generator)
4600 IF laut(generator)<0 OR laut(generator)>15 THEN 45
90
4610 GOSUB 8040
4620 ENV generator,p(1),q(1),r(1),p(2),q(2),r(2),p(3),q
(3),r(3),p(4),q(4),r(4),p(5),q(5),r(5)
4630 :
4640 CLS #7
4650 :
4660 PRINT #7,"ENV";generator;
4670 :
4680 FOR i=1 TO anzahl
4690    PRINT #7,",";MID$(STR$(p(i)),2,LEN(STR$(p(i)))-
1);
```

Abb. 5.7: Soundmixer (Fortsetzung)

```
4700    PRINT #7,",";MID$(STR$(q(i)),2,LEN(STR$(q(i)))-
1);
4710    PRINT #7,",";MID$(STR$(r(i)),2,LEN(STR$(r(i)))-
1);
4720 NEXT
4730 :
4740 PRINT #7
4750 GOSUB 8360
4760 PRINT #7," ENTER "
4770 GOSUB 8410
4780 WHILE INKEY$="":WEND
4790 GOSUB 8450
4800 a$=INKEY$:REM   Tastendruck abfangen
4810 GOTO 3060
4820 :
4830 REM ------------------------------
4840 REM ---  ENT-Kurve festlegen   ---
4850 REM ------------------------------
4860 :
4870 CLS #7
4880 :
4890 INPUT #7,"Wieviele Abschnitte moechten sie eingebe
n ";anzahl
4900 IF anzahl<1 OR anzahl>5 THEN 4890
4910 :
4920 FOR i=1 TO anzahl
4930    PRINT #7,"Abschnitt";i;": Schrittzahl";
4940    INPUT #7,t(i)
4950    IF t(i)<0 OR t(i)>239 THEN 4930
4960    PRINT #7,"            bis zu welcher Abweichung
 ";
4970    INPUT #7,diff(i)
4980    IF ABS(diff(i))>diff THEN diff=ABS(diff(i))
4990    v(i)=INT(diff(i)-diff(i-1))/t(i)
5000    PRINT #7,"            Schrittgroesse =";v(i)
5010    PRINT #7,"            Pausenlaenge ";
5020    INPUT #7,w(i)
5030    IF w(i)<0 OR w(i)>255 THEN 5010
5040    xt(generator,i)=xt(generator,i-1)+t(i)*w(i)
5050    yt(generator,i)=diff(i)
5060 NEXT
5070 :
5080 FOR i=anzahl+1 TO 5
5090    t(i)=0
5100    v(i)=0
5110    w(i)=0
5120    xt(generator,i)=xt(generator,anzahl)
5130    yt(generator,i)=yt(generator,anzahl)
5140 NEXT
5150 :
5160 GOSUB 8200
5170 ENT generator,t(1),v(1),w(1),t(2),v(2),w(2),t(3),v
(3),w(3),t(4),v(4),w(4),t(5),v(5),w(5)
5180 :
5190 CLS #7
5200 :
5210 PRINT #7,"ENT";generator;
5220 :
5230 FOR i=1 TO anzahl
5240    PRINT #7,",";MID$(STR$(t(i)),2,LEN(STR$(t(i)))-
1);
5250    PRINT #7,",";MID$(STR$(v(i)),2,LEN(STR$(v(i)))-
1);
5260    PRINT #7,",";MID$(STR$(w(i)),2,LEN(STR$(w(i)))-
1);
5270 NEXT
5280 :
5290 PRINT #7
5300 GOSUB 8360
5310 PRINT #7," ENTER "
5320 GOSUB 8410
5330 WHILE INKEY$="":WEND
5340 GOSUB 8450
5350 a$=INKEY$:REM   Tastendruck abfangen
5360 GOTO 3060
5380 :
```

Abb. 5.7: Soundmixer (Fortsetzung)

```
8000 REM --------------------------
8010 REM ---   Unterprogramme   ---
8020 REM --------------------------
8030 :
8040 REM --------------------------
8050 REM --- ENV-Kurve zeichnen  ---
8060 REM --------------------------
8070    :
8080    CLS #generator
8090    :
8100    ORIGIN 10+213*(generator-1),194
8110    MOVE 0,90:DRAW 0,0:DRAW 190,0
8120    MOVE 0,0
8130    :
8140    FOR ii=1 TO 5
8150       DRAW xv(generator,ii)/xv(generator,5)*185,yv
(generator,ii)/15*85
8160    NEXT ii
8170    :
8180 RETURN
8190    :
8200 REM --------------------------
8210 REM --- ENT-Kurve zeichnen  ---
8220 REM --------------------------
8230    :
8240    CLS #generator+3
8250    :
8260    ORIGIN 10+213*(generator-1),130
8270    MOVE 0,45:DRAW 0,-45:MOVE 0,0:DRAW 190,0
8280    MOVE 0,0
8290    :
8300    FOR ii=1 TO 5
8310       DRAW xt(generator,ii)/xt(generator,5)*185,yt
(generator,ii)/diff*40
8320    NEXT ii
8330    :
8340 RETURN
8350    :
8360 REM --- inverse Darstellung ---
8370    :
8380    PAPER 1:PEN 0
8390 RETURN
8400    :
8410 REM --- normale Darstellung ---
8420    :
8430    PAPER 0:PEN 1
8440 RETURN
8450 REM ---   Menue anzeigen    ---
8460    :
8470    CLS#7
8480    :
8490    PRINT #7,"Menue:                          ";CHR$(242)
;"  /  ";CHR$(240);"  /  ";CHR$(243);"  /  ";CHR$(241);
"  /  ENTER"
8500    PRINT #7,"Ton spielen :  1 = Gen. 1  /  2 = Gen
. 2  /  3 = Gen. 3"
8510 RETURN
```

Abb. 5.7: Soundmixer (Fortsetzung)

Programmbeschreibung

Auch hier wollen wir die Programmbeschreibung wieder nach Zeilenbereichen gliedern.

Zeile 1000 - 1090: Dimensionieren der Felder für die Sound-Parameter und die Koordinaten für die ENV- und die ENT-Kurve.

Zeile 1150 - 1260: Das Vorbesetzen der Koordinaten für die ENV- und die ENT-Kurve wird in einer Schleife erledigt, da die Werte für alle Generatoren gleich sind. xv und yv sind jeweils die Koordinaten für die Eckpunkte der ENV-Kurve, xt und yt die Koordinaten für die ENT-Kurve. Der erste Index (i) gibt den Generator an, der zweite die Nummer des Punktes.

Zeile 1300 - 1390: Vorbesetzen der Variablen für die Menüanzeige. Die im Programm aufrufbaren Funktionen erhalten die Nummern 1 bis 5. Die erste Variable gibt an, in welcher Zeile (auf dem Bildschirm) die Funktion steht, die zweite Variable, die Stringvariable, enthält den Namen der Funktion.

Diese Texte werden benötigt, um den Namen der Funktion, die mit dem nächsten ENTER aufgerufen wird, zu invertieren. Dazu werden Hintergrundfarbe (PAPER) und Schriftfarbe (PEN) vertauscht und der alte Text überschrieben.

Der Text, der vorher invertiert war, wird in der normalen Farbkombination überschrieben. Jetzt dürfte auch ersichtlich sein, wozu die entsprechenden Zeilenpositionen verwendet werden.

Zeile 1470 -1600: Hier wird der Bildschirmaufbau festgelegt. In Zeile 1470 werden der Bildschirmmodus (80 Zeichen) und die Farben festgelegt. Die Grafiken werden später eingefügt.

Zeile 1640 - 1680: Einfügen der Linien in die Bildschirmmaske.

Zeile 1700 - 1760: Um die grafischen Darstellungen der ENV- und der ENT-Kurve schnell löschen zu können, werden die Windows #1 bis #6 definiert, die genau über den Grafiken liegen. Wenn z. B. die ENV-Kurve von Generator 1 gelöscht werden soll, weil eine neue Kurve gezeichnet werden muß, wird einfach Window #1 mit CLS #1 gelöscht. Das ist eine sehr effektive Methode, Teile des Bildschirms zu löschen.

Zeile 1800 - 1840: Farbeinstellung für Windows. Für Window #7 (Eingabefenster) wird die inverse Farbeinstellung gewählt.

Zeile 1880 - 1930: Der voreingestellte Generator und die voreingestellte Funktion werden invertiert. Mit den GOSUB-Befehlen in den Zeilen 1880 und 1930 wird die Farbeinstellung verändert.

Zeile 1990 - 2080: Voreinstellung aller Sound-Parameter für die Generatoren 1 bis 3 und Zeichnen der zugehörigen ENV- und ENT-Kurven. Die GOSUB-Befehle in den Zeilen 2000 und 2010 springen zu den Unterprogrammen, die die Kurven zeichnen. Die Frequenzzuweisung ,,freq(generator) = 220*2^generator" bewirkt, daß Generator 1 den Ton a mit 440 Hertz zugewiesen bekommt und die anderen Generatoren den gleichen Ton jeweils eine Oktave höher.

Zeile 2100 - 2110: Voreinstellung: Es werden Generator 1 und Funktion 3 gewählt.

Zeile 2120: Anzeigen des Menüs im Eingabefenster. Der Programmteil ist als Unterprogramm ausgeführt, weil er nach jeder Eingabe aufgerufen werden muß.

Zeile 3060 - 3170: Hier beginnt das Hauptprogramm. Der erste Teil des Hauptprogramms übernimmt die Tastaturabfrage. In Zeile 3060 wird der Variablen i$ ein über die Tastatur eingegebenes Zeichen zugewiesen. Ist kein Zeichen eingegeben worden, geht die Tastaturabfrage nach Überprüfung in Zeile 3070 weiter.

In den Zeilen 3080 bis 3100 werden nach Drücken der Tasten 1, 2 oder 3 die entsprechenden Töne gespielt.

Zeile 3120 überprüft, ob die ENTER-Taste gedrückt wurde, und die Zeilen 3130 bis 3160 prüfen auf Cursortasten.

Zeile 3210: Wenn die ENTER-Taste gedrückt wurde, wird in Zeile 3120 hierher verzweigt. Der ON. . .GOTO-Befehl verzweigt, entsprechend dem Inhalt der Variablen befehl - sie enthält die Funktions-(Befehls-)Nummer, also eine der Zahlen von eins bis fünf -, weiter in die Programmteile, die die gewünschte Funktion ausführen.

Zeile 3230 - 3770: Diese vier Programmteile weisen den Variablen generator und befehl neue Werte zu, wenn eine Cursortaste gedrückt wurde, und invertieren den entsprechenden Text auf dem Bildschirm. Als Beispiel sei hier der Programmteil für ,,Cursor links" erklärt: Wenn die linke Cursortaste gedrückt wird, muß die Variable generator um 1 vermindert werden; d. h. es wird der Generator gewählt, der auf dem Bildschirm eine Spalte weiter links steht.

In Zeile 3250 wird deshalb erst einmal überprüft, ob nicht sowieso schon der Generator 1, der am linken Bildrand angezeigt wird, eingestellt ist. Wenn generator den kleinstmöglichen Wert, nämlich 1, enthält, wird wieder zur Tastaturabfrage zurückgesprungen.

Die Formel

9 + (generator – 1)*27

in Zeile 3260 gibt an, an welcher Position in Bildschirmzeile 5 der Text „Generator i" steht. Sie ergibt für die Generatoren 1 bis 3 die Werte 9, 36 und 53.

Dieser Text ist invers dargestellt und muß wieder in den normalen Farben geschrieben werden. Nachdem mit dem LOCATE-Befehl der Cursor in Position gebracht worden ist, wird mit dem PRINT-Befehl in der nächsten Zeile der alte Text überschrieben. Das gleiche geschieht mit dem Funktionsnamen, nur daß hier der Cursor in Abhängigkeit von der gerade eingestellten Funktion in verschiedenen Zeilen positioniert werden muß und der Text (die Funktionsbezeichnung) durch die indizierte Variable y$(i) bestimmt wird.

Wenn das geschehen ist, wird generator um 1 vermindert. Jetzt müssen die Änderungen auf dem Bildschirm angezeigt werden. Der Cursor wird also wieder mit LOCATE an die richtige Stelle gebracht, und dann werden „Generator i" und die Funktionsbezeichnung geschrieben. Zuvor werden jedoch in Zeile 3320, bzw. im Unterprogramm ab Zeile 8360 Hintergrund- und Schreibfarbe vertauscht.

In Zeile 3360 werden wieder die normalen Farben eingestellt, und dann springt das Programm zurück zur Tastaturabfrage.

Zeile 3790 - 3930: In den Zeilen 3790 bis 4250 befinden sich die Programmteile, die von dem ON. . .GOTO-Befehl in Zeile 3210 aufgerufen werden und die zur Verfügung stehende Funktion ausführen. Die Zeilen 3790 bis 3930 werden aufgerufen, wenn befehl den Wert 1 enthält, was bedeutet, daß die Frequenz neu eingestellt werden soll.

Als erstes wird mit „CLS #7" das Eingabefenster gelöscht. In Zeile 3860 erfolgt die Eingabe, nachdem noch einmal daran erinnert wurde, was eingegeben werden soll. Die Variable freq, die die Frequenz enthält, ist durch die Variable generator indiziert. So wird die neue Frequenz automatisch dem richtigen Generator zugeordnet.

Zeile 3870 überprüft, ob der eingegebene Wert überhaupt zulässig ist. Ist das der Fall, muß der neue Wert auf dem Bildschirm angezeigt werden. Da es aber sein kann, daß der neue Wert eine kür-

zere Zahl ergibt als der alte, der auf dem Bildschirm gerade angezeigt wird, muß die alte Bildschirmanzeige gelöscht werden. Mit Hilfe der Formel

10 + (generator – 1)*27

und der Variablen y(3) (die die Nummer der Zeile enthält, in der die Frequenz angezeigt wird) wird der Cursor an die richtige Stelle gebracht und der alte Text mit Leerstellen überschrieben.

Danach wird der Cursor noch einmal an die gleiche Stelle gebracht, und jetzt kann die neue Frequenz auf den Bildschirm geschrieben werden. Zum Schluß wird noch mit „GOSUB 8450" das Menü ins Eingabefenster gesetzt und mit „GOTO 3060" zur Tastaturabfrage zurückgesprungen.

Zeile 3950 - 4090: Eingabe der Tondauer; analog zur Frequenzeingabe.

Zeile 4110 - 4250: Eingabe der Geräuschperiode.

Zeile 4270 - 4810: ENV-Kurve festlegen. Die ENV-Kurve kann durch ein bis fünf Abschnitte bestimmt werden. In Zeile 4330 wird erfragt, wie viele Abschnitte definiert werden sollen, und in Zeile 4340 wird die Eingabe überprüft.

Die weitere Eingabe erfolgt in einer Schleife, da für jeden Abschnitt die gleichen Werte erfragt werden müssen. Im einzelnen müssen die Schrittzahlen für jeden Abschnitt eingegeben werden: mit welcher Lautstärke der Generator am Ende des Abschnittes spielen soll, wonach das Programm die Schrittgröße ermittelt und die Pausenlänge, das ist die Dauer eines jeden Schrittes in einem Abschnitt.

Nach den eingegebenen Werten werden die Eckpunktkoordinaten bestimmt, nach denen das Unterprogramm ab Zeile 8040 die ENV-Kurve zeichnen soll.

In der nächsten Schleife von Zeile 4510 bis 4570 werden die Variablen, die keine Eingaben erhalten, weil weniger als fünf Abschnitte definiert wurden, besetzt. Die Parameter für den ENV-Befehl werden auf Null gesetzt, und die übriggebliebenen Eckpunkte erhalten die Koordinaten des letzten definierten Eckpunktes, so daß sie alle mit dem letzten Punkt zusammenfallen und in der Grafik unsichtbar bleiben.

Nach Eingabe der Gesamtlautstärke in Zeile 4600 und Zeichnen der Kurve in Zeile 4610 wird der ENV-Befehl ausgeführt und so, wie er in einem Programm stehen müßte, angezeigt (Zeilen 4680 bis 4720). Der ENV-Befehl bleibt so lange im Eingabefenster stehen, bis eine Taste gedrückt wird.

Solange keine Taste gedrückt ist, wird die WHILE-Schleife in Zeile 4780 ausgeführt. Danach wird wieder das Menü im Eingabefenster angezeigt und zur Tastaturabfrage zurückgesprungen.

Zeile 4830 - 5360: Eingabe der ENT-Kurve; genau wie die Eingabe der ENV-Kurve.

Unterprogramme

Zeile 8040 -8180: Zeichnen der ENV-Kurve. Mit „CLS #generator" wird der Bildschirmbereich, der innerhalb des Fensters liegt, gelöscht, damit die neue ENV-Kurve dort gezeichnet werden kann. In Zeile 8100 wird der Koordinatensprung für den entsprechenden Generator festgelegt. Auf diese Weise benötigt man nur ein Programm für alle drei ENV-Kurven. In Zeile 8110 wird das Koordinatensystem eingezeichnet; danach wird der Zeichencursor wieder im Koordinatenursprung positioniert, um die Kurve zeichnen zu können.

Das Zeichnen der Kurve selbst wird in einer Schleife (Zeilen 8140–8160) ausgeführt, in welcher der Reihe nach alle fünf Punkte mit dem DRAW-Befehl angesteuert werden. In Zeile 8180 wird ins Hauptprogramm zurückgekehrt.

Zeile 8200 - 8340: ENT-Kurve zeichnen; analog zur ENV-Kurve, jedoch mit Berücksichtigung der maximalen Abweichung.

Zeile 8360 - 8440: Hintergrund und Schreibfarbe werden vertauscht, um eine inverse Darstellung zu erhalten.

Zeile 8450 - 8510: Wie Zeilen 8360 bis 8390, jedoch Umschaltung auf normale Darstellung.

Und nun viel Spaß beim Probieren!

Kapitel 6

Tips und Tricks

Im folgenden wollen wir Ihnen einige - kurzgefaßte - Hilfestellungen für den Umgang mit Ihrem CPC vermitteln. Die einzelnen Punkte sind alphabetisch nach Stichworten geordnet. Vielleicht finden Sie etwas, was Sie schon lange gesucht haben.

BASIC-Zeilen per Programm einfügen

Das Einfügen von BASIC-Zeilen während der Programmausführung hat bei manchen Problemen eine wichtige Funktion. Wenn z. B. ein Programm zur grafischen Darstellung von Funktionen läuft, so benötigt es verschiedene Funktionen. Diese unterschiedlichen Funktionen kann man fest einprogrammieren, als String durch eine (langsame) Routine interpretieren lassen oder, wenn ein Diskettenlaufwerk vorhanden ist, mit dem Befehl CHAIN MERGE einfügen. Dazu muß erst eine ASCII-Datei mit OPENOUT zum Beschreiben eröffnet werden. Auf diese Datei kann die gewünschte Zeile in ASCII-Form und unter Beachtung der Syntax mit dem Befehl

```
PRINT #9, "Zeichenkette der Zeile"
```

in die Datei geschrieben werden. Dann muß die Datei mit CLOSEOUT geschlossen werden. Führt man anschließend den Befehl CHAIN MERGE "Name der ASCII-Datei" durch, so wird der Inhalt der Datei in das BASIC-Programm eingefügt und das Programm von vorne gestartet.

Die Routine zum Erstellen der ASCII-Datei und der Befehl CHAIN MERGE dürfen deshalb nicht am Programmanfang stehen, da sonst eine Endlosschleife betreten wird, die immer wieder die gleiche Datei erstellt und einfügt und das Programm neu startet.

Beispiel: Die Zeile 20 mit dem Inhalt „Y = SIN(x)/x" soll eingefügt werden. Dazu wird wie folgt vorgegangen:

```
OPENOUT "INSERT.DAT"
A$ = "20 Y = SIN(x)/x"
PRINT #9,A$
CLOSEOUT
CHAIN MERGE "INSERT.DAT"
```

Auf diese Befehlssequenz startet das Programm neu. In der Zeile 20 steht dann Y = SIN(x)/x. Dieser Trick geht auch mit dem Kassettenlaufwerk, nur müßte dazu bei dem CHAIN MERGE-Befehl die Kassette zum Anfang der Datei zurückgespult werden.

Binärdaten; Länge und Anfang bestimmen
(Nur für Kassettendateien)

Manchmal ist es notwendig, mehr Informationen über eine Binärdatei zu haben als den Namen und die Länge in KB. Gehen Sie dann wie folgt vor:

1. Führen Sie zuerst einen CAT-Befehl für den ersten Block der Datei aus, deren Dateiparameter Sie bestimmen wollen.
2. Brechen Sie nach dem erfolgreichen Lesen der Datei (nach OK) den CAT-Befehl durch Drücken von ESC ab.
3. Nun sind die Dateiparameter in einem Puffer abgelegt und können ausgelesen werden.

Folgende Dateiparameter können an den angegebenen Speicherzellen ausgelesen werden:

Name der Datei: B88Ch–B89Bh (unbenutzte Bytes enthalten 0)
Dateianfang: B8A1h und B8A2h
Dateilänge: B8A4h und B8A5h
Startadresse: B8A6h und B8A7h

Der Anfang der Datei ergibt sich aus

```
PEEK(&B8A1) + PEEK(&B8A2)*256
```

Die Länge ergibt sich aus

```
PEEK(&B8A4) + PEEK(&B8A5)*256
```

und die eventuell vom Anfang verschiedene Startadresse für das Maschinenprogramm aus

```
PEEK(&B8A6) + PEEK(&B8A7)*256
```

Zum Laden der Binärdatei muß nun der BASIC-Speicher auf

```
MEMORY ANFANG-1
```

beschränkt werden. Dann kann das Band zurückgespult und die Datei mit LOAD“ “ geladen werden. Zum erneuten Abspeichern der Datei ist dann

```
SAVE "Name der Datei", B, Anfang, Länge, Startadresse
```

als Anweisung notwendig.

Somit lassen sich Dateien duplizieren, deren Adressen man vergessen oder verlegt hat, ohne daß man sein Monitorprogramm zur Suche bemühen muß.

Deutscher Zeichensatz (weitere Möglichkeit)

Dank der einfachen Möglichkeit, die Zeichen des CPC abzuändern und die Tastatur umzubelegen, kann man aus der US-Tastatur eine der deutschen ähnliche mit deutschen Sonderzeichen erstellen. Zur Definierung der deutschen Sonderzeichen ist die Anweisung

```
SYMBOL AFTER 64
```

notwendig. Diese Anweisung darf nur vor einer MEMORY-Anweisung gegeben werden, sonst tritt meist eine „Improper Argument"-Fehlermeldung auf. Parallel zur Definition des Zeichensatzes wird die Tastatur umbelegt. Das Pfundzeichen wird durch ↑ ersetzt, und ↑ wird durch ß ersetzt.

```
30000   Definition des deutschen Zeichensatzes und geri
nge Tastaturanpassung:
30010 SYMBOL AFTER 64:' Zeichensatz ab Paragraphenzeich
en in den Speicher laden
30020 SYMBOL 64,56,64,56,68,56,4,56,0:        'Definitio
n des Paragraphen
30030 SYMBOL 91,195,24,60,102,126,102,102,0: 'Definitio
n des großen Ä
30040 SYMBOL 92,195,60,102,102,102,102,60,0: 'Definitio
n des großen Ö
30050 SYMBOL 93,102,0,102,102,102,102,62,0:  'Definitio
n des großen Ü
30060 SYMBOL 123,102,0,120,12,124,204,118,0: 'Definitio
n des kleinen ä
30070 SYMBOL 124,102,0,60,102,102,102,60,0: '"Definitio
n des kleinen ö
30080 SYMBOL 125,0,102,0,102,102,102,60,0:   'Definitio
n des kleinen ü
30090 SYMBOL 126,124,198,198,220,198,198,220,192: 'Defi
nition des ß
30100 ' Tastenumdefinierung , v. a. Großbuchstaben mit
Kleinbuchstaben
30110 ' vertauschen. Außerdem wird der Paragraph auf de
n Backslash gelegt
30120 ' und das große ö auf SHIFT'at' . Das ß kommt auf
 die Taste mit dem
30130 ' Pfundzeichen . Die Umbelegung ist also nicht fü
r Leser, die im
30140 ' Zehnfingersystem schreiben, da sich die Sonder-
 und Satzzeichen nicht
30150 ' an den durch die DIN vorgeschriebenen Plätzen b
efinden.
30160 KEY DEF 17,1,123,91:' Taste 'Klammer auf' wird zu
m Ä
30170 KEY DEF 26,1,124,92:' Taste 'at' wird zum ö
30180 KEY DEF 19,1,125,93:' Taste 'Klammer zu' wird zum
 Ü
30190 KEY DEF 24,1,126,94:' Taste '^' und 'Pfund' wird
zu 'ß' und '^'
30200 KEY DEF 22,1,64,96:'  Taste 'Backslash' wird zu '
Paragraph'
30210 KEY DEF 43,1,&7A,&5A,&1A:' Taste Y wird zu Z
30220 KEY DEF 71,1,&79,&59,&19:' und Taste Z wird zu Y
```

Abb. 6.1: Deutscher Zeichensatz (Programm)

Die Tasten mit den eckigen Klammern werden so umbelegt, daß die den eckigen Klammern entsprechenden Vokale groß Ä und groß Ü mit klein ä und klein ü vertauscht werden. Dadurch erreicht man die Großbuchstaben mit der SHIFT-Taste. Des weiteren wird die Backslash-Taste (umgekehrter Schrägstrich) mit dem Paragraphen belegt. Der Paragraph übernimmt die Funktionen, die vorher @ hatte, also den Variablenpointer. Die @-Taste erhält im Normalzustand das kleine ö und im SHIFT-Modus das große Ö.

Das Ansprechen eines RSX (Resident System Extension - Systemerweiterung, siehe auch Kapitel 8) muß jetzt mit dem kleinen ö erfolgen. Die DIV-Funktion des Backslash muß nun mit dem großen Ö durchgeführt werden.

Ein abgedrucktes BASIC-Programm übernimmt die Umdefinierung. Dieses Programm kann durch seine hohen Zeilennummern an die meisten anderen BASIC-Programme mit MERGE angehängt werden.

ESC-Taste ausschalten

Die ESC-Tatse läßt sich mit mehreren CALLs blockieren, wobei eigentlich nur einer interessant ist, da die anderen eine bereits vorhandene Tastaturumbelegung wieder löschen.

Die Blockade der ESC-Taste ist, ebenso wie die ON BREAK GOSUB-Anweisung, nur wirksam, wenn das BASIC-Programm nicht steht, also keine INPUT-Anweisung ausgeführt wird. Außerdem wird die Blockade bei der Beendigung eines Programms gelöscht.

Der CALL für die Blockade lautet:

```
CALL &BB48
```

Funktionstasten; alle benutzen

Während des Resets werden beim CPC nur die Funktionstasten 128 bis 140 auf den Zehnerblock gelegt. Diese Funktionstasten entsprechen den KEYs 0 bis 12. Um nun weitere Funktionstasten nutzen zu können, ist die Neudefinition der Tastatur notwendig.

Es empfiehlt sich, die Zehnerblocktasten zum Teil direkt mit den Zahlen 0 bis 9, dem Punkt, dem Komma und der ENTER-Taste zu belegen. Das hat den Vorteil, daß diese nicht als Funktionstasten gelten und damit keinen Platz im Speicher für die Funktionstasten belegen.

Um die Tastatur umzudefinieren, ist der BASIC-Befehl KEY DEF nötig. Der erste Parameter dieses Befehls gibt die Tastaturnummer der Taste an, der

man etwas Neues zuweisen will. Die Tastaturnummer erhält man aus dem Anhang III/Seite 16 des CPC-Bedienungshandbuchs.

Der zweite Parameter bestimmt die Wiederholbarkeit der Taste; der Zehnerblock wird zweckmäßigerweise auf nichtwiederholend eingestellt, wenn Funktionstasten darauf liegen. Der dritte, vierte und fünfte Parameter geben den Tastenwert im Normal-, SHIFT- und CONTROL(CTRL)-Zustand an.

Um nun mehr als dreizehn Funktionstasten nutzen zu können, ist eine sinnvolle Umbelegung der Tastatur nötig. Um auf eine Taste eine Funktionstaste zu legen, muß man die Taste mit

```
KEY DEF Tastennummer,0,Unshift,Shift,Ctrl
```

umdefinieren. Damit die Funktionstasten nur auf den Shift- und Ctrl-Werten liegen, sind sie als zweite und dritte Parameter anzugeben. Die anzugebenden Werte entsprechen dabei der Summe von Keynummer und 128. KEY 0 muß also mit dem Wert 128, KEY 1 mit dem Wert 129 usw. definiert werden. Unser Vorschlag zur Belegung des Zehnerblocks:

TASTE	SHIFT	CTRL
0	KEY 0	KEY 10
1	KEY 1	KEY 11
2	KEY 2	KEY 12
3	KEY 3	KEY 13
4	KEY 4	KEY 14
5	KEY 5	KEY 15
6	KEY 6	KEY 16
7	KEY 7	KEY 17
8	KEY 8	KEY 18
9	KEY 9	KEY 19
.	,	KEY 20
ENTER	ENTER	KEY 21
CLR	CLR	KEY 22

Die Belegung erfolgt für den Normalzustand mit den ASCII-Werten des Tastenaufdrucks. Damit entspricht jede Taste im Normalzustand ihrem Aufdruck, im SHIFT-Zustand liegt das Komma auf der Punkt-Taste (nützlich bei längeren DATA-Eingaben von Zahlen). Die KEYs erreicht man für den SHIFT-Zustand der Zifferndaten mit der Ziffernangabe auf der Taste. Im CTRL-Zustand addiert man einfach 10 zu der Ziffer der Taste. Die restlichen drei sind der Reihe nach auf die Punkt-, die ENTER- und die CLR-Taste verteilt. Durch diese Belegungsweise bleiben außerdem noch die KEYs 23 bis 31 für eigene Tastaturdefinitionen übrig. Diese kann man ja, wenn sie auch noch benötigt werden, z. B. auf den CTRL-Zustand der oberen Zahlenreihe der Tastatur legen (außer CTRL-2 und 0). In Abb. 6.2 sehen Sie das zugehörige Programm.

Abb. 6.2: Tasten umdefinieren (Programm)

```
20000 ' Tastenumdefinierung für mindestens 24 Funktions
tasten
20010 KEY DEF 15,0,&30,128,138 : ' Taste  0 belegen
20020 KEY DEF 13,0,&31,129,139 : ' Taste  1 belegen
20030 KEY DEF 14,0,&32,130,140 : ' Taste  2 belegen
20040 KEY DEF  5,0,&33,131,141 : ' Taste  3 belegen
20050 KEY DEF 20,0,&34,132,142 : ' Taste  4 belegen
20060 KEY DEF 12,0,&35,133,143 : ' Taste  5 belegen
20070 KEY DEF  4,0,&36,134,144 : ' Taste  6 belegen
20080 KEY DEF 10,0,&37,135,145 : ' Taste  7 belegen
20090 KEY DEF 11,0,&38,136,146 : ' Taste  8 belegen
20100 KEY DEF  3,0,&39,137,147 : ' Taste  9 belegen
20110 KEY DEF  7,0,&2E,&2C,148 : ' Taste  . belegen
20120 KEY DEF  6,0,&D,&D,149   : ' Taste  ENTER  belege
n
20130 KEY DEF 16,1,&10,&10,150 : ' Taste  CLR    belege
n
20140 ' Die Tasten des Zehnerblocks sind nun wie folgt
belegt:
20150 ' 1. Im Normalzustand sind sie der Aufschrift ent
sprechend belegt.
20160 ' 2. Es liegt folgende SHIFT- und CONTROL-Belegun
g vor :
20170 'Taste:    SHIFT        CONTROL
20180 ' 0        KEY 0         KEY 10
20190 ' 1        KEY 1         KEY 11
20200 ' 2        KEY 2         KEY 12
20210 ' 3        KEY 3         KEY 13
20220 ' 4        KEY 4         KEY 14
20230 ' 5        KEY 5         KEY 15
20240 ' 6        KEY 6         KEY 16
20250 ' 7        KEY 7         KEY 17
20260 ' 8        KEY 8         KEY 18
20270 ' 9        KEY 9         KEY 19
20280 ' .         ","          KEY 20
20290 'ENTER     ENTER         KEY 21
20300 ' CLR       CLR          KEY 22
20310 ' Sollte noch keine Umbelegung erfolgt sein, so s
ind die Zahlen auch noch
20320 ' in der SHIFT-Ebene vorhanden, '.' liegt jedoch
auf CTRL-0 ENTER auf
20330 ' CTRL-1 und RUN" auf CTRL-2 des Zehnerblocks. Di
e Funktionstasten
20340 ' 23 bis 31 köen per KEY DEF  mit den Werten 151
bis 159 noch
20350 ' anderen Tasten der Tastatur zugewiesen werden.
Vorausgesetzt man
20360 ' hat noch genügend Platz im Funktionstastenpuffe
r.
20370 ' Folgende nicht ganz willkürliche Funktionstaste
nbelegung
20380 ' würden wir vorschlagen:
20390 KEY 0,"RUN "
20400 KEY 1,"GOTO "
20410 KEY 2,"GOSUB "
20420 KEY 3,"RETURN"
20430 KEY 4,"WHILE "
20440 KEY 5,"WEND"
20450 KEY 6,"FOR "
20460 KEY 7,"NEXT "
20470 KEY 8,"PRINT "
20480 KEY 9,"INPUT "
20490 KEY 10,"INKEY$"
20500 KEY 11,"EVERY "
20510 KEY 12,"AFTER "
20520 KEY 13,"SAVE"+CHR$(34)
20530 KEY 14,"LOAD"+CHR$(34)
20540 KEY 15,"DATA "
20550 KEY 16,"READ "
20560 KEY 17,"LIST "
20570 KEY 18,"EDIT "
20580 KEY 19,"MODE "
20590 KEY 20,"KEY "
```

```
20600 KEY 21,"DEF"
20610 KEY 22,"CLS"+CHR$(13)
20620 ' Mit CTRL und der CLR-Taste läßt sich nun tatsäc
hlich
20630 ' der Bildschirm löchen, allerdings leider nur im
 Basic.
```

Abb. 6.2: Tasten umdefinieren (Fortsetzung)

Geschützte BASIC-Programme laden

Zum Laden von geschützten BASIC-Programmen muß man folgendermaßen vorgehen:

1. Öffnen einer Datei mit dem Namen des zu ladenden Files. Bei Kassettenbetrieb darf der Name auch fehlen, dann wird einfach die nächste Datei genommen.
2. Direktes Einlesen des Programms an eine anzugebende Speicheradresse; diese Speicheradresse ist bei BASIC-Programmen die BASIC-Anfangsadresse 170H.
3. Anschließendes Speichern mit der ASCII-Option:

   ```
   SAVE "name",A
   ```

   damit eventuell vorhandene Sicherungsbytes herausfallen. Das Programm kann bereits gelistet werden.
4. Dann kann das abgespeicherte Programm erneut eingelesen und ganz normal gestartet oder gespeichert werden.

Im Detail sieht das so aus:

1. Das Eröffnen einer Datei zum Einlesen geschieht durch den Aufruf von BC77H. Dabei muß im B-Register der CPU die Länge des Dateinamens angegeben werden. Der Name muß bei Diskettenbetrieb immer angegeben werden. Der Wert im Register B kann bei Kassettenbetrieb 0 sein. Die im HL-Registerpaar zu übergebende Adresse des Namens kann dann entfallen.

   Im Registerpaar DE muß die Adresse eines 2KByte-Puffers immer angegeben werden. Der Puffer kann im ganzen Speicher liegen, also z. B. auch im Bildspeicher. Der Aufruf erfolgt also mit folgenden Parametern:

   B = Länge des Dateinamens (des Programmnamens)
   HL = Adresse des Namens (kann bei B = 0 entfallen)
   DE = Adresse eines 2K-Puffers

2. Das Einlesen der Datei geschieht durch den Aufruf von BC83H. Dabei muß im HL-Registerpaar die Zieladresse der Datei überreicht werden (bei BASIC-Programmen HL = 170H).

   Kurzes Beispielprogramm:

   Zur Anwendung müssen die folgenden 14 Bytes in einen mit MEMORY geschützten Speicherbereich gePOKEd werden. Die Assemblerbefehle bzw. Bytes in Hexadezimalschreibweise sehen so aus:

```
LD      B,0          &06 , &00
LD      DE,9800H     &11 , &00 , &98
CALL    BC77H        &CD , &77 , &BC
LD      HL,170H      &21 , &70 , &01
JP      BCB3H        &23 , &83 , &BC
```

   Diese Bytes müssen hintereinander im Speicher stehen. Gestartet wird dieses Maschinenprogramm mit dem Aufruf der ersten Speicherzelle, die es belegt. Das Programm liest die erste Datei von der Kassette, egal, ob es ein BASIC-Programm ist oder nicht, und legt sie an den Anfang des BASIC-Speichers. Dateien von Disketten können nicht geladen werden, da der Dateiname nicht überreicht wird.

   Sollten Sie während des Diskettenbetriebs eine Datei damit laden wollen, so müssen Sie sicherstellen, daß der Lesezugriff auf die Kassette erfolgt. Die Rückkehr aus diesem Maschinenprogramm erfolgt mit dem RETURN aus der Routine BC83H.

   Das erhaltene Programm kann gelistet, aber nicht ausgeführt werden, deshalb ist die anschließende Abspeicherung als ASCII-Datei, wie oben schon erwähnt, notwendig.

   Mit diesem Programm soll keineswegs ds Knacken von käuflicher Software unterstützt werden. Es ist zu empfehlen, wenn sich einmal die ungeschützte Version eines Programms nicht mehr laden läßt. Außerdem gibt es noch etwas bessere Schutzmechanismen für CPC-Programme, sofern sich die Programmentwickler mit den Möglichkeiten des CPC auseinandersetzen.

INPUT-Befehl

Soll bei der INPUT-Anweisung der Zeilenvorschub unterbunden werden, so ist nur eine geringe Änderung im Befehl notwendig. Wenn eine Variable ohne Textausgabe eingelesen werden soll, so muß nur vor dem Variablennamen ein Semikolon geschrieben werden. Wird Text mit ausgegeben, so gehört dieser

Strichpunkt vor den Ausgabetext. Der Befehl muß also folgendermaßen aussehen:

```
INPUT; Variablenname
```
oder
```
INPUT; "Text"; Variablenname
```

Der zweite Strichpunkt muß bei Textausgabe immer mit angegeben werden.

Will man vielleicht das Fragezeichen nach dem Text unterdrücken, so ist lediglich der zweite Strichpunkt durch ein Komma zu ersetzen. Das Komma vor einer einzulesenden Variablen unterdrückt nämlich das Fragezeichen.

Die Unterdrückung von Zeilenvorschub und von Fragezeichen ist nur bei Textausgabe im INPUT-Befehl möglich. Doch keine Regel ohne Ausnahme: Die Unterdrückung von Zeilenvorschub und Fragezeichen ist ebenfalls durch den LINE INPUT-Befehl möglich. LINE INPUT ist allerdings nur für eine einzige (String-)Variable geeignet, da er sämtliche Kommata überliest.

Inverse/normale Schrift umschalten

Neben der in Kapitel 2 dargestellten Möglichkeit, inverse Schrift zu erzeugen und diesen Vorgang rückgängig zu machen, die „normale" Art und Weise mittels PEN und PAPER, gibt es noch eine weitere, die besonders für Maschinensprache interessant ist:

```
CALL &BB9C
```

In Assembler müssen die beiden Bytes natürlich umgedreht werden:

```
CALL 9C,BB
```

Kassettenmotor anhalten

Wenn Sie die Kassette mit Drücken der PLAY-Taste gestartet haben, können Sie sie mit

```
CALL &BC71
```

wieder anhalten.

Kassettenmotor starten

Wenn Sie den Kassettenrecorder mit dem angegebenen Befehl angehalten haben, können Sie ihn mit

```
CALL &BC6E
```

wieder starten.

Kassettenrecorder auf ca. 4000 Baud beschleunigen

Der Datenrecorder des CPC ist für höhere Geschwindigkeiten ausgelegt, als sie der CPC unter BASIC benutzt. Die Einstellung der durchschnittlichen Baudrate erfolgt über einen Sprung in der Sprungtabelle des CPC.

Der Sprung erfolgt nach BC68H. Das HL-Registerpaar muß dabei die Dauer für ein halbes Null-Bit in Mikrosekunden enthalten. Im Register A muß die Vorprüflänge für ein Datenbit, ebenfalls in Mikrosekunden, enthalten sein. Die Vorprüflänge ist zur Beschleunigung nicht so wichtig, sie kann ca. 60 betragen. Die eigentliche Baudrate wird durch den Wert im Registerpaar HL bestimmt. Die Dauer für ein Eins-Bit ist zweimal so lang wie für ein Null-Bit. Darauf folgt eine durchschnittliche Baudrate von 1/(3*HL) Baud.

Eine ausgetestete Baudrate von ca. 4000 Baud (HL = 0053h) war die größtmögliche Einstellung. Bei kleineren HL-Inhalten trat dann ein „Write Error" auf, d. h. das Betriebssystem war nicht mehr in der Lage, mit dieser Geschwindigkeit zu schreiben. Die zu Ihrem CPC passende Baudrate müssen Sie durch Ausprobieren herausfinden.

Bei diesen Baudraten ist bisher noch kein Lesefehler aufgetreten, solange die Kassette am gleichen Computer abgespielt wurde. Die Kompatibilität zwischen den CPCs ist bei dieser Geschwindigkeit nicht mehr gewährleistet. Die Einstellung einer höheren Baudrate erfolgt mit dem folgenden kurzen Maschinenprogramm:

```
LD  HL,60H      &21 , &60 , &00 ; ca. 3500 Bd.
LD  A,60D       &3E , &3C
JP  BC68H       &C3 , &68 , &BC
```

Der Aufruf geschieht mit einem CALL an das erste Byte dieses im Speicher abgelegten Maschinenprogramms. Der Rücksprung erfolgt über die Routine. Sollte wieder eine niedrigere Baudrate gewünscht werden, so muß die Baudrate durch die Kommandos SPEED WRITE 0 oder 1 oder durch den Aufruf von BC65H, der die Kassettenschnittstelle zurücksetzt, umgestellt werden.

Die Routine BC65H benötigt keine Parameter, der Aufruf kann also durch CALL &BC65 geschehen.

Maschinenprogramme in niedrigen Speicherbereich laden

Wenn Maschinenprogramme in einen sehr niedrigen Speicherbereich geladen werden sollen, so bringt das BASIC oft die Fehlermeldung „Memory full". Der Speicherbereich, in den die Programme geladen werden sollen, muß geschützt sein. Der Speicher muß also mit dem MEMORY-Befehl reserviert

werden. Erfolgt trotzdem die oben genannte Fehlermeldung, so ist dem BASIC-Interpreter der Speicherplatz unterhalb von HIMEM zu klein, um einen 2-KByte-Puffer für die Kassettenoperation anzulegen.

Dieses Problem läßt sich dadurch umgehen, daß folgende Befehle zur Beschränkung des BASIC-Speichers gegeben werden:

```
OPENOUT "DUMMY"   Beliebige Datei für Ausgabe eröffnen
MEMORY nnnn       Beschränkung auf benötigten Speicherbereich
CLOSEOUT          Datei schließen
```

Mit dieser Befehlskombination wird der Kassettenrecorder nämlich im obersten Speicherbereich angelegt und muß deshalb nicht unterhalb von HIMEM zur Verfügung stehen.

Ein ähnlicher Trick kann auch zur Verhinderung der gefürchteten Garbage Collection durchgeführt werden, vorausgesetzt man hat 4 KByte überflüssigen Speicher zur Verfügung. Der Trick besteht aus einer Anweisung, nämlich

```
OPENOUT "DUMMY" : MEMORY HIMEM-1 : CLOSEOUT
```

Dieser Trick bringt nur etwas, wenn Daten oder ein Programm abgespeichert werden sollen. Sind diese Anweisungen nämlich am Anfang des Programms ausgeführt worden, so muß keine Garbage Collection durchgeführt werden, um die 4 KByte Kassettenpuffer freizumachen.

Programmsicherung und Schutzverhinderung bei mit p geschützten Programmen

Der Schutz von Programmen wird durch ein einziges Byte im Speicher gekennzeichnet. Dieses Byte hat die Adresse &AE45. Steht in diesem Byte der Wert Null, so gilt dieses Programm als nicht geschützt. Steht darin ein von Null verschiedener Wert, so ist das Programm als geschützt markiert und wird beim Eintritt in den Direktmodus (Ready-Modus) mit dem NEW-Befehl gelöscht. Vor dem Eintritt in den Ready-Modus wird jedoch eine Adresse aufgerufen, die im RAM steht. Normalerweise steht dort ein RETURN-Befehl (&C9).

Ersetzt man die drei Bytes an dieser Adresse durch einen Sprungbefehl an eine eigene Routine, die das Schutzbyte löscht, so gilt das Programm als ungeschützt und kann, wenn es beendet wird oder unterbrochen werden kann, wie ein ungeschütztes Programm gelistet und abgespeichert werden.

Die folgenden POKEs definieren ein Programm als ungeschützt:

In BASIC:	In Assembler:
POKE &AC01,&C3	ORG &AC01
POKE &AC02,&00	JP &A600
POKE &AC03,&A6	

Diese POKE-Befehle bewirken einen Sprung zu der Routine, die das Schutzbyte löscht. Es wird dabei der Vektor vom Ready-Modus (normalerweise ein RETurn) zu einem Sprung auf unsere Ersatzroutine verbogen. Die Ersatzroutine steht an der Adresse &A600.

Das zu ,,entschützende Programm darf also die Speicherzellen &AC01 bis &AC03 und die Speicherzellen &AC600 bis &A604 (dort steht die Ersatzroutine) nicht überschreiben. Die Ersatzroutine an der Adresse &A600 lautet folgendermaßen:

In BASIC:	In Assembler:
POKE &A600,&AF	ORG &A600
POKE &A601,&32	XOR A
POKE &A602,&45	LD (&AE45),A
POKE &A603,&AE	RET
POKE &A604,&C9	

Nun wollen wir das Gegenteil betrachten: eine Falle für Leute, die ein Programm gegen den Willen des Programmierers in den Ready-Modus holen wollen.

Dazu muß die Speicherzelle &AC01 nur mit dem Wert &C7 (einem ReStarT 0) geladen werden. Bei einer Programmunterbrechung, die in den Ready-Modus führt, wird zuerst die Adresse &AC01 aufgerufen, worin ein RST 0 steht. Dieser Restart führt zu einem Reset (er verzweigt auf Adresse 0), wenn das Programm unterbrochen wird. Dieser Programmschutz sollte am besten in der ersten Adresse eines Programms stehen, und zwar als

```
POKE &AC01,C7
```

Damit ist ein Programm gegen den oben genannten ,,Entschützer" gefeit.

Noch ein Tip: Das Programm zum ,,Entschützen" muß in den Speicher gePOKEd worden sein, bevor das geschützte Programm mit RUN gestartet wird.

REM-Zeilen mit Sonderzeichen

Sollten Sie schon einmal Zeichen selbst definiert und diese dann in REM-Zeilen erklärt haben, so werden Sie sich geärgert haben, daß da plötzlich BASIC-Befehle in den Zeilen standen.

Dieser Effekt tritt dann auf, wenn die definierten Zeichen, die einen Wert, der größer ist als 127, besitzen, mit dem COPY-Cursor eingetragen wurden. Der Umwandlung von als Token erkannten Zeichen beim Listen kann man sehr einfach entgegenwirken. Man muß nur hinter dem REM-Befehl oder dem Apostroph ein Anführungszeichen einfügen. Das BASIC behandelt dann die folgenden Zeichen wie eine Zeichenkette.

Der Effekt tritt noch schwerwiegender auf, wenn Sie sich einen deutschen Zeichensatz definiert haben und in den REM-Zeilen das kleine ö verwenden. Das ö ist nämlich sonst das Kennzeichen für einen RSX-Befehl und bewirkt, daß das hinter dem ö stehende Zeichen zwar abgespeichert, aber beim LIST-Befehl nicht mehr angezeigt wird. Beheben läßt sich dies durch das Einfügen eines Leerzeichens nach jedem ö oder eines Anführungszeichens nach dem REM-Kommando.

Wie oben bei den Symbolen mit Werten, die größer als 128 sind, so wirkt auch hier bei dem kleinen ö ein einziges Anführungszeichen nach dem REM-Befehl wahre Wunder. Der Programmablauf wird dann nicht mehr durch Fehlermeldungen ohne Grund unterbrochen.

Tastendruck abwarten

Neben der allgemein üblichen Möglichkeit mit

```
100 A$=INKEY$ : IF A$="" " THEN 100
```

gibt es noch eine Betriebssystemroutine, die gleiches erfüllt:

```
CALL &BB06
```

Maschinenprogrammierer können aus dem Akkumulator den Code der gedrückten Taste entnehmen.

Die Programmkompatibilität zu anderen Rechnern geht allerdings verloren.

Uhr einstellen/anzeigen

Der CPC besitzt zwar eine Uhr, jedoch wird sie nur beim Einschalten zurückgesetzt und kann mit einem BASIC-Befehl nicht eingestellt werden (PRINT TIME gibt nur die seit dem Einschalten verstrichene Zeit in $^1/_{300}$ Sekunden aus).

Da die Uhr dabei irgendwo weitergeschaltet werden muß - was natürlich nur im RAM geschieht -, gilt es, diese Speicherzellen ausfindig zu machen. Diese sind:

45447 bis 45450

Das Programm in Abb. 6.3 erlaubt Ihnen das Einstellen einer beliebigen Uhrzeit.

```
100 PRINT"Zeit in Stunden:Minuten:Sekunden"
110 INPUT"hh:mm:ss";t$
120 :
130 IF LEN(t$) <> 8 THEN 110
140 IF MID$(t$,3,1) <>":" THEN 110
150 IF MID$(t$,6,1) <>":" THEN 110
160 :
170 h=VAL(LEFT$(t$,2))
180 m=VAL(MID$(t$,4,2))
190 s=VAL(RIGHT$(t$,2))
200 :
210 t = (h*3600 + m*60 + s) * 300
220 :
230 t1 = INT(t/2^24)
240 t = t-t1*2^24
250 :
260 t2 = INT(t/2^16)
270 t = t-t2*2^16
280 :
290 t3 = INT(t/2^8)
300 t = t-t3*2^8
310 :
320 POKE 45447,t
330 POKE 45448,t3
340 POKE 45449,t2
350 POKE 45450,t1
360 :
370 CLS
380 LOCATE 1,1
390 GOSUB 500
400 PRINT "Stunden     ";h
410 PRINT "Minuten     ";m
420 PRINT "Sekunden    ";s
430 GOTO 380
440 :
500 s=INT(TIME/300)
510 m=INT(s/60)
520 h=INT(s/3600)
530 m=m-h*60
540 s=s-m*60-h*3600
550 RETURN
```

Abb. 6.3: Zeitanzeige-Programm

Zunächst wird die Zeit in der üblichen Schreibweise erfaßt und anschließend eine Plausibilitätsprüfung durchgeführt, die ähnlich der für das Datum aus Kapitel 1 funktioniert.

Als Übung können Sie eine weitere Prüfung für Stunden (je nach Wunsch 12 oder 24), Minuten und Sekunden einfügen. Dann werden die Zahlenvariablen h (Stunden), m (Minuten) und s (Sekunden) entsprechend besetzt.

Da der Rechner in Einheiten von $^1/_{300}$ Sekunden rechnet, gibt t nach Zeile 210 die Zeit in dieser Grundeinheit an.

Die vier vorgenannten Bytes können als 4-Byte-Zahl (positive Zahlen bis 4.29497 E+09 darstellbar) aufgefaßt werden. Wie immer in Computern kommt das niederwertigste Byte zuerst. Als Binärzahl ergibt sich somit das Schema aus Abb. 6.4.

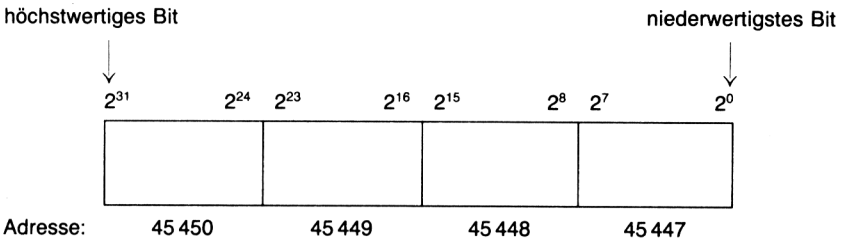


Abb. 6.4: 4-Byte-Zahl für TIME

Es ist also nicht so, daß jedes Bit für eine Zeiteinteilung (Stunden, Minuten, Sekunden, 1/300 Sekunden) steht, was für Byte 45447 auch gar nicht möglich wäre.

Ab Zeile 230 werden die Werte für die einzelnen Bytes errechnet, wobei mit dem höchstwertigen Byte begonnen wird. Ab Zeile 320 werden die erhaltenen Werte ins RAM übertragen.

Die Zeilen ab 370 dienen der Kontrollausgabe, wobei wieder eine Umrechnung in Stunden, Minuten und Sekunden erfolgt (Unterprogramm ab Zeile 500). Durch String-Operationen und unter Verwendung der in Kapitel 1 dargestellten Routine zum Einfügen führender Nullen können Sie auch eine Ausgabe der Zeit wie bei der Eingabe realisieren.

Zufallszahlen

Der CPC hat einen eingebauten Pseudo-Zufallsgenerator (RND-Befehl), der mit dem RANDOMIZE-Befehl initialisiert wird. Die Folge der ausgegebenen Zufallszahlen ist vom Startwert (Variable hinter RANDOMIZE) abhängig und bei gleichen Startwert immer identisch (mit einer kleinen FOR. . .NEXT-Schleife können Sie sich selbst überzeugen).

Wenn Sie bei Aufruf eines Programms immer unterschiedliche Zufallszahlen haben möchten (statistische Simulationen, Spiele), so ist es zweckmäßig, die Variable nach RANDOMIZE zunächst mit INPUT zu erfassen.

Aber der CPC hat auch eine Variable, die sich ständig ändert: TIME. Kürzer geht es also mit:

```
RANDOMIZE TIME
```

Kapitel 7

Die wichtigsten Betriebssystemroutinen

In diesem Kapitel werden wir einige der wichtigsten Betriebssystemroutinen behandeln. Wegen dem äußerst umfangreichen Betriebssystem des CPC können nur wenige Routinen angesprochen werden. Es werden die Routinen, die in diesem Buch in Programmen vorkommen, näher behandelt und ein paar allgemein brauchbare beschrieben. Bei der Beschreibung der Routinen wird nach der Einsprungadresse vorgegangen, damit sind die Routinen zugleich nach Zusammengehörigkeit gegliedert.

Tastaturroutinen

BB00h: Vollständiger RESET der Tastaturverwaltung. Alle Definitionen gehen verloren. Kann während eines Programms zur Blockade der ESC-Taste benutzt werden.

BB03h: Initialisierung des Tastaturpuffers, kann ebenfalls zur Blockade der ESC-Taste eingesetzt werden.

BB06h: Wartet auf ein Zeichen von der Tastatur. Zeichen wird im Akku zurückgeliefert.

BB09h: Einmalige Tastaturabfrage. Wenn Zeichen vorhanden, wird das Carry-Flag gesetzt. Sonst wie BB06h.

BB18h: Wartet auf einen Tastendruck; der Tastenwert wird im Akku zurückgeliefert.

BB1Bh: Einmalige Tastenabfrage, Carry wird bei Tastendruck gesetzt. Sonst wie BB18h.

BB1Eh: Prüft die Taste mit der Tastennummer, die im Akku überreicht wird. Wurde die Taste gedrückt, so ist das ZERO-Flag zurückgesetzt. Im Register C werden die Zustände der SHIFT- und CTRL-Taste übergeben. Bit 7 von C bedeutet CTRL-Taste und Bit 5 SHIFT-Taste.

BB24h: Holt den momentanen Joystickzustand. Joy 0 wird im H- und A-Register und Joy 1 im L-Register angegeben. Die Bitzustände sind wie bei dem BASIC-Befehl JOY( ).

BB3Fh: Setzt die Anfangsverzögerung und die Wiederholgeschwindigkeit mit einer Genauigkeit von 1/50 s. H enthält die Verzögerung, und L enthält die Wiederholgeschwindigkeit in 1/50 Sekunden.

BB42h: Holt die Anfangsverzögerung und die Wiederholgeschwindigkeit. H = Verzögerung, L = Wiederholgeschwindigkeit, jeweils in 1/50 Sekunden.

BB48h: Sperrt Unterbrechungen, ESC-Taste wird blockiert.

Textausgaberoutinen

BB5Ah: Gibt das Zeichen im Akku auf den Bildschirm aus. Control-Codes werden ausgeführt.

BB60h: Liest ein Zeichen von der aktuellen Cursorposition des aktuellen Fensters. Wenn ein Zeichen erkannt wurde, wird Carry gesetzt und das Zeichen im Akku zurückgeliefert. Sonst ist Carry zurückgesetzt.

BB66h: Das momentane Fenster wird auf die in H und D enthaltenen Spalten und die in L und E enthaltenen Zeilen gesetzt. Der Cursor wird auf die linke obere Ecke des Fensters gesetzt. Die linke obere Ecke des Bildschirms hat die Koordinaten 0,0.

BB69h: Die Koordinaten des aktuellen Fensters werden zurückgeliefert. Carry ist gesetzt, wenn das Fenster den ganzen Bildschirm umfaßt. Die linke obere Bildschirmecke hat die Koordinate (0,0).

BB6Ch: Löscht das aktuelle Fenster mit der Farbe des Papers.

BB75h: Setzt den Cursor im Fenster auf die in HL übergebene Koordinate. H enthält die Spalte und L die Zeile. Linke obere Ecke des Fensters ist 1,1.

BB78h: Holt die aktuelle Cursorposition mit H-Spalte und L-Zeile. Die Cursorposition links oben im aktuellen Fenster ist 1,1.

BB90h: Setzt die PEN-Farbe des Fensters auf den Wert im Akku.

BB93h: Gibt die Farbe des Pens im Akku zurück.

BB96h: Setzt die PAPER-Farbe des Fensters auf den Wert im Akku.

BB99h: Gibt die PAPER-Farbe im Akku zurück.

BB9Ch: Wechselt PEN- und PAPER-Farbe aus.

Grafikroutinen

Bei allen Koordinaten erfolgt die Darstellung als Zweierkomplement (Zahl mit Vorzeichen).

BBBAh: Führt einen RESET des Grafiksystems durch.

BBC0h: MOVE zu dem Y-Wert in HL und X-Wert in DE relativ zum Ursprung.

BBC3h: MOVER : Relative Bewegung des Grafikcursors zur alten Position. Y = HL, X = DE.

BBC6h: Holt die aktuelle Grafikcursorposition relativ zum Ursprung. HL = Y-Koordinate und DE-Register = X-Koordinate.

BBC9h: Setzt den Ursprung auf die in HL und DE übergebenen Koordinaten. Y = HL, X = DE-Register.

BBCCh: Holt die Ursprungskoordinate in das HL- und das DE-Register, HL = Y-, DE = X-Koordinate.

BBCFh: Setzt den linken und rechten Rand des Grafikfensters auf die in HL und DE spezifizierten Werte. Der Rand wird immer auf ganze Bytes gelegt.

BBD2h: Setzt den oberen und unteren Rand des Grafikfensters auf den Wert in HL und in DE.

BBD5h: Holt den rechten und linken Rand des Grafikfensters, DE = rechts und HL = links.

BBD8h: Holt den oberen und unteren Rand des Grafikfensters, DE = oben und HL = unten.

BBDBh: Löscht das Grafikfenster mit dem Grafik-Paper.

BBDEh: Setzt den Grafik-Pen auf den Wert im Akku.

BBE1h: Gibt den Grafik-Pen im Akku zurück.

BBE4h: Setzt das Grafik-Paper auf den Wert im Akku.

BBE7h: Gibt das Grafik-Paper im Akku zurück.

BBEAh: PLOT zu dem Y-Wert in HL und X-Wert in DE bezüglich des Grafiksprungs.

BBEDh: PLOTR: Relativ zur alten Position einen Punkt plotten, X-Offset = DE, Y-Offset = HL.

BBF0h: TEST des Punktes mit den Koordinaten (DE,HL) bezüglich des Grafiksprungs.

BBF3h: TESTR: Der Punkt mit den zur alten Position relativen Koordinaten (DE,HL) wird getestet. Die beiden Testroutinen liefern im Akku die Farbe zurück.

BBF6h: DRAW zu den durch DE und HL spezifizierten Punkt.

BBF9h: DRAWR: Relatives Zeichen zur alten Position mit dem X-Offset im DE-Register und dem Y-Offset im HL-Register.

Bildschirmroutinen

BC0Eh: Setzt den Bildschirmmodus auf den Wert im Akku.

BC11h: Gibt den Bildschirmmodus im Akku zurück.

BC14h: Löschen des gesamten Bildes mit INK 0.

BC17h: Gibt im Register B die maximale Spaltenadresse und im Register C die maximale Zeilenadresse (immer 24) in Abhängigkeit vom Modus zurück.

BC1Dh: Übersetzt die Koordinaten eines Punktes in die Adresse des Bildschirmspeichers. Die Koordinaten werden in das DE(X-Wert)- und HL(Y-Wert)-Register übergeben und beziehen sich immer auf ganze Pixel, d. h. im Modus 0 ist die maximale X-Koordinate 159, und die maximale Y-Koordinate ist immer 199. Diese Koordinaten beziehen sich immer auf links unten (0,0).

BC26h: Übersetzt die im HL-Register übergebene Bildschirmadresse in eine um eine Zeile darunterliegende und gibt diese wieder im HL-Register zurück.

BC2Ch: Codiert die im Akku überreichte Farbe, so daß sie in den Bildspeicher eingeschrieben werden kann.

BC32h: Setzt die im Akku angegebene Farbe auf die in B und C angegebenen Farbnummern.

BC35h: Gibt die beiden Farbnummern in B und C zurück, auf die die Farbe im Akku gesetzt ist.

BC38h: Setzt den Border auf die Farbnummern in B und C.

BC3Bh: Gibt die Farbnummern des Borders in B und C zurück.

BC3Eh: Setzt die Farbwechselfrequenz auf die Werte im H- und L-Register. Die Werte geben die Farbdauer in 1/50 s an. H ist für die erste und L für die zweite Farbnummer zuständig.

BC41h: Gibt die Blinkperiodendauer in der Form zurück, wie sie mit BC3Eh gesetzt wird.

Routinen für den Massenspeicher (Kassette/Floppy)

BC65h: Führt einen RESET der Kassettenverwaltung durch.

BC68h: Stellt die Schreibgeschwindigkeit ein. Baudrate = 1/(3*HL) Mikrosekunden. A = Vorprüflänge in Mikrosekunden (30 bis 60).

BC77h: Eröffnet eine Datei für Eingabe: B = Länge des Dateinamens, HL = Adresse des Dateinamens und DE = Adresse eines 2-KByte-Puffers.

BC7Ah: Eingabedatei schließen.

BCB0h: Ein Zeichen von der Eingabedatei im Akku liefern.

BC83h: Datei direkt zur Adresse in HL laden.

BC89h: Test auf Dateiende (EOF). Bei Dateiende wird das Carry-Flag zurückgesetzt.

BCBCh: Datei für Ausgabe eröffnen: B = Länge des Dateinamens, HL = Adresse des Dateinamens und DE = Adresse eines 2-KByte-Puffers.

BC8Fh: Ausgabedatei schließen.

BC95h: Das Zeichen im Akku in die Ausgabedatei übertragen.

BC98h: Ausgabedatei direkt ausgeben. Der Anfang des Speicherbereichs steht in HL, die Länge in DE, die Startadresse in BC und die Dateiart im Akku.

BC9Bh: Erzeugt ein Inhaltsverzeichnis des Datenträgers. Im DE-Register muß dazu die Adresse eines 2-KByte-Puffers bei Kassettenbetrieb übergeben werden.

Soundroutinen

BCA7h bis BCC5h:
Ist für die Soundgenerator-Ansteuerung zuständig.

Befehlserweiterungen

BCC8h und BCCBh:
Initialisieren von Hintergrund-ROMs.

BCD1h: Führt eine RSX in das System ein. Dazu wird im Register BC die Adresse der Kommandos und im Register HL die Adresse von vier Speicherzellen für die Verwaltung der RSX-Kommandos übergeben.

BCD4h: Sucht RSX-Kommandos. Die Adresse des Namens wird im HL-Register übergeben. Wird die RSX gefunden, so kehrt die Routine mit gesetztem Carry-Flag, der Einsprungadresse im HL-Register und, wenn es in einem Hintergrund-ROM gefunden wurde, mit der ROM-Nummer in C zurück.

Interrupt-Steuerung

BCD7h bis BD0Ah:
Sind zur Verwaltung der Zeitereignisse und der sonstigen Unterbrechungen bestimmt.

BD0Dh: Holt die Zeit der Uhr mit $^1/_{300}$ s Genauigkeit in die Register DE und HL als 4-Byte-Wert. Dabei ist im Register D das höchstwertige Byte und in L das mit der niedrigsten Wertigkeit.

BD10h: Setzt die Uhr auf den 4-Byte-Wert DEHL.

BD19h: Wartet auf den Strahlrücklauf des Bildschirms.

Druckersteuerung

BD2Bh: Versucht das Zeichen im Akku an den Drucker auszugeben. Kann ein Zeichen nach ca. 0.4 sec immer noch nicht abgeschickt werden, so setzt die Routine das Carry und kehrt zurück.

BD2Eh: Testet den Centronics-Port darauf, ob das BUSY-Signal (beschäftigt) anliegt. Ist dies der Fall, so wird das Carry-Flag gesetzt zurückgeliefert.

BD37h: Setzt die Sprungtabelle auf die Standardsprünge.

Sollten Sie eigene Routinen schreiben, so bedienen Sie sich am besten nur der Sprungtabelle und hier sogar nur der indirekten. Das hat den Vorteil, daß man immer eine „Notbremse" ziehen kann, wenn man zu viele Sprünge verbogen hat. Außerdem können dann die Routinen anderer Programmierer über Ihre Sprünge geleitet werden. Das kann z. B. auftreten, wenn Sie einen Druckertreiber oder Spooler in das System integriert haben und diese Routinen nicht nur Ihrer Software zugänglich sein sollen.

Zum Umlenken der Sprünge ist die indirekte Sprungtabelle sehr gut geeignet. Durch eine weitere Tabelle, über die die BASIC-Funktionen verwaltet werden, kann man sogar dem BASIC, wenn genauere Mathematikroutinen gewünscht werden, im wahrsten Sinne des Wortes auf die „Sprünge helfen".

Kapitel 8

Befehlserweiterungen

Obwohl der Schneider CPC ein umfangreiches BASIC besitzt, ist es für Einzelfälle doch interessant, das vorhandene um eigene Routinen zu ergänzen.

VORGEHENSWEISE

Eigentlich kann man sie nicht als Befehlserweiterungen bezeichnen, die RSX-Kommandos des CPC. RSX heißt Resident System Extension und ist eine in den Speicher geladene Systemerweiterung. Die RSX-Kommandos sind nichts anderes als komfortable CALL-Befehle und werden auch wie solche behandelt.

Damit Sie die Funktion eines RSX verstehen, soll zuerst der CALL-Befehl erläutert werden. CALL kann bis zu 32 Parameter übernehmen. Diese Parameter müssen alle vom Typ Integer sein, können also nur ganze Zahlen im Bereich von −32768 bis +32767 annehmen.

Der erste Wert von CALL ist eine Speicheradresse und kann daher auch eine Zahlenvariable sein. Bei der Übernahme von Adresse oder Parameter wird eine Fließkommavariable vom BASIC in einen Integer-Wert umgewandelt. Nach dem Aufruf eines Maschinenprogramms mit CALL wird im Register A (Akkumulator) die Anzahl der Parameter angegeben. Dies ist wichtig, wenn die Gültigkeit des Aufrufs überprüft werden soll.

Falls Parameter übergeben wurden, so sind diese alle über das IX-Register der Z80-CPU zu erreichen. Die Parameter werden auf den Stapelspeicher (Stack) des Z80 abgelegt, d. h. der erste Parameter wird zuerst abgelegt, der zweite als zweiter usw.

Dann wird das IX-Register so gesetzt, daß es auf den letzten Parameter zeigt. Wenn man nun den letzten Parameter einlesen will, so liegt dessen niederwertiges Byte an der Adresse IX+0, das höherwertige Byte liegt eine Speicherzelle weiter, also auf IX+1. Allgemein liegt das niederwertige Byte des n-ten von i Parametern an der Adresse (i-n)*2+IX.

Wie überreicht man nun aber Fließkommawerte oder gar Zeichenketten (Strings), wenn ein Parameter nur Integer-Werte annehmen darf? Dieses Problem kann ebenfalls gelöst werden. Das BASIC kennt einen Variablenpointer, es ist das Zeichen @ oder, wie manche sagen, der Klammeraffe.

Steht dieses Zeichen vor einer Variablen, egal welchen Typs, so liefert der Ausdruck die Adresse dieser Variablen. Die Variable muß dazu aber einmal einen Wert zugewiesen bekommen haben, sonst kommt die Fehlermeldung „Improper Argument", da der CPC die Variable nicht finden kann.

Die zurückgelieferte Adresse hat je nach Variablentyp unterschiedliche Bedeutung. Bei Integer-Variablen ist es die Adresse des Integer-Wertes, bei Fließkommavariablen ist es die Adresse der fünf Bytes des Fließkommawertes, und bei Strings ist es die Adresse eines String-Deskriptors.

Der String-Deskriptor besteht aus den drei folgenden Bytes: der Länge des Strings und einem 2-Byte-Wert, der Adresse des Strings. Soll von einem Maschinenprogramm aus ein Wert oder String zurückgeliefert werden, so muß deshalb die Adresse der zu ändernden Variablen übergeben werden.

Alle bei CALL oder RSX überreichten Parameter werden durch Kommata getrennt.

Die bisher beschriebenen Tatsachen für den CALL-Befehl treffen auch für RSX zu. Der einzige Unterschied ist, daß CALL Adresse, Parameter1, P2, P3 durch den senkrechten Strich | (SHIFT @) mit dem Befehlsnamen, Parameter1, P2, P3 ersetzt wird. Es wird also keine Adresse mehr angegeben.

Das Einbinden der Systemerweiterung wird durch das Betriebssystem unterstützt. Man geht folgendermaßen vor:

1. Das BC-Registerpaar erhält die Adresse der einzubindenden Befehlstabelle. Die Tabelle beginnt mit der Adresse der Befehlsnamen. Auf diese Adresse folgen die Sprünge zu den einzelnen Befehlsroutinen.

   Die Befehlsnamen müssen in Großbuchstaben im Speicher stehen. Das letzte Byte jedes Befehls muß im höchstwertigen Bit (MSB) gesetzt sein, um das Ende des Namens zu kennzeichnen.

   Nach dem letzten der Befehlsnamen muß mindestens ein Byte mit dem Wert Null folgen. Dies kennzeichnet das Ende der Befehlstabelle. Namenstabelle und Sprungtabelle müssen jeweils gleich viele Einträge haben.

2. Das HL-Registerpaar erhält die Adresse von vier freien Bytes, in die das Betriebssystem Eintragungen über die neuen RSX-Kommandos machen kann.

3. Dann wird die Adresse BCD1H aufgerufen, und die Routine des Betriebssystems erledigt dann den Rest.

   Die Befehlstabelle und die vier freien Bytes müssen allerdings zwischen den Adressen 4000H und BFFFH liegen, damit kein Konflikt mit den ROMs auftreten kann.

BEISPIEL EINER BEFEHLSERWEITERUNG

Jetzt soll ein kurzes Beispielprogramm die Einbindung von RSX-Kommandos am Beispiel der Routinen DEEK und DOKE (doppeltes PEEK und POKE) darstellen. Das Programm ist als Assemblerlisting mit Kommentaren versehen (Abb. 8.1).

```
                        10 *H DEEK und DOKE am CPC 464
                        20
8FB0                    30          ENT   #8FB0
8FB0                    40          ORG   #8FB0
BCD1                    50 SYSEXT:  EQU   #BCD1
8FB0  018A8F            60          LD    BC,TABELLE          ; Adresse der Befehle
8FB3  219B8F            70          LD    HL,EXTTAB           ; 4 Bytes für System
8FB6  CDD1BC            80          CALL  SYSEXT              ; Erweiterung aufrufen
8FB9  C9                90          RET                       ; zurück zum BASIC
                        100
8FBA  928F              110 TABELL: DEFW  NAMEN               ; Adresse der Namen
8FBC  C39F8F            120         JP    DOKE                ; Sprung zur Dokeroutine
8FBF  C3B28F            130         JP    DEEK                ; Sprung zur Deekroutine
8FC2  444F4BC5          140 NAMEN:  DEFB  "D","O","K","E"+#80 ; Namen mit Bit 7 vom
8FC6  444545CB          150         DEFB  "D","E","E","K"+#80 ; letzten Byte gesetzt
8FCA  00                160         DEFB  #00                 ; Ende der Tabelle
8FCB                    170 EXTTAB: DEFS  4                   ; 4 Bytes für System
                        180
8F9F  FE02              190 DOKE:   CP    2                   ; 2 Argumente ?
8FA1  C0                200         RET   NZ                  ; sonst zurück
8FA2  DD5601            210         LD    D,(IX+1)            ;Highbyte des 2.Argument
8FA5  DD5E00            220         LD    E,(IX+0)            ; Lowbyte des 2.Argument
8FA8  DD6603            230         LD    H,(IX+3)            ; Highbyte der Adresse
8FAB  DD6E02            240         LD    L,(IX+2)            ; Lowbyte der Adresse
8FAE  73                250         LD    (HL),E              ; Argument --> Speicher
8FAF  23                260         INC   HL                  ; Ziel=Ziel+1
8FB0  72                270         LD    (HL),D              ; Argument --> Speicher
8FB1  C9                280         RET                       ; zurück ins Basic
                        290
8FB2  FE02              300 DEEK:   CP    2                   ; 2 Argumente ?
8FB4  C0                310         RET   NZ                  ; sonst zurück
8FB5  DD6601            320         LD    H,(IX+1)            ; Variablenadresse
8FB8  DD6E00            330         LD    L,(IX+0)            ; ins HL Register
8FBB  DD5603            340         LD    D,(IX+3)            ; Speicheradresse
8FBE  DD5E02            350         LD    E,(IX+2)            ; ins DE Register
8FC1  1A                360         LD    A,(DE)              ; 1. Byte von Quelle
8FC2  77                370         LD    (HL),A              ; in Variable kopieren
8FC3  13                380         INC   DE                  ; nächste Quelladresse
8FC4  23                390         INC   HL                  ; nächste Zieladresse
8FC5  1A                400         LD    A,(DE)              ; 2. Byte von Quelle
8FC6  77                410         LD    (HL),A              ; in Variable kopieren
8FC7  C9                420         RET                       ; zurück ins Basic
```

Abb. 8.1: DEEK und DOKE

Die Prozedur DOKE wird wie folgt verwendet: Aufruf von RSX mit zwei Parametern. Der erste Parameter ist die Zieladresse, und der zweite Parameter ist der Integer-Wert, der in die zwei durch den ersten Parameter bestimmten Speicherzellen abgelegt werden soll. DOKE kopiert den als zweiten Parameter angegebenen Integer-Wert an die durch den ersten bestimmte Adresse.

Die Funktion DEEK benötigt ebenfalls zwei Parameter. Der erste gibt die Quelladresse an, und der zweite muß die Adresse einer Integervariablen sein, ist also mit @ Variable anzugeben. DEEK kopiert die beiden durch den ersten Parameter adressierten Bytes in die Integer-Variable, wo sie weiter verwendet werden können.

Abb. 8.2: BASIC-Lader

```
25000 ' Dies ist der BASIC-Lader zu den beiden Systemer
weiterungen
25010 ' DOKE und DEEK, die ein doppeltes POKE und PEEK
darstellen.
25020 ' DOKE wird mit SHIFT'at'DOKE,Adresse,Wert aufger
ufen.
25030 ' Daraufhin wird Adresse mit dem Wert MOD 256 und
25040 ' Adresse+1 mit dem Wert DIV 256 geladen.
25050 '"DEEK benötigt ebenfalls zwei Argumente, nämlich
 die Adresse
25060 ' des auszulesenden Bytepaares und die Adresse ei
ner Integer-
25070 ' variablen ( mit 'at'Variable ). Bei DEEK wird d
as Byte von
25080 ' Adresse nach Lowbyte der Variablen und von Adre
sse+1 nach
25090 ' Highbyte der Variablen kopiert.
25100 ' Die beiden Systemerweiterungen werden durch CAL
L &8F80
25110 ' in das Betriebssystem eingebunden.
25120 ' Die DEEKroutine läßt sich allerdings einfacher
in
25130 ' Basic realisieren . Dazu wird sie mit DEF FNdee
k
25140 ' als Funktion unter Basic definiert .
25150 ' Die Definition lautet :
25160 '
25170 DEF FNdeek(adresse)=PEEK(adresse)+PEEK(adresse+1)
*256
25180 '
25190 ' Man sieht , die Definition von DEEK unter Basic
 und
25200 ' die Anwendung ist einfacher . Man kann nähmlich
25210 ' einer Variablen direkt den Wert zuweisen .
25220 ' Folgendes Beispiel :
25230 '
25240 Wert =FNdeek(&170)
25250 '
25260 ' Die Variable Wert müsste nun die Länge der erst
en
25270 ' Basiczeile enthalten . Die Zeile 25000 ist
25280 ' 66 Zeichen lang , bitte vergleichen Sie .
25290 ' DOKE ist allerdings wieder leichter als System-
25300 ' erweiterung zu handhaben .
25310 '
25500 MEMORY &8F7F
25510 FOR n%=&8F80 TO &8FC7
25520     READ byte$
25530     byte=VAL("&"+byte$)
25540     POKE n%,byte
25550     summe=summe+byte
25560 NEXT n%
```

```
25570 IF summe<>&1F50 THEN PRINT" DOKE & DEEK nicht kor
rekt geladen !!!":END
25580 DATA 01,8A,8F,21,9B,8F,CD,D1,BC,C9,92,8F,C3,9F,8F
,C3,B2,8F
25590 DATA 44,4F,4B,C5,44,45,45,CB,00,00,00,00,00,FE,02
,C0
25600 DATA DD,56,01,DD,5E,00,DD,66,03,DD,6E,02,73,23,72
,C9
25610 DATA FE,02,C0,DD,66,01,DD,6E,00,DD,56,03,DD,5E,02
,1A
25620 DATA 77,13,23,1A,77,C9
```

Abb. 8.2: BASIC-Lader (Fortsetzung)

Anhang A
BASIC-Tokens

BASIC-Befehl	Sedezimales Token	Dezimales Token
ABS	&FF, &00	255, 0
AFTER	&80	128
AND	&FA	250
ASC	&FF, &01	255, 1
ATN	&FF, &02	255, 2
AUTO	&81	129
BIN$	&FF, &71	255, 113
BORDER	&82	130
CALL	&83	131
CAT	&84	132
CHAIN	&85	133
CHR$	&FF, &03	255, 3
CINT	&FF, &04	255, 4
CLEAR	&86	134
CLG	&87	135
CLOSEIN	&88	136
CLOSEOUT	&89	137
CLS	&8A	138
CONT	&8B	139
COS	&FF, &05	255, 5
CREAL	&FF, &06	255, 6
DATA	&8C	140
DEC$	&FF, &72	255, 114
DEF	&8D	141
DEFINT	&8E	142
DEFREAL	&8F	143
DEFSTR	&90	144
DEG	&91	145
DELETE	&92	146
DI	&DB	219
DIM	&93	147
DRAW	&94	148
DRAWR	&95	149
EDIT	&96	150

BASIC-Befehl	Sedezimales Token	Dezimales Token
EI	&DC	220
ELSE	&97	151
END	&98	152
ENT	&99	153
ENV	&9A	154
EOF	&FF, &40	255, 64
ERASE	&9B	155
ERL	&E3	227
ERR	&FF, &41	255, 65
ERROR	&9C	156
EVERY	&9D	157
EXP	&FF, &07	255, 7
FIX	&FF, &08	255, 8
FN	&E4	228
FOR	&9E	158
FRE	&FF, &09	255, 9
GOSUB	&9F	159
GOTO	&A0	160
HEX$	&FF, &73	255, 115
HIMEM	&FF, &42	255, 66
IF	&A1	161
INK	&A2	162
INKEY	&FF, &0A	255, 10
INKEY$	&FF, &43	255, 67
INP	&FF, &0B	255, 11
INPUT	&A3	163
INSTR	&FF, &74	255, 116
INT	&FF, &0C	255, 12
JOY	&FF, &0D	255, 13
KEY	&A4	164
LEFT$	&FF, &75	255, 117
LEN	&FF, &0E	255, 14
LET	&A5	165
LINE	&A6	166
LIST	&A7	167
LOAD	&A8	168
LOCATE	&A9	169
LOG	&FF, &0F	255, 15
LOG10	&FF, &10	255, 16
LOWER$	&FF, &11	255, 17
MAX	&FF, &76	255, 118

BASIC-Befehl	Sedezimales Token	Dezimales Token
MEMORY	&AA	170
MERGE	&AB	171
MID$	$AC	172
MIN	&FF, &77	255, 119
MOD	&FB	251
MODE	&AD	173
MOVE	&AE	174
MOVER	&AF	175
NEXT	&B0	176
NEW	&B1	177
NOT	&FE	254
ON	&B2	178
ON BREAK	&B3	179
ON ERROR GOTO	&B4	180
ON SQ	&B5	181
OPENIN	&B6	182
OPENOUT	&B7	183
OR	&FC	252
ORIGIN	&B8	184
OUT	&B9	185
PAPER	&BA	186
PEEK	&FF, &12	255, 18
PEN	&BB	187
PI	&FF, &44	255, 68
PLOT	&BC	188
PLOTR	&BD	189
POKE	&BE	190
POS	&FF, &78	255, 120
PRINT	&BF	191
RAD	&C1	193
RANDOMIZE	&C2	194
READ	&C3	195
RELEASE	&C4	196
REM	&C5	197
REMAIN	&FF, &13	255, 19
RENUM	&C6	198
RESTORE	&C7	199
RESUME	&C8	200
RETURN	&C9	201
RIGHT$	&FF, &79	255, 121
RND	&FF, &45	255, 69
ROUND	&FF, &7A	255, 122
RUN	&CA	202

BASIC-Befehl	Sedezimales Token	Dezimales Token
SAVE	&CB	203
SGN	&FF, &14	255, 20
SIN	&FF, &15	255, 21
SOUND	&CC	204
SPACE$	&FF, &16	255, 22
SPC	&E5	229
SPEED	&CD	205
SQ	&FF, &17	255, 23
SQR	&FF, &18	255, 24
STEP	&E6	230
STOP	&CE	206
STR$	&FF, &19	255, 25
STRING$	&FF, &7B	255, 123
SWAP	&E7	231
SYMBOL	&CF	207
TAB	&EA	234
TAG	&DQ	208
TAGOFF	&D1	209
TAN	&FF, &1A	255, 26
TEST	&FF, &7C	255, 124
TESTR	&FF, &7D	255, 125
THEN	&EB	235
TIME	&FF, &46	255, 70
TO	&EC	236
TROFF	&D2	210
TRON	&D3	211
UNT	&FF, &1B	255, 27
UPPER$	&FF, &1C	255, 28
USING	&ED	237
VAL	&FF, &1D	255, 29
VPOS	&FF, &7F	255, 127
WAIT	&D4	212
WEND	&D5	213
WHILE	&D6	214
WIDTH	&D7	215
WINDOW	&D8	216
WRITE	&D9	217
XOR	&FD	253
XPOS	§FF, &47	255, 71
YPOS	&FF, &48	255, 72
ZONE	&DA	218

BASIC-Befehl	Sedezimales Token	Dezimales Token
>	&EE	238
=	&EF	239
>= o. =>	&F0	240
<	&F1	241
< >	&F2	242
<=	&F3	243
+	&F4	244
-	&F5	245
*	&F6	246
/	&F7	247
↑	&F8	248
\	&F9	249
'(REM)	&C0	192

Anhang B
BASIC-Befehle mit Parametern

Obwohl die Beschreibung der BASIC-Befehle im Handbuch als sehr gut zu bezeichnen ist, fehlt für das schnelle Nachschlagen doch eine Kurzübersicht, was wir hiermit nachholen.

A

ABS (X)	Absolutwert bilden
AFTER IN, UHR GOSUB Zeile	Periodischer Ablauf eines Unterprogramms
AND	UND-Verknüpfung
ASC (X$)	ASCII-Wert des ersten Zeichens
ATN (X)	Arcus-Tangens
AUTO Zeile, Schritt	Automatische Zeilennumerierung

B

BIN$ X,Stellen	Umrechnung in Binär
BORDER F1,F2	Rahmenfarbe

C

CALL Adresse, Parameterliste	Aufruf eines Maschinenprogramms
CAT	Inhaltsverzeichnis
CHAIN Name, Zeile	Neues Programm laden
CHAIN MERGE Name, Zeile	DELETE Zeilenbereich; anderes Programm hinzuladen
CHR$ (X)	Zeichen aus Tabelle
CINT (X)	Gerundeter ganzzahliger Wert
CLEAR	Variablen löschen
CLG PEN	Grafikschema mit Farbe füllen

CLOSEIN	Eingabedatei schließen
CLOSEOUT	Ausgabedatei schließen
CLS #Nummer	Bildschirmbereich löschen
CONT	Programm fortsetzen
COS (X)	Kosinus
CREAL (X)	Umwandlung in reale Zahl

D

DATA	Dateneingabe in Programmen
DEC$((X,Y$)	Formatiert X mit „USING" Y$ zu einem String *(Achtung:* 2× Klammer auf; 1× Klammer zu).
DEF FN Name = Ausdruck	Definieren einer Funktion
DEFINT Bereich	Variablentyp festlegen (Integer)
DEFSTR Bereich	Variablentyp festlegen (String)
DEFREAL Bereich	Variablentyp festlegen (Real)
DEG	Umschalten auf Winkelmaß
DELETE Zeilenbereich	Programmzeilen löschen
DI	Unterbrechungen verhindern (Disable Interrupts)
DIM Variablenliste	Indizierte Variablen (Felder, Arrays) dimensionieren
DRAW X,Y,PEN	Linie ziehen (absolut)
DRAWR X,Y,PEN	Linie ziehen (relativ)

E

EDIT Zeile	Programmzeile bearbeiten
EI	Unterbrechungen ermöglichen (Enable Interrupts)
END	Programmende
ENT Nummer, Veränderungsfolge	Tonhöhe variieren
ENV Nummer, Veränderungsfolge	Tonlautstärke variieren
EOF	Funktion erfragt Dateiende (−1)
ERASE Variablenliste	Löscht Variablen (nur indizierte)
ERR	Fehlernummer
ERL	Fehlerzeile
ERROR X	Erzeugt Fehler mit der Nummer X
EVERY X,X GOSUB Zeilennummer	Unterprogramm periodisch aufrufen
EXP (X)	e-Funktion

F

FIX (X)	Dezimalstellen abschneiden
FOR I = A TO E STEP S	Programmschleifenbeginn
FRE (X)	Freier Speicher für BASIC
FRE (" ")	Garbage Collection (Speicherbereinigung)

G

GOSUB Zeile	Unterprogrammaufruf
GOTO Zeile	Programmsprung

H

HEX$ X,Y	Umrechnung in Hexadezimal
HIMEM	Letzte Speicheradresse vom BASIC

I

IF Bedingung THEN Anweisung ELSE Anweisung	Programmverzweigung
INK Pen, F1, F2	Farbenzuordnung
INKEY (X)	Gedrückte Taste
INKEY$	Wert der gedrückten Taste
INP Adresse	Wert an Eingabeschnittstelle
INPUT	Datenerfassung
INSTR (X, T1$, T2$)	String-in-String-Suche
INT (X)	Nächstkleinerer gerundeter ganzzahliger Ausdruck

J

JOY (Nummer)	Joystickwert

K

KEY, Nummer, Text	Funktionstaste belegen
KEY DEF	Tastenzuordnung ändern

L

LEFT$ (X$,X)	Linker Teil einer Zeichenkette
LEN (X$)	Länge einer Zeichenkette
LET Ausdruck	Zuweisung
LINE INPUT. . .	Zeile von Gerät einlesen
LIST. . .	Programm anzeigen
LOAD Name, Adresse	Programm laden
LOCATE #Nummer,X,Y	Cursor positionieren
LOG (X)	Natürlicher Logarithmus
LOG10 (X)	Dekadischer Logarithmus
LOWER$ (X$)	Umwandlung in Kleinschrift

M

MAX (A,B,C,. . .)	Maximum der Elemente
MEMORY Adresse	BASIC-Bereich begrenzen
MERGE Name	Programm mit vorhandenem mischen
MID$ (X$,X,Y)	Mittelteil einer Zeichenkette
MIN (A,B,C,. . .)	Minimum der Elemente
MOD	Modulo-Funktion
MODE Nummer	Bildschirmmodus ändern
MOVE X, Y	Grafikcursor positionieren (absolut)
MOVER X, Y	Grafikcursor positionieren (relativ)

N

NEW	Programm löschen
NEXT	Ende einer Programmschleife

O

ON X GOSUB Z1,Z2,. . .	Programmverzweigung in Unterprogramme
ON X GOTO Z1,Z2,. . .	Programmverteiler durch Sprünge
ON BREAK GOSUB Zeile	Unterlaufprogrammaufruf bei zweimaligem ESC
ON BREAK STOP	Programmabbruch bei zweimaligem ESC
ON ERROR GOTO Zeile	Sprung, wenn Fehler
ON SQ Kanal GOSUB	Sprung bei Platz in Kanal
OPENIN Name	Eingabedatei öffnen

OPENOUT Name	Ausgabedatei öffnen
OR	ODER-Verknüpfung
ORIGIN X,Y,L,R,O,U	Ursprung des Grafikcursors
OUT Integer-Adresse, X	Übergabe von X (0-255) an Interface

P

PAPER #Nummer, INK	Papierfarbe setzen
PEEK (Adresse)	Inhalt einer Adresse auslesen
PEN #Nummer, INK	Schriftfarbe setzen
PI	$\pi = 3.14159265...$
PLOT X,Y,Farbe	Punkt ausgeben (absolut)
PLOTR X,Y,Farbe	Punkt ausgeben (relativ)
POKE Adresse, X	Wert an Adresse übergeben
POS (#Nummer)	X-Position des Cursors
PRINT (USING)	Datenausgabe

R

RAD	Umschalten auf Bogenmaß
RANDOMIZE X	Anfangswert Zufallsgenerator
READ Variablenliste	Daten aus DATA-Anweisungen lesen
RELEASE Kanäle	Wartezustand aufheben
REM	Kommentar
REMAIN (X)	Restzeit eines Timers
RENUM...	Umnumerieren des Programms
RESTORE Zeile	Zeiger für DATA-Anweisungen zurücksetzen
RESUME Zeile	RETURN für ON...ERROR
RESUME NEXT	GOTO
RETURN	Rücksprung aus Unterprogramm
RIGHT$ (X$,X)	Rechter Teil einer Zeichenkette
RND (X)	Nächste Zufallszahl
ROUND (X,Y)	Stellengenaues Runden
RUN Text	Programm laden und starten
RUN Zeile	Definierter Programmstart

S

SAVE...	Programm speichern
SGN (X)	Vorzeichen feststellen

SIN (X)	Sinus
SOUND. . .	Ton erzeugen
SPACE$(X)	String aus Leerzeichen
SPC(X)	Während PRINT werden X Leerzeichen erzeugt
SPEED INK X,Y	Geschwindigkeit bei Farbwechsel
SPEED KEY Verzögerung, Periode	REPEAT-Werte einstellen
SPEED WRITE X	Schreibgeschwindigkeit für Kassette
SQ (Kanal)	Zahl der freien Plätze in Tonwarteschlange
SQR (X)	Wurzelberechnung
STOP	Programmabbruch
STR$ (X)	Umwandlung in Zeichen
STRING$ (X,X$)	String aus X$ Zeichen
SYMBOL Zeichen	Neubelegung eines Zeichens
SYMBOL AFTER X	Bereich für benutzereigene Zeichen

T

TAB(X)	Während PRINT wird Cursor an Position X gesetzt
TAG #Nummer	Zeichenausgabe auf Grafikcursorposition ermöglichen
TAGOFF #Nummer	TAG ausschalten
TAN (X)	Tangens
TEST (X, Y)	Farbstiftnummer der Koordinate (absolut)
TESTR (X,Y)	Farbstiftnummer der Koordinate (relativ)
TIME	Zeit seit Einschalten des Rechners in 1/300 Sekunden
TRON	Trace einschalten
TROFF	Trace ausschalten

U

UNT (X)	Ganzzahligen Wert ausgeben
UPPER$ (X$)	Umwandlung in Großbuchstaben

V

VAL (X$)	Wert der Zeichenkette ausgeben
VPOS(#Nummer)	Y-Position des Textcursors

W

WAIT Integer, AND,XOR	Wartebefehl
WEND	Ende einer WHILE-Schleife
WHILE Bedingung	Wiederholfunktion
WIDTH X	Spaltenbegrenzung bei Druckerausgabe
WINDOW #Nummer,L,R,O,U	Fenster festlegen
WINDOW SWAP Nummer,Nummer	Bildschirmbereiche austauschen
WRITE #Nummer, Liste	Werte ausgeben

X

XOR	EXCLUSIV-ODER-Verknüpfung
XPOS	X-Koordinate des Grafikcursors

Y

YPOS	Y-Koordinate des Grafikcursors

Z

ZONE X	Druckzonen verändern

Anhang C

Übersicht der in den Variablenübersichten verwendeten Abkürzungen und Bedeutungen

Art einer Variablen:

A - Ausgabeparameter (von Unterprogramm an Hauptprogramm)
E - Eingabeparameter (von Hauptprogramm an Unterprogramm)
G - Globale Variable (der Bedeutung nach)
H - Hilfsvariable (temporäre Bedeutung)
L - Lokale Variable (wird nur im Unterprogramm benutzt)
R - Rückgabeparameter (von Unterprogramm einer tieferen Ebene an Hauptprogramm)
T - Transianter Parameter (von Hauptprogramm an Unterprogramm einer tieferen Ebene)

Bereich:

Hier wurde der jeweils gültige Bereich für die Variablen eingetragen, was beim Testen sehr hilfreich sein kann (liegt aktueller Wert im gültigen Bereich?). Weiterhin ergibt sich darauf eine bessere Aussagekraft der Variablen.

Für Felder (Arrays) wurde - durch / getrennt - auch der Bereich der *gültigen* Indizes aufgeführt. Wenn z. B. der Index 0 nicht benutzt wird, erscheint 1. . .20.

Stichwortverzeichnis

Die SYBEX Bibliothek

Einf. Literatur

MEIN HEIMCOMPUTER
von N. Hesselmann – zeigt, was ein Heimcomputer ist und was man mit ihm anfangen kann, von den Chips bis zu Tips für den Kauf. 256 Seiten, 124 Abb., Best.-Nr. **3064** (1985)

EINFÜHRUNG IN DIE TEXTVERARBEITUNG
von Hal Glatzer – beschreibt, woraus eine Textverarbeitungsanlage besteht, wie man sie nutzen kann und wozu sie fähig ist. Beispiele verschiedener Anwendungen und Kriterien für den Kauf eines Systems. 248 Seiten, 67 Abbildungen, Best.-Nr. **3018** (1983)

SYBEX MIKROCOMPUTER LEXIKON
– die schnelle Informationsbörse! Über 1500 Definitionen, Kurzformeln, Begriffsschema der Mikroprozessor-Technik, englisch/deutsches und französisch/deutsches Wörterbuch, Bezugsquellen. 192 Seiten, Format 12,5 x 18 cm, Best.-Nr.: **3035** (1984)

COMPUTER TOTAL VERRÜCKT
von Daniel Le Noury – mit diesem Buch kommen Sie wieder zur Besinnung, nachdem Sie sich halbtot gelacht haben. Ca. 100 Cartoons rund um den Computer. 96 Seiten, Best.-Nr. **3042** (1984)

Sprachen
BASIC

BASIC COMPUTER SPIELE/Band 1
herausgegeben von David H. Ahl – die besten Mikrocomputerspiele aus der Zeitschrift „Creative Computing" in deutscher Fassung mit Probelauf und Programmlisting. 208 Seiten, 56 Abbildungen, Best.-Nr. **3009**

BASIC COMPUTER SPIELE/Band 2
herausgegeben von David H. Ahl – 84 weitere Mikrocomputerspiele aus „Creative Computing". Alle in Microsoft-BASIC geschrieben mit Listing und Probelauf. 224 Seiten, 61 Abbildungen, Best.-Nr.: **3010**

BASIC PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Bibliothek von Programmen zu den wichtigsten Problemlösungen mit numerischen Verfahren, alle in BASIC geschrieben, mit Musterlauf und Programmlisting. 352 Seiten, 147 Abbildungen, Best.-Nr.: **3015** (1983)

PLANEN UND ENTSCHEIDEN MIT BASIC
von X. T. Bui – eine Sammlung von interaktiven, kommerziell-orientierten BASIC-Programmen für Management- und Planungsentscheidungen. 200 Seiten, 53 Abbildungen, Best.-Nr.: **3025** (1983)

GRUNDKURS IN BASIC
von U. Ströbel – die Einführung in die meistgenutzte Programmiersprache für Lehrer, Schüler und das Selbststudium (Reihe SYBEX Informatik). 208 Seiten, mit Abb., Best.-Nr. **3058** (1985), Lehrerbegleitheft Best.-Nr. **3091** (1985)

Pascal

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen. 480 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3005** (1982)

50 PASCAL-PROGRAMME
von B. Hunter – eine kommentierte Sammlung nützlicher Programme für Anwendungen im Geschäft und Privatbereich, für mathematische Anwendungen oder Spiele. Ca. 340 S., ca. 48 Abb., Best.-Nr. **3065** (erscheint 1985)

GRUNDKURS IN PASCAL Bd. 1
von K.-H. Rollke – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

GRUNDKURS IN PASCAL BAND 2
von K.H. Rollke – Mit diesem Buch wird der Pascal-Grundkurs aus der Reihe SYBEX Informatik abgerundet. Für Lehrer, Schüler, Teilnehmer an Pascal-Kursen, Studenten und Autodidakten. 224 Seiten, mit Abb., Best.-Nr. **3061** (1985), Lehrerbegleitheft Best.-Nr. **3090**

Assembler

PROGRAMMIERUNG DES Z80
von Rodnay Zaks – ein umfassendes Nachschlagewerk zum Z80-Mikroprozessor – jetzt in einer durch Lösungen ergänzten Ausgabe. 2., erweiterte Ausgabe. 656 Seiten, 176 Abbildungen, Best.-Nr.: **3099** (1985)

Z80 ANWENDUNGEN
von J. W. Coffron – vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren. 296 Seiten, 204 Abbildungen, Best.-Nr.: **3037** (1984)

PROGRAMMIERUNG DES 6502 mit 6510/65C02/65SC02
von Rodnay Zaks – Programmierung in Maschinensprache mit dem Mikro-Prozessor 6502 und anderen Mitgliedern der 65xx Familie, von den Grundkonzepten bis hin zu fortgeschrittenen Informationsstrukturen. 3. überarbeitete und erweiterte Ausgabe. Ca. 450 Seiten, ca. 170 Abbildungen, Best.-Nr.: **3600** (1985)

Spezielle Geräte

Apple

APPLE II BASIC HANDBUCH
von D. Hergert – ein handliches Nachschlagewerk, das neben Ihren Apple II, II+ oder IIe stehen sollte. Dank vieler Tips und Vorschläge eine wesentliche Erleichterung fürs Programmieren. 304 Seiten, 116 Abbildungen, Best.-Nr. **3036** (1984)

WAS IST WO IM APPLE
von William F. Luebbert – eine systematische Arbeitshilfe für Besitzer eines Apple II, II+ oder IIe mit vielen alphabetisch und numerisch sortierten Apple-Routinen und Speicheradressen, die Ihnen hilft, viel Arbeit und Zeit zu sparen. 496 Seiten, mit 84 Abb., Best.-Nr. **3098** (erscheint 1985)

APPLE II/IIe ASSEMBLER KURS
Reihe MISTER MICRO – Assembler-Programmierung auf dem Apple leicht gemacht. Das Buch vermittelt alle Instruktionen für den 6502-Prozessor. Der Assembler kann jederzeit für eigene Programme eingesetzt werden. 240 Seiten, Buch und Diskette, Best.-Nr. **3408** (1984)

ARBEITEN MIT DEM MACINTOSH
von N. Hesselmann – alles über den leistungsfähigen Apple-Rechner mit einer Erläuterung wichtiger kommerzieller Software-Pakete und deren Einsatz, Anleitung zur Programmierung in Microsoft-BASIC. Viele konkrete Anwendungs-Beispiele. 416 Seiten, 320 Abb., Best.-Nr. **3080** (1984)

Atari

ATARI BASIC HANDBUCH
von J. Reschke – das vollständige ABC der BASIC-Programmierung für den Atari mit Erläuterung durch viele praktische Beispiele. 208 Seiten, mit Abb., Best.-Nr. **3083** (1984)

ATARI PROGRAMM-SAMMLUNG
von S. R. Trost – sollte neben keinem Atari-Computer fehlen. Es bietet einen Satz ausgetesteter Programme für eine Fülle von Anwendungen. 190 S., 150 Abb., Best.-Nr. **3068** (1984)

SPIELEN, LERNEN, ARBEITEN MIT DEM ATARI
von G.-P. Raabe/K.-J. Schmidt – viele Beispielprogramme, eine umfassende Bedienungs-Anleitung für den Atari und eine Einführung in die BASIC-Programmierung. 280 Seiten, 56 Abb., Best.-Nr. **3082** (1985)

Commodore

COMMODORE 64 BASIC HANDBUCH
von D. Hergert – zeigt Ihnen alle Anwendungsmöglichkeiten Ihres C64 und beschreibt das vollständige BASIC-Vokabular anhand von praktischen Beispielen. 208 Seiten, 92 Abbildungen, Best.-Nr.: **3048** (1984)

FARBSPIELE MIT DEM COMMODORE 64
von W. Black und M. Richter – 20 herrliche Farbspiele für Ihren C64, mit Beschreibung, Programmlisten und Bildschirm-Darstellungen. Für mehr Freizeit-Spaß mit Ihrem Commodore! 176 Seiten, 58 Abbildungen, Best.-Nr.: **3044** (1984)

COMMODORE 64 – GRAFIK + DESIGN
von Ch. Platt – Eine Schritt-für-Schritt-Einführung in die Grafik-Programmierung Ihres C 64. Tips, die Sie in keinem Handbuch finden. 280 S., 150 Abb., teils vierfarbig. Best.-Nr. **3073** (1984)

COMMODORE 64 PROGRAMMSAMMLUNG
von S. R. Trost – mehr als 70 getestete Anwenderprogramme, die direkt eingegeben werden können. Erläuterungen gewährleisten eine optimale Nutzung. 192 Seiten, 160 Abbildungen, Best.-Nr.: **3051** (1983)

SPASS AN MATHE MIT DEM COMMODORE 64
von H. Danielsson – zeigt Ihnen mit vielen Beispielen, wie der C 64 für schulische oder private Berechnungen genutzt werden kann. 280 Seiten mit Abb., Best.-Nr. **3072** (1985)

COMMODORE 64 BASIC-KURS MIT HONEY-AID
Reihe MISTER MICRO – BASIC auf dem C64 durch Praxis lernen; mit dem integrierten Lernpaket (Buch + Software). Außer vielen Übungsprogrammen: Honey-Aid – eine universell einsetzbare BASIC-Erweiterung mit 28 zusätzlichen Befehlen. 352 Seiten, Buch und Kassette, Best.-Nr. **3400**, Buch und Diskette, Best.-Nr. **3401** (1984)

COMMODORE 64 ASSEMBLER-KURS
Reihe MISTER MICRO – zeigt in Theorie und Praxis, wie Sie den 6510-Prozessor Ihres C64 programmieren. Der mitgelieferte Assembler ist universell einsetzbar. 296 Seiten, Buch und Kassette, Best.-Nr. **3402**, Buch und Diskette, Best.-Nr. **3403** (1984)

Sinclair

SINCLAIR ZX SPECTRUM BASIC HANDBUCH
von D. Hergert – eine wichtige Hilfe für jeden SPECTRUM-Anwender. Gibt eine Übersicht aller BASIC-Begriffe, die auf diesem Rechner verwendet werden können, und erläutert sie ausführlich anhand von Beispielen. 288 Seiten, 188 Abbildungen, Best.-Nr.: **3027** (1983)

SINCLAIR ZX SPECTRUM Programme zum Lernen und Spielen
von T. Hartnell – ein Buch zur praktischen Anwendung. Grundzüge des Programmierens aus dem kaufmännischen Bereich sowie Spiele, Lehr- und Lernprogramme in BASIC. 232 Seiten, 140 Abbildungen, Best.-Nr. **3022** (1983)

SPECTRUM BASIC ABENTEUER
Der fremde Planet
Reihe MISTER MICRO – Kinder und Jugendliche lernen mit dem kompletten Lernpaket auf spielerische Weise, den Spectrum in BASIC zu programmieren. 128 Seiten, Buch und Kassette, Best.-Nr. **3410** (1984)

SPECTRUM BASIC KURS
Reihe MISTER MICRO – BASIC auf dem Spectrum schnell gelernt mit dem integrierten Lernpaket, das die Programmierung in Theorie und Praxis vermittelt. 312 Seiten, Buch und Kassette, Best.-Nr. **3409** (1984)

Systemsoftware

CP/M-HANDBUCH
on Rodnay Zaks – das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M., 2. überarbeitete Ausgabe. 356 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

PROGRAMMIEREN MIT CP/M
von A. R. Miller – vermittelt die Feinheiten von CP/M und hilft, die Möglichkeiten dieses populären Betriebssystems zu erweitern. 424 Seiten, 97 Abb., Best.-Nr. **3077** (1985)

Anwendungssoftware

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

PRAKTISCHE WORDSTAR-ANWENDUNGEN
von J. A. Arca – das Buch für Einsteiger, um nach kurzer Zeit praktische Textverarbeitungs-Probleme zu lösen, eine programmierte Unterweisung zur Leistungsoptimierung mit WORDSTAR. 368 Seiten, 69 Abbildungen, Best.-Nr.: **3057** (1985)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

Fordern Sie ein Gesamtverzeichnis unserer Verlagsproduktion an:

SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (02 11) 62 64 41
Telex: 8 588 163

SYBEX INC.
2344 Sixth Street
Berkeley, CA 94710, USA
Tel.: (415) 848-8233
Telex: 287639 SYBEX UR

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX

Arbeiten mit dem Schneider CPC

Das wird nach der Lektüre dieses Buches kein Problem mehr für Sie sein. Es wurde für alle geschrieben, die den Schneider CPC für ihre tägliche Arbeit im Büro oder zu Hause, aber auch für ihr Hobby einsetzen möchten. Der Autor hat eine Fülle unentbehrlicher Informationen so aufbereitet, daß Ihnen die Umsetzung eines Problems in eine computergerechte Lösung sehr leicht gemacht wird.

Arbeiten mit dem Schneider CPC bietet Ihnen u.a.:

- Allgemeine Routinen, die immer wieder benötigt werden, wie z. B. Sortieren und Selektieren;
- Einsatzmöglichkeiten und Änderung des Zeichensatzes;
- Hochauflösende Grafik und ihre Anwendung mit Joystick, als Diagramm und für Schaltpläne. Dabei werden auch Routinen zur Darstellung von Sprites und Hardcopies vorgestellt;
- Eine Dateiverwaltung;
- Einen Sound-Editor und Möglichkeiten zur Programmierung von Musik und Geräuschen;
- Eine Übersicht der wichtigsten Betriebssystem-Routinen;
- Vorgehensweise bei BASIC-Erweiterungen und
- Viele Tips und Tricks für Fortgeschrittene.

Arbeiten mit dem Schneider CPC ist eine leicht verständliche und umfassende Arbeitshilfe, die Ihnen alles Wissenswerte vermittelt, um Ihren leistungsstarken Rechner von Beginn an effizient zu benutzen.

Über den Autor

Hans Lorenz Schneider studierte an der Bundeswehrhochschule in München Informatik. Seine Diplomarbeit schrieb er am bereits legendären PET. Seit 1980 ist H. L. Schneider Inhaber eines Software-Hauses, in dem er individuelle Programme für Personal-Computer anbietet. Seine Erfahrungen publiziert er seit 1982 in Büchern und zahlreichen Fachzeitschriften-Artikeln.

ISBN 3-88745-603-3

Arbeiten mit dem Schneider CPC

Schneider

SYBEX

3603

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/