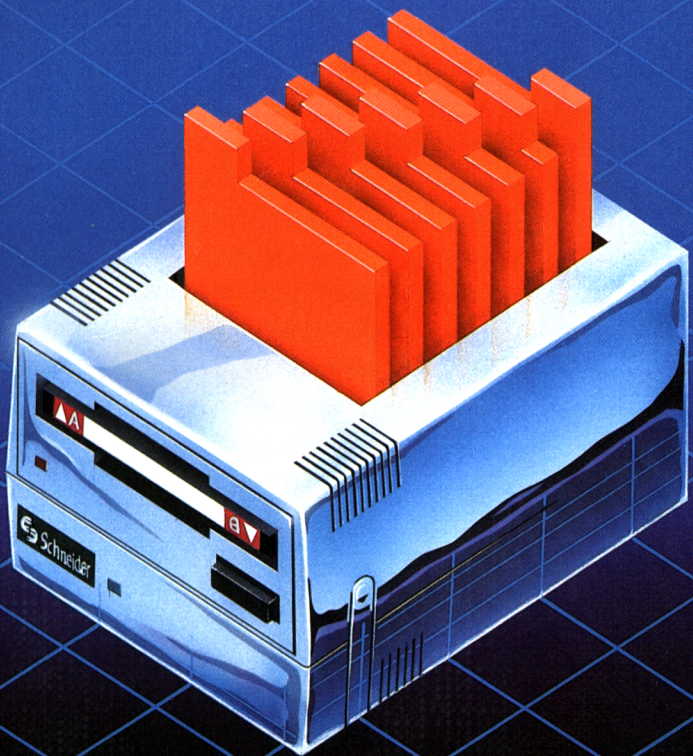
Manfred Walter Thoma

CPC-464/664 Praxis II

Band 2: Datenverwaltung



Hüthig

Manfred Walter Thoma · CPC 464/664 - Praxis II

Manfred Walter Thoma

CPC-464/664 — Praxis II

Band 2: Datenverwaltung

Dr. Alfred Hüthig Verlag Heidelberg

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Thoma, Manfred Walter:
[CPC-vierhundertvierundsechzig-Praxis]
CPC-464/664-Praxis II / Manfred Walter Thoma. —
Heidelberg : Hüthig

Bd. 2. Datenverwaltung. – 1986 –
ISBN 3-7785-1209-9

Printed in Germany
Satz und Druck: Druckerei Bitsch GmbH, Birkenau

0.0 Vorwort

Die Verwaltung von Daten jeder Art ist und bleibt das Anwendungsgebiet für den Computer überhaupt (EDV = Elektronische DatenVerarbeitung!). Der komfortable und schnelle Zugriff auf eine möglichst große Datenmenge zeichnet eine gute Datenverwaltung aus.

Daten, und das teilweise in erheblichen Mengen, treten nicht nur im kommerziellen sondern auch im privaten Bereich auf. Haushaltsbücher, Adressdateien, Lehr- und Lernprogramme, aber auch viele Spiele basieren auf dem Prinzip der Datenverarbeitung.
Aus diesem Grund bleibt es nicht aus, daß sich auch der Hobby- und Freizeitprogrammierer mit der Datenverwaltung auseinandersetzen muß.

Zur langfristigen Sicherung von Daten (Dateien) ist immer ein externes Speichermedium notwendig. Der Schneider CPC-464 mit dem eingebauten Datenrecorder oder besser mit der externen Diskettenstation DDI-1 sowie der Schneider CPC-664 mit der integrierten Diskettenstation besitzen alle Voraussetzungen für eine komfortable Verwaltung von Daten.

Anhand von fünf sehr umfangreichen und leistungsstarken Anwenderprogrammen findet in diesem Buch eine grundlegende Darstellung der wichtigsten Methoden zur Datenverwaltung statt. Neben der Behandlung von Konstanten, der sequentiellen Datei und der sogenannten „RAM-Floppy" (relative Adressierung) darf natürlich der Direktzugriff auf Daten nicht fehlen. Anhand eines sehr komfortablen Diskettenmonitor wird der Aufbau der Diskette genauer untersucht. Doch auch Such- und Sortierverfahren sowie die Handhabung und Programmierung von Index-Dateien finden hier ihren Platz.

Alle Programme gehen weit über das übliche Niveau hinaus (sie sind bis auf den Direktzugriff auch mit dem Datenrecorder nutzbar) und lassen sich in der alltäglichen Praxis mit dem Schneider CPC-464 und CPC-664 hervorragend anwenden. Die Programme brauchen durch ihre Qualität einen Vergleich mit kommerziellen Programmen nicht zu scheuen.

Ich möchte es an dieser Stelle nicht versäumen, mich für die Durchsicht des Manuskripts durch Frau Monika Ohlfest und Herrn Wolfgang Eggerichs herzlichst zu bedanken.

Hamburg, November 1985

Inhaltsverzeichnis

Programmverzeichnis

1.0 Einleitung

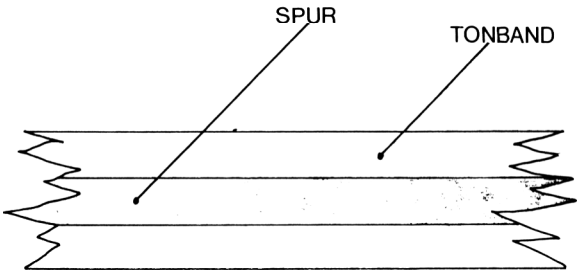
Diskette und Kassette sind zwei hervoragende und erprobte Speichermedien; und doch unterscheiden sie sich erheblich. Nicht nur in der Übertragungsgeschwindigkeit und im Komfort sind die Unterschiede zu finden, sondern vor allem in der Art und Weise, wie die Daten auf dem Datenträger abgelegt werden.

Beide Speichermedien bestehen aus einem magnetisierbaren Stoff. Jedes Bit einer Information wird in einer bestimmten Form verschlüsselt und auf die Diskette/Kassette als „magnetischer Zustand" abgelegt. Doch schon durch die äußere Form unterscheidet sich die Kassette von der Diskette:

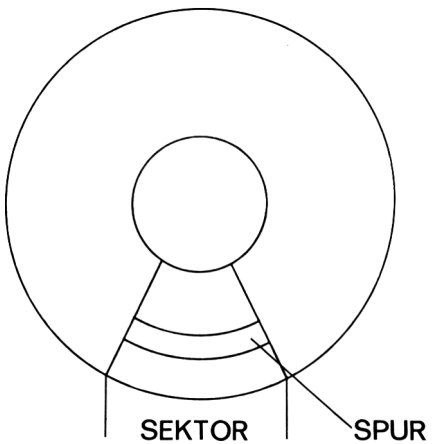
- Die Kassette besteht aus einem (Ton-) Band, d.h. die Daten müssen zwangsläufig hintereinander, Bit für Bit, abgelegt werden.
- Die Diskette ist eine runde Scheibe, die sich mit hoher Geschwindigkeit dreht (300 U/Min). Der Schreib/Lesekopf ist weiterhin beweglich und kann auf einen beliebigen Ort der „Scheibe" zugreifen. Durch diese Zugriffsmöglichkeit müssen die Daten nicht umbedingt hintereinander folgen, sondern können auf der Diskette verteilt liegen.

Damit Daten zu einem späteren Zeitpunkt auch wieder auf der Kassette/Diskette gefunden werden können, müssen sie sich an einer exakt bestimmbaren Stelle befinden.

Vom Band der Kassette wird nur ein kleiner Teil benutzt. Der Tonkopf (Schreib/Lesekopf) magnetisiert nur einen Teil des Bandes und benutzt diesen als Speicherbereich (siehe Abb. 1). Dabei ist der Ort der „Spur" abhängig vom eingestellten Tonkopf. Wenn sich der Tonkopf nicht verstellt, greift er immer wieder auf diesen Bereich zu.



Auch auf der Diskette werden bestimmte Bereiche für Daten reserviert und gekennzeichnet. Dieses Reservieren von Speicherbereichen auf der Diskette nennt sich „Formatieren". Beim Formatieren werden (bei der DDI-1 oder der eingebauten Diskettenstation im CPC-664) 40 konzentrische Ringe (Spuren oder Tracks) magnetisiert. Gleichzeitig erfolgt eine Einteilung der Spuren in 9 Blöcke (Sektoren), die jeweils 512 Bytes an Informationen aufnehmen können.



Jeder der 360 Blöcke (40 Spuren * 9 Sektoren) bildet eine zusammenhängende Speichereinheit. Zur Sicherung eines 700 Bytes großen Programms werden 2 Blöcke (2 * 512 Byte) benötigt.

Bei der *Kassette* ist ebenfalls eine Aufteilung in „Blöcke" zu finden. Beim Sichern von Daten (Programme) wird die gesamte Datenmenge in 2 kByte (2048 Bytes) große Blöcke zerlegt und hintereinander gespeichert. Ein Programm

mit 5000 Bytes besteht dann auf der Kassette aus 3 Blöcken a 2048 Bytes, da 5000 Bytes nicht in 2 Blöcke passen (2 * 2048 = 4096 Bytes). Im ersten Block einer Datei befinden sich in den ersten 64 Bytes noch zusätzliche Informationen über die Länge der Datei (wieviele Blöcke folgen), den Namen der Datei, etc.. Jedes gesicherte Programm (Datei) besitzt diesen sogenannten „Header". Beim Suchen eines bestimmten Programms auf der Kassette wird die gesamte Kassette nach einem Header durchsucht. Ist er gefunden, wird geprüft, ob dies das gewünschte Programm ist (Vergleich der Programmnamen). Wenn ja, werden alle folgenden, zum Programm gehörenden Blöcke in den Speicher des Rechners geladen. Stimmt der Programmname nicht überein, wird der nächste Header gesucht.

Da auf alle Informationen einer *Diskette* sofort zugegriffen werden kann, muß eine Diskette nicht nach einem Programm durchsucht werden. Vielmehr muß man nur wissen, wo (in welchen Blöcken) sich das gewünschte Programm auf der Diskette befindet. In einem speziellen Verzeichnis (Directory) werden alle Programme der Diskette geführt. Die Directory besteht im wesentlichen aus einer Liste der Programmnamen mit den Angaben, in welchen Blöcken das Programm auf der Diskette zu finden ist. Die Verwaltung der Directory übernimmt ein Programm, daß alle Aktivitäten der Diskettenstation steuert. Ein solches Programm nennt sich DOS (Disc Operating System). Das beim CPC-464/664 verwendete DOS heißt AMSDOS (Amstrad Disc Operating System). DOS ist ein sehr umfangreiches Programm und belegt daher einen nicht unerheblichen Speicherraum des Rechners.

Im CPC-464/664 befindet sich das AMSDOS in einem ROM, der den Bereich &C000 bis &FFFF überlagert (siehe Band 1, Kapitel 1.0). Der CPC-664 hat diesen ROM direkt in der Konsole eingebaut. Der CPC-464 besitzt diesen ROM nicht, sondern er befindet sich in dem „Beikasten" der Diskettenstation DDI-1.

Neben diesem ROM belegt das AMSDOS noch zusätzlich 1284 Bytes des Arbeitsspeichers (RAM) des Rechners. Aus diesem Grunde verkleinert sich der freie Speicherraum bei angeschlossener Diskettenstation oder beim CPC-664 um 1284 Bytes auf 42619 Bytes. Der frei zur Verfügung stehende Speicherraum beträgt damit beim

Kassettenbetrieb:	43533 Bytes
Diskettenbetrieb:	42249 Bytes

Was ist nun besser: Diskette oder Kassette?

Man kann diese Frage so nicht stellen. Es kommt immer darauf an, was Sie mit Ihrem Computer anstellen wollen. Wollen Sie gelegentlich nur mal ein Spiel spielen, so ist der Datenrecorder des CPC-464 vollkommen ausreichend. Haben Sie allerdings große Mengen Daten zu verwalten, ist eine Diskettenstation notwendig. Natürlich ist das Arbeiten mit einer Diskettenstation wesentlich komfortabler und wer einmal mit einer Diskettenstation gearbeitet hat, will sie auch nicht mehr missen. Doch ob Ihnen das rund 800.- DM Wert ist, müssen Sie entscheiden.

1.1 Übertragungsgeschwindigkeit

Neben der komfortableren Handhabung ist die Übertragungsgeschwindigkeit der Daten zwischen dem Rechner und der Diskettenstation ein gravierender Vorteil gegenüber dem Datenrecorder. Die Geschwindigkeit der Übermittelung ist einheitlich mit der Bezeichnung „Baud" geregelt.

1 Baud ≙= 1 Bit pro Sekunde

Die Übertragungsgeschwindigkeit von 8 Baud entspricht einem Byte pro Sekunde. Der im CPC-464 eingebaute Datenrecorder besitzt normalerweise eine Übertragungsgeschwindigkeit von 1000 Baud, daß entspricht 125 Bytes pro Sekunde (1000/8=125). Durch den BASIC-Befehl

SPEED WRITE 1

läßt sich die Schreibgeschwindigkeit um den Faktor 2 auf 2000 Baud erhöhen. Die Erhöhung der Schreibgeschwindigkeit mittels SPEED WRITE ist ausschließlich im Kassettenbetrieb möglich! Mit

SPEED WRITE 0

wird die Schreibgeschwindigkeit der Kassette wieder auf 1000 Baud gesetzt.

Die Einlesegeschwindigkeit ist abhängig von der Geschwindigkeit, in der die Daten aufgezeichnet sind. Sind sie mit SPEED WRITE 0 aufgezeichnet, erfolgt das Einlesen automatisch mit 1000 Baud und damit unabhängig von der momentan eingestellten Schreibgeschwindigkeit (z.B. SPEED WRITE 1). Ein mit 1000 Baud aufgezeichnetes Programm wird durch Erhöhung der Schreibgeschwindigkeit nicht schneller eingeladen. Falls Sie ihre Daten (Programme etc.) grundsätzlich mit 2000 Baud aufzeichnen, sollten Sie hochwertige Kassetten benutzen, damit die Gefahr von Schreib- und damit Lesefehlern möglichst ausgeschlossen wird.

Bei einer Übertragungsrate von 2000 Baud (250 Bytes pro Sekunde) wäre ein 16 kByte langes Programm theoretisch in 65 Sekunden (16384/250) von der Kassette geladen. Doch in der Praxis sieht dies anders aus: Die Kassette benötigt knapp 2 Minuten zum Laden. Daraus ergibt sich eine Übertragungsgeschwindigkeit von rund 1000 Baud bei SPEED WRITE 1! Die Ursache für dieses Verhalten ist vor allem in dem ständigen Stoppen der Kassette vor jedem neuen 2 kByte großen Datenblock zu finden.

Bei der Diskettenstation DDI-1 oder der im CPC-664 eingebauten Diskettenstation liegt die Übertragungsgeschwindigkeit wesentlich höher als bei dem Datenrecorder. Ein 16 kByte langes Programm ist in ca. 5 Sekunden geladen. Die Übertragungsrate liegt damit bei ca. 25000 Baud. Das Sichern auf Diskette dauert etwas länger: Für 16 kByte rund 8 Sekunden (bedingt durch die Aktualisierung der Directory).

Die Übertragungsgeschwindigkeit zur Diskettenstation ist beeindruckend und durchaus für den professionellen Einsatz geeignet. Im „Homecomputer"-Bereich ist sie z.Z. konkurrenzlos: Die Diskettenstation VC 1541 zum COMMODORE 64 braucht für 16 kByte immerhin 50 Sekunden!

1.2 Formatieren einer Diskette

Jede neue, ungebrauchte Diskette ist vor ihrer Benutzung zu „formatieren". Durch das Formatieren werden bestimmte Bereiche der Diskette (Spur/Sektor, siehe 1.0) für die Datensicherung vorbereitet. Dieser Vorgang ist nur innerhalb des CP/M-Betriebssystems mit dem Dienstprogramm „FORMAT" möglich. Insgesamt unterstützt das AMSDOS vier verschiedene Aufzeichnungsformate, die teilweise einen anderen Diskettenaufbau (Format) benötigen. Das jeweilige Diskettenformat kann mit Hilfe eines Index beim Formatierungs-Befehl bestimmt werden:

```
FORMAT    : AMS-Format
FORMAT V  : Vendor-Format
FORMAT D  : Daten-Format
FORMAT I  : IBM-Format
```

Das AMS-Format ist das „Standardformat" und sollte generell für die Programmaufzeichnung benutzt werden. Dieses Format besitzt insgesamt 40 Spuren (0-39) und 9 Sektoren (0-8, intern als &41-&49 bezeichnet). Zusätzlich erfolgt bei diesem Format noch das Kopieren des CP/M-Betriebssystems in die Spuren 0 und 1 der Diskette. Damit steht Ihnen auf der Diskette das CP/M-Betriebssystem jederzeit zur Verfügung.

Das Vendor-Format ist dem AMS-Format ähnlich, kopiert allerdings die CP/M-Spuren 0 und 1 nicht mit. Gekaufte Programmdisketten besitzen generell dieses Format, da das CP/M-Betriebssystem nicht mit verkauft werden darf.

Das Daten-Format ist speziell für die Sicherung von Daten ausgelegt. Der Vorteil liegt darin, daß auch die Spuren 0 und 1 für die Datensicherung nutzbar sind, so daß insgesamt 9 kByte mehr Speicherraum zur Verfügung stehen (intern sind die Sektoren mit &C1 bis &C9 bezeichnet).

Mit 40 Spuren und 8 (!) Sektoren unterscheidet sich das IBM-Format erheblich von den anderen Formaten und ist identisch mit dem IBM-Format unter CP/M. Nur die erste Spur (Spur 0) ist als Systemspur reserviert. Intern sind die Sektoren mit &01 bis &07 bezeichnet.

Wie schon erwähnt, bietet sich das AMS-Format an, damit Sie jederzeit Zugriff auf das CP/M-Betriebssystem haben, ohne ständig die Diskette zu wechseln.

2.0 Gemeinsame Befehle

Alle Standard-BASIC-Befehle, die der Kassette zugeordnet sind (LOAD, RUN, SAVE etc), beziehen sich sowohl auf den Datenrecorder als auch auf die Diskettenstation. Nach dem Einschalten des CPC-464 prüft der Rechner, ob eine Diskettenstation angeschlossen ist. Wenn dieses der Fall ist, beziehen sich alle Befehle (mit Ausnahme von SPEED WRITE) auf die Diskettenstation, andernfalls auf den eingebauten Datenrecorder. Aus diesem Grunde ist es notwendig, erst die Diskettenstation und dann den Monitor und Rechner einzuschalten, damit der Rechner eine aktive Diskettenstation vorfindet.

Der nachfolgende Teil faßt alle gemeinsamen Befehle zusammen und zeigt an kurzen Beispielen, wie sie anzuwenden sind. Im Kassetten- und Diskettenbetrieb erzielen alle Befehle weitestgehend die gleichen Ergebnisse. Auf Unterschiede wird besonders hingewiesen. Die einzige Ausnahme bildet der Befehl SPEED WRITE, da eine Änderung der Schreibgeschwindigkeit ausschließlich im Kassettenbetrieb möglich ist (siehe 1.1).

2.1 SAVE – Sichern von Programmen

Der SAVE-Befehl (save = sichern) gliedert sich in 4 verschiedene Funktionen:

- Sichern als normales Programm
- Sichern als geschütztes Programm (P)
- Sichern als ASCII-Datei (A)
- Sichern als binäre Datei (B)

Alle 4 Formate sind mit der Kassette oder Diskette möglich. Das zu sichernde Programm muß einen eindeutigen Namen tragen. Bei dem SAVEn auf Kassette darf der Filename eine Länge von maximal 16 beliebigen Zeichen beinhalten.

```
SAVE "KASSETTENFILE"          (maximal 16 Zeichen)
```

Ist der eingegebene Filename länger als 16 Zeichen, begrenzt der Rechner den Filenamen automatisch auf 16 Zeichen. Besteht der Filename aus einem Leerstring

```
SAVE ''''
oder SAVE ''
```

benennt der Rechner das Programm automatisch mit

```
UNNAMED FILE        (Unbenanntes Programm).
```

Nach der Eingabe des SAVE-Befehls meldet sich der Rechner mit der Anweisung:

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

und fordert den Benutzer auf, gemeinsam die RECord und PLAY-Taste des Datenrecorders zu drücken. Durch Betätigen einer beliebigen Taste beginnt dann der SAVE-Vorgang. Dabei wird das Programm (Daten) in Blöcken a 2 kByte auf die Kassette übertragen. Auf dem Bildschirm kann verfolgt werden, welchen Block der Rechner gerade sichert, da er ständig die Meldung:

```
Saving  TESTFILE                    block  x
```

ausgibt. Die Unterdrückung dieser Meldungen ist mit einem vorgestellten „!" im Filenamen möglich:

```
SAVE ''!TEST''
```

Doch beachten Sie, daß ebenfalls die Meldung „Press REC and ..." nicht mehr erfolgt und das Programm sofort gesichert wird !!

Beim Sichern auf der Diskette darf der Filename maximal aus 8 Zeichen und einer durch den Dezimalpunkt „." getrennten Typenbezeichnung bestehen. Diese Typenbezeichnung ist aus dem CP/M abgeleitet und enthält eine zusätzliche Information über die Art des Files. Der 3-stellige Filetyp kann dabei aus beliebigen ASCII-Zeichen bestehen. Allerdings werden einige Typenbezeichnungen auch vom AMSDOS als Unterscheidungsmerkmal benutzt (siehe 6.0). So z.B.

BAS	BASIC-Programm
BAK	Backup-Kopie eines Files
BIN	Binäre Daten (z.B. Maschinenprogramm)

Werden im Filenamen unzulässige Zeichen benutzt (z.B. * ? / ! ,) oder ist der Filename ein Leerstring, erfolgt die Fehlermeldung:

Bad Command

Der SAVE-Befehl hat beim Diskettenbetrieb zunächst folgendes Format:

SAVE ''filename.typ''

Das Sichern des Programms findet sofort ohne weitere Meldungen statt. Befindet sich in dem angesprochenen Diskettenlaufwerk keine Diskette, erfolgt die Meldung:

Drive x: disk missing
Retry, Ignore or Cancel?

Durch Eingabe einer der drei Anfangsbuchstaben (R, I, C) kann der (SAVE-) Vorgang

R	wiederholt
I	ignoriert oder
C	abgebrochen (cancel)

werden. Legen Sie dann die Diskette in das Laufwerk und wiederholen (Retry) den Vorgang.

Befindet sich auf der Diskette bereits ein File mit der Bezeichnung „filename.typ", so benennt das AMSDOS den alten File in „filename.BAK" um und sichert den neuen File mit dem eingegebenen Filenamen/Typ ab. Damit befindet sich das ursprüngliche Programm weiterhin als „Sicherheitskopie" (Backup) auf der Diskette.

Fehlt hinter dem Filenamen die Angabe des Filetyps, so setzt das AMSDOS den Typ auf „BAS" (BASIC-Programm).

SAVE ''filename'' ≙ SAVE ''filename.BAS''

Vor dem Filenamen können zwei weitere Parameter folgen, die das Laufwerk (A oder B) und die Benutzer-Ebene (siehe 3.4) bezeichnen, wo das bezeichnete Programm abgelegt werden soll:

SAVE ''⟨user⟩⟨drive:⟩filename.typ''

Die ,,user''-Nummer gibt die Benutzer-Ebene (0-15) und der Parameter ,,drive'' (A oder B) das gewünschte Laufwerk an. Dadurch ist es z.B. möglich, einen File auf eine andere Benutzer-Ebene oder wahlfrei auf eines der zwei möglichen Laufwerke zu sichern. Diese beiden Parameter sind ebenfalls bei den Befehlen LOAD, RUN, MERGE, CHAIN MERGE zulässig.

Alle bisherigen Betrachtungen bezogen sich ausschließlich auf das Sichern von ,,normalen'' BASIC-Programmen. Wie anfangs schon erwähnt, bestehen aber noch drei weitere Möglichkeiten des SAVEns. Alle 3 Arten des Sicherns sind sowohl auf Diskette als auch auf Kassette möglich.

SAVE ''filename⟨.typ⟩'',P

Das nachgestellte, durch Komma getrennte ,,P'' steht für Protected (geschützt) und sichert den bezeichneten File in einer geschützten Form ab. Das Programm ist nach dem SAVEn nur noch mit dem RUN ''...''-Befehl (siehe 2.3) lad- und startbar. Weiterhin ist damit gewährleistet, daß dieses Programm nach dem Laden nicht geLISTet werden kann. Diese Form des Sicherns von Programmen ist sehr zweifelhaft, da es meistens nicht notwendig ist, Programme zu ,,schützen'' und zum andern diese Art von ,,Schutz'' ohne große Probleme aufzuheben ist.

SAVE ''filename⟨.typ⟩'',A

Wesentlich interessanter ist die Möglichkeit, ein Programm als ASCII-Datei abzulegen. Normalerweise erfolgt das Speichern eines Programms in einer verkürzten Form (Token). Alle BASIC-Befehle sind in ein oder zwei Byte große Worte verschlüsselt und belegen deshalb wesentlich weniger Speicherkapazität als das eigentliche Befehlswort (z.B. ,,PRINT''). Mit der Option ,,A'' (ASCII) findet das Sichern in der Form statt, wie das Programm beim Listen sichtbar ist, d.h. alle Token sind wieder in Befehle übersetzt. Der große Vorteil besteht darin, daß so Programmlistings ohne Probleme in Textverarbeitungssysteme eingelesen werden können.

SAVE ''filename⟨.typ⟩'',B,startadresse,länge

Die letzte Form des SAVE-Befehls ist das Sichern als binäre Datei. Dabei wird ein Speicherbereich des Rechners (bestimmt durch die „startadresse" und die „länge") so auf die Diskette/Kassette kopiert, wie die Daten dort liegen. Sehr einfach lassen sich so Maschinenprogramme oder Bildschirminhalte sichern. Das folgende Beispiel SAVEt den Bildschirminhalt auf Kassette/Diskette ab:

```
SAVE "VIDEO.BIN",B,&C000,&3FFF
```

Mit der Diskette dauert dieser Vorgang 8 Sekunden, mit der Kassette hingegen ca. 2 Minuten (SPEED WRITE 1).

Beim SAVEn einer binären Datei ohne Typ-Kennzeichnung auf Diskette erhält der File automatisch den Typ „.BIN".

2.2 LOAD – Laden von Daten

Der LOAD-Befehl lädt Daten (Programme) von der Diskette/Kassette wieder in den Arbeitsspeicher des CPC-464 bzw. 664. Alle Beschränkungen bei der Bezeichnung eines Files (Namensgebung) sind mit den Angaben vom SAVE-Befehl identisch.

```
LOAD "filename\.typ\"
```

lädt das Programm „filename" (beim Diskettenbetrieb mit „typ"-Angabe) von der Diskette oder Kassette in den Rechner. Dabei erfolgt das Ablegen des Programms grundsätzlich an den Anfang des freien BASIC-Speichers. Im Kassettenbetrieb folgt die Aufforderung:

```
Press PLAY then any key
```

Dann läuft der Datenrecorder an und sucht ab seiner aktuellen Bandstellung die Kassette nach dem gewünschten File ab. Ist ein Programm gefunden, dessen Name mit „filename" übereinstimmt, werden alle 2 kByte großen Blöcke des Programms eingeladen. Dabei gibt der CPC-464 die Meldung:

```
Loading filename  block x
```

aus. Ist der Ladevorgang korrekt abgeschlossen, erscheint die READY-meldung auf dem Bildschirm. Traten bei der Übertragung Fehler auf, erscheint eine der drei Kassettenfehlermeldungen:

Read error a	Lesefehler (Biterkennung)
Read error b	Checksummenfehler
Read error d	Block zu lang

Die Fehlermeldung „Read error a" tritt bei weitem am häufigsten auf und ist auf eine „schlechte" Kassette oder auf „leiern" der Kassette zurückzuführen.

Besteht der Filename aus einem Leerstring

```
    LOAD ""
oder LOAD ""
```

wird das nächste Programm auf der Kassette in den CPC-464 eingeladen.

Beim Diskettenbetrieb bewirkt dieser Ladebefehl die Fehlermeldung:

```
Bad Command
```

Das Laden von der Diskette ist identisch. Fehlt aber die Typenangabe, so wird als Typ die Bezeichnung „.BAS" angenommen. Befindet sich das gewünschte Programm nicht auf der Diskette, erfolgt die Fehlermeldung:

```
filename.typ    not found
```

Haben Sie ein Programm als ASCII-Datei gesichert ("",A"), können Sie diese „Zeichenfolge" wie ein gewöhnliches BASIC-Programm laden und starten. Nach dem Laden bindet der Rechner diese Zeichenfolge automatisch als BASIC-Zeilen ein. Damit besteht die Möglichkeit, ein BASIC-Programm mit einem Textverarbeitungssystem zu erstellen und später wie ein BASIC-Programm in den Rechner zu laden.

Eine Besonderheit ist das Laden einer binären Datei. Handelt es sich bei dem zu ladenden File um eine binäre Datei, erfolgt das Laden automatisch an die Stelle, von der es beim Sichern geholt wurde. Der Bildschirm, der mit

```
SAVE "VIDEO.BIN",B,&C000,&3FFF
```

gesichert ist, kann mit

```
LOAD ''VIDEO.BIN''
```

wieder direkt (!) in den Bildschirm geladen werden. Durch die Angabe einer zusätzlichen Startadresse kann die binäre Datei aber auch an einen beliebigen anderen Platz innerhalb des Arbeitsspeichers geladen werden (achten Sie aber darauf, daß Sie keine wichtigen Informationen überschreiben):

```
(MEMORY 15999)
LOAD ''VIDEO.BIN'',16000
```

Die binäre Datei „VIDEO" liegt bei diesem Beispiel nach dem Laden ab Adresse 16000.

2.3 RUN "..." – Laden und Starten

Dieser Befehl arbeitet ähnlich wie der LOAD-Befehl, startet aber nach dem Laden das Programm automatisch (Autostart).

```
RUN ''filename⟨.typ⟩''
```

Folgt im Kassettenbetrieb hinter RUN ''...'' ein Leerstring,

```
      RUN ''''
oder  RUN ''
```

wird das nächstfolgende Programm von der Kassette geladen und gestartet. Im Diskettenbetrieb bewirkt dieses die Fehlermeldung

```
Bad Command,
```

da immer ein Filename anzugeben ist.
Geschützte Programme lassen sich nur mit diesem Befehl laden und starten!

2.4 CAT – Inhaltsverzeichnis

Der Befehl

CAT (catalogue = Verzeichnis)

zeigt eine Liste aller Files, die sich auf der Diskette befinden. Im Kassettenbetrieb erfolgt die Ausgabe aller gefundenen Programmblöcke (ab der aktuellen Bandstellung). Nach der Aufforderung, die PLAY-Taste des Datenrecorders zu betätigen, werden neben den Filenamen die Blocknummern und eine Kennzeichnung des Filetyps ausgegeben. Gängige Filetypkennzeichen sind im Kassettenbetrieb:

$ BASIC-Programm
% Geschütztes Programm (P)
* ASCII-Datei (A)
& Binäre Datei (B)

Wurde der gelesene Block korrekt gefunden (es liegen keine Fehler vor), bestätigt dies der Rechner mit einer „OK"-Meldung hinter dem Filetype. Andernfalls folgt eine Fehlermeldung (Read error).

Im Diskettenbetrieb erfolgt nach kurzer Ladezeit die Liste aller Filenamen mit Typenbezeichnung und ihrer Längenangabe.

```
Drive A: user 0
VIDEO    .BIN    16    TEST     .BAS    3
150k free
```

2.5 MERGE – Zusammenfügen von Programmen

Der Aufbau einer Unterprogrammbibliothek ist immer dann von Vorteil, wenn Sie bestimmte Unterprogramme in verschiedenen Programmen benötigen (z.B. Sortierroutine, Balkendiagramm etc.). Sie brauchen dann diesen Programmteil nur einmal einzugeben. Mit Hilfe des Befehls MERGE (merge=zusammenfügen) lassen sich verschiedene Programme zusammenfügen.

MERGE ''filename⟨.typ⟩''

Das Programm „filename" wird zusätzlich zu dem im Rechner befindlichen Programm geladen und eingebunden. Belegt das nachgeladene Programm gleiche Zeilennummern wie das Ursprungsprogramm, so werden diese von ihm überschrieben. Beachten Sie deshalb, daß der Zeilenbereich immer außerhalb des Ursprungsprogramms liegt. Nach dem Zuladen eines Programms sind alle Variablen und benutzerdefinierten Funktionen gelöscht.

Doch leider hat sich im Betriebssystem des CPC-464 ein kleiner Fehler eingeschlichen, der das Zuladen von Programmen (fast) unmöglich macht. Wenn das Betriebssystem beim Nachladen auf den Wert &1A (Steuerzeichen für „EOF" = End Of File) trifft, beendet er seine Arbeit mit der Meldung

EOF met

Inzwischen hat die Firma Schneider diesen Fehler behoben, so daß bei neueren CPC-464 und dem CPC-664 dieser Fehler nicht mehr auftritt.

Tritt bei Ihrer Version dieser Fehler noch auf, können Sie sich damit behelfen, daß Sie alle Programme, die Sie zuladen wollen, als ASCII-Datei ablegen. Beim Sichern als ASCII-Datei wird das gesamte Programm unkodiert, also als ASCII-Zeichenfolge abgelegt, so daß kein „EOF" auftritt (siehe 2.2). Zwar dauert das Laden etwas länger und der benötigte Speicherbedarf auf der Kassette/Diskette ist geringfügig größer, aber der MERGE-Vorgang arbeitet so einwandfrei.

Das gleiche Problem tritt beim Nachladen mit dem CHAIN MERGE- Befehl auf. Auch hier legen Sie dann die nachzuladenden Programme als ASCII-Dateien ab.

2.6 CHAIN MERGE – Nachladen von Programmteilen

Ganz besonders interessant ist der CHAIN MERGE-Befehl (chain = anketten) gerade bei der Entwicklung von größeren Programmen oder Programmpaketen. Der CHAIN MERGE-Befehl ermöglicht es, Programmteile zuzuladen und an einem beliebigen Punkt des Programms mit seiner Arbeit fortzufahren. Ist ein Programm z.B. so groß, daß es nicht mehr in seiner Gesamtheit in den Rechner paßt, könnte es in Programmblöcke zerlegt und diese bei Bedarf nachgeladen werden. Oder aber Sie laden von einem Hauptprogramm bei Bedarf bestimmte Programmteile nach, die dann nach ihrer Ausführung wieder in das

Hauptprogramm zurückführen. Der CHAIN MERGE-Befehl bietet weiterhin die Möglichkeit, vor dem Nachladen eines Programms Bereiche des „alten" Programms zu löschen, um z.B. Platz für den neuen Programmteil zu schaffen.

CHAIN MERGE''filename⟨.typ⟩'',⟨zeilennummer⟩,⟨DELETE bereich⟩

Nach dem Programmnamen folgt bei Bedarf eine Zeilennummer, die angibt, in welcher Programmzeile das Programm nach dem Laden fortgesetzt werden soll. Ist keine Zeilennummer angegeben, startet das Programm ab der niedrigsten Zeilennummer. Die zweite Option löscht vor dem Nachladen die nach DELETE angegebenen Zeilennummern.

CHAIN MERGE ''TESTFILE.BAS'',5000,DELETE 10000-

Das Beispiel lädt das Programm „TESTFILE" nach und setzt das Programm dann in Zeile 5000 fort. Vor dem Laden werden allerdings alle Zeilennummern größer oder gleich 10000 gelöscht.
Nach dem Zuladen bleiben alle Variableninhalte weitestgehend erhalten und stehen damit weiter zur Verfügung.

Das nachfolgende Beispielprogramm zeigt, wie der CHAIN MERGE-Befehl eingesetzt werden könnte. Das Hauptprogramm lädt bei Bedarf ein Unterprogramm nach. Die Unterprogramme (Kreis- und Dreiecksberechnung) müssen auf der Diskette unter den Namen „KREIS.BAS" und „DREI.BAS" abgelegt sein (gegebenenfalls als ASCII-Datei). Nach kurzer Ladezeit liegt das Unterprogramm dann ab Zeilennummer 5000 und kehrt nach Beendigung seiner Arbeit in das Hauptprogramm zurück. Diese Anwendung des CHAIN MERGE-Befehls ist nur im Diskettenbetrieb sinnvoll, da nur hier der Zugriff auf beliebige Programme möglich ist.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                                *
1020 REM *          CHAIN MERGE - Demonstration           *
1030 REM *                                                *
1040 REM ************************************************
1050 MODE 2
1060 PRINT "<Hauptprogramm>"
1070 PRINT:PRINT
1080 PRINT " <1> Kreisberechnung ....."
1090 PRINT " <2> Dreiecksberechnung ....."
1100 PRINT " <?> ....................."
1110 PRINT " <9> Programmende ....."
```

```
1120 PRINT
1130 PRINT "      Bitte Ihre Wahl ?"
1140 in$=INKEY$:IF in$="" THEN 1140
1150 in=VAL(in$)
1160 IF in=1 THEN nl$="KREIS":GOTO 1200
1170 IF in=2 THEN nl$="DREI":GOTO 1200
1180 IF in=9 THEN END
1190 PRINT CHR$(7);:GOTO 1140
1200 CHAIN MERGE nl$,5000,DELETE 5000-
1210 '
5000 REM ********** Hier nachgeladene Programme **********
```

```
5000 REM ********************************** M.W.Thoma **
5010 REM *                                              *
5020 REM *        Unterprogramm DREIECKSBERECHNUNG      *
5030 REM *                                              *
5040 REM ************************************************
5050 CLS
5060 PRINT "<Dreiecksberechnung>
5070 PRINT:PRINT
5080 INPUT " >Kathede (a) in cm :";a
5090 INPUT " >Kathede (b) in cm :";b
5100 PRINT:PRINT
5110 PRINT " >Flaeche =";a*b/2;"cm2"
5120 c=SQR(a^2+b^2)
5130 PRINT " >Hypothe.=";c;"cm"
5140 PRINT " >Umfang  =";a+b+c;"cm"
5150 PRINT
5160 PRINT " >Weitere Berechnungen (J/N) ?"
5170 in$=INKEY$:IF in$="" THEN 5170
5180 IF UPPER$(in$)="N" THEN 1000
5190 GOTO 5070
5200 REM ************************************************
```

```
5000 REM ********************************** M.W.Thoma **
5010 REM *                                              *
5020 REM *        Unterprogramm KREISBERECHNUNG         *
5030 REM *                                              *
5040 REM ************************************************
5050 CLS
5060 PRINT "<Kreisberechnung>"
5070 PRINT:PRINT
5080 INPUT " >Radius (cm) :";r
5090 PRINT
5100 PRINT " >Umfang  =";2*r*PI;"cm"
5110 PRINT " >Flaeche =";(2*r)^2*PI/4;"cm2"
5120 PRINT
```

```
5130 PRINT " >Weitere Berechnungen (J/N) ?"
5140 in$=INKEY$:IF in$="" THEN 5140
5150 IF UPPER$(in$)="N" THEN 1000
5160 GOTO 5070
5170 REM ************************************************
```

2.7 OPENIN und OPENOUT

Die folgenden Befehle sind global als „Dateibefehle" bezeichnet und ermöglichen die Bearbeitung von sequentiellen Dateien. Die Form der sequentiellen Datei ist die einzige im BASIC des CPC-464 oder CPC-664 vorgesehene Möglichkeit, Daten extern zu sichern. Andere Zugriffsmethoden sind nur im CP/M-Betriebssystem (siehe Band 3) oder mit Hilfe von Maschinenprogrammen, die bestimmte Betriebssystemroutinen nutzen, möglich (siehe 10.0). Im Kassettenbetrieb ist ausschließlich eine sequentielle Datenbearbeitung möglich.

Damit Daten zwischen dem Rechner und einem externen Speichermedium (Diskette/Kassette) übermittelt werden können, ist ein „Datenweg" zwischen ihnen notwendig. Das Schneider-BASIC unterscheidet zwei Datenrichtungen:

OPENIN	Datentransfer externer Speicher – Rechner
OPENOUT	Datentransfer Rechner – externer Speicher

Die „Datenwegnummer", über die die Daten zu senden oder zu empfangen sind, ist beim CPC-464/664 fest auf die Nummer 9 gelegt (die Nummern 0-7 sind für die Windows und die Nummer 8 für die Centronics-Schnittstelle reserviert). Ist der Datenweg einmal eröffnet (open=eröffnen), können mit INPUT #9 oder PRINT #9 Daten aus der Datei gelesen oder in die Datei geschrieben werden. Die zu öffnende Datei ist mit einem Namen zu kennzeichnen (beim Diskettenbetrieb mit Typenbezeichnung):

```
OPENIN  "filename⟨.typ⟩"
OPENOUT "filename⟨.typ⟩"
```

Der OPENOUT-Befehl legt eine neue Datei mit dem Namen „filename" an und erstellt im Diskettenbetrieb einen neuen Directory-Eintrag. Alle Daten, die auf die Kassette oder Diskette geschrieben werden sollen, befinden sich zunächst in einem Zwischenspeicher, der sich normalerweise am Ende des freien

Arbeitsspeichers befindet. Der Zwischenspeicher ist 4 kByte groß, davon werden allerdings nur 2 kByte genutzt. Ist dieser Zwischenspeicher gefüllt, erscheint im Kassettenbetrieb die Aufforderung:

Press REC an PLAY then any key

Nach Betätigung einer Taste schreibt der Rechner alle Daten, die sich im Zwischenspeicher befinden, in die Datei. Im Diskettenbetrieb erfolgt dieser Vorgang automatisch ohne weitere Aufforderungen. Das Beispielprogramm „OPENOUT" eröffnet die Datei „TEST.DAT" und schreibt dann nacheinander die Zahlen 0 bis 400 in die Datei.

```
1000 REM ************************************* M.W.Thoma **
1010 REM *                                                 *
1020 REM *             OPENOUT - Demonstration             *
1030 REM *                                                 *
1040 REM ***************************************************
1050 MODE 2:SPEED WRITE 1
1060 PRINT " >OPENOUT-Demonstration"
1070 PRINT:PRINT
1080 PRINT " >Schreibe die Zahlen 0-400"
1090 PRINT " >in die Datei 'TEST.DAT'"
1100 PRINT
1110 OPENOUT "TEST.DAT"
1120 FOR zahl=0 TO 400
1130 PRINT#9,zahl
1135 LOCATE 1,8:PRINT USING " >###";zahl
1140 NEXT zahl
1150 CLOSEOUT
1160 END
2000 REM ***************************************************
```

Ganz deutlich ist zu erkennen, daß bei der Zahl 307 der Zwischenspeicher gefüllt ist. Im Kassettenbetrieb erfolgt nun die Aufforderung, die RECORD- und PLAY-Taste des Datenrecorders zu drücken. Beim Diskettenbetrieb ist nur eine kurze Unterbrechung und das Anlaufen des Laufwerks zu bemerken. Anschließend sichert das Programm den Zwischenspeicher und setzt seine Arbeit mit der Übermittelung der restlichen Zahlen 308 bis 400 fort.

Am Ende des Programms steht der Befehl CLOSEOUT (close=schließen). Der CLOSEOUT-Befehl ist sehr wichtig, da er zwei Dinge sicherstellt:

1. Die restlichen Daten des Zwischenspeichers werden in die Datei geschrieben.
2. Der Datenweg ist damit geschlossen und kann für andere Dateien genutzt werden.

Der OPENIN-Befehl eröffnet ebenfalls einen Datenweg vom externen Speicher (Diskette/Kassette) zum CPC-464. Eine vorhandene (angelegte) Datei kann dann wieder in den Rechner eingelesen werden. Wie beim Eröffnen einer Datei ist der Name der gewünschten Datei anzugeben.

```
OPENIN ''filename⟨.typ⟩''
```

Im Kassettenbetrieb erfolgt die Aufforderung, die PLAY-Taste des Datenrecorders zu betätigen. Daraufhin sucht der Rechner solange die gewünschte Datei auf der Kassette, bis sie gefunden ist (oder das Ende der Kassette erreicht ist). Ist im Diskettenbetrieb die gewünschte Datei nicht auf der eingelegten Diskette, erscheint die Fehlermeldung:

```
filename.typ  not found
```

Leider ist es beim CPC-464 nicht möglich, diese Fehlermeldung z.B. mit einer ON ERROR-Routine aufzufangen (siehe Band 1, Kap.17.0). Dieser Fehler verursacht immer einen Programmabbruch! Im Gegensatz dazu besitzt die Fehlervariable ERR beim CPC-664 den Wert 32, falls ein Diskettenfehler auftrat. Anhand der Fehlervariablen ERR kann allerdings nur festgestellt werden, daß ein Diskettenfehler aufgetreten ist, aber nicht welcher!

Ein Problem beim Einlesen von Dateien ist das Feststellen, wieviele Daten eigentlich in der Datei vorhanden sind. Ist die Anzahl der vorhandenen Daten nicht bekannt, bietet die Systemvariable EOF (End of File) eine komfortable Möglichkeit, festzustellen, ob das letzte Datum einer Datei bereits eingelesen ist.

```
EOF =  0   Es folgen weitere Daten
EOF = -1   Letztes Datum eingelesen
```

Aus einer zum Lesen eröffneten Datei können solange Daten gelesen werden, bis die Systemvariable EOF gleich −1 ist. Versuchen Sie über das Dateiende hinaus Daten zu lesen, erfolgt die Fehlermeldung

```
EOF met
```

Die gleiche Fehlermeldung erscheint auch dann, wenn versucht wird, eine OPENIN-Datei zu beschreiben (PRINT#9 oder WRITE#9) oder eine OPENOUT-Datei zu lesen (INPUT#9 oder LINE INPUT#9).

Der Lesevorgang ist, wie Sie dem Beispielprogramm entnehmen können, fast identisch dem des Schreibens. Trifft das Programm auf ein „EOF" (EOF = – 1), endet das Programm in Zeile 1160 (vorher schließen des Datenweges mit CLOSEIN).

```
1000 REM ****************************************** M.W.Thoma **
1010 REM *                                                     *
1020 REM *              OPENIN - Demonstration                 *
1030 REM *                                                     *
1040 REM *******************************************************
1050 MODE 2:SPEED WRITE 1
1060 PRINT " >OPENIN-Demonstration"
1070 PRINT:PRINT
1080 PRINT " >Lese die Zahlen 0-400"
1090 PRINT " >aus der Datei 'TEST.DAT'"
1100 PRINT
1110 OPENIN "TEST.DAT"
1120 INPUT#9,zahl
1130 LOCATE 1,8:PRINT USING " >###";zahl
1140 IF EOF <>-1 THEN 1120
1150 CLOSEIN
1160 END
1170 REM *******************************************************
```

Wie schon erwähnt, liegt der Zwischenspeicher, der für den Datenaustausch benötigt wird, am Ende des freien BASIC-Speichers. Diese Tatsache ist sehr bedauerlich, da vom BASIC-RAM-Ende abwärts ebenfalls die Stringvariableninhalte abgelegt sind. Beim Eröffnen eines Datenweges (OPENIN, OPENOUT) wird gleichzeitig ein 4 kByte großer Datenpuffer am Ende des BASIC-RAMs angelegt. Da hier aber wichtigte Stringinhalte liegen können, müssen alle Stringinhalte um 4 kByte nach unten verschoben und „aufgeräumt" werden. Dieses „Aufräumen" (Garbage Collection, nicht mehr benutzte Stringinhalte eleminieren) ist eine sehr zeitaufwendige Routine des Betriebssystems und daher sehr lästig. Die Garbage Collection kann je nach Anzahl der vorhandenen Stringinhalte bis zu 10 Minuten (!!!) dauern. Während der Garbage Collection passiert „nichts". Sie können die Garbage Collection nicht einmal mit der Escape-Taste unterbrechen. Sie müssen dann leider warten!

Umgangen werden kann dieses Problem dadurch, daß zu Beginn des Programms ein Datenpuffer angelegt und vor dem Überschreiben durch Stringvariablen geschützt wird (Mit dem MEMORY-Befehl das RAM-Ende nach unten verschieben).

```
OPENOUT ''DUMMY''
MEMORY HIMEM-1
CLOSEOUT
```

Mit dieser Zeile am Beginn des Programms verhindern Sie die durch das Anlegen des Datenpuffers erzwungene Garbage Collection. Der zur Verfügung stehende Speicherplatz verringert sich damit allerdings um 4 kByte.

Mit einem einfachen Test können Sie dies feststellen: Führen Sie einen Reset durch (CRTL/SHIFT/ESC) und fragen mit

```
PRINT HIMEM
```

die obere BASIC-Speichergrenze ab. Im Diskettenbetrieb erhalten Sie die Adresse 42619, im Kassettenbetrieb die Adresse 43903. Geben Sie nun

```
OPENOUT ''DUMMY''
MEMORY HIMEM-1
CLOSEOUT
```

ein und fragen nocheinmal nach der letzten freien Speicheradresse. Im Kassettenbetrieb wird die Adresse 39806 und im Diskettenbetrieb die Adresse 38522 gemeldet. Das sind genau 4 kByte weniger. Gleichzeitig ist der Speicherbereich oberhalb dieser Adressen vor dem Überschreiben geschützt (MEMORY HIMEM-1).

3.0 Spezielle Diskettenbefehle

Neben den bisher beschriebenen Befehlen, die fest im BASIC eingebunden sind, verfügt der CPC-464 bei angeschlossener Diskettenstation (oder beim CPC-664) über 13 weitere, sogenannte RSX-Befehle (Resident System Extension). Diese Befehle sind im ROM der Diskettenstation enthalten und daher ausschließlich mit angeschlossener Diskettenstation oder dem CPC-664 möglich.

Viele dieser Befehle sind aus dem CP/M abgeleitet und ermöglichen im wesentlichen die komfortable Bedienung der Diskettenstation. Schlagen Sie auch dort gegebenenfalls noch einmal nach (Band 3).

Allen Befehlen ist gemein, daß ihnen ein „|" (gleichzeitiges Betätigen der SHIFT- und der „@"-Taste) vorzustellen ist.

3.1 |TAPE, |TAPE.IN und |TAPE.OUT

Bei angeschlossener Diskettenstation oder beim CPC-664 beziehen sich alle gemeinsamen Befehle (siehe 2.0) ausschließlich auf die Diskettenstation. Damit der eingebaute Datenrecorder (beim CPC-664 ein externer Kassettenrecorder über die DIN-Buchse) trotzdem zu benutzen ist, können alle gemeinsamen Befehle mit

```
|TAPE
```

auf den Datenrecorder umgeleitet werden. Alle Befehle (SAVE, LOAD, CAT, OPENIN ....) beziehen sich dann ausschließlich auf den Datenrecorder (externen Kassettenrecoder).

Weiterhin sieht das AMSDOS ein splitten der gemeinsamen Befehle auf Diskette und Kassetten vor. Mit

```
|TAPE.IN
```

beziehen sich alle Operationen, die einen Lesezugriff auslösen, auf die Kassette (LOAD, MERGE, CHAIN MERGE, OPENIN und CAT). Der Befehl

```
|TAPE.OUT
```

aktiviert den Datenrecorder als Ausgabegerät und alle Schreiboperationen (SAVE und OPENOUT) beziehen sich ab sofort auf den Datenrecorder (externen Kassettenrecorder).

3.2 |DISC, |DISC.IN und |DISC.OUT

Der Befehl

```
|DISC
```

leitet alle Ein- oder Ausgabeoperationen auf die Diskette um. Damit werden gleichzeitig die Befehle |TAPE, |TAPE.IN oder |TAPE.OUT aufgehoben. Ebenso wie beim |TAPE-Befehl kann auch die Diskettenstation ausschließlich als Ein- oder Ausgabegerät genutzt werden:

```
|DISC.IN
```

Mit |DISC.IN beziehen sich alle Leseoperationen (LOAD, MERGE, CHAIN MERGE, OPENIN und CAT) auf die Diskettenstation. Gleichzeitig wird der Befehl |TAPE.IN damit aufgehoben.

```
|DISC.OUT
```

Die beiden Ausgabeoperationen SAVE und OPENOUT beziehen sich ab sofort auf die Diskettenstattion (Hebt |TAPE.OUT auf).

Das nachfolgende Beispiel zeigt eine Anwendung der Befehle |TAPE.IN, |TAPE.OUT, |DISK.IN und |DISK.OUT. Eine sequentielle Datei (in diesem Fall die Datei „TEST.DAT“ aus dem Kapitel 2.7) kann sowohl von Diskette auf Kassette wie auch von Kassette auf Diskette kopiert werden.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *     Kopieren  einer  sequentiellen Datei      *
1030 REM *        Disk/Tape        Tape/Disk             *
1040 REM *                                               *
1050 REM *************************************************
1060 MODE 2
1070 PRINT "Kopiere sequentielle Datei 'TEST.DAT'"
1080 PRINT
1090 INPUT "> (1)=Tape/Disk (2)=Disk/Tape ",x
1100 IF x<1 OR x>2 THEN 1090
1110 IF x=1 THEN |TAPE.IN:|DISC.OUT:GOTO 1130
1120 |DISC.IN:|TAPE.OUT
1130 OPENIN "test.dat"
1140 OPENOUT "test.cop"
1150 INPUT#9,z
1160 PRINT#9,z
1170 IF EOF=0 THEN 1150
1180 CLOSEIN:CLOSEOUT
1190 END
1200 REM *************************************************
```

3.3 |A, |B und |DRIVE

Das AMSDOS unterstützt das Arbeiten mit 2 Diskettenstationen. Dabei besitzt die eingebaute (CPC-664) oder die im Rechner eingesteckte Diskettenstation die Bezeichnung „A". Die zweite (eingeschleifte) Diskettenstation die Bezeichnung „B". Mit den Befehlen

```
|A
und |B
```

erfolgt die Umleitung aller Diskettenbefehle auf das bezeichnete Laufwerk. Ist das bezeichnete Laufwerk nicht angeschlossen, erfolgt die Fehlermeldung

```
Drive x: disk missing
```

Beim Arbeiten mit zwei Diskettenlaufwerken kann so sehr einfach zwischen diesen beiden umgeschaltet werden.

Der Befehl

```
|DRIVE
```

ist identisch mit den Befehlen |A und |B. Ihm muß allerdings eine Stringvariable übergeben werden, die ein „A" oder ein „B" für die Laufwerkbezeichnung enthält. Die Übergabe von Strings ist im AMSDOS nicht direkt möglich. Mit Hilfe des vorgestellten Klammeraffe „@" wird dem Befehl (Programm) die Adresse mitgeteilt, wo dieser String im Speicher liegt (siehe 4.0, Variablenpointer). Der Befehl holt sich dann von dort die notwendigen Informationen.

```
u$="B"
|DRIVE,@u$
```

3.4 |USER – Umschalten der Benutzer-Ebene

Bis zu 16 verschiedene Benutzer (user=Benutzer) können sich eine Diskette miteinander teilen. Den Benutzern ist eine User-Nummer (Benutzer-Ebene) zwischen 0 und 15 zugeordnet. Im Einschaltzustand (Reset) befinden Sie sich immer in der Benutzer-Ebene 0. Normalerweise beziehen sich alle Kommandos ausschließlich auf die aktuelle Benutzer-Ebene. Der CAT oder |DIR-Befehl zeigt ausschließlich die Programme an, die in dieser Ebene gespeichert sind. Das Löschen von Programmen ist ebenfalls nur in der aktuellen Ebene möglich. Mit

```
|USER,x
```

schalten sie die Diskettenstation auf die durch „x" angegebene Ebene um (x=0 bis 15). Bei der Eingabe einer unzulässigen Usernummer erfolgt die Fehlermeldung „Bad Command". Die Anwendung des |USER-Befehls ist nur dann ratsam, wenn Sie sich eine Diskette tatsächlich mit einem weiteren Benutzer teilen. Sind Programme auf vielen Ebenen verteilt, fehlt Ihnen bald der Überblick, wo welche Programme liegen. Der Befehl ist identisch mit dem CP/M-Befehl USER.

3.5 |DIR – die Directory

Fast identisch mit dem CAT-Befehl ist der |DIR-Befehl (directory=Adressbuch, Inhaltsverzeichnis). Im Gegensatz zum CAT-Befehl fehlt die Angabe

zur Programmgröße und es ist eine Auswahl bestimmter Files, die angezeigt werden sollen, möglich. Der Befehl

```
|DIR
```

zeigt alle Files der aktuellen Benutzer-Ebene an. Folgt hinter dem |DIR-Befehl eine Option in Form einer Stringvariablen, wird aus der Directory eine bestimmte Gruppe von Files selektiert und angezeigt. Dabei ist die Verwendung von Wildcards (siehe 5.0) zulässig.

```
s$=''*.BAS''
|DIR,@s$
```

Listet alle Files vom Typ „.BAS" in der Directory auf.

Die Angabe von ''*.*'' entspricht dem Befehl |DIR. Dieser Befehl ist ebenfalls identisch mit dem DIR-Befehl des CP/M-Betriebssystems.

3.6 |REN – File umbenennen

Mit |REN (rename=umbenennen) kann ein File auf der Diskette einen neuen Namen erhalten. Dabei sind dem Befehl zwei eindeutige Filenamen mit Typenangabe zu übergeben. Die Übergabe der Namen erfolgt wie beim |DRIVE, |DIR und |ERA-Befehl mit Stringvariablen, die vorher zugewiesen sein müssen.

```
alt$=''ALTNAME.BAS''
neu$=''NEUNAME.BAS''
|REN,@neu$,@alt$
```

Nach dem |REN-Befehl folgt, jeweils durch Komma getrennt, der neue und dann der alte Name. Den Variablen ist wieder das „@"-Zeichen vorzustellen. Ist ein unzulässiger Filename angegeben (Wildcards, länger als 8 Zeichen etc.) erfolgt eine Fehlermeldung (Bad command), ebenso wenn der umzubenennende File nicht auf der Diskette vorhanden ist (filename.typ not found). Dieser Befehl ist identisch mit dem CP/M-Befehl REN.

3.7 |ERA – File(s) löschen

Dieser letzte Diskettenbefehl (|ERA) löscht einen oder mehrere Files auf der Diskette (erase=löschen). Nach dem Befehl folgt durch Komma getrennt ein String, der aus einem eindeutigen Filenamen oder auch aus Wildcards bestehen darf.

```
k$=''TEST.DAT''
|ERA,@k$
```

Das Beispiel löscht den File „TEST.DAT" aus der Directory. Ist der File nicht auf der Diskette, meldet sich der Rechner mit „filename.typ not found". Bei der Benutzung von Wildcards werden alle Files gelöscht, bei denen die Bezeichnung zutrifft.

```
k$=''*.BAS''
|ERA,@k$
```

Diese Befehlsfolge löscht alle Files vom Typ „.BAS" auf der Diskette (Wildcards siehe 5.0). Zum Löschen der gesamten Diskette ist ''*.*'' anzugeben. Im Gegensatz zum ERA-Befehl des CP/M-Betriebssystems findet hier keine Sicherheitsabfrage statt, sondern es wird sofort gelöscht. Seien Sie mit dem |ERA-Befehl darum immer besonders vorsichtig.

4.0 Der Variablenpointer

Etwas Besonderes bietet der sogenannte Variablenpointer „@", den Sie schon bei einigen Diskettenbefehlen kennengelernt haben (|REN, |ERA, usw.). Diese Funktion gibt die Adresse wieder, an der eine bestimmte Variable abgelegt ist. Besonders interessant ist dieses für die Stringübergabe in Maschinenprogrammen. Doch auch in BASIC-Programmen läßt sich diese Funktion hervorragend anwenden (siehe DATA-FLEX 9.3).

```
PRINT @variable
```

liefert die Adresse, an der die Variable innerhalb des RAM-Speichers abgelegt ist. Ist der Variablen noch kein Wert zugewiesen worden (unbenutzt), erfolgt die Fehlermeldung „Improper argument", andernfalls ein numerischer Wert zwischen 0 und 65535.

Der CPC-464/664 unterscheidet drei verschiedene Variablentypen, die unterschiedlich viel Speicherraum zur Ablage eines Variableninhaltes benötigen. Der durch „PRINT @variable" ermittelte Wert zeigt jeweils auf das 1. Byte der im folgenden beschriebenen Felder:

1.) Integer (v%) Da eine Integervariable nur ganzzahlige Werte zwischen 0 und 65535 annehmen kann, benötigt sie nur 2 Bytes Speicherraum. Das 1. Byte bestimmt das Lowbyte und das 2. Byte das Highbyte der Integervariablen.

2.) Real (v) Eine Real- oder Fließkommavariable benötigt insgesamt 5 Bytes, um Werte zwischen 1.7E+38 und 1.7E-38 darstellen zukönnen. Die ersten 4 Bytes bilden die Mantisse, das Byte 5 den Exponenten.

3.) String (v$) Bei Stringvariablen (Zeichenketten) finden Sie ein 3 Byte großes „Zeigerfeld" (Stringdescriptor) vor, das im 1. Byte die Länge des Strings und in den folgenden 2 Bytes die Adresse wiedergibt (Lowbyte/Highbyte), wo der Stringinhalt zu finden ist.

Das nachfolgende Beispiel zeigt, wie z.B. eine Stringvariable innerhalb des RAM-Speichers gefunden und angezeigt werden könnte. Der Vorgang ist in Maschinensprache identisch und die Betriebssystemroutinen für |REN oder |ERA verfahren genauso (Übergabe der Adresse des Zeigerblocks, ermitteln der Stringlänge, Zeichen lesen und ggf. prüfen).

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                Variablenpointer               *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 2
1060 PRINT " >Variablenpointer"
1070 PRINT
1080 v$="Ich habe Durst"
1090 PRINT " >Habe die Variable v$=";CHR$(24);v$;CHR$(24);
1100 PRINT " angelegt !":PRINT
1110 PRINT " >Zeigerblock ab Adresse :";@v$
1120 l=PEEK(@v$)
1130 PRINT " >Laenge des Strings     :";l
1140 adr=PEEK(@v$+1)+256*PEEK(@v$+2)
1150 PRINT " >String liegt ab Adresse:";adr
1160 PRINT " >Stringinhalt           : ";
1170 FOR i=adr TO adr+l-1
1180 PRINT CHR$(PEEK(i));
1190 NEXT i
1200 END
1210 REM *************************************************
```

Der umgekehrte Weg, bestimmte Speicherinhalte einer Variablen zuzuweisen, ist besonders reizvoll, da sie somit extrem schnell getauscht werden können. Auch hierzu ein Beispiel:

```
100 MEMORY &9FFF
110 FOR i = &A000 TO &A000+254
120 POKE i,ASC("A")
130 NEXT i
140 s$=""
150 PRINT "String zuweisen":t=TIME
160 FOR i = &A000 TO &A000+254
170 s$=s$+CHR$(PEEK(i))
180 NEXT i
190 PRINT (TIME-t)/300;"Sekunden"
200 PRINT s$
```

In einem geschützten Speicherraum (hier ab Adresse &A000) werden 255 ASCII-Zeichen abgelegt („A"). Danach werden alle 255 Zeichen (maximale Länge eines Strings) in die Variable „s$" eingelesen und anschließend angezeigt. Das Zuweisen dieser Stringvariablen mit 255 Zeichen dauert fast 1.5 Sekunden. Wesentlich schneller geht das Zuweisen einer Variablen, indem einfach der Zeigerblock eines Strings auf diese 255 Zeichen gesetzt wird.

```
100 MEMORY &9FFF
110 FOR i = &A000 TO &A000+254
120 POKE i,ASC("A")
130 NEXT i
140 s$="dummy"
150 PRINT "String zuweisen":t=TIME
160 POKE @$,255      :'Stringlaenge
170 POKE @s$+1,&00   :'Adresse Lowbyte
180 POKE @s$+2,&A0   :'Adresse Highbyte
190 PRINT (TIME-t)/100;" 1/100 Sekunden
200 PRINT s$
```

Die Zuweisung der 255 Zeichen dauert nur noch rund 4/100 Sekunden !! Schreiben Sie die POKEs in den Zeilen 160-180 dann noch in eine Zeile, liegt der Zeitbedarf bei 2/100 Sekunden. Das ist fast 100 mal schneller als das normale Zuweisen einer Variablen. Im Programm „DATA-FLEX" (siehe 9.3) finden Sie ein Anwendungsbeispiel.

5.0 Die Wildcards „?“ und „*“

Bei einigen Disketten- und CP/M-Kommandos ist die Benutzung von sogenannten „Wildcards“ zulässig. Dabei unterscheidet das AMSDOS (CP/M) zwei verschiedene Zeichen:

* Steht an Stelle des Filenamens und/oder der Typenbezeichnung der Stern („*“), ersetzt dieser den gesamten Namen oder Typen.

*.BAS	selektiert alle Dateien vom Typ „.BAS“.
TEST.*	selektiert alle Dateien mit dem Namen „TEST“.
.	Alle Dateien.

? Das Fragezeichen („?“) kennzeichnet ein nicht relevantes Zeichen innerhalb des Filenamens und/oder der Typenbezeichnung.

T??.BAS	selektiert alle Dateien, die einen 3-stelligen Namen tragen, mit „T“ beginnen und vom Typ „.BAS“ sind.
TEST.B??	selektiert alle Dateien, deren Typ mit „B“ beginnt (z.B. BAS, BAK, BIN) und die den Namen „Test“ tragen.

Die Angabe „????????.???“ entspricht der Angabe „*.*“.

Die Anwendung der Wildcards findet vor allen in den Befehlen |DIR (DIR) und |ERA (ERA) statt. Beim Directory-Befehl können so nur bestimmte Programme angezeigt werden. Die Kommandofolge

```
s$="*.BAS"
|DIR,@s$
```

zeigt z.B. nur die Programme vom Typ „.BAS“ an. Achten Sie besonders beim |ERA- (ERA) Befehl auf die Benutzung der Wildcards, damit Sie sich keine wichtigen Dateien zerstören.

6.0 Datei-Typ (Filetyp)

Im AMSDOS und im CP/M ist vorgesehen, daß nach dem 8-stelligen Filenamen eine durch den Dezimalpunkt „.“ getrennte Typenkennzeichnung folgt. Grundsätzlich kann der Typ wahlfrei aus bis zu 3 ASCII-Zeichen gebildet werden und ist eine zusätzliche Information zum Filenamen. Einige Typenkennzeichnungen werden allerdings vom AMSDOS oder CP/M unterstützt und sind somit zwingend (z.B. „.COM“ für unter CP/M ladbare Programme). Im folgenden finden Sie eine Liste von gebräuchlichen Typenkennzeichnungen. Auch Kassettenbenutzer sollten sich an diese Kennzeichung gewöhnen und ihre Programme auf der Kassette mit einem 8-stelligen Filenamen und der Typenkennzeichnung sichern.

.ASC	ASCII-File
.BAK	Backup-File
.BAS	BASIC-Programm
.BIN	Binäre Datei
.COM	Unter CP/M ausführbares Programm
.DAT	Datei allgemein
.HEX	File im Intel-HEX-Format
.OBJ	Objektcode (Maschinenprogramm)
.PRN	Assembler-Dokumentations-Datei
.SEQ	Sequentielle Datei
.SUB	Kommando-Datei (SUBMIT)
.TXT	Textfile
.$$$	Temporäre Datei

7.0 Datenverwaltung

Die Verwaltung von großen Datenmengen ist heute das Hauptanwendungsgebiet der Computer schlechthin. Sei es eine Datenbank oder nur eine einfache Adressdatei: Es geht immer um den (möglichst) schnellen Datenzugriff. Durch den Einsatz von Diskettenstationen ist auch im sogenannten „Heimcomputer-Bereich" die Verwaltung von erheblichen Datenmengen möglich, die weit über den Bedarf des „Hausgebrauchs" liegen. Auf eine 3 Zoll Diskette des DDI-1 können immerhin 184.320 Bytes Informationen untergebracht werden. D.h. Sie könnten eine Adressdatei mit über 1800 Einträgen (Name, Ort, etc. mit jeweils 100 Zeichen) verwalten! Das ist für einen kleinen Betrieb durchaus ausreichend. Heute bieten einige Firmen schon Disketten mit einer Speicherkapazität von 760 kByte für den CPC-464/664 an. Hier könnten Sie schon fast 8000 Adresseinträge verwalten.

Es gibt eine Reihe von Datenverwaltungsmöglichkeiten, die alle ihre Vorzüge besitzen. Grundsätzlich ist zu sagen, daß der Zugriff auf Daten, die sich innerhalb des RAM-Speichers des Rechners befinden, am schnellsten stattfinden kann. Kleinere Datenmenge sollten deshalb vor ihrer Bearbeitung komplett in den Rechner geladen werden. Große Datenmengen, die nicht mehr komplett in den Rechner passen, sind auf die Kassette oder Diskette auszulagern. Bei Bedarf werden dann die benötigten Datensegmente (Name, Ort,PLZ..) in den Rechner geladen und bearbeitet.

Das AMSDOS unterstützt von BASIC aus nur die sequentielle Verwaltung von Daten. Im Kassettenbetrieb ist generell nur diese Verwaltungsform möglich. Hierbei sind alle Daten „hintereinander" abgelegt und nur in dieser Form wieder einzulesen (ggf. überlesen). Ein direktes Ändern eines Bytes der sequentiellen Datei ist nur durch Bearbeitung der gesamten Datei möglich.

Obwohl das AMSDOS intern den Direktzugriff auf Daten ermöglicht (Random Access), ist diese Fähigkeit nicht im BASIC eingebunden. Nur mit Hilfe eines kleinen Maschinenprogramms, das diese Routinen des AMSDOS nutzt, kann auch diese Form der Datenverwaltung genutzt werden.

Die folgenden Kapitel des Buches gehen ausführlich auf die fünf wichtigsten Arten der Datenverwaltung ein.

* Verwaltung in DATA-Zeilen (Konstanten)
* Sequentielle Datei
* RAM-Floppy
* Direktzugriff (Random Access)
* Index-Datei (im Zusammenhang mit der Random-Datei)

Außer dem Direktzugriff auf Daten (siehe 10.0) sind alle anderen Methoden auch mit dem Datenrecorder anwendbar.

7.1 DATA-Zeilen – Liste von Konstanten

Die einfachste und wohl auch bekannteste Form, Daten in ein Programm einzubinden, ist die Möglichkeit der DATA-Zeilen. Die DATA-Anweisung ist kein Befehl im eigentlichen Sinne, sondern eine Kennzeichnung, daß in der Zeile eine Folge von Daten liegt. Alle DATA-Zeilen bilden einen gemeinsamen „Datenblock" und werden vom Programmablauf ignoriert. Es ist sinnvoll, alle DATA-Zeilen gesammelt ans Ende des Programms zu legen (schnellere Verarbeitung und bessere Übersicht).

Die Handhabung von Daten, die in DATA-Zeilen eingebunden sind, ist im Gegensatz zu anderen Methoden der Datenverwaltung sehr unkomfortabel. Die Daten müssen bei der Programmerstellung fest in das Programm eingebunden werden. Eine Änderung dieser Daten ist später vom Programm aus nicht mehr (oder nur mit Tricks) möglich. Aus diesem Grunde bietet sich diese Form der Datenverwaltung nur für Konstanten (Daten die nicht veränderbar sind) an.

Nach der DATA-Anweisung kann eine beliebige Liste von numerischen und/oder String-Konstanten folgen. Als Trennzeichen zwischen den Konstanten steht jeweils ein Komma („,"). Die Länge einer DATA-Zeile ist nur durch die maximale Aufnahmekapazität einer BASIC-Zeile begrenzt (255 Zeichen). Am Ende einer DATA-Zeile darf kein Komma stehen.

```
DATA konstante 1, konstante 2, .... konstante n
```

Die Konstanten können sowohl numerischen oder String Charakter aufweisen. Stringkonstanten müssen nur dann in Anführungsstrichen stehen, wenn die

Stringkonstante Trennzeichen, wie das Komma und/oder den Doppelpunkt enthält. Die Anführungsstriche sind allerdings kein Bestandteil des Strings, sondern schließen diesen nur ein.

```
      DATA Montag,Dienstag,Mittwoch
oder  DATA ''Montag'',''Dienstag'',''Mittwoch''
```

Die Stringkonstante

```
DATA ''Werteliste : 1,2,3,4,5''
```

muß, da sie Trennzeichen beinhaltet, zwischen Anführungsstrichen stehen. Reicht eine DATA-Zeile nicht zur Ablage aller Konstanten, können beliebig viele DATA-Zeilen folgen.

```
10000 DATA Montag,Dienstag,Mittwoch
10010 DATA Donnerstag,Freitag,Samstag
10020 DATA Sonntag
```

7.2 READ und der DATA-Zeiger

Der READ-Befehl (read=lesen) hat die Aufgabe, die Konstante, auf die ein sogenannter „DATA-Zeiger" zeigt, einer Variablen zuzuweisen. Der DATA-Zeiger weist nach einem Programmstart (RUN oder einem RESTORE-Befehl) stets auf die erste Konstante einer DATA-Liste (eine oder mehrere DATA-Zeilen). Sind mehrere DATA-Zeilen vorhanden, zeigt er auf das erste Element der DATA-Zeile mit der niedrigsten Zeilennummer. Nach dem Einlesen verschiebt sich der DATA-Zeiger automatisch und zeigt auf das nächste Element des DATA-Blocks.

```
READ variable ⟨,variable, ....⟩
```

Mit dem Read-Befehl kann die aktuelle Konstante aus der Datenliste der Variablen zugewiesen werden. Gleichfalls ermöglicht der Befehl das Einlesen mehrerer Konstanten mit einem Aufruf. Dabei sind die Befehle:

```
      READ A: READ B
und   READ A,B
```

identisch. Beim Einlesen einer Konstanten ist stets darauf zu achten, daß der Variablentyp mit der einzulesenden Konstanten übereinstimmt. Andernfalls kann die Fehlermeldung „Syntax Error" auftreten oder aber es stehen falsche Daten in den Variablen.

```
100 FOR i=1 TO 3
110 READ a$
120 PRINT a$
130 NEXT i
140 END
150 DATA ''Heute'',''Morgen'',''Immer''
```

Das kleine Beispielprogramm liest nacheinander drei Stringkonstanten in die Variable „a$" und gibt sie auf dem Bildschirm aus. Wird versucht, eine vierte, nicht vorhandene Konstante einzulesen, beendet der Rechner seine Arbeit mit der Meldung

```
DATA exhausted
```

und zeigt damit eine Überschreitung des DATA-Blocks an. Es ist die Aufgabe des Programmierers, sicherzustellen, daß niemals mehr Konstanten eingelesen werden als tatsächlich vorhanden sind.

Der Zugriff auf eine bestimmte Konstante (z.B. die 7. Konstante der Datenliste) ist nur mit einem kleinen Programmiertrick möglich, da ein Positionieren des DATA-Zeigers in dieser Form nicht vorgesehen ist. Das Verfahren ist denkbar einfach: Vom Beginn der Datenliste werden so viele Konstanten „überlesen", bis der DATA-Zeiger auf die gewünschte Konstante zeigt. Um die 7. Konstante einzulesen, müssen 6 Konstanten überlesen werden, dann zeigt der DATA-Zeiger auf die 7. Konstante. Das ist mit einer kleinen Schleife leicht zu realisieren:

```
100 FOR i=1 TO 6
110 READ dummy$
120 NEXT i
130 READ gefunden$
```

Bei diesem Verfahren ist es notwendig, den DATA-Zeiger vor dem Zugriff auf das erste Element der Datenliste zu positionieren, damit die gewünschte Konstante richtig gefunden wird (siehe 7.3.).

7.3 RESTORE – Rücksetzen des DATA-Zeigers

Mit Hilfe des RESTORE-Befehls läßt sich der DATA-Zeiger auf das erste Element der Datenliste positionieren (restore = wiederherstellen).

RESTORE ⟨zeilennummer⟩

Folgt nach dem RESTORE-Befehl keine Zeilennummer, so wird der DATA-Zeiger auf das erste Element der gesamten Datenliste zurückgesetzt. Mit einer nachgestellten Zeilennummer ist es weiterhin möglich, den DATA-Zeiger auf das erste Element der angegebenen Programmzeile (DATA-Zeile) zu positionieren. Leider darf die Zeilennummer keine Variable sein (Syntax Error), sondern muß im Programm als Wert festgelegt sein. Diese fehlende Möglichkeit des variablen oder berechneten RESTOREs ist bei der Behandlung großer Konstantenmengen äußerst störend. Um aus einer Datenliste von 1000 Elementen die 900. Konstante einzulesen, müssen immerhin 899 Konstanten überlesen werden. Der Zeitaufwand ist nicht unerheblich, gerade dann, wenn viele Suchoperationen innerhalb der Datenliste notwendig sind.

Das nachfolgende Programm ermöglicht durch einen Trick auch ein berechnetes RESTORE. Die Zeilennummer, auf die der DATA-Zeiger gesetzt werden soll, muß sich vor Aufruf dieses Unterprogramms in der Variablen „znr" befinden. Nach Rückkehr aus diesem Unterprogramm ist der DATA-Zeiger auf die Zeilennummer „znr" positioniert.

Der Trick besteht darin, daß vor Erreichen des RESTORE-Befehls in Programmzeile 65150 der Zeilennummernwert (hier 65535 als Dummy-Zeilennummer) direkt im Speicher geändert wird. Die Zeile 65120 ermittelt die Speicheradresse, in der die momentane Programmzeile im RAM beginnt. Aus den Adressen &AE36 und &AE37 können Sie die Adresse der momentanen Programmzeile erfahren (hier der Zeile 65120). Die Zeilennummer des RESTORE-Befehls liegt 109 Bytes höher im RAM (Lowbyte, Highbyte). Die Zeilen 65130 und 65140 legen die neue Zeilennummer „znr" genau in diese Speicherstellen (Lowbyte und Highbyte von „znr"). Bei Erreichen der Zeile 65150 ist die Zeilennummer geändert und der RESTORE-Befehl arbeitet einwandfrei. Bitte beachten Sie, daß die Zeilen 65130-65150 genauso eingegeben werden müssen! Es dürfen keine weiteren Zeichen in diesen Zeilen stehen (auch keine Leerzeichen am Ende der Programmzeile), da sonst der Abstand von 109 Bytes bis zur Zeilennummer des RESTORE-Befehls nicht mehr gegeben ist!!!

```
65000 REM ************************************* M.W.Thoma **
65010 REM *                                                *
65020 REM *            Variabler RESTORE-Befehl            *
65030 REM *                                                *
65040 REM *      In 'ZNR' muss die Zeilennummer liegen     *
65050 REM *                                                *
65060 REM *    ACHTUNG ! Die Zeilen 60130 - 60150 MUESSEN  *
65070 REM *    genauso eingeben werden !! Es duerfen auch  *
65080 REM *    am  Ende der Zeilen keine  ueberfluessigen  *
65090 REM *               Leerzeichen stehen!              *
65100 REM *                                                *
65110 REM **************************************************
65120 znradr=PEEK(&AE37)*256+PEEK(&AE36)
65130 POKE znradr+109,&FF AND UNT(znr)
65140 POKE znradr+110,INT(znr/256)
65150 RESTORE 65535
65160 RETURN
65170 REM **************************************************
```

Das Programm „Variabler RESTORE" ist im Speicher des Rechners frei verschiebbar (RENUM).

Die Anwendung des variablen RESTORE-Befehls bietet erhebliche Vorteile, wie das folgende Beispiel verdeutlichen soll:
Eine Datenliste von 1000 Daten liegt von der Zeilennummer 10000 aufwärts. In jeder DATA-Zeile befinden sich genau 10 Konstanten. Die Schrittweite der Zeilennummer ist auf 10 festgelegt. Daraus folgt, daß die 1. Konstante in der Zeile 10000 an 1. Stelle liegt, die 12. in der Zeile 10010 an 2. Stelle usw. Mit einer einfachen Rechnung läßt sich feststellen, in welcher Zeile und an welcher Position innerhalb der Zeile die gewünschte Konstante liegt.

$$
\begin{aligned}
z &= \text{INT}((k-1)/10) \\
p &= (k-1)\ \text{MOD}\ 10 \\
\text{Zeilennummer} &= z*10+10000 \\
\text{Position} &= p+1
\end{aligned}
$$

Die Variable „k" gibt die gewünschte Konstante (1-1000) an. Jetzt braucht nur noch der DATA-Zeiger auf die Zeilennummer positioniert und „p" Konstanten leer eingelesen werden. Ein Zahlenbeispiel:

$$
\begin{aligned}
k &= 98 \\
z &= \text{int}((98-1)/10) = 9 \\
p &= (k-1)\ \text{MOD}\ 10 = 8 \\
\text{znr} &= 9*10+10000 = 10090
\end{aligned}
$$

Die Konstante befindet sich in der Programmzeile 10090. Es müssen 8 Konstanten („p“) überlesen werden, dann zeigt der DATA-Zeiger auf die gewünschte Konstante. D.h. mit maximal 9 Datenzugriffen ist jede gewünschte Konstante gefunden und eingelesen! Ohne den variablen RESTORE-Befehl wären immerhin maximal 999 Zugriffe nötig!

7.4 Das Jahrhundertspiel

Das erste größere Programm ist gleich ein Spiel. Es ist ein Spiel mit einer großen Menge geschichtlicher Daten, die als DATA-Zeilen abgelegt sind. Die Spielidee ist sehr einfach und trotzdem faszinierend. Je nach Schwierigkeitsgrad werden bis zu 20 geschichtliche Kurzinformationen über herausragende Ereignisse dieses Jahrhunderts angezeigt. Die Aufgabe besteht darin, möglichst alle Kurzinformationen so zu ordnen, daß sie in der zeitlich richtigen Reihenfolge liegen. Dabei wird zusätzlich gegen die Zeit gespielt (je schneller desto besser). Damit das Problem nicht zu schwierig ist, sind ausschließlich Daten der deutschen Geschichte vorhanden. Zum verändern der Reihenfolgen stehen 2 Befehle zur Verfügung:

T : Tausche zwei Daten miteinander
L : Lege ein Datum an eine bestimmte Stelle

Weiterhin sind zwei Hilfestellungen vorhanden: „H" legt das erste und letzte Datum an die richtige Stelle. Mit „S" werden alle z.Z. richtigen Daten gekennzeichnet. Beide Hilfestellungen sind jeweils nur einmal pro Spiel möglich und bedingen einen Zeitzuschlag von 20 bzw. 30 Sekunden. Mit „E" für Ende kann jederzeit das Spiel beendet und bewertet werden.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *       Das Jahrhundertspiel 1900 - 1975        *
1030 REM *                                               *
1040 REM *************************************************
1050 MODE 2
1060 DIM j$(20),d$(20),r(20),rr(20)
1070 INK 0,1:INK 1,26:BORDER 5
1080 WINDOW#0,1,80,22,24
1090 WINDOW#1,1,80,1,20
1100 WINDOW#2,1,80,25,25
1110 PAPER 1:PEN 0:CLS
1120 LOCATE#1,21,1
1130 PRINT#1,"D a s  J a h r h u n d e r t s p i e l"
1140 PRINT#1
1150 PRINT#1,," Ihre Aufgabe ist es,die Stichworte ";
1160 PRINT#1, "in die zeitlich"
1170 PRINT#1,," richtige Reihenfolge zu bringen.";
1180 PRINT#1, "Es werden Ihnen je"
1190 PRINT#1,," nach Schwierigkeitsgrad  3 bis 20 ";
1200 PRINT#1, "Daten geliefert."
1210 PRINT#1,," Da diese  Aufgabe   nicht  ";
```

```
1220 PRINT#1, "ganz einfach zu loesen"
1230 PRINT#1,," ist, beginnen Sie mit einer ";
1240 PRINT#1, "einfachen Stufe (3-6)."
1250 PRINT#1
1260 PRINT#1,," Es sind zwei Hilfestellungen moeglich:"
1270 PRINT#1,,"    1.) Mit 'H' werden  Ihnen das ";
1280 PRINT#1, "erste und letzte"
1290 PRINT#1,,"        Datum an die richtige Stelle gelegt."
1300 PRINT#1,,"    2.) Mit 'S' werden  alle z. Z. ";
1310 PRINT#1, "richtigen Daten"
1320 PRINT#1,,"        gekennzeichnet."
1330 PRINT#1
1340 PRINT#1,," Mit den Befehlen:"
1350 PRINT#1,,"    <T>ausche werden 2 Daten ";
1360 PRINT#1, "miteinander vertauscht."
1370 PRINT#1,,"    <L>ege ein Datum an die ";
1380 PRINT#1, "bestimmte Stelle."
1390 PRINT#1,,"    <E>nde es werden alle richtigen ";
1400 PRINT#1, "Daten angezeigt"
1410 PRINT#1,,"     und Ihr Ergebnis bewertet."
1420 n$="Nobody"
1430 LOCATE 2,2
1440 t=0:h1=0:h2=0
1450 INPUT "> Ihren Namen bitte (bis 10 Zeichen) :";na$
1460 IF LEN(na$)>10 THEN PRINT CHR$(7):GOTO 1430
1470 IF na$<>"" THEN n$=na$
1480 PRINT#2," Name: ";n$;
1490 PRINT#2,TAB(35);">>        <<",
1500 PRINT#2,TAB(65);"Best: ";USING "########";best;
1510 CLS
1520 LOCATE 2,2
1530 INPUT ">Schwierigkeitsgrad (3-20) :";sg
1540 IF sg<3 OR sg>20 THEN CLS:END
1550 CLS:CLS#1
1560 '
1570 REM ************************************************
1580 REM *                                              *
1590 REM *            Zufaellige Daten einlesen         *
1600 REM *                                              *
1610 REM ************************************************
1620 LOCATE 2,2
1630 PRINT"> Bitte warten ..... suche Daten !!!"
1640 FOR i=1 TO sg
1650 z=INT(RND(1)*200)+1
1660 FOR c=1 TO i-1
1670 IF z=r(c) THEN 1650
1680 NEXT c
1690 znr=3190+z*10:GOSUB 65000
1700 READ j$(i),d$(i)
```

```
1710 r(i)=z:rr(i)=z
1720 NEXT i
1730 FOR i=1 TO sg-1
1740 FOR j=i+1 TO sg
1750 IF rr(i)<rr(j) THEN 1770
1760 rr(0)=rr(i):rr(i)=rr(j):rr(j)=rr(0)
1770 NEXT j
1780 NEXT i
1790 GOSUB 2920
1800 '
1810 REM ************************************************
1820 REM *                                              *
1830 REM *                    M e n u e                 *
1840 REM *                                              *
1850 REM ************************************************
1860 EVERY 50 GOSUB 3040
1870 EI:t=0:GOSUB 3040
1880 CLS
1890 LOCATE 2,2
1900 PRINT "> Ihr Kommando ";
1910 k$=INKEY$:IF k$="" THEN 1910
1920 k$=UPPER$(k$)
1930 IF k$="E" THEN DI:GOTO 2470
1940 IF k$="T" THEN GOSUB 2010:GOTO 1880
1950 IF k$="L" THEN GOSUB 2190:GOTO 1880
1960 IF k$="H" AND h1=0 THEN GOSUB 2640:GOTO 1880
1970 IF k$="S" AND h2=0 THEN GOSUB 2800:GOTO 1880
1980 PRINT CHR$(7)
1990 GOTO 1890
2000 '
2010 REM ************************************************
2020 REM *                                              *
2030 REM *            Tauschen zweier Daten             *
2040 REM *                                              *
2050 REM ************************************************
2060 CLS
2070 LOCATE 2,2
2080 INPUT "> Tausche ";t1
2090 IF t1<1 OR t1>sg THEN PRINT CHR$(7):RETURN
2100 LOCATE 20,2
2110 INPUT "> mit ";t2
2120 IF t2<1 OR t2>sg THEN PRINT CHR$(7):RETURN
2130 j$(0)=j$(t1):d$(0)=d$(t1):r(0)=r(t1)
2140 j$(t1)=j$(t2):d$(t1)=d$(t2):r(t1)=r(t2)
2150 j$(t2)=j$(0):d$(t2)=d$(0):r(t2)=r(0)
2160 GOSUB 2920
2170 RETURN
2180 '
```

```
2190 REM ************************************************
2200 REM *                                              *
2210 REM *       Datum an eine bestimmte Stelle legen   *
2220 REM *                                              *
2230 REM ************************************************
2240 CLS
2250 LOCATE 2,2
2260 INPUT "> Lege Nummer ";t1
2270 IF t1<1 OR t1>sg THEN PRINT CHR$(7):RETURN
2280 LOCATE 20,2
2290 INPUT "> an Stelle ";t2
2300 IF t2<1 OR t2>sg THEN PRINT CHR$(7):RETURN
2310 IF t2<t1 THEN 2390
2320 j$(0)=j$(t1):d$(0)=d$(t1):r(0)=r(t1)
2330 FOR i=t1 TO t2-1
2340 j$(i)=j$(i+1):d$(i)=d$(i+1):r(i)=r(i+1)
2350 NEXT i
2360 j$(t2)=j$(0):d$(t2)=d$(0):r(t2)=r(0)
2370 GOSUB 2920
2380 GOTO 2450
2390 j$(0)=j$(t1):d$(0)=d$(t1):r(0)=r(t1)
2400 FOR i=t1 TO t2+1 STEP-1
2410 j$(i)=j$(i-1):d$(i)=d$(i-1):r(i)=r(i-1)
2420 NEXT i
2430 j$(t2)=j$(0):d$(t2)=d$(0):r(t2)=r(0)
2440 GOSUB 2920
2450 RETURN
2460 '
2470 REM ************************************************
2480 REM *                                              *
2490 REM *            Spielende und Bewertung           *
2500 REM *                                              *
2510 REM ************************************************
2520 FOR i=1 TO sg
2530 IF r(i)=rr(i) THEN p=p+1:PRINT#1,CHR$(24);:in=1
2540 LOCATE#1,14,i
2550 PRINT#1,d$(i)+SPACE$(62-LEN(d$(i)))+j$(i)
2560 IF in=1 THEN in=0:PRINT#1,CHR$(24);
2570 NEXT i
2580 p=p*sg*10000/t
2590 IF p>best THEN best=p:SOUND 1,400,30,5
2600 LOCATE#2,21,1:PRINT#2,USING "#######";p:p=0
2610 CLS
2620 GOTO 1430
2630 '
2640 REM ************************************************
2650 REM *                                              *
2660 REM *     Hilfestellung - erstes und letztes Datum *
2670 REM *                                              *
2680 REM ************************************************
```

```
2690 FOR g=1 TO sg
2700 IF r(g)=rr(1) THEN t1=g:t2=1:GOSUB 2310:GOTO 2720
2710 NEXT g
2720 FOR g=2 TO sg
2730 IF r(g)=rr(sg)THEN t1=g:t2=sg:GOSUB 2310:GOTO 2750
2740 NEXT g
2750 LOCATE#1,76,1:PRINT#1,j$(1)
2760 LOCATE#1,76,sg:PRINT#1,j$(sg)
2770 t=T+20:h1=1
2780 RETURN
2790 '
2800 REM ************************************************
2810 REM *                                              *
2820 REM *  Hilfestellung - Zeigen aller richtigen Daten  *
2830 REM *                                              *
2840 REM ************************************************
2850 FOR i=1 TO sg
2860 LOCATE#1,76,i
2870 IF r(i)=rr(i) THEN PRINT#1,j$(i)
2880 NEXT i
2890 T=t+30:h2=1
2900 RETURN
2910 '
2920 REM ************************************************
2930 REM *                                              *
2940 REM *         Ausgaberoutine fuer die Daten        *
2950 REM *                                              *
2960 REM ************************************************
2970 LOCATE#1,1,1
2980 FOR i=1 TO sg
2990 PRINT#1," ";CHR$(24);USING"##";i;
3000 PRINT#1," ";CHR$(24),d$(i)+SPACE$(66-LEN(d$(i)))
3010 NEXT i
3020 RETURN
3030 '
3040 REM ************************************************
3050 REM *                                              *
3060 REM *        Interruptroutine fuer die Zeit        *
3070 REM *                                              *
3080 REM ************************************************
3090 t=t+1
3100 LOCATE#2,38,1
3110 PRINT#2,USING "######";t
3120 RETURN
3130 REM ************************************************
3140 '
3150 REM ************************************************
3160 REM *                                              *
3170 REM *    Geschichtliche Daten in zeitlicher Folge  *
3180 REM *                                              *
3190 REM ************************************************
```

```
3200 DATA 1900,Graf von Buelow deutscher Reichskanzler
3210 DATA 1900,Buergerliches Gesetzbuch tritt in Kraft
3220 DATA 1900,Erster Flug des Luftschiffes 'Zeppelin'
3230 DATA 1900,C.A. von Welsbach erfindet Osmium-Gluehlampe
3240 DATA 1900,Max Planck begruendet die Quantenphysik
3250 DATA 1900,Sigmund Freud schreibt 'Traumdeutung'
3260 DATA 1901,Britisch-deut. Buendnisgespraeche scheitern
3270 DATA 1901,Medizin-Nobelpreis an Emil von Behringer
3280 DATA 1901,Physik-Nobelpreis an Wilhelm Roentgen
3290 DATA 1901,Thomas Mann schrieb 'Buddenbrooks'
3300 DATA 1901,Konsruktion des 'Mercedes'
3310 DATA 1902,Literatur-Nobelpreis an Theodor Mommsen
3320 DATA 1902,Rudolf Bosch erfindet Zuendkerze
3330 DATA 1903,Aufstand der Hottentotten in S.W. Afrika
3340 DATA 1903,Oskar von Miller gruendet 'Deutsches Museum
3350 DATA 1905,Medizin-Nobelpreis an Robert Koch
3360 DATA 1905,Albert Einstein - spez. Relativitaetstheorie
3370 DATA 1905,Gruendung des 'Deutschen Staedtetages'
3380 DATA 1905,Otto Hahn entdeckt Radiothor
3390 DATA 1906,Serumdiagnose der Syphillis
3400 DATA 1907,Chemie-Nobelpreis an Edmund Buchner
3410 DATA 1907,Geheimes Wahlrecht in Oesterreich
3420 DATA 1908,Fund des aeltsten menschlichen Skeletteils
3430 DATA 1909,Reichskanzler von Buelow tritt zurueck
3440 DATA 1909,Entdeckung des 'Salvarsan' (Syphillis-Mittel)
3450 DATA 1910,Zarenbesuch in Potsdam
3460 DATA 1910,Erster Dieselmotor fuer KFZ
3470 DATA 1911,Hugo von Hofmannsthal schrieb 'Jedermann'
3480 DATA 1911,Beginn der Pfadfinderbewegung
3490 DATA 1912,SPD staerkste Partei im Reichstag
3500 DATA 1912,Literatur-Nobelpreis an Gerhard Hauptmann
3510 DATA 1913,Groenlandforschungen A.Wegeners/J.P. Koch
3520 DATA 1914,Ermordung von Franz Ferdinand in Sarajevo
3530 DATA 1914,Beginn erster Weltkrieg
3540 DATA 1915,Stellungskrieg im Westen
3550 DATA 1915,Albert Einstein allg. Relativitaetstheorie
3560 DATA 1915,Alfred Wegner Kontinentalverschiebungstheorie
3570 DATA 1916,Seeschlacht vor dem Skagerrak
3580 DATA 1917,Gruendung der Salzburger Festspiele
3590 DATA 1918,Hamburger Aufstand
3600 DATA 1918,Revolution in Muenchen
3610 DATA 1918,Ende des ersten Weltkriegs
3620 DATA 1918,Gesetzlicher 8-Stunden-Arbeitstag
3630 DATA 1918,Physik-Nobelpreis an Max Planck
3640 DATA 1919,Gruendung der Kommunistischen P. Deutschlands
3650 DATA 1919,Ermordung von K. Liebknecht und R. Luxemburg
3660 DATA 1919,Weimarer Verfassung
3670 DATA 1920,Versailler Vertrag tritt in Kraft
3680 DATA 1920,Kapp-Putsch in Berlin
```

```
3690 DATA 1920,Gruendung der 'NSDAP'
3700 DATA 1921,Anerkennung der Reparationsforderungen
3710 DATA 1921,Adolf Hitler wird Vorsitzender der 'NSDAP'
3720 DATA 1921,Physik-Nobelpreis an Albert Einstein
3730 DATA 1923,Ruhrgebietsbesetzung
3740 DATA 1923,Erste Rundfunk-Unterhaltungssendung
3750 DATA 1924,Thomas Mann schrieb 'Der Zauberberg'
3760 DATA 1925,Hindenburg wird Reichspraesident
3770 DATA 1925,Gruendung des 'Bauhauses'
3780 DATA 1926,Friedens-Nobelpreis an Gustav Stresemann
3790 DATA 1926,Gruendung der Deutschen Lufthansa
3800 DATA 1926,Erwin Schroedinger begruendet Wellenmechanik
3810 DATA 1926,Tibet-Expedition Wilhelm Filchners
3820 DATA 1927,Arbeitslosenversicherung eingefuehrt
3830 DATA 1927,Erste oeffentliche Bildtelegrafenlinie
3840 DATA 1928,Indienststellung von 'Graf Zeppelin'
3850 DATA 1928,Brecht 'Dreigroschenoper'
3860 DATA 1928,Erfindung des Elektronenzaehlrohrs
3870 DATA 1929,'Graf Zeppelin' umfliegt die Erde
3880 DATA 1929,Literatur-Nobelpreis an Thomas Mann
3890 DATA 1929,Herzkatheterisierung (W. Forssmann)
3900 DATA 1930,Beginn der Weltwirtschaftskrise
3910 DATA 1930,Sigmund Freud 'Unbehagen in der Kultur'
3920 DATA 1930,Baubeginn Grossglocknerstrasse
3930 DATA 1931,Chemie-Nobelpreis an Carl Bosch
3940 DATA 1932,Physik-Nobelpreis an W. Heisenberg
3950 DATA 1932,NSDAP staerkste 'Partei'
3960 DATA 1933,Adolf Hiter zum Reichskanzler ernannt
3970 DATA 1933,Reichstagsbrand
3980 DATA 1933,Ermaechtigungsgesetz
3990 DATA 1933,Aufloesung der Gewerkschaften
4000 DATA 1933,Gruendung der 'Gestapo'
4010 DATA 1933,Verbot der SPD
4020 DATA 1933,Austritt aus dem Voelkerbund
4030 DATA 1933,Buecherverbrennung unliebsamer Autoren
4040 DATA 1934,Roehn-Putsch
4050 DATA 1935,Saar-Abstimmung
4060 DATA 1935,Arbeitspflicht fuer deutsche Jugendliche
4070 DATA 1935,Hakenkreuzfahne wird Nationalflagge
4080 DATA 1935,Erstes regelmaessiges Fernsehprogramm
4090 DATA 1936,Olympische Spiele in Berlin
4100 DATA 1936,Carl Orff 'Carmina Burana'
4110 DATA 1937,Annahme des Nobelpreis fuer Deutsche verboten
4120 DATA 1938,Adolf Hitler Oberfehlshaber der Wehrmacht
4130 DATA 1938,Anschluss Oesterreich ans Deutsche Reich
4140 DATA 1938,'Kristallnacht'
4150 DATA 1939,Beginn des 2. Weltkriegs
4160 DATA 1939,Einmarsch deutscher Truppen in Polen
4170 DATA 1939,Nachweis der Spaltbarkeit des Urans (Hahn)
```

```
4180 DATA 1940,Dreimaechtepakt Deutschland-Italien-Japan
4190 DATA 1940,'Luftschlacht um England'
4200 DATA 1940,'Euthanasiegesetze'
4210 DATA 1941,Juden werden gezwungen 'Judenstern' zu tragen
4220 DATA 1941,Beginn von 'Fall Barbarrossa'
4230 DATA 1942,Wannsee-Konferenz
4240 DATA 1942,Attentat auf R. Heydrich in Prag
4250 DATA 1943,Kapitulation in Stalingrad
4260 DATA 1943,Erhebung der Juden im Warschauer Ghetto
4270 DATA 1943,Heeresgruppe Afrika kapituliert
4280 DATA 1943,Physik-Nobelpreis an Otto Stern
4290 DATA 1944,Staufenbergs Attentat auf A. Hitler
4300 DATA 1944,Schlacht bei Arnheim
4310 DATA 1944,Chemie-Nobelpreis an Otto Hahn
4320 DATA 1945,Selbstmord Adolf Hitlers
4330 DATA 1945,Ende der 2. Weltkriegs
4340 DATA 1945,Aufteilung in 4 Besatzungszonen
4350 DATA 1946,'Nuernberger Prozesse'
4360 DATA 1946,Literatur-Nobelpreis an Hermann Hesse
4370 DATA 1946,Carl Zuckmayer 'Des Teufels General'
4380 DATA 1947,Gruendung der Universitaet Saarbruecken
4390 DATA 1947,Thomas Mann 'Doktor Faustus'
4400 DATA 1948,Waehrungsreform
4410 DATA 1948,Berliner Blockade 'Luftbruecke'
4420 DATA 1948,Gruendung der freien Universitaet Berlin
4430 DATA 1949,Vorlaeufiges Grundgesetz (BRD)
4440 DATA 1949,Wahlen zum 1. Bundestag
4450 DATA 1949,Theodor Heuss 1. Bundespraesident
4460 DATA 1949,Konrad Adenauer wird Bundeskanzler
4470 DATA 1949,'Petersburger Abkommen'
4480 DATA 1949,Verfassung der DDR
4490 DATA 1950,Aufhebung der Lebensmittelrationierung (BRD)
4500 DATA 1950,Anerkennung der Oder-Neisse-Linie
4510 DATA 1950,Chemie-Nobelpreis an Otto Diels
4520 DATA 1951,BRD wird Mitglied des Europarates
4530 DATA 1952,Wiedergutmachungsabkommen BRD-Israel
4540 DATA 1952,Tod von Kurt Schumacher
4550 DATA 1952,Friedens-Nobelpreis an Albert Schweitzer
4560 DATA 1953,'Volksaufstand' in Ost-Berlin und DDR
4570 DATA 1953,Chemie-Nobelpreis an Hermann Staudinger
4580 DATA 1954,Beginn des 'Wirtschaftswunder'
4590 DATA 1954,Entdeckung des 'Moessbauer-Effekt'
4600 DATA 1955,UDSSR beendet formel Krieg mit Deutschland
4610 DATA 1955,'Hallsteindoktrien'
4620 DATA 1956,Allgemeine Wehrpflicht (BRD)
4630 DATA 1956,KPD fuer verfassungswidrig erklaert
4640 DATA 1956,F.J. Strauss wird Verteidigungsminister
4650 DATA 1956,Medizin-Nobelpreis an W. Frossmann
4660 DATA 1957,Saarland 10.Bundesland
```

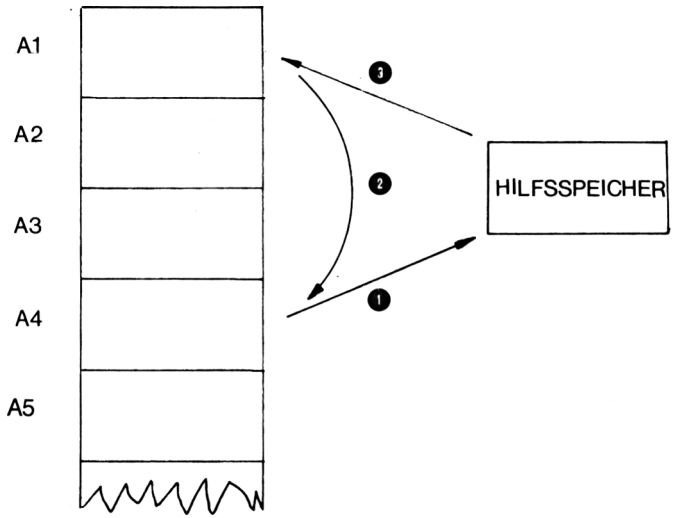
```
4670 DATA 1957,Willi Brandt Buergermeister von Berlin
4680 DATA 1957,DDR macht Vorschlag zur Wiedervereinigung
4690 DATA 1959,Heinrich Luebke wird Bundespraesident
4700 DATA 1959,Godesberger-Grundsatzprogramm der SPD
4710 DATA 1961,Sturmflut-Katastrophe in Hamburg
4720 DATA 1961,Errichtung der Berliner Mauer
4730 DATA 1961,Physik-Nobelpreis an Rudolf Moessbauer
4740 DATA 1962,'Spiegel-Affaere'
4750 DATA 1962,Gruendung der Universitaet Regensburg
4760 DATA 1963,J.F. Kennedy in West-Berlin
4770 DATA 1963,Ruecktritt Konrad Adenauer
4780 DATA 1963,Ludwig Erhard wird Bundeskanzler
4790 DATA 1963,Theodor Heuss stirbt
4800 DATA 1963,Eroeffnung der 'Vogelfluglinie'
4810 DATA 1963,Rolf Hochhuth 'Der Stellvertreter'
4820 DATA 1964,Willy Brandt wird Parteivorsitzender (SPD)
4830 DATA 1964,Gruendung Universitaet Konstanz
4840 DATA 1965,Aufnahme diplomatischer Beziehungen BRD-Israel
4850 DATA 1966,'Grosse Koalition'
4860 DATA 1966,Georg Kiesinger wird Bundeskanzler
4870 DATA 1967,Tod von Konrad Adenauer
4880 DATA 1967,Benno Ohnesorg von Polizei erschossen
4890 DATA 1967,'Ausserparlamentarische Opposition'
4900 DATA 1967,Chemie-Nobelpreis an Manfred Eigen
4910 DATA 1968,Attentat auf Rudi Dutschke (SDS)
4920 DATA 1968,Inkrafttreten der Notstandsartikel (BRD)
4930 DATA 1968,Neue National Galerie (M.v.d. Rohe) in Berlin
4940 DATA 1969,Gustav Heinemann wird Bundespraesident
4950 DATA 1969,1.Kongress der DKP
4960 DATA 1969,Schwere Studentenunruhen in Frankfurt
4970 DATA 1969,Willy Brandt wird Bundeskanzler
4980 DATA 1969,1. Koalition SPD-FDP
4990 DATA 1969,Unterzeichnung des Atomwaffensperrvertrags
5000 DATA 1970,Erstes Zusammentreffen Brandt-Stoph (Erfurt)
5010 DATA 1970,Gruendung der 1. integrierten Gesamthochschule
5020 DATA 1971,Rainer Barzel wird Parteivorsitzender (CDU)
5030 DATA 1971,Friedens-Nobelpreis an Willy Brandt
5040 DATA 1972,Tod von Heinrich Luebke
5050 DATA 1972,1. Staatsvertrag BRD-DDR unterzeichnet
5060 DATA 1972,Olympische Sommerspiele in Muenchen
5070 DATA 1972,Festnahme des Kerns der 'Baader-Meinhof-Bande'
5080 DATA 1972,Literatur-Nobelpreis an Heinrich Boell
5090 DATA 1973,Helmut Kohl neuer CDU-Vorsitzender
5100 DATA 1973,Aufnahme der BRD in die UNO
5110 DATA 1973,'Oelkrise'
5120 DATA 1973,'Dollar-Schwemme'
5130 DATA 1973,Medizin-Nobelpreis an Karl von Frisch
5140 DATA 1974,'Affaere Guillaume'
5150 DATA 1974,Willy Brandt tritt zurueck
```

```
5160 DATA 1974,Walter Scheel wird Bundespraesident
5170 DATA 1974,Helmut Schmidt wird Bundeskanzler
5180 DATA 1975,Entfuehrung von P.Lorenz (Berlin)
5190 DATA 1975,Ueberfall auf deutsche Botschaft in Schweden
5200 REM ************************************************
5210 '
65000 REM ************************************ M.W.Thoma **
65010 REM *                                               *
65020 REM *           Variabler RESTORE-Befehl            *
65030 REM *                                               *
65040 REM *     In 'ZNR' muss die Zeilennummer liegen     *
65050 REM *                                               *
65060 REM *   ACHTUNG ! Die Zeilen 60130 - 60150 MUESSEN  *
65070 REM *   genauso eingegeben werden. Es duerfen auch  *
65080 REM *   am  Ende der Zeilen keine  ueberfluessigen  *
65090 REM *              Leerzeichen stehen!              *
65100 REM *                                               *
65110 REM ************************************************
65120 znradr=PEEK(&AE37)*256+PEEK(&AE36)
65130 POKE znradr+109,&FF AND UNT(znr)
65140 POKE znradr+110,INT(znr/256)
65150 RESTORE 3240
65160 RETURN
65170 REM ************************************************
```

In dem Programm „Das Jahrhundertspiel" sind viele grundlegende Techniken der Datenverwaltung enthalten. Die Aufgabe des Programms ist es im wesentlichen, Daten innerhalb einer Liste zu verschieben (Tauschen). Ein Tauschen von Daten nach einem bestimmten Kriterium (z.B. alphabetisch) ist die Grundlage zum Sortieren von Daten.

7.5 Daten innerhalb einer Liste vertauschen

Das Tauschen zweier Variableninhalte (Daten) ist nicht direkt, sondern nur über einen Hilfsspeicher möglich.



In einer Datenliste, die hier in den Variablen A(1) bis A(n) liegt, sollen die Daten des Feldes A(1) und A(4) vertauscht werden. Der Tauschvorgang gliedert sich in drei Schritte:

1. Variable A(4) im Hilfsspeicher sichern
2. Inhalt der Variablen A(1) nach A(4) kopieren
3. Variable A(1) mit dem Hilfsspeicher laden

Nach diesem Vorgang sind beide Elemente miteinander vertauscht. Der komplette Tauschvorgang sieht in BASIC-Zeilen folgendermaßen aus:

```
H    = A(4)
A(4) = A(1)
A(1) = H
```

Den Tauchvorgan finden sie im „Jahrhundertspiel“ in den Zeilen 2150 bis 2170 wieder (Als Hilfsvariable dienen immer die Felder j$(0),d$(0) und r(0)).

7.6 Sortieren von Daten

Das Sortieren von Daten ist in der Datenverarbeitung ein altes Problem. Es gibt heute unzählige Sortiermethoden, die sich im Programmaufwand und in der Verarbeitungsgeschwindigkeit erheblich unterscheiden. Eine verbreitete Sortierroutine ist das sogenannte „Bubble-Sort" (sort=sortieren, bubble=Blase), da es leicht zu verstehen und programmtechnisch einfach aufgebaut ist. Die Grundlage dieser Sortiermethode ist, wie bei den meisten Sortierverfahren, das Tauschen von Daten.

An einem kleinen Beispiel möchte ich Ihnen das Sortierverfahren erläutern. In den Feldern A(1) bis A(3) liegen drei (unsortierte) Werte.

A(1)=9
A(2)=1
A(3)=5

Im ersten Durchgang des Sortierens wird das kleinste Element der Liste gesucht und in das 1. Feld (A(1)) gelegt:

- Vergleiche A(1) mit A(2).
- Ist A(2) kleiner als A(1), dann vertausche beide Inhalte.
- Anschließend vergleiche A(1) mit A(3) und tausche gegebenenfalls die Inhalte.

Nach dem ersten Durchgang sieht die Liste folgendermaßen aus:

A(1)=1 kleinstes Element !!
A(2)=9
A(3)=5

Der zweite Durchgang hat die Aufgabe, das zweitkleinste Element der Liste in das Feld A(2) zu legen. Da sich im Feld A(1) schon das kleinste Element der Liste befindet, ist das Prüfen nur noch ab A(2) notwendig.

- Vergleiche A(2) mit A(3)
- Ist A(3) kleiner als A(2), vertausche beide Inhalte

Nach zwei Durchgängen liegt die Liste mit 3 Daten in sortierter Form vor. Um eine Liste mit 100 Daten zu sortieren, sind 99 Durchgänge notwendig

(Datenanzahl-1). Das Unterprogramm „Bubble-Sort" sortiert eine beliebige Liste numerischer Felder in aufsteigender Folge. Durch Ändern des Array „wert(..)" in ein Stringarray (z.B. wert$(..)) lassen sich auch Strings alphabetisch ordnen.

```
10000 REM ************************************************
10010 REM *                                              *
10020 REM *                  BUBBLE-SORT                 *
10030 REM *                                              *
10040 REM *   anzahl=Anzahl der vorhandenen Daten        *
10050 REM *   wert(1) bis wert(anzahl) = Daten           *
10060 REM *                                              *
10070 REM ************************************************
10080 FOR i=1 TO anzahl-1
10090 FOR j=i+1 TO anzahl
10100 IF wert(i) <= wert(j) THEN 10120
10110 help=wert(i):wert(i)=wert(j):wert(j)=help
10120 NEXT j
10130 NEXT i
10140 RETURN
10150 REM ************************************************
```

Probieren Sie das Sortierprogramm einmal aus:

```
10 anzahl=100: DIM wert(anzahl)
20 PRINT "Unsortierte Liste"
30 FOR i=1 TO anzahl
40 wert(i)=INT(rnd(1)*100): PRINT wert(i),
50 NEXT i
60 PRINT "Ich sortiere ...."
70 GOSUB 10000
80 PRINT "Sortierte Liste"
90 FOR i=1 TO anzahl: PRINT wert(i),: NEXT i
99 END
```

Das Sortieren von 100 Daten dauert ca. 40 Sekunden. Natürlich gibt es heute wesentlich schnellere Sortierverfahren, deren Beschreibung aber den Rahmen dieses Buches sprengen würde. In der Praxis ist ein Sortieren großer Datenmengen mit dem Bubble-Sort nicht anwendbar, da das Sortieren von 1000 Zahlen mit dem Bubble-Sort-Programm gut 1 Stunde dauert. Mit einem kompilierten Bubble-Sort-Programm verkürzt sich die Sortierzeit in etwa um die Hälfte.

Ein wesentlich besseres Sortierprogramm ist der sogenannte „Shell-Metzner-Sort". Der Programmieraufwand ist ebenfalls sehr gering und die Sortiergeschwindigkeit liegt deutlich über der des Bubble-Sorts. Falls Sie größere Datenmengen sortieren müssen, nutzen Sie bitte den Shell-Metzner-Sort, damit Sie nicht „ewig" warten müssen.

```
10000 REM ************************************************
10010 REM *                                              *
10020 REM *              SHELL-METZNER-SORT              *
10030 REM *                                              *
10040 REM *   anzahl=Anzahl der vorhandenen Daten        *
10050 REM *   wert(1) bis wert(anzahl) = Daten           *
10060 REM *                                              *
10070 REM ************************************************
10080 me=anzahl
10090 me=INT(me/2):IF me=0 THEN RETURN
10100 j=1:k=anzahl-me
10110 i=j
10120 l=i+me
10130 IF wert(i) <= wert(l) THEN 10170
10140 help=wert(i):wert(i)=wert(l):wert(l)=help
10150 i=i-me
10160 IF i >0 THEN 10120
10170 j=j+1
10180 IF j>k THEN 10090:ELSE 10110
10190 REM ************************************************
```

Die Shell-Metzner-Methode arbeitet nach folgendem Verfahren: Das Programm teilt die Daten in zwei Hälften und vergleicht das 1.Element der ersten Hälfte mit dem ersten Element der zweiten Hälfte. Ist das Element der ersten Hälfte größer als das erste Element der zweiten Hälfte, werden beide Elemente getauscht. Danach das zweite Element mit dem zweiten Element etc., bis beide Hälften verglichen und gegebenenfalls vertauscht sind. Jetzt werden beide Hälften nach demselben Verfahren sortiert usw.

Das Shell-Metzner-Sort sortiert 100 Zahlen in ca. 10 Sekunden und 1000 Zahlen in gut 3.5 Minuten. Das ist 20 mal schneller als der Bubble-Sort (die Shell-Metzner-Methode ist auch schneller als „Quick-Sort").

7.7 Suchen in einer sortierten Liste

Das Suchen nach bestimmten Daten innerhalb einer Datenliste ist immer eine sehr zeitkritische Aktion. Innerhalb einer unsortierten Datenliste verfährt man im allgemeinen so, daß alle Daten mit dem Suchbegriff verglichen werden, bis eine Übereinstimmung gefunden ist. Bei einer Datenliste von 1000 Daten sind im extremen Fall 1000 Daten zu vergleichen, wenn sich der Suchbegriff zufällig an letzter Stelle der Liste befindet. Liegen Daten in einer sortierten Form vor, läßt sich innerhalb dieser Liste sehr komfortabel und vor allem schneller nach Daten suchen. Wie, das soll Ihnen folgendes kleine Spiel verdeutlichen:

Es soll eine Zahl zwischen 1 und 100 erraten werden. Als Hilfestellung erhält man die Aussage, ob die geratene Zahl zu groß oder zu klein ist. Um möglichst schnell die richtige Zahl zu „erraten", verfährt man nach dem Prinzip, immer die Mitte der zur Verfügung stehenden Zahlen zu wählen. Zuerst also die 50! Ist die gesuchte Zahl kleiner, so kann sie nur noch zwischen 1 und 49 liegen. Alle anderen Zahlen (50-100) sind damit schon jetzt ausgeschlossen. Jetzt wählt man die 25 (wiederum die Mitte der zur Verfügung stehenden Zahlen). Ist die gesuchte Zahl größer als 25, so liegt die gesuchte Zahl zwischen 26 und 49. Verfährt man nach diesem Prinzip weiter, findet man auf dem kürzesten Weg die richtige Zahl.

Die Anzahl der maximal notwendigen Versuche läßt sich mit der Gleichung

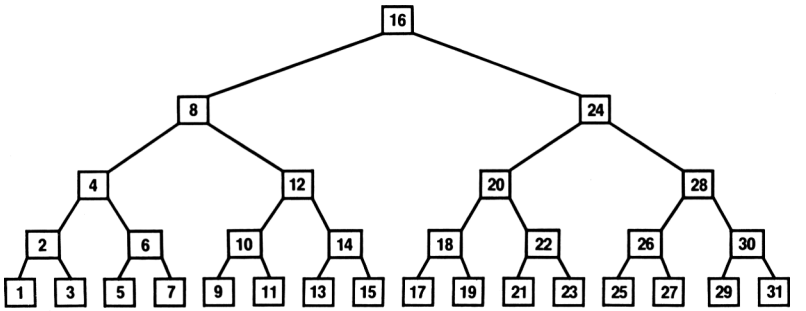
$$\text{versuche} = \text{INT}(\text{LOG}(\text{möglichkeiten})/\text{LOG}(2)+1)$$

errechnen. Das Suchen einer Zahl zwischen 1 und 1000 ist damit mit maximal

$$\text{INT}(\text{LOG}(1000)/\text{LOG}(2)+1) = 10$$

10 Versuchen möglich. Ein solches Suchverfahren ist natürlich nur dann möglich, wenn die Daten (Zahlen), aus der ein bestimmter Wert gesucht wird, in einer sortierten Form (geordnet) vorliegen.

Grafisch sieht diese Suchmethode aus wie ein Baum: Man spricht darum auch von dem „binären Baum".



Ein kurzes Beispiel zeigt, daß eine beliebige Zahl (Wert) zwischen 1 und 31 in maximal 5 Versuchen (INT(LOG(31)/LOG(2)+1)) zu erraten ist:

Gesuchte Zahl: 11

1. Versuch:	16	Antwort: kleiner
2. Versuch:	8	Antwort: größer
3. Versuch:	12	Antwort: kleiner
4. Versuch:	10	Antwort: größer
5. Versuch:	11	Antwort: gefunden

Dieses Suchverfahren läßt sich ohne weiteres für das Suchen z.B. in einem sortierten Array (siehe 7.6) nutzen. Gehen wir davon aus , daß sich im Array ,,wert$(..)'' eine geordnete Liste von 31 Namen befindet:

```
wert$(1) =''Andreas''
wert$(2) =''Boris''
wert$(3) =''Carl''
usw. ....
```

Durch die alphabetische Ordnung des Arrays ist sichergestellt, daß der Inhalt des Feldes ,,wert$(1)'' kleiner ist als der Inhalt von ,,wert$(2)'' (für den Computer sind Zeichen auch nur Zahlen, deren Wertigkeit von 65 (A) bis 90 (Z) steigend geordnet ist – ASCII-Code –). Wie beim Zahlenratespiel wird nun die Indexnummer des Arrays ermittelt (zuerst also ,,wert$(16)'') und der Inhalt des Feldes mit dem Suchbegriff verglichen. Ist der Suchbegriff größer als das geprüfte Feld, kann der gesuchte Name nur in den Feldern 17 bis 31 liegen. Die weitere Vorgehensweise ist identisch mit dem Zahlenratespiel.

Doch was passiert, wenn der Suchbegriff gar nicht in der Liste enthalten ist. Logisch, man kann ihn nicht finden. Doch woher weiß der Rechner, wann er

mit dem Suchen aufhören soll? Mit der weiter oben beschriebenen Gleichung ist die maximale Anzahl der Versuche berechnet worden, die notwendig sind, um jeden beliebigen Wert innerhalb der Liste zu finden. Ist ein Wert nicht mit den maximalen Versuchen zu finden, ist sichergestellt, daß sich der Wert nicht in der Liste befindet! An dieser Stelle kann das Suchen abgebrochen werden.

Das nachfolgende Unterprogramm ermöglicht das Suchen nach der oben beschriebenen Methode. Als Übergabeparameter sind die Anzahl der vorhandenen Daten und der Suchbegriff zu übergeben. Die Daten liegen im Array „wert$(1)" bis „wert$(anzahl)". Nach der Rückkehr befindet sich in der Variablen „vh" entweder eine Null (Suchbegriff nicht gefunden) oder die Indexnummer des gefundenen Feldes (Wollen Sie nach numerischen Werten suchen, ist das Unterprogramm zu ändern: Alle „wert$(..)" in „wert(..)" und alle „such$" in „such").

```
20000 REM ********************************** M.W. Thoma **
20010 REM *                                               *
20020 REM *                Binaerer Baum                  *
20030 REM *                                               *
20040 REM *   anzahl = Anzahl der vorhandenen Daten       *
20050 REM * wert$(..)= Datenarray numerisch/string       *
20060 REM *    such$ = Gesuchter Wert nummer/string       *
20070 REM *       vh =  0 : Datum nicht gefunden          *
20080 REM *       vh <> 0 : vh=indexnummer gefunden       *
20090 REM *                                               *
20100 REM *************************************************
20110 mst=INT(LOG(anzahl)/LOG(2)+1)
20120 z=mst-1:mst=2^mst:nr=mst/2
20130 IF z<0 THEN vh=0:RETURN
20140 z=z-1
20150 IF nr>anzahl THEN nr=nr-2^z:GOTO 20130
20160 IF such$=wert$(nr) THEN vh=nr:RETURN
20170 IF such$<wert$(nr) THEN nr=nr-2^z:GOTO 20130
20180 nr=nr+2^z
20190 GOTO 20130
20200 REM *************************************************
```

Zeile 20110 des Programms errechnet die maximal notwendigen Versuche („mst"), um einen Wert zu finden. Die Variable „z" ist ein Zähler, der zählt, wieviele Versuche bisher durchgeführt wurden (Wert nicht vorhanden wenn „z" kleiner als 0, Programmzeile 20130) und „nr" gibt die zu prüfende Indexnummer an.

Mit der Programmzeile 20150 hat es eine besondere Bedeutung, auf die an dieser Stelle etwas genauer eingegangen werden muß.

Der Name „binärer Baum" kommt nicht von ungefähr. Für die Funktion der Suchmethode ist es immer notwendig, daß die Anzahl der vorhandenen Daten („wert$(..)") eine Zahl ist, die binär nur aus „1" besteht:

```
1111    =  15
11111   =  31
111111  =  63
1111111 = 127
```

Probleme treten auf, wenn die Anzahl der Daten eine hiervon abweichende Zahl ist. Zur Lösung des Problems wird dann zunächst die nächsthöhere Zweier-Potenz gewählt. Sind es beispielsweise 40 Daten, so ist die Datenanzahl auf 63 zu erhöhen. Nun sind es aber tatsächlich nur 40 und nicht 63 Daten. Versucht das Programm z.B. auf das Feld „wert$(48)" zuzugreifen, so erfolgt die Fehlermeldung „Subscript out of range", da es außerhalb des dimensionierten Bereichs liegt (zudem befinden sich hier keine Daten mehr). Weiterhin ist damit aber auch geklärt, daß der Suchbegriff auf jeden Fall „tiefer" in der Datenlisten liegen muß! Die Programmzeile 20150 stellt sicher, daß niemals auf Daten zugegriffen wird, die nicht vorhanden sind.

Dazu ein konkretes Zahlenbeispiel: Es sind 40 Daten in der Liste vorhanden. Die Anzahl wird auf 63 erhöht.

Gesucht wird das Feld 40:

1. Versuch: 32	Ergebnis: größer (Zeile 20180)
2. Versuch: 48	Ergebnis: kleiner, da das Feld außerhalb der Liste liegt (Zeile 20150)!
3. Versuch: 40	Ergebnis: gefunden (Zeile 20160)

Durch die Programmzeile 20150 läßt sich innerhalb einer beliebig großen, sortierten Datenliste nach der binären Suchmethode nach Daten suchen.

Testen Sie das Unterprogramm mit diesem kleinen Beispiel, um die Arbeitsweise der binären Suchmethode besser zu verstehen. Zuvor fügen Sie bitte noch die Zeile

```
20145 PRINT z,nr
```

in das Unterprogramm ein. Sie können dann gut verfolgen, wie das Programm arbeitet.

```
100 DIM wert$(26)
110 anzahl=26
120 FOR i=1 TO anzahl
130 wert$(i)=CHR$(64+i)
140 PRINT USING "⟨##⟩";i,: PRINT wert$(i)
150 NEXT i
160 PRINT
170 INPUT "Suchbuchstabe :";such$
180 GOSUB 20000
190 IF vh=0 THEN PRINT "Nicht vorhanden!": GOTO 170
200 PRINT "Gefunden in";vh,wert$(vh)
210 GOTO 170
```

7.8 Suchen mit INSTR

In vorangegangenen Kapitel 7.7 wurde das Suchen innerhalb einer (sortierten) Liste beschrieben. Dabei wurde ein Wert nur dann gefunden, wenn der Suchbegriff exakt mit einem Wert der Datenliste übereinstimmte. Besteht die Datenliste beispielsweise aus einer Namensliste (Name/Vorname), so muß der Suchbegriff stets eindeutig sein. Ist der Eintrag „Müller/Heinz" zu suchen, ist als Suchbegriff auch „Müller/Heinz" anzugeben. Wird nur „Müller" angegeben, wird ein entsprechender Eintrag mit Sicherheit nicht gefunden. Die Ursache für dieses Verhalten ist in der Abfrage

IF such$=wert$(...) THEN .....

zu finden, denn die Bedingung ist nur dann erfüllt (wahr), wenn beide Ausdrücke exakt identisch sind. Mit Hilfe der bekannten String-Funktionen LEFT$, RIGHT$ und MID$ können bestimmte Teile der zu prüfenden Daten isoliert und damit auf eine Teilinformation hin geprüft werden. Besteht der Suchbegriff z.B. aus

such$="Müller"

so ist die Datenliste nur auf den Ausdruck „Müller", der am Anfang des zu prüfenden Strings stehen soll, zu testen. Das ist mit der Funktion LEFT$ noch recht einfach zu bewerkstelligen:

```
IF such$=LEFT$(wert$(...),LEN(such$)) THEN .....
```

Von dem zu prüfenden String „wert$(...)" wird ein Teil getestet, der genau so lang ist, wie der Suchbegriff selber. Damit ist die Bedingung erfüllt, wenn der Anfang des zu testenden Strings mit dem Suchbegriff übereinstimmt. Gefunden werden also alle Einträge, die mit „Müller" beginnen.

Wesentlich komplizierter wird die Angelegenheit, wenn zu prüfen ist, ob sich an einer beliebigen Stelle des Strings der Ausdruck „Müller" befindet. Normalerweise müßte der gesamte String Stück für Stück auf seinen Informationsgehalt getestet werden. Doch der SCHNEIDER CPC-464/664 stellt eine sehr komfortable „String-Durchsuchfunktion" zur Verfügung: Die INSTR-Funktion.

```
INSTR (⟨position⟩, string$, such$)
```

Die INSTR-Funktion durchsucht einen String „string$" nach dem Ausdruck „such$". Befindet sich der gesuchte (Teil-) String „such$" innerhalb des zu prüfenden Strings „string$", gibt die Funktion eine Zahl größer als 0 wieder. Diese Zahl bezeichnet die Stelle, wo innerhalb des Strings der gesuchte Ausdruck beginnt. Liefert die INSTR-Funktion den Wert 0, befindet sich der Suchbegriff nicht im String. Als Option kann der INSTR-Funktion noch mitgeteilt werden, ab welcher Position („position") das Prüfen des Strings beginnen soll. Damit ist es u.a. möglich, den String erst ab dem 10. Zeichen nach einem Suchbegriff zu durchsuchen. Mit der Abfrage

```
IF INSTR(wert$(...),such$) > 0 THEN ..... gefunden
```

kann festgestellt werden, ob sich der Suchbegriff an einer beliebigen Position innerhalb eines Strings befindet. Die INSTR-Funktion ist zum Durchsuchen eines Strings eine sehr große Hilfe.

Besondere Stärken zeigt die INSTR-Funktion dann, wenn innerhalb eines Strings mehr als eine Information untergebracht ist. Besteht ein String z.B. aus den (Teil-) Informationen Name, Vorname, Ort und PLZ, die sich an bestimmten Stellen des Strings befinden, läßt sich der String auf mehrere Informationen hin prüfen.

```
          1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
wert$(..)="Na me          Vor n a me      Or t          PLZ"
```

Wollen Sie nun alle „Müller" suchen, die in der Postleitzahl 2000 wohnen, so wäre das mit den Anweisungen:

```
IF LEFT$(wert$(..),6)="Müller" AND INSTR (31,wert$(..),"2000") THEN .... gefunden!
```

möglich. Wesentlich universeller (dafür aber etwas ungenauer) ist eine einfache Verknüpfung von zwei Suchbegriffen, die sich an einer beliebigen Stelle der Strings befinden dürfen:

```
such1$="Müller"
such2$="2000"
IF SGN(INSTR(wert$(..),such1$)) AND
   SGN(INSTR(wert$(..),such2$)) = 1 THEN ..... gefunden!
```

Die SGN-Funktion gibt das Vorzeichen eines Wertes in Form einer Zahl wieder. Ist der Wert negativ, ist das Ergebnis – 1, ist der Wert 0, ist das Ergebnis ebenfalls 0 und bei einem positiven Wert liefert die SGN-Funktion das Ergebnis (+) 1. Haben die beiden INSTR-Funktionen den Suchbegriff gefunden, liefern sie ein positives Ergebnis (größer 0). Die SGN-Funktion liefert also eine 1 für gefunden und eine 0 für nicht gefunden. Verknüpft man diese beiden Werte mit der AND-(UND)-Funktion,

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1 (wahr)
```

so ist das Ergebnis nur dann „1" (wahr), wenn beide INSTR-Funktionen einen Wert größer als 0 liefern und somit die Suchbegriffe gefunden haben.
Suchen Sie allerdings nach diesem Prinzip z.B. den Namen „Peter", so liefert die SGN- bzw. INSTR-Funktion auch dann den Wert 1 (wahr), wenn der Nachnahme „Petersen" ist!

Ich möchte an dieser Stelle nicht weiter auf die Anwendungen der INSTR-Funktion eingehen. Die hier erläuterten Beispiele sollen Ihnen einige Anregungen geben, um Ihre persönlichen Dateiprogramme noch besser und kom-

fortabler aufzubauen. Gerade das verknüpfte Suchen innerhalb einer Datei zeichnet heute gute Datei- und Datenbankprogramme aus. Mit Hilfe der INSTR-Funktion ist ein verknüpftes Suchen leicht und einfach, wenn auch mit gewissen Einschränkungen, zu realisieren.

8.0 Die sequentielle Datei

Die sequentielle Verwaltung ist die einzige im CPC-464/664 vorgesehene externe Datenverwaltung auf Diskette/Kassette. Die Handhabung und Programmierung einer sequentiellen Datei wurde schon im Kapitel 2.7 beschrieben. Alle Daten, die einer Datei zugeordnet sind, liegen innerhalb der Datei auf der Diskette/Kassette hintereinander. D.h. nach dem ersten Datum folgt das zweite usw. Der Nachteil einer sequentiellen Datei ist, wie schon bei der Verwaltung von DATA-Zeilen deutlich geworden ist, daß der Zugriff auf ein bestimmtes Datum nur durch „überlesen" der nicht benötigten Daten erfolgen kann (siehe 7.2). Weiterhin kann ein bestimmtes Datum nicht direkt innerhalb der Datei auf der Diskette/Kassette geändert werden, sondern zur Bearbeitung ist die Datei komplett in den Rechner zu laden. Nach der Bearbeitung wird die geänderte Datei wieder komplett zurückgeschrieben. Aus diesem Grund darf die zu bearbeitende Datenmenge nicht größer sein als der zur Verfügung stehende Speicherraum im Rechner. Diese Einschränkungen werden allerdings durch die überaus einfache und komfortable Handhabung wieder gutgemacht.

Der Datentransport zwischen Rechner und externem Speicher findet bei der sequentiellen Datenverwaltung immer nach folgendem Schema statt:

1. Datenweg öffnen (OPENIN oder OPENOUT)
2. Daten übertragen (Speicher-Rechner, Rechner-Speicher)
3. Datenweg schließen (CLOSEIN oder CLOSEOUT)

Die Daten befinden sich vor oder nach der Datenübertragung meistens in einem Variablenfeld (Array). Dadurch ist eine bequeme Bearbeitung möglich, da jedes Datum einer bestimmten Variablen zugeordnet ist (z.B. das 99. Datum in A(99)).

Eine sequentielle Datei besteht aus mehreren Daten bzw. Datensätzen. Datensätze sind in Felder aufgeteilt, wobei jedes dieser Datenfelder wiederum Daten enthalten kann. Wenn eine Datei nur die Werte 1 bis 99 enthält, so sind 99 Datensätze mit je einem Datenfeld vorhanden. Ein Datensatz kann aus mehreren Datenfeldern bestehen. Bei einer Telefondatei besteht ein Datensatz z.B.

aus 3 Datenfeldern, wobei das erste Feld den Namen, das zweite die Adresse und das dritte Datenfeld die Telefonnummer enthält. Bei einem Telefonverzeichnis von 100 Teilnehmern sind 100 Datensätze mit je 3 Datenfeldern notwendig.

Datensatz (1):	Feld 1	Feld 2	Feld 3
.....	.....	.....	.....
Datensatz (n):	Feld 1	Feld 2	Feld 3

Alle Datenfelder eines Datensatzes bilden eine zusammengehörige Einheit. Alle Felder haben einen bestimmten Bezug zueinander. Als Beispiel soll eine vorhandene Telfondatei mit 100 Datensätzen a 3 Datenfelder in den Rechner eingeladen werden:

```
100 DIM name$(100),adresse$(100),telenr$(100)
110 OPENIN "TELEFON.DAT"
120 FOR i=1 TO 100
130 INPUT#9,name$(i),adresse$(i),telenr$(i)
140 NEXT i
150 CLOSEIN
```

Jedes Datenfeld ist einem bestimmten Variablenfeld zugeordnet. Die Datensätze bestehen aus:

```
Datensatz 1   = name$(1),   adresse$(1),   telenr$(1)
Datensatz 2   = name$(2),   adresse$(2),   telenr$(2)
.........     = name$(..),  adresse$(..),  telenr$(..)
Datensatz 100 = name$(100), adresse$(100), telenr$(100)
```

Eine andere Zuordnung der Datenfelder, die sich vor allem bei einer großen Anzahl von Datenfeldern anbietet, ist die Zuweisung der Daten in einem 2-dimensionalen Variablenfeld.

```
100 DIM feld$ (100,2)
110 FOR satznr=1 TO 100
120 FOR feldnr=0 TO 2
130 INPUT#9,feld$(satznr,feldnr)
140 NEXT feldnr
150 NEXT satznr
```

Die erste Dimension (satznr) bestimmt die Datensatznummer und die 2.Dimension (feldnr) die entsprechende Datenfeldnummer.

```
Datensatz 1 = feld$(1,0), feld$(1,1), feld$(1,2)
Datensatz 9 = feld$(9,0), feld$(9,1), feld$(9,2)
```

Die Behandlung einer sequentiellen Datei ist im allgemeinen unproblematisch, wenn folgende Punkte beachtet werden:

1. Alle Datenfelder müssen in derselben Reihenfolge wieder in den Rechner eingelesen werden, wie sie gesichert worden sind. Andernfalls stehen in den Datenfeldern falsche Informationen.
2. Ist ein Datenfeld als numerisches Feld gesichert, muß das Einlesen wiederum in eine numerische Variable erfolgen.
3. Zur Trennung zweier Daten innerhalb der sequentiellen Datei stehen dem CPC-464 das Carriage Return (CHR$(13)) und das Komma („,“) zur Verfügung.

```
          INPUT#9,a,b
gleich    INPUT#9,a:INPUT#9,b
```

Befindet sich innerhalb eines Strings ein Komma als Satzzeichen, interpretiert der CPC-464 dieses Komma beim Einlesen als Trennzeichen zwischen zwei Daten:

```
10 PRINT#9, "Ja, ich habe durst"
20 INPUT#9, a$
30 PRINT a$

Ja (!!!!!)
```

Falls sich in einem String ein Komma als Satzzeichen befindet, muß dieses Datum mit dem LINE INPUT-Befehl eingelesen werden. Der LINE INPUT-Befehl interpretiert nur das Carrige Return als Trennzeichen. Aber auch die Anwendung des LINE INPUT-Befehls ist nicht unkritisch:

```
PRINT#9,"Heute","Morgen","Immer"
LINE INPUT#9, a$
a$ ist  "HeuteMorgenImmer"  !!!!!!
```

Obwohl drei Daten, getrennt durch das Komma, gesichert sind, liest der LINE INPUT-Befehl diese Daten als ein Datum ein!! Beachten Sie, daß Sie die Daten immer so einlesen, wie sie gesichert sind, damit es zu keinen ungewollten Verkettungen von Daten kommt.

8.1 Ein Haushaltsbuch

Das nachfolgende Programm „Haushaltsbuch" zeigt eine typische Anwendung einer sequentiellen Datensicherung. Jeder Datensatz besteht aus 11 Datenfeldern:

da$(x)	Datum des Eintrags
wert(1,x)	Ausgabe Nummer 1
.....	
wert(9,x)	Ausgabe Nummer 9
su(x)	Summe aller Ausgaben

Das Einlesen und Speichern der Datei finden Sie in den Programmzeilen 2080 bis 2570. Hier werden die Datensätze nacheinander in die Datei geschrieben oder aus ihr geholt.

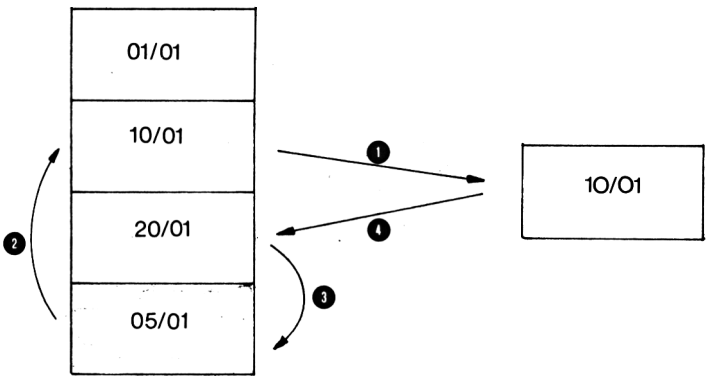
Das Haushaltsbuch erinnert in seinem Aufbau stark an Tabellen-Kalkulations-Programme. In der Waagerechten (Zeile) finden Sie die Ausgaben für einen Tag, aufgeteilt in 9 Warengruppen (Läden), sowie die Summe aller Ausgaben. In der Senkrechten (Spalte) stehen alle Warengruppen untereinander.

```
Gesamt - Abrechnung

   Bitte warten !

Summe :  621.00 DM
```

```
Aldi    54.00   Backw   55.00   Budni   55.00
COMET   55.00   EDEKA   55.00   Fisch  154.00
Trink   55.00   Markt   64.00   Sonst   74.00
                Summe :   621.00

          Weiter bitte eine Taste
```

Datum	% Aldi	8.86 Backw	8.86 Budni	8.86 COMET	8.86 EDEKA	24.80 Fisch	8.86 Trink	10.31 Markt	11.92 Sonst	100.00 Summe
12/01 :	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	45.00
15/01 :	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
04/04 :	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00
01/05 :	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	7.00	63.00
01/06 :	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	3.00	27.00
02/06 :	1.00	1.00	1.00	1.00	1.00	1.00	1.00	11.00	1.00	19.00
06/06 :	9.00	9.00	9.00	9.00	9.00	99.00	9.00	9.00	9.00	171.00
10/10 :	0.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	32.00
12/12 :	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	6.00	54.00
31/12 :	2.00	2.00	2.00	2.00	2.00	11.00	2.00	1.00	21.00	45.00
../.. :	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Insgesamt stehen 14 Befehle für ein komfortables Bearbeiten der Daten zur Verfügung. Bei der Eingabe (E) einer neuen Tagesausgabe fragt das Programm nach dem Datum, das in Form von 5 Zeichen eingegeben werden kann. Dabei bietet sich die Angabe von „01/12" an. Ist bereits ein Eintrag unter diesem Tag vorhanden, ist keine Eingabe möglich. Der alte Eintrag ist dann mit den weiteren Ausgaben entsprechend zu ändern (A). Ist kein Eintrag vorhanden, kann ein neuer Eintrag mit 9 Warengruppen eingegeben werden. Nach der kompletten Eingabe erfolgt das Einsortieren des Eintrags (nach Datum) an die richtige Stelle der Tabelle. Da sich die Daten vor einer Neueingabe immer in einer sortierten Reihenfolge befinden, ist kein eigentliches Sortieren mit einem Sortierprogramm nötig. Gesucht werden muß nur der Ort (vom Beginn der Datenliste), wo ein Datum gefunden wird, daß größer ist als das neue Datum. Alle Daten unterhalb dieses Datums verschieben sich dann jeweils eine Position nach „unten". Die folgende Abbildung soll das Vefahren verdeutlichen:



In einer (sortierten) Liste mit drei Daten wird das neue Datum (05/01) zunächst an das Ende der Liste gelegt (dient als Zwischenspeicher). Das Datum 05/01 muß die zweite Position innerhalb der Liste einnehmen. Das Datum, daß sich an dieser Position befindet (10/01), wird zunächst in einen Hilfsspeicher gerettet (1). Nun nimmt das neue Datum diesen Platz ein (2). Alle Daten unterhalb dieser Position wandern einen Listenplatz nach „unten" (3) und schaffen somit Platz für den Eintrag, der sich im Hilfsspeicher befindet, da dieses Datum nach dem Neueintrag folgen muß (4).

Dieses etwas kompliziert erscheinende Verfahren ist im Prinzip nur ein Tauschen von Daten innerhalb einer Liste. Der Geschwindigkeitsvorteil gegenüber eines normalen Sortierprogramms (Bubble-Sort etc.) ist beträchtlich.

Das Einsortieren der Daten finden Sie in den Programmzeilen 3320-3500. In diesem Fall müssen natürlich alle Datenfelder mitvertauscht werden, damit sich alle Daten an der richtigen Position befinden.

Das Programm „Haushaltsbuch" arbeitet sowohl mit einer Diskettenstation als auch mit dem Datenrecorder (Die Bedienungsanweisungen sind für den Datenrecorder ausgelegt). Die Befehle im Programm haben im einzelnen folgende Funktionen:

(H)ilfe	Zeigt eine Liste aller möglichen Befehle (Hilfe)
(L)aden	Lädt eine vorhandene Datei von Kassette/Diskette
(S)ichern	Sichert die aktuelle Datei auf Kassette/Diskette (nur möglich, wenn sich Daten im Rechner befinden)
(Q)uit	Programmende (gibt eine Meldung aus, falls sich Daten im Rechner befinden).
(E)ingabe	Neue Daten eingeben. Ist schon ein Eintrag unter diesem Datum vorhanden, ist eine Eingabe nicht möglich (Ändern).
(A)endern	Einen Eintrag ändern. Ist der gewünschte Eintrag nicht vorhanden, erfolgt eine Meldung, andernfalls kann der gesamte Eintrag gelöscht oder die Daten nacheinander geändert werden. Mit der ENTER-Taste wird der alte Wert beibehalten.
(F)inden	Sucht ein bestimmtes Datum der Liste und setzt es (falls vorhanden) in die oberste Zeile der Tabelle.
(P)age	Zeigt die nächste volle Seite der Tabelle (falls soviele Daten vorhanden sind).
up	Verschiebt die Tabelle um eine Position nach oben.
down	Verschiebt die Tabelle um eine Position nach unten.
(R)ekons	Bringt die Tabelle in Endstellung (Ende der Liste).
(M)onat	Erstellt eine Monatsabrechnung eines bestimmten Monats. Gibt dabei die Summen und prozentualen Verhältnisse aller Warengruppen aus.
(G)esamt	Gesamtabrechnung. Wie Monatsabrechnung, allerdings für die gesamte Datei.

Das Programm ist sehr leicht an Ihre persönlichen Wünschen anzupassen und es ist für die grafische Ausgabe der Daten vorbereitet. Sie können dafür eines

der 4 Grafikprogramme aus dem Band 1 (Kapitel 7.0 bis 7.5) benutzen. Auch auf die Ausgabe der Daten auf dem Drucker ist verzichtet worden. Da sich aber alle Daten in den Variablen „su", „da$" und „wert" befinden, ist die Ausgabe vollkommen unproblematisch.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *            H A U S H A L T S B U C H          *
1030 REM *                                               *
1040 REM ***************************************************
1050 SPEED WRITE 1:MODE 2
1060 OPENOUT "dummy": MEMORY HIMEM-1:CLOSEOUT
1070 DIM laden$(10),wert(9,200),da$(200),su(200),eg(9)
1080 INK 0,0:INK 1,26:BORDER 0
1090 laden$(0)="Datum  "
1100 laden$(1)="Aldi "
1110 laden$(2)="Backw"
1120 laden$(3)="Budni"
1130 laden$(4)="COMET"
1140 laden$(5)="EDEKA"
1150 laden$(6)="Fisch"
1160 laden$(7)="Trink"
1170 laden$(8)="Markt"
1180 laden$(9)="Sonst"
1190 laden$(10)="Summe"
1200 11$="../.."
1210 '
1220 REM ***************************************************
1230 REM *                                               *
1240 REM *                Bildschirmaufbau               *
1250 REM *                                               *
1260 REM ***************************************************
1270 WINDOW#0,1,80,13,24
1280 WINDOW#1,2,23,2,8
1290 WINDOW#2,30,79,2,8
1300 WINDOW#3,1,80,11,11
1310 WINDOW#4,1,80,10,10
1320 MOVE 7,384:DRAW 7,271,1:DRAW 184,271,1
1330 DRAW 184,384,1:DRAW 7,384,1
1340 MOVE 0,24:DRAW 639,24,1
1350 MOVE 0,216:DRAW 639,216,1
1360 MOVE 231,384:DRAW 632,384,1:DRAW 632,271,1
1370 DRAW 231,271,1:DRAW 231,384,1
1380 FOR i=0 TO 10
1390 PRINT#3,laden$(i);"  ";
1400 NEXT i
1410 GOSUB 4830
```

```
1420 '
1430 REM ************************************************
1440 REM *                                              *
1450 REM *                M  E  N  U  E                 *
1460 REM *                                              *
1470 REM ************************************************
1480 CLS#1
1490 PRINT#1," Bitte Kommando ?"
1500 LOCATE#1,20,1: PRINT#1,CHR$(232)
1510 a$=INKEY$
1520 IF a$=""THEN LOCATE#1,20,1:PRINT#1,CHR$(233):GOTO 1500
1530 a$=UPPER$(a$)
1540 IF a$="H" THEN GOSUB 1710: GOTO 1480
1550 IF a$="M" THEN GOSUB 3780: GOTO 1480
1560 IF a$="S" THEN GOSUB 2320: GOTO 1480
1570 IF a$="L" THEN GOSUB 2080: GOTO 1480
1580 IF a$="F" THEN GOSUB 2590: GOTO 1480
1590 IF a$="Q" THEN GOSUB 2810: GOTO 1480
1600 IF a$="E" THEN GOSUB 3080: GOTO 1480
1610 IF a$="R" THEN GOSUB 4830: GOTO 1480
1620 IF a$="P" THEN GOSUB 3520: GOTO 1480
1630 IF a$="G" THEN GOSUB 4220
1640 IF c=1 THEN c=0:GOTO 1220
1650 IF a$="A" THEN GOSUB 4360: GOTO 1480
1660 IF a$=CHR$(240) THEN GOSUB 3630: GOTO 1480
1670 IF a$=CHR$(241) THEN GOSUB 3730: GOTO 1480
1680 SOUND 1,600,5
1690 GOTO 1480
1700 '
1710 REM ************************************************
1720 REM *                                              *
1730 REM * <H>        Ausgabe eines HELP-Textes         *
1740 REM *                                              *
1750 REM ************************************************
1760 CLS#2
1770 PRINT#2," <H>  :  H E L P - Erlaeuterung der Befehle!"
1780 PRINT#2," <L>  :  L O A D - ";
1790 PRINT#2,"Laden einer Datei von Kassette!";
1800 PRINT#2," <S>  :  S A V E - Speichern der Datei!"
1810 PRINT#2," <F>  :  F I N D - ";
1820 PRINT#2,"Suche einen Eintrag (Datum)!"
1830 PRINT#2," <Q>  :  Q U I T - Programmende!"
1840 PRINT#2
1850 PRINT#2,"                  Bitte eine Taste"
1860 a$=INKEY$:IF a$="" THEN 1860
1870 CLS#2
1880 PRINT#2," <E>  :  EINGABE - neuer Daten!"
1890 PRINT#2," <A>  :  AENDERN - oder loeschen von Daten!"
1900 PRINT#2," <M>  :  MONATS  - Abrechnung!"
```

```
1910 PRINT#2," <G>  :  GESAMT  - Abrechnung der Datei!"
1920 PRINT#2," <R>  :  REKONS  - ";
1930 PRINT#2,"Bringt Tabelle in Ausgangspos.!"
1940 PRINT#2,"                  Bitte eine Taste"
1950 a$=INKEY$:IF a$="" THEN 1950
1960 CLS#2
1970 PRINT#2," <";CHR$(240);">  :  UP     - ";
1980 PRINT#2,"Tabelle nach oben verschieben!"
1990 PRINT#2," <";CHR$(241);">  :  DOWN   - ";
2000 PRINT#2,"Tabelle nach unten verschieben!"
2010 PRINT#2," <P>  :  PAGE   - Zeigt naechste volle Seite!"
2020 :PRINT#2:PRINT#2
2030 PRINT#2,"                  Bitte eine Taste"
2040 a$=INKEY$:IF a$="" THEN 2040
2050 CLS#2
2060 RETURN
2070 '
2080 REM ************************************************
2090 REM *                                              *
2100 REM * <L>   Vorhandene Datei von Kass/Disk laden   *
2110 REM *                                              *
2120 REM ************************************************
2130 CLS#1
2140 PRINT#1,"  Laden einer Datei"
2150 PRINT#1
2160 INPUT#1," Name:";name$
2170 IF LEN(name$)=0 OR LEN(n$)>8 THEN RETURN
2180 PRINT#1:PRINT#1," Kassette vorbereiten"
2190 PRINT#1,"     press Play"
2200 PRINT#1," und Taste betaetigen"
2210 a$=INKEY$:IF a$="" THEN 2210
2220 OPENIN "!"+name$+".dat"
2230 INPUT#9,tg
2240 FOR i = 1 TO tg: INPUT#9,da$(i)
2250 FOR j = 1 TO 9: INPUT#9,wert(j,i)
2260 NEXT j
2270 INPUT#9,su(i):NEXT i
2280 CLOSEIN
2290 GOSUB 4830
2300 RETURN
2310 '
2320 REM ************************************************
2330 REM *                                              *
2340 REM * <S>      Sichern der Daten auf Kass/Disk     *
2350 REM *                                              *
2360 REM ************************************************
2370 CLS#1
2380 PRINT#1,"  Sichern der Daten"
2390 PRINT#1
```

```
2400 IF tg=0 THEN SOUND 1,800,200 : GOTO 1430
2410 INPUT#1," Name :";name$
2420 IF LEN(name$)=0 OR LEN(n$)>8 THEN 1430
2430 PRINT#1:PRINT#1," Kassette vorbereiten"
2440 PRINT#1,"  press Play /Record"
2450 PRINT#1," und Taste betaetigen"
2460 a$=INKEY$:IF a$="" THEN 2460
2470 OPENOUT "!"+name$+".dat"
2480 PRINT#9,tg
2490 FOR i=1 TO tg
2500 PRINT#9,da$(i)
2510 FOR j=1 TO 9
2520 PRINT#9,wert(j,i)
2530 NEXT j
2540 PRINT#9,su(i)
2550 NEXT i
2560 CLOSEOUT
2570 RETURN
2580 '
2590 REM ************************************************
2600 REM *                                              *
2610 REM * <F>            Suchen nach Datum             *
2620 REM *                                              *
2630 REM ************************************************
2640 CLS#1
2650 PRINT#1," Suchen nach Datum"
2660 PRINT#1
2670 INPUT#1," Datum :";d$
2680 IF LEN(d$)=0 THEN 1430
2690 FOR f=1 TO tg
2700 IF d$=da$(f) THEN 2770
2710 NEXT f
2720 PRINT#1
2730 PRINT#1," N I C H T vorhanden!"
2740 SOUND 1,200,20
2750 FOR i=1 TO 2000: NEXT i
2760 RETURN
2770 tt=f:t1=f+10:IF t1>tg THEN t1=tg
2780 GOSUB 4900
2790 RETURN
2800 '
2810 REM ************************************************
2820 REM *                                              *
2830 REM * <0>              Programmende                *
2840 REM *                                              *
2850 REM ************************************************
2860 CLS#1
2870 PRINT#1,"     Programmende"
2880 PRINT#1
```

```
2890 IF tg=0 THEN  3000
2900 PRINT#1,"      A C H T U N G"
2910 PRINT#1," Daten im Rechner !!!!"
2920 FOR i= 1 TO 10:SOUND 1,100/i,5
2930 FOR j=1 TO 50:NEXT j
2940 NEXT i
2950 PRINT#1,"    Ende  <j/n> ?"
2960 a$=INKEY$:IF a$="" THEN 2960
2970 IF UPPER$(a$)="N" THEN RETURN
2980 IF UPPER$(a$)="J" THEN 3000
2990 GOTO 2960
3000 CLS#1
3010 PRINT#1," Manfred Walter Thoma"
3020 PRINT#1,"      Hamburg       "
3030 PRINT#1
3040 PRINT#1,"     Good-bye !!"
3050 WINDOW SWAP 0,1
3060 END
3070 '
3080 REM ************************************************
3090 REM *                                              *
3100 REM * <E>          Eingabe neuer Daten             *
3110 REM *                                              *
3120 REM ************************************************
3130 CLS#2
3140 PRINT#2,"            Eingabe neuer Daten !!"
3150 tg=tg+1
3160 LOCATE#2,3,3:PRINT#2,laden$(0);:INPUT#2, da$(tg)
3170 IF da$(tg)="" THEN tg=tg-1:CLS#2:RETURN
3180 IF LEN(da$(tg)) <> 5 THEN SOUND 1,200,10: GOTO 3160
3190 FOR i= 1 TO tg-1
3200 IF da$(i)=da$(tg) THEN 3230
3210 NEXT i
3220 GOTO 3280
3230 PRINT#2,"  Eintrag unter diesem Tag ";
3240 PRINT#2,"ist schon vorhanden ! "
3250 SOUND 1,800,200
3260 FOR i= 1 TO 2000:NEXT i
3270 tg=tg-1:CLS#2:RETURN
3280 su(tg)=0:FOR i = 1 TO 9:PRINT#2,"  ";laden$(i);
3290 INPUT#2,wert(i,tg):su(tg)=su(tg)+wert(i,tg)
3300 NEXT i
3310 PRINT#2,"      Sortiere Eintrag neu ein !!"
3320 FOR i= 1 TO tg-1
3330 IF MID$(da$(i),4,2)=MID$(da$(tg),4,2) THEN  3380
3340 IF MID$(da$(i),4,2)>MID$(da$(tg),4,2) THEN  3390
3350 NEXT i
3360 CLS#2:GOSUB 4830
3370 RETURN
```

```
3380 IF LEFT$(da$(i),2)<LEFT$(da$(tg),2) THEN 3350
3390 FOR j=1 TO 9:wert(j,0)=wert(j,i):NEXT j
3400 su(0)=su(i):da$(0)=da$(i)
3410 FOR j=1 TO 9:wert(j,i)=wert(j,tg):NEXT j
3420 su(i)=su(tg):da$(i)=da$(tg)
3430 FOR j=tg-1 TO i+1 STEP -1
3440 FOR z=1 TO 9:wert(z,j+1)=wert(z,j):NEXT z
3450 su(j+1)=su(j):da$(j+1)=da$(j)
3460 NEXT j
3470 FOR j=1 TO 9:wert(j,i+1)=wert(j,0):NEXT j
3480 su(i+1)=su(0):da$(i+1)=da$(0)
3490 IF i >= tt THEN GOSUB 4830
3500 CLS#2:RETURN
3510 '
3520 REM ************************************************
3530 REM *                                              *
3540 REM * <P>      Naechste volle Seite anzeigen       *
3550 REM *                                              *
3560 REM ************************************************
3570 IF tt=1 THEN SOUND 1,200,20:GOTO 3610
3580 IF tg<11 THEN SOUND 1,300,50:GOSUB 4830:GOTO 3610
3590 IF tt-10 < 1 THEN tt=1:t1=11:GOSUB 4900:GOTO 3610
3600 t1=tt:tt=tt-10:GOSUB 4900
3610 RETURN
3620 '
3630 REM ************************************************
3640 REM *                                              *
3650 REM *          Nach oben und unten scrollen        *
3660 REM *                                              *
3670 REM ************************************************
3680 IF tt=1 THEN SOUND 1,200,20:GOTO 3710
3690 tt=tt-1:IF t1>tt+10 THEN t1=t1-1
3700 GOSUB 4900
3710 RETURN
3720 '
3730 IF tt=tg OR tg=0 THEN SOUND 1,200,20:GOTO 3760
3740 tt=tt+1:t1=t1+1:IF t1>tg THEN t1=tg
3750 GOSUB 4900
3760 RETURN
3770 '
3780 REM ************************************************
3790 REM *                                              *
3800 REM * <M>           Monats-Abrechnung              *
3810 REM *                                              *
3820 REM ************************************************
3830 CLS#1
3840 FOR i=1 TO 9:eg(i)=0:NEXT:sg=0:maxi=0
3850 PRINT#1,"  Monats - Abrechnung"
3860 PRINT#1
```

```
3870 INPUT#1, " Monat :";m$
3880 IF LEN(m$)=0 THEN RETURN
3890 FOR i=1 TO tg
3900 IF m$=RIGHT$(da$(i),2) THEN GOSUB 4150
3910 NEXT i
3920 IF sg>0 THEN 3960
3930 PRINT#1," Keine Daten vorhanden"
3940 FOR i=1 TO 2000:NEXT i
3950 RETURN
3960 PRINT#1
3970 PRINT#1," Summe : ";USING "####.##";sg;
3980 PRINT#1," DM"
3990 PRINT#4,"    %  ";
4000 FOR i=1 TO 9
4010 PRINT#4, USING "####.##";eg(i)*100/Sg;
4020 NEXT i
4030 PRINT#4,USING "####.##";sg*100/sg
4040 PRINT#2
4050 FOR i=1 TO 9
4060 PRINT#2,"   ";laden$(i);USING "#####.##";eg(i);
4070 NEXT i
4080 PRINT#2
4090 PRINT#2,TAB(15);"Summe : ";USING "######.##";Sg
4100 PRINT#2
4110 PRINT#2,TAB(12);"Weiter bitte eine Taste"
4120 in$=INKEY$:IF in$= "" THEN 4120
4130 CLS#2:CLS#4
4140 RETURN
4150 FOR j=1 TO 9
4160 IF wert(j,i) > maxi THEN maxi=wert(j,i)
4170 eg(j)=eg(j)+wert(j,i)
4180 NEXT j
4190 sg=sg+su(i)
4200 RETURN
4210 '
4220 REM ************************************************
4230 REM *                                              *
4240 REM * <G>          Gesamt-Abrechnung               *
4250 REM *                                              *
4260 REM ************************************************
4270 CLS#1
4280 FOR i=1 TO 9:eg(i)=0:NEXT:sg=0:maxi=0
4290 PRINT#1,"  Gesamt - Abrechnung"
4300 PRINT#1
4310 PRINT#1,"     Bitte warten !"
4320 FOR i=1 TO tg: GOSUB 4150: NEXT i
4330 IF sg=0 THEN 3930
4340 GOTO 3960
4350 '
```

```
4360 REM ************************************************
4370 REM *                                              *
4380 REM * <A>          Aendern oder Loeschen           *
4390 REM *                                              *
4400 REM ************************************************
4410 CLS#1
4420 PRINT#1,"Aendern eines Eintrags"
4430 PRINT#1
4440 INPUT#1, " Datum :";d$
4450 IF LEN(d$)=0 THEN RETURN
4460 FOR f=1 TO tg
4470 IF da$(f)=d$ THEN 4550
4480 NEXT f
4490 PRINT#1
4500 PRINT#1," N I C H T  vorhanden"
4510 SOUND 1,200,20
4520 FOR i=1 TO 2000:NEXT i
4530 RETURN
4540 '
4550 GOSUB 2770
4560 PRINT#1
4570 PRINT#1," Loeschen  (j/n)  ?"
4580 a$=INKEY$:IF a$="" THEN 4580
4590 IF UPPER$(a$)="J" THEN 4620
4600 IF UPPER$(a$)="N" THEN 4730
4610 GOTO 4580
4620 IF f=tg THEN  4700
4630 FOR i=f TO tg-1
4640 FOR j=1 TO 9
4650 wert(j,i)=wert(j,i+1)
4660 NEXT j
4670 da$(i)=da$(i+1)
4680 su(i)=su(i+1)
4690 NEXT i
4700 tg=tg-1:GOSUB 4830
4710 RETURN
4720 '
4730 su(f)=0:FOR i=1 TO 9
4740 PRINT#2,"    ";laden$(i);USING "#####.##";wert(i,f),
4750 INPUT#2,"      > ";ein$
4760 IF ein$<> "" THEN wert(i,f)=VAL(ein$)
4770 su(f)=su(f)+wert(i,f)
4780 NEXT i
4790 GOSUB 4830
4800 CLS#2
4810 RETURN
4820 '
```

```
4830 REM ************************************************
4840 REM *                                              *
4850 REM *        Tabelle berechnen und ausgeben        *
4860 REM *                                              *
4870 REM ************************************************
4880 IF tg<11 THEN tt=1:t1=tg:GOTO 4900
4890 tt=tg-10:t1=tg
4900 LOCATE 1,1
4910 FOR i=tt TO t1
4920 PRINT da$(i);" :";
4930 FOR j=1 TO 9
4940 PRINT USING "####.##";wert(j,i);
4950 NEXT j
4960 PRINT USING "####.##";su(i);
4970 NEXT i
4980 FOR i=tg-tt+2 TO 11
4990 PRINT l1$;" :";
5000 FOR j=1 TO 10
5010 PRINT USING "####.##";0;
5020 NEXT j
5030 NEXT i
5040 SOUND 1,100,20
5050 RETURN
5060 '
6070 REM ************************************************
```

9.0 RAM-Floppy

In den vergangenen Jahren sind die Preise speziell für Speicherbausteine (RAM) derart gefallen, daß die sogenannte RAM-Floppy immer mehr Verbreitung findet. Eine RAM-Floppy besteht aus handelsüblichen RAM-Bausteinen, die meistens auf einer externen Platine untergebracht sind. Die Speichergröße dieser Karte ist fast beliebig und nur vom benutzten System abhängig. Die Verwaltung einer RAM-Floppy erfolgt allgemein über das Bank-Switching-Verfahren, welches auch im CPC-464/664 zur Verwaltung des Betriebssystems und des Erweiterungs-ROM benutzt wird (siehe Band 1, Kap. 1.0).

Der Vorteil einer RAM-Floppy gegenüber einer Diskettenstation liegt im extrem schnellen Datenzugriff. Da sich die Daten „im Rechner" befinden, kann jedes Datum normal adressiert werden. Weiterhin läßt sich somit auf ein beliebiges Datum der RAM-Floppy zugreifen. Auch wenn Sie keine RAM-Floppy besitzen (für der CPC-464/664 gibt es z.Z. auch keine), können Sie einen Teil des RAM-Speichers vom CPC-464/664 als RAM-Floppy nutzen. Die Vorteile, besonders bei der Bearbeitung kleinerer Datenmengen (bis ca. 30 kByte), ist gegenüber den bisher beschriebenen Verfahren der Datenverwaltung derart groß, daß an dieser Stelle auf dieses Verfahren besonders eingegangen werden soll.

Die Benutzer eines Schneider CPC-6128 haben die Möglichkeit, die gesamte zweite Bank (64 kByte) als RAM-Floppy zu benutzen. Für die Verwaltung der RAM-Floppy stehen hier zusätzlich die Befehle BANKOPEN, BANKREAD und BANKWRITE zur Verfügung, auf die an dieser Stelle nicht weiter eingegangen wird.

Der erste Schritt ist das Bestimmen eines (RAM-) Speicherbereichs, der als RAM-Floppy dienen soll. Im folgenden Beispiel dient der Adressbereich 20000 bis 39999 als RAM-Floppy. Somit stehen 20000 Bytes als Speicher zur

Verfügung. Damit dieser Speicherraum nicht vom Programm oder von dem im Programm benutzten Variablen zerstört (überschrieben) wird, ist der RAM-Bereich oberhalb 19999 mit

```
MEMORY 19999
```

zu schützen. In der RAM-Floppy (die Datei) befinden sich jetzt noch die Werte, die sich vorher in diesem Bereich befunden haben (Variableninhalte, Programm etc). Um sicherzustellen, daß sich keine unsinnigen Werte in der Datei befinden, wird mit

```
100 MEMORY 19999
110 FOR adr=20000 TO 39999
120 POKE adr,32
130 NEXT adr
```

die gesamte Datei mit dem Wert 32 (ASCII-Wert für das Space-Zeichen) gefüllt und somit „gelöscht". Nun besteht die Datei aus 20000 Leerzeichen. Damit ist eine neue Datei kreiert und mit Leerzeichen gefüllt worden, in die Sie nun Daten ablegen können.

9.1 Die relative Adressierung

Im Gegensatz zur sequentiellen Datenverwaltung müssen alle Datensätze, die in der RAM-Floppy abzulegen sind, eine konstante Länge aufweisen. Nur dadurch ist gewährleistet, daß jedes beliebige Datum der Datei wiedergefunden werden kann. Die relative Adressierung geht davon aus, daß sich ein bestimmtes Datum an einer exakt definierten Position innerhalb der Datei (RAM-Bereich) befindet. Diese Position verhält sich „relativ" zum Anfang der Datei. Ein Beispiel soll das Prinzip der relativen Adressierung verdeutlichen.

Es sind 3 Datensätze mit je 40 Zeichen vorhanden. Der erste Datensatz belegt die ersten 40 Zeichen der Datei (0-39), der zweite die Zeichen 40 bis 79, usw..

Somit hat jeder Datensatz eine eindeutig definierte Position innerhalb der Datei. Der Anfang eines Datensatzes ist nun leicht zu bestimmen:

(Datensatznummer - 1) * Länge des Datensatzes

(3-1)*40 = 80 : Datensatz 3 beginnt ab Position 80
(2-1)*40 = 40 : Datensatz 2 beginnt ab Position 40
(1-1)*40 = 0 : Datensatz 1 beginnt ab Position 0

Um einen Datensatz innerhalb der RAM-Floppy zu finden, ist zur errechneten Position noch die Startadresse (hier 20000) der Datei zu addieren.

20000+(3-1)*40 = 20080

Die errechnete Adresse (20080) ist nun die Adresse, in der der gesuchte Datensatz beginnt. Die nächsten 40 Zeichen (Datensatzlänge) bilden den gesamten Datensatz. Mit einer einfachen Schleife können alle Zeichen einer Variablen zugewiesen werden:

```
100 FOR adr=20080 TO 20080+39
110 text$=text$+CHR$(PEEK(adr))
120 NEXT adr
130 PRINT text$
```

Das Schreiben eines Datensatzes in die Datei ist identisch. Der zu sichernde Datensatz befindet sich in der Variablen „text$" und besitzt eine Länge von exakt 40 Zeichen (ggf. mit Leerzeichen füllen). Nacheinander wird der Text (Zeichen für Zeichen) in einen Datensatz der Datei abgelegt. Die Variable „dsnr" gibt dabei die Datensatznummer an, in der der Text abzulegen ist.

```
100 INPUT "Datensatznummer :";dsnr
110 text$="Testeingabe mit genau 40 Zeichen
120 FOR i=0 TO 39
130 adr=20000+(dsnr-1)*40+i
140 POKE adr,ASC(MID$(text$,i+1,1))
150 NEXT i
```

Die Handhabung und Programmierung der relativen Datei ist recht einfach. Die Vorteile liegen u.a. in der einfachen Datensicherung auf Diskette oder Kassette. Die gesamte Datei kann mit

SAVE "DATEI.BIN",B,20000,20000

gesichert werden (siehe 2.1). Das Laden dieser binären Datei erfolgt wie gewohnt mit

```
LOAD ''DATEI.BIN''
```

Dabei wird die Datei automatisch wieder direkt in den Speicherbereich geladen, von woaus sie beim SAVEn geholt wurde. Ein weiterer Vorteil bei dieser Methode der Datenverwaltung ist der, daß sich die Daten nicht in Variablen(felder) befinden. Dadurch läßt sich die teilweise sehr störende Garbage Collection verhindern.

9.2 Datensatz und Datenfeld

Da alle Datensätze innerhalb der relativen Datei eine konstante Länge aufweisen müssen, kommt der Planung einer Datei eine besondere Bedeutung zu. Besteht ein Datensatz aus mehreren Datenfeldern, muß auch für jedes Datenfeld ein bestimmter Bereich innerhalb des Datensatzes reserviert sein. Die Länge aller Datenfelder ergibt dann die Gesamtlänge des Datensatzes.

Datenfeld 1 : Name	20 Zeichen
Datenfeld 2 : Ort	15 Zeichen
Datenfeld 3 : Straße	20 Zeichen
Datenfeld 4 : Telefon	10 Zeichen
Datensatz	65 Zeichen

Ein Datensatz benötigt in dem obigen Beispiel einen Speicherplatz von 65 Zeichen. Jedes Datenfeld besitzt eine konstante Größe und befindet sich damit auch an einer bestimmten Position innerhalb des Datensatzes: Die ersten 20 Zeichen des Datensatzes beinhalten den Namen, die folgenden 15 den Ort usw.. Ist ein Datenfeld kleiner als der für ihn reservierte Platz, muß das Datenfeld mit Zeichen (z.B. Leerzeichen) gefüllt werden, um damit die Datenfeldlänge einzuhalten.
Dieser hier als Beispiel gewählte Aufbau eines Datensatzes ist zwingend für alle weiteren Datensätze der Datei. Egal auf welchen Datensatz der Datei zugegriffen wird, in den ersten 20 Zeichen befindet sich der Name.

Das Auffüllen eines Datensatzes bis zu seiner festgelegten Länge ist mit Hilfe der beiden Befehle STRING$ und SPACE$ im Schneider-BASIC sehr einfach möglich:

```
df$=df$+SPACE$(20-LEN(df$))          mit Leerzeichen auffüllen
df$=df$+STRING$(20-LEN(df$),"*")     mit ,,*'' auffüllen
```

In der Variablen df$ befindet sich das Datenfeld (z.B. Name), welches bis zu einer Länge von 20 Zeichen aufzufüllen ist. Der SPACE$-Befehl gibt einen String zurück, der soviel Leerzeichen (Space-Zeichen) enthält, wie das Argument angibt. Beim STRING$-Befehl kann der erzeugte String aus beliebigen ASCII-Zeichen bestehen (hier ,,*'').

Der Datensatz ergibt sich aus dem Zusammenfassen aller Datenfelder:

Datensatz = Datenfeld 1 + Datenfeld 2 + ... Datenfeld n

Wie schon in 9.1 beschrieben, kann nur der so entstandene Datensatz in die relative Datei (RAM-Floppy) geschrieben werden.

Umgekehrt ist ein Datensatz wieder in Datenfelder zu zerlegen, indem Datenfeldern bestimmte Teile des Datensatzes zugewiesen werden. Das Zerlegen eines Datensatzes (ds$) in die oben bezeichneten Datenfelder läßt sich mit den BASIC-Befehlen MID$, LEFT$ und RIGHT$ durchführen:

```
df1$=MID$(ds$,1,20)          oder LEFT$(ds$,20)
df2$=MID$(ds$,21,15)
df3$=MID$(ds$,36,20)
df4$=MID$(ds$,56,10)         oder RIGHT$(ds$,10)
```

9.3 DATA-FLEX – eine flexible Datei

Auf den nächsten Seiten finden Sie ein Dateiprogramm, welches nach dem Prinzip der „RAM-Floppy" arbeitet. Das Programm selbst ist 14 kByte lang (plus 4 kByte für den Kassetten-/Diskettenpuffer), so daß für die Daten ein Speicherraum von 22500 Bytes am Ende des RAM-Speichers reserviert werden konnte. Beim DATA-FLEX-Programm handelt es sich mehr um einen „Dateigenerator" als um eine Datei. Mit Hilfe dieses Programms können Sie sich eine oder mehrere Dateien nach Ihren ganz speziellen Wünschen erstellen. Sei es eine Bücherkartei, Videoverwaltung oder eine Adressendatei.

Das Programm gliedert sich in drei Teile:

1. Eine neue Kartei erstellen.
2. Kartei bearbeiten.
3. Kartei laden und sichern.

In den Programmzeilen 3790 bis 4330 finden Sie den Teil zum Generieren einer neuen Kartei. Hier können Sie sich eine Kartei mit bis zu 15 Datenfeldern (jeweils bis zu 50 Zeichen) pro Datensatz erstellen. Die Bezeichnung und die Länge jedes Datenfeldes ist ebenso wie die Anzahl der Datenfelder wahlfrei zu gestalten. Alle Daten der so entstandenen Karteimaske werden beim Erstellen in mehreren Variablen gespeichert und auf dem Bildschirm sichtbar gemacht.

z	:Anzahl der Datenfelder
ft$(1) – ft$(z)	:Bezeichnung der Datenfelder 1 bis z
fl(1) – fl(z)	:Länge der Datenfelder 1 bis z
gl	:Datensatzlänge (Summe der Datenfelder)
ma	:Anzahl der möglichen Datensätze (22500/gl)
na$	:Name der Datei (interne Bezeichung)

Diese dateispezifischen Informationen werden neben den eigentlichen Daten in einer sequentiellen Datei geführt. Beim Laden der Kartei werden zunächst diese Informationen geladen und bestimmen damit die Form und Aufgabe der Kartei. Im Programmteil 1390 bis 1730 ist das Laden einer bereits erstellten und gesicherten Kartei zu finden.

```
                    Neue Kartei erstellen

) 1:Autor: Name,Vorname: [....................]
) 2:Titel              : [....................]
) 3:Verlag             : [................]
) 4:ISBN               : [................]
) 5:Preis              : [......]
) 6:Seiten             : [....]
) 7:Rechnertyp         : [................]
) 8:Schwerpunkt        : [........................................]
) 9:                   [
)10:                   [
)11:                   [
)12:                   [
)13:                   [
)14:                   [
)15:                   [

) Gesamtlaenge eines Datensatzes    :  155 Byte
) Anzahl der moeglichen Datensaetze :  145

) Alle Angaben richtig (j/n) ? J
) Name der Datei :EDV - Buecher (Neuerscheinungen 1985)
```

In dem obigen Bildschirmausdruck (das Hardcopy-Programm finden Sie im 1. Band Kapitel 12.4) ist eine typische Bücherkartei generiert worden. Von den 15 möglichen Datenfeldern sind 8 benutzt. Die Datenfeldbezeichnung von Feld 1 ist „Autor: Name,Vorname:" und die Datenfeldlänge beträgt 20 Zeichen (20 Punkte). Nachdem die Karteimaske erstellt ist (beim Feldtitel nur die RETURN-Taste betätigen), errechnet sich das Programm die Datensatzlänge (gl) und daraus die Anzahl der möglichen Datensätze (22500/155). Nach der Bestätigung, daß alle Angaben richtig sind, können Sie der erstellten Kartei einen Namen geben (bis 80 Zeichen). Damit ist eine neue Kartei generiert und das Arbeiten mit der erstellten Kartei kann beginnen.

```
)DS Nummer :    1  )DS belegt :    1   )DS frei  :  144   )DS loesch :    0

                    EDV - Buecher (Neuerscheinungen 1985)

1  Autor: Name,Vorname:  Thoma, Manfred Walter
2  Titel               : CPC 464/664 - Praxis III
3  Verlag              : Huethig
4  ISBN                : 3-7785-1149-1
5  Preis               :  34.00
6  Seiten              :  200
7  Rechnertyp          : CPC 464/664
8  Schwerpunkt         : CP/M, Anwendung, Programmierung, Programme

) Bitte Ihr Kommando
```

Der Arbeitsbildschirm gliedert sich in drei Teile. Die oberste Zeile ist die Statuszeile, die alle wichtigen Informationen über den Zustand der Kartei enthält. Die „DS Nummer" gibt die Karteikartennummer der aktuell auf dem Bildschirm sichtbaren Karte wieder. „DS belegt" zeigt die letzte belegte Datensatznummer an und „DS frei" zeigt an, wieviel Datensätze noch zur Verfügung stehen. Die letzte Information „DS loesch" gibt die Anzahl der gelöschten Datensätze zwischen dem ersten und dem letzten belegten Datensatz wieder.
Der größte Teil des Bildschirms stellt die Karteikarte der Datei dar, wie sie beim Erstellen der Kartei generiert wurde.
Der untere Teil ist der „Eingabe-Bildschirm" und dient zur Eingabe von Befehlen und zur Ausgabe weiterer Informationen.

Das Bearbeiten der Datei nimmt natürlich den größten Teil des Programms in Anspruch. Es stehen insgesamt 10 Befehle zur Verfügung:

(H)ilfe — Zeigt alle Befehle an.

(E)ingabe — Eingabe einer neuen Karteikarte.
Sie können nacheinander alle Datenfelder mit Informationen füllen. Betätigen Sie beim ersten Datenfeld nur die ENTER-Taste, kommen Sie zurück in die Kommandoebene (Abbruch, bei allen anderen Befehlen ähnlich). Ist kein Platz mehr in der Datei, erfolgt eine Meldung. Befindet sich zwischen dem ersten und letzten benutzten Datensatz ein gelöschter Eintrag („DS loesch" größer als 0), sucht das Programm zunächst diesen Platz und füllt ihn mit den neuen Eingaben.

(A)endern — Ändern eines Datensatzes.
Nach Eingabe der Datensatznummer kann der Inhalt geändert oder gelöscht werden. Mit Betätigung der RETURN-Taste wird beim Ändern der alte Inhalt des jeweiligen Datenfeldes beibehalten.

(F)inden — Sucht einen bestimmten Datensatz.
DATA-FLEX ermöglicht einen Suchvorgang nach jedem Datenfeld. Das Programm fragt nach der (Daten) Feldnummer. Geben Sie z.B. eine „2" ein, so können Sie Daten nach dem Kriterium „Titel" suchen. Mit Eingabe der „0" können Sie nach einer Kartennummer (DS Nummer) suchen.
Nachdem Sie eine Feldnummer gewählt haben, können Sie die Daten eingeben, nach welchen das Programm die Datensätze aus-

wählen und anzeigen soll. Haben Sie das Feld 2 gewählt (Titel) und geben als Suchbegriff nur das „C" ein, so wird die erste Karteikarte der Datei gesucht, deren Datenfeld 2 mit einem „C" beginnt. Ist ein entsprechender Eintrag gefunden, können Sie nach weiteren zutreffenden Daten suchen. Die Länge des Suchbegriffs kann von einem Zeichen bis zur maximalen Feldlänge reichen. Als weitere Option dient das Fragezeichen „?" um nichtrelevante, führende Zeichen zu ignoriren. Mit der Eingabe von „??oma" sucht das Programm einen Eintrag, der an 3. bis 5. Stelle die Folge „oma" besitzt.

(S)ichern — Sichern der Datei.
Sichert die Daten (.BIN) sowie die Hilfsdatei (.SEQ) mit den dateispezifischen Informationen auf Diskette/Kassette.

(Q)uit — Programmende.
Nach der Bestätigung gelangen Sie wieder in das Hauptmenü (siehe weiter unten).

(P)rint — Leitet alle Ausgaben auf den Drucker um (Ein-/Ausschalter).

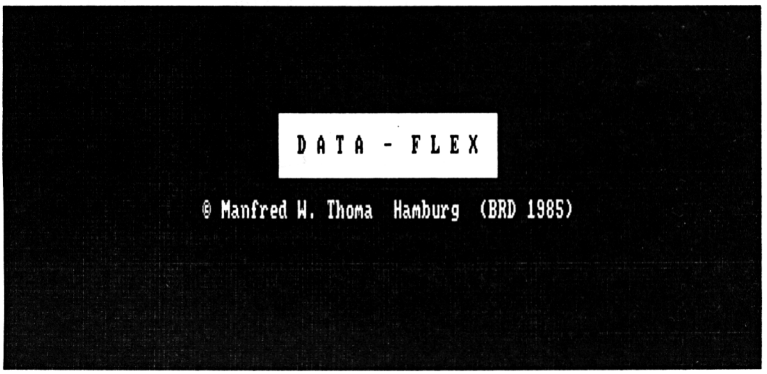
\+ — Zeigt die nächste Karteikarte,
die auf die aktuelle folgt. Ist sie gelöscht, erfolgt eine Meldung.

– — Zeigt die vorherige Karteikarte,
die vor der aktuellen liegt. Ist sie gelöscht, erfolgt eine Meldung.

* — Zeigt nacheinander alle Karten der Datei an.
Mit der SPACE-Taste kann die Ausgabe gestoppt und weitergeführt werden. Ein Abbruch dieser Funktion ist mit der „/"-Taste möglich.

Nach dem Laden des Programms gelangen Sie in das Hauptmenü, von wo aus Sie wählen können, ob Sie

V	Eine vorhandene Kartei laden,
N	eine neue Kartei erstellen
E	oder das Programm beenden wollen.




```
) (V)orhandene Kartei laden .....
) (N)eue Kartei erstellen .....
) (E)nde des Programms .....

) Bitte Ihre Wahl ? N
```

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *              D A T A  -  F L E X              *
1030 REM *                                               *
1040 REM ************************************************
1050 SPEED WRITE 1
1060 MEMORY 19903
1070 INK 0,1:INK 1,26:BORDER 1
1080 MODE 2
1090 DEF FN hb(ad)=&FF AND INT(ad/256)
1100 DEF FN lb(ad)=&FF AND UNT(ad)
1110 DIM ft$(15),fl(15),eg$(15),fo$(15)
1120 WINDOW#0,1,80,21,25
1130 WINDOW#2,1,80,3,19
1140 WINDOW#1,1,80,1,1
1150 MOVE 0,376:DRAW 639,376,1
1160 MOVE 0,86:DRAW 639,86,1
1170 sp=19904:p=2
1180 CLS:CLS#1:PRINT#2,CHR$(24);:CLS#2
1190 LOCATE#2,30,6
1200 PRINT#2,CHR$(24);SPACE$(23)
1210 LOCATE#2,30,7
1220 PRINT#2, "  D A T A  -  F L E X  "
1230 LOCATE#2,30,8
1240 PRINT#2,SPACE$(23);CHR$(24)
1250 LOCATE#2,22,10
1260 PRINT#2,CHR$(164);
```

```
1270 PRINT#2, " Manfred W. Thoma  Hamburg  (BRD 1985)"
1280 PRINT TAB(25);"> <V>orhandene Kartei laden ........."
1290 PRINT TAB(25);"> <N>eue Kartei erstellen ..........."
1300 PRINT TAB(25);"> <E>nde des Programms ............."
1310 LOCATE 25,5
1320 INPUT "> Bitte Ihre Wahl ";in$
1330 IF UPPER$(in$)="N" THEN 3790
1340 IF UPPER$(in$)="V" THEN 1390
1350 IF UPPER$(in$)="E" THEN PRINT#2,CHR$(24):CLS:END
1360 PRINT CHR$(7);
1370 GOTO 1310
1380 '
1390 REM ************************************************
1400 REM *                                              *
1410 REM * <V>        Vorhandene Kartei laden           *
1420 REM *                                              *
1430 REM ************************************************
1440 CLS:PRINT#2,CHR$(24);
1450 PRINT " >    Vorhandene Kartei laden!    <"
1460 LOCATE 1,3
1470 PRINT " > Bereiten Sie die Kassette vor  <"
1480 PRINT " > und betaetigen dann eine Taste.<"
1490 in$=INKEY$:IF in$="" THEN 1490
1500 LOCATE 40,1
1510 INPUT " >   Name (bis 8 Zeichen) :";n$
1520 IF n$="" THEN 1180
1530 IF LEN(n$)> 8 THEN PRINT CHR$(7);:GOTO 1500
1540 LOCATE 40,3
1550 PRINT" >   Druecken Sie die PLAY-Taste und   <"
1560 LOCATE 40,4
1570 PRINT" > dann bitte irgendeine andere Taste! <"
1580 in$=INKEY$:IF in$="" THEN 1580
1590 OPENIN "!"+n$+".seq"
1600 LINE INPUT#9,na$
1610 INPUT#9,z
1620 INPUT#9,gl
1630 INPUT#9,ma
1640 INPUT#9,an
1650 INPUT#9,gel
1660 FOR i=1 TO z
1670 LINE INPUT#9,ft$(i)
1680 INPUT#9,fl(i)
1690 NEXT i
1700 CLOSEIN
1710 LOAD "!"+n$+".bin",sp
1720 GOSUB 4470:GOSUB 4790:GOSUB 5040
1730 '
```

```
1740 REM ************************************************
1750 REM *                                              *
1760 REM *                  M e n u e                   *
1770 REM *                                              *
1780 REM ************************************************
1790 CLS
1800 LOCATE 2,1
1810 PRINT " > Bitte Ihr Kommando "
1820 LOCATE 72,1:IF p=8 THEN PRINT "<Drucker>":GOTO 1840
1830 PRINT "          "
1840 in$=INKEY$:IF in$="" THEN 1840
1850 in$=UPPER$(in$)
1860 IF in$="E" THEN 1990
1870 IF in$="F" THEN 2360
1880 IF in$="S" THEN 3440
1890 IF in$="A" THEN 3050
1900 IF in$="Q" THEN 4900
1910 IF in$="P" THEN IF p=2 THEN p=8:ELSE p=2
1920 IF in$="+" AND dn<an THEN dn=dn+1:GOTO 2790
1930 IF in$="-" AND dn>1 THEN dn=dn-1:GOTO 2790
1940 IF in$="*" THEN 4390
1950 IF in$="H" THEN 5210
1960 PRINT CHR$(7);
1970 GOTO 1800
1980 '
1990 REM ************************************************
2000 REM *                                              *
2010 REM * <E>          Eingaben neuer Karten           *
2020 REM *                                              *
2030 REM ************************************************
2040 zp=sp+an*gl:dn=(zp-sp)/gl+1
2050 IF an=ma AND gel=0 THEN 2320
2060 IF gel=0 THEN 2140
2070 PRINT " > Suche freien Speicherraum <"
2080 FOR i=sp TO sp+an*gl STEP gl
2090 IF PEEK(i)=255 THEN 2120
2100 NEXT i
2110 STOP
2120 dn=(i-sp)/gl+1
2130 zp=i:cl=1
2140 CLS:GOSUB 4790:GOSUB 4660
2150 PRINT " > Eingabe neuer Daten <"
2160 FOR i= 1 TO z
2170 PRINT " > ";ft$(i);"  ";STRING$(fl(i),".");CHR$(11)
2180 LOCATE 26,VPOS(#0):LINE INPUT eg$(i)
2190 IF eg$(1)="" THEN 1740
2200 IF LEN(eg$(i))>fl(i) THEN PRINT CHR$(7);:GOTO 2170
2210 eg$(i)=eg$(i)+SPACE$(fl(i)-LEN(eg$(i)))
2220 LOCATE#2,27,i+2
```

```
2230 PRINT#2,eg$(i)
2240 FOR j=1 TO fl(i)
2250 POKE j+zp-1,ASC(MID$(eg$(i),j,1))
2260 NEXT j
2270 zp=zp+fl(i)
2280 NEXT i
2290 IF cl=1 THEN gel=gel-1:cl=0:GOSUB 4710:GOTO 1740
2300 IF dn>an THEN an=an+1: GOSUB 4710
2310 GOTO 1740
2320 PRINT "  > Keine weiteren Eingaben moeglich !!!"
2330 FOR i=1 TO 100:SOUND 1,800,5,7:NEXT i
2340 GOTO 1740
2350 '
2360 REM ************************************************
2370 REM *                                              *
2380 REM * <F>          Finden von Eintraegen           *
2390 REM *                                              *
2400 REM ************************************************
2410 CLS
2420 IF an=0 THEN 1740
2430 PRINT " > Suchen nach Kategorie <"
2440 PRINT
2450 INPUT " > Feldnummer :";kn$:kn=VAL(kn$)
2460 IF kn <0 OR kn>z THEN 1740
2470 IF kn>0 THEN 2520
2480 LOCATE 1,3
2490 INPUT " > Karten-Nummer :";dn$:dn=VAL(dn$)
2500 IF dn<=0 OR dn>an THEN 1740
2510 GOTO 2790
2520 LOCATE 1,3
2530 PRINT " > ";ft$(kn);STRING$(fl(kn),".");CHR$(11)
2540 LOCATE 24,VPOS(#0):LINE INPUT eg$(0)
2550 IF eg$(0)="" THEN 1740
2560 IF LEN(eg$(0))>fl(kn) THEN PRINT CHR$(7);:GOTO 2520
2570 lz=0
2580 IF MID$(eg$(0),lz+1,1)="?" THEN lz=lz+1:GOTO 2580
2590 IF lz>0 THEN eg$(0)=MID$(eg$(0),lz+1,LEN(eg$(0))-lz)
2600 b1=ASC(LEFT$(eg$(0),1))
2610 su=0:z1=0
2620 FOR i=1 TO kn-1
2630 su=su+fl(i)
2640 NEXT i
2650 FOR i=sp+su+lz TO sp+an*gl+lz STEP gl
2660 z1=z1+1
2670 IF PEEK(i) <> b1 THEN 2720
2680 FOR j=1 TO LEN(eg$(0))
2690 IF PEEK(j+i-1)<>ASC(MID$(eg$(0),j,1)) THEN 2720
2700 NEXT j
2710 GOTO 2780
```

```
2720 NEXT i
2730 CLS
2740 PRINT CHR$(7);
2750 PRINT " > Keine (weiteren) Daten vorhanden !!! <"
2760 FOR i=1 TO 3000:NEXT i
2770 kn=0:GOTO 1740
2780 dn=z1
2790 PRINT CHR$(7);
2800 ad=sp+dn*gl-gl:GOSUB 5130
2810 IF ASC(LEFT$(fo$(1),1))<>255 THEN 2880
2820 IF kn=-2 THEN 4410
2830 CLS
2840 PRINT " > Karteikarte";dn;"ist geloescht !!! <"
2850 PRINT CHR$(7);
2860 FOR i=1 TO 3000:NEXT i
2870 GOTO 1740
2880 IF p=2 THEN 2910
2890 PRINT#8
2900 PRINT#8,"Datensatznummer    :";USING"####";dn
2910 FOR w=1 TO z
2920 LOCATE#2,27,w+2:PRINT#2,fo$(w)
2930 IF p=2 THEN 2950
2940 PRINT#8,ft$(w);fo$(w)
2950 NEXT w
2960 GOSUB 4710
2970 IF kn=-1 THEN 3170
2980 IF kn=-2 THEN 4410
2990 IF z1=an OR kn=0 THEN 3030
3000 CLS
3010 PRINT " > Weitere '";eg$(0);"' suchen ";:INPUT in$
3020 IF UPPER$(in$)="J" OR in$="" THEN 2720
3030 kn=0:GOTO 1740
3040 '
3050 REM ************************************************
3060 REM *                                              *
3070 REM * <A>         Aendern oder Loeschen            *
3080 REM *                                              *
3090 REM ************************************************
3100 CLS
3110 PRINT " > Aendern oder Loeschen <"
3120 LOCATE 1,3
3130 INPUT " > Karten-Nummer :";dn$:dn=VAL(dn$)
3140 IF dn<1 OR dn>an THEN 1740
3150 kn=-1
3160 GOTO 2790:kn=0
3170 INPUT " > (A)endern oder (L)oeschen ";in$
3180 IF UPPER$(in$)="A" THEN 3260
3190 IF UPPER$(in$)="L" THEN 3210
3200 GOTO 1740
```

```
3210 FOR i=sp+dn*gl-gl TO sp+dn*gl-1
3220 POKE i,255
3230 NEXT i:gel=gel+1:IF dn=an THEN an=an-1:gel=gel-1
3240 dn=0:GOSUB 4660:GOSUB 4790
3250 GOTO 1740
3260 CLS:zp=sp+dn*gl-gl
3270 FOR i= 1 TO z
3280 PRINT ft$(i);"  ";fo$(i)
3290 PRINT TAB(21);"> ";STRING$(fl(i),".");CHR$(11)
3300 LOCATE 23,VPOS(#0):LINE INPUT eg$(0)
3310 IF eg$(0)="" THEN eg$(i)=fo$(i):GOTO 3360
3320 IF LEN(eg$(0))>fl(i) THEN PRINT CHR$(7);:GOTO 3280
3330 IF LEFT$(eg$(0),1)<> CHR$(92) THEN 3350
3340 eg$(i)=SPACE$(fl(i)):GOTO 3360
3350 eg$(i)=eg$(0)+SPACE$(fl(i)-LEN(eg$(0)))
3360 LOCATE#2,27,i+2:PRINT#2,eg$(i)
3370 FOR j=1 TO fl(i)
3380 POKE zp+j-1,ASC(MID$(eg$(i),j,1))
3390 NEXT j
3400 zp=zp+fl(i)
3410 NEXT i
3420 GOTO 1740
3430 '
3440 REM ************************************************
3450 REM *                                              *
3460 REM * <S>      Sichern (SAVE) der Kartei           *
3470 REM *                                              *
3480 REM ************************************************
3490 IF an=0 THEN 1740
3500 CLS:PRINT " > Sichern der vorhandenen Kartei <"
3510 LOCATE 1,3
3520 PRINT " > Bereiten Sie die Kassette vor  <"
3530 PRINT " > und betaetigen dann eine Taste.<"
3540 in$=INKEY$:IF in$="" THEN 3540
3550 LOCATE 40,1
3560 INPUT " > Name (bis 8 Zeichen) :";n$
3570 IF n$="" THEN 1740
3580 IF LEN(n$)> 8 THEN PRINT CHR$(7);:GOTO 3550
3590 LOCATE 40,3
3600 PRINT" > Druecken Sie die REC und PLAY-Taste <"
3610 LOCATE 40,4
3620 PRINT" > dann bitte irgendeine andere Taste! <"
3630 in$=INKEY$:IF in$="" THEN 3630
3640 OPENOUT "!"+n$+".seq"
3650 PRINT#9,na$
3660 PRINT#9,z
3670 PRINT#9,gl
3680 PRINT#9,ma
3690 PRINT#9,an
```

```
3700 PRINT#9,gel
3710 FOR i=1 TO z
3720 PRINT#9,ft$(i)
3730 PRINT#9,fl(i)
3740 NEXT i
3750 CLOSEOUT
3760 SAVE "!"+n$+".bin",b,sp,an*gl
3770 GOTO 1740
3780 '
3790 REM ******************************************************
3800 REM *                                                    *
3810 REM * <N>          Neue Kartei erstellen                 *
3820 REM *                                                    *
3830 REM ******************************************************
3840 gl=0
3850 CLS:PRINT#2,CHR$(24);:CLS#2
3860 PRINT#1,TAB(20);CHR$(24);
3870 PRINT#1," N e u e  K a r t e i  e r s t e l l e n "
3880 PRINT#1,CHR$(24);
3890 PRINT#2,CHR$(24)
3900 FOR i=1 TO 15
3910 PRINT#2," >";USING "##";i;
3920 PRINT#2,":";SPACE$(20);CHR$(24);" ";CHR$(24);"Ä"
3930 NEXT i
3940 z=1
3950 CLS
3960 PRINT " > Feldtitel ";z;" :";:LINE INPUT ft$(z)
3970 IF LEN(ft$(z))=0 THEN 4130
3980 IF LEN(ft$(z))>19 THEN PRINT CHR$(7):GOTO 3950
3990 ft$(z)=ft$(z)+SPACE$(19-LEN(ft$(z)))+":"
4000 LOCATE#2,6,z+1
4010 PRINT#2,ft$(z)
4020 LOCATE 1,3
4030 PRINT " > Feldlaenge";z;" :";:INPUT eg$(0)
4040 fl(z)=VAL(eg$(0))
4050 IF fl(z)<1 OR fl(z)>50 THEN PRINT CHR$(7): GOTO 4020
4060 LOCATE#2,28,z+1
4070 FOR i=1 TO fl(z)
4080 PRINT#2,".";
4090 NEXT i
4100 PRINT#2,"ü"
4110 z=z+1:IF z=16 THEN 4130
4120 GOTO 3950
4130 PRINT#2,CHR$(24)
4140 z=z-1:IF z=0 THEN 1180
4150 FOR i=1 TO z:gl=gl+fl(i):NEXT i
4160 CLS
4170 PRINT " > Gesamtlaenge eines Datensatzes    :";
4180 PRINT  USING "#####";gl;
```

```
4190 PRINT " Byte"
4200 ma=INT(22500/gl)
4210 PRINT " > Anzahl der moeglichen Datensaetze :";
4220 PRINT  USING "#####";ma;
4230 LOCATE 1,4
4240 INPUT " > Alle Angaben richtig (j/n) ";in$
4250 IF UPPER$(in$)="N" THEN PRINT#2,CHR$(24):GOTO 3840
4260 IF UPPER$(in$)="J" THEN 4290
4270 PRINT CHR$(7)
4280 GOTO 4230
4290 LINE INPUT " > Name der Datei :";na$
4300 an=0:dn=0:gel=0
4310 GOSUB 4470:GOSUB 4790:GOSUB 5040
4320 GOTO 1740
4330 '
4340 REM ******************************************************
4350 REM *                                                    *
4360 REM * <*>          Anzeigen aller Datensaetze            *
4370 REM *                                                    *
4380 REM ******************************************************
4390 kn=-2:FOR dn=1 TO an
4400 GOTO 2790
4410 IF INKEY$="" THEN 4440
4420 in$=INKEY$:IF in$="" THEN 4420
4430 IF IN$="ö" THEN 4450
4440 NEXT dn
4450 kn=0:GOTO 1740
4460 '
4470 REM ******************************************************
4480 REM *                                                    *
4490 REM * <N>             Bildschirmaufbau                   *
4500 REM *                                                    *
4510 REM ******************************************************
4520 CLS:CLS#2
4530 PRINT#2,CHR$(24);
4540 i=INT((80-LEN(na$))/2)
4550 PRINT#2,SPACE$(i);na$;
4560 PRINT#2,SPACE$(80-i-LEN(na$))
4570 PRINT#2,CHR$(24);
4580 FOR i= 1 TO z
4590 LOCATE#2, 1,i+2
4600 PRINT#2,USING "##";i;
4610 PRINT#2,"  ";ft$(i);
4620 NEXT i
4630 GOSUB 4710
4640 RETURN
4650 '
```

```
4660 REM ******************************************************
4670 REM *                                                    *
4680 REM *                    Status-Zeile                    *
4690 REM *                                                    *
4700 REM ******************************************************
4710 LOCATE#1,1,1
4720 PRINT#1,"  >DS Nummer :";
4730 PRINT#1,CHR$(24);USING "#####";dn;
4740 PRINT#1," ";CHR$(24);"  >DS belegt :";USING "#####";an;
4750 PRINT#1,"   >DS frei   :";USING "#####";ma-an;
4760 PRINT#1,"   >DS loesch :";USING "#####";gel;
4770 RETURN
4780 '
4790 REM ******************************************************
4800 REM *                                                    *
4810 REM *              Feldlaenge markieren                  *
4820 REM *                                                    *
4830 REM ******************************************************
4840 FOR i=1 TO z
4850 LOCATE#2,27,i+2
4860 PRINT#2,STRING$(fl(i),".")
4870 NEXT i
4880 RETURN
4890 '
4900 REM ******************************************************
4910 REM *                                                    *
4920 REM * <Q>            Programmende                        *
4930 REM *                                                    *
4940 REM ******************************************************
4950 CLS
4960 BORDER 1,26
4970 PRINT " >   A C H T U N G   !!  <"
4980 PRINT " > Daten gehen verloren !<"
4990 INPUT " > Sind Sie sicher ????";in$
5000 BORDER 1
5010 IF UPPER$(in$)="J" THEN 1180
5020 GOTO 1740
5030 '
5040 REM ******************************************************
5050 REM *                                                    *
5060 REM *                Stringpointer setzen                *
5070 REM *                                                    *
5080 REM ******************************************************
5090 FOR i=1 TO z
5100 POKE §fo$(i),fl(i)
5110 NEXT i
5120 RETURN
5130 '
5140 FOR w=1 TO z
```

```
5150 ad=ad+fl(w-1)
5160 POKE §fo$(w)+1,FN lb(ad)
5170 POKE §fo$(w)+2,FN hb(ad)
5180 NEXT w
5190 RETURN
5200 '
5210 REM ************************************************
5220 REM *                                              *
5230 REM * <H>          Ausgabe aller Befehle           *
5240 REM *                                              *
5250 REM ************************************************
5260 CLS
5270 PRINT " <H> Hilfe    : <E> Eingabe   : <A> Aendern"
5280 PRINT " <F> Finden   : <L> Laden     : <S> Sichern"
5290 PRINT " <Q> Ende     : <P> Drucker   : <+> Vor"
5300 PRINT " <-> Zurueck  : <*> Blaettern"
5310 IF INKEY$="" THEN 5310
5320 GOTO 1740
5330 REM ************************************************
```

Das Programm überzeugt durch seine überraschend hohe Suchgeschwindigkeit und vor allem durch die Möglichkeit, nach jedem Datenfeld zu suchen. Der Aufbau des Bildschirms ist durch die Zuweisung der Variablen mit Hilfe der Variablen-Pointer (siehe 4.0) ausreichend schnell. Für kleine Dateien ist DATA-FLEX ein ideales Werkzeug (Anm.: Die Register der Buchreihe CPC-464/664 sind mit diesem Programm erstellt und bearbeitet worden), gerade dann, wenn Sie nur den Datenrecorder besitzen. Auf Grund der begrenzten Speicherkapazität (22500 Bytes) ist natürlich eine professionelle Nutzung nicht möglich. Doch für kleine private Anwendungen, wie z.B. eine Adressendatei, reicht die Kapazität allemal aus (knapp 300 Einträge bei einer Datensatzlänge von 80 Bytes). Um größere Datenmengen zu verwalten, ist die Benutzung einer (besser zwei) Diskettenstation zwingend erforderlich.

10.0 Direktzugriff (Random Access)

Alle bisher beschriebenen Datenverwaltungsverfahren waren sowohl mit der Diskettenstation als auch mit dem Datenrecorder möglich. Die Anzahl der zu bearbeitenden Daten war maßgeblich von der Speicherkapazität des CPC-464/664 abhängig und damit relativ „klein". Ist eine Datenmenge vorhanden, die über die Speicherkapazität hinaus geht, müssen mehrere, unabhängige Dateien aufgebaut und bei Bedarf aufgerufen werden. Obwohl die Handhabung von Teildateien sehr einfach ist, ist dieses Verfahren gerade im Diskettenbetrieb unbefriedigend. Durch den Aufbau und die Arbeitsweise einer Diskettenstation ist zumindest der Zugriff auf einen beliebigen Block der Diskette möglich. Damit ist die Aufnahmekapazität einer Datei nur noch durch die Speicherkapazität der Diskette begrenzt.

Die Diskettenstation DDI-1 besitzt bei einer formatierten Diskette 40 Spuren a 9 Sektoren (Blöcke), also 360 Blöcke. Jeder Block kann 512 Bytes (1/2 kByte) aufnehmen. Somit könnte eine Datei insgesamt 184.320 Bytes (360*512) Daten aufnehmen. Heute bieten Fremdhersteller Diskettenstationen für den Schneider an, deren Kapazität weit über 500 kByte liegt.

Betrachtet man zunächst einen Block (512 Bytes) als Datensatz einer Datei, so steht eine Datei mit 360 Datensätzen zur Verfügung. Um den ersten Datensatz zu bearbeiten, ist der erste Block der Diskette in den CPC-464/664 zu laden und anschließend wieder in den ersten Block der Diskette zurückzuschreiben. D.h. von der Datei ist maximal nur ein Datensatz, also 512 Bytes, innerhalb des Computers.

Der Zugriff auf einen beliebigen Block der Diskette ist nicht in das BASIC des CPC-464/664 eingebunden, obwohl dieses sehr einfach möglich gewesen wäre. Bei der näheren Untersuchung des DOS (Disketten-ROM) findet man einige „versteckte Befehle", von denen im CPC-Handbuch keine Rede ist. U.a. auch zum Lesen und Beschreiben eines beliebigen Blockes.

10.1 Lesen und Beschreiben eines Blocks

Im folgenden möchte ich Ihnen ein Maschinenunterprogramm vorstellen, das das Lesen und Beschreiben eines beliebigen Blocks auf der Diskette ermöglicht. An dieser Stelle kann nicht detailliert auf die Arbeitsweise des Programms eingegangen werden, da dieses eine grundlegende Analyse des DOS bedingen und damit den Rahmen dieses Buches sprengen würde. Ich verweise deshalb auf weiterführende Bücher, die sich speziell mit dem DOS des CPC-464/664 auseinandersetzen.

Das Programm gliedert sich in zwei Teile: Einlesen eines Blocks und Beschreiben eines Blocks.

Beim Einlesen wird ein Block (512 Bytes) in einen 512 Byte großen Speicherraum (Puffer) des Rechners abgelegt. Nach der Bearbeitung dieses Puffers kann dieser 512 Bytes große Speicherraum wieder zurück auf die Diskette geschrieben werden.

Um das Programm möglichst variabel zu halten, ist der Pufferbereich wahlfrei innerhalb des RAM-Speichers verschiebbar. Der Aufruf zum Lesen eines Blockes von der Diskette hat folgendes Format:

Lesen: CALL &A000, drive, spur, sektor, pufferadresse

Nach dem Einsprung in das Maschinenprogramm (Adresse &A000) folgen jeweils durch ein Komma getrennt 4 Parameter. Mit „drive" ist die Auswahl zwischen den beiden Laufwerken A und B möglich.

drive = 0 Laufwerk A
drive = 1 Laufwerk B

Die beiden folgenden Parameter „spur" und „sektor" bestimmen einen Block auf der Diskette. Dabei darf

spur = 0 bis 39 (40 Spuren) und
sektor = 0 bis 8 (9 Sektoren)

sein. Der letzte Parameter „pufferadresse" bestimmt die Startadresse eines 512 Bytes großen Speicherraums, in den der Block eingelesen wird. Zu beachten ist dabei natürlich, daß der Puffer nicht in einem Bereich liegt, in dem sich wichtige Daten (Programm, Variableninhalte, etc.) befinden. Der eingelesene

Block würde diese Daten zerstören. Auch sollte der Puffer in einem geschützten RAM-Bereich liegen (z.B. MEMORY &8FFF – Pufferadresse ab &9000).

```
CALL &A000,0,9,5,&9000
```

Das obige Beispiel liest den Block Spur 9/Sektor 5 vom Laufwerk A in einen 512 Bytes großen Speicher ab Adresse &9000 ein.

Der Aufruf zum Beschreiben eines Blocks auf der Diskette ist von den Parametern her identisch. Geändert hat sich nur die Aufrufadresse (&A00C):

```
Schreiben: CALL &A00C,0,12,0,&9000
```

Das Beispiel schreibt den Inhalt des 512 Bytes großen Puffers, der ab Adresse &9000 liegt, in den Block Spur 12/Sektor 0 auf das Laufwerk A zurück.

Nach dem Aufruf beider Befehle werden die angegebenen Parameter von Drive, Spur und Sektor jeweils auf ihre Korrektheit überprüft. Liegt einer der Parameter außerhalb des zulässigen Bereichs, wird die gewünschte Routine nicht ausgeführt. Gleichzeitig wird die Speicherstelle &A086 auf &FF (255) gelegt.

&A086 = &00	Routine korrekt eingeleitet.
&A086 = &FF	Parameterfehler. Routine nicht eingeleitet.

Mit einer einfachen Prüfung ist somit festzustellen, ob die gewünschte Routine erfolgreich eingeleitet ist (Die Übergabe und Prüfung der Parameter nimmt den größten Teil des Maschinenprogramms in Anspruch).

```
64000 REM ************************************ M.W. Thoma **
64010 REM *                                                *
64020 REM *         Direktzugriff auf die Diskette         *
64030 REM *                                                *
64040 REM *     Lesen: CALL &A000,DRIVE,SPUR,SEKTOR,PUFFER  *
64050 REM * Schreiben: CALL &A00C,DRIVE,SPUR,SEKTOR,PUFFER  *
64060 REM *                                                *
64070 REM ************************************************
64080 summe=0:FOR i=&A000 TO &A086
64090 READ a$:a=VAL("&"+a$):POKE i,a:summe=summe+a
64100 NEXT i
64110 IF summe=16125 THEN RETURN
```

```
64120 PRINT "Fehler in den DATA-Zeilen":END
64130 '
64140 DATA CD,18,A0,3E,84,32,85,A0,CD,5A,A0,C9,CD,18,A0,3E
64150 DATA 85,32,85,A0,CD,5A,A0,C9,F5,3E,00,32,86,A0,F1,FE
64160 DATA 04,C2,76,A0,DD,7E,07,DD,7E,06,FE,02,D2,76,A0,32
64170 DATA 7D,A0,DD,7E,05,DD,7E,04,FE,28,D2,76,A0,32,7E,A0
64180 DATA DD,7E,03,DD,7E,02,FE,0A,D2,76,A0,CE,40,32,7F,A0
64190 DATA DD,66,01,DD,6E,00,22,80,A0,C9,21,85,A0,CD,D4,BC
64200 DATA 22,82,A0,79,32,84,A0,21,7D,A0,5E,23,56,23,4E,2A
64210 DATA 80,A0,DF,82,A0,C9,E1,3E,FF,32,86,A0,C9,00,00,00
64220 DATA 00,00,00,00,00,00,00
64230 REM ******************************************************
```

```
                        ;*********************************************
                        ;*       Assemblerlisting Direktzugriff      *
                        ;*                                           *
                        ;* Achtung:Das Programm ist für die Formate *
                        ;*      FORMAT und FORMAT V ausgelegt!       *
                        ;*                                           *
                        ;*********************************************
                        ;
                        ORG  &A000
                        ;
                        ;*********************************************
                        ;*   Einsprung für Sektor einlesen &A000     *
                        ;*********************************************
A000 CD18A0    READ     CALL PARA           ;Prüfe Parameter
A003 3E84               LD   A,&84          ;Diskbefehl merken
A005 3285A0             LD   (BEFEHL),A
A008 CD5AA0             CALL DISK           ;und ausführen
A00B C9                 RET                 ;Zurück ins BASIC
                        ;
                        ;*********************************************
                        ;*   Einsprung für Sektor schreiben &A00C    *
                        ;*********************************************
A00C CD18A0    WRITE    CALL PARA           ;Prüfe Parameter
A00F 3E85               LD   A,&85          ;Diskbefehl merken
A011 3285A0             LD   (BEFEHL),A
A014 CD5AA0             CALL DISK           ;und ausführen
A017 C9                 RET                 ;Zurück ins BASIC
                        ;
                        ;*********************************************
                        ;*  Prüft. ob alle Parameter korrekt sind    *
                        ;*********************************************
A018 F5        PARA     PUSH AF
A019 3E00               LD   A,&00          ;Setzt Error-Flag auf 0
A01B 3286A0             LD   (ERROUT),A
A01E F1                 POP  AF
```

```
A01F FE04               CP   &04            ;Folgen 4 Parameter ?
A021 C276A0             JP   NZ,FEHLER      ;Nein, dann Fehler!
A024 DD7E07             LD   A,(IX+7)       ;überlesen
A027 DD7E06             LD   A,(IX+6)       ;Drive-Nummer
A02A FE02               CP   &02            ;Größer 1 ?
A02C D276A0             JP   NC,FEHLER      ;Ja, dann Fehler!
A02F 327DA0             LD   (DRIVE),A      ;Drive-Nummer merken
A032 DD7E05             LD   A,(IX+5)       ;überlesen
A035 DD7E04             LD   A,(IX+4)       ;Spur
A038 FE28               CP   &28            ;Größer 39 ?
A03A D276A0             JP   NC,FEHLER      ;Ja, dann Fehler!
A03D 327EA0             LD   (SPUR),A       ;Spur merken
A040 DD7E03             LD   A,(IX+3)       ;überlesen
A043 DD7E02             LD   A,(IX+2)       ;Sektor
                        ;

                    ;***********************************************
                    ;*  ACHTUNG: Beim IBM-Format Adresse &A047  *
                    ;*             in &09 umändern!!            *
                    ;***********************************************
A046 FE04           CP   &0A            ;Größer 9 ?
A048 D276A0         JP   NC,FEHLER      ;Ja, dann Fehler!
                    ;
                    ;***********************************************
                    ;*  ACHTUNG: Die interne Sektorbezeichnung  *
                    ;*  unterscheidet sich bei den Formaten!!!  *
                    ;*  Bitte die Adresse &A04C ändern:         *
                    ;*  FORMAT und FORMAT V :   &A04C = &40     *
                    ;*  FORMAT D            :   &A04C = &C0     *
                    ;*  FORMAT I            :   &A04C = &00     *
                    ;***********************************************
A04B CE40           ADC  A,&40          ;Interne Sektornummer
A04D 327FA0         LD   (SEKTOR),A     ;Sektor merken
A050 DD6601         LD   H,(IX+1)       ;Pufferadresse Highbyte
A053 DD6E00         LD   L,(IX+0)       ;Pufferadresse Lowbyte
A056 2280A0         LD   (PUFFER),HL    ;Pufferadresse merken
A059 C9             RET
                    ;
                    ;***********************************************
                    ;*  Diskettenbefehl ausführen (ROM-Aufruf)  *
                    ;***********************************************
A05A 2185A0   DISK  LD   HL,BEFEHL
A0ED CDD4BC         CALL &BCD4
A060 2282A0         LD   (ADRESS),HL
A063 79             LD   A,C
A064 3284A0         LD   (ROMSEL),A
A067 217DA0         LD   HL,DRIVE
A06A 5E             LD   E,(HL)         ;Drivenummer ins E-Reg.
```

```
A06B 23                 INC  HL
A06C 56                 LD   D,(HL)          ;Spur ins D-Reg.
A06D 23                 INC  HL
A06E 4E                 LD   C,(HL)          ;Sektor ins C-Reg.
A06F 2A80A0             LD   HL,(PUFFER)     ;Pufferadresse in HL-Re
A072 DF                 RST  &18
A073 82A0               DW   ADRESS
A075 C9                 RET
                        ;

                        ;*****************************************
                        ;* Fehlerroutine: Setzt &A086 auf &FF (255)
                        ;*****************************************
A076 E1        FEHLER   POP  HL              ;Stapel reorganisieren!!
A077 3EFF               LD   A,&FF
A079 3286A0             LD   (ERROUT),A
A07C C9                 RET                  ;Zurück ins BASIC!!!
                        ;
                        ;*****************************************
                        ;*   Speicherraum für Zwischenergebnisse
                        ;*****************************************
A07D           DRIVE    DS   &01
A07E           SPUR     DS   &01
A07F           SEKTOR   DS   &01
A080           PUFFER   DW   &0000
A082           ADRESS   DS   &02
A084           ROMSEL   DS   &01
A085           BEFEHL   DS   &01
A086           ERROUT   DS   &01
```

Testen Sie das Maschinenprogramm mit dem folgenden kleinen Beispiel:

```
 10 MEMORY &8FFF: MODE 2
 20 INPUT " Spur   :";spur
 30 INPUT " Sektor :";sektor
 40 CALL &A000,0,spur,sektor,&9000
 50 IF PEEK(&A086)=&FF THEN PRINT "Fehler !": GOTO 20
 60 FOR i=&9000 TO &9000+511
 70 w=PEEK(i)
 80 IF w<32 OR w>127 THEN w=46
 90 PRINT CHR$(w);
100 NEXT i
110 PRINT
120 GOTO 20
```

Das Testprogramm liest einen Block von der Diskette in den Puffer ab Adresse &9000 und zeigt anschließend den Block als ASCII-Dump an (alle ASCII-Zeichen kleiner als 32 und größer als 127 werden als Punkt angezeigt). Liegt der Wert für Spur und/oder Sektor außerhalb des zulässigen Bereichs, erfolgt eine Fehlermeldung (Zeile 50). Geben Sie einmal Spur 2 und Sektor 0 ein. Hier können Sie einen Teil der Directory erkennen.

Das Arbeiten mit diesen beiden neuen „Befehlen" ist in der Praxis unproblematisch. Besondere Vorsicht ist allerdings bei dem Zurückschreiben eines Blocks auf die Diskette geboten, da der alte Blockinhalt komplett überschrieben und damit zerstört wird.

Die Anwendungsmöglichkeiten der beiden Routinen sind sehr vielfältig. Naheliegend wäre z.B. ein Diskettenmonitor.

10.2 Der Diskettenmonitor

Um den Aufbau und die Funktion der Diskette(nstation) besser verstehen zu können, ist der „Diskettenmonitor" eine unerläßliche Hilfe. Der Anwendungsbereich eines Diskettenmonitors geht weit über den „Lerneffekt" hinaus: Es lassen sich gelöschte Dateien „retten", Programme gegen Überschreiben schützen oder sie in der Directory unsichtbar machen etc..

Die Funktion gliedert sich im wesentlichen in drei Teile:

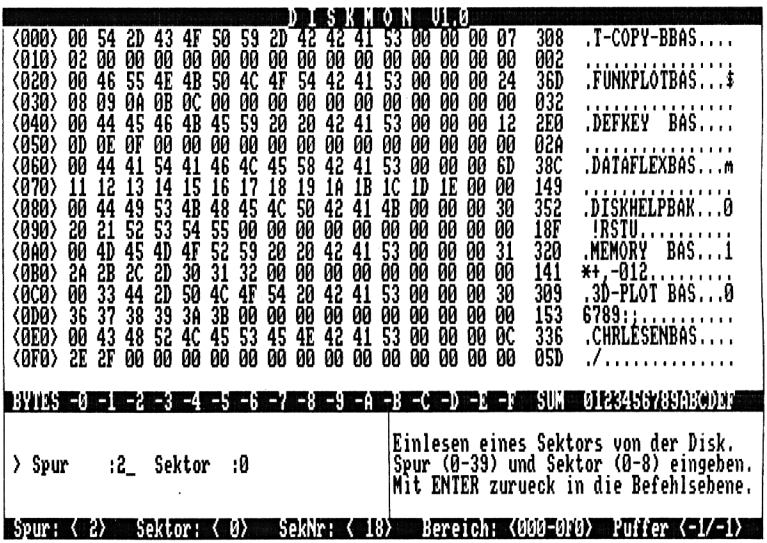
1. Sektor einlesen
2. Sektor bearbeiten (ansehen und ggf. ändern)
3. Sektor zurückschreiben

Das Einlesen und Zurückschreiben eines Sektors ist mit Hilfe des unter 10.1 beschriebenen Programms kein Problem mehr. Der Inhalt des eingelesenen Sektors ist nur noch in einer geeigneten Form auf den Bildschirm zu bringen.

Nicht ohne Grund hat sich bei der Darstellung von Speicherinhalten der sogenannte „HEX-DUMP" durchgesetzt. Der Vorteil besteht darin, daß jeder Wert zwischen 0 und 255 mit zwei hexadezimalen Zeichen darstellbar ist. Somit lassen sich möglichst viele Informationen gleichzeitig auf dem Bildschirm betrachten (Anm.: Auf das hexadezimale Zahlensystem wird in diesem Band nicht weiter eingegangen, da es ausführlich im Band 1, Kapitel 20.4 beschrieben ist. Schlagen Sie dort bitte gegebenenfalls nach.).

Da sich innerhalb eines Speicherbereichs (Sektor) auch Textinformationen (Texte, Anleitungen) befinden können, sollten diese Werte, falls es sich um „sinnvolle" ASCII-Werte handelt (32-127), noch zusätzlich als Zeichen ausgegeben werden (ASCII-DUMP). Der ASCII-DUMP ermöglicht ein schnelleres und leichteres Finden von bestimmten Informationen innerhalb des dargestellten Bereichs.

Das nachfolgende Programm „DISKMON" arbeitet sowohl mit dem HEX-DUMP als auch mit einem ASCII-DUMP. Der Bildschirm zeigt jeweils 256 Bytes des eingelesenen Sektors an. Damit möglichst einfach auf den gesamten Sektor zugegriffen werden kann, kann mit den Cursor-Tasten der Sektor vor- oder zurückgescrollt werden.



Das Programm gewährleistet ein einfaches und sicheres Arbeiten mit der Diskette. Besonders sei an dieser Stelle auf die COPY-Funktion hingewiesen: Mit der COPY-Funktion wird der aktuelle Sektor zusätzlich in einen geschützten Bereich des Computers kopiert (&9200-&93FF). Falls Sie den aktuell zu bearbeitenden Sektor aus irgendeinem Grund auf der Diskette „zerstören" (falsche Informationen zurückschreiben), haben Sie immer noch den alten Sektorinhalt als Kopie im Rechner. Sie sollten sich deshalb immer den Sektor, den Sie gerade bearbeiten, kopieren. Sicher ist sicher.

Cursor up	Scrollt den Bildschirm um eine Zeile (16 Bytes), falls möglich, nach oben.
Cursor down	Scrollt den Bildschirm um eine Zeile (16 Bytes), falls möglich, nach unten.
SHIFT/up	Zeigt die ersten 256 Bytes (&000 - &0FF) des aktuellen Sektors.
SHIFT/down	Zeigt die letzten 256 Bytes (&100 - &1FF) des aktuellen Sektors.
SHIFT/left	Lädt den Sektor, der sich vor dem aktuellen Sektor auf der Diskette befindet, in den Rechner ein.
SHIFT/right	Lädt den Sektor, der sich nach dem aktuellen Sektor auf der Diskette befindet, in den Rechner ein.
copy	Nach Betätigung der COPY-Taste (Cursorblock) kann der aktuelle Sektor (Arbeitspuffer) in den Hilfsspeicher (Hilfspuffer) oder der Hilfspuffer in den Arbeitspuffer kopiert werden. Ein leerer Hilfspuffer (– 1/ – 1) kann nicht in den Arbeitspuffer kopiert werden. Nach dem Kopieren des Arbeitspuffers in den Hilfspuffer zeigt die Pufferanzeige den Sektor an (Spur/Sektor), der sich im Hilfspuffer befindet. Wird der Hilfspuffer in den Arbeitspuffer kopiert, wird zusätzlich die Information, um welchen Sektor es sich handelt, in die Anzeige übernommen. Der Hilfspuffer bleibt dabei unverändert.
(A)endern	Der aktuelle Sektor kann geändert werden. Das Programm fragt nach der Startadresse (HEX), wo das Ändern beginnen soll. Ist die Startadresse eingegeben, scrollt der Bildschirm zunächst zu der Position, wo die zu ändernden Bytes sichtbar sind. Jetzt können Sie im Eingabefenster die Bytes ändern (HEX-Wert eingeben) oder mit der ENTER-Taste übernehmen. Mit der Eingabe von „/“ kann die Eingabe jederzeit abgebrochen werden.
(B)lock laden	Die Blocknummer (aus dem Directory-Eintrag zu entnehmen) gibt einen 1 kByte großen Bereich auf der Diskette an, wo bestimmte Informationen eines Programms zu finden sind (siehe 10.3). Der Befehl rechnet diese Information in eine Spur/Sektor Angabe um und lädt den entsprechenden Sektor.

(D)rucken	Druckt den aktuellen Sektor auf dem (eingeschalteten!) Drucker aus.
(H)ilfe	Zeigt alle möglichen Befehle an.
(L)aden	Lädt einen bestimmten Sektor von der Diskette in den Arbeitspuffer (Spur 0-39, Sektor 0-8). Der aktuelle Inhalt des Arbeitspuffers wird damit überschrieben.
(S)chreiben	Schreibt den aktuellen Arbeitspuffer in einen bestimmten Sektor zurück (Spur 0-39, Sektor 0-8). Seien Sie beim Zurückschreiben besonders vorsichtig!
(Q)uit	Programmende

(ACHTUNG: Geben Sie die Programmzeilen 1050,4020 und 4030 erst dann ein, wenn Ihr Programm ohne Fehler läuft! Die Fehlerabfrage ist für die Umrechnung hexadezimal nach dezimal notwendig!)

```
1000 REM ****************************************** M.W. Thoma **
1010 REM *                                                      *
1020 REM *          Disketten-Monitor DISKMON V1.0B             *
1030 REM *                                                      *
1040 REM ********************************************************
1050 ON ERROR GOTO 4020
1060 MODE 2:INK 0,0:INK 1,26:BORDER 0
1070 DRAW 0,399,1:MOVE 639,0:DRAW 639,399
1080 MOVE 324,0:DRAW 324,106,1
1090 LOCATE 1,19:PRINT CHR$(24);" BYTES -0 -1 -2 -3 -4 -5 ";
1100 PRINT "-6 -7 -8 -9 -A -B -C -D -E -F  SUM  0123456789";
1110 PRINT "ABCDEF   ";CHR$(24)
1120 WINDOW#0,2,40,21,24
1130 WINDOW#1,1,80,1,1
1140 WINDOW#2,2,79,2,17
1150 WINDOW#3,1,80,25,25
1160 WINDOW#4,42,79,21,24
1170 PEN#1,0:PAPER#1,1:CLS#1
1180 PRINT#1,STRING$(30,"_");"D I S K M O N  V1.0";
1190 PRINT#1,STRING$(31,"_");
1200 PEN#3,0:PAPER#3,1:CLS#3
1210 PRINT#3," Spur: <##>   Sektor: <##>   SekNr: <###>   ";
1220 PRINT#3,"Bereich: <###-###>  Puffer <-1/-1>"
1230 MEMORY &8FFF
1240 GOSUB 4180
1250 sca=0:sce=0:leer=0:sp1=-1:se1=-1:dr=2
1260 '
```

```
1270 REM ************************************************************
1280 REM *                                                          *
1290 REM *                          M E N U E                       *
1300 REM *                                                          *
1310 REM ************************************************************
1320 tx=1:GOSUB 3690
1330 CLS:PRINT ">Kommando :"
1340 in$=INKEY$:IF in$="" THEN 1340
1350 IF in$=CHR$(&F0) THEN GOSUB 2000:GOTO 1340
1360 IF in$=CHR$(&F1) THEN GOSUB 2000:GOTO 1340
1370 IF in$=CHR$(&F4) THEN GOSUB 2330:GOTO 1340
1380 IF in$=CHR$(&F5) THEN GOSUB 2330:GOTO 1340
1390 IF in$=CHR$(&F6) THEN GOSUB 2400:GOTO 1320
1400 IF in$=CHR$(&F7) THEN GOSUB 2400:GOTO 1320
1410 IF in$=CHR$(&E0) THEN GOSUB 2950:GOTO 1320
1420 IF UPPER$(in$)="L" THEN GOSUB 2580:GOTO 1320
1430 IF UPPER$(in$)="B" THEN GOSUB 3220:GOTO 1320
1440 IF UPPER$(in$)="A" THEN GOSUB 1510:GOTO 1320
1450 IF UPPER$(in$)="S" THEN GOSUB 1780:GOTO 1320
1460 IF UPPER$(in$)="H" THEN GOSUB 3350:GOTO 1320
1470 IF UPPER$(in$)="D" THEN GOSUB 4050:GOTO 1320
1480 IF UPPER$(in$)="Q" THEN MODE 2:PRINT "Tschuess":END
1490 GOTO 1340
1500 '
1510 REM ************************************************************
1520 REM *                                                          *
1530 REM * (A)          Aktuellen Block aendern                     *
1540 REM *                                                          *
1550 REM ************************************************************
1560 IF leer=0 THEN PRINT CHR$(7):RETURN
1570 tx=6:GOSUB 3690:CLS
1580 INPUT "> Bitte Startadresse in HEX :";i1$
1590 ad=VAL("&"+i1$):IF i1$="" THEN PRINT CHR$(7):RETURN
1600 IF ad<0 OR ad>&1FE THEN PRINT CHR$(7):RETURN
1610 CLS:ad1=ad
1620 IF ad<sca*16 THEN GOSUB 2150:GOTO 1620
1630 IF ad>sce*16 AND ad<496 THEN GOSUB 2080:GOTO 1630
1640 WHILE ad<&1FF
1650 PRINT"<";HEX$(ad,3);">  ";HEX$(PEEK(ad+&9000),2),,
1660 PRINT"[";BIN$(PEEK(ad+&9000),8);"]";
1670 LOCATE 15,VPOS(#0):LINE INPUT;i1$
1680 IF i1$="\" THEN ad=&1FF:GOTO 1740
1690 IF i1$="" THEN PRINT HEX$(PEEK(ad+&9000),2):GOTO 1720
1700 nw=VAL("&"+i1$):IF i1$="" THEN ad=&1FF:GOTO 1740
1710 POKE ad+&9000,nw:PRINT
1720 ad=ad+1
1730 IF ad>sce*16 AND ad<496 THEN GOSUB 2080
1740 WEND
1750 IF ad1<&100 THEN GOSUB 2380:ELSE 2370
```

```
1760 RETURN
1770 '
1780 REM ************************************************
1790 REM *                                              *
1800 REM * <S>    Aktuellen Block zurueckschreiben      *
1810 REM *                                              *
1820 REM ************************************************
1830 IF leer=0 THEN PRINT CHR$(7):RETURN
1840 CLS:tx=7:GOSUB 3690
1850 PRINT
1860 PRINT "> Spur    :__";
1870 PRINT "  Sektor  :_"
1880 LOCATE 12,2:LINE INPUT i1$:LOCATE 25,2:LINE INPUT i2$
1890 IF i1$="" OR i2$="" THEN PRINT CHR$(7):RETURN
1900 i1=VAL(i1$):i2=VAL(i2$)
1910 IF i1<0 OR i1>39 THEN PRINT CHR$(7);:RETURN
1920 IF i2<0 OR i2>8  THEN PRINT CHR$(7);:RETURN
1930 PRINT "> Sind Sie wirklich sicher (J/N) ?"
1940 i1$=INKEY$:IF i1$="" THEN 1940
1950 IF UPPER$(i1$)<>"J" THEN PRINT CHR$(7):RETURN
1960 sp=i1:se=i2:leer=1:bl=sp*9+se
1970 CALL &A00C,0,sp,se,&9000
1980 RETURN
1990 '
2000 REM ************************************************
2010 REM *                                              *
2020 REM *          Scrolling des Ausgabefensters       *
2030 REM *                                              *
2040 REM ************************************************
2050 IF leer=0 THEN PRINT CHR$(7);:RETURN
2060 IF in$=CHR$(&F0) THEN 2140
2070 IF sce=31 THEN PRINT CHR$(7);:RETURN
2080 LOCATE#2,1,17
2090 sce=sce+1:sca=sca+1
2100 ada=&9000+sce*16
2110 GOSUB 3640:GOSUB 2210
2120 RETURN
2130 '
2140 IF sca=0 THEN PRINT CHR$(7);:RETURN
2150 LOCATE#2,1,1:PRINT#2,CHR$(11);
2160 sca=sca-1:sce=sce-1
2170 ada=&9000+sca*16
2180 GOSUB 3640:GOSUB 2210
2190 RETURN
2200 '
2210 PRINT#2,"<";HEX$(ada-&9000,3);"> ";
2220 z1=0:summe=0:dump$=""
2230 WHILE z1<16
2240 wert=PEEK(ada)
```

```
2250 PRINT#2,HEX$(wert,2);" ";
2260 summe=summe+wert
2270 IF wert<32 OR wert>127 THEN wert=46
2280 dump$=dump$+CHR$(wert)
2290 ada=ada+1:z1=z1+1
2300 WEND
2310 PRINT#2," ";HEX$(summe,3);"  ";dump$
2320 RETURN
2330 '
2340 IF leer=0 THEN PRINT CHR$(7):RETURN
2350 CLS#2
2360 IF in$=CHR$(&F4) THEN 2380
2370 sca=16:sce=31:GOSUB 3640:ada=&9100:GOSUB 2760:RETURN
2380 sca=0:sce=15:GOSUB 3640:ada=&9000:GOSUB 2760:RETURN
2390 '
2400 REM ******************************************************
2410 REM *                                                    *
2420 REM *              Block vor oder zurueck                *
2430 REM *                                                    *
2440 REM ******************************************************
2450 tx=3:GOSUB 3690
2460 IF leer=0 THEN PRINT CHR$(7):RETURN
2470 IF in$=CHR$(&F6) THEN 2530
2480 IF bl=359 THEN PRINT CHR$(7):RETURN
2490 bl=bl+1:sp=INT(bl/9):se=bl MOD 9
2500 GOSUB 2730
2510 RETURN
2520 '
2530 IF bl=0 THEN PRINT CHR$(7):RETURN
2540 bl=bl-1:sp=INT(bl/9):se=bl MOD 9
2550 GOSUB 2730
2560 RETURN
2570 '
2580 REM ******************************************************
2590 REM *                                                    *
2600 REM * <L>          Block einlesen                        *
2610 REM *                                                    *
2620 REM ******************************************************
2630 CLS:tx=2:GOSUB 3690
2640 PRINT
2650 PRINT "> Spur    :__";
2660 PRINT "  Sektor  :_"
2670 LOCATE 12,2:LINE INPUT i1$:LOCATE 25,2:LINE INPUT i2$
2680 IF i1$="" OR i2$="" THEN PRINT CHR$(7):RETURN
2690 i1=VAL(i1$):i2=VAL(i2$)
2700 IF i1<0 OR i1>39 THEN PRINT CHR$(7);:RETURN
2710 IF i2<0 OR i2>8  THEN PRINT CHR$(7);:RETURN
2720 sp=i1:se=i2:leer=1:bl=sp*9+se
2730 CALL &A000,0,sp,se,&9000
```

```
2740 ada=&9000:sca=0:sce=15
2750 GOSUB 3560
2760 CLS#2
2770 z1=0:z2=0:summe=0:dump$=""
2780 WHILE z1<16
2790 z2=0
2800 PRINT#dr,"<";HEX$(ada-&9000,3);"> ";
2810 WHILE z2<16
2820 wert=PEEK(ada)
2830 PRINT#dr,HEX$(wert,2);" ";
2840 summe=summe+wert
2850 IF wert<32 OR wert>127 THEN wert=46
2860 dump$=dump$+CHR$(wert)
2870 ada=ada+1:z2=z2+1
2880 WEND
2890 PRINT#dr," ";HEX$(summe,3);"  ";dump$
2900 dump$="":summe=0
2910 z1=z1+1
2920 WEND
2930 RETURN
2940 '
2950 REM ************************************************************
2960 REM *                                                          *
2970 REM *                     Block kopieren                       *
2980 REM *                                                          *
2990 REM ************************************************************
3000 tx=4:GOSUB 3690
3010 IF leer=0 THEN PRINT CHR$(7):RETURN
3020 CLS:INPUT"> Bitte <A> oder <H> :";i1$
3030 IF UPPER$(i1$)="A" THEN 3140
3040 IF UPPER$(i1$) <>"H" THEN PRINT CHR$(7):RETURN
3050 PRINT
3060 IF se1=-1 THEN PRINT CHR$(7):RETURN
3070 PRINT "Lege Hilfspuffer in den Arbeitspuffer"
3080 sp=sp1:se=se1:bl=sp*9+se
3090 FOR i=&9200 TO &93FF
3100 POKE i-&200,PEEK(i)
3110 NEXT i
3120 GOSUB 2740:RETURN
3130 '
3140 PRINT
3150 PRINT "Lege Arbeitspuffer in den Hilfspuffer"
3160 sp1=sp:se1=se:GOSUB 3560
3170 FOR i=&9000 TO &91FF
3180 POKE i+&200,PEEK(i)
3190 NEXT i
3200 RETURN
3210 '
```

```
3220 REM ************************************************************
3230 REM *                                                          *
3240 REM * <B>          Laden nach der Blocknummer                  *
3250 REM *                                                          *
3260 REM ************************************************************
3270 tx=5:GOSUB 3690:i1$=""
3280 CLS: INPUT "> Blocknummer in HEX :";i1$
3290 bn=VAL("&"+i1$)
3300 IF bn<2 OR bn>169 THEN PRINT CHR$(7):RETURN
3310 i1=INT((bn*2+18)/9):i2=(bn*2+18) MOD 9
3320 GOSUB 2720
3330 RETURN
3340 '
3350 REM ************************************************************
3360 REM *                                                          *
3370 REM * <H>                 Hilfe                                *
3380 REM *                                                          *
3390 REM ************************************************************
3400 CLS
3410 PRINT"<L> Block laden  <S> Block schreiben"
3420 PRINT"<B> Blocknummer  <A> Block aendern";
3430 PRINT"<H> Hilfe        <Q> Programmende"
3440 PRINT"      <D> Block ausdrucken"
3450 CLS#4
3460 PRINT#4,"<";CHR$(240);"> Zeile vor   ";
3470 PRINT#4,"<";CHR$(241);"> Zeile zurueck"
3480 PRINT#4,"<S/";CHR$(240);"> Seite vor  ";
3490 PRINT#4,"<S/";CHR$(241);"> Seite zurueck"
3500 PRINT#4,"<S/";CHR$(242);"> Block vor  ";
3510 PRINT#4,"<S/";CHR$(243);"> Block zurueck";
3520 PRINT#4,"     <COPY> Kopieren des Blocks"
3530 i1$=INKEY$:IF i1$="" THEN 3530
3540 RETURN
3550 '
3560 REM ************************************************************
3570 REM *                                                          *
3580 REM *               Info-Zeile ausgeben                        *
3590 REM *                                                          *
3600 REM ************************************************************
3610 LOCATE#3,9,1:PRINT#3,USING "##";sp;
3620 LOCATE#3,24,1:PRINT#3,USING "##";se;
3630 LOCATE#3,38,1:PRINT#3,USING "###";bl
3640 LOCATE#3,55,1:PRINT#3,HEX$(sca*16,3);"-";HEX$(sce*16,3)
3650 LOCATE#3,73,1:PRINT#3,USING"##";sp1;
3660 LOCATE#3,76,1:PRINT#3,USING"##";se1;
3670 RETURN
3680 '
```

```
3690 REM ************************************************************
3700 REM *                                                          *
3710 REM *              Informationen ausgeben                      *
3720 REM *                                                          *
3730 REM ************************************************************
3740 CLS#4
3750 ON tx GOTO  3760,3790,3830,3860,3900,3940,3980
3760 PRINT#4,"  Bitte geben Sie einen Befehl ein!
3770 PRINT#4,"     Hilfe erhalten Sie mit <H> !
3780 RETURN
3790 PRINT#4,"Einlesen eines Sektors von der Disk."
3800 PRINT#4,"Spur (0-39) und Sektor (0-8) eingeben.";
3810 PRINT#4,"Mit ENTER zurueck in die Befehlsebene.";
3820 RETURN
3830 PRINT#4
3840 PRINT#4,"Lese naechsten oder vorigen Block ein."
3850 RETURN
3860 PRINT#4,"A = kopiert Arbeits- i.d. Hilfspuffer"
3870 PRINT#4,"H = kopiert Hilfs- i.d. Arbeitspuffer"
3880 PRINT#4,"ACHTUNG: AlterZustand wird zerstoert!"
3890 RETURN
3900 PRINT#4,"Bitte die Blocknummer des gewuenschten";
3910 PRINT#4,"Blocks im hexadezimal eingeben. Der"
3920 PRINT#4,"Block wird errechnet und eingelesen."
3930 RETURN
3940 PRINT#4,"Aendern des aktuellen Blocks.Bitte die";
3950 PRINT#4,"die Startadr. angeben. Mit ENTER wird"
3960 PRINT#4,"alter Wert uebernommen. \ Eingabeende!";
3970 RETURN
3980 PRINT#4,"Aktuellen Block auf Disk schreiben."
3990 PRINT#4,"Bitte ueberlegen Sie diesen Schritt"
4000 PRINT#4,"genau. Der alte Block wird zerstoert!!";
4010 RETURN
4020 '
4030 SOUND 1,400,10:i1$="":RESUME NEXT
4040 '
4050 REM ************************************************************
4060 REM *                                                          *
4070 REM * <D>              Druckerausgabe                          *
4080 REM *                                                          *
4090 REM ************************************************************
4100 IF leer=0 THEN PRINT CHR$(7):RETURN
4110 CLS:dr=8
4120 PRINT:PRINT ">Druckausgabe des aktuellen Blocks"
4130 PRINT#dr,"Spur :";sp,"Sektor :";se,"Block :";bl
4140 PRINT#dr
4150 ada=&9000:GOSUB 2770:GOSUB 2770
4160 dr=2:RETURN
4170 '
```

```
4180 REM ************************************ M.W. Thoma **
4190 REM *                                                *
4200 REM *         Direktzugriff auf die Diskette         *
4210 REM *                                                *
4220 REM *     Lesen: CALL &A000,DRIVE,SPUR,SEKTOR,PUFFER  *
4230 REM * Schreiben: CALL &A00C,DRIVE,SPUR,SEKTOR,PUFFER  *
4240 REM *                                                *
4250 REM ************************************************
4260 summe=0:FOR i=&A000 TO &A086
4270 READ a$:a=VAL("&"+a$):POKE i,a:summe=summe+a
4280 '
4290 NEXT i
4300 IF summe=16125 THEN RETURN
4310 PRINT "Fehler in den DATA-Zeilen":END
4320 '
4330 DATA CD,18,A0,3E,84,32,85,A0,CD,5A,A0,C9,CD,18,A0,3E
4340 DATA 85,32,85,A0,CD,5A,A0,C9,F5,3E,00,32,86,A0,F1,FE
4350 DATA 04,C2,76,A0,DD,7E,07,DD,7E,06,FE,02,D2,76,A0,32
4360 DATA 7D,A0,DD,7E,05,DD,7E,04,FE,28,D2,76,A0,32,7E,A0
4370 DATA DD,7E,03,DD,7E,02,FE,0A,D2,76,A0,CE,40,32,7F,A0
4380 DATA DD,66,01,DD,6E,00,22,80,A0,C9,21,85,A0,CD,D4,BC
4390 DATA 22,82,A0,79,32,84,A0,21,7D,A0,5E,23,56,23,4E,2A
4400 DATA 80,A0,DF,82,A0,C9,E1,3E,FF,32,86,A0,C9,00,00,00
4410 DATA 00,00,00,00,00,00,00,00
4420 REM ************************************************
```

10.3 Der Diskette aufs Byte geschaut

Der Diskettenmonitor ermöglicht es, den Aufbau der Diskette genauer zu betrachten. Dieses Kapitel soll Ihnen einige Grundlagen über den Diskettenaufbau vermitteln. Zunächst müssen allerdings einige verwirrende Bezeichnungen erklärt werden.

Als physikalische Speichereinheit dient auf der Diskette ein 512 Bytes großer Sektor. Insgesamt besteht die Diskette aus 360 Sektoren (40 Spuren * 9 Sektoren). Jeder Sektor ist weiterhin in 4 (logische) Einheiten a 128 Bytes gegliedert. Diese 128 Byte-Einheiten nennen sich „Records" und bilden die kleinste Speichereinheit auf der Diskette. Ist auch nur ein Byte auf der Diskette zu speichern, so wird zumindest ein Record mit Daten aufgefüllt!

Die Speicherung von Programmen (Dateien) findet immer in 1 kByte großen „Blöcken" statt. Ein Block besteht damit aus 2 Sektoren und somit aus 8 Records. Diese Blöcke (1024 Bytes) bilden die kleinste Speichereinheit für ein Programm (Datei)! D.h. ein Programm bis zu einem (1) kByte Länge belegt

einen Block, mit 1025-2048 Bytes zwei Blöcke etc.. Anhand dieser Blocknummer, die in der Directory verzeichnet ist, kann das DOS feststellen, wo sich das Programm auf der Diskette befindet.

Bei einer „normal" formatierten Diskette (FORMAT) befinden sich auf den ersten beiden Spuren (Spur 0 und 1) eine Kopie des CP/M-Betriebssystems (alle weiteren Betrachtungen beziehen sich auf dieses Format!!).

```
Spur 0 / Sektor 0   : Boot Sektor
Spur 0 / Sektor 1   : Konfigurations-Sektor
Spur 0 / Sektor 2-6 : Unbenutzt
Spur 0 / Sektor 7-8 : Anfang CCP
Spur 1 / Sektor 0-8 : Ende CCP und BDOS
```

Diese beiden Spuren sind grundsätzlich reserviert und vor dem Überschreiben durch Programme (Dateien) geschützt. Lesen Sie einmal den Sektor Spur 0/Sektor 7 ein und Sie werden hier u.a. die Copyright-Meldung des CP/M-Betriebssystems finden. Damit stehen für die Datensicherung nur noch 38 Spuren zur Verfügung (38*9 = 342 Sektoren).

Das Inhaltsverzeichnis (Directory) belegt die ersten 4 Sektoren der Spur 2 (Spur 2/Sektor 0-3). Auch diese Sektoren können für die Sicherung von Programmen nicht genutzt werden. Die maximale Speicherkapazität einer formatierten Diskette beträgt

338 Sektoren * 512 Bytes = 173.056 Bytes = 169 kByte

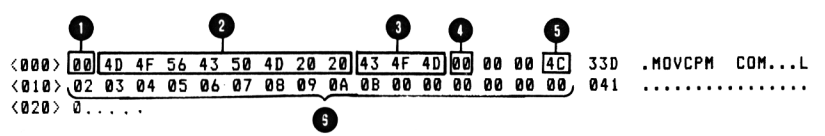
10.4 Directory-Aufbau

Für die weiteren Untersuchungen legen Sie bitte eine Kopie der Systemdiskette in das Laufwerk (A) und laden sich den ersten Direktory-Sektor (Spur 2/Sektor 0) ein. Auf dem Bildschirm erhalten Sie dann folgenden Ausdruck (hier ist der gesamte Block ausgedruckt):

```
Spur : 2    Sektor : 0    Block : 18

<000> 00 4D 4F 56 43 50 4D 20 20 43 4F 4D 00 00 00 4C  33D  .MOVCPM  COM...L
<010> 02 03 04 05 06 07 08 09 0A 0B 00 00 00 00 00 00  041  ................
<020> 00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A  2A2  .PIP     COM...:
<030> 0C 0D 0E 0F 10 11 12 13 00 00 00 00 00 00 00 00  07C  ................
<040> 00 53 55 42 4D 49 54 20 20 43 4F 4D 00 00 00 0A  2FD  .SUBMIT  COM....
<050> 14 15 00 00 00 00 00 00 00 00 00 00 00 00 00 00  029  ................
<060> 00 58 53 55 42 20 20 20 20 43 4F 4D 00 00 00 06  2A7  .XSUB    COM....
<070> 16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  016  ................
<080> 00 45 44 20 20 20 20 20 20 43 4F 4D 00 00 00 34  25C  .ED      COM...4
<090> 17 18 19 1A 1B 1C 1D 00 00 00 00 00 00 00 00 00  0B6  ................
<0A0> 00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40  2A0  .ASM     COM...@
<0B0> 1E 1F 20 21 22 23 24 25 00 00 00 00 00 00 00 00  10C  .. !"#$%........
<0C0> 00 44 44 54 20 20 20 20 20 43 4F 4D 00 00 00 26  281  .DDT     COM...&
<0D0> 26.27 28 29 2A 00 00 00 00 00 00 00 00 00 00 00  0C8  &'()*...........
<0E0> 00 4C 4F 41 44 20 20 20 20 43 4F 4D 00 00 00 0E  28D  .LOAD    COM....
<0F0> 2B 2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00  057  +,..............
<100> 00 53 54 41 54 20 20 20 20 43 4F 4D 00 00 00 29  2C4  .STAT    COM...)
<110> 2D 2E 2F 30 31 32 00 00 00 00 00 00 00 00 00 00  11D  -./012..........
<120> 00 44 55 4D 50 20 20 20 20 43 4F 4D 00 00 00 04  299  .DUMP    COM....
<130> 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  033  3...............
<140> 00 44 55 4D 50 20 20 20 20 41 53 4D 00 00 00 21  2B8  .DUMP    ASM...!
<150> 34 35 36 37 38 00 00 00 00 00 00 00 00 00 00 00  10E  45678...........
<160> 00 41 4D 53 44 4F 53 20 20 43 4F 4D 00 00 00 02  2E8  .AMSDOS  COM....
<170> 39 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  039  9...............
<180> 00 46 49 4C 45 43 4F 50 59 43 4F 4D 00 00 00 16  350  .FILECOPYCOM....
<190> 3A 3B 3C 00 00 00 00 00 00 00 00 00 00 00 00 00  0B1  :;<.............
<1A0> 00 53 59 53 47 45 4E 20 20 43 4F 4D 00 00 00 0C  304  .SYSGEN  COM....
<1B0> 3D 3E 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07B  =>..............
<1C0> 00 42 4F 4F 54 47 45 4E 20 43 4F 4D 00 00 00 0A  317  .BOOTGEN COM....
<1D0> 3F 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00  07F  ?@..............
<1E0> 00 43 4F 50 59 44 49 53 43 43 4F 4D 00 00 00 15  352  .COPYDISCCOM....
<1F0> 41 42 43 00 00 00 00 00 00 00 00 00 00 00 00 00  0C6  ABC.............
```

Anhand des ASCII-Dumps kann man ganz deutlich die Programmnamen und Typenbezeichnungen erkennen. Jeder Directory-Eintrag für ein Programm ist 32 Bytes groß. Der Eintrag für das Programm „MOVCPM" belegt z.B. die Bytes &000 bis &01F des Sektors (eingerahmt). Es passen jeweils 16 Directory-Einträge in einen Directory-Sektor und die gesamte Directory kann damit 64 Einträge (4*16) aufnehmen.

Innerhalb des 32 Bytes großen Eintrags befinden sich alle wichtigen Informationen, um das entsprechende Programm auf der Diskette zu finden. Da alle Einträge identisch aufgebaut sind, kann sich die Beschreibung auf einen Eintrag beschränken. Sehen Sie sich den Eintrag für das Programm „MOVCPM" bitte genauer an:



(1) Das erste Byte (&00) bestimmt, in welcher Benutzerebene (USER-Ebene) das Programm gesichert wurde (&00 bis &0F). Hier befindet sich der Wert &00 im Byte &00, also wurde das Programm „MOVCPM" in der USER-Ebene 0 (&00) abgelegt. Anhand dieses Bytes prüft das DOS, ob die Ebene mit der aktuellen USER-Ebene übereinstimmt (Der Befehl CAT zeigt z.B nur die Programme an, die auch in der aktuellen USER-Ebene abgelegt sind. Siehe 2.4).

Doch dieses Byte hat noch eine andere Funktion: Wird das Programm gelöscht, erhält das Byte den Wert &E5! Immer dann, wenn dieses Byte den Wert &E5 enthält, betrachtet das DOS diesen Eintrag als nicht vorhanden und übergeht ihn. Beim Löschen wird tatsächlich nur das erste Byte verändert. Haben Sie ein Programm versehentlich gelöscht, so brauchen Sie nur das erste Byte wieder mit dem Wert &00 (bzw. USER-Nummer) zu laden.

(2) Die folgenden 8 Bytes (&01 bis &08) geben den Filenamen in Großbuchstaben wieder (ggf. mit Leerzeichen (&20) aufgefüllt).

(3) In den Bytes &09 bis &0B befindet sich die 3-stellige Typenkennzeichnung des Programms (ebenfalls Großbuchstaben). Innerhalb der ersten beiden Bytes (&09 und &0A) der Typenkennzeichnung befinden sich allerdings noch weitere Informationen. Jeweils das Bit 7 der beiden Bytes bilden das „Attribut" des Programms:

Byte &09 : Bit 7 = 0 (0xxx xxxx)
Programm kann gelöscht werden

Bit 7 = 1 (1xxx xxxx)
Programm kann weder gelöscht noch umbenannt werden (READ ONLY). In der Directory wird nach der Typangabe ein zusätzlicher Stern „*" ausgegeben.

Byte &0A : Bit 7 = 0 (0xxx xxxx)
Das Programm wird in der Directory (|DIR oder CAT) geführt (sichtbar).

Bit 7 = 1 (1xxx xxxx)
Programm wird nicht in der Directory geführt. Obwohl das Programm vorhanden ist, wird es beim |DIR oder CAT Befehl nicht mit angezeigt.

Um z.B. ein Programm vor dem Überschreiben zu schützen, brauchen Sie nur das Bit 7 des Bytes &09 zu setzen (zum alten Wert &80 hinzuaddieren) und den Sektor zurückzuschreiben.

(4) Das Byte &0C hat eine besondere Funktion (Extent), die im Punkt 6 näher beschrieben wird.

(5) In dem Byte &0F steht die Länge des Programms in der Maßeinheit „Record" (128 Bytes). Beim Programm „MOVCPM" ist hier ein &4C zu finden (&4C = 76 dezimal). Das bedeutet, das Programm belegt

76 Records * 128 Bytes = 9728 Bytes

(6) Die Bytes &10 bis &1F kennzeichnen, wo das Programm auf der Diskette zu finden ist. Hier taucht auch die Größenordnung „Block" erstmals auf. Die hier folgenden Werte geben jeweils die Blocknummer an, in welcher die Programmdaten liegen. „MOVCPM" belegt die Blöcke &02, &03 ..... bis &0B. Wird auf ein Nullbyte (&00) getroffen, endet das Programm. Aus den Blocknummern läßt sich die Spur und der Sektor errechnen, wo der Block auf der Diskette beginnt.

Geht man davon aus, daß die Diskette erst in der Spur 2 beginnt (Spur 0 und 1 ist für das CP/M reserviert), so befindet sich der erste Block (Block 0!!) in den Sektoren Spur 2/Sektor 0 und 1.

Block 1 belegt dann die Sektoren Spur 2/Sektor 2 und 3, Block 2 die Sektoren Spur 2/Sektor 4 und 5 usw..

Mit den beiden Gleichungen

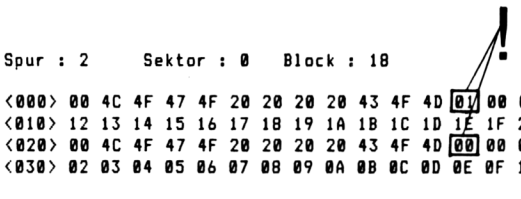
Spur = INT((blocknummer*2+18)/9)
Sektor = (blocknummer*2+18) MOD 9

lassen sich aus einer beliebigen Blocknummer die Spur und der Sektor errechnen (Mit dem Diskettenmonitor ist es möglich, direkt einen Block einzuladen – B –, siehe 10.2).

In einen Directory-Eintrag können maximal 16 Blocknummern aufgenommen werden (Bytes &10 bis &1F). D.h. ein Programm kann maximal 16 kByte lang sein !!!! Doch was passiert, wenn ein Programm mehr Speicherplatz benötigt als 16 kByte? Dann werden weitere Directory-Einträge mit demselben Namen

(und USER-Nummer) angelegt. (Einen 16 Byte großen Eintrag nennt man auch „Extent" – Ausdehnung –). Damit das DOS erkennen kann, daß dieser Eintrag zu einem schon bestehenden Eintrag gehört, ist dieses im Byte &0C (4) des jeweiligen Eintrags vermerkt. Die zusammengehörigen Einträge sind im Byte &0C von Null an durchnumeriert. Immer dann, wenn das Byte &0F (Recordanzahl) den Wert &80 (128, maximale Recordanzahl pro Eintrag) enthält, sucht das DOS einen Eintrag mit der nächsthöheren Extent-Nummer.

Sehen Sie sich dazu einmal die Rückseite der Systemdiskette an. Das Programm „LOGO" ist 32 kByte lang und belegt somit zwei Einträge:



10.5 Der Programmkopf

Aus der Directory können Sie nun entnehmen, wo ein bestimmtes Programm auf der Diskette abgelegt ist. Um die Speicherung des eigentlichen Programms genauer zu untersuchen, sichern Sie bitte das folgende Programm einmal „normal" und dann als ASCII-Datei auf einer Diskette ab.

```
10 REM **** Testprogramm ****
20 PRINT "Hallo"
30 END

SAVE ''TEST.BAS''
SAVE ''TEST.ASC'',A
```

Aus dem Directory-Sektor können Sie die Blocknummern entnehmen, wo das Programm abgelegt ist. Laden Sie sich bitte zunächst den Block ein, in dem das Programm „TEST.BAS" abgelegt ist (Mit dem Befehl „B" des Diskettenmonitors).

```
Spur : 2     Sektor : 4   Block : 22

<000> 00 54 45 53 54 20 20 20 20 42 41 53 00 00 00 00  296  .TEST    BAS....
<010> 00 00 00 00 00 70 01 00 33 00 00 00 00 00 00 00  0A4  .....p..3.......
<020> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  000  ................
<030> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  000  ................
<040> 33 00 00 6D 03 1D 00 0A 00 C5 20 2A 2A 2A 2A 20  277  3..m...... ****
<050> 54 65 73 74 70 72 6F 67 72 61 6D 6D 20 2A 2A 2A  5A3  Testprogramm ***
<060> 2A 00 0E 00 14 00 BF 20 22 48 61 6C 6C 6F 22 00  35F  *...... "Hallo".
<070> 06 00 1E 00 98 00 00 00 1A 2C 30 30 00 3A 00 44  1E0  .........,00.:.D
<080> 1D 00 0A 00 C5 20 2A 2A 2A 2A 20 54 65 73 74 70  3E4  ..... **** Testp
<090> 72 6F 67 72 61 6D 6D 20 2A 2A 2A 2A 00 0E 00 14  3DF  rogramm ****....
<0A0> 00 BF 20 22 48 61 6C 6C 6F 22 00 06 00 1E 00 98  3CF  .. "Hallo".......
<0B0> 00 00 00 1A 2C 30 30 00 3A 00 44 11 C5 20 2A 2A  26E  ....,00.:.D.. **
<0C0> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<0D0> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<0E0> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<0F0> 2A 00 00 00 1A 00 00 00 1A 24 27 00 00 00 1A 50  113  *........$'....P
<100> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<110> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<120> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<130> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
```

In der Abbildung sehen Sie den Ausdruck des Sektors, in dem sich das Programm „TEST.BAS“ befindet. Einige Bereiche und Bytes sind hier schon markiert worden, damit das Zahlen-Wirrwarr etwas überschaubarer ist.

Als erstes fällt ins Auge, daß ab der Adresse &100 alle Bytes den Wert &E5 enthalten. Benutzt werden von dem Programm nur die ersten 256 Bytes, also 2 Records (Beim Formatieren der Diskette wird sie komplett mit dem Wert &E5 beschrieben). Weiterhin steht in den Bytes &001 bis &00B wieder der Programmname und die Typenkennzeichnung. Dieser Bereich kann damit nicht zum eigentlichen Programm gehören! Es ist der sogenannte Programmheader oder Programmkopf. Der Programmkopf belegt den gesamten 1. Record (&000 - &7F).

Das eigentliche Programm beginnt erst mit dem Byte &080 (2.Record). Sehen Sie sich diesen Bereich genauer an. Hier liegen die BASIC-Befehle (verschlüsselt als Token), die Zeilennummern (Low- und Highbyte), die Länge einer Programmzeile und die ASCII-Zeichen (Text), die besonders gut mit dem ASCII-DUMP zu sehen sind. Auffällig sind die drei Null-Bytes in den Bytes &0B0 bis &0B2. An dieser Stelle ist das Programm tatsächlich zu Ende (als Endmarkierung wird die Programmzeilenlänge auf den Wert &0000 gelegt). Nach den drei Null-Bytes folgt noch der Wert &1A, der normalerweise das Ende eines Textes kennzeichnet (EOF = End Of File). Die restlichen Bytes &0B4 bis &0FF füllen den benutzten Record mit beliebigen Werten auf. Das Programm belegt damit nur die Bytes &080 bis &0B3 (&33 Bytes lang).

Der Programmkopf gibt neben dem Programmnamen (Bytes &001-&008) und der Typenbezeichnung (Bytes &009-&00B) noch einige andere interessante Informationen preis.

Byte &000	:	USER-Nummer (&00-&0F)
Byte &001-&008	:	Programmname (ggf. mit Leerzeichen (&20) gefüllt)
Byte &009-&00B	:	Typenkennzeichnung
Byte &013	:	Kennzeichnung, wie das Programm gesichert wurde: = &00 : Normal geSAVEt = &01 : Geschützt geSAVEt (Protected) = &02 : Binär geSAVEt (,B)
Byte &015-&016	:	Gibt die Adresse an (Low- und Highbyte), wohin das Programm beim Laden gelegt wird (in dem Beispiel ist es der BASIC-Anfang &0170)
Byte &018-&019	:	Programmlänge (Low- und Highbyte)
Byte &040-&041	:	Programmlänge (Low- und Highbyte)
Byte &043-&044	:	Summe der Bytes &000-&042 (Prüfsumme)!!

Mit der Prüfsumme (&043-&044) hat es eine besondere Bewandtnis. Addieren Sie die Werte der Bytes &000-&042 zusammen, so erhalten Sie den Wert &036D. Das ist genau der Wert, der in den Bytes &043 (Lowbyte) und &044 (Highbyte) abgelegt ist. Immer dann, wenn die Summe der ersten 67 Bytes mit dem Wert aus den Bytes &043 und &044 übereinstimmt, ist dieser Record ein Programmkopf!! Stimmt die Prüfsumme nicht überein, wird dieser Record nicht als Programmkopf interpretiert! Was soll aber das Ganze?

Laden Sie bitte jetzt den Sektor mit dem Programm „TEST.ASC“ ein.

```
Spur : 2    Sektor : 6   Block : 24

<000> 31 30 20 52 45 4D 20 2A 2A 2A 2A 20 54 65 73 74  3ED  10 REM **** Test
<010> 70 72 6F 67 72 61 6D 6D 20 2A 2A 2A 2A 0D 0A 32  476  programm ****..2
<020> 30 20 50 52 49 4E 54 20 22 48 61 6C 6C 6F 22 0D  43E  0 PRINT "Hallo".
<030> 0A 33 30 20 45 4E 44 0D 0A 1A 44 11 C5 20 2A 2A  323  .30 END...D.. **
<040> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<050> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<060> 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A 2A  2A0  ****************
<070> 2A 00 00 00 1A 00 00 00 1A 24 27 00 00 00 1A 50  113  *.........$'....P
<080> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<090> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<0A0> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
<0B0> E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  E50  ................
```

Sie sehen es sofort. Eine ASCII-Datei besitzt keinen Programmkopf(!!), sondern es geht sofort mit den eigentlichen ASCII-Informationen los! Es besteht zwar die Möglichkeit, daß auch bei einer ASCII-Datei die Summe der ersten 67 Bytes zufällig den Wert der Bytes &043 und &044 ergibt, aber die Wahrscheinlichkeit ist sehr gering (wäre dieses tatsächlich der Fall, so würde dieser Record als Programmkopf interpretiert werden!!).

Das Ende des ASCII-Files ist dann erreicht, wenn auf den Wert &1A (EOF = End Of File) getroffen wird (hier in Byte &39). Die restlichen Informationen bis zum Recordende (&041-&0FF) sind unrelevant.

Noch ein paar wichtige Anmerkungen zu den unterschiedlichen Diskettenformaten. Alle bisherigen Betrachtungen bezogen sich ausschließlich auf das Format FORMAT. Ist die zu untersuchende Diskette mit FORMAT V formatiert worden, befindet sich auf den Spuren 0 und 1 nicht das CP/M-Betriebssystem (mit &E5 gefüllt). Alle anderen Angabe treffen zu.

Wie schon aus dem Assemblerlisting zum Direktzugriffsprogramm (siehe 10.0) deutlich wurde, arbeitet das Programm nur in den Formaten FORMAT und FORMAT V. Für das Datenformat (FORMAT D) und das IBM-Format (FORMAT I) sind entsprechende Änderungen vorzunehmen. Der Aufbau der Diskette ändert sich wie folgt:

FORMAT D — Im Datenformat sind keine Spuren für das CP/M reserviert. In der Spur 0/Sektor 0-3 befindet sich die Directory! Der Block &02 beginnt damit auch bei Spur 0/Sektor 4! Intern sind die Sektoren von &C1 bis &C9 bezeichnet!

FORMAT I — Das IBM-Format unterscheidet sich mit nur 8 Sektoren erheblich von den anderen Formaten! Es ist nur eine Spur (Spur 0) für das CP/M reserviert, wodurch die Directory in Spur 1/Sektor 0 beginnt! Der Block &02 beginnt in Spur 1/Sektor 4. Intern sind die Sektoren von &01 bis &08 bezeichnet!

Der weitere Aufbau der Diskette (Directory-Aufbau, Programm- und Datenablage) ist bei allen Formaten mit dem vorher beschriebenen identisch!

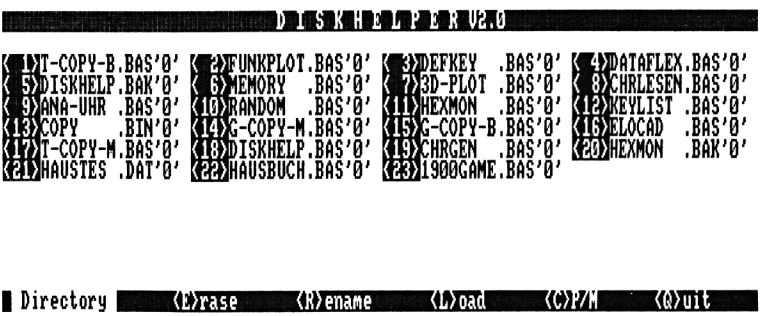
10.6 DISKHELP

In der täglichen Praxis kommt es mehr als einmal vor, daß Programme oder Programmteile auf der Diskette gelöscht und umbenannt werden müssen. Gerade durch die sehr eigenwillige Übermittelung von Diskettenbefehlen (siehe 3.0) ist das Bearbeiten einer Diskette eine umständliche und zeitraubende Arbeit. Das Programm DISKHELP ist für das Aufräumen von Disketten eine große Hilfe. Mit DISKHELP haben Sie gleichzeitig Zugriff auf alle USER-Ebenen. Es lassen sich sehr komfortabel Programme löschen, umbenennen und starten. Durch die Möglichkeit, ein beliebiges Programm zu starten, bietet sich DISKHELP als grundsätzliches Startprogramm an. Sie können sofort alle Programme (aller USER-Ebenen) überblicken und ein beliebiges auswählen. Ändern Sie alle Programme dahingehend, daß es mit dem Programmende automatisch wieder DISKHELP lädt, so kommem Sie immer wieder in den DISKHELPer! Das könnte folgendermaßen aussehen:

```
Ende, dann .... CALL &BC02
                CALL &BB4E
                PEN 1
                RUN''DISKHELP''
```

Die beiden CALLs bewirken einen Bildschirmreset (Aufheben aller Windows und Farben), mit „PEN 1" wird die aktuelle Schreibfarbe ausgewählt und RUN«...« lädt das Programm.

Die Arbeit mit DISKHELP ist denkbar einfach. Allen Programmen (Files) ist eine Nummer zwischen 1 und 64 zugeordnet. Anstatt des Programmnamens ist bei allen Funktionen des Programms stets nur die Programmnummer anzugeben. Das beschleunigt die Arbeit erheblich.



In der Abbildung ist deutlich zu erkennen, daß nach der Programmnummer der Programmname folgt. Die in Hochkommata nachgestellte Zahl gibt die USER-Ebene an, in der sich das Programm befindet.

Hier alle Befehle von DISKHELP auf einen Blick:

(D)irectory — Lädt die Directory von der aktuellen Diskette ein.

(E)rase — Löscht ein, mehrere oder alle Programme auf der Diskette. Geben Sie nur eine Programmnummer an, so wird diese nach einer Bestätigung gelöscht. Mit der Eingabe des Sterns „*" können Sie alle Programme löschen oder aus bestimmten Programmen auswählen, welche zu löschen sind.
Bei der Auswahl werden alle Programme angezeigt, die mit „J" zum Löschen bestätigt werden müssen („N" = nicht löschen). Vor dem tatsächlichen Löschen sind alle ausgewählten Programme noch einmal zu bestätigen.

(R)ename — Umbenennen eines Programms. Nach der Auswahl, welches Programm umbenannt werden soll, müssen sie einen neuen Namen mit Typenbezeichnung (z.B. „.BAS") angeben. Das Programm prüft, ob der neue Name bereits auf der Diskette existiert. Falls ja, gibt der Rechner eine Meldung aus und es muß bestätigt werden, ob der neue Name beibehalten werden soll.

(L)oad — Lädt und startet das ausgewählte Programm.

(C)P/M — Übergabe an das CP/M-Betriebssystem.

(Q)uit — Programmende.

```
1000 REM ************************************ M.W. Thoma **
1010 REM *                                                *
1020 REM *                  DISKHELP V2.0                 *
1030 REM *                                                *
1040 REM ********************************************************
1050 MODE 2:MEMORY &8FFF:ZONE 20
1060 DIM u(64),et$(64),lo(64)
1070 WINDOW#0,1,80,3,18
1080 WINDOW#1,1,80,1,1
1090 WINDOW#2,1,80,21,25
1100 WINDOW#3,1,80,19,19
1110 PEN#1,0:PAPER#1,1:CLS#1
1120 PEN#2,0:PAPER#2,1:CLS#2
```

```
1130 PEN#3,0:PAPER#3,1:CLS#3
1140 PRINT#1,TAB(30);"D I S K H E L P E R V2.0"
1150 kz$=" <D>irectory      <E>rase      <R>ename
1160 kz$=kz$+"<L>oad      <C>P/M      <Q>uit"
1170 et$(0)="*.*":i$=CHR$(24)
1180 GOSUB 2420:PRINT#3,kz$:GOSUB 1300
1190 CLS#2:PRINT#3,kz$
1200 in$=INKEY$:IF in$="" THEN 1200
1210 in$=UPPER$(in$)
1220 IF in$="D" THEN GOSUB 1300:GOTO 1190
1230 IF in$="E" THEN GOSUB 1520:GOTO 1190
1240 IF in$="R" THEN GOSUB 1980:GOTO 1190
1250 IF in$="L" THEN GOSUB 2330:GOTO 1190
1260 IF in$="Q" THEN MODE 2 :END
1270 IF in$="C" THEN öCPM:END
1280 PRINT#2,CHR$(7);:GOTO 1200
1290 '
1300 REM ************************************************
1310 REM *                                              *
1320 REM * <D>     Directory einlesen und anzeigen      *
1330 REM *                                              *
1340 REM ************************************************
1350 CLS:nr=0
1360 LOCATE#3,2,1:PRINT#3,i$;" Directory ";i$
1370 FOR se=0 TO 3
1380 CALL &A000,0,2,se,&9000
1390 FOR a1=&9000 TO &91FF STEP 32
1400 IF PEEK(a1+12)>0 OR PEEK(a1)>15 THEN 1490
1410 nr=nr+1
1420 u(nr)=PEEK(a1):et$(nr)=""
1430 FOR a2=a1+1 TO a1+11
1440 IF a2=a1+9 THEN et$(nr)=et$(nr)+"."
1450 et$(nr)=et$(nr)+CHR$(PEEK(a2))
1460 NEXT a2
1470 PRINT i$;USING "<##>";nr;:PRINT i$;et$(nr);
1480 PRINT "'";HEX$(u(nr));"'",
1490 NEXT a1
1500 NEXT se
1510 RETURN
1520 '
1530 REM ************************************************
1540 REM *                                              *
1550 REM * <E>         Loeschen von Programmen          *
1560 REM *                                              *
1570 REM ************************************************
1580 LOCATE#3,19,1:PRINT#3,i$;" Erase ";i$
1590 INPUT#2," >Bitte Filenummer oder '*' eingeben :";er$
1600 IF LEN(er$)=0 THEN RETURN
1610 IF er$="*" THEN 1700
```

```
1620 no=VAL(er$):IF no<1 OR no>nr THEN RETURN
1630 PRINT#2
1640 PRINT#2," >Erase : ";i$;" ";et$(no);" ";i$
1650 PRINT#2," >Sind Sie sicher (J/N) ?"
1660 in$=INKEY$:IF in$="" THEN 1660
1670 IF UPPER$(in$)<>"J" THEN RETURN
1680 er$=STR$(u(no))+":"+et$(no)
1690 öERA,§er$:öUSER,0:PRINT#3,kz$:GOSUB 1300:RETURN
1700 CLS#2:INPUT#2," >Gesamte Diskette loeschen (J/N) ";in$
1710 IF UPPER$(in$)<>"J" THEN 1790
1720 PRINT#2
1730 PRINT#2," >Sind Sie ";i$;" wirklich ";i$;
1740 PRINT#2," sicher (J/N) ????"
1750 in$=INKEY$:IF in$="" THEN 1750
1760 IF UPPER$(in$)="N" THEN RETURN
1770 IF UPPER$(in$)="J" THEN öERA,§et$(0):RETURN
1780 PRINT#2,CHR$(7);:GOTO 1750
1790 an=0:CLS#2
1800 FOR i=1 TO nr
1810 PRINT#2," > ";et$(i),"Loeschen (J/N)  ";
1820 in$=INKEY$:IF in$="" THEN 1820
1830 IF UPPER$(in$)="N" THEN PRINT#2,i$;" Nein ";i$:GOTO 1870
1840 IF UPPER$(in$)<>"J" THEN 1820
1850 PRINT#2,i$;"  Ja  ";i$
1860 an=an+1:lo(an)=i
1870 NEXT i
1880 IF an=0 THEN RETURN
1890 CLS#2:FOR i=1 TO an
1900 PRINT#2," >";et$(lo(i)),"  Sicher ? ",
1910 in$=INKEY$:IF in$="" THEN 1910
1920 IF UPPER$(in$)<>"J" THEN PRINT#2,i$;" Nein ";i$:GOTO 1960
1930 er$=STR$(u(lo(i)))+":"+et$(lo(i))
1940 PRINT#2,i$;" Geloescht ";i$
1950 öERA,§er$
1960 NEXT i
1970 öUSER,0:PRINT#3,kz$:GOSUB 1300:RETURN
1980 '
1990 REM ************************************************
2000 REM *                                              *
2010 REM * <R>           Programm umbennen              *
2020 REM *                                              *
2030 REM ************************************************
2040 LOCATE#3,32,1:PRINT#3,i$;" Rename ";i$
2050 INPUT#2," >Bitte Filenummer eingeben :";er$
2060 no=VAL(er$):IF no<1 OR no>nr THEN RETURN
2070 PRINT#2
2080 PRINT#2," >Alt:";i$;" ";et$(no);" ";i$;
2090 PRINT#2," = Neu:";
2100 INPUT#2,neu$
```

```
2110 IF LEN(neu$)<5 OR LEN(neu$)>12 THEN RETURN
2120 IF MID$(neu$,LEN(neu$)-3,1)<> "." THEN RETURN
2130 neu$=UPPER$(neu$)
2140 FOR i=1 TO nr
2150 t1$=LEFT$(et$(i),LEN(neu$)-4)
2160 t2$=LEFT$(neu$,LEN(neu$)-4)
2170 IF t1$=t2$ THEN 2190
2180 NEXT i:GOTO 2230
2190 PRINT#2," >ACHTUNG: Filename vorhanden im Eintrag";i
2200 PRINT#2," >Name beibehalten (J/N) ?"
2210 in$=INKEY$:IF in$="" THEN 2210
2220 IF UPPER$(in$)<>"J" THEN RETURN
2230 neu$=STR$(u(no))+":"+neu$
2240 alt$=STR$(u(no))+":"+et$(no)
2250 öREN,§neu$,§alt$
2260 öUSER,0:PRINT#3,kz$:GOSUB 1300:RETURN
2270 '
2280 REM ************************************************
2290 REM *                                              *
2300 REM * <L>    Laden und starten eines Programms     *
2310 REM *                                              *
2320 REM ************************************************
2330 LOCATE#3,46,1:PRINT#3,i$;" Load ";i$
2340 INPUT#2," >Filenummer (0=Menue) :";no
2350 IF no<1 OR no>nr THEN RETURN
2360 prg$=STR$(u(no))+":"+et$(no)
2370 CALL &BC02:CALL &BB4E:PEN 1:MEMORY &A67B
2380 LOCATE 20,23
2390 PRINT "Lade Programm :";prg$
2400 RUN prg$:END
2410 '
2420 REM ************************************ M.W. Thoma **
2430 REM *                                              *
2440 REM *        Direktzugriff auf die Diskette        *
2450 REM *                                              *
2460 REM *     Lesen: CALL &A000,DRIVE,SPUR,SEKTOR,PUFFER  *
2470 REM * Schreiben: CALL &A00C,DRIVE,SPUR,SEKTOR,PUFFER  *
2480 REM *                                              *
2490 REM ************************************************
2500 summe=0:FOR i=&A000 TO &A086
2510 READ a$:a=VAL("&"+a$):POKE i,a:summe=summe+a
2520 NEXT i
2530 IF summe=16125 THEN RETURN
2540 PRINT "Fehler in den DATA-Zeilen":END
2550 '
2560 DATA CD,18,A0,3E,84,32,85,A0,CD,5A,A0,C9,CD,18,A0,3E
2570 DATA 85,32,85,A0,CD,5A,A0,C9,F5,3E,00,32,86,A0,F1,FE
2580 DATA 04,C2,76,A0,DD,7E,07,DD,7E,06,FE,02,D2,76,A0,32
2590 DATA 7D,A0,DD,7E,05,DD,7E,04,FE,28,D2,76,A0,32,7E,A0
```

```
2600 DATA DD,7E,03,DD,7E,02,FE,0A,D2,76,A0,CE,40,32,7F,A0
2610 DATA DD,66,01,DD,6E,00,22,80,A0,C9,21,85,A0,CD,D4,BC
2620 DATA 22,82,A0,79,32,84,A0,21,7D,A0,5E,23,56,23,4E,2A
2630 DATA 80,A0,DF,82,A0,C9,E1,3E,FF,32,86,A0,C9,00,00,00
2640 DATA 00,00,00,00,00,00,00
2650 REM ************************************************
```

11.0 Daten in einen Block ablegen

Nachdem ein Werkzeug zur Verfügung steht, um Blöcke der Diskette einzulesen, zu ändern und wieder zurückzuschreiben (siehe Abschn. 10.1), kann ein Block mit beliebigen Informationen (Daten) gefüllt werden. Betrachtet man einen Block als einen Datensatz, so ist ein Zugriff auf einen beliebigen Datensatz möglich (Random Access). Bei einer Datensatzgröße von 512 Bytes (1 Block) finden insgesamt 360 Datensätze (40 Spuren * 9 Sektoren) auf einer Diskette Platz. Numeriert man die Datensätze von 0 bis 359 durch, so kann jeder Datensatz nach seiner Nummer exakt auf der Diskette lokalisiert werden (da ein Datensatz gleich einem Block ist, kann hier auch mit der Blocknummer gearbeitet werden).

```
Spur   =  INT(blocknr/9)
Sektor =  blocknr MOD 9
```

Doch nicht immer belegt ein Datensatz gleich einen gesamten Block der Diskette. Ist der Datensatz z.B. 128 Bytes lang, so passen 4 Datensätze in einen Block und damit 1440 (4*360) Datensätze auf die Diskette. Aus der nachfolgenden Tabelle können Sie die möglichen Variationen der Datensatzlänge (in Bytes) entnehmen.

DSlänge	DS/Block	DS/Diskette	DSnummern
1	512	184320	0-184319
2	256	92160	0- 92159
4	128	46080	0- 46079
8	64	23040	0- 23039
16	32	11520	0- 11519
32	16	5760	0- 5759
64	8	2880	0- 2879
128	4	1440	0- 1449
256	2	720	0- 719
512	1	360	0- 359

Damit ein möglichst einfacher Zugriff auf die Datensätze gewährleistet ist, sollten die hier angegebenen Datensatzlängen (immer eine 2er Potenz) benutzt werden! D.h., wenn Sie einen Datensatz mit 58 Zeichen benötigen, so müssen Sie die Datensatzlänge auf 64 Zeichen erhöhen.

Mit den nachstehenden Gleichungen können Sie dann genau berechnen, wo sich der gewünschte Datensatz auf der Diskette befindet (Spur/Sektor) und ab welcher Position innerhalb des Blocks er beginnt.

```
anzahl   = 512/DSlänge
blocknr  = INT(DSnummer/anzahl)
spur     = INT(blocknr/9)
sektor   = blocknr MOD 9
position = (DSnummer MOD anzahl)*DSlänge
```

Zur Erläuterung ein Zahlenbeispiel: Gesucht wird der Datensatz mit der Nummer 90. Die Datensatzlänge beträgt 128 Bytes.

```
      512/128 =  4          4 Datensätze pro Block
    INT(90/4) = 22          Der Datensatz liegt im 22. Block
    INT(22/9) =  2          Der Block 22 liegt in der Spur 2
    22 MOD 9  =  4          und im Sektor 4
(90 MOD 4)*128 = 2*128= 256 Ab Byte 256 des Blocks folgt der
                            Datensatz
```

Der Datensatz Nummer 90 befindet sich also in dem Block Spur 2/Sektor 4. Nach dem Laden des Blockes liegt der Datensatz innerhalb des Puffers ab dem Byte 256.

Nach den bisherigen Erläuterungen dürfte es Ihnen keine Schwierigkeiten bereiten, eine kleine Datei im Direktzugriff aufzubauen. Anzumerken ist noch, daß bei den bisherigen Betrachtungen die gesamte Diskette, also auch die Systemspuren (Spur 0 und 1) sowie die Directoryspur (Spur 2) für Daten genutzt wurden. Das hat den Vorteil, daß wesentlich mehr Datensätze auf der Disketten Platz finden. Der Nachteil liegt in der Tatsache, daß zum einen kein CP/M mehr zur Verfügung steht und zum anderen, daß die Diskette keine Directory mehr besitzt. Umgangen werden kann dieses Problem, indem die ersten drei Spuren (Spur 0-2) nicht benutzt werden (zur errechneten Spur 3 Spuren hinzuzählen).

11.1 Index-Dateien – Hilfe beim Finden von Datensätzen

Bisher war ein Suchen nach Datensätzen nur nach der Datensatznummer möglich. Bei einer Adressendatei ist dieses natürlich sehr unbefriedigend, da Sie immer wissen müssen, in welchem Datensatz sich ein bestimmter Eintrag befindet (Die Adresse von Meyer liegt z.B. im Datensatz Nummer 18). Diese Arbeit, zu wissen wo ein bestimmter Datensatz innerhalb der gesamten Datei liegt, übernimmt im allgemeinen eine sogenannte Index-Datei (Index = Verzeichnis, Register). Innerhalb der Index-Datei befindet sich eine Liste, bestehend aus Suchbegriff (z.B. dem Namen) und dem Ort (Datensatznummer), wo der Datensatz sich innerhalb der Datei befindet. Das bedeutet, über eine kleine Datenmenge (Index-Datei) wird eine große Datenmenge selektiert (gesamte Datei).

Ist innerhalb der Datei nach dem Namen zu suchen, so muß sich in der Index-Datei der Name mit der dazugehörigen Datensatznummer befinden.

DSnummer	Name
1	Meyer
2	Huber
3	Kalli

Um die Daten für „Meyer" zu finden, ist nur die Index-Datei zu durchsuchen, bis „Meyer" gefunden ist. Aus der Index-Datei wird dann die Datensatznummer entnommen. Mit dieser Datensatznummer kann wiederum die Position des Datensatzes errechnet werden (siehe 10.2).
Das bedeutet, daß das eigentliche Suchen nach einem Suchbegriff nur in der Index-Datei durchgeführt wird. Die Index-Datei befindet sich in der Regel im Computer, um auf die Daten möglichst schnell zugreifen zu können.

Der Aufbau einer Index-Datei kann sehr unterschiedlich sein. Eine Möglichkeit besteht in der Zuordnung der Index-Nummer eines Arrays zur Datensatznummer. Für jeden Datensatz ist ein Index-Feld notwendig. Für 100 Datensätze benötigt man dann ein Array mit 100 Feldern (DIM index$(100)). Die Index-Nummer wird der Datensatznummer zugeordnet:

index$(12) = Datensatznummer 12
index$(74) = Datensatznummer 74

Innerhalb der Variablen index$(..) befindet sich der Suchbegriff (z.B. der Name).

```
10 INPUT "Suchbegriff :";name$
20 FOR i= 1 TO 100
30 IF name$ = index$(i) THEN 70
40 NEXT i
50 PRINT "Suchbegriff nicht vorhanden"
60 GOTO 10
70 PRINT "Datensatznummer :";i
```

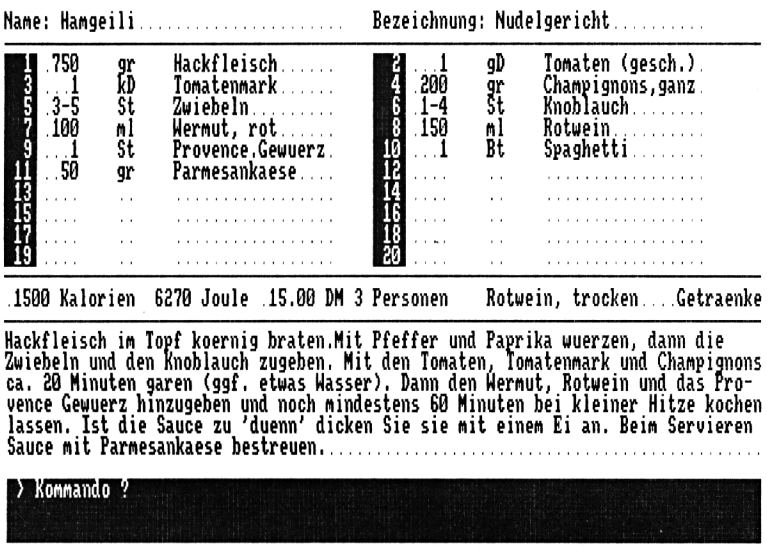
Ist der gewünschte Begriff gefunden (Zeile 30), gibt die Index-Nummer (i) die Datensatznummer an. Mit Hilfe der Datensatznummer ist wiederum der Ort zu errechnen (siehe 11.0), wo sich der Datensatz auf der Diskette befindet.

Für die Verwaltung von Index-Dateien bietet sich eine sequentielle Datei an. Man spricht auch von einem sequentiell-indizierten Datenprinzip. Alle Änderungen, die in der Datei durchgeführt werden, müssen selbstverständlich auch in der Index-Datei durchgeführt werden. Wird ein Datensatz gelöscht, so ist auch in der Index-Datei zu kennzeichnen, daß der Datensatz wieder frei (nicht belegt) ist.

11.2 Kochbuch – eine Direktzugriffs-Datei

Das Programm „Kochbuch" faßt den Direktzugriff noch einmal zusammen. Das Kochbuch verwaltet 160 Rezepte im Direktzugriff. Das Suchen nach Rezepten findet (mit Ausnahme des „Super Suchens") innerhalb einer Index-Datei statt. Jedes Rezept (Datensatz) besteht aus 1024 (1 kByte, 2 Blöcke) Zeichen, so daß insgesamt 320 Blöcke der Diskette genutzt werden (Ab Spur 3/Sektor 2, damit bleiben die Systemspuren und die Directoryspur unberührt).

Pro Rezept können bis zu 20 Zutaten und eine Kochanleitung von 480 Zeichen (6 Zeilen) eingegeben werden. Weiterhin bekommt das Rezept einen Namen und eine Bereichsbezeichnung. Als zusätzliche Informationen finden eine Kalorienangabe (gleichzeitige Anzeige in Joule), der Preis, die Anzahl der Personen und ein empfohlenes Getränk Platz.



Nach dem Laden und Starten des „Kochbuchs" fragt das Programm nach dem Namen der Index-Datei. Ist noch keine Index-Datei angelegt, geben Sie den Namen der neuanzulegenden Datei mit einem vorgestellten Stern „*" an.

Neuanlage: Name der Index-Datei : *dateiname

Jetzt legt das Programm eine neue (Index-) Datei an. Ist bereits eine Datei angelegt, so geben Sie nur den Dateinamen an (ohne Stern)! Das Programm lädt dann die vorhandene Index-Datei.

Nachdem die Index-Datei angelegt oder geladen ist, fordert das Programm Sie auf, die Datendiskette in das Laufwerk zu legen. Für die Datendiskette ist grundsätzlich eine extra Diskette zu benutzen! Bei der Neuanlage einer Datei sollte immer eine neue, formatierte Diskette benutzt werden. Wenn Sie die Datendiskette eingelegt und das Wechseln mit einer Taste quittiert haben, befinden Sie sich in der Kommandoebene.

Hier stehen Ihnen zunächst 5 Befehle zur Verfügung, die später detaillierter erläutert werden:

B	Rezept bearbeiten
E	Neues Rezept anlegen

F	Finden von bestimmten Rezepten
S	Sichern eines Rezeptes auf Diskette
Q	Programmende (sichern der Index-Datei)

Nur das Rezept bzw. die Rezeptkarte, die momentan auf dem Bildschirm steht, kann bearbeitet oder in die Datei gesichert werden. Um ein Rezept zu sichern, sind 3 Felder der Rezeptkarte zwingend auszufüllen: Der Name, die Bezeichnung und der Preis des Gerichtes. Fehlt eine dieser Angaben, ist das Rezept nicht zu sichern. Diese 3 Eingaben bilden gleichzeitig den Eintrag in die Index-Datei.

(E)ingabe Mit dem Kommando „E" ist ein neues Rezept anzulegen. Die Bildschirmmaske wird gelöscht und das Programm fragt nach dem Rezeptnamen (bis zu 30 Zeichen). Betätigen Sie nur die ENTER-Taste, kehren Sie in die Kommandoebene zurück. Nach der Eingabe des Rezeptnamens wird geprüft, ob sich bereits ein Rezept mit diesem Namen in der Datei befindet. Ist dies der Fall, erfolgt eine (Fehler-) Meldung. Ist das Rezept noch nicht vorhanden, wird der Rezeptname in die neue Rezeptkarte übernommen. Automatisch befinden Sie sich dann in der Bearbeitungsebene.

(B)earbeiten Die Bearbeitungsebene ist so ausgelegt, daß Sie jedes Feld des Rezeptes anlegen, ändern oder löschen können. Sie erhalten hier folgende Meldung:

Bearbeiten (BZ/01-20/KA/PR/PS/GT/TX) :

In Klammern stehen die hier möglichen Befehle (2-stellig eingeben!). Nach der Eingabe von „BZ" können Sie das „Bezeichungsfeld" eingeben, ändern oder löschen. „01-20" steht für die Zutatennummern. Geben Sie z.B. eine „9" ein, so können Sie das Zutatenfeld Nummer 9 bearbeiten. Die Zutatenfelder sind in 3 Bereiche geteilt: Ein 4-stelliges Größenfeld, ein 2-stelliges Einheitenfeld (z.B. EL für Eßlöffel) und ein 17-stelliges Bezeichnungsfeld. Jedes Feld ist bei der Eingabe mit der ENTER-Taste abzuschließen. „KA" steht für Kalorien. Nach der Übergabe der Kalorien errechnet sich das Programm eigenständig den Joulewert und trägt diesen in das Joulefeld ein. Die Felder „PR" (Preis), „PS" (Personen) und „GT" (Getränk) werden ebenso eingegeben, geändert oder gelöscht.

Der letzte Befehl „TX" (Text) ermöglicht eine Eingabe in das 480 Zeichen große Zubereitungsfeld. Sie können direkt in das Feld schreiben und mit der DEL-Taste das Zeichen links vom Cursor löschen. Ist die Eingabe beendet, betätigen Sie die ENTER-Taste (aber nur dann!).
Jedesmal, wenn Sie ein Feld des Rezeptes bearbeiten wollen, müssen Sie es komplett neu eingeben (vielleicht ändern Sie diesen Programmteil noch ?). Ist das Rezept bearbeitet, betätigen Sie einfach die ENTER-Taste und Sie kehren zurück in die Kommandoebene (passiert auch dann, wenn Sie einen unbekannten Bearbeitungsbefehl eingeben).

(S)ichern

Mit dem Kommando „S" kann die aktuelle Rezeptkarte in die Datei gesichert werden. Zunächst wird geprüft, ob die Felder Name, Bezeichnung und Preis eingetragen sind (falls nicht, Meldung). Nun wird geprüft, ob sich das Rezept bereits in der Datei befindet. Falls ja, fragt das Programm, ob das alte Rezept überschrieben werden soll (J/N). Handelt es sich um eine Neuanlage, wird gleichzeitig geprüft, ob noch ein freier Speicherplatz vorhanden ist.
Dann wird das Index-Feld aufbereitet und die Karteikarte in die Datei (Diskette) geschrieben (Dieser Vorgang dauert einige Sekunden, da der gesamte Puffer mit der Karteikarte gefüllt werden muß.).

(F)inden

Die Suchebene gliedert sich wiederum in 5 Befehle.
Mit „GI" (Gesamt-Index) wird die gesamte Index-Datei im Zubereitungsfeld in 5er Blöcken aufgelistet.
Mit dem Befehl „TI" (TEil-Index) können Sie innerhalb der Index-Datei nach einem beliebigen Ausdruck suchen (Z.B. Nudel). Es werden Ihnen dann alle zutreffenden Indexeinträge angezeigt.
„HR" (Hole Rezept) liest ein Rezept von der Diskette in die Bildschirmmaske ein. Die Auswahl des Rezeptes findet nach der Rezeptnummer statt, die Sie aus der Index-Datei erhalten (Dieses Rezept könnte auch in der Bearbeitungsebene bearbeitet und mit „S" geändert in die Datei zurückgeschrieben werden.).
„LR" steht für „Lösche Rezept" und löscht das angegebene Rezept (Nummer). Zur Sicherheit wird das zu löschende Rezept

noch einmal angezeigt und Sie können entscheiden (J/N), ob das Rezept tatsächlich zu löschen ist.
Als letzte Funktion bietet „SS" (Super Suchen) etwas besonderes. Hier können Sie bis zu 20 Zutaten eingeben, die Ihnen zum Erstellen einer Speise zur Verfügung stehen. Die gesamte Datei wird auf diese Zutaten hin geprüft. Ist ein Rezept gefunden, welches alle angegebenen Zutaten enthält, erfolgt eine Meldung und das Rezept erscheint auf dem Bildschirm. Sie können dann entscheiden, ob nach weiteren, zutreffenden Rezepten gesucht werden soll (Dieser Suchvorgang benötigt bei vielen Rezepten schon eine Weile). Den Suchvorgang können Sie jederzeit mit der Betätigung einer beliebigen Tasten abbrechen.

(Q)uit — Programmende. Nach der Aufforderung, die Datendiskette aus dem Laufwerk zu nehmen und die Index-Diskette einzulegen, sichert das Programm die Index-Datei zurück auf die Diskette.

```
1000 REM ************************************ M.W.Thoma **
1010 REM *                                               *
1020 REM *                    Kochbuch                   *
1030 REM *                                               *
1040 REM ************************************************
1050 MODE 2:INK 0,13:INK 1,0:BORDER 13
1060 'INK 0,26:'INK 1,0:'BORDER 13:'Green Monitor
1070 OPENOUT"dummy":MEMORY HIMEM-1:CLOSEOUT
1080 MEMORY &8FFF:GOSUB 4930
1090 DEF FN lb(adr)=&FF AND UNT(adr)
1100 DEF FN hb(adr)=&FF AND INT(adr/256)
1110 WINDOW#0,1,80,23,25
1120 WINDOW#1,1,80, 1,15
1130 WINDOW#2,1,80,16,22
1140 PAPER 1:PEN 0:CLS
1150 SYMBOL 240,0,0,0,0,0,0,0,8,0
1160 DIM fl(32),e$(32),sz$(20),d$(160)
1170 fl(1)=30:fl(2)=22
1180 FOR i=3 TO 22:fl(i)=23:NEXT i
1190 FOR i=23 TO 28:fl(i)=80:NEXT i
1200 fl(29)=5:fl(30)=6:fl(31)=1:fl(32)=20
1210 p$=CHR$(240):s$=CHR$(95):l$="     ":iv$=CHR$(24)
1220 ba1=&9000:be1=&91FF:ba2=&9200:be2=&93FF
1230 GOSUB 3950
1240 '
1250 CLS:INPUT " >Name der INDEX-Datei :";na$
1260 IF LEFT$(na$,1)<>"*" THEN 1310
```

```
1270 na$=MID$(na$,2,LEN(na$)-1)
1280 OPENOUT na$+".seq":PRINT#9,0:PRINT#9,0:CLOSEOUT
1290 CLS:GOTO 1350
1300 '
1310 OPENIN na$+".seq":INPUT#9,an:INPUT#9,dp
1320 IF an=0 THEN 1340
1330 FOR i=1 TO an:LINE INPUT#9,d$(i):NEXT i
1340 CLOSEIN: CLS
1350 PRINT " >Bitte Datendiskette einlegen (ENTER) !!"
1360 in$=INKEY$:IF in$="" THEN 1360
1370 '
1380 REM ************************************************
1390 REM *                                              *
1400 REM *                Kommandoeingabe               *
1410 REM *                                              *
1420 REM ************************************************
1430 CLS:PRINT ">Bitte Kommando ?";CHR$(232)
1440 in$=INKEY$:IF in$="" THEN 1440
1450 in$=UPPER$(in$)
1460 IF in$="E" THEN GOSUB 1530
1470 IF in$="B" THEN GOSUB 1700
1480 IF in$="S" THEN GOSUB 2310
1490 IF in$="F" THEN GOSUB 2740
1500 IF in$="Q" THEN GOSUB 3800
1510 PRINT CHR$(7);:GOTO 1430
1520 '
1530 REM ************************************************
1540 REM *                                              *
1550 REM * <E>         Neues Rezept anlegen             *
1560 REM *                                              *
1570 REM ************************************************
1580 CLS:PRINT " >Neues Rezept anlegen
1590 GOSUB 4660:z=1
1600 LOCATE 1,2:INPUT " >Name   :";e$(z)
1610 IF LEN(e$(z))=0 THEN RETURN
1620 IF LEN(e$(z))>fl(z) THEN PRINT CHR$(7):GOTO 1600
1630 e$(z)=e$(z)+STRING$(fl(z)-LEN(e$(z)),p$)
1640 FOR i=1 TO an
1650 IF LEFT$(d$(i),30)=e$(1) THEN 1670
1660 NEXT i:GOSUB 4130:GOTO 1700
1670 BORDER 6: PRINT " >Rezept ist schon vorhanden !"
1680 GOSUB 4850:BORDER 13:RETURN
1690 '
1700 REM ************************************************
1710 REM *                                              *
1720 REM * <B>              Bearbeiten                  *
1730 REM *                                              *
1740 REM ************************************************
1750 CLS:INPUT" >Bearbeiten (BZ/01-20/KA/PR/PS/GT/TX) :";e1$
```

```
1760 e1$=UPPER$(e1$)
1770 IF e1$="BZ" THEN z= 2:GOTO 1960
1780 IF e1$="KA" THEN z=29:GOTO 2020
1790 IF e1$="PR" THEN z=30:GOTO 2020
1800 IF e1$="PS" THEN z=31:GOTO 2020
1810 IF e1$="GT" THEN z=32:GOTO 1960
1820 IF e1$="TX" THEN GOSUB 4740:GOSUB 2070:GOTO 1700
1830 e=VAL(e1$):IF e<1 OR e>20 THEN RETURN
1840 '
1850 LOCATE 3,2:z=e+2:PRINT iv$;LEFT$(e$(z),4);iv$;"  ";
1860 PRINT iv$;MID$(e$(z),5,2);iv$;"  ";
1870 PRINT iv$;RIGHT$(e$(z),17);iv$
1880 LOCATE 3,3:LINE INPUT e1$:LOCATE 9,3:LINE INPUT e2$
1890 LOCATE 13,3:LINE INPUT e3$
1900 IF LEN(e1$)>4 OR LEN(e2$)>2 OR LEN(e3$)>17 THEN 1880
1910 e1$=STRING$(4-LEN(e1$),p$)+e1$
1920 e2$=STRING$(2-LEN(e2$),p$)+e2$
1930 e3$=e3$+STRING$(17-LEN(e3$),p$)
1940 e$(z)=e1$+e2$+e3$:GOSUB 4130:GOTO 1700
1950 '
1960 PRINT " ";iv$;e$(z);iv$
1970 LOCATE 2,3:LINE INPUT e1$
1980 IF e1$="" OR LEN(e1$)>f1(z) THEN 1970
1990 e1$=e1$+STRING$(f1(z)-LEN(e1$),p$)
2000 e$(z)=e1$:GOSUB 4130:GOTO 1700
2010 '
2020 PRINT " ";iv$;e$(z);iv$
2030 LOCATE 2,3:LINE INPUT e1$
2040 IF LEN(e1$)>f1(z) THEN 2030
2050 e1$=STRING$(f1(z)-LEN(e1$),p$)+e1$:GOTO 2000
2060 '
2070 REM ************************************************
2080 REM *        Texteingabe im Zubereitungsfeld       *
2090 REM ************************************************
2100 CLS
2110 PRINT " >Bitte Anleitung eingeben (Ende mit ENTER)<"
2120 zei=1:spa=1:LOCATE#2,spa,zei:adr=ba2
2130 PRINT#2,s$;
2140 i$=INKEY$:IF i$="" THEN 2140
2150 ip=ASC(i$):IF ip=13 THEN 2280
2160 IF ip<32 OR ip>127 THEN PRINT CHR$(7);:GOTO 2140
2170 IF ip=127 THEN 2230
2180 IF zei=7 AND spa=1 THEN PRINT CHR$(7);:GOTO 2140
2190 LOCATE#2,spa,zei:PRINT#2,i$;
2200 POKE adr,ip:adr=adr+1
2210 IF spa=80 THEN zei=zei+1:spa=1:ELSE spa=spa+1
2220 IF adr=be2 THEN 2140:ELSE 2130
2230 IF adr=ba2 THEN 2140
2240 LOCATE#2,spa,zei:PRINT#2,p$:
```

```
2250 IF spa=1 THEN spa=80:zei=zei-1:ELSE spa=spa-1
2260 adr=adr-1:POKE adr,32:LOCATE#2,spa,zei
2270 GOTO 2130
2280 FOR i=adr TO ba2+479:POKE i,240:NEXT i
2290 GOSUB 4530:RETURN
2300 '
2310 REM ************************************************
2320 REM *                                              *
2330 REM *  <S>             Rezept sichern              *
2340 REM *                                              *
2350 REM ************************************************
2360 CLS: PRINT " >Sichern des Rezeptes":f=0
2370 IF LEFT$(e$(1),1)=p$ THEN f=1
2380 IF LEFT$(e$(2),1)=p$ THEN f=1
2390 IF e$(30)=STRING$(6,p$) THEN f=1
2400 IF f=0 THEN 2470
2410 '
2420 BORDER 6
2430 PRINT " Die Eingaben der Felder";
2440 PRINT " NAME, BEREICH und PREIS sind zwingend!"
2450 GOSUB 4850:BORDER 13:RETURN
2460 '
2470 FOR i=1 TO an
2480 IF LEFT$(d$(i),30)=e$(1) THEN 2580
2490 NEXT i
2500 IF an<159 THEN dp=dp+1:GOTO 2650
2510 FOR i=1 TO an
2520 IF LEFT$(d$(i),1)=p$ THEN dp=i:GOTO 2660
2530 NEXT i
2540 '
2550 BORDER 6:PRINT " KEIN SPEICHERPLATZ MEHR VORHANDEN"
2560 GOSUB 4850:BORDER 13:RETURN
2570 '
2580 BORDER 6
2590 PRINT " Soll altes Rezept ueberschrieben werden (J/N)?"
2600 in$=INKEY$:IF in$="" THEN 2600
2610 BORDER 10
2620 IF UPPER$(in$)="J" THEN dp=i:GOTO 2660
2630 RETURN
2640 '
2650 an=an+1
2660 d$(dp)=e$(1)+e$(2)+e$(30)
2670 GOSUB 4430
2680 gs=dp*2+27:spur=INT(gs/9):sektor=gs MOD 9
2690 CALL &A00C,0,spur,sektor,ba1
2700 sektor=sektor+1:IF sektor=9 THEN sektor=0:spur=spur+1
2710 CALL &A00C,0,spur,sektor,ba2
2720 BORDER 13:RETURN
2730 '
```

```
2740 REM ************************************************
2750 REM *                                              *
2760 REM *  <F>          Finden von Rezepten             *
2770 REM *                                              *
2780 REM ************************************************
2790 CLS: INPUT " >Suchen von Daten (GI/TI/HR/LR/SS) :";in$
2800 in$=UPPER$(in$)
2810 IF in$="GI" THEN 2880
2820 IF in$="TI" THEN 3000
2830 IF in$="HR" THEN 3130
2840 IF in$="LR" THEN 3260
2850 IF in$="SS" THEN 3400
2860 RETURN
2870 '
2880 CLS#2:w=0:FOR i=1 TO an
2890 IF LEFT$(d$(i),1)=p$ THEN 2960
2900 w=w+1
2910 PRINT#2,iv$;USING "####";i;
2920 PRINT#2,iv$;l$;LEFT$(d$(i),30);l$;
2930 PRINT#2,MID$(d$(i),31,22);l$;RIGHT$(d$(i),6)
2940 IF w<5 THEN 2960
2950 GOSUB 3740:IF in$="E" THEN 2980
2960 NEXT i
2970 GOSUB 3690
2980 GOSUB 4820:GOTO 2740
2990 '
3000 INPUT " >Suchbegriff :";sb$
3010 IF sb$="" THEN RETURN
3020 CLS#2:w=0:FOR i=1 TO an
3030 IF INSTR(d$(i),sb$)=0 THEN 3090
3040 PRINT#2,iv$;USING "####";i;
3050 PRINT#2,iv$;l$;LEFT$(d$(i),30);l$;
3060 PRINT#2,MID$(d$(i),31,22);l$;RIGHT$(d$(i),6)
3070 w=w+1:IF w<5 THEN 3090
3080 GOSUB 3740:IF in$="E" THEN 3110
3090 NEXT i
3100 GOSUB 3690
3110 GOSUB 4820:GOTO 2740
3120 '
3130 CLS:INPUT " >Bitte Rezept-Nummer :";rn
3140 IF rn<1 OR rn>an THEN 3170
3150 IF LEFT$(d$(rn),1)=p$ THEN 3170
3160 GOTO 3200
3170 PRINT " >Gewuenschtes Rezept nicht vorhanden !!!"
3180 BORDER 6:GOSUB 4850:BORDER 13:RETURN
3190 '
3200 gs=rn*2+27:spur=INT(gs/9):sektor=gs MOD 9
3210 CALL &A000,0,spur,sektor,ba1
3220 sektor=sektor+1:IF sektor=9 THEN sektor=0:spur=spur+1
```

```
3230 CALL &A000,0,spur,sektor,ba2
3240 GOSUB 4590:GOSUB 4770:RETURN
3250 '
3260 CLS:INPUT " >Bitte Rezept-Nummer :";rn
3270 IF rn<1 OR rn>an THEN RETURN
3280 IF LEFT$(d$(rn),1)=p$ THEN 3300
3290 GOTO 3330
3300 PRINT " >Rezept ist schon geloescht !!!"
3310 BORDER 6:GOSUB 4850:BORDER 13:RETURN
3320 '
3330 GOSUB 3200
3340 PRINT " >Sind Sie sicher (J/N) ?"
3350 in$=INKEY$:IF in$="" THEN 3350
3360 IF UPPER$(in$)="J" THEN 3380
3370 RETURN
3380 GOSUB 4660:d$(rn)=e$(1)+e$(2)+e$(30):RETURN
3390 '
3400 PRINT " >Bitte geben Sie die vorhandenen Zutaten ein"
3410 x=1
3420 INPUT " >Zutat :";sz$(x)
3430 IF sz$(x)="" AND x=1 THEN RETURN
3440 IF sz$(x)="" THEN 3470
3450 x=x+1:IF x=21 THEN 3470:ELSE 3420
3460 '
3470 CLS:BORDER 6:PRINT " > Bitte warten ....."
3480 x=x-1:x1=0
3490 FOR rn=1 TO an
3500 IF LEFT$(d$(rn),1)=p$ THEN 3610
3510 GOSUB 3200
3520 FOR i=3 TO 22
3530 FOR j=1 TO x
3540 tz$=MID$(e$(i),7,LEN(sz$(J)))
3550 IF tz$<>sz$(j) THEN 3570
3560 x1=x1+1:IF x1=x THEN 3630
3570 NEXT j
3580 NEXT i
3590 x1=0
3600 IF INKEY$<>"" THEN rn=an
3610 NEXT rn
3620 BORDER 13:RETURN
3630 CLS:PRINT " > GEFUNDEN !!!":BORDER 6,5
3640 PRINT " >Weitere Rezepte suchen (J/N) ?":GOSUB 4850
3650 in$=INKEY$:IF in$="" THEN 3650
3660 BORDER 6
3670 IF UPPER$(in$)="J" THEN 3590:ELSE BORDER 13:RETURN
3680 '
3690 LOCATE#2,24,6
3700 PRINT#2,iv$;" Keine (weiteren) Daten vorhanden ";iv$
3710 in$=INKEY$:IF in$="" THEN 3710
```

```
3720 RETURN
3730 '
3740 LOCATE#2,29,6
3750 PRINT#2,iv$;" Weiter mit ENTER (E=Ende) ";iv$:w=0
3760 in$=INKEY$:IF in$="" THEN 3760
3770 in$=UPPER$(in$)
3780 CLS#2:RETURN
3790 '
3800 REM ************************************************
3810 REM *                                              *
3820 REM * <0>             Programmende                 *
3830 REM *                                              *
3840 REM ************************************************
3850 CLS
3860 BORDER 6:PRINT " >Bitte INDEX-Diskette einlegen (ENTER)"
3870 in$=INKEY$:IF in$="" THEN 3870
3880 OPENOUT na$+".seq"
3890 PRINT#9,an:PRINT#9,dp
3900 IF an=0 THEN  3920
3910 FOR i=1 TO an:PRINT#9,d$(i):NEXT i
3920 CLOSEOUT
3930 MODE 2:PEN 1:PAPER 0:END
3940 '
3950 REM ************************************************
3960 REM *               Bildschirm Maske               *
3970 REM ************************************************
3980 LOCATE#1,1,1:LOCATE#2,1,1:BORDER 10
3990 PRINT#1,"Name:";TAB(40);"Bezeichnung:"
4000 PRINT#1:MOVE 0,374:DRAW 639,374,1
4010 FOR i=3 TO 22
4020 PRINT#1,iv$;USING " ##";i-2;:PRINT#1,iv$;CHR$(133),,,
4030 NEXT i
4040 MOVE 0,200:DRAW 639,200,1
4050 PRINT#1:PRINT#1,TAB(6);" Kalorien ";
4060 PRINT#1,SPC(6);"Joule ";
4070 PRINT#1,SPC(6);" DM   ";
4080 PRINT#1,"Personen";SPC(24);"Getraenke"
4090 MOVE 0,166:DRAW 639,166,1:LOCATE#2,1,1
4100 GOSUB 4690
4110 BORDER 13:RETURN
4120 '
4130 REM ************************************************
4140 REM *        Ausgabe der Daten in die Maske        *
4150 REM ************************************************
4160 IF z= 1 THEN zei= 1:spa= 7:GOTO 4300
4170 IF z= 2 THEN zei= 1:spa=53:GOTO 4300
4180 IF z=29 THEN zei=14:spa= 1:GOTO 4320
4190 IF z=30 THEN zei=14:spa=28:GOTO 4300
4200 IF z=31 THEN zei=14:spa=38:GOTO 4300
```

```
4210 IF z=32 THEN zei=14:spa=52:GOTO 4300
4220 IF z> 2 AND z<23 THEN 4240
4230 IF z>22 AND z<29 THEN 4400
4240 zei=2+INT((z-1)/2):spa=5
4250 IF z/2=INT(z/2) THEN spa=44
4260 LOCATE#1,spa,zei
4270 PRINT#1,LEFT$(e$(z),4);l$;MID$(e$(z),5,2);
4280 PRINT#1,l$;RIGHT$(e$(z),17)
4290 RETURN
4300 LOCATE#1,spa,zei:PRINT#1,e$(z)
4310 RETURN
4320 LOCATE#1,spa,zei:PRINT#1,e$(z)
4330 j$="":FOR i=1 TO fl(z)
4340 j1$=MID$(e$(z),i,1)
4350 IF j1$=p$ THEN j$=j$+"0":ELSE j$=j$+j1$
4360 NEXT i
4370 joule=VAL(j$)*4.18
4380 LOCATE#1,15,14:PRINT#1,DEC$((joule,"######")
4390 RETURN
4400 LOCATE#2,1,z-22:PRINT#2,e$(z)
4410 RETURN
4420 '
4430 REM ************************************************
4440 REM *              Daten in Puffer legen           *
4450 REM ************************************************
4460 PRINT " >Bitte warten ....."
4470 adr=ba1
4480 FOR i=1 TO 32
4490 FOR j=1 TO fl(i)
4500 POKE adr,ASC(MID$(e$(i),j,1))
4510 adr=adr+1:NEXT j
4520 NEXT i:BORDER 13:RETURN
4530 adr=ba2:BORDER 10
4540 FOR i=0 TO 5
4550 POKE §e$(23+i),80:POKE §e$(23+i)+1,FN lb(adr)
4560 POKE §e$(23+i)+2,FN hb(adr)
4570 adr=adr+80:NEXT i
4580 BORDER 13:RETURN
4590 '
4600 adr=ba1
4610 FOR i=1 TO 32
4620 POKE §e$(i),fl(i):POKE §e$(i)+1,FN lb(adr)
4630 POKE §e$(i)+2,FN hb(adr):adr=adr+fl(i)
4640 NEXT i:RETURN
4650 '
4660 REM ************************************************
4670 REM *               Variablen loeschen             *
4680 REM ************************************************
4690 BORDER 10
```

```
4700 FOR z=1 TO 32:e$(z)=STRING$(fl(z),p$)
4710 GOSUB 4130
4720 NEXT z:BORDER 13:RETURN
4730 '
4740 FOR z=23 TO 28::e$(z)=STRING$(fl(z),p$)
4750 GOSUB 4130:NEXT z: RETURN
4760 '
4770 REM ************************************************
4780 REM *              Variablen ausgeben              *
4790 REM ************************************************
4800 FOR z= 1 TO 22:GOSUB 4130:NEXT z
4810 FOR z=29 TO 32:GOSUB 4130:NEXT z
4820 FOR z=23 TO 28:GOSUB 4130:NEXT z
4830 RETURN
4840 '
4850 REM ************************************************
4860 REM *                  Warnton ausgeben            *
4870 REM ************************************************
4880 FOR i=1 TO 10:SOUND 1,100/i,5
4890 FOR j=1 TO 50:NEXT j:NEXT i
4900 FOR i=1 TO 3000:NEXT i
4910 RETURN
4920 '
4930 REM ************************************ M.W.Thoma **
4940 REM *                                              *
4950 REM *        Direktzugriff auf die Diskette        *
4960 REM *                                              *
4970 REM *     Lesen: CALL &A000,DRIVE,SPUR,SEKTOR,PUFFER  *
4980 REM * Schreiben: CALL &A00C,DRIVE,SPUR,SEKTOR,PUFFER  *
4990 REM *                                              *
5000 REM ************************************************
5010 summe=0:FOR i=&A000 TO &A086
5020 READ a$:a=VAL("&"+a$):POKE i,a:summe=summe+a
5030 NEXT i
5040 IF summe=16125 THEN RETURN
5050 PRINT "Fehler in den DATA-Zeilen":END
5060 '
5070 DATA CD,18,A0,3E,84,32,85,A0,CD,5A,A0,C9,CD,18,A0,3E
5080 DATA 85,32,85,A0,CD,5A,A0,C9,F5,3E,00,32,86,A0,F1,FE
5090 DATA 04,C2,76,A0,DD,7E,07,DD,7E,06,FE,02,D2,76,A0,32
5100 DATA 7D,A0,DD,7E,05,DD,7E,04,FE,28,D2,76,A0,32,7E,A0
5110 DATA DD,7E,03,DD,7E,02,FE,0A,D2,76,A0,CE,40,32,7F,A0
5120 DATA DD,66,01,DD,6E,00,22,80,A0,C9,21,85,A0,CD,D4,BC
5130 DATA 22,82,A0,79,32,84,A0,21,7D,A0,5E,23,56,23,4E,2A
5140 DATA 80,A0,DF,82,A0,C9,E1,3E,FF,32,86,A0,C9,00,00,00
5150 DATA 00,00,00,00,00,00,00,00
5160 REM ************************************************
```

Register

Hüthig

Manfred Walter Thoma

CPC 464/664 - Praxis

Band 3: CP/M

1986, ca. 200 S., kart.,
ca. DM 35,—
ISBN 3-7785-1261-7

Durch seine CP/M-Fähigkeit eröffnet der Schneider CPC-464/664 dem Benutzer eine neue Welt vielschichtiger Möglichkeiten.

Schritt für Schritt leitet dieses Buch den Leser hin zur selbständigen und problemlosen Benutzung des Betriebsystems CP/M. Es ist dabei so konzipiert worden, daß es auch wie ein Nachschlagewerk benutzt werden kann.

Der erste Teil des Buches beschäftigt sich umfassend mit den CP/M-Kommandos sowie mit allen 22 CP/M-Dienstprogrammen (SETUP, SUBMIT etc.).

Im zweiten Teil des Buches findet eine grundlegende Einführung in die interne Arbeitsweise des Betriebssystems und in die Programmierung unter CP/M statt. Sämtliche BDOS-Routinen werden detailiert an vielen Beispielprogrammen erklärt. An einigen umfangreicheren Programmen (DISKMON, HELPER, DUMP etc.) wird die praktische Anwendung der BDOS-Routinen vertieft.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Thomas Tai

Hüthig

Computer-Abenteuer auf dem CPC 464/664

1986, ca. 150 S., zahlr. Abb.,
kart., ca. DM 35,—
ISBN 3-7785-1235-8

Adventures erfreuten sich immer schon großer Beliebtheit und auch der CPC 464/664 Besitzer kann aus einem unüberschaubaren Angebot solcher Spiele wählen.

Dieses Buch zeigt dem Leser den Weg, wie er Adventures schneller lösen kann und auftretende Probleme überwindet. Es gibt Tips für immer wiederkehrende Situationen und gewährt einen Blick in die Karten der Adventureprofis, die ein Spiel oft schon nach einer Woche gelöst haben. Die abgebildeten und kopierbaren Entwurfsblätter dienen zur leichten Erstellung eines Lageplans und zur übersichtlichen Aufzeichnung aller wichtigen Spieldaten.

Verzweifelte Spieler werden die vollständigen Lösungen von drei populären Grafikadventures begrüßen, wobei die Anwendung der Entwurfsblätter noch einmal ganz deutlich wird. Zu den sehr komplexen Spielen DUN DURACH und LORDS OF MIDNIGHT werden Tips und Erläuterungen gegeben, so daß der Reiz dieser für die Ewigkeit bestimmten Programme nicht verloren geht.

Den Schluß bildet eine Anleitung zur Erstellung eigener Computer-Abenteuer mit dem berühmten QUILL ADVENTUREWRITER.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Manfred Walter Thoma

CPC 464/664-Praxis

Band 1: Graphik

1985, 188 S., zahlr. Abb., kart.,
DM 34,—
ISBN 3-7785-1149-1

Der erste Band des dreibändigen Werkes beschäftigt sich schwerpunktmäßig mit den Graphikmöglichkeiten des Schneider CPC 464/664. Es ist besonders darauf geachtet worden, daß alle behandelten Probleme und Programmiertechniken einen konkreten Bezug haben. Alle Programme sind aus einem konkreten Bereich entwickelt und mit den behandelten Befehlen realisiert worden.
Neben unzähligen Beispielen findet der Benutzer viele Programme und Unterprogramme, die seine Arbeit mit dem Schneider CPC 464/664 erheblich erleichert. Gezeigt werden Programme von der Multicolorgraphik bis hin zum kompletten CAD-System, vom Tortendiagramm bis zum 3D-Funktionspilot, von der 3-dimensionalen Wertdarstellung bis zur Analoguhr. Alle Programme sind so ausgelegt, daß sie ohne große Schwierigkeiten auch speziellen Wünschen angepaßt werden können. Trotzdem handelt es sich bei diesem Buch nicht um eine Programmsammlung im eigentlichen Sinne. Es ist vielmehr ein Arbeits- und Handbuch, welches durch praktische Anwendungen (Programme) ergänzt ist, die den professionellen Charakter des Schneider CPC 464/664 deutlich unterstreichen.
Neben der Graphik ist weiterhin die Centronics-Schnittstelle mit mehreren Hardcopy-Programmen, der Joystick, die Tastatur und die Interruptsteuerung ausführlich behandelt. Die Kassetten- und Diskettenbedienung findet ebenso Platz wie alle Besonderheiten des BASIC-Interpreters.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Anhand von fünf sehr umfangreichen und leistungsstarken Programmen findet im Band 2 der CPC-464/664 Praxis-Reihe eine grundlegende Darstellung der wichtigsten Methoden zur Datenverarbeitung der wichtigsten Methoden zur Datenverwaltung statt. Neben der Behandlung von Konstanten, der sequentiellen Datei und der RAM-Floppy (relative Adressierung) darf natürlich der Direktzugriff auf Daten nicht fehlen. Mit einem sehr komfortablen Diskettenmonitor wird der Aufbau der Diskette genauer untersucht. Doch auch Such- und Sortierverfahren sowie die Handhabung und Programmierung von Index-Dateien finden hier ihren Platz.

Die erarbeiteten Programme gehen weit über das übliche Niveau hinaus (sie sind, bis auf den Direktzugriff auch mit dem Datenrecorder nutzbar) und lassen sich in der alltäglichen Praxis hervorragend anwenden. Die Programme brauchen durch ihre Qualität einen Vergleich mit kommerziellen Programmen nicht zu scheuen.

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/