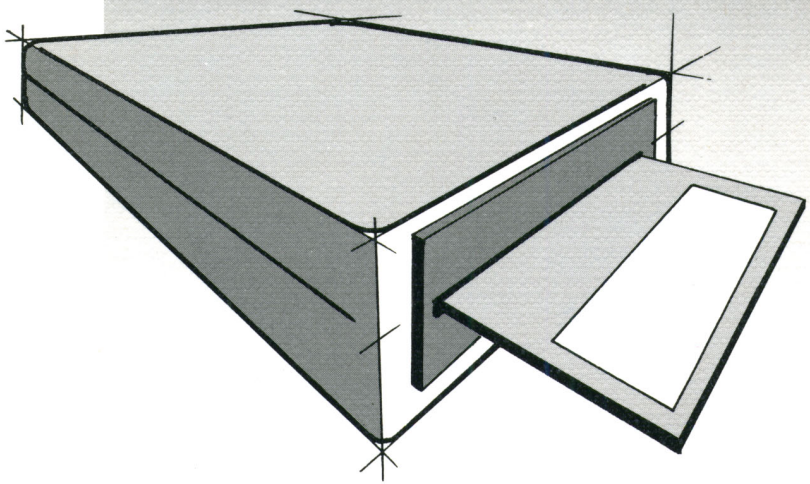
Schönleber

64

Das Commodore Floppy Buch



DATA BECKER

Schönleber

Das Commodore Floppybuch

DATA BECKER

1. Auflage 1988

ISBN 3-89011-269-2

DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Text verarbeitet mit Word 4.0, Microsoft
Ausgedruckt mit Hewlett Packard LaserJet II
Druck und Verarbeitung Graf und Pflügge, Düsseldorf

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

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

Für Birgit

Vorwort

Obwohl sich durch massive Verkaufserfolge inzwischen der PC-Standard im kommerziellen Bereich durchgesetzt hat, so ist es dennoch gerechtfertigt, ein Buch über Diskettenlaufwerke für Homecomputer zu schreiben. Gerade die Homecomputerserien von Commodore sind ein ideales Werkzeug zum Lernen und Experimentieren. Darüberhinaus sind sie, bei richtiger Verwendung, wesentlich flexibler und kostengünstiger als die großen und meist nicht sehr handlichen PCs. Vor allem die zu diesen Homecomputern dazugehörigen Diskettenlaufwerke (zum Beispiel aus der 15xx-Serie) sind, obwohl von vielen als Spielzeug und als zu langsam verpönt, doch sehr viel intelligenter, als man auf den ersten Blick vermuten würde. Durch eine laufwerkseigene CPU sind diese Peripheriegeräte in der Lage, fast selbständig bestimmte Operationen durchzuführen, während der angeschlossene Rechner sich schon mit anderen Aufgaben beschäftigt.

Dieses Buch soll denjenigen helfen, die sich gerade mit solchen Konfigurationen einführend beschäftigen wollen und ihre ersten Erfahrungen mit einem Commodore Homecomputer sammeln. Da zu einem vernünftigen Betrieb auch die Benutzung von Diskettenlaufwerken gehört, will dieses Buch sich gerade mit diesem Thema auseinandersetzen, da das Erlernen einer Programmiersprache, wie z.B. BASIC oder Pascal, meist beim Diskettenbetrieb aufhört. Richtiges Programmieren fängt aber oft erst an dem Punkt an, wo man gezwungen ist, große Datenmengen mit Hilfe eines externen Speichermediums (Diskette, Festplatte) zu verwalten.

Als Ergänzung zu den mitgelieferten Handbüchern bespricht dieses Buch den sinnvollen Einsatz von Diskettenlaufwerken der 15xx-Serie von Commodore, so daß Sie nach der Lektüre dieses Buches in der Lage sind, die elementaren Begriffe der Datenverwaltung zu verstehen und anzuwenden.

Um aus dem Buch den größtmöglichen Nutzen zu ziehen, sollten Sie über einige Kenntnisse in der Programmiersprache BASIC (Commodore Dialekt) verfügen, da die Beispielprogramme alle in Commodore-BASIC formuliert sind. Falls Sie solche Kenntnisse noch nicht haben, sei auf die umfangreiche Literatur verwiesen, die sich mit diesem Thema beschäftigt[1]. Im Anhang dieses Buches befindet sich ein kleines Literaturverzeichnis, mit dessen Hilfe Sie sich einen kleinen Überblick verschaffen können.

Ich möchte noch erwähnen, daß man auch die Beschäftigung mit dem Computer nicht allein durch Lesen lernt, sondern daß die Praxis ein wichtiger Teil des Lernprozesses ist. Versuchen Sie, jede Tatsache, die in diesem Buch beschrieben ist, und jedes Programm auf Ihrem Gerät nachzuvollziehen, und arbeiten Sie erst weiter, wenn Sie den jeweiligen Teil verstanden haben. Um das Buch effektiv durchzuarbeiten, sollten Sie Absatz für Absatz vorgehen. Haben Sie einen Absatz nicht verstanden, wiederholen Sie ihn oder die vorhergehenden. Das ist die beste Methode, alles in diesem Buch zu verstehen. Viel Vergnügen!

Zum Schluß möchte ich mich bei Cornelius Keck bedanken, dessen vielseitige Anregungen mir bei der Erstellung dieses Buches unentbehrlich waren. Auch André "Mikado" Busche möchte ich für seine konstruktiven Antworten ("Weiß nich...!") danken. Weiterhin ganz, ganz herzlichen Dank an Martina, die sich der schwierigen Aufgabe des Korrekturlesens annahm. Ganz besondere Dankbarkeit gilt Herrn Anschütz, der mir in vielen Gesprächen half, Lehren und Lernen besser zu verstehen.

Kiel, den 23. November 1988

---

1 Schönleber: Hitchhacker's Guide to BASIC, Kiel 1987, Verlag Claus Schönleber

Vereinbarungen:

Um die Befehle in möglichst allgemeiner Form darzustellen, ist es notwendig, statt expliziter Angaben, die das Objekt bezeichnen, auf das sich der Befehl bezieht, allgemeine Angaben über die Art der einzusetzenden Angaben zu machen. Ein Beispiel:

Um eine Datei namens "TESTPROGRAMM" zu laden, muß man den Befehl

```
LOAD "TESTPROGRAMM",8
```

eingeben. Allerdings möchte man auch die allgemeine Form des Befehls angeben, da ja bei weitem nicht alle Dateien "TEST-PROGRAMM" heißen. Deswegen muß für die Angabe "TEST-PROGRAMM", die ja nur ein Beispiel darstellt, etwas gefunden werden, das sich in der Form von dem Beispiel unterscheidet. Man braucht einen Platzhalter, einen "Dummy", der zwar bezeichnet, was für ihn stehen soll, aber eine von der einzusetzenden Angabe unterscheidbare Form besitzt. Dafür benutzt man Metazeichen. Metazeichen werden beim Einsatz des Befehls nicht mit eingegeben. Beispiel einer allgemeinen Form für LOAD:

```
LOAD "<dateiname>",8
```

Dabei ist die Zeichenreihe '<dateiname>' der Platzhalter, die spitzen Klammern schließen den Bezeichner für die eigentlich einzusetzende Angabe ein. Der Platzhalter wird mitsamt den Klammern durch die eigentliche Angabe ersetzt. Wenn also in einer allgemeinen Form so etwas auftritt, dann lesen Sie, was in den spitzen Klammern steht und ersetzen alles durch die eigentliche Angabe. Alle anderen Zeichen vor und nach diesen Platzhaltern müssen ganz genauso eingegeben werden, wie sie angeführt sind.

Da dieses Buch verschiedene BASIC-Versionen benutzt, ist es notwendig, die verschiedenen Befehlsversionen zu unterscheiden. Tritt so ein Fall auf, wird die jeweilige Version eingeleitet mit

BASIC ab/bis x.x:

wobei 'xx' für die einzelnen Laufwerksversionen steht. Beendet wird die Version mit einer Sternreihe:

*****

Durch "BASIC ab/bis x.x:" und "*****" wird also die jeweilige Version eindeutig geklammert.

Als Grundregel gilt: Läßt die DOS-Version eines Laufwerks eine Funktion zu, so ist sie, unabhängig von der BASIC-Version, immer wenigstens durch den OPEN-Befehl mit geeignetem Kommando ausführbar.

Inhaltsverzeichnis

1. Einleitung

"Das Schönste, was ich je gehört habe, war der Klang der Speichertrommel einer IBM 650, als der Computer starb."

Vallee[2]

Wir erfahren etwas über den Ursprung der Diskette, was eine "Schlappscheibe" ist, was für Disketten es gibt, wie sie aussehen und was man wie auf ihnen speichert. Darüber hinaus wird auf die Geräte eingegangen, mit denen man bei Commodore-Disketten rotieren läßt.

Wozu Disketten, Festplatten oder Cassetten?

Ein Computer ohne ein dazugehöriges Programm wäre wie ein teurer Sportwagen ohne Motor: schön anzusehen, aber völlig nutzlos. Das Dumme an Computern ist jedoch, daß sie ein Programm nur "kennen" (im Speicher behalten können), solange sie angeschaltet bleiben. Sobald man einen Computer ausschaltet, "vergißt" er das Programm.

Da man als Anwender natürlich Besseres zu tun hat, als vor jeder Sitzung ein benötigtes Programm erst einzutippen, gibt es Speichermedien, auf denen die Programme permanent gespeichert werden können und von wo man wieder Programme in den Speicher laden kann. Selbstverständlich kann man natürlich auch Programme auf solchen Speichermedien verändern oder wieder löschen.

Wie war das früher?

---

2 Vallee: Computernetze, Träume und Alpträume von einer neuen Welt, Rowohlt 1984

In den Anfängen der Computerei wurden Lochkarten oder Lochstreifen als Speichermedien benutzt. Aber sie waren nicht sehr flexibel zu handhaben. Es gibt noch alte Programmierer, die davon berichten können, wieviel Schweiß und Tränen es gekostet hat, einen Stapel Lochkarten, der versehentlich auf den Boden gefallen war und sich streng nach den Gesetzen der Statistik verteilt hat, wieder aufzusammeln. Um die richtige Reihenfolge wieder herzustellen, gab es Maschinen, die solche Karten sortierten, wenn solch ein Mißgeschick passiert war.

Magnetbänder waren dann die große Neuerung. Es paßten viel mehr Daten drauf, und man kam viel schneller an die Daten heran. Allerdings fand man auch hier wieder negative Eigenschaften: Die Magnetbandgeräte waren sehr groß und damit nicht sehr handlich; und nachdem man sich an die Bänder gewöhnt hatte, war den Programmierern auch das Magnetband etwas zu langsam.

Plattenlaufwerke waren die Lösung. Das sind magnetisch beschichtete Platten, mit denen man viel schneller auf die Daten zugreifen kann.

Wie ist das heute?

Heute werden Lochkarten und Lochstreifen nicht mehr zur Programmierung verwendet. Zur Organisation eines Lagerbestandes sind Lochkarten jedoch immer noch eine wertvolle Hilfe. Magnetbänder sind noch heute in Benutzung. Man muß zwar etwas länger warten, aber es passen noch immer genügend Daten auf die Bänder, und sie lassen sich im Übrigen besser archivieren als Platten. Allerdings können sich nur Betreiber von Großrechnern Bandbetrieb leisten, da die Geräte dazu immer noch ziemlich groß und damit auch teuer sind.

Magnetbänder könnten in Zukunft von optischen Platten abgelöst werden. Sie sind sehr klein, billig und, was die erfaßbare Datenmenge betrifft, mit den Bändern schon konkurrenzfähig. Allerdings ist diese Technik zwar schon serienreif, sie ist aber

wohl noch nicht marktreif. Optische Platten, die man nur einmal beschreiben und beliebig oft lesen kann, sind schon längere Zeit auf dem Markt (CD-ROMs).

Mit der Einführung von Disketten oder Floppy Disks ("Weiche Scheiben", Spaßvögel nennen sie auch "Schlappscheiben") wurde der Plattenbetrieb sehr preiswert und für jedermann erschwinglich. Die Diskette hat den Markt für Micro- und Minicomputer erobert.

Was für Disketten gibt es?

Das älteste Format ist die 8-Zoll-Diskette. Format bedeutet: Diskettendurchmesser. Inzwischen ist die etwas kleinere 5¼-Zoll-Diskette (in Worten: Fünfeinviertelzolldiskette) Standard. Aber es geht noch kleiner! 3½-Zoll-Disketten setzen sich immer mehr als neuer Standard durch. Die Einheit "Zoll" wird mit zwei hochgestellten Strichen symbolisiert: ("). In diesem Buch wollen wir uns weitestgehend auf das 5¼"-Format beschränken.

Wie werden die Daten auf einer Diskette organisiert?

Die Diskette wird in einzelne Spuren aufgeteilt. Das sind konzentrische Kreise und nicht eine Spur, die, wie auf einer Schallplatte, spiralförmig verläuft. Weiterhin wird die Diskette noch in Sektoren aufgeteilt wie eine Torte. Das Ganze ist natürlich nicht zu sehen, genausowenig wie man Musik auf Tonbandcassetten sehen kann.

Die Anordnung von Spurenzahl und Sektorzahl nach diesem Prinzip kann man natürlich nach Belieben variieren. Leider heißt auch dieses Kriterium - wie die Größenangabe für Disketten - Format. Damit sind aber die Aufzeichnungsparameter und -verfahren gemeint. Aus dem Zusammenhang muß dann klar werden, welche Interpretation gemeint ist.

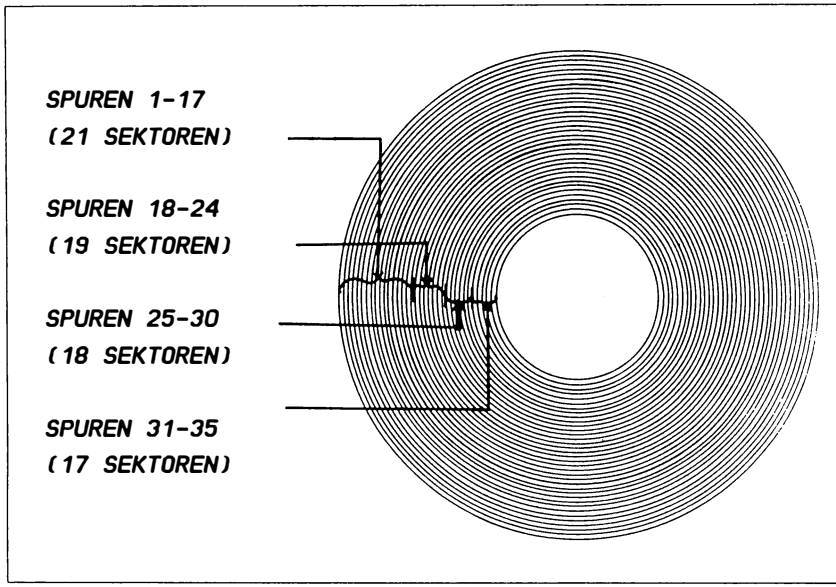


Abb. 1: Aufteilung einer Diskette

Es ist klar, daß man sich auch in dieser Beziehung wenig um einen Standard bemüht hat. So gibt es nicht nur von jedem Hersteller ein anderes Aufzeichnungsformat, sondern oft unterscheiden sich sogar die Aufzeichnungsformate verschiedener Rechnertypen desselben Herstellers. Große Unterschiede bestehen zwischen Homecomputern und Personalcomputern. In der obigen Abbildung wird die Aufteilung für Commodore-Diskettenlaufwerke der Typen 1541 und 1570/71 gezeigt.

Für so viele unterschiedliche Anwendungen gibt es natürlich auch verschiedene Qualitätsstufen bei den verkauften Disketten. Es ergeben sich fünf große Gruppen bei den 5¼"-Disketten, die sich anhand ihrer Einsatzgebiete gravierend voneinander unterscheiden:

Einseitige Disketten mit normaler Schreibdichte:

Reichen für alle Laufwerke bis maximal 40 Spuren, die nur einseitig schreiben, z.B. für Commodore 1541.

Einseitige Disketten mit doppelter Schreibdichte:

Können für Laufwerke mit 40 Spuren benutzt werden, einseitig beschreibbar. Bezeichnung: SSDD oder 1D.

Zweiseitige Disketten mit einfacher Schreibdichte:

Für Laufwerke bis 40 Spuren benutzbar, beidseitig beschreibbar. Sogenannte 48 tpi Disketten werden für PC-kompatible Rechner benötigt. (tpi = tracks per inch; d.h. Spuren pro Zoll). Bezeichnung: DSSD oder 2D-48tpi.

Zweiseitige Disketten mit doppelter Schreibdichte:

Können für Laufwerke bis 80 Spuren benutzt werden, beidseitig beschreibbar. Sogenannte 96 tpi Disketten. (tpi = tracks per inch; d.h. Spuren pro Zoll). Bezeichnung: DSDD oder 2D-96tpi.

Zweiseitige Disketten mit hoher Schreibdichte:

Müssen für Laufwerke mit über 80 Spuren benutzt werden, beidseitig beschreibbar. Sogenannte HD Disketten (HD = High Density; besonders hohe Schreibdichte) werden für die 1,2 MByte-Laufwerke der AT-kompatiblen Computer benötigt. Sie können nicht in normalen Laufwerken benutzt werden. Bezeichnung: HD

Einseitig heißt übrigens, daß die Diskette zwar zweiseitig hergestellt wurde, aber in der Endkontrolle auf einer Seite entweder nicht geprüft wurde oder durchgefallen ist. Deswegen ist die verbreitete Sitte, einseitig deklarierte Disketten zweiseitig zu verwenden, mit Vorsicht zu genießen. Für den privaten Gebrauch kann man solche Disketten nach eingehender Prüfung durch entsprechende Programme durchaus zweiseitig benutzen. Für wichtige und wertvolle Daten, vor allem im Geschäftsbereich, ist es jedoch sicherlich billiger, die etwas teureren Dis-

ketten zu benutzen, als durch Datenverlust weit mehr Geld zu verlieren. Für unsere Zwecke ist es ausreichend, Disketten mit der Bezeichnung 2D-48tpi zu benutzen.

Commodore-Diskettenlaufwerke

Auch bei Commodore gibt es verschiedene Typen des Diskettenformats und somit auch verschiedene Laufwerke. Allerdings können Commodore-Disketten, die auf einem Laufwerkstyp formatiert wurden, von den anderen Laufwerken oft gelesen werden. In diesem Buch wollen wir uns mit Diskettenlaufwerken für Homecomputer dieser Firma beschäftigen, als da wären:

Laufwerkstyp	passender Computer
1541, 1541c, 1541-II	(C116,C16,C64, Plus 4,C128)
1570/71	(C116,C16,C64, Plus 4,C128)
1551	(Nur C16,C116,Plus 4)
1581	(C116,C16,C64, Plus 4,C128)

Die beiden letzten sollen hier nur am Rande besprochen werden. Sehen wir uns zunächst einmal an, worin sich diese Laufwerke optisch und mechanisch unterscheiden:

Alle Laufwerke, bis auf die 1581 und 1541-II, welche beide ein externes Netzgerät besitzen, haben ein langes Gehäuse mit eingebautem Netzteil. Dabei haben die Laufwerke 1541, 1541c, 1551 und 1570 das klassische Gehäuse, wie es fast alle C64-Fans kennen. Modernes, fast futuristisches Design besitzen die 1571 und die 1581. Allerdings haben nur das 1541 und das 1570 einen Knebelverschluß (Drehverschluß). Die Laufwerke 1541c, 1551 und 1571 haben einen Hebelverschluß. Die Ausnahme ist das 1581. Dieses Laufwerk hat den üblichen Verschluß für 3½"-Laufwerke (Auswurfknopf). Das Diskettenformat der übrigen Laufwerke ist das gewohnte 5¼"-Format. Der Farbton der Laufwerkseinschübe ist hell bei 1541c und 1570, dunkel bei 1541 und 1551. Die 1571 und 1581 besitzen ein einheitliches Gehäuse in hellem Farbton.

2. Vom Auspacken bis zum Anschalten

"The first thing you will need to do with your disk drive is unpack it."

Commodore Electronics Ltd.[3]

Das Auspacken

Vor dem Einsatz unseres neuen Commodore-Laufwerks haben die Hersteller die Verpackung gesetzt. Also machen wir uns an die Kartonage. Wir finden folgende Einzelteile:

- Das Laufwerk an sich
- Bei Laufwerken mit externem Netzteil: Netzteil
- Eine beigelegte Diskette
- Ein Kabel
- Ein Handbuch (oder mehrere; manchmal liegt neben dem englisch-sprachigen Original auch noch die Übersetzung.)
- Die Garantiekarte (natürlich)

Vermissen Sie etwas, haben Sie beim Einkauf wohl nicht aufgepaßt. Dann sollten Sie vor dem Weiterlesen sofort wieder zu Ihrem Händler gehen und die fehlenden Teile reklamieren.

Haben Sie alles gefunden, können wir fortfahren.

---

3 Commodore Electronic Ltd.: 1551 disk drive User's Guide

Die Installation des Laufwerks

Betrachten wir zunächst den Hauptteil des Pakets. Je nach Ausführung ist es ein mehr oder weniger längliches Gehäuse, das auf der einen Seite einen Schlitz mit Verschluß besitzt (Vorderseite) und auf der anderen Seite den Kabelanschluß (Rückseite).

Wir betrachten zunächst die Laufwerke mit eingebautem Netzteil. Der Stecker des Netzkabels wird in eine Netzsteckdose gesteckt. Dann wird das beigelegte Verbindungskabel Laufwerk-Computer montiert. Hier gibt es verschiedene Möglichkeiten.

1551:
Hier hat das Verbindungskabel ein sehr "dickes Ende". Eine Art "Cassette" bildet hier den Stecker, der in den Computer gesteckt werden soll. Als Anschlußbuchse dient hier (C116, C16, Plus4) der sogenannte Memory Expansion Port. Die "Cassette" wird eingesteckt, und das Laufwerk ist verkabelt. Will man ein weiteres Laufwerk anschließen, so steckt man dessen Stecker einfach hinten in den ersten Stecker des Diskettenlaufwerks, der dafür vorbereitet ist.

1541 und ähnliche:
Das Verbindungskabel zum Computer sieht wie ein normales DIN-Stereo-Kabel aus. Das ist das Kabel für den seriellen Bus. Es gibt im Computer (C64) zwei Buchsen für dieses Kabel. Welche der beiden Buchsen wir verwenden, ist völlig gleichgültig.

Wir schließen das Laufwerk nun an den Rechner an und nehmen es in Betrieb:

1. Achten Sie darauf, daß der Computer auf jeden Fall ausgeschaltet ist.

2. Stellen Sie das Laufwerk am gewünschten Platz auf. Allerdings ist die Entfernung vom Computer durch die Länge des Verbindungskabels festgelegt.

3. Verbinden Sie Laufwerk und Computer mit dem beigelegten Kabel.

4. Schließen Sie das Laufwerk mit dem Netzkabel an das Stromnetz an. Achten Sie bei der Gelegenheit darauf, daß das Laufwerk dabei ausgeschaltet ist.

5. Schauen Sie auf die Klappe des Laufwerks: sie muß offen sein.

6. Jetzt, endlich, dürfen Sie das Laufwerk einschalten!

7. Zu guter Letzt wird der Computer eingeschaltet.

Bei dieser Gelegenheit ein kleiner Tip:

Beim Einschalten eines Computers und seiner Peripherie (alle an den eigentlichen Rechner angeschlossenen Geräte) wird immer erst die Peripherie, also Laufwerke, Monitor, etc. eingeschaltet, erst dann der Computer. Damit wird das gefürchtete Einschaltknacken vom Rechner ferngehalten, das ihn beschädigen könnte. An dieser Stelle können Sie eine Diskette in das Laufwerk schieben. Aber benutzen Sie nicht gerade eine wichtige Diskette. Denn trotz aller Endkontrollen und Prüfungen kann es durchaus sein, daß Sie versehentlich ein defektes Laufwerk bekommen haben. Und das sollten Sie nicht gerade mit einer wertvollen Diskette austesten.

Probieren Sie zuerst am besten eine leere Diskette aus, die Sie mit Hilfe der Anleitung oder dieses Buches formatieren. Haben Sie eine formatierte Diskette zur Verfügung, so können Sie schon mal versuchen, ob sich ein BASIC-Programm abspeichern läßt. Sollten Sie noch nicht wissen, wie das funktioniert, so wird das auf den nächsten Seiten erklärt. Blättern Sie dann einfach etwas vor.

Wenn etwas nicht funktioniert

Wir haben das Laufwerk angeschlossen, alles gemacht, was in der Anleitung oder in diesem Buch steht, und trotzdem funktioniert nichts. Werfen Sie weder dieses Buch noch die Anleitung in den Mülleimer, und bewahren Sie Ruhe. Wir werden Schritt für Schritt vorgehen und den Fehler suchen.

Checkliste:

Allgemeine Vorprüfung

Ist das Laufwerk aus europäischer Fertigung (220V, 50Hz)?

Ist das Verbindungskabel Laufwerk-Computer richtig eingesteckt?

Ist das Netzkabel richtig gesteckt?

Paßt das Laufwerk zum Rechnertyp?

Anschaltprüfung

Leuchtet beim Anschalten die Bereitschaftsanzeige auf?

(rote Leuchtdiode bei 1571 und 1581, grüne Leuchtdiode bei 1541, 1551, 1570)

Bei Nichtleuchten:

Hat die Steckdose Spannung?

Ist die Sicherung für die Steckdose in Ordnung?

Ist die Sicherung im Laufwerk in Ordnung?

Leuchtet die Diskettenfunktionsanzeige nach dem Anschalten kurz auf?

Funktionsprüfung

Funktioniert das Laden des Catalogs (Inhaltsverzeichnis)?

Läuft der Funktionstest auf der Demodiskette einwandfrei?

Können Sie in dieser Checkliste alle Fragen mit "ja" beantworten, ist das Laufwerk in Ordnung. Beantworten Sie irgendeine Frage mit "nein", so können Sie anhand der Frage entscheiden, ob Sie selber in der Lage sind, die Fehlerquelle zu beseitigen oder ob Sie ein Fachgeschäft zu Rate ziehen müssen. Dabei sei aber auf die zwar selbstverständliche, aber nicht sehr verbreitete Erkenntnis hingewiesen, daß bei einem Eingriff in das Laufwerk, der nicht von einem Vertragshändler (bzw. -werkstatt) vorgenommen wird, die Garantie verloren geht. Und damit meine ich nicht nur die gesetzliche Garantieleistung des Herstellers, sondern auch die Garantie, daß das Laufwerk hinterher auch noch korrekt "läuft"!

Im Zweifelsfalle also immer fachlichen und kompetenten Rat einholen!

Kleine Fehleranalyse

Wenn die Bereitschaftsanzeige nicht leuchtet, haben Sie entweder vergessen, das Netzkabel anzuschließen oder ein defektes Netzkabel oder das Laufwerk nicht angeschaltet oder eine defekte Sicherung im Netzteil des Laufwerks oder eine defekte Haussicherung.

Abhilfe: Netzkabel nach Anleitung installieren oder Netzkabel austauschen oder Laufwerk anschalten oder Sicherung im Laufwerk ersetzen oder die Haussicherung wieder auf Durchlaß schalten.

Wenn die Diskettenfunktionsanzeige flackert, bevor Sie irgendetwas getan haben, dann hat der Laufwerksselbsttest, der beim Anschalten automatisch gestartet wird, einen Fehler diagnostiziert.

Abhilfe: Schalten Sie das Laufwerk für einige Minuten aus, und schalten Sie dann wieder ein. Tritt der Fehler wieder auf, wiederholen Sie den vorangegangenen Schritt mit abgezogenem Computerverbindungskabel. Tritt der Fehler wieder auf, sollten Sie den Service einer Vertragswerkstatt in Anspruch nehmen.

Wenn sich ein BASIC-Programm nicht laden läßt und die Diskettenfunktionsanzeige erst flackert und dann blinkt, dann konnte das Laufwerk nicht korrekt auf die Daten zugreifen.

Abhilfe: Lesen Sie den Fehlerkanal bzw. die Fehlervariablen, und schauen Sie in der Fehlerliste in der Anleitung oder im Anhang dieses Buches nach der Erklärung. Meist läßt sich der Fehler durch die Erklärung schnell beheben. Oft ist aber auch die Diskette physisch defekt. Dann ist meist nichts mehr zu retten, und Sie haben hoffentlich eine Sicherheitskopie Ihrer Diskette, auf die Sie nun zurückgreifen können.

Hinweis: Sollten Sie noch nicht wissen, wie man den "Fehlerkanal ausliest", so schauen Sie bitte im Kapitel 6 nach.

Wenn beim Laden eines Programms die Meldung "DEVICE NOT PRESENT" erscheint, dann haben Sie vergessen, das Laufwerk anzuschalten oder das Verbindungskabel nicht ordentlich gesteckt oder ein defektes Kabel.

Abhilfe: Laufwerk anschalten oder das Kabel auf ordnungsgemäße Verbindung überprüfen oder ein neues Kabel kaufen.

Wenn sich ein Programm nicht laden läßt und auch keine Fehlermeldung erscheint, dann stört höchstwahrscheinlich ein anderes Peripheriegerät oder aber die Datei, in der das Programm stehen soll, ist leer.

Abhilfe: Alle anderen Peripheriegeräte abstöpseln, dann die Handlung wiederholen, bei der der Fehler auftrat.

Tritt dann kein Fehler auf, dann sollten Sie ein Gerät zur Zeit wieder anstöpseln und den vorigen Schritt wiederholen. Tritt der Fehler reproduzierbar bei einem bestimmten Gerät auf, ist der Übeltäter gefunden. Ist die Datei wirklich leer (Es werden im Inhaltsverzeichnis 0 Blocks angezeigt), dann hilft nichts, es steht eben nichts drin. Im übrigen zeigt sich dieses Verhalten auch bei dem Versuch, Maschinenprogramme mit BASIC-Ladebefehlen zu laden.

Wenn sich Disketten von anderen Commodore-Laufwerksbenutzern gar nicht oder nur mangelhaft lesen lassen, dann ist Ihr Laufwerk höchstwahrscheinlich dejustiert.

Abhilfe: Die Fachwerkstadt

Wenn sich alte Disketten nicht mehr lesen lassen, neu beschriebene Disketten dagegen klaglos funktionieren, dann sind Ihre alten Disketten kaputt oder Ihr Laufwerk ist dejustiert.

Abhilfe: Vorbeugend von Zeit zu Zeit Sicherheitskopien von älteren Disketten anfertigen oder die Fachwerkstadt.

Pflege ist die beste Abhilfe

Es kann vorkommen, daß Ihr Laufwerk plötzlich streikt, obwohl wirklich kein sichtbarer Grund vorliegt. Dann ist ihm wahrscheinlich zu heiß geworden. Zwar machen Transistoren nicht mehr so viel Hitze wie alte Röhren; aber wenn man nicht dafür sorgt, daß das Laufwerk genügend belüftet wird, baut sich ein Hitzestau im Gehäuse auf, und verschiedene Bauteile sind nicht mehr arbeitsfähig. Weiterhin sollte man regelmäßig den Schreib-/Lesekopf säubern und die Schienen schmieren. Am besten lassen Sie das den Vertragshändler besorgen, besonders während der Garantiezeit. Danach können Sie das Säubern mit reinem Isopropylalkohol und das Schmieren mit Molybdän (Kein Fett!) besorgen.

Verwenden Sie nur Disketten mit intakter, glatter Oberfläche, damit der Lesekopf nicht beschädigt wird. Außerdem sollten sie leicht laufen. Schwergängige Disketten sollte man meiden. Achten Sie darauf, daß Sie magnetische Dinge vom Schreib-/Lesekopf fernhalten! Er ist empfindlicher als Disketten! Staub, Rauch, Dampf, Feuchtigkeit, usw. sind nicht nur für Disketten tödlich, auch die Laufwerksmechanik mag so etwas nicht. Ein Tip zum Schluß: Nehmen Sie nie eine Diskette aus dem Laufwerk (auch nicht abschalten), wenn das Laufwerk schreibt!

3. Dateien

"Keine Panik!"
Douglas Adams[4]

Hier erfahren wir, wie man es geschafft hat, altmodische Aktenordner in Computer zu stopfen, was ein Aktenschrank mit einer Diskette gemeinsam hat und wie es ein Computer fertigbringt, eine gigantische Menge von Daten zu behalten ohne dabei den Überblick zu verlieren. Zum Schluß wird erklärt, was ein DOS ist und wozu es gut ist.

Vor der Erfindung der EDV war es schon eine Mordsplackerei, die Übersicht über alle Daten zu behalten, die sich so in den Aktenordnern ansammelten. Die Aktenordner türmten sich in den Aktenschränken, und ein guter Bürokrat zeichnete sich dadurch aus, daß er seine Akten nicht nur in Ordnung hielt, sondern auch auswendig wußte, wo jedes einzelne Datum zu finden war. Mit der Einführung der EDV hat sich das eigentlich nicht sonderlich geändert. Nur fällt es jetzt plötzlich leicht, die Übersicht zu behalten, denn der Computer vergißt kein einzelnes seiner Daten. Natürlich gibt es durch die elektronische Datenverarbeitung viel mehr Möglichkeiten, Daten zu manipulieren oder auf Daten zuzugreifen, aber im Prinzip gibt es immer noch Akten und Aktenordner:

- Eine Diskette entspricht einem Aktenordner,
- Eine Datei entspricht einer Akte.

Natürlich gibt es auch einen Fachausdruck für Datei und der ist natürlich englisch: "file", was nichts anderes bedeutet als Akte.

---

4 Douglas Adams: Per Anhalter durch die Galaxis, Ullstein 1988

Man sieht, es wurde mit einem alten Namen ein neuer Begriff geschaffen. Es wurde aber im Prinzip nichts Neues erfunden, man hat nur etwas Altes mittels einer neuen Methode weiterentwickelt. Datenverarbeitung hat man schon seit der Antike betrieben; Man könnte das manuelle Datenverarbeitung nennen (MDV). Mit der Entdeckung der Elektronik gibt es jetzt eben die Elektronische Datenverarbeitung (EDV).

Was ist eine Datei konkret?

Eine Datei ist eine Zusammenfassung von beliebigen Einzeldaten zu einer sinnvollen Gruppe. Eine Datei heißt auch file.

Beispiele:

- Wenn die Einzeldaten Buchstaben des Alphabets sind, ist eine Datei ein Text (Brief-, Buchtext,...).

- Wenn die Einzeldaten Zahlen sind, dann ist eine Datei etwa eine Reihe von Meßdaten eines Experiments oder Daten einer Bilanz.

- Sind die Einzeldaten Zeichen aus einem fest vorgegebenen Zeichenrepertoire mit Regelwerk, das bestimmt, wie die Daten aneinandergereiht werden dürfen, so ist eine Datei ein Programm (Quelltext) in einer Programmiersprache.

Nehmen wir an, wir möchten den kurzen Satz

"DIES IST EIN TEXT"

als Datei auf der Diskette speichern. Dazu werden die einzelnen Buchstaben im Computer in ihren Code umgewandelt. Der Code, nach dem das geschieht, heißt ASCII. Das ist die Abkürzung für:

"American Standard Code of Information Interchange"

und ist nichts anderes als eine Industrienorm. Eine Tabelle dieses Codes finden Sie im Anhang.

Der Grund für diese merkwürdige Umwandlung liegt darin, daß ein Computer eigentlich nur Zahlen "verstehen" und mit ihnen arbeiten kann. Um auch mit Buchstaben oder anderen Zeichen arbeiten zu können, hat man diesen einfach bestimmte Zahlen zugeordnet. Diese Zuordnung ist gerade die Norm ASCII.

Nach der Tabelle können die Buchstaben des Beispieltextes in ihr Zahlenäquivalent umgewandelt werden:

D	68
I	73
E	69
S	83
	32
I	73
S	83
T	84
	32
E	69
I	73
N	78
	32
T	84
E	69
X	88
T	84

Ja, Sie sehen richtig! Auch die Zwischenräume gelten als Buchstaben des Computeralphabets. Sie heißen Leerschritt (engl.: blank oder space) und haben den Code 32. Sie gelten als vollwertiges Zeichen, und auch wenn man es nicht zu sehen glaubt, so ist es doch da, und um es schreiben zu können, muß der Computer auch dafür einen Code wissen.

Damit der Rechner weiß, wann eine Datei zu Ende ist, muß an den Schluß des Textes noch eine Marke gesetzt werden, die das Dateiende anzeigt. Ich möchte dieses Zeichen hier nur mit <EOF> (=End of file, Dateiende) bezeichnen.

Die Datei würde nun so aussehen:

68 73 69 83 32 73 83 84 32 69 73 78 32 84 69 88 84 <EOF>

Nach diesem Prinzip wird eine Datei dann auf Diskette abgespeichert. Ich will nicht verschweigen, daß das Ganze eigentlich noch etwas komplizierter abläuft, aber für unsere Zwecke reicht dieses Prinzip zunächst.

Auf eine Diskette passen nun wesentlich mehr Daten, als in so einer kleinen Datei vorkommen. Man kann deswegen auf einer Diskette mehr als eine Datei anlegen. Deswegen auch der Vergleich mit einem Aktenschrank, da in so einen Schrank ja auch mehr als ein Ordner hineinpaßt.

Wie behält der Rechner die Übersicht?

Um die ganzen Dateien im Griff zu behalten, muß natürlich dafür gesorgt werden, daß über jede einzelne Datei "Buch geführt" wird:

- Es muß für jede Datei eine Bezeichnung geführt werden: Der Dateiname.
- Es muß die Art der Datei bekannt sein: Der Dateityp.
- Es muß die Länge der Datei (Beginn und Anzahl der benutzten Sektoren bzw. Blöcke) bekannt sein.

Für diesen Zweck wird auf der Diskette ein besonderer Bereich freigehalten, der Directory (Inhaltsverzeichnis) heißt. Dort werden die Daten gespeichert, die für die Verwaltung jeder einzelnen Datei notwendig sind. In einem Directory werden alle Dateien über ihren Namen mit den Informationen über Typ und Länge in einer Liste (Inhaltsverzeichnis) geführt. Für das Directory gibt es auf Disketten einen reservierten Bereich.

Wir fassen zusammen:

- Daten werden zu sinnvollen Gruppen zusammengefaßt (Texte, Meßwerte, Programmquelltexte, Programme). Diese Gruppen heißen Dateien.

- Dateien werden auf externen Speichermedien (Diskette, Festplatte, Magnetband) langfristig gespeichert.

- Verwaltet werden diese Dateien mittels besonderer Daten, die im Directory (Inhaltsverzeichnis) stehen. Diese Daten geben Auskunft über Länge und Typ der Dateien.

- Dateien auf derselben Diskette/Festplatte werden durch ihre Bezeichnung (Dateiname) eindeutig identifiziert.

Angesichts der Tatsache, daß man alle Daten, die man auf der Diskette/Festplatte unterbringt, als Dateien darstellen muß, ist selbstverständlich auch das Directory eine (wenn auch besondere) Datei. Bei den Commodore-Laufwerken der 15xx-Serie hat diese Datei sogar einen richtigen Namen: "$". Für solch komplizierte Aufgaben der Dateiverwaltung gibt es natürlich ein Programm, welches (in 15xx-Commodore-Laufwerken) schon fest eingebaut ist: das DOS. DOS ist wieder eine Abkürzung und bedeutet "Disk Operating System" und wird mit "Diskettenbetriebssystem" übersetzt. Dieses Programm übernimmt alle Aufgaben, welche sich mit der Erzeugung, Verwaltung und Löschung von Dateien auf Disketten ergeben.

Jedes unserer Laufwerke hat so ein DOS fest eingebaut. Je nach Laufwerkstyp ist es mehr oder weniger leistungsfähig. Die DOS-Befehle, mit denen man die Funktionen zur Dateiverwaltung aktivieren kann, wurden bequemerweise in das BASIC des jeweiligen Commodore-Computers integriert, so daß sie für Außenstehende wie BASIC-Befehle aussehen. Sie sind aber alles andere als das. Sie sind Teil des DOS, das mit dem BASIC Hand-in-Hand zusammenarbeitet.

4. Erster Diskettenbetrieb

"(A file is)...a structure consisting of a sequence of components - - all of which are of the same type."

Jensen/Wirth[5]

Wir erfahren, wie die Datenströme organisiert werden, wie man eine Diskette auf ihren Einsatz vorbereitet, was es mit "Öffnen" und "Schließen" auf sich hat und wie man endlich ein Inhaltsverzeichnis zu sehen kriegt.

Ich gehe davon aus, daß Sie es inzwischen mit Hilfe der Gebrauchsanleitung geschafft haben, Ihren Commodore-Computer mitsamt Laufwerk auszupacken, zusammenzustöpseln und ohne größere Schäden anzuschalten. Außerdem setze ich voraus, daß Sie es trotz aller Euphorie über den neuerworbenen Rechensklaven nicht versäumt haben, sich mit einem ausreichenden Vorrat an Disketten zu versorgen. Mindestens eine frische Diskette brauchen Sie jetzt. Wenn Sie noch keine haben.

Filekonzept

Bevor wir uns jedoch sofort ganz konkret auf die Tastatur stürzen, müssen wir noch ein bißchen Theorie treiben. Die Frage, die wir uns zu stellen haben, ist:

Wie werden Datenströme organisiert, die vom Rechner kommen oder zum Rechner gehen?

Alle Ein- und Ausgabedaten werden gleich behandelt, egal ob sie z.B. zum Drucker gehen oder von der Diskette kommen. Allerdings muß der Rechner zu diesem Zweck mit dem gewünsch-

5 Jensen/Wirth: Pascal User Manual and Report, Springer 1985 (3. Edition)

ten Gerät logisch verbunden werden. Physikalisch besteht schon eine Verbindung über die Kabelverbindungen. Die logische Verbindung wird durch das Betriebssystem vorgenommen. Diese logische Verbindung wird auch als Kanal bezeichnet. Dazu muß dem Computer mitgeteilt werden, mit welchem Gerät er kommunizieren soll (Drucker, Laufwerk).

Man darf die Daten jedoch nicht wahllos im Gerät verstreuen, sondern muß sie in logischen Einheiten zusammenfassen. Dafür gibt es eine universelle Struktur, die wir schon kennengelernt haben: Dateien (engl.: files). Um solche Dateien in einem Gerät (z.B. Diskettenlaufwerk) zu erzeugen, muß die (logische) Verbindung Rechner-Gerät hergestellt und eindeutig gekennzeichnet werden. Die Kennzeichnung erfolgt durch die sogenannte logische Filenummer, eine positive ganze Zahl, durch die man jetzt in der Lage ist, auf das Gerät zuzugreifen. Die logische Filenummer ist sozusagen der Hinweis darauf, auf welches Gerät bzw. auf welchem Kanal die Daten gesendet werden sollen.

Dieses Herstellen der Verbindung nennt man Öffnen der Datei. Die Bezeichnung des Endgerätes (Drucker, Laufwerk) erfolgt ebenfalls durch eine positive, ganze Zahl, ist aber festgelegt, da jedem Gerät sozusagen vom Werk aus eine Bezeichnung zugewiesen ist. Der Drucker hat die Bezeichnung (=Geräteadresse) '4', dem ersten Laufwerk ist die Geräteadresse '8' zugewiesen. (Es ist aber möglich, einem Laufwerk eine andere Geräteadresse zuzuweisen.)

Ist die Datenübertragung beendet, wird die Verbindung Rechner-Gerät wieder unterbrochen, da sie nicht mehr benötigt wird. Diesen Vorgang nennt man Schließen der Datei. Das ist notwendig, da beim Schließen eventuell noch nicht übertragene Daten endgültig an die Datei geschickt werden; würde das nicht passieren, gingen diese Daten verloren.

Das Öffnen einer Datei geschieht allgemein mit dem Befehl OPEN:

Allgemeine Form:

```
OPEN <lfn>,<ga>
```

Dieser Befehl enthält schon alle Informationen, die der Rechner braucht, um eine Verbindung zu einem Gerät herzustellen. Für Arbeiten, die die Organisation der Datei betreffen, gibt es noch einige Parameter mehr:

Allgemeine Form:

```
OPEN <lfn>,<ga>,<sad>,"<cmd>:<spez>"
```

Die Bedeutungen der einzelnen Komponenten:

<lfn>	logische Filenummer
<ga>	Geräteadresse
<sad>	Sekundäradresse, Kanal 0,1: Sind für LOAD und SAVE reserviert. 2..14: frei verfügbar. 15: Für den Fehler- und Befehlskanal reserviert
<cmd>	Kommando
<spez>	Spezifikationen/Dateiname

Das Kommando und der Doppelpunkt (<cmd>:) können weggelassen werden. Dann geht es um die Datenübertragung von und zu Dateien. Mit <cmd>: werden Vorgänge ausgelöst, die sich auf die Datei als ganzes beziehen, wie beispielsweise Löschen, Kopieren und ähnliches. Wird ein Kommando (mit Doppelpunkt) eingesetzt, dann muß als Sekundäradresse 15 eingesetzt werden. In <spez> stehen dann die Parameter, auf die sich der Befehl bezieht. Für viele OPEN-Anweisungen gibt es ab der BASIC-Version 3.0 "Kosmetikvarianten", die die Benutzung erleichtern. Sollte übrigens jemand in Assembler Disketten- und Dateizugriffe zu programmieren wünschen, muß er/sie die "alten", also ungeschönten Zugriffsmethoden benutzen.

Die Sekundäradresse ist ein Wert zwischen 2 und 14. Der Wert 15 ist für den Befehls- und Fehlerkanal reserviert, die Werte 0 und 1 sind für LOAD bzw. SAVE freizuhalten. Wozu diese Kanäle dienen, wird noch behandelt werden. Die Sekundäradresse hat die Aufgabe, mehrere Dateien, die gleichzeitig auf dasselbe Gerät geschickt werden sollen, logisch zu unterscheiden. Dazu sind nur bestimmte Geräte, wie beispielsweise ein Diskettenlaufwerk, fähig.

Nach dem Öffnen einer Datei interessiert uns naturgemäß auch das Schließen. Diese Aufgabe wird vom CLOSE-Befehl übernommen:

Allgemeine Form:

```
CLOSE <lfn>
```

So, das war die Theorie, nun wollen wir uns auf die Praxis (sprich: Diskette) stürzen.

Wir formatieren eine Diskette

Bevor wir nun mit einer Diskette arbeiten können, müssen wir sie erst vorbereiten, denn sie kommt vollständig leer zum Benutzer. Diesen Vorgang des Vorbereitens nennt man Formatieren. Dabei wird die Diskette in die Bereiche eingeteilt, in denen nachher die Information gespeichert werden kann (Spuren, Sektoren).

Da auch die Diskette wie alle anderen Geräte angesprochen wird, müssen wir selbstverständlich auch zum Formatieren einen Kanal öffnen. Dabei geschieht das je nach BASIC-Version entweder sichtbar über den Befehl OPEN oder versteckt in einem neuem Befehl. Für Befehle, die mit OPEN aufgerufen werden, muß der Fehler- oder Befehlskanal (Sekundäradresse 15) benutzt werden. Dieser Kanal wird noch besprochen werden. Doch kommen wir jetzt zum Formatieren der ersten Diskette.

Achtung: Wird eine Diskette formatiert, werden ALLE auf ihr enthaltenen Daten gelöscht! Eine Diskette muß nur einmal formatiert werden. Ist eine Diskette formatiert, kann man immer mit ihr auf dem Computer- und Laufwerkstyp arbeiten, auf dem sie formatiert wurde.

BASIC bis 2.0:

Allgemeine Form:

```
OPEN1,<ga>,15,"N:<name>,<id>"
```

N	Befehl NEW, neue Diskette vorbereiten, formatieren
<name>	Name der Diskette (nur zum Archivieren)
<id>	Zwei Zeichen, dienen zur Identifizierung (für das DOS wichtig)
<ga>	Geräteadresse des Laufwerks

Beispiel:

```
OPEN1,8,15,"N:ERSTE DISKETTE,A1"
```

*****

BASIC ab 3.0:

Allgemeine Form:

```
HEADER "<name>",I<id>,D<ln>,U<ga>
```

<name>	Name der Diskette (nur zum Archivieren)
<id>	Zwei Zeichen, dienen zur Identifizierung (Für das DOS wichtig)
<ln>	Laufwerksnummer 0 = Laufwerk 1 1 = Laufwerk 2
<ga>	Geräteadresse des Laufwerks

Beispiel:

```
HEADER "ERSTE DISKETTE",IA1,D0,U8
```

Hinweis: Obwohl in beiden Fällen (OPEN und HEADER) ein Kanal zum Diskettenlaufwerk geöffnet werden muß, ist dieses Öffnen im zweiten Fall im Befehl HEADER eingeschlossen/versteckt.

*****

Bei Nichtverwendung der <id> wird nur das Inhaltsverzeichnis gelöscht.

Eine kurze Bemerkung zu der Laufwerksnummer, die in diesem Befehl zum ersten Mal auftaucht. Die Einrichtung der Laufwerksbezeichnung stammt aus der Zeit, als es für Commodore-Computer (z.B. 3000er-, 8000er-Serie) noch Doppelfloppystationen gab. Diese hatten nur eine Geräteadresse, aber man hatte zwei Laufwerke zu unterscheiden. So wurden die beiden Laufwerke mit 0 bzw. 1 bezeichnet. Für Einzellaufwerke, wie wir sie hier besprechen, ist diese Angabe jedoch überflüssig. Also wird ab sofort bei jedem Befehl, bei dem solche Zusätze auftauchen, auf deren Erwähnung verzichtet.

Doch zurück zum Thema: Haben Sie den vorigen (für Ihr Laufwerk gültigen) Befehl ausgeführt, wird die Diskette formatiert, erhält einen Namen (ERSTE DISKETTE) und eine für den Computer wichtige Identifizierungsnummer oder ID: A1. Jede Diskette, die Sie formatieren, sollte eine andere ID bekommen, damit das DOS einen Diskettenwechsel mitbekommt. Denn das DOS unterscheidet verschiedene Disketten nicht am Namen, sondern an der ID. Haben Sie den Befehl ausgeführt, läuft das Laufwerk einige Zeit, dann bleibt es stehen. Die Diskette ist fertig präpariert und kann ab sofort benutzt werden.

BASIC bis 2.0:

Da wir eine Datei geöffnet haben, müssen wir sie jetzt auch wieder schließen.

```
CLOSE1
```

*****

BASIC ab 3.0:

Es ist kein Schließen notwendig, da es im HEADER-Befehl implizit enthalten ist.

*****

Das Formatieren ist damit abgeschlossen und bereit zur Benutzung auf dem bei der Formatierung benutzten Computertyp.

Der Catalog "$"

Um zu erfahren, welche Programme sich auf der Diskette befinden, muß man das Inhaltsverzeichnis (das Directory) sichtbar machen.

BASIC bis 2.0:

Das geschieht durch den Befehl LOAD:

```
LOAD "$",8
```

Die Syntax für das Laden des Inhaltsverzeichnisses ist also festgeschrieben. Das Directory wird bei diesem Laufwerk wie ein BASIC-Programm geladen (siehe später). Das bringt einige Nachteile mit sich, die wir noch besprechen werden. Das '$'-Zeichen ist der Name des "Programms" Directory, die '8' ist (normalerweise) die Geräteadresse des (ersten) Laufwerks.

Wollen wir uns anschauen, was wir geladen haben, müssen wir uns das "Programm", das in diesem Fall ja das Inhaltsverzeichnis ist, ansehen, und zwar (wie in BASIC gewohnt) mit LIST.

```
LIST
```

Diese Methode ist weder praktisch noch verbreitet. Man fragt sich, was sich die Programmierer bei dieser Geschichte gedacht haben. Aber bei BASIC-Versionen höher als 2.0 hat man das wieder entfernt und die übliche Methode eingeführt, die auch im BASIC ab Version 3.0 vorhanden ist.

*****

BASIC ab 3.0:

Hier ist wie gesagt alles viel einfacher. Sie geben einfach den Befehl

```
DIRECTORY
```

ein, und das Directory wird sofort auf dem Bildschirm ausgegeben. Ein im Speicher befindliches BASIC-Programm wird im Gegensatz zum BASIC 2.0 nicht gelöscht.

*****

Ist auf der Diskette noch nichts gespeichert, so sehen wir folgendes Bild (LAUFWERK 1541/1570):

```
0 "ERSTE DISKETTE " A1 2A
664 BLOCKS FREE.
```

Diesem Bild können wir die Information entnehmen, daß noch 664 Blöcke (Blocks) frei sind. Ein Block enthält 254 Bytes; also sind auf der Diskette 168656 Bytes = 168656/1024 = 165 kBytes frei (Beim Laufwerk 1571 sind es doppelt so viel!). Ich werde aber meist die Bezeichnung "Sektor" statt "Block" verwenden.

Wenn man bedenkt, daß 1 Byte entweder einem Zeichen oder einem BASIC-Befehl entspricht, dann ist demnach für sehr viele Übungsprogramme Platz!

5. Programmierbetrieb in BASIC

"Und dieser Computer, der Die Erde hieß, war so groß, daß er oft fälschlich für einen Planeten gehalten wurde - besonders von den merkwürdigen affenartigen Wesen, die auf seiner Oberfläche herumrasten und absolut keinen Schimmer davon hatten, daß sie lediglich Bestandteil eines gigantischen Computerprogramms waren."

Douglas Adams[6]

Wir lernen, wie man BASIC-Programme, nachdem man sie geschrieben hat, auf Diskette sichert, nach Wunsch wieder in den Computerspeicher holt und warum man den "Klammeraffen" meiden sollte. Danach werden noch einige nützliche Werkzeuge vorgestellt, mit deren Hilfe man eine ganze Menge auf der Diskette anstellen kann. Zum Schluß lernen wir die "wilden Karten" kennen, die ja eigentlich ganz anders heißen.

Wir wollen in diesem Kapitel davon ausgehen, daß sich im Speicher des Rechners ein BASIC-Programm befindet. Zu diesem Zwecke werden wir ein kleines Programm in den Rechner eingeben, um damit üben zu können.

Zwar reicht für unsere Zwecke irgendein beliebiges Programm, aber ich mache Ihnen, um Ihre Phantasie etwas zu unterstützen, hier einen Vorschlag:

```
10 REM *** PROGRAMM ZUR BERECHNUNG
20 REM *** DER QUADRATZAHLEN VON 1 BIS 20
30 REM
40 FOR I=1 TO 20
50 PRINT I;"*";I;"=";I*I
60 NEXT I
70 END
```

6 Douglas Adams: Das Restaurant am Ende des Universums, Ullstein 1988

Ich gehe nun davon aus, daß sich das Programm inzwischen im Speicher befindet, Sie es also abgetippt haben. Um das Programm abzuspeichern, müssen Sie zunächst dafür sorgen, daß Ihr Diskettenlaufwerk an den Computer angeschlossen ist und der Netzschalter angeschaltet ist.

BASIC bis 2.0:

Bei dieser BASIC-Version ist es wichtig, daß Sie vor der Eingabe des BASIC-Programms den Befehl NEW eingeben (Löschen des Programmspeichers), da das Inhaltsverzeichnis ja wie ein BASIC-Programm geladen wird und sich nur dadurch wieder entfernen läßt. Andernfalls ist das Directory noch im Speicher und stört das Programm!

Das Programm wird dann durch den Befehl SAVE auf Diskette abgespeichert:

```
SAVE "QUADRATZAHLEN",8
```

Vielleicht kennen Sie den Befehl schon ohne den Zusatz ",8". So wird ein Programm ja auf Cassette abgespeichert. Der Zusatz ",8" bedeutet lediglich, daß jetzt nicht die Cassette, sondern das Diskettenlaufwerk mit der Geräteadresse 8 der Empfänger ist. Erinnern Sie sich bei der Gelegenheit, was "Geräteadresse", "Logische Filenummer" und "Sekundäradresse" sind!

Allgemein sieht der Befehl dann so aus:

```
SAVE "<dateiname>",<ga>
```

*****

BASIC ab 3.0:

Der Befehl, der bei dieser BASIC-Version dafür sorgt, daß ein Programm auf Diskette abgespeichert wird, heißt DSAVE:

```
DSAVE "QUADRATZAHLEN"
```

Um dem Benutzer den Betrieb zu erleichtern, hat man die Bezeichnung, die beschreibt, auf welches Gerät die Daten sollen, einfach vor das SAVE gehängt. Das "D" steht selbstverständlich für Diskette.

Hier muß man nicht auf das Inhaltsverzeichnis achten, denn das wird hier ja nur auf den Bildschirm gechrieben, nicht in den Speicher. Wenn das Programm auf ein anderes Laufwerk gespeichert werden soll, braucht man zusätzliche Information. So sieht die allgemeine Form des Befehls aus:

```
DSAVE "<dateiname>",D<lw>,U<ga>
```

Dabei bedeutet:

<lw> Laufwerk 0 oder 1

<ga> Geräteadresse: 8, 9, 10, 11

*****

Um zu sehen, was auf der Diskette passiert ist, müssen wir das, was wir über das Sichtbarmachen der Directory gelernt haben, in die Praxis umsetzen. Je nachdem, welches Laufwerk Sie haben, wird sich das folgende Bild oder ein ähnliches auf dem Bildschirm einstellen, wenn Sie die letzte Aufgabe erfolgreich gelöst haben. Als Beispiel soll das Directory der 1541 gezeigt werden.

```
0 "ERSTE DISKETTE " A1 2A
1 "QUADRATZAHLEN" PRG
663 BLOCKS FREE.
```

Jetzt sieht das Inhaltsverzeichnis schon anders aus als vorhin! Erstens ist ein Eintrag hinzugekommen, nämlich unser 'QUADRATZAHLEN'-Programm, und die letzte Zeile weist jetzt einen Block weniger auf.

Die Zahl links vom Programmnamen bezeichnet die Anzahl der Blöcke (Sektoren), die das Programm belegt. Da immer nur ein

ganzer Block belegt werden kann, egal, wie voll er wirklich ist, handelt es sich immer um eine ganze Zahl. In unserem Fall belegt das Programm nur einen Block.

Die Bezeichnung rechts vom Namen sagt aus, daß es sich um ein BASIC-Programm handelt (PRG). Wir werden später noch weitere Einträge kennenlernen (z.B. SEQ, REL).

Wie bekommen wir ein Programm wieder?

Tja, nun haben wir eine Diskette formatiert, ein Programm geschrieben und auf der Diskette gespeichert. Wir können also unseren Computer ruhig ausschalten und uns anderen Beschäftigungen widmen. Aber, und das ist die Frage, wie bekommt man das Programm wieder in den Computer?

Eigentlich habe ich das schon verraten, als es um das Laden des Inhaltsverzeichnisses ging (BASIC bis 2.0). Dort lautete der Befehl

```
LOAD "$",8.
```

Da wir das Zeichen '$' als Dateinamen des Directorys erklärt haben, muß das Laden eines Programmes genauso erfolgen. Mit dem einen Unterschied, daß statt des Directorynamens "$" der eigentliche Dateiname eingesetzt werden muß. Laden wir also unser Programm "QUADRATZAHLEN":

BASIC bis 2.0:

```
LOAD "QUADRATZAHLEN",8
```

Allgemein:

```
LOAD "<dateiname>",<ga>
```

Steht die Datei auf einer Diskette im zweiten Laufwerk (Geräteadresse = <ga> = 9), muß selbstverständlich statt der 8 eine 9 eingesetzt werden.

*****

BASIC ab 3.0

```
DLOAD "QUADRATZAHLEN"
```

Allgemeine Form:

```
DLOAD "<Dateiname>",U<ga>
```

*****

Nach der Ausführung dieses Befehls steht das Programm startbereit im Arbeitsspeicher und wartet nur auf das RUN oder LIST. Stand vor dem Laden ein anderes Programm im Speicher (oder das Directory bei BASIC-Versionen bis 2.0), dann ist dies jetzt gelöscht.

BASIC bis 2.0:

Ich will noch einmal aufzeigen, welche Merkwürdigkeiten bei BASIC-Versionen bis 2.0 auftreten, und darlegen, wie diese mit dem Directory umgehen. Die Tatsache, daß das Directory wie ein BASIC-Programm geladen wird, muß hier eingehend betrachtet werden. Es können zwei Fälle auftreten:

- Sie haben ein BASIC-Programm im Speicher und wollen es abspeichern. Speichern Sie es, bevor Sie das Directory ansehen, denn um das Directory zu sehen, muß man es bei dieser BASIC-Version als Programm in den Speicher laden! Bei dieser Aktion wird das Programm dann gelöscht und durch das Directory ersetzt

  Wenn Sie jetzt versuchen abzuspeichern, haben Sie ein altes Directory auf Diskette abgespeichert, was nur zu Verwirrun-

gen führt. Also: Haben Sie ein Programm im Speicher, sichern Sie es, bevor Sie das Directory ansehen!

- Sie haben ein Directory im Speicher und wollen ein neues Programm schreiben (eintippen). Sorgen Sie dafür, daß das im Speicher stehende Directory durch NEW vor dem Eintippen des neuen Programms aus dem Speicher gelöscht wird. Versäumen Sie das, so steht Ihr Programm ineinander geschachtelt mit dem Directory zusammen im Speicher, und es wird nicht richtig laufen.

*****

An dieser Stelle muß ich nun auf eine Tatsache eingehen, die mir, wenn ich damit konfrontiert werde, entweder leises Schmunzeln entlockt oder echten Ärger hervorruft, je nachdem, ob ich zuschaue oder selber beteiligt bin. Es geht um das Abspeichern einer Datei, deren Name auf Diskette schon existiert.

Der Vorgang ist alltäglich: Sie haben in einem Programm einen Fehler entdeckt, oder Sie möchten eine Erweiterung einbauen. Das Programm, um das es geht, steht unter irgendeinem Namen, den Sie gewählt haben, auf der Diskette. Das verbesserte Programm soll jetzt natürlich wieder auf der Diskette gespeichert werden; selbstverständlich unter demselben Namen wie das alte Programm. Außerdem kann das alte Programm gelöscht werden, da es ja nicht vollständig oder fehlerhaft war.

Mit dem bisher gelernten SAVE- oder DSAVE-Kommando führt diese Vorgehensweise allerdings zu einer Fehlermeldung, da hier ein Schutzmechanismus eingebaut ist. Es ist also nicht möglich, eine bestehende Datei mit diesen Kommandos zu überschreiben. Wir üben das an einem Beispiel: Nehmen wir an, das Programm sei das obige Beispielprogramm "QUADRATZAHLEN".

Also laden wir das Programm, ändern es und speichern es wieder so ab, wie wir es gelernt haben. Aber stop! So einfach

geht das hier nicht. Nach einem Versuch beginnt die rote Lampe der Floppy zu blinken (= Fehler!). Wir merken uns:

Soll eine Programm- oder eine andere Datei auf der Diskette Überschrieben, also ersetzt werden, so muß das durch das Zeichen Klammeraffe ("@") explizit angegeben werden.

Wir schauen uns das jetzt genau an:

BASIC bis 2.0:

Um eine bestehende Datei überschreiben zu können, muß vor dem Dateinamen die Zeichenfolge "@:" stehen.

Beispiel:

```
SAVE "@:QUADRATZAHLEN",8
```

Allgemein:

```
SAVE "@:<dateiname>",8
```

*****

BASIC ab 3.0:

Um eine bestehende Diskettendatei überschreiben zu können, muß vor den Dateinamen das Zeichen "@" stehen.

Beispiel:

```
DSAVE "@QUADRATZAHLEN"
```

Allgemein:

```
DSAVE "@<dateiname>",U<ga>
```

Achten Sie hier darauf, daß der Doppelpunkt ":" bei diesem Befehl nicht hinter dem Klammeraffen steht!

*****

Der Wermutstropfen

Ja, das hört sich alles ganz ordentlich an, wenn da nicht etwas wäre, was die ganze Angelegenheit stören würde. Zwar ist in der Theorie alles gut und schön, jedoch bei der Umsetzung der Theorie in die Praxis war wohl kein guter Tag zum Programmieren gewesen, denn es hat sich ein Fehler eingeschlichen, der den guten Vorsatz, etwas Vernünftiges zu produzieren, zunichte gemacht hat.

Wenn nämlich die Diskette fast voll ist, und das passiert schnell, dann spielt der Klammeraffe "verrückt"! Er bringt die Organisation des Directorys völlig durcheinander, und das bewirkt, daß Sie Programme zerstören oder verlieren können. Oder es bekommt ein Programm plötzlich einen anderen Namen zugeteilt, während sich das Programm, dem dieser Name gehörte, in Luft aufgelöst hat. Ebenso kann es passieren, daß das gerade abgespeicherte Programm unter zwei verschiedenen Namen gleichzeitig existiert. Schade nur, daß einer der beiden Namen früher schon eine Datei bezeichnet hat, und diese ist jetzt natürlich zerstört. Es lohnt sich vielleicht, in dieser Richtung selber zu experimentieren. So merken Sie schnell, wie der Hase bzw. Klammeraffe läuft, und Sie können dabei einiges an Erfahrung sammeln (und Dateien verlieren).

Nun wollen Sie natürlich auch eine Lösung für dieses Problem. Bevor Sie das Programm unter demselben Namen auf Diskette speichern können, müssen Sie das bestehende mit einem speziellen Befehl löschen - wir werden ihn noch behandeln. Bis dahin behelfen Sie sich bitte damit, neue Versionen unter geänderten Namen abzuspeichern, also QUADRATZAHLEN1, QUADRATZAHLEN2, QUADRATZAHLEN3 usw.

Maschinenprogramme und Grafiken laden

Manchmal möchte man Dateien laden, die sich von den (bisher besprochenen) üblichen BASIC-Programmdateien signifikant unterscheiden. Zum Beispiel: Maschinenprogramme, Unterroutinen für BASIC-Programme und nicht zuletzt Graphikdateien, in

denen man irgendwelche Bilder gespeichert hat. Mit den Befehlen SAVE bzw. DSAVE geht das leider nicht, da diese Befehle die Datei an eine fest vorbestimmte Stelle im Speicher laden: in den sogenannten BASIC-Speicherbereich. Maschinenprogramme sollen aber oft in andere Speicherbereiche geladen werden, und es ist natürlich wünschenswert, daß Bilder in den Speicherbereich geladen werden, der den Bildschirminhalt auf dem Monitor darstellt. Ich beschränke mich hier auf die Befehle der BASIC-Versionen größer als 2.0, also übergehe ich hier z.B. das BASIC des C64.

Die Befehle (ab BASIC 3.0), die dafür zuständig sind, heißen

```
BSAVE
```

und

```
BLOAD.
```

Syntax:

```
BSAVE "<dateiname>",U<ga>,ON B<bank>,P<aa> TO P<ea>
```

<ga>	Geräteadresse
<bank>	Nummer der Speicherbank (bei Rechnern mit mehr als 64k Speicher)
<aa>	Anfangsadresse des zu speichernden Datenbereichs
<ea>	Endadresse +1 des zu speichernden Datenbereichs

Beispiel:

```
BSAVE "MEMORYDUMP",U8,ON B0,P2000 TO P3001
```

Damit wird der Speicherbereich von 2000 bis 3000 in Speicherbank 0 in die Datei "MEMORYDUMP" auf Gerät 8 gespeichert.

Die nicht sehr einsichtige Tatsache, daß man bei der Endadresse 1 addieren muß, liegt daran, daß der Befehl das letzte Byte nicht

mitspeichert. Deshalb muß man als Endadresse immer die nächsthöhere Adresse, also Adresse+1, angeben.

Syntax:

```
BLOAD "<dateiname>",U<ga>,ON B<bank>,P<aa>
```

Beispiel:

```
BLOAD "MEMORYDUMP",U8,ON B0,P2000
```

Mit diesem Beispiel wird der Inhalt der Datei "MEMORY-DUMP", die sich auf dem Laufwerk mit der Geräteadresse 8 befindet, ab der Adresse 2000 der Speicherbank 0 in den Speicher geladen.

Eine Endadresse hier anzugeben ist selbsverständlich überflüssig. Falls Sie das nicht einsehen, lassen Sie sich vor dem Weiterlesen Zeit, diese Tatsache zu überlegen.

Maschinenprogramme unter BASIC

Ein kurz erklärtes, aber nicht unwichtiges Gebiet bei der Floppy-Arbeit ist das Laden von Programmen, die in Maschinensprache geschrieben wurden und auch als solche auf der Diskette abgelegt wurden. Diese Programmsorte lädt man mit dem Befehl

Allgemeine Syntax:

```
LOAD "<dateiname>",<ga>,1
```

Beispiel:

```
LOAD "MASCHPROG",8,1
```

Das ",1" zeigt dem Rechner, daß hier ein Maschinenprogramm geladen werden soll. Lädt man ein solches Programm, so ändert sich augenscheinlich gar nichts. Sollte zum Beispiel (z.B. bei

BASIC bis 2.0) vorher das Directory geladen worden sein, so ist dieses, im Gegensatz zu früheren Ausführungen, nach wie vor im Arbeitsspeicher!

Wo ist das geladene Maschinenprogramm geblieben?

Maschinenprogramme müssen nicht wie BASIC-Programme immer an einer festen Stelle beginnen, sondern sie beginnen dort, wo der Programmierer es bestimmt hat. BASIC-Programme beginnen immer an derselben Speicherstelle (beim C64 ist das Adresse 2048, bei C116, C16 und Plus4 ist es die Adresse 4096). Maschinenprogrammierer nutzen meistens andere Bereiche (z.b. C64: ab 49152, weil hier noch einmal 4 kByte Speicher frei verfügbar sind), sogar der Bildschirmbereich wird nicht ausgelassen. Die Anfangsadresse des Programmes, die nicht unbedingt auch die Startadresse sein muß, ist zu Beginn der Datei gespeichert, die das Maschinenprogramm enthält. Will man nun ein solches Programm starten, muß man es direkt aufrufen, indem man mittels des BASIC-Befehles SYS unter Angabe der Startadresse den Rechner veranlaßt, an dieser Stelle die Folge von Maschinencodes abzuarbeiten.

Allgemeine Syntax:

```
SYS <adresse>
```

Beispiel:

```
SYS 4711
```

Nun ergibt sich das folgende Problem: Wo ist die Startadresse? Die einfachste Lösung ist selbstverständlich die Kenntnis der Startadresse. Beginnt das Programm bei der Adresse 50000, so muß der Benutzer für den Start des Programmes folgendes eingeben:

```
LOAD "MASCHPROG",8,1
SYS 50000
```

Nach dieser Sequenz läuft das Programm. Eine Hilfe für das strapazierte Gedächtnis ist der Vermerk der Startadresse im Dateinamen. Das Laden einer solchen Datei würde dann so aussehen:

```
LOAD "MASCHPROG 50000",8,1
SYS 50000
```

Viele Maschinenprogramme haben zu Beginn eine einzige BASIC-Zeile, und dahinter liegt unsichtbar das Maschinenprogramm. Diese eine Zeile enthält dann auch schon den SYS-Befehl mit der richtigen Startadresse. Ist dies aber nicht der Fall und ist die Anfangsadresse des Programms nicht bekannt, so gibt es noch eine Möglichkeit, die Startadresse herauszufinden. Diese steht nämlich ganz zu Beginn der Datei auf der Diskette in den beiden ersten Bytes. Das folgende Programm ermittelt die Startadresse:

```
10 REM Programm zum Ermitteln der Startadresse eines Programms
20 PRINT CHR$(147)
30 INPUT "Bitte geben Sie den Dateinamen ein"; N$
40 OPEN 3,8,3,N$
50 GET#3,A1$: A1$ = A1$ + CHR$(0): A1 = ASC(A1$)
60 GET#3,A2$: A2$ = A2$ + CHR$(0): A2 = ASC(A2$)
70 CLOSE 3
80 AD = A2 * 256 + A1
90 PRINT "Die Startadresse ist: "; AD
```

Wenn Sie dieses Programm abgetippt und unter dem Namen "Adresse" gespeichert haben, können Sie es gleich am Programm "Adresse" ausprobieren. Sie müßten dann (beim C64) als Ergebnis die Zahl 2049 angezeigt bekommen, weil das Programm ein BASIC-Programm ist und der BASIC-Speicher ab 2049 beginnt.

Verifizieren - Was ist das und wie geht das?

Verifizieren kann man mit "Auf seine Richtigkeit überprüfen" übersetzen. Das muß man auch mit Programmen, die sich auf der Diskette befinden. Allerdings wird diese Tätigkeit zum größten Teil vom DOS erledigt.

Trotzdem hat man einen Befehl eingebaut, der nach dem Abspeichern überprüfen soll, ob das Programm auch richtig auf der Diskette angekommen ist. Das Kommando heißt VERIFY. Als Beispiel verifizieren wir das Programm "QUADRATZAHLEN":

BASIC bis 2.0:

```
VERIFY "QUADRATZAHLEN",8
```

Allgemeine Form:

```
VERIFY "<dateiname>",<ga>
```

*****

BASIC ab 3.0:

```
DVERIFY "QUADRATZAHLEN"
```

Allgemeine Form:

```
DVERIFY "<dateiname>",<ga>
```

*****

Der Befehl entstammt der Zeit, als man noch Programme und Daten auf einem Datenrecorder abgespeichert hat, weil es noch keine billigen Diskettenlaufwerke gab. Beim Cassettenbetrieb ist es nämlich nicht möglich, sofort nach dem Speichern zu überprüfen, ob auf dem Band tatsächlich alles richtig angekommen ist. Ich erinnere mich noch gut, wie ich bei meinem ersten Computer (ein alter Video-Genie) manchmal bis zu fünfmal das SAVE-Kommando benutzen mußte, ehe alles richtig auf dem Band war. (Anmerkung von Commodore-Benutzer Cornelius: "Ganz so schlimm war es mit der Datensicherheit bei Commodore-Datasetten nie, nur viiiiiel langsamer!") Zu diesem Zweck hatte der Computer das VERIFY-Kommando, um das Programm

im Speicher mit der Information auf dem Band zu vergleichen. Waren beide gleich, war alles angekommen; waren die Daten unterschiedlich, folgte eine Fehlermeldung.

Beim Diskettenbetrieb verliert diese Funktion ihre Bedeutung, denn das DOS übernimmt das Verifizieren automatisch. Commodore-Laufwerke überprüfen die Daten sogar mehrmals, bis sie meinen, daß alles gutgegangen ist. So ist denn dieser Befehl für Diskettendateien überflüssig; das könnte man wenigstens meinen.

Falsch! Denn dieser Befehl läßt sich doch noch gewinnbringend verwenden.

Angenommen, Sie haben auf zwei Disketten unter demselben Namen jeweils ein Programm, und Sie wissen nicht mehr, ob damit dasselbe gemeint ist, oder etwa eine neuere Version bzw. ein ganz anderes Programm. Dann können Sie folgendermaßen vorgehen:

- Laden Sie das erste Programm in den Speicher

- Geben Sie bei VERIFY den Namen des zweiten Programms an. Die Diskette mit dem zweiten Programm muß dann im Laufwerk sein.

Folgt dann die Meldung "OK", sind beide Programme identisch, kommt eine Fehlermeldung, sind die Programme verschieden. So ist dieser Befehl doch nicht ganz überflüssig.

Nützliche Werkzeuge

Bis jetzt haben wir die notwendigen Kommandos zum Formatieren, Speichern, Laden und Verifizieren von Programmen gelernt. Aber das reicht natürlich nicht. Man muß zum Beispiel einen Dateinamen ändern können, falls der alte Name nicht mehr gefällt. Oder man mag nicht nur den Dateinamen nicht mehr, sondern die ganze Datei ist überflüssig geworden. Dann

muß man einzelne Diskettendateien löschen können. Und das sind noch nicht alle Wünsche, die einem auf Anhieb einfallen.

Basis für all diese Kommandos ist der OPEN-Befehl, wie wir ihn schon vom Formatieren her kennen. Schauen Sie sich also noch einmal genau die allgemeine Form dieses Befehls an, wenn Sie sie nicht mehr genau im Kopf haben. Zuerst wollen wir uns um das Ändern eines Dateinamens kümmern. Umbenennen heißt RENAME. Das Kommando für diesen Befehl ist also der Buchstabe "R" bzw. "RENAME". Der Befehl sieht dann so aus:

BASIC bis 2.0:

Allgemeine Form:

```
OPEN1,<ga>,15,"R:<neuer dateiname>=<alter dateiname>"
```

Als Beispiel wollen wir unser Beispielprogramm "QUADRAT-ZAHLEN" in "QUADRATE" umbenennen. Die Datei befindet sich auf einer Diskette im Laufwerk mit der Geräteadresse 8. Das sieht dann so aus:

```
OPEN1,8,15,"R:QUADRATE=QUADRATZAHLEN"
```

*****

BASIC ab 3.0:

Allgemeine Form:

```
RENAME "<alter Dateiname>" TO "<neuer Dateiname>",U<ga>
```

Für unser Beispiel auch hier die Umbenennung von "QUA-DRATZAHLEN" nach "QUADRATE", Geräteadresse 8:

```
RENAME "QUADRATZAHLEN" TO "QUADRATE"
```

*****

Beachten Sie bitte, daß in Versionen bis 2.0 zuerst der neue Dateiname kommt, dann der alte Dateiname. Bei Versionen ab 3.0 ist das genau umgekehrt!!!

Als nächstes wollen wir uns daran machen, eine Datei zu kopieren. Das Kopieren gehört zu den wichtigsten Funktionen, die ein Betriebssystem oder DOS können muß. Allein zur sehr wichtigen Datensicherung muß man in der Lage sein, von einer Datei eine Sicherheitskopie (engl.: Backup) anzulegen. Auch muß man ab und zu seine Disketten umorganisieren, und das heißt eben, Dateien zu kopieren. Der Befehl dazu heißt "C" bzw. "COPY". Allerdings fragt man sich auch hier wieder, warum eine gute Idee und weit verbreitete Methode, Kopierfunktionen so zu bauen, daß man auch zwischen mehreren Laufwerken kopieren kann, hier keine Anwendung gefunden hat.

Man findet wieder die Ursache in der Vergangenheit, als es eben noch Floppydoppelstationen gab. Die Befehle sind dafür gebaut worden. Hier war (und ist) es möglich, Dateien zwischen zwei Laufwerken zu kopieren. Mit den Einzellaufwerken geht das nun leider nicht mehr. Aber auch wenn Sie nur ein Laufwerk besitzen, können Sie diesen Befehl doch sinnvoll zum Duplizieren von Dateien auf einer Diskette einsetzen. Denn man muß manchmal an Dateien herumexperimentieren, und da empfiehlt es sich natürlich, von der Originaldatei eine oder zwei Kopien anzulegen. Auf diese Weise hat man Ersatz, falls die Experimente fehlschlagen und die Originaldatei zerstören.

Eine zweite Anwendung ist das Verketten von sequentiellen Dateien. Was das ist, wird etwas später besprochen. Jetzt wollen wir nur festhalten, daß man mit dem Kopierbefehl mehrere solcher Dateien zu einer großen Datei zusammenfügen kann.

Als Beispiel soll nun die Datei "QUADRATE" in eine Datei "QUADRATE.BAK" kopiert werden. Das ".BAK" soll anzeigen, daß es sich hier um eine Sicherheitskopie (BACKUP-Datei) handelt. Die Originaldatei (Quelldatei) befindet sich auf der Dis-

kette im Laufwerk mit der Geräteadresse 8, die Sicherheitskopie (Zieldatei) wird unter anderem Namen auf dieselbe Diskette kopiert.

BASIC bis 2.0:

Allgemeine Form:

```
OPEN1,<ga>,15,"C:<Zielname>=<Quellname(n)>"
```

wobei <Quellname(n)> entweder <Dateiname> oder <Dateiname1>, <Dateiname2>,... bedeutet.

Beispielfall:

```
OPEN1,8,15,"C:QUADRATE.BAK=QUADRATE"
```

Werden mehrere Dateinamen für "Quellname" durch Komma getrennt angegeben, so werden die einzelnen Dateien zu einer neuen Datei mit dem Namen "Zielname" verbunden. Dies kann beispielsweise nützlich sein, wenn man mehrere Textteile zu einem großen Text verbinden will.

*****

BASIC ab 3.0:

Allgemeine Form:

```
COPY "<Quelldatei>" TO "<Zieldatei>" ON U<ga>
```

Beispielfall:

```
COPY "QUADRATE" TO "QUADRATE.BAK" ON U8
```

Für das Aneinanderketten einer Datei an eine andere gibt es hier noch eine weitere, einprägsamere Form:

```
CONCAT "<Quelldateiname>" TO "<Zieldateiname>" ON <ga>
```

Dabei wird die Quelldatei an die Zieldatei angehängt. Dies wird gerne bei Dateien angewandt, zu denen laufend neue Daten dazukommen (siehe sequentielle Files).

*****

Das nächste Werkzeug, das besprochen werden soll, ist das Kommando zum Löschen einer Datei löschen Datei. Es gibt viele Namen für dieses Kommando: Delete, Erase, Scratch, Kill,...Für Commodore-Laufwerke wählte man den Namen SCRATCH.

Aber Löschen und Löschen ist nicht dasselbe! Es gibt bei Computern allgemein zwei Arten, Dateien zu löschen:

Logisches Löschen

Die Datei wird nur als "gelöscht" gekennzeichnet. Das bedeutet, daß die Datei zwar immer noch unberührt auf der Diskette steht, das Betriebssystem oder DOS aber nicht mehr darauf zugreifen kann. Erst wenn andere Dateien auf der Diskette gespeichert werden, wird die Datei ganz oder teilweise zerstört, da die von ihr bisher benutzten Sektoren vom Betriebssystem zur Benutzung freigegeben worden sind. Diese Tatsache bietet die erfreuliche Möglichkeit, eine Datei, die so gelöscht worden ist, mit Hilfe intimer Kenntnisse über die Dateiorganisation auf einer Diskette wieder zu "entlöschen" bzw. wieder für das Betriebssystem sichtbar zu machen. Allerdings muß das unbedingt vor jeder weiteren Schreiboperation geschehen.

Physikalisches Löschen

Der Inhalt der zu der Datei gehörenden Sektoren wird völlig zerstört. Das bedeutet, daß nach dem Löschvorgang auf der Diskette keinerlei Spur von der gelöschten Datei mehr existiert. Sie ist nicht mehr rekonstruierbar. Allerdings ist das mit erheblichem Aufwand verbunden.

Auch bei Commodore-Laufwerken wird nur logisch gelöscht, so daß mit diversen Hilfsprogrammen eine Datei unmittelbar nach ihrem Löschen wiederhergestellt werden kann. Solch ein Hilfs-

programm ist auf der beigepackten Demodiskette vorhanden. Als Beispiel wollen wir an dieser Stelle auf unserer Übungsdiskette die Datei "QUADRATE.BAK" löschen, da wir uns entschlossen haben, daß keine Sicherheitskopie notwendig ist.

BASIC bis 2.0:

```
OPEN1,8,15,"S:QUADRATE.BAK"
```

Allgemeine Form:

```
OPEN1,<ga>,15,"S:<Dateiname>"
```

*****

BASIC ab 3.0:

```
SCRATCH "QUADRATE.BAK"
```

Allgemeine Form:

```
SCRATCH "<Dateiname>",U<ga>
```

*****

Bevor wir den nächsten Befehl besprechen, müssen wir uns ein wenig damit beschäftigen, wie das Directory bei Änderungen gehandhabt wird. Um immer die Übersicht über alle belegten bzw. freien Sektoren (Blöcke) zu behalten, merkt sich das DOS die Liste aller noch verfügbaren Blöcke, die sogenannte Block Availability Map oder kurz BAM, in einem besonderen Speicherbereich im Speicher des Laufwerks. Dort wird über jede Änderung Buch geführt, ob ein Block belegt ist oder als freigegeben gilt.

Nun kann aber ein Fall eintreten, der zwar alltäglich, aber für das DOS nicht vorhersehbar ist: Sie wechseln die Diskette. Der Computer merkt das natürlich nicht, und das DOS vermutet im-

Computer merkt das natürlich nicht, und das DOS vermutet immer noch die alte Diskette im Laufwerk. Das Problem ist jetzt aber, daß die BAM der alten Diskette natürlich nicht mehr mit der neuen Diskette übereinstimmt. Das kann zu erheblichen Fehlreaktionen führen, wenn man nichts dagegen unternimmt. Um zu verhindern, daß so etwas laufend passiert, hat man zwei Dinge eingeführt:

- Die ID auf der Diskette, die beim Formatieren in das Directory eingetragen wird. (Falls Sie sich nicht mehr daran erinnern, sollten Sie jetzt noch einmal den Abschnitt über das Formatieren lesen!) Aus diesem Grund sollten alle Disketten, die Sie benutzen eine unterschiedliche ID besitzen. Dann nämlich (und nur dann) wird das Aktualisieren der BAM automatisch durchgeführt.

- Den Befehl INITIALIZE. Der Name steht für Initialisieren (auf einen definierten Anfangszustand bringen) und sorgt dafür, daß Sie diesen Vorgang auch "zu Fuß" ausführen lassen können. Meist benutzt man ihn in Programmen, die mit Dateien auf unterschiedlichen Disketten arbeiten.

Der Befehl selbst ist festgeschrieben, der einzige Parameter, der frei ist, ist die Geräteadresse des Laufwerks.

BASIC bis 2.0:

```
OPEN1,8,15,"INITIALIZE"
```

wobei das Wort "INITIALIZE" auch durch den Buchstaben "I" abgekürzt werden kann.

*****

BASIC ab 3.0:

(siehe BASIC bis 2.0)

*****

Wenn Sie im Zweifel sind, ob die Diskette, die Sie gerade eingelegt haben, eine andere ID hat, führen Sie auf jeden Fall ein INITIALIZE durch, dann können Sie sicher sein, daß die BAM in Ordnung ist.

Es ist natürlich immer möglich, daß bei den vielen Operationen, die so ablaufen, der eine oder andere Sektor einfach verloren geht. Sei es, daß trotz aller Sicherungsmaßnahmen plötzlich ein Sektor als belegt gekennzeichnet wurde, der eigentlich frei ist, oder sei es, daß Sie mit den sogenannten "Direktzugriffsbefehlen" (wir sprechen noch am Schluß des Buches davon) ein wenig herumgespielt haben. Damit können Sie auch einen eigentlich freien Sektor mit Information belegen, so daß er aus der Liste der freien Sektoren gestrichen wird.

Solange es nur wenig Sektoren sind, macht das nur wenig aus. Aber mit der Zeit können sich auf einer vielbenutzten Diskette eine ganze Menge solcher "Karteileichen" ansammeln. Dann wird es Zeit, aufzuräumen. Das Putzteufelgeschwader, das alle Sektoren, die nicht zu einer ordnungsgemäß geschlossenen Datei gehören, wieder als frei kennzeichnet, wird mit dem Befehl VALIDATE in Marsch gesetzt.

BASIC bis 2.0:

```
OPEN1,8,15,"VALIDATE"
```

Auch hier gilt: Hat das gewünschte Laufwerk eine andere Geräteadresse, dann muß statt der 8 diese Geräteadresse eingetragen werden.

*****

BASIC ab 3.0:

```
COLLECT
```

*****

Beachten Sie bitte, daß durch VALIDATE bzw. COLLECT alle Blöcke (Sektoren), die nicht zu ordnungsgemäß geschlossenen Dateien gehören, freigegeben werden. Also auch alle Blöcke, die zu Dateien gehören, die zwar beschrieben und im Directory eingetragen sind, bei denen aber nach dem Öffnen mit OPEN kein oder ein fehlerhaftes CLOSE durchgeführt wurde (Im Directory mit einem Stern "*" gekennzeichnet).

Nach einem VALIDATE muß das Laufwerk mit INITIALIZE neu initialisiert werden, da die neue Belegungsinformation zwar schon auf die Diskette geschrieben, aber noch nicht im Speicher des Diskettenlaufwerks vermerkt sind. Das gilt auch für sog. Direktzugriffsdateien!

Diese Dateien sind anders organisiert. Da VALIDATE bzw. COLLECT alle Dateieinträge löscht, die freizugebende Sektoren enthalten (also auch Sektoren von Direktzugriffsdateien), können Sie so versehentlich Dateien löschen. Wenden Sie diesen Befehl also nur bei Disketten an, die keine Direktzugriffsdateien enthalten. Relative Dateien dagegen überstehen diesen Befehl problemlos.

Das waren alle Werkzeuge, die es zum Betrieb von Laufwerken gibt. Übrig bleiben noch drei Befehle, die ich eigentlich nur der Vollständigkeit halber erwähnen möchte. Denn diese Befehle haben entweder keine Bedeutung mehr oder wenig Anwendungsmöglichkeiten. Außerdem gelten sie nur für BASIC-Versionen ab 3.0:

BACKUP

Dieser Befehl dient zum Duplizieren (Kopieren) ganzer Disketten. Aber auch dieser Befehl wurde für die Doppellaufwerke eingerichtet, so daß er bei den Einzellaufwerken, wie wir sie hier besprechen, nicht einzusetzen ist. Für diejenigen, die das Glück haben, noch solche schönen Stücke zu ihrer Sammlung zählen zu können, hier ausnahmsweise die Syntax dieses Befehls, jedoch ohne Beispiel:

```
BACKUP D<ln> TO D<ln>,U<ga>
```

wobei <ln> die Laufwerksnummer (0 oder 1) ist.

```
DCLEAR
```

DCLEAR schließt alle Kanäle, die eventuell eingerichtet worden sind. Da aber dabei noch offene Dateien nicht geschlossen werden, ist dieser Befehl nur sehr eingeschränkt einsetzbar.

```
BOOT
```

Dieser Befehl ist nur ab BASIC-Version 7.0 einsetzbar. Außerdem gehört dazu noch eine Diskette mit dem Betriebssystem CP/M. Das ist ein älteres Betriebssystem, für das es eine ganze Menge Software gibt. Es kann als Vorgänger von MS-DOS gelten und wird von diesem PC-Betriebssystem inzwischen abgelöst. Trotzdem ist CP/M ein wunderbares kleines Betriebssystem, und wenn Sie das Glück haben, noch eine Version erwischt zu haben, stehen Ihnen viele Möglichkeiten der Programmierung und Anwendung offen, von denen Sie vorher nicht geträumt haben. Für den C128 steht noch die CP/M-Version 3.0 zur Verfügung.

Damit dieses Betriebssystem in den Speicher geladen wird (denn beim Anschalten meldet sich ja nur BASIC!), muß dieser spezielle Befehl eingegeben werden. Da der Vorgang "booten" genannt wird und das Ladeprogramm "Bootstraploader" heißt, lautet der Befehl natürlich

```
BOOT "<Dateiname>"
```

Wenn Sie mehr über das Betriebssystem CP/M oder MS-DOS wissen wollen, verweise ich Sie wieder auf die Literatur, die ich im Anhang zusammengestellt habe.

*****

Wildcards - Was, wie und wozu?

Bisher waren wir immer genötigt, einen Dateinamen vollständig einzugeben, wenn wir auf eine Datei zugreifen wollten (Laden, Löschen,...). Das können wir uns von jetzt an sparen. Man hat nämlich auch hier eine feine Einrichtung eingebaut, die in anderen Betriebssystemen üblich ist: die Wildcards.

Mit Hilfe dieser Wildcards kann man den Dateinamen teilweise oder ganz ersetzen und dadurch nicht nur eine Datei, sondern auch mehrere oder alle Dateien auf einer Diskette mit einem Befehl ansprechen. Die Wildcards sind Platzhalter, die man anstelle der Buchstaben in den Dateinamen einsetzt. Manchmal ist auch der Ausdruck "Joker" gebräuchlich. Diese Zeichen stehen stellvertretend für die eigentlichen Buchstaben im Dateinamen, und es kann jeder zulässige Buchstabe dafür eingesetzt werden.

Es gibt zwei Wildcards:

- Das Zeichen "?": Für jedes "?" kann ein beliebiges Zeichen eingesetzt werden.

- Das Zeichen "*": Für einen Stern können beliebig viele Zeichen eingesetzt werden.

Bevor wir einige Beispiele besprechen, soll noch etwas über die übliche Form von Dateinamen gesagt werden. Unabhängig von der Dateinamenregelung des jeweiligen Betriebssystems hat sich folgende Anwendung in den meisten Fällen durchgesetzt (Diese Konvention existiert bei Commodore-DOS nicht, es lohnt sich aber, sie einzuhalten.).

Ein Dateiname besteht aus dem eigentlichen Namen, dem ein Punkt (".") folgt. Dem Punkt folgt eine Typbezeichnung, in der dem Benutzer mitgeteilt wird, welcher Art die Datei ist, mit der er es zu tun hat.

Der Ursprung dieser Regel liegt im Betriebssystem CP/M, wo ein Dateiname aus acht Stellen, einem Punkt und wieder drei Stellen besteht. In dem PC-Betriebssystem MS-DOS wurde diese Tradition fortgesetzt.

Als Typbezeichnung für Dateien haben sich einige Kürzel eingebürgert, die es zu kennen lohnt. Im folgenden eine kleine Tabelle, in der die gebräuchlichsten aufgeführt sind:

Kürzel	Erklärung
TXT	Dateien, die Texte im ASCII-Format enthalten.
DAT	Dateien, die Daten (Meßwerte, Rechener- gebnisse,...) enthalten.
PAS	Pascal-Programme (Quelltext)
BAS	BASIC-Programme (Quelltext)
BAK	BACKUP Datei, Sicherungskopie oder vorletzte Version eines Textes.
ASM	Assembler-Programm (Quelltext)
OVR	Overlaydatei, Zusatzfunktionen zu einem OVL Programm

Das ist nur eine kleine Auswahl von Konventionen. Selbstverständlich gibt es sehr viel mehr, und jeder Programmierer erfindet seinen eigenen Dateityp. Aber mit dieser kleinen Sammlung hat man schon einen recht guten Überblick. Solche Typenbezeichnungen zu verwenden hat einen großen Vorteil, den Sie gleich einsehen werden.

Wir üben die Wildcards nun an einigen Beispielen:

A???? Alle Dateien, deren Namen mit "A" beginnen und die noch weitere vier Zeichen im Namen enthalten.

?B??? Alle Dateinamen, deren erstes Zeichen beliebig, deren zweites Zeichen ein "B" ist und die noch drei beliebige weitere Zeichen enthalten.

BRIEF* Alle Dateinamen, die mit "BRIEF" beginnen und die beliebig viele und beliebige Zeichen danach aufweisen.

*	Alle Dateien (VORSICHT beim Löschen!)
*.BAK	Alle Dateien, die beliebige und beliebig viele Zeichen vor dem Punkt haben und die nach dem Punkt die Zeichen "BAK" enthalten.
??TEST.*	Alle Dateien, die an den ersten beiden Stellen beliebige Zeichen, an den Stellen drei bis sechs die Zeichen "TEST", an der Stelle sieben einen Punkt und hinter dem Punkt beliebig viele und beliebige Zeichen stehen haben.

Vor allem am vorletzten Beispiel kann man sehr deutlich sehen, daß die Benutzung von Typenbezeichnungen sehr bequem zum Ordnen des Directorys ist. Mit Hilfe der Wildcards kann man solchermaßen bezeichnete Dateien sehr schön in einem Befehl zu Gruppen zusammenfassen, zum Beispiel beim Kopieren oder Löschen. Überall, wo in einem Befehl ein Dateiname vorkommt, darf diese Methode der Dateibezeichnung eingesetzt werden. Vorsicht ist beim Löschbefehl angebracht, wenn als Dateibezeichnung "*" eingesetzt wird. Damit werden alle Dateien gelöscht.

Aber damit ist das Commodore-DOS noch nicht ausgereizt. Hier geht in Verbindung mit dem Gleichheitszeichen "=" noch Folgendes:

*=S	selektiert nur sequentielle Dateien
*=P	selektiert nur BASIC-Programmdateien
*=R	selektiert nur relative Dateien
*=U	selektiert nur USER-Dateien

Nun noch etwas zum Schluß dieses Kapitels: Wir haben bisher immer die verschiedene BASIC-Versionen gegenübergestellt. Dabei habe ich Ihnen eine wichtige Tatsache unterschlagen:

Alle Befehle, die bis Version 2.0 gelten, funktionieren auch genauso ab Version 3.0. Die anderen Namen der Befehle oder die andere Form sind Vereinfachungen für den Anwender, denn intern wird dieser Befehl auch nur in den zugehörigen OPEN-Befehl umgewandelt und dann erst ausgeführt. Trotzdem ist es manchmal von Vorteil, die neuen Befehle zu benutzen, falls die BASIC-Version es zuläßt. Denn sie ist oft einleuchtender und komfortabler als die komplizierte OPEN-Version.

6. Allgemeines über Dateien, Fehler

"Dead denotes a state of a character with all bytes set to BLANC (=20H)"

Herbert A.[7]

Wir erfahren, wie man Dateien anordnen kann, was ST bedeutet, wie Fehler behandelt werden, was ein EOF ist und wie man ihn findet.

Dateiorganisation

Um Daten auf einer Diskette, Festplatte oder auf Band zu speichern, muß man sich auf gewisse Organisationsprinzipien einigen. Denn man will die Daten ja möglichst zuverlässig und schnell wiederfinden. Das bedeutet, daß die Daten nicht "willenlos" auf der Platte verteilt werden dürfen.

Zunächst ist es sinnvoll, zu entscheiden, ob die Daten zeichenweise oder in sogenannten Datensätzen abgespeichert werden. Texte kann man zeichenweise anordnen, eine Kundendatei dagegen mit Rubriken wie Name, Straße, Stadt, etc. wird natürlich nicht zeichenweise gespeichert. Hier muß man alle Daten, die einen Kunden betreffen, zu einer Einheit zusammenfassen. Solch eine Einheit nennt man Datensatz. Ein Datensatz ist die Zusammenfassung aller Eigenschaften eines Objektes, das zusammen mit anderen, gleichartigen Objekten in einer Datei gespeichert werden soll. Ein Datensatz heißt auch record.

Nachdem man entschieden hat, auf welche Weise die Daten angeordnet werden sollen, kommt als nächstes die Wahl des Prin-

---

7 Herbert A.: The Tragedy of Hamlet, Reference Manual

zips. Es gibt mehrere Prinzipien, nach denen Daten auf ihren Datenträgern angeordnet werden. Die für Mikrocomputer gebräuchlichen möchte ich hier vorstellen:

Sequentielle Dateien

Die Daten können zeichenweise oder in Datensätzen angeordnet werden. Dabei können sich die Datensätze in der Länge unterscheiden. Das ist auch das Hauptmerkmal von sequentiellen Dateien. Die Daten werden hintereinander angeordnet. Wird ein Datensatz gesucht, muß die Datei wegen der unterschiedlichen Länge der Datensätze von Anfang an durchsucht werden, bis entweder der Datensatz gefunden oder bis das Ende der Datei erreicht worden ist. Es ist sehr umständlich, Datensätze zu entfernen oder einzufügen.

Dateien mit wahlfreiem Zugriff

Die Daten müssen in Datensätzen angeordnet werden. Dabei müssen die Datensätze alle dieselbe Länge haben. Da die Länge eines Datensatzes bekannt ist (weil konstant), kann man jeden Datensatz mit Hilfe seiner Nummer sofort ansprechen (relative Positionierung), ohne sich durch die ganze Datei hindurch hangeln zu müssen. Es ist ziemlich einfach, Datensätze zu löschen und einzufügen. Dateien mit wahlfreiem Zugriff heißen Random Access Files; bei Commodore heißen sie relative Dateien.

Indexsequentielle Dateien

Die Informationen werden in Datensätzen relativ angeordnet. Der Unterschied zu den rein relativen Dateien ist, daß aus jedem Datensatz ein Teilelement (der Index) zusätzlich in einer sequentiellen Datei (Indexdatei) zusammen mit der Nummer seines Datensatzes gespeichert wird. Da diese Indexdatei ja nur einen Teil der Gesamtdatei beinhaltet, kann Sie im Computerspeicher bleiben und dort leicht verändert werden. So wird eine indexsequentielle Datei nicht dadurch sortiert, daß alle Datensätze sortiert werden, sondern es wird nur die kleinere Indexdatei im Computerspeicher sortiert.

Ganz am Anfang des Buches haben wir schon ein Beispiel für eine sequentielle Datei kennengelernt. Erinnern Sie sich noch? Ich habe gezeigt, wie ein Text abgespeichert wird. (Falls Sie sich nicht erinnern, schlagen Sie es noch mal nach.) Ein Text ist natürlich eine sequentielle Datei.

Wenn Sie sich das Beispiel am Anfang des Buches genau ansehen, dann fällt Ihnen das "Zeichen" <EOF> auf. Das Zeichen <EOF> bedeutet End of File, Dateiende. Dabei habe ich Ihnen um des allgemeinen Prinzips willen unterschlagen, daß es bei unseren Commodore-Laufwerken dieses Zeichen nicht explizit gibt. Aber es gibt trotzdem eine Möglichkeit, das Dateiende zu erkennen.

Die Statusvariable

Im Commodore-BASIC gibt es eine vordefinierte Variable, die die Information enthält, die wir benötigen: die Variable ST.

Der Name ST kommt von "Status". Deswegen wird ST auch die Statusvariable genannt. In dieser Variablen werden alle notwendigen Informationen abgelegt, die den Datentransfer von und zu den Geräten Cassettenrecorder und Diskettenlaufwerk betreffen. Da wir hier nicht über Cassettenbetrieb sprechen, werden nur die Teile erwähnt, die den Diskettenbetrieb betreffen. Man sollte meinen, daß in einer Variablen nur eine Zahl gespeichert werden kann. Aber dabei vergißt man, daß eine Variable im Speicher eine Struktur ist, die aus acht Stellen besteht, die durch Zeichen des Zweiersystems gefüllt werden. Da ST nur eine Speicherzelle belegt, besteht ST aus 8 Bit. Man kann nun jedes Bit dieser Variablen für sich alleine betrachten. So hat man Platz für acht "Ja/Nein"-Informationen. Und wenn man jedem Platz noch eine bestimmte Bedeutung gibt, dann kann man sehr viel Information unterbringen.

Sie fragen sich nun aber mit Recht, wie man aus einer BASIC-Variablen Informationen über die einzelnen Bits bekommt. Das kann mit Hilfe der logischen Operationen AND geschehen. Falls

Sie noch nicht sicher in der Anwendung solcher Operationen sind, empfehle ich Ihnen den Anhang oder die umfangreiche Literatur[8]. Um ein Bit abzufragen, muß man zunächst wissen, welches Bit was bedeutet. Deswegen hier eine kleine Übersicht:

Bitposition	dezimaler Wert	Bedeutung
0	1	Time-out Schreiben
1	2	Time-out Lesen
2		
.		
.	(für Cassettenbetrieb)	
.		
5		
6	64	Ende der Datei
7	128	Ende des Blocks

Eine kurze Erklärung zum "Time-out": Wenn ein Peripheriegerät vom Computer angesprochen wird, bekommt dieses Gerät eine gewisse Zeit, um sich bereit zu melden. Verstreicht diese Zeit, ohne daß das Gerät seine Bereitschaft meldet (nicht angeschaltet, nicht angeschlossen oder defekt), so findet ein Time-out statt, und der Computer weiß, daß das Gerät nicht bereit ist, Daten zu empfangen. Meistens erfolgt dann eine Fehlermeldung.

Um zu erkennen, daß eine Datei zu Ende ist, muß Bit 6 mit der Zahl 64 maskiert werden. Das geht folgendermaßen: Wenn zwei Bytes mit AND verknüpft werden, dann bleiben als Ergebnis nur die Bits übrig, die bei beiden Bytes an derselben Position gesetzt (auf 1) waren. Da die Zahl 64 nur an der Stelle 6 eine 1 hat, gibt es als Ergebnis nur dann das Ergebnis 'wahr', wenn beim Statusbit auch an der Position 6 eine 1 steht. Was an den

8 z.B. Schönleber: Hitchhacker's Guide to BASIC, Kiel 1987, Verlag Claus Schönleber

Wirth: Systematisches Programmieren, Stuttgart 1985, 3.Auflage

Wirth: Algorithmen und Datenstrukturen, Stuttgart 1983, 3.Auflage

anderen Stellen steht, ist nicht erheblich, da durch die Nullen der 64 alle anderen Werte der Statusvariablen "ausmaskiert" werden.

In BASIC sieht das dann zum Beispiel so aus:

```
EOF = 64 AND ST
```

Dadurch bekommt die Variable EOF den Inhalt 64, wenn das Dateiende noch nicht erreicht ist und den Inhalt 0, wenn das Dateiende erreicht ist. Daraus kann man sich mit Hilfe der etwas laxen Programmierweise von BASIC eine wunderschöne Abfrage nach dem Dateiende basteln. Wir brauchen nämlich für die Frage nach dem Zeichen EOF (IF EOF THEN...) in EOF einen logischen Wert. Die Zahl 64 ist aber alles andere als ein logischer Wert, der ja bekanntlich (z.B. in Pascal) "true" oder "false" sein muß. Aber, BASIC sei's gepriesen, hier wird die Zahl 0 als FALSE interpretiert und alles, was ungleich 0 ist, als TRUE!

So kommen wir auf einem Umweg zu unserem EOF-Zeichen. Und die Variable EOF im obigen Beispiel kann ohne Probleme zwischen IF..THEN... eingesetzt werden, ohne die Syntax oder Semantik von BASIC zu verletzen. So kann man denn dieses Zeichen wunderbar abfragen, was wir auch in Zukunft so tun werden. Sollten Sie diesem schnell vorübergeeilten Gedanken nicht sofort folgen können, sei hier auf den Anhang verwiesen, in dem dieses Thema noch einmal ausführlich dargestellt wird.

Fehlerkanal, DS, DS$ - Hilfe zur Fehlererkennung

Fehler beim Diskettenbetrieb werden in Commodore-Homecomputern durch zwei verschiedene Arten angezeigt. Erstens blinkt eine Leuchtdiode, die am Laufwerk sitzt, ziemlich hektisch. Das bedeutet, irgendetwas ist schiefgelaufen, und was das war, das sollte vor der nächsten Aktion erst einmal analysiert werden. Die Analyse läuft dann, abhängig von der BASIC-Version, entweder über den Fehlerkanal oder über die Fehlervariablen.

BASIC-Versionen, die Versionsnummern ab 3.0 besitzen, benutzen die Fehlervariablen, alle Versionen davor (bis 2.0) benutzen den Fehlerkanal. Den wollen wir zuerst besprechen:

Der Fehlerkanal

Um einen Fehler, der beim Diskettenbetrieb auftrat, zu analysieren, muß, nachdem das Blinken des Laufwerks erkannt wurde, die Fehlermeldung auf den Bildschirm gebracht werden. Dazu ist folgendes kleines Programm notwendig:

```
10 OPEN1,8,15
20 INPUT#1,EN,ET$,T,S
30 PRINT EN,ET$,T,S
40 CLOSE1
```

Dabei bedeuten:

EN	Error number: Nummer des Fehlers
ET$	Error Text: Fehlermeldung im Klartext
T	Track: Spur auf der Diskette, auf der der Fehler auftrat
S	Sektor: Sektor, in dem der Fehler auftrat

Nachdem dieses kleine Programm aufgerufen wurde, hört die Diode auf zu blinken, und es erscheint auf dem Bildschirm zuerst eine Zahl (Fehlernummer), dann ein Text (Fehlermeldung) und zum Schluß zwei Zahlen, die die Spurnummer und den Sektor auf dieser Spur angeben, wo der fehlerhafte Vorgang stattgefunden hat.

Versuchen Sie zum Beispiel, ein Programm zu laden, dessen Dateiname nicht in dem Directory verzeichnet ist. Zum Beispiel:

```
LOAD "DATEIOHNENAMEN",8
```

Danach wird das Laufwerk versuchen, die Datei zu finden (Meldung "SEARCHING"). Da der Name aber nicht in meinem Beispieldirectory steht, wird das Suchen vergeblich sein. Die

Leuchtdiode fängt zu blinken an. Nachdem Sie das obige Programm eingetippt und ausgeführt haben, erscheint auf dem Bildschirm folgender Text:

```
62 FILE NOT FOUND 0 0
```

Da die Datei nicht existiert, erfolgt Fehler Nummer 62 mit der Bedeutung "Datei nicht gefunden", deren englisches Äquivalent als Fehlertext erscheint. Als Spur- bzw. Sektornummer wird 0 angegeben, da der Fehler nicht explizit in einem bestimmten Sektor der Diskette passierte. Es wurde eben einfach nichts gefunden.

Am besten hängen Sie das Programm zur Anzeige einer Fehlermeldung an das Ende Ihrer BASIC-Programme, beispielsweise immer ab Zeilennummer 50000. Sollte dann einmal ein Fehler auftreten, können Sie mit

```
GOTO 50000
```

den Fehlerkanal auslesen und die Fehlerursache bestimmen.

Fehlervariablen

Besser haben es die Benutzer der BASIC-Versionen ab Versionsnummer 3.0, denn dort gibt es die Fehlervariablen. Diese Fehlervariablen ersparen das umständliche Hantieren mit dem Fehlerkanal. Diese Variablen haben besondere Namen, genau wie die Statusvariable:

DS enthält nur die Fehlernummer

DS$ enthält die komplette Fehlermeldung im Format:

```
<EN>,<Meldung im Klartext>,<T>,<S>
```

Sie sehen, die Syntax ist also wie beim Fehlerkanal!

Wenn also ein Fehler aufgetreten ist, enthält DS die Nummer des Fehlers, und in D$ ist die gesamte Fehlermeldung mit Fehlernummer, Fehlertext, Spur- und Sektornummer enthalten.

Am besten tritt kein Fehler auf, aber auch für den Fall, daß kein Fehler auftritt, können der Fehlerkanal oder die Fehlervariablen benutzt werden. Die Fehlermeldung sieht dann so aus:

```
00 OK 00 00
```

Die "Fehlernummer" ist dann 00, der Fehlertext ist OK, was selbstverständlich bedeutet, daß alles in Ordnung ist.

7. Sequentielle Dateien

"Ein PAL...ist etwas, das wir nicht sehen oder nicht sehen können oder das unser Gehirn uns nicht sehen läßt, weil wir denken, es sei das Problem anderer Leute. Genau das bedeutet PAL. Problem Anderer Leute."

Douglas Adams[9]

Hier erfahren wir genau, wie man sequentielle Dateien aufbaut, Informationen aus Ihnen zurückholt und manipuliert. Am Ende des Kapitels bemühen wir uns um einen ersten Versuch, eine Datei zu sortieren.

In den letzten Kapiteln haben wir zu den "normalen" Befehlen auch die "Kosmetik"-Befehle der höheren BASIC-Versionen kennengelernt. Da bei Benutzung dieser Befehle auch nichts anderes an das Laufwerk geschickt wird, als der gute alte OPEN-Befehl mit seinen wirren Befehlskürzeln, werde ich in Zukunft auf diesen Luxus nur am Rande eingehen. Denn der OPEN-Befehl ist konsequenter in seiner Form als die Vielzahl der Befehle, die mit "D" anfangen.

Bevor ich genau darauf eingehe, wie eine sequentielle Datei aufgebaut ist, will ich Ihnen zuerst zeigen, wie man überhaupt eine erzeugt und die Information dann wiederbekommt. Vor dem Senden der Daten in eine Datei, muß man diese natürlich auf der Diskette erst eröffnen. Dann erst kann der Datentransfer beginnen. Nach getaner Arbeit muß die Datei wieder geschlossen werden. Nehmen wir an, aus irgendeinem dubiosen Grunde wollen wir die Quadrate der Zahlen von 1 bis 20 auf Diskette

9 Douglas Adams: Das Leben, das Universum und der ganze Rest, Ullstein 1988

speichern. Ich habe dieses einfache, aber praxisferne Beispiel gewählt, damit Sie am Anfang nicht gleich die Übersicht verlieren.

Also los:

```
10 OPEN1,8,2,"QUADZAHL,S,W"
20 FOR x = 1 TO 20
30 PRINT#1,x * x
40 NEXT x 50 CLOSE1
```

Wenn das Programm ausgeführt worden ist, erscheint in dem Directory eine neue Datei (Klar!) mit einer anderen Typbezeichnung. Bisher hatten unsere BASIC-Programme immer den Typ "PRG", die neue Datei hat den Typ "SEQ". Man braucht nicht zu raten, wofür das Kürzel steht; es soll natürlich eine sequentielle Datei bezeichnen.

Schauen wir uns den benutzten OPEN-Befehl etwas genauer an. Die allgemeine Form sieht so aus:

```
OPEN<lfn>,<ga>,<sad>,"@:<dateiname>,<typ>,<modus>"
```

Die Zeichenfolge "@:" ist auch hier nur dann notwendig, wenn eine bereits bestehende Datei gleichen Namens überschrieben werden soll.

Die Bedeutungen der einzelnen Komponenten:

<lfn>	logische Filenummer
<ga>	Geräteadresse
<sad>	Sekundäradresse 0,1: Sind für LOAD und SAVE reserviert. 2..14: frei verfügbar. 15: Für den Fehlerkanal reserviert
<dateiname>	Name der Datendatei
<typ>	Art der Datenorganisation S = Sequentiell
<modus>	Art des Zugriffs R = READ (Lesen)

W = WRITE (Schreiben)
A = APPEND (Anhängen)
M = (wird noch besprochen)

Wir haben die Sekundäradresse ja schon kennengelernt. Trotzdem will ich noch einmal zur Gedächtnisstütze eine kurze Erklärung bringen: Die Sekundäradresse hat die Aufgabe, mehrere Dateien, die gleichzeitig auf dasselbe Gerät geschickt werden sollen, logisch zu unterscheiden.

In unserem Beispiel wird eine Diskettendatei namens 'QUADZAHL' geöffnet, die die Dateinummer 1 erhält und sich auf dem Gerät mit der Geräteadresse 8 (= Laufwerk 1) befindet. Art der Datenorganisation ist S (= sequentiell), Art des Zugriffs ist W (= Schreiben), d.h. wir wollen Daten in die Datei schreiben.

Die "Kosmetikversion" sieht so aus:

```
DOPEN#<lfn>,"<dateiname>",U<ga>,<typ>
```

wobei <typ> durch "S" für sequentielle Dateien ersetzt wird.

Der CLOSE-Befehl sieht dann so aus:

```
DCLOSE#<lfn> ON U<ga>
```

Wir bleiben jedoch bei der normalen OPEN-Version.

Der CLOSE-Befehl ist sehr wichtig, nachdem Daten in die Datei geschrieben worden sind. Denn mit PRINT# werden die Daten nicht sofort auf die Diskette geschrieben. Das würde zu viele Umstände bedeuten, denn man hätte wegen jedes Zeichens das Laufwerk anzuwerfen und die Datei zu verändern. (Dateiende suchen, Daten in den nächsten freien Sektor, Directory aktualisieren,...) Deswegen gibt es im Laufwerk einen eigenen Speicher, den sog. Buffer oder auf deutsch Pufferspeicher. Hier werden die Daten, die vom Computer kommen und in die Datei sollen, solange hineingeschrieben, bis er voll ist. Erst dann wird das ganze gesammelte Paket auf die Diskette geschrieben. Der

CLOSE-Befehl bewirkt nun, daß der Rest der Daten im Puffer auf die Diskette geschrieben wird. Vergißt man nach dem Beschreiben einer sequentiellen Datei das CLOSE, so geht der Inhalt des Puffers verloren und das Directory wird nicht aktualisiert. Eine solche Datei ist nicht ordnungsgemäß geschlossen und hat als Zeichen in der Directory einen Stern vor dem Dateityp. Den CLOSE-Befehl zu vergessen, das ist ein beliebter Fehler, der auch vielen erfahrenen Programmierern manchmal noch passiert. Deswegen sollten Sie ein Programm immer daraufhin überprüfen, ob vielleicht nicht ein CLOSE-Befehl vergessen worden ist.

So, wir haben unsere errechneten Daten auf Diskette abgespeichert. Der nächste Abschnitt beschäftigt sich mit dem Laden solcher Daten in den Speicher, also mit dem umgekehrten Vorgang. Aber wir wollen uns zunächst noch das neue Inhaltsverzeichnis unserer Beispieldiskette ansehen. Wir gehen davon aus, daß mindestens das Programm 'QUADRATE' und die sequentielle Datei 'QUADZAHL', die ja unsere Daten (Quadratzahlen) enthält, auf Diskette gespeichert sind. Außerdem wollen wir das Programm, mit dem wir die sequentielle Datei erzeugt haben, unter dem Namen "SEQTEST" abspeichern.

```
0 "ERSTE DISKETTE " A1 2A
1 "QUADRATE" PRG
1 "SEQTEST" PRG
2 "QUADZAHL" SEQ

660 BLOCKS FREE.
```

Der letzte Eintrag bezieht sich also auf unsere Quadratzahlentabelle, die wir gerade auf Diskette gespeichert haben. Sollte das Inhaltsverzeichnis bei Ihnen etwas anders aussehen, brauchen Sie keine Sorge zu haben, dann haben Sie einfach die Übungen nicht ganz korrekt nachvollzogen. Das ist aber nicht schlimm. Statt PRG steht hinter dem Namen unserer Quadratzahlendatei der Eintrag SEQ, was einfach bedeutet, daß es sich hierbei nicht um ein ausführbares BASIC-Programm, sondern um eine sequentielle Datei handelt. Je nachdem, wieviel Daten sie enthält, werden entsprechend viele Blöcke belegt.

Nun wollen wir uns die Quadratzahlen aus der Datei "QUADZAHL" wiederbeschaffen. Zuerst müssen wir die Datei zum Lesen öffnen:

```
OPEN1,8,2,"QUADZAHL,S,R"
```

Das "R" zeigt an, daß die Datei nur zum Lesen geöffnet wird. Es gibt nun zwei Befehle, die ein Lesen der Daten in den Computerspeicher erlauben: INPUT# und GET#. Bitte bedenken Sie, daß INPUT# mit dem bekannten INPUT-Befehl nur das Prinzip der Eingabe gemeinsam hat. Es sind trotzdem zwei verschiedene Befehle! Dasselbe gilt für GET# und GET.

Der INPUT#-Befehl funktioniert ähnlich wie INPUT. Beim C64 darf der String, der mit INPUT# eingelesen wird, nur 88, beim C128 nur 160 Zeichen enthalten. Ist der String länger, führt das zu einer Fehlermeldung ("STRING TOO LONG"). INPUT# liest solange aus einer Datei, bis ein Zeilenendezeichen oder EOL (End of line) auftritt. Es handelt sich üblicherweise um das Zeichen 13, das mit "Carriage Return" bezeichnet wird und auf der Tastatur den Namen "Return" hat.

Einige Zeichen können mit dem INPUT#-Befehl genausowenig wie mit seinem Kollegen INPUT in einen String eingelesen werden. Das sind die sog. Trennzeichen. Es handelt sich um die Sonderzeichen ",", ";" und ":". Er behandelt sie eben als Trennzeichen und nicht als Daten. Beachten Sie auch, daß Sie nur Stringvariablen im Zusammenhang mit INPUT# benutzen, da nach Strings, die versehentlich einer numerischen Variablen zugewiesen wurden, der Fehler "FILE DATA ERROR" ausgegeben wird. Diese freiwillige Beschränkung ist kein Hindernis, denn mit Hilfe der VAL-Funktion können Strings bekanntlich in Zahlen umgewandelt werden.

Im Gegensatz zum INPUT#-Befehl holt der GET#-Befehl nur jeweils ein Zeichen. Beim GET#-Befehl ist jedoch eines zu beachten: Sobald das Zeichen 0 auftaucht, wird der Variablen,

die das Argument von GET darstellt, nicht 0, sondern der Leerstring ("") zugewiesen. Abhilfe kann folgende Befehlsfolge schaffen (Datei hat log. Filenummer 1):

```
GET#1,X$ X=ASC(X$+CHR$(0)) X$=CHR$(x)
```

Andere Dinge sind bei GET# nicht zu beachten.

Wir werden mit Hilfe von INPUT# Daten aus der Datei lesen. Das komplette Programm:

```
10 OPEN1,8,2,"QUADZAHL,S,R"
20 FOR x = 0 TO 10
30 INPUT#1,q
40 PRINT q
50 NEXT x
60 CLOSE1
70 END
```

Wenn Sie das Programm eingetippt, abgespeichert und dann gestartet haben, wird auf dem Bildschirm unsere Quadratzahlentabelle erscheinen, obwohl im ganzen Programm kein Quadrat ausgerechnet wird. Das ist der Beweis, daß die Zahlen von der Diskettendatei kommen.

Wir wollen uns nun an ein größeres Projekt machen. Mit diesem Beispiel werden wir einige Details üben, die man leicht auf größere Problemstellungen übertragen kann. Außerdem soll ein solches Beispiel natürlich auch gleich irgendwie Etiketten beschriften nützlich sein. Also bauen wir ein Programm zum Beschriften von Etiketten. Etiketten braucht man nicht nur zum Archivieren von Disketten. Vom Computer bedruckte Etiketten machen sich auch gut auf Geschäftsbriefen, sowohl als Adress- wie auch als Absenderaufkleber. Es gibt also viele Anwendungen, bei denen man dieses Programm Gewinn bringend einsetzen kann. Wie vor jedem Projekt hat man sich zu überlegen, was das Programm können soll. Wir fordern folgende Fähigkeiten:

- Schreiben und Korrigieren des Textes
- Drucken des Textes

- Listen des Textes
- Löschen des Textes im Speicher
- Speichern und Laden des Textes von Diskette

Das Programm soll in (besonders einfacher) Menütechnik geschrieben werden, was einen modularen und damit klaren Aufbau erlaubt. Daß ich im übrigen in einem Buch über Diskettenlaufwerke ein wenig über Programmiertechnik plaudere, ist durchaus angebracht. Denn was soll man mit Laufwerken anfangen, wenn man sie nicht ordentlich zu programmieren weiß?

Um den Text des Etiketts manipulieren zu können, muß er im Speicher verfügbar sein. Wir richten deswegen ein Feld mit neun Elementen ein, die die Zeilen darstellen. Damit man mit den Indizes eins bis neun arbeiten kann, ignorieren wir einfach die Existenz des Feldelements mit dem Index Null. Außerdem lassen wir die erste und letzte Zeile als Rand frei. Als erstes formulieren wir die Initialisierung und das Menü:

```
10 REM ***** DISKETTEN-ETIKETTEN-DRUCK-PROGRAMM
20 REM *** Z$() TEXTFELD, E$ EINGABEBEFEHL
30 DIM Z$(9)
40 PRINT:PRINT
50 PRINT "DEDP V1.0":PRINT
60 PRINT "(E)INGEBEN, (K)ORRIGIEREN, (D)RUCKEN, (Z)EI GEN, (N)EU,
(S)ICHERN, (L)ADEN, (Q)UIT":INPUT E$
70 IF E$="E" THEN GOSUB 200
80 IF E$="K" THEN GOSUB 300
90 IF E$="D" THEN GOSUB 400
100 IF E$="Z" THEN GOSUB 500
110 IF E$="N" THEN GOSUB 600
120 IF E$="S" THEN GOSUB 700
130 IF E$="L" THEN GOSUB 800 130 IF E$="Q" THEN GOSUB 900
140 PRINT "*** FALSCHE EINGABE!":GOTO 60
```

Jetzt bauen wir das Programm Modul für Modul zusammen. Die Teile, die nicht auf die Diskette zugreifen, werden ohne Kommentare in das Listing integriert:

```
200 REM +++ EINGABE, ZEILENLAENGE 30 ZEICHEN
210 PRINT "30 ZEICHEN PRO ZEILE!"
220 FOR X=2 TO 8
230 PRINT X;". ZEILE: ";:INPUT Z$(X)
```

```
240 Z$(X)=LEFT$(Z$(X),30)
250 NEXT X
260 E$="":RETURN
300 REM +++ KORREKTUR
310 INPUT "WELCHE ZEILE";X
320 IF (X<2) OR (X>8) THEN PRINT "*** NICHT ERLAUBT!": E$="":RETURN
330 PRINT "ALTE ZEILE: ";Z$(X)
340 INPUT "NEUE ZEILE: ";Z$(X)
350 E$="":RETURN
400 REM +++ DRUCKEN
410 PRINT:INPUT "WIE OFT";C
420 OPEN1,4
430 FOR X=1 TO C
440 FOR Y=1 TO 9
450 PRINT#1,Z$(Y)
460 NEXT Y
470 NEXT X
480 CLOSE1
490 E$="":RETURN
500 REM +++ ZEIGEN, LISTEN
510 FOR X=1 TO 9
520 PRINT X;": ";Z$(X)
530 NEXT X
540 E$="":RETURN
600 REM +++ TEXT LOESCHEN
610 FOR X=1 TO 9
620 Z$(X)=""
630 NEXT X
640 E$="":RETURN
```

So, nach dieser Vorarbeit können wir uns an die beiden Module machen, die sich mit der Diskette beschäftigen. Ich werde diesen Teil auch ausführlicher kommentieren.

Um verschiedene Texte auf der Diskette speichern zu können, müssen die Dateien, die die Texte enthalten, verschiedene Dateinamen besitzen. Da ein Dateiname ein String ist, kann ein Dateiname über die Tastatur eingegeben werden. Diese Variable wird dann in den OPEN-Befehl eingesetzt. Das werden wir jetzt tun:

```
700 REM +++ SICHERN
710 PRINT:INPUT "DATEINAME";DN$
720 OPEN1,8,2,DN$ + ",S,W"
730 FOR X=1 TO 9
740 PRINT#1,Z$(X)
750 NEXT X
```

```
760 CLOSE1
770 E$="":RETURN
```

Hinter dem Dateinamen müssen alle Zeichen, die zum Befehl gehören, in einem Textstring erscheinen (siehe Zeile 720). Vergessen Sie dabei die Kommata nicht!

Nun folgt das Laden des Textes. Im Prinzip müssen wir den gesamten Vorgang umkehren. Die Datei muß zum Lesen geöffnet werden, die Textzeilen werden in die Feldelemente eingelesen, und dann muß die Datei wieder geschlossen werden:

```
800 REM +++ LADEN
810 PRINT:INPUT "DATEINAME";DN$
820 OPEN1,8,2,DN$ + ",S,R
830 FOR X=1 TO 9
840 INPUT#1,T$:Z$(X)=T$
850 NEXT X
860 CLOSE1
870 E$="":RETURN
```

In Zeile 840 muß für den INPUT-Befehl eine normale Variable eingeführt werden, denn der INPUT#-Befehl verträgt keine Feldelemente. Also muß der Einlesevorgang in zwei Schritten erfolgen:

- Einlesen in eine normale Stringvariable
- Zuweisen des Stringvariableninhalts in das entsprechende Feldelement

Zum Schluß noch der Teil zum Beenden:

```
900 REM +++ ENDE
910 INPUT "TEXT SICHERN (J/N) ";A$
920 IF A$="J" THEN GOSUB 700
930 PRINT:PRINT "*** ENDE DEDP V1.0"
940 END
```

Hier ist eine kleine Sicherung eingebaut, um zu verhindern, daß in der Hektik der Arbeit ein eingegebener Text verloren geht. Dabei erweist sich im übrigen der GOSUB- Befehl als sehr

hilfreich. Probieren Sie das Programm einfach mal aus, und versuchen Sie, den roten Faden vor allem in den beiden Diskettenmodulen zu verfolgen.

Dieses Programm hat natürlich noch einige kleine Lücken. Zum Beispiel möchten Sie einen Text in eine Datei speichern, die schon existiert. Entweder war es ein Versehen, oder Sie möchten die Datei wirklich überschreiben. Oder irgendetwas geht beim Diskettenbetrieb schief, und das Diskettenlaufwerk meldet einen Fehler durch Blinken der Leuchtdiode. Außerdem kann der INPUT-Befehl nicht alle Zeichen vertragen. Es gibt also noch eine Menge Arbeit für Sie!

Aufgabe: Bauen Sie ein zusätzliches Modul, das den Fehlerkanal abfragt und die Fehlermeldung auf dem Bildschirm ausgibt. Bauen Sie ein weiteres Modul, das auf verschiedene Fehler reagiert.

Das Etikettenprogramm ist eine schöne, einfache Übung, um mit sequentiellen Dateien vertraut zu werden. Aber in diesem Programm ist ja die Länge der Datei bekannt (9 Zeilen Text), deswegen darf die INPUT#-Anweisung in eine FOR-NEXT-Schleife eingebettet sein.

Wie geht das aber, wenn man das nicht weiß?

Um dies ein wenig zu üben, wollen wir ein anderes Programm schreiben. Das neue Programm soll uns helfen, ein wenig Ordnung in unsere Kuchenrezeptsammlung zu bringen. Natürlich kann man statt Kuchenrezepte auch andere Rezepte für Cocktails, Gerichte oder für ähnliche Anwendungen mit diesem Programm verwalten. Die Lösung für dieses Problem mag vielleicht nicht die eleganteste sein, aber sie genügt den Kriterien für Einfachheit und Übersichtlichkeit.

Das Programm soll folgende Fähigkeiten besitzen:

- Eingabe von neuen Rezepten
- Ausgabe von Rezepten (Bildschirm oder Drucker)
- Liste aller Rezepte

Jedes Rezept ist in einer eigenen sequentiellen Datei untergebracht, die Namen der Dateien werden in einer gesonderten Datei untergebracht. Das Sortieren sparen wir uns, das wäre jetzt noch zu früh und würde nur verwirren. Außerdem funktioniert dies mit sequentiellen Dateien außerordentlich schlecht.

Die Rezeptsammlung

Die Datenstrukturen für dieses Programm sehen so aus: Für ein Rezept wird Text eingegeben. Der Text wird zeilenweise über INPUT eingelesen und sofort in die Datei abgespeichert. Somit lassen wir hier Korrekturen nur innerhalb einer Zeile zu. Sie können sich ja später damit beschäftigen, wie man das verbessern könnte. Da man nicht weiß, wie lang das Rezept wird, geben wir am Schluß das Wort "ENDE" ein. Der Name des Rezeptes wird als Dateiname benutzt und sollte deshalb 16 Zeichen nicht überschreiten. Also, an die Arbeit:

```
10 REM ***** REZEPTSAMMLUNG V1.0
20 REM
30 PRINT:PRINT "(E)INGEBEN, (A)USGEBEN, (L)ISTE, (Q)UIT"
40 INPUT E$
50 IF E$="E" THEN GOTO 100
60 IF E$="A" THEN GOTO 200
70 IF E$="L" THEN GOTO 300
75 IF E$="Q" THEN PRINT "*** ENDE":END
80 PRINT "+++ FALSCHE EINGABE!":GOTO 30
90 REM
100 REM +++ EINGABE
110 INPUT "WIE HEISST DAS REZEPT";D$
120 IF (LEN (D$) > 16) THEN D$=LEFT$(D$,16)
130 OPEN3,8,3,"LISTE,S,A":PRINT#2,D$:CLOSE3
140 OPEN2,8,2,D$ + ",S,W"
150 INPUT Z$
160 IF (Z$ <> "ENDE") THEN PRINT#2,Z$:GOTO 150
170 CLOSE2
180 GOTO 30
190 REM
200 REM +++ AUSGABE
```

```
210 INPUT "WIE HEISST DAS REZEPT";D$
220 IF (LEN (D$) > 16) THEN D$=LEFT$(D$,16)
230 OPEN2,8,2,D$ + ",S,R"
250 INPUT#2,Z$
260 EOF = (64 AND ST)
270 IF NOT EOF THEN PRINT Z$:GOTO 250
280 CLOSE2
290 GOTO 30
300 REM
310 REM +++ LISTE
320 OPEN2,8,2,"LISTE,S,R"
330 INPUT#2,N$
340 EOF = (64 AND ST)
350 IF NOT EOF THEN PRINT N$:GOTO 330
360 CLOSE2
370 GOTO 30
```

Das Programm ist in diesem Stadium noch sehr primitiv und dumm. Aber es zeigt schon, wie man mit einem unbekannten Dateiende umgeht. Es wird einfach mit INPUT# eine Eingabe von der Datei angefordert. Wenn das Dateiende noch nicht erreicht ist, geht alles gut, und die Eingabevariable bekommt den entsprechenden Wert zugewiesen. Im anderen Fall erfolgt in ST die Meldung "Dateiende erreicht", und das Programm terminiert (stoppt). So kann das Dateiende einfach abgefragt werden.

Eines stört aber noch an dem schönen, kleinen Programm! Beim Ausgeben der Rezepte muß man ja einen Dateinamen angeben. Ebenso beim Eingeben. Da kann es vorkommen, daß der Dateiname schon (Eingabe) oder noch nicht (Ausgabe) existiert. In beiden Fällen muß vorher eine Überprüfung stattfinden. Das geht auf zwei Arten.

1) Man prüft mit Hilfe der Fehlervariablen bzw. des Fehlerkanals die Art des Fehlers (Datei nicht/schon vorhanden) und reagiert mit einem Hinweis, daß doch bitte ein anderer Name verwendet werden soll.

2) Man prüft, ob der Dateiname in der Liste vorkommt oder nicht. Danach kann ebenfalls eine entsprechende Meldung ausgegeben werden.

Die Datei "LISTE", die alle Datei- bzw. Rezeptnamen enthält, wird systematisch, d.h. in geordneter Folge nach dem Namen durchsucht. Ist der Name gefunden oder die Datei zu Ende, kann die Suche abgebrochen und entsprechend weiterverfahren werden. Das veränderte Programm sieht dann so aus:

```
10 REM ***** REZEPTSAMMLUNG V1.0
20 REM
30 PRINT:PRINT "(E)INGEBEN, (A)USGEBEN, (L)ISTE, (Q)UIT"
40 INPUT E$
50 IF E$="E" THEN GOTO 100
60 IF E$="A" THEN GOTO 200
70 IF E$="L" THEN GOTO 300
75 IF E$="Q" THEN PRINT "*** ENDE":END
80 PRINT "+++ FALSCHE EINGABE!":GOTO 30
100 REM +++ EINGABE
110 INPUT "WIE HEISST DAS REZEPT";D$
120 IF (LEN (D$) > 16) THEN D$=LEFT$(D$,16)
130 GOSUB 1000
140 IF NOT EOF THEN PRINT "+++ REZEPT SCHON VORHANDEN!" :GOTO 30
150 OPEN3,8,3,"LISTE,S,A":PRINT#2,D$:CLOSE2
160 OPEN2,8,2,D$ + ",S,W"
170 INPUT Z$
180 IF (Z$ <> "ENDE") THEN PRINT#1,Z$:GOTO 170
185 CLOSE1
190 GOTO 30
195 REM
200 REM +++ AUSGABE
210 INPUT "WIE HEISST DAS REZEPT";D$
220 IF (LEN (D$) > 16) THEN D$=LEFT$(D$,16)
225 GOSUB 1000
230 IF EOF THEN PRINT "+++ REZEPT NICHT GEFUNDEN!":GOTO 30
240 OPEN2,8,2,D$ + ",S,R"
250 INPUT#2,Z$
260 EOF = (64 AND ST)
270 IF NOT EOF THEN PRINT Z$:GOTO 250
280 CLOSE2
290 GOTO 30
300 REM
310 REM +++ LISTE
320 OPEN2,8,2,"LISTE,S,R"
330 INPUT#2,N$
340 EOF = (64 AND ST)
350 IF NOT EOF THEN PRINT N$:GOTO 330
360 CLOSE2
370 GOTO 30
1000 OPEN2,8,2,"LISTE,S,R"
1010 INPUT#2,R$
1020 EOF = (64 AND ST)
```

```
1030 IF NOT EOF AND (R$ <> D$) THEN GOTO 1010
1040 CLOSE2
1050 RETURN
```

Das Prinzip

Zwei Eigenschaften charakterisieren eine sequentielle Datei:

- Es kann nur auf eine Komponente der Datei zur Zeit zugegriffen werden. Ein Zeiger, der innerhalb der Datei bewegt werden kann, zeigt immer auf diese eine Komponente. Die Position des Zeigers kann entweder nur auf den nächsten Datensatz oder auf den Dateianfang geändert werden.

- Die Zugriffsmodi 'Schreiben' und 'Lesen' werden strikt getrennt. Das heißt, es kann, während die Datei geöffnet ist, entweder nur gelesen oder nur geschrieben werden.

Das jedenfalls ist die strenge Definition einer sequentiellen Datei. Aber, wie das immer mit strengen (sprich: theoretischen) Vorgaben ist, sie reichen für den Alltagsbetrieb nicht aus. In diesem Fall ist es sehr mühsam, Daten an eine sequentielle Datei anzuhängen. Man müßte die gesamte Datei in den Speicher einlesen, die gewünschten Daten anhängen und das Ganze in eine "neue" Datei schreiben. Sehr viel einfacher wäre es, wenn man "Anhängen" als dritten Zugriffsmodus hätte. Und genau das geht in Commodore-BASIC: Es gibt im OPEN-Befehl außer W (Schreiben) und R (Lesen) noch A (Anhängen). Das erleichtert die Arbeit mit sequentiellen Dateien sehr.

Es gibt noch einen weiteren Modus, der in "Das große Floppy Buch zur 1541" entdeckt worden ist: Modus M. Mit

```
OPEN2,8,2,"<dateiname>,S,M"
```

kann eine nicht ordnungsgemäß geschlossene Datei gelesen werden, was mit "R" nicht möglich ist.

Obwohl die Arbeit mit sequentiellen Dateien nicht sehr handlich ist, hat sie doch ihre Daseinsberechtigung. Denn es gibt immer noch moderne Speichermedien, die nur sequentielle Dateien zulassen. Man denke nur an die Magnetbandgeräte. Auch Datenübertragung zwischen einzelnen Speichermedien läßt sich nur mit Hilfe von sequentiellen Dateien durchführen. Und die wichtigen Textdateien können ausschließlich als sequentielle Dateien organisiert werden.

Aus den vorher erwähnten Eigenschaften von sequentiellen Dateien ergeben sich noch Folgerungen, die leicht nachvollzogen werden können:

- Jeder Schreibvorgang kann nur am Dateiende erfolgen. (Anhängen von Daten)
- Jeder Lesevorgang kann nur innerhalb der Datei erfolgen, d.h. wenn die Bedingung EOF nocht nicht erfüllt ist.

Wie ich oben erwähnt habe, kann eine sequentielle Datei sowohl zeichenorientiert als auch mit Datensätzen aufgebaut werden. Wenn sie zeichenorientiert strukturiert worden ist, ist keine weitere Ordnung vorhanden. Sie heißt dann Textdatei. Hat man den Datensatz als Struktur gewählt, baut man in eine sequentielle Datei Unterstrukturen ein. Zwar kann die Datei noch immer nur sequentiell (d.h. strenggenommen zeichenweise) gelesen werden, aber durch die Vorgabe einer strengen Reihenfolge von Einzeldaten kann man in solch einer Datei durchaus sinnvoll Datensätze organisieren.

Die bisherigen Beispiele arbeiten alle ohne Datensätze, also zeichenweise. Jetzt wollen wir eine sequentielle Datei durch Datensätze strukturieren.

Datensätze in sequentiellen Dateien

Wir bauen ein Telefonverzeichnis. Der Datensatz, der in unserer Datei zur Geltung kommen soll, besteht aus zwei Datenangaben:

1. der Name (Vor- u. Nachname der Einfachheit halber als Einheit betrachtet)
2. die Telefonnummer

Die einzelnen Datenangaben, Datenfelder, bilden eine Einheit, den Datensatz. Wie das mit den Datensätzen funktioniert, kann man sich mit einer Graphik anschaulich klarmachen:

```
1. Datensatz: Name_1 Telefonnummer_1
2. Datensatz: Name_2 Telefonnummer_2
3. Datensatz: Name_3 Telefonnummer_3
. . .
. . .
. . .
n. Datensatz: Name_n Telefonnummer_n
Dateiende: <EOF>
```

Natürlich werden auch diese "Datensätze" zeichen- oder (bei Textdateien) zeilenweise gelesen, denn es ist ja gar nichts anderes möglich. Aber wenn man zwischen die einzelnen Datenangaben Trennzeichen einfügt (z.B. Zeilenvorschubzeichen), dann ist es möglich, beim Einlesen die einzelnen Datensätze zu unterscheiden. Die erste Zeichenkette ist der erste Name, die zweite Zeichenkette ist die erste Telefonnummer, die dritte Zeichenkette ist der zweite Name, die vierte Zeichenkette ist die zweite Telefonnummer, usw.

Wenn man die Trennzeichen von Null beginnend durchnumeriert und die Null als gerade Zahl betrachtet, so gilt folgender Satz:

Jeder Name steht hinter einem Trennzeichen mit gerader Nummer, jede Telefonnummer hinter einem Trennzeichen mit ungerader Nummer. Ein neuer Datensatz beginnt also hinter einem Trennzeichen mit gerader Nummer. Bevor ich das realisiere, muß ich noch einige Bemerkungen zur Technik machen.

Commodore Spezialitäten

Was passiert bei den besprochenen Laufwerken während der Benutzung von sequentiellen Dateien? Wir werden drei Abschnitte besprechen: Das Öffnen (Schreib- u. Lesemodus), das Bearbeiten (Schreiben und Lesen) und das Schließen der Datei.

Beim Öffnen (Schreibmodus) der sequentiellen Datei geschehen mehrere Dinge:

- Es wird geprüft, ob auf der Diskette bereits eine Datei gleichen Namens existiert. Wenn ja, wird eine Fehlermeldung ("FILE EXISTS") ausgegeben. Die Datei wird dann nicht eingerichtet. Wenn Sie vor dem Dateinamen den Klammeraffen "@" benutzen, wird die bestehende Datei gelöscht.

- In dem Directory wird der Eintrag vorgenommen, der den Filetyp festlegt (hier: sequentielle Datei). Dabei wird in dem Directory festgehalten, daß diese Datei noch nicht ordnungsgemäß geschlossen ist, da ja noch Daten in die Datei geschrieben werden sollen. Dieser Zustand wird in dem Directory mit einem Stern (engl.: Asterisk) "*" vor dem Filetyp gekennzeichnet.

- Nun wird ein freier Sektor (Block) gesucht, auf dem der erste Datenschub gespeichert werden kann. Die Adresse dieses Blocks (Spur- und Sektornummer) wird im Fileeintrag dieser Datei in dem Directory festgehalten.

- In diesem Stadium enthält die Datei formell noch 0 Blocks (da sie ja noch nicht geschlossen ist und noch nicht bekannt ist, wieviel Blöcke sie enthält).

Alle Dateiorganisationsformen, die unsere Laufwerke von Haus aus bearbeiten können, sind, bis auf Typ R, sequentielle Dateien. Die formale Unterscheidung dient nur Archivierungszwecken. Ein weiterer kleiner Unterschied ist, daß Programmdateien am Anfang noch zwei Bytes zusätzlich (LOW-Byte/HIGH-Byte) enthalten, die angeben, an welcher Spei-

cheradresse das BASIC-Programm starten soll. Sogenannte "USER-Dateien" und sequentielle Dateien unterscheiden sich überhaupt nicht. Eigentlich sind USER-Dateien für einen anderen Zweck gedacht. Mit deren Hilfe kann man nämlich eigene Programme in den Floppyspeicher laden und dort laufen lassen. Außerdem werden solche ausführbaren Dateien sofort nach dem Laden automatisch gestartet, weswegen sie Autostartdateien genannt werden.

Daß man eigene Programme statt im Computerspeicher im Floppyspeicher laufen lassen kann, eröffnet viele interessante Möglichkeiten. Da die Floppy einen eigenen, vom Computer unabhängigen Prozessor besitzt, kann man zum Beispiel sehr rechenintensive Programme (z.B. "Apfelmännchen") teilweise in die Floppy auslagern und durch geeignete Aufteilung eines Programmes auf zwei Prozessoren die Rechengeschwindigkeit enorm steigern, indem man echte Parallelverarbeitung einsetzt. Das setzt aber besondere Kenntnisse voraus, und es würde den Rahmen dieses Buches sprengen, hier ausführlich darauf einzugehen. Mehr darüber finden Sie in der Literatur[10]. Für unsere Zwecke können Sie sich merken, daß USER-Dateien genauso behandelt werden können wie sequentielle Dateien.

Nun will ich Ihnen noch in einer Tabelle die vier Zugriffsmodi aufführen, die für den OPEN-Befehl einsetzbar sind:

Typ	Erklärung
W	Schreiben in eine neue Datei. Eine bereits bestehende Datei desselben Typs wird dabei gelöscht (bei Verwendung des Klammeraffens "@").
R	Lesen aus einer bestehenden Datei.
A	Anhängen an eine bestehende sequentielle Datei. Wenn die Datei nicht existiert, wird Sie neu angelegt.
M	Einzige Möglichkeit, Daten aus einer nicht ordnungsgemäß geschlossenen Datei noch zu retten (zu lesen).

10 siehe z.B.: Ellinger: Commodore 1571 & 1570, Das große Floppybuch, Düsseldorf 1986, Data Becker, 2.Auflage

Eine kleine Besonderheit besitzen die logischen Filenummern in diesem Zusammenhang. Werden Filenummern von 1 bis 127 benutzt, wird nach einem PRINT#-Befehl nur das Zeichen 13 (Carriage Return) an die Datei gesendet. Bei Filenummern von 128 bis 255 wird dagegen die Zeichenfolge <CR><LF> gesendet (Zeichen 13, Carriage Return und Zeichen 10, Line Feed). Die zweite Methode braucht man zum Beispiel bei Druckern, die nicht in der Lage sind, ein sog. Autolinefeed zu erzeugen. Allerdings gibt es solche Drucker fast nicht mehr zu kaufen. Nur noch bei den billigsten kann solch ein Manko auftreten.

Einen weiteren Punkt sollte man bei den Sekundäradressen beachten. Denn es sind zwar freie Sekundäradressen zwischen 2 und 14 zugelassen, aber es kann nur eine ganz bestimmte Anzahl von Kanälen, die ja durch die Sekundäradresse repräsentiert werden, gleichzeitig mit Daten beschickt werden. Dabei ist zu beachten, daß zur Verwaltung von sequentiellen Dateien ein Kanal, zur Verwaltung von relativen Dateien jedoch zwei Kanäle benötigt werden. Bei maximal drei gleichzeitig geöffneten Kanälen ergeben sich also folgende Konfigurationen:

- 1 relative und 1 sequentielle Datei
- 3 sequentielle Dateien

Unter dem Thema "Pufferspeicher" wird diese Beschränkung später noch genauer besprochen werden.

Das Telefonverzeichnis - Erster Versuch

Das folgende Beispiel ist noch nicht selbständig lauffähig, da es die Existenz der Datei "TELEFON" schon voraussetzt. Außerdem kann das Programm nur Datensätze an die bereits bestehende Datei anhängen oder Daten suchen. Aber an diesem Rudiment eines Programms kann man schon sehr viel sehen. Im übrigen werden wir dieses Programm schon noch lauffähig machen!

```
10 REM ***** TELEFONVERZEICHNIS V0.1
20 REM
30 PRINT "(E)INGABE, (A)USGABE, (Q)UIT"
```

```
40 INPUT E$
50 IF E$="E" THEN GOTO 100
60 IF E$="A" THEN GOTO 200
70 IF E$="Q" THEN GOTO 300
80 PRINT "+++ FALSCHE EINGABE!":GOTO 30
90 REM
100 REM +++ EINGABE
110 OPEN2,8,2,"TELEFON,S,A"
120 INPUT "NAME";NA$
130 INPUT "TEL.";TN$
140 PRINT#2,NA$
150 PRINT#2,TN$
160 CLOSE2
170 GOTO 30
200 REM +++ AUSGABE
210 INPUT "WESSEN TELEFONNUMMER SUCHEN";SN$
220 OPEN2,8,2,"TELEFON,S,R"
230 INPUT#2,NA$:INPUT#2,TN$
240 EOF = (64 AND ST)
250 IF NOT EOF AND (NA$ <> SN$) THEN GOTO 230
260 IF EOF THEN PRINT "+++ NICHT GEFUNDEN!":GOTO 280
270 PRINT "TELEFONNR. VON ";NA$;" IST: ";TN$
280 CLOSE2
290 GOTO 30
300 END
```

Bei diesem Programm ist der wichtige Teil die Zeile 230. Denn man sollte bei jedem Lesevorgang den gesamten Datensatz in den Speicher lesen, da man sonst schnell durcheinanderkommt. Und da der gesamte Datensatz aus dem Namen und der Telefonnummer besteht, müssen beide Daten eingelesen werden. Das Trennzeichen ist hier der "Zeilenvorschub", der durch die beiden PRINT#-Anweisungen in den Zeilen 140 und 150 erzeugt wird. (Erinnern Sie sich, PRINT# macht dasselbe auf dem Ausgabegerät wie PRINT auf dem Bildschirm!) Versuchen Sie jetzt, das Programm so zu erweitern, daß es voll einsatzfähig wird und auch entsprechende Fehlerroutinen enthält. Ich möchte das Programm in diesem Kapitel nicht weiterverfolgen, das soll im folgenden Kapitel "Relative Dateien" geschehen.

8. Relative Dateien

"Es kann sein, daß es sich beim Licht am Ende des Tunnels um die Scheinwerfer einer Lokomotive handelt."

Murphologische Weisheit

Wir erfahren Interessantes über die Eigenschaften relativer Dateien, wie man den Dateizeiger positioniert, wie ein Datensatz aussieht, wie er zusammengebaut und wieder zerlegt wird, was die "Sprudelmethode" ist und wie man damit auch in einer Datei sortieren kann.

Es wird zwar immer wieder behauptet, die einfachere Form, die sequentielle Datei, wäre auch leichter zu handhaben. Daß dem nicht so ist, merkt man spätestens dann, wenn man gelernt hat, mit relativen Dateien umzugehen. Das einzige, worauf man zu achten hat, ist, daß die Datensätze alle dieselbe Länge haben. An dieser Stelle möchte ich noch einmal betonen, daß die Bezeichnung "Relative Datei" von Commodore gewählt worden ist. Sie bezeichnet einen Dateityp, der im normalen Sprachgebrauch als Datei mit wahlfreiem Zugriff (engl.: Random Access File) bezeichnet wird. Im weiteren werde ich mich aber an die Nomenklatur von Commodore halten.

Relative Dateien haben, wie sequentielle Dateien, bestimmte Eigenschaften, die sie von anderen Dateitypen unterscheiden.

Die Eigenschaften der relativen Dateien sind:

- Alle Datensätze müssen dieselbe Länge besitzen. Dadurch kann die Position eines bestimmten Datensatzes sehr einfach berechnet werden. Ein Zeiger bewegt sich frei in der Datei hin und her.

- Während eines Zugriffs kann sowohl gelesen als auch geschrieben werden. Es wird also kein Unterschied im Öffnungsmodus gemacht wie bei sequentiellen Dateien.

Aus diesen Eigenschaften ergeben sich eindeutige Unterschiede, die die relativen Dateien gegenüber den sequentiellen abgrenzen. Es ist jetzt jeder Datensatz ohne Umwege sofort anwählbar ("wahlfreier Zugriff"). Außerdem können alle Arbeiten direkt auf der Diskette ausgeführt werden, so daß der Speicher des Rechners weitestgehend entlastet wird. Es ist ebenfalls leicht, einzelne Datensätze zu löschen.

Und wieder Commodore's Spezialitäten

Die 1541 und die 1570/71 können Datensätze in höchstens 720 Sektoren pro relativer Datei verwalten. Das liegt an den sogenannten Side Sectors. Ich gehe im Anhang unter dem Titel "Nähkästchen" genauer darauf ein.

Um eine relative Datei zu öffnen, braucht man...na, was? Klar, den OPEN-Befehl.

BASIC bis 2.0:

```
OPEN<lfn>,<ga>,<sad>,"<dateiname>,<typ>,"+CHR$(<dsl>)
```

Die ersten vier Parameter kennen wir und wissen damit umzugehen. Der Inhalt des sechsten Parameters hat sich geändert. Der sechste Parameter ist neu:

<typ> Dateityp S = Sequentiell (kennen wir schon und steht nur der Vollständigkeit halber da) L = Relativ (Das L steht für length. Hier muß die feste Länge des Datensatzes angegeben werden, der in dieser relativen Datei benutzt wird.)

<dsl> Datensatzlänge

Für ein Beispiel wählen wir als Dateinamen "TEST", die Datensatzlänge soll 10 betragen:

```
OPEN2,8,2,"TEST,L,"+CHR$(11)
```

Hoppla! Warum steht denn da 11 im Beispiel statt 10??? Es ist kein Druckfehler, und ich erkläre es auch gleich.

*****

BASIC ab 3.0

Auch hier treten wieder die Kosmetikbefehle auf, die so aussehen:

```
DOPEN#<lfn>,"<dateiname>",U<sad>,L<dsl>
```

wobei <dsl> für Datensatzlänge steht. Ein Beispiel mit Datensatzlänge 10:

```
DOPEN#1,"TEST",L11
```

Hoppla! Warum steht denn auch hier eine 11 im Beispiel statt einer 10??? Die Antwort folgt sofort.

*****

Warum steht nun eine 11 an einer Stelle, wo man nach den ganzen Definitionen und Erklärungen eine 10 erwarten würde? Das liegt an der Art, wie man die Eingabe in die relative Datei vornimmt. Daten werden mittels PRINT# an die Datei gesendet. Da PRINT# genau wie PRINT nach den eigentlichen Daten ein Carriage Return (ASCII-Zeichen 13) sendet, bekommt das Datenpaket, das mit PRINT# gesendet wird, am Ende noch dieses Zeichen mit. Auf dem Bildschirm (PRINT) macht sich das durch einen Wagenrücklauf mit Zeilenvorschub bemerkbar. In der Datei steht an dieser Stelle das Trennzeichen <CR> (ASCII-Zeichen 13). Um diesen "Zeilenvorschub" zu verhindern, müssen Sie hinter dem PRINT#-Befehl ein Semikolon (";") anfügen.

Will man solchermaßen abgespeicherte Daten wieder einlesen, muß die Datensatzlänge um eins größer sein als geplant, da das Zeichen <CR> im Datensatz noch Platz haben soll. Wenn Sie so

einen Datensatz mit INPUT# wieder einlesen, ist dieses <CR> Bestandteil des Datensatzes! Also muß beim Öffnen einer relativen Datei für ein Zeichen mehr Platz im Datensatz sein, wenn mit INPUT# eingelesen wird.

Da aber INPUT# (wie auch INPUT) nur eingesetzt werden kann, wenn der zugehörige String nicht länger als 88 Zeichen ist (160 beim C128), gilt das nur für Datensätze von bis zu 87 Zeichen Länge.

Ansprechen von Datensätzen

Da jeder Datensatz sofort angewählt werden kann, muß es auch einen Befehl geben, der das bewerkstelligt. Dieser Positionierbefehl richtet einen Zeiger auf den gewünschten Datensatz aus, so daß eine folgende Operation sich auf ihn bezieht. Tatsächlich liegt es weitgehend beim Programmierer, sich um die Positionierung zu kümmern. Man hat also dafür zu sorgen, daß vor jeder Operation der Dateizeiger auf den richtigen Datensatz gerichtet ist.

Beim Öffnen ist noch eine kleine Besonderheit zu beachten: Bei den Commodore-Laufwerken müssen Datensätze erst freigegeben werden. Freigeben heißt, daß die Datensätze mit dem ASCII-Zeichen 255 beschrieben werden. Der Witz an der Sache ist, daß Sie durch das Freigeben eines Datensatzes mit einer Nummer größer als 1 alle Datensätze bis zu dieser Nummer freigeben. Angenommen, Sie geben Datensatz 100 frei, dann haben Sie automatisch auch alle Datensätze davor (Nummern 1 bis 99) freigegeben. Bevor eine relative Datei benutzt wird, sollten mindestens soviele Datensätze freigegeben werden, wie man voraussichtlich benutzen wird, da das spätere Freigeben einiges an Zeit kostet. Bei der 1541 dauert das Freigeben von 1000 Datensätzen ca. 10 Minuten.

Aber zunächst der Befehl zum Positionieren:

BASIC bis 2.0:

Natürlich benutzen wir einen OPEN-Befehl. Da er einen Befehl enthält (Befehl "P", positionieren), müssen wir den Fehler- und Befehlskanal (Sekundäradresse 15) benutzen:

```
OPEN<lfn>,<ga>,15
...
PRINT#<lfn>,"P"+CHR$(<lfn>)+CHR$(low)+CHR$(high)+CHR$(pos)
```

Dabei bedeuten:

<lfn> logische Filenummer des Befehlskanals

<low> Lowbyte der Datensatznummer

<high> Highbyte der Datensatznummer

<pos> Position des gewünschten Bytes im Datensatz

Lowbyte und Highbyte, was soll das? Ganz einfach: Nummern für einen Datensatz können Zahlen bis 65535 sein. Mit 8 Bit, also 1 Byte, kann man genau 256 Zahlen darstellen. Das sind die Zahlen von 0 bis 255. Um mehr Zahlen darstellen zu können, nimmt man ein zweites Byte hinzu und hat somit nicht 8 Bits, sondern 16 Bits. Und mit 16 Bits (2 Bytes) kann man genau 65536 Zahlen, also 0 bis 65535, darstellen. Der Computer kann aber nicht 16 Bits auf einmal ansteuern. Also teilt man das in zwei Schritte auf. Zuerst kommt das Byte dran, das auf der niederwertigen Stelle (rechts) steht, das sog. Lowbyte, dann kommt das höherwertige Byte (links), das Highbyte. Um eine Datensatznummer für den Computer zuzubereiten, muß man folgende Berechnungen anstellen (als BASIC-Programmfragment formuliert):

```
10 REM BERECHNUNG VON LOW- U. HIGHBYTE AUS DATENSATZNUMMER
20 REM
30 INPUT "DATENSATZNUMMER";DN
40 HB = INT (DN / 256)
50 LB = DN - HB * 256
60 PRINT "LOWBYTE = ";LB;" HIGHBYTE = ";HB
```

Mit diesem kleinen Umrechnungsprogramm können Sie sofort die gewünschten Daten errechnen. Für Interessierte befindet sich im Anhang unter dem Titel "Zahlensysteme" unter Anderem eine kleine Einführung in die Zahlendarstellung des Dualsystems, aus dem die Problematik des Low-/Highbytes entsteht.

Ein Beispiel für den Positionierbefehl:

```
OPEN2,8,15:REM BEFEHLSKANAL
PRINT#2,"P"+CHR$(2)+CHR$(232)+CHR$(3)+CHR$(1)
```

Damit wird der Dateizeiger über den Befehlskanal auf das erste Byte im Datensatz mit der Nummer 1000 gesetzt. (Lowbyte 323 + Highbyte 3 = 16-Bit Zahl 1000)

*****

BASIC ab 3.0:

Hier ist wie immer alles viel einfacher. Da der Begriff "Datensatz" im Englischen Record heißt, wurde auch der Positionierbefehl so genannt.

```
RECORD#<lfn>,<dsn>,<pos>
```

wobei

<lfn> die logische Filenummer,

<dsn> die Datensatznummer,

<pos> die Byteposition im Datensatz bedeuten.

Ein Beispiel:

```
RECORD#2,1000,1
```

Über Filenummer 2 (Befehlskanal) wird auf Datensatz 1000, Byteposition 1 positioniert.

*****

Die Angabe über die Position kann in beiden Befehlen weggelassen werden.

Natürlich muß auch eine relative Datei wieder geschlossen werden. Egal, was in anderen Büchern steht, tun Sie Ihren Daten etwas Gutes, schließen Sie relative Dateien mit CLOSE (bis 2.0) bzw. DCLOSE (ab 3.0). Diese Befehle haben wir ja schon kennengelernt.

Um relative Dateien kennenzulernen, werden wir gleich ein größeres Projekt angehen und uns Schritt für Schritt vorantasten. Das Beispiel "Telefonverzeichnis", das wir ja schon im Kapitel "Sequentielle Dateien" angefangen haben, wird nun verändert und vervollständigt. Die Daten werden wir nicht in einer sequentiellen Datei, sondern (klar!) in einer relativen Datei abspeichern. Dadurch sind wir in der Lage, schnell und flexibel auf die Daten zuzugreifen und sie zu manipulieren.

Zunächst aber, nach gutem Programmierbrauch, eine kleine Liste der Fähigkeiten, die das Programm haben soll. Es gibt Leute, die dazu Pflichtenheft sagen, aber mir klingt dieser Begriff zu sehr nach geschriebener Order. Solch ein "Pflichtenheft" soll ja mehr eine Wunschliste sein, in der alle notwendigen und wünschenswerten Eigenschaften stehen, die das spätere Programm besitzen soll. Übrigens sollte so eine Liste zusammen von Programmierer und Anwender erstellt werden.

Was soll unser Programm können?

Es soll mindestens folgende Aufgaben bewältigen können:

- Anlegen der Datei
- Eingabe von Daten
- Löschen von Einträgen
- Korrigieren von Einträgen
- Ausdrucken einer Telefonliste

- Suche nach Nummern bei Eingabe des Namens
- Alphabetisches Sortieren der Namenseinträge

Zwar ist diese Liste ganz schön lang, aber wir werden Schritt für Schritt einen Punkt nach dem anderen abhaken. Der Übersicht wegen werde ich das Programm mit den Befehlen der Version ab 3.0 ausstatten; an den geeigneten Stellen werde ich aber immer auf die 2.0-Version hinweisen. Allerdings müssen wir uns vor der eigentlichen Arbeit noch ein wenig damit beschäftigen, wie ein Datensatz in BASIC überhaupt aussieht.

Vorbereiten eines Datensatzes

Zunächst muß man sich darüber im klaren sein, was überhaupt in den Datensatz an Information hineingesteckt werden soll. In unserem Fall sind es nur zwei Datenfelder, nämlich der Name und die Telefonnummer. Stellen wir also die Rechnung auf:

Datensatz:	
Name	33 Zeichen
Telefonnummer	13 Zeichen
Summe:	46 Zeichen

Dabei gilt: 1 Zeichen = 1 Byte. Also ist die Länge des Datensatzes 46 Byte. Nun müssen wir noch festlegen, wie lang die Datei anfangs werden soll. Verlängern kann man sie später immer noch, aber um die Geschwindigkeit des Zugriffs auf eine erträgliche Größenordnung herunterzuschrauben, einigen wir uns darauf, schon mal 100 Datensätze freizugeben.

Ein Datensatz ist in einem String untergebracht. Aber dabei kommt es auf die Position der einzelnen Datenfelder an! Bevor ich das erkläre, kurz eine kleine Graphik:

Wir wollen als Name "DAMPF HANS" und als Telefonnummer "0815/4711" annehmen. Der String DT$, der den Datensatz darstellen soll, muß dann so gefüllt werden:

```
"DAMPF.HANS.....0815/4711..."
```

Bereich für den Namen Bereich für die Telefonnummer

Nochmal im Klartext: Die einzelnen Datenfelder müssen in einem einzigen String "verpackt" werden, wobei man die Datenfelder solange mit Leerzeichen (ASCII-Zeichen 32, Blank) auffüllt, bis die für dieses Feld angegebene Feldlänge erreicht ist. In unserem Beispiel ist der Name mit dem trennenden Leerzeichen genau 10 Zeichen (Bytes) lang. Für den Namen sind aber maximal 33 Zeichen vorgesehen. Also müssen hinter dem eigentlichen Namen noch 33 minus 10 gleich 23 Leerzeichen angefügt werden. Auf gleiche Weise wird mit der Telefonnummer verfahren. Die Telefonnummer ist 9 Zeichen (Bytes) lang. Vorgesehen sind aber 13 Bytes. Also 13-9=4, es müssen noch vier Leerzeichen angehängt werden.

Am besten definiert man sich am Anfang eines Programms eine Stringvariable, die genau soviele Blanks (Leerzeichen) enthält wie es der Datensatzlänge entspricht. Die einfachste Vorgehensweise ist, die Datensatzlänge in einer Variablen vorzugeben, alles andere ergibt sich. Ein Beispiel:

```
10 REM BEISPIEL ZUR VORBEREITUNG
20 B$ = "":DL = 46
30 FOR X = 1 TO DL
40 B$ = B$ + " "
50 NEXT X
60 REM Rest des Programms
```

Nun wollen wir noch den Namen und die Nummer in der Datensatzvariablen unterbringen. Der folgende Programmausschnitt zeigt, wie das geht:

```
60 REM ZUBEREITUNG DES DATENSATZES
70 N$ = "DAMPF HANS"
80 T$ = "0815/7411"
90 DT$ = N$ + LEFT$ (B$,33 - LEN (N$))
100 DT$ = DT$ + T$ + LEFT$ (B$,13 - LEN (T$))
110 REM .....
```

Danach ist der Datensatz für das Beispiel fertig und kann in die Datei gesendet werden. Aber zurück zu unserem Programm. Zur Positionierung verwenden wir hier die BASIC 2.0 Version. Als leichte Fingerübung am Anfang wollen wir zunächst das Menü schreiben:

```
10 REM TELEFONNUMMERNVERWALTUNG V1.0
20 REM A$  = ANTWORTZEICHEN FUER MENUE
30 REM DT$ = DATENSATZSTRING
40 REM DL  = DATENSATZLAENGE
50 REM DN  = DATENSATZNUMMER
60 REM B$  = BLANKSTRING
70 REM N$  = NAME
80 REM T$  = TELEFONNUMMER
85 REM TT$,NN$ := KOPIEN VON N$,T$
90 DL = 46:B$ = "":OPEN15,8,15:REM OEFFNEN BEFEHLSKANAL
95 FOR X=1 TO DL:B$ = B$ + " ":NEXT X
100 PRINT "TELEFONLISTE V1.0"
110 PRINT "(1) NEU ANLEGEN"
120 PRINT "(2) EINGEBEN"
130 PRINT "(3) LOESCHEN"
140 PRINT "(4) KORRIGIEREN"
150 PRINT "(5) DRUCKEN"
160 PRINT "(6) SUCHEN"
170 PRINT "(7) SORTIEREN"
180 PRINT "(8) ENDE"
190 GET A$:IF A$="" THEN GOTO 180
200 IF (A$<"1") OR (A$>8) THEN PRINT "+++ EINGABEFEHLER":GOTO 100
210 ON VAL(A$) GOTO 300,400,500,600,700,800,900,1000
```

Das Menü wird noch schöner, wenn man vor die PRINT-Anweisung in Zeile 100 einen Befehl zum Löschen des Bildschirms einfügt. Dann steht das Menü immer auf einem "sauberen" Bildschirm.

Um das Programm zwar leistungsfähig, aber trotzdem noch recht einfach zu halten, soll die Liste in einer festen Datei namens "TELEFON" abgespeichert werden. Da der Computer nicht wissen kann, ob die Datei auf Diskette vorhanden ist, muß man das irgendwie überprüfen. Doch keine Sorge! Der OPEN-Befehl nimmt uns diese Arbeit ab, wenn wir eine Kleinigkeit beachten:

Wenn mit OPEN eine relative Datei geöffnet wird, passiert folgendes:

- Existiert die Datei noch nicht, so wird sie angelegt und dann geöffnet.

- Existiert die Datei, wird sie geöffnet.

Im ersten Fall ist zu beachten, daß dem OPEN-Befehl die Datensatzlänge mitgegeben wird. Im zweiten Fall kann sie weggelassen werden. Läßt man sie im zweiten Fall nicht weg, dann ist es für den fehlerfreien Ablauf des Programms zwingend notwendig, daß dieselbe Datensatzlänge angegeben wird wie beim ersten Anlegen der Datei. Merke: Ist die Datensatzlänge einmal festgelegt, halte immer daran fest!

An dem Punkt "Anlegen einer Datei" kommen wir trotzdem nicht vorbei (Freigeben von Datensätzen). Machen wir uns an Punkt eins, Anlegen der Datei:

```
300 REM +++ ANLEGEN EINER NEUEN DATEI
305 PRINT "+++ DATEI 'TELEFON' WIRD ANGELEGT"
310 REM OEFFNEN MIT DATENSATZLAENGE 46
315 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
320 PRINT "+++ FREIGABE VON 100 DATENSAETZEN"
325 REM 100 DATENSAETZE FREIGEBEN
330 PRINT#15;"P"+CHR$(2)+CHR$(100)+CHR$(0)+CHR$(1)
335 PRINT#1,CHR$(255);
340 PRINT "+++ DATEI 'TELEFON' ANGELEGT"
335 CLOSE1
340 GOTO 100
```

Mit diesem Modul wird die Datei auf Wunsch angelegt und der Dateizeiger auf den 101. Datensatz positioniert. Existierte die Datei noch nicht, werden 100 Datensätze freigegeben. Diese Lösung ist zwar nicht die eleganteste, aber für den Anfang reicht sie. Falls sie Ihnen nicht zusagt, steht es Ihnen frei, sie nach Belieben zu verbessern. Das ist zugleich auch eine gute Übung. Der nächste Punkt ist die Eingabe. Wir gehen gleich an die Arbeit:

```
400 REM +++ EINGABE
405 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
410 DT$ = "":REM VORBEREITEN DES NEUEN DATENSATZES
415 INPUT "DATENSATZNUMMER";DN
420 INPUT "NAME";N$
```

```
425 INPUT "TELEFON";T$
430 DT$ = N$ + LEFT$ (B$,33 - LEN (N$))
435 DT$ = DT$ + T$ + LEFT$ (B$,13 - LEN (T$))
440 REM POSITIONIERUNG AUF DATENSATZ RN, BYTE 1
445 PRINT#15,"P"+CHR$(2)+CHR$(DN)+CHR$(0)+CHR$(1)
450 REM SCHREIBEN DES DATENSATZES
455 PRINT#1,DT$;
460 INPUT "NOCHMAL (J/N) ";AA$
465 IF (AA$ <> "N") THEN GOTO 410
470 CLOSE1:GOTO 100
```

Tja, sehr komfortabel ist das Ganze nicht! Das liegt daran, daß immer die Datensatznummer mit angegeben werden muß. Aber es gibt eine Lösung, und Sie sollen später, wenn wir das Programm vollständig besprochen haben, versuchen, diese zu realisieren.

Ihnen ist ja bekannt, wie viele Datensätze Sie freigegeben haben (im Beispiel sind es 100). Nun gilt es, zwei Dinge zu überwachen:

- Erstens haben Sie darüber Buch zu führen, welche Datensätze belegt sind.

- Zweitens ist zu prüfen, ob man mehr Datensätze freigeben muß.

Punkt 1 könnte man mit Hilfe einer sequentiellen Datei lösen, in der einfach die Datensatznummern der belegten Datensätze gespeichert werden (Siehe auch das Kapitel über "ISAM-Dateien"). Über Punkt 2 dürfen Sie selbständig nachdenken.

Wir werden mit unserem einfachen Beispiel fortfahren. Beim Betrieb ist aber darauf zu achten, daß Sie immer "von Hand" über belegte Datensatznummern Buch führen. Das Programm ist noch nicht fehlertolerant, reagiert also nicht korrekt auf Fehler in der Eingabe und kann deshalb bei fehlerhaften Eingaben einer Fehlermeldung stoppen. Arbeiten Sie auch in dieser Richtung Lösungsmöglichkeiten aus. Der Menüpunkt "Löschen" ist auch nicht so ohne weiteres zu lösen. Es gibt nämlich keinen expliziten Befehl zum Löschen eines Datensatzes. Nichtsdestoweni-

ger ist diese Funktion wichtig, und so möchte ich folgende Lösung vorschlagen: Der Datensatz, der gelöscht werden soll, erhält als Inhalt lauter Blanks, also den Inhalt von B$.

```
500 REM +++ LOESCHEN
505 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
510 PRINT "WELCHER DATENSATZ WIRD GELOESCHT?
515 PRINT "(EINGABE VON 0 KEHRT INS MENUE ZURUECK)"
520 INPUT DN
525 IF (DN = 0) THEN GOTO 100
530 PRINT#15,"P"+CHR$(2)+CHR$(DN)+CHR$(0)+CHR$(1)
535 DT$ = B$
540 PRINT#1,DT$;
545 PRINT "+++ DATENSATZ NR.";DN;" GELOESCHT!"
550 CLOSE1
555 GOTO 100
```

Interessant wird es jetzt! Denn nun wollen wir das Modul "Korrigieren" ausarbeiten. Dabei muß man den Datensatz, nachdem man ihn eingelesen hat, wieder in seine Bestandteile zerlegen. Und das ist das Interessante. Das geht natürlich genau umgekehrt, wie beim Zusammensetzen. Bevor Sie weiterlesen, versuchen Sie sich die Lösung selber zu überlegen. Danach können Sie kontrollieren, ob Ihre Lösung stimmt. Hier ist meine Lösung:

```
600 REM +++ KORRIGIEREN
605 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
610 INPUT "WELCHEN DATENSATZ KONTROLLIEREN";DN
615 PRINT#15,"P"+CHR$(2)+CHR$(DN)+CHR$(0)+CHR$(1)
620 INPUT#1,DT$
625 IF (DT$ = B$) THEN PRINT "+++ DATENSATZ GELOESCHT!":
GOTO 100
630 N$ = LEFT$ (DT$,33)
635 T$ = RIGHT$ (DT$,13)
640 PRINT "INHALT DES DATENSATZES NR.";DN;":"
645 PRINT "(RETURN ALLEIN = KEINE AENDERUNG)"
650 PRINT N$;:INPUT NN$:IF (NN$ <> "") THEN N$ = NN$
655 PRINT T$;:INPUT TT$:IF (TT$ <> "") THEN T$ = TT$
660 DT$ = "":N$ = LEFT$ (N$,33):T$ = LEFT$ (T$,13)
665 IF (LEN (N$) < 33) THEN N$ = N$ + LEFT$ (B$,33 - LEN (N$))
670 IF (LEN(T$) < 13) THEN T$ = T$ + LEFT$ (B$,13-LEN (T$))
675 DT$ = N$ + T$
680 PRINT#15,"P"+CHR$(2)+CHR$(DN)+CHR$(0)+CHR$(1)
685 PRINT#1,DT$;
690 CLOSE1
695 GOTO 100
```

Die Art, wie der Datensatz in den Zeilen 625 und 630 zerlegt wird, ist speziell auf unser Beispiel mit nur zwei Datenfeldern zugeschnitten. Bei Datensätzen mit mehr als zwei Datenfeldern muß man mit MID$ arbeiten.

Die Zeilen 660 und 665 sind deshalb mit der IF-THEN-Abfrage ausgestattet, weil die Strings beim Zerlegen in die maximale Datenfeldlänge aufgeteilt werden. Wenn aber der Inhalt in diesem Modul geändert wird, hat der String nicht mehr unbedingt die maximale Länge. Also muß er in diesem Fall wieder mit Blanks aufgefüllt werden. Die Erklärung der anderen Kleinigkeiten in diesem Modul dürfen Sie sich als Aufgabe wieder selber suchen.

Weiter geht es mit dem Modul "Drucken". Dabei möchte ich nicht auf irgendwelche Besonderheiten von Commodore-Druckern eingehen, bei denen man mit verschiedenen Sekundäradressen verschiedene Dinge einstellen kann. Der Drucker sei auf Geräteadresse 4 eingestellt.

```
700 REM +++ DRUCKEN
705 OPEN1,8,2,"TELEFON,L,"+CHR$(DL):OPEN4,4
710 Z = 1
715 PRINT#15,"P"+CHR$(2)+CHR$(Z)+CHR$(0)+CHR$(1)
720 INPUT#1,DT$
725 EOF = (64 AND ST)
730 IF EOF THEN GOTO 755
735 IF (DT$ = B$) THEN GOTO 745
740 PRINT#4,DT$
745 Z = Z + 1
750 GOTO 715
755 CLOSE1:CLOSE4
760 GOTO 100
```

Der Datensatz wird beim Drucken nicht zerlegt, da er ja sehr schön in einem String untergebracht ist, und da die meisten Drucker mindestens 80 Zeichen pro Zeile schaffen, so ist unser Datensatz mit seinen 46 Zeichen Länge locker in einer Druckzeile unterzubringen. Auch diese Druckroutine ist sehr einfach und nicht sehr komfortabel gehalten. Sie können ja versuchen,

sie so abzuändern, daß das Programm nach jeder Druckseite einen Blattvorschub vornimmt, vielleicht kann man ja jede Seite numerieren und mit einem Titel versehen.

Wir sind beim vorletzten Punkt angelangt, "Suchen". Daß das schnelle Suchen in einer Datei fast eine Wissenschaft für sich ist, weiß jeder, der sich einmal mit dem Fach Datenbanken beschäftigt hat. Es gibt keinen generellen Suchalgorithmus, und so hat man viele mehr oder weniger gute für alle möglichen Problemstellungen entwickelt. Allerdings werden solche Verfahren erst interessant, wenn die Menge der Daten sehr groß wird, oder wenn die Suche zeitkritisch ist. Unsere Telefondatei ist aber nicht so groß, als daß sich ein aufwendiges Verfahren lohnte. So werden wir die einfache sequentielle Suchmethode verwenden. Die Datensätze werden einfach der Reihe nach durchsucht. Da Namen nicht eindeutig sind, wird einfach jeder Datensatz ausgegeben, der den gleichen Namen enthält.

```
800 REM +++ SUCHEN
805 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
810 INPUT "WELCHER NAME WIRD GESUCHT";NA$
815 Z = 1
820 PRINT#15,"P"+CHR$(2)+CHR$(Z)+CHR$(0)+CHR$(1)
825 INPUT#1,DT$
830 EOF = (64 AND ST)
835 IF EOF THEN GOTO 855
840 IF (NA$ = LEFT$ (DT$,LEN (NA$)) THEN PRINT DT$
845 Z = Z + 1
850 GOTO 820
855 CLOSE1
860 GOTO 100
```

Die Formulierung von Zeile 840 erschlägt gleich zwei Fliegen mit einer Klappe:

- Man braucht den Datensatz DT$ nicht in seine Datenfelder zu zerlegen. Da der Name am Anfang des Datensatzes steht, kann man mit LEFT$ darauf zugreifen.

- Da man LEFT$ verwendet und jedes Vorkommen ausdrucken läßt, kann man auch Namensanfänge suchen lassen.

Das Ausdrucken und das Suchen haben nur Sinn, wenn die Liste sortiert ist. Beim Ausdrucken, damit man eine schöne sortierte Liste bekommt, beim Suchen, damit bei komplizierteren Suchverfahren richtig zugegriffen werden kann.

Wie sortiert man aber? Ganz einfach: Man nehme Bubblesort.

Sortieren nach der "Sprudelmethode"

Auch Sortierverfahren gibt es wie Sand am Meer. Wir werden die langsamste, aber einfachste Methode verwenden. Das Verfahren heißt Bubblesort. Bei dieser Methode wird eine Liste von Daten der Reihe nach durchgearbeitet und immer benachbarte Daten auf ihre richtige Reihenfolge geprüft. Stehen sie nicht in der richtigen Reihenfolge, werden die beiden getauscht. Daß nach einem Durchgang die Liste sortiert ist, kann man sich, glaube ich, sehr schnell klarmachen. Also geht man die Liste solange von vorne bis hinten durch, bis nach einem Durchgang nicht mehr getauscht werden mußte. Die Tatsache, ob getauscht wurde oder nicht, hält man mit einem Flag fest. (Siehe /SCH1/.)

Für dieses Modul nehmen wir an, daß genau 100 Datensätze freigegeben worden sind.

```
900 REM +++ SORTIEREN
905 REM H$ := HILFSVARIABLE FUER TAUSCHVORGANG
910 REM SW := 'SWAPPED', GETAUSCHT-FLAG
915 OPEN1,8,2,"TELEFON,L,"+CHR$(DL):PRINT "+++ SORTIEREN";
920 SW = 0:REM FLAG INITIALISIEREN
925 FOR X = 1 TO 99
930 PRINT#15,"P"+CHR$(2)+CHR$(X)+CHR$(0)+CHR$(1)
935 INPUT#1,D1$
940 PRINT#15,"P"+CHR$(2)+CHR$(X + 1)+CHR$(0)+CHR$(1)
945 INPUT#1,D2$
950 IF (D1$ < D2$) THEN GOTO 980
955 PRINT#15,"P"+CHR$(2)+CHR$(X)+CHR$(0)+CHR$(1)
960 PRINT#1,D2$;
965 PRINT#15,"P"+CHR$(2)+CHR$(X + 1)+CHR$(0)+CHR$(1)
970 PRINT#1,D1$;
975 SW = -1
980 NEXT X
```

```
985 IF SW THEN GOTO 920
990 PRINT " BEENDET"
995 GOTO 100
```

Dabei werden auch nicht belegte und gelöschte Datensätze so einsortiert, daß sie ein Päckchen bilden. Sie sind so leicht zu manipulieren. Auch diese Lösung ist nicht die eleganteste, aber ein bißchen Arbeit sollten Sie auch noch haben.

Um das Programm zu beenden, ist nicht viel notwendig. Der Befehlskanal, den wir die ganze Zeit geöffnet hielten, wird geschlossen, nachdem die Absicht, das Programm zu beenden, bestätigt worden ist.

```
1000 REM +++ BEENDEN
1010 INPUT "BEENDEN? (J/N) ";AA$
1020 IF (AA$ <> "J") THEN GOTO 100
1030 CLOSE2:REM SCHLIESSEN DES BEFEHLSKANALS
1040 PRINT "THAT'S ALL, FOLKS!"
1050 END
```

Damit ist das Programm im großen und ganzen fertig. Es ist zwar schon lauffähig, aber es sollte eher der theoretischen Demonstration der Dateiverwaltung relativer Dateien dienen. Um es im Alltag einzusetzen, sind noch einige kleine "Schönheitsoperationen" notwendig. Um Erfahrung zu sammeln, sollten Sie sich mit der Verbesserung des Programms beschäftigen. Zur Übersicht das komplette Programm als Listing (Diesmal mit dem BASIC 3.0 Befehl RECORD#):

```
10 REM TELEFONNUMMERNVERWALTUNG V1.0
20 REM A$ := ANTWORTZEICHEN FUER MENUE
30 REM DT$ := DATENSATZSTRING
40 REM DL := DATENSATZLAENGE
50 REM DN := DATENSATZNUMMER
60 REM B$ := BLANKSTRING
70 REM N$ := NAME
80 REM T$ := TELEFONNUMMER85 REM TT$,NN$ := KOPIEN VON N$,T$
90 DL = 46:B$ = "":OPEN15,8,15:REM OEFFNEN BEFEHLSKANAL
95 FOR X=1 TO DL:B$ = B$ + " ":NEXT X
100 PRINT "TELEFONLISTE V1.0"
110 PRINT "(1) NEU ANLEGEN"
120 PRINT "(2) EINGEBEN"
130 PRINT "(3) LOESCHEN"
140 PRINT "(4) KORRIGIEREN"
```

```
150 PRINT "(5) DRUCKEN"
160 PRINT "(6) SUCHEN"
170 PRINT "(7) SORTIEREN"
180 PRINT "(8) ENDE"
190 GET A$:IF A$="" THEN GOTO 180
200 IF (A$<"1") OR (A$>7) THEN PRINT "+++ EINGABEFEHLER":GOTO 100
210 ON VAL(A$) GOTO 300,400,500,600,700,800,900,1000
300 REM +++ ANLEGEN EINER NEUEN DATEI
305 PRINT "+++ DATEI 'TELEFON' WIRD ANGELEGT"
310 REM OEFFNEN MIT DATENSATZLAENGE 46
315 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
320 PRINT "+++ FREIGABE VON 100 DATENSAETZEN"
325 REM 100 DATENSAETZE FREIGEBEN
330 RECORD#15,100,1:PRINT#1,CHR$(255);
335 PRINT "+++ DATEI 'TELEFON' ANGELEGT"
340 CLOSE1
345 GOTO 100
400 REM +++ EINGABE
405 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
410 DT$ = "":REM VORBEREITEN DES NEUEN DATENSATZES
415 INPUT "DATENSATZNUMMER";DN
420 INPUT "NAME";N$
425 INPUT "TELEFON";T$
430 DT$ = N$ + LEFT$ (B$,33 - LEN (N$))
435 DT$ = DT$ + T$ + LEFT$ (B$,13 - LEN (T$))
440 REM POSITIONIERUNG AUF DATENSATZ RN, BYTE 1
445 RECORD#15,DN,1
450 REM SCHREIBEN DES DATENSATZES
455 PRINT#1,DT$;
460 INPUT "NOCHMAL (J/N) ";AA$
465 IF (AA$ <> "N") THEN GOTO 410
470 CLOSE1:GOTO 100
500 REM +++ LOESCHEN
505 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
510 PRINT "WELCHER DATENSATZ WIRD GELOESCHT?
515 PRINT "(EINGABE VON 0 KEHRT INS MENUE ZURUECK)"
520 INPUT DN
525 IF (DN = 0) THEN GOTO 100
530 RECORD#15,DN,1
535 DT$ = B$
540 PRINT#1,DT$;
545 PRINT "+++ DATENSATZ NR.";DN;" GELOESCHT!"
550 CLOSE1
555 GOTO 100
600 REM +++ KORRIGIEREN
605 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
610 INPUT "WELCHEN DATENSATZ KONTROLLIEREN";DN
615 RECORD#15,DN,1
620 INPUT#1,DT$
625 IF (DT$ = B$) THEN PRINT "+++ DATENSATZ GELOESCHT!":GOTO 100
630 N$ = LEFT$ (DT$,33)
635 T$ = RIGHT$ (DT$,13)
```

```
640 PRINT "INHALT DES DATENSATZES NR.";DN;":"
645 PRINT "(RETURN ALLEIN = KEINE AENDERUNG)"
650 PRINT N$;:INPUT NN$:IF (NN$ <> "") THEN N$ = NN$
655 PRINT T$;:INPUT TT$:IF (TT$ <> "") THEN T$ = TT$
660 DT$ = "":N$ = LEFT$ (N$,33):T$ = LEFT$ (T$,13)
665 IF (LEN (N$) < 33) THEN N$ = N$ + LEFT$ (B$,33 - LEN (N$))
670 IF (LEN(T$) < 13) THEN T$ = T$ + LEFT$ (B$,13-LEN (T$))
675 DT$ = N$ + T$
680 RECORD#15,DN,1
685 PRINT#1,DT$;
690 CLOSE1
695 GOTO 100
700 REM +++ DRUCKEN
705 OPEN1,8,2,"TELEFON,L,"+CHR$(DL):OPEN4,4
710 Z = 1
715 RECORD#15,Z,1
720 INPUT#1,DT$
725 EOF = (64 AND ST)
730 IF EOF THEN GOTO 755
735 IF (DT$ = B$) THEN GOTO 745
740 PRINT#4,DT$
745 Z = Z + 1
750 GOTO 715
755 CLOSE1:CLOSE4
760 GOTO 100
800 REM +++ SUCHEN
805 OPEN1,8,2,"TELEFON,L,"+CHR$(DL)
810 INPUT "WELCHER NAME WIRD GESUCHT";NA$
815 Z = 1
820 RECORD#15,Z,1
825 INPUT#1,DT$
830 EOF = (64 AND ST)
835 IF EOF THEN GOTO 855
840 IF (NA$ = LEFT$ (DT$,LEN (NA$)) THEN PRINT DT$
845 Z = Z + 1
850 GOTO 820
855 CLOSE1
860 GOTO 100
900 REM +++ SORTIEREN
905 REM H$ := HILFSVARIABLE FUER TAUSCHVORGANG
910 REM SW := 'SWAPPED', GETAUSCHT-FLAG
915 OPEN1,8,2,"TELEFON,L,"+CHR$(DL):PRINT "+++ SORTIEREN";
920 SW = 0:REM FLAG INITIALISIEREN
925 FOR X = 1 TO 99
930 RECORD#15,X,1
935 INPUT#1,D1$
940 RECORD#15,X+1,1
945 INPUT#1,D2$
950 IF (D1$ < D2$) THEN GOTO 980
955 RECORD#15,X,1
960 PRINT#1,D2$;
965 RECORD#15,X+1,1
```

```
970 PRINT#1,D1$;
975 SW = -1
980 NEXT X
985 IF SW THEN GOTO 920
990 PRINT " BEENDET"
995 GOTO 100
1000 REM +++ BEENDEN
1010 INPUT "BEENDEN? (J/N) ";AA$
1020 IF (AA$ <> "J") THEN GOTO 100
1030 CLOSE15:REM SCHLIESSEN DES BEFEHLSKANALS
1040 PRINT "THAT'S ALL, FOLKS!"
1050 END
```

9. ISAM-Dateien

"digital 1) mit dem Finger. 2) ziffernmäßig. 3) In der Datenverarbeitung: zahlenmäßig; in der Meßtechnik: quantitativ."

Großes Universal Lexikon in Farbe[11]

Wir lernen eine weitere Methode des Dateizugriffs kennen, lernen die Vorzüge und Nachteile dieser Methode und bekommen an einem Beispiel demonstriert, wie man sie realisiert.

Bisher haben wir über verschiedene Dateitypen gesprochen, die von den Commodore-Laufwerken unterstützt werden. Wenn man jedoch sehr große Datenmengen organisieren will, tauchen neue Probleme und natürlich auch zusätzliche Wünsche auf. Ist zum Beispiel der Datenbestand zu aktualisieren, so muß man eine große Datenmenge "umschaufeln", was viel Zeit kostet. Aber bei Datenbanken im harten Alltagseinsatz ist Zeit sehr teuer. Die Leitungsgebühren zu den Datenbanken, Zugriffszeiten, all das kostet viel Geld. Man ist also bemüht, die Suchzeiten zu minimieren.

Ein weiterer Wunsch wäre, die Daten nach verschiedenen Kriterien gleichzeitig zu sortieren. Stellen wir uns folgenden Situation vor:

Eine Datenbank bietet Literaturhinweise aus dem Bereich Medizin an. Kunden einer solchen Datenbank kommen aus verschiedenen Bereichen und haben somit auch verschiedene Interessen. Die eine Benutzergruppe hätte gerne alle Daten über Krebs, eine weitere Gruppe möchte alle Publikationen aus dem Jahre 1987 wissen, um ihren Bestand zu aktualisieren, eine dritte Gruppe gar möchte alle deutschen und englischen Publikationen über

11 Großes Universal Lexikon in Farbe, Honos Verlags AG Zug

Allergien seit dem Jahre 1980 wissen. Sie sehen, wie vielfältig die Wünsche sein können. Sie sehen wahrscheinlich aber auch, daß es mit den bisher besprochenen Dateiorganisations-Methoden nicht möglich ist, alle diese Wünsche in sinnvoller Zeit zu erfüllen. Es ist auch nicht vertretbar, bei jeder neuen Anfrage die gesamte Datei zu sortieren. Man braucht also eine andere Methode. Diese Methode ist unter dem Namen ISAM bekannt. ISAM steht für "Index Sequentiell Access Method".

Wie der Name schon verrät, ist diese Methode mit der sequentiellen Datei verwandt, hat aber nicht deren Nachteile.

Die Methode

Wenn man eine Datei auf die herkömmliche Methode sortiert, so ist man gezwungen, die einzelnen Daten wirklich "physisch" auszutauschen, was, wie schon erwähnt, viel Zeit in Anspruch nimmt. Folgender Trick schafft nun Abhilfe:

Wenn die Datensätze das erste Mal in die Datei eingelesen werden, werden sie unabhängig von ihrem Sortierungsgrad numeriert. Die Nummern der Datensätze werden in genau dieser Reihenfolge in eine weitere Datei, die sogenannte "Indexdatei", geschreiben. Soll die Datei sortiert werden, so werden nicht die eigentlichen Datensätze, sondern nur deren Nummern in der Indexdatei sortiert. Natürlich muß man dann auf die Datensätze immer über die Indexdatei zugreifen, denn die eigentliche Datei ist ja immer noch unberührt!

Ich will dies an einem Bild erläutern: Angenommen, sie wollen Namenschildchen sortieren. Der Einfachheit halber nehmen wir an, daß es nur drei Namen sind. Die eine Methode besteht darin, die Schilder auf dem Tisch solange hin und her zu bewegen, bis die Namen von oben nach unten dem Alphabet nach sortiert da liegen. Die andere Methode verlangt eine Numerierung der Schildchen. Dann werden die Nummern auf eine Liste geschrieben. Hier greift der Sortiervorgang nicht auf die Namensschildchen zu, sondern auf die Liste mit den Nummern. Es werden

nur die Nummern der Liste sortiert. Will man dann die Namen auf den Schildchen in der richtigen Reihenfolge lesen, dann werden einfach die Namensschildchen in der Reihenfolge vorgelesen, die auf der Liste vermerkt ist. Eine solche Liste kann nach dem Sortieren so aussehen:

Namensschildchen	Liste	
Susanna	1	3
Martina	2	2
Andrea	3	1

Wieder auf unser Problem übertragen entsprechen die Namensschildchen der eigentlichen Datei, die Liste entspricht der Indexdatei. Wahrscheinlich haben Sie jetzt schon einen weiteren Vorteil erkannt, den Indexdateien mit sich bringen:

Was hindert uns denn daran, für eine Datei mehrere Indexdateien zu erzeugen, die nach verschiednen Kriterien sortiert worden sind? Nichts! Richtig! Und genau so wird es gemacht. Die Kriterien nennt man Sortierschlüssel und diese erlauben tatsächlich die mehrfache Sortierung ein und derselben Datei. ISAM-Dateien sind also nicht nur schneller während des Sortiervorgangs und des Suchzugriffs, sie sind auch flexibler in der Handhabung von Sortierschlüsseln.

Natürlich hat auch die ISAM-Datei einige Nachteile, die ich Ihnen nicht vorenthalten will. Um Datensätze einzufügen, braucht man Platz innerhalb der Datei. Man hat also dafür zu sorgen, daß immer genügend Platz zwischen den Datensätzen ist. Ansonsten hat man Platz zu schaffen. Die Schlüsselverwaltung ist sehr aufwendig, da man zum Beispiel beim Einfügen oder Löschen eines Datensatzes nicht nur die Datei, sondern alle zugehörigen Indexdateien ebenfalls zu ändern hat. ISAM-Dateiverwaltungen sind also nicht nur komplex, sondern es werden meist auch längere Programme.

Für ISAM-Dateien gelten im übrigen alle Regeln, die auch für sequentielle Dateien gelten. Das "S" in ISAM steht ja auch für sequentiell. Man darf also nur am Ende der Datei Datensätze

anfügen, und man darf entweder nur lesen oder nur schreiben. Es gibt viel Literatur über dieses Thema, einführend[12] oder theoretisch[13]. Aber ein kleines Beispiel will ich Ihnen nicht vorenthalten. Es ist eine Erweiterung unseres Telefonregisters und verwaltet komplette Adressen.

Zum Programm

Um ordentlich mit der Datei arbeiten zu können, werden wir in der Datei im ersten Eintrag die Anzahl der enthaltenen Datensätze vermerken. Wird ein Datensatz gelöscht, wird er nicht wirklich entfernt, sondern nur als gelöscht gekennzeichet. Soll ein neuer Datensatz eingefügt werden, werden solchermaßen gekennzeichneten Plätze in der Datei vorrangig belegt. Bei der Benutzung des Programms wird man im Gegensatz zu anderen Methoden, vor allem bei großen Dateien, eine merkliche Geschwindigkeitszunahme beobachten können.

```
10 DEFFNH(X)=INT(X/256)
20 DEFFNL(X)=X-FNH(X)*256
30 XD=100:DIM S%(XD),S1%(XD)
40 OPEN 1,8,15,"IO":C3$=CHR$(13)
50 GOSUB 10000
60 OPEN 9,8,9,"0:ISAM.DTA,L,"+CHR$(130)
1000 REM MENUE
1001 PRINT:PRINT
1010 PRINT"ISAM-ADRESSDATEIVERWALTUNG"
1020 PRINT"=========================="
1030 PRINT
1040 PRINT"EINGABE ---------------- 1"
1050 PRINT"AUSGABE ---------------- 2"
1060 PRINT"LOESCHEN --------------- 3"
1070 PRINT"SUCHEN ----------------- 4"
1075 PRINT"ALLES RUECKSETZEN ------ 5"
1080 PRINT"ENDE ------------------- 6"
1090 PRINT
1100 INPUT"UND JETZT? ";WI
1110 IF WI>6 OR WI<1 OR WI<>ABS(INT(WI)) THEN 1000
1120 ON WI GOSUB 24000,22000,20000,23000,30000,9000
```

12 z.B. Schönleber: Hitchhacker's Guide to BASIC, Kiel 1987, Verlag Claus Schönleber

13 etwa Wirth: Algorithmen und Datenstrukturen, Stuttgart 1983, 3.Auflage

```
1130 GOTO 1000
9000 CLOSE 9:REM REL-FILE DICHTMACHEN
9010 GOSUB 11000:REM INDEX WEGLEGEN
9020 CLOSE 1
9030 END
10000 REM EINLESEN DER INDEXDATEI, VORHER LOESCHEN DES ALTEN INDEX
10001 FOR I=0 TO XD:S%(I)=0:NEXT I
10010 OPEN 8,8,8,"0:ISAM.IDX,S,R"
10011 INPUT#1,XX:IF XX<>0 THEN MX=0:GOTO 10060
10020 INPUT#8,MX:IF MX<=0 THEN 10060
10030 FOR I=1 TO MX
10040 INPUT#8,S%(I)
10050 NEXT I
10060 CLOSE 8
10070 RETURN
11000 REM WEGLEGEN DER INDEXDATEI, VORHER LOESCHEN DER ALTEN
11001 PRINT#1,"S0:ISAM.IDX"
11010 OPEN 8,8,8,"0:ISAM.IDX,S,W"
11020 PRINT#8,MX:IF MX<=0 THEN 11060
11030 FOR I=1 TO MX
11040 PRINT#8,S%(I)
11050 NEXT I
11060 CLOSE 8
11070 RETURN
12000 REM EINLESEN EINES DATENSATZES VON (PHYSIKALISCHER) POS. R
12010 REM REL. DATEI #9 MUSS OFFEN SEIN, EBENSO FEHLERKANAL #1
12020 PRINT#1,"P"+CHR$(96+9)+CHR$(FNL(R))+CHR$(FNH(R))+CHR$(0)
12021 REM DAS ERSTE ZEICHEN EINES DATENSATZES ENTHAELT CHR$(255),
12022 REM WENN DER SATZ NOCH NIE ANGELEGT WURDE, SONST (EIN BENUTZER-
12023 REM DEFINIERTES ZEICHEN. WENN CHR$(255) VORGEFUNDEN WIRD,
12024 REM DARF SOMIT NICHT WEITERGELESEN WERDEN.
12025 GET#9,M$:REM MARKIERUNG LESEN
12026 IF M$=CHR$(255) THEN N$="":W$="":T$="":S$="":GOTO 12070:REM LEER
      SATZ
12030 INPUT#9,N$
12040 INPUT#9,S$
12050 INPUT#9,W$
12060 INPUT#9,T$
12070 PRINT#1,"P"+CHR$(96+9)+CHR$(FNL(R))+CHR$(FNH(R))+CHR$(0)
12071 LZ=LZ+1:REM ANZAHL LESEZUGRIFFE
12080 RETURN
13000 REM WEGLEGEN EINES DATENSATZES AB (PHYSIKALISCHER) POS. R
13020 PRINT#1,"P"+CHR$(96+9)+CHR$(FNL(R))+CHR$(FNH(R))+CHR$(0)
13030 D$=" "+N$+C3$+S$+C3$+W$+C3$+T$+C3$
13040 PRINT#9,D$
13050 PRINT#1,"P"+CHR$(96+9)+CHR$(FNL(R))+CHR$(FNH(R))+CHR$(0)
13060 RETURN
20000 REM LOESCHEN DES DATENSATZES MIT INDEX ID
20001 INPUT"INDEX DES ZU LOESCHENDEN SATZES";ID
20002 IF ID<0 OR ID>MX OR ID<>INT(ABS(ID)) THEN RETURN
20010 R=S%(ID):REM POSITION ERMITTELN
20020 GOSUB 12000:REM SATZ LESEN
```

```
20030 PRINT N$:PRINT S$:PRINT W$:PRINT T$
20040 PRINT"DIESEN SATZ LOESCHEN (J/N) ?";
20050 GET C$:IF C$<>"J" AND C$<>"N" THEN 20050
20060 PRINT C$
20070 IF C$<>"J" THEN RETURN
20080 REM SATZ LOESCHEN DADURCH, DASS NACHFOLGENDE INDICES UM EINE
20090 REM STELLE VORGEZOGEN WERDEN
20100 FOR I=ID TO XD-1
20110 S%(I)=S%(I+1)
20120 NEXT I
20130 MX=MX-1:RETURN
22000 REM AUSGABE ALLER DATENSAETZE
22010 IF MX<=0 THEN RETURN : REM NICHTS AUSZUGEBEN
22020 FOR I=1 TO MX
22030 R=S%(I):REM POS. IN DATEI
22040 GOSUB 12000: REM SATZ LESEN
22050 FOR J=1 TO 35:PRINT"-";:NEXT J
22055 PRINT:PRINT"POS. LOG.:";I;"; POS. PHYS. :";S%(I)
22060 PRINT N$:PRINT S$:PRINT W$:PRINT T$
22070 NEXT I
22080 RETURN
23000 REM AUFSUCHEN EINER PERSON IN DER DATEI
23001 LZ=0
23010 IF MX<=0 THEN RETURN
23020 INPUT"WAS SUCHEN";N1$
23030 REM GESCHACHTELTE SUCHE MIT INTERVALLHALBIERUNG
23031 REM DATENSAETZE AUF INTERVALLGRENZEN VORRANGIG ABFRAGEN
23032 R=S%(1):GOSUB 12000:IF N$=N1$ THEN 23260
23033 R=S%(MX):GOSUB 12000:IF N$=N1$ THEN 23260
23040 G1=1:G2=MX:REM INITIALE BEREICHSGRENZEN
23050 REM SUCHSCHLEIFE
23060 G3=INT((G1+G2)/2):REM ZUGRIFF IMMER IN INTERVALLMITTE
23066 REM PRINT"*";G1;G2;G3
23070 R=S%(G3):REM SATZ LESEN
23080 GOSUB 12000
23090 IF N1$=N$ THEN 23260
23100 IF N$<N1$ THEN 23160
23110 REM HIER IST DER GESUCHTE DATENSATZ GROESSER ALS DER AN
23120 REM DER POS. G3 VORGEFUNDENE, ALSO MUSS DER GEWUENSCHTE
23130 REM IN DER UNTEREN HAELFTE DES INTERVALLS LIEGEN
23140 REM VERLEGE ALSO DIE ALTE OBERE GRENZE G2 IN DIE INTERVALLMITTE
23150 G2=G3:GOTO 23230
23160 REM HIER IST DER GESUCHTE DATENSATZ KLEINER ALS DER AN
23170 REM DER POS. G3 VORGEFUNDENE, ALSO MUSS DER GEWUENSCHTE
23180 REM IN DER OBEREN HAELFTE DES INTERVALLS LIEGEN
23190 REM VERLEGE ALSO DIE ALTE UNTERE GRENZE G2 IN DIE INTERVALLMITTE
23200 G1=G3
23210 REM PRUEFE, OB INTERVALL MINIMAL GEWORDEN IST. WENN JA,
23220 REM DANN BEENDE SUCHE, DA EINTRAG NICHT AUFFINDBAR
23230 REM PRINT"#";G1;G2;G3
23235 IF G1<G2-1 THEN 23060
23240 PRINTN1$;" GIBT'S HIER LEIDER NICHT."
```

```
23250 GOTO 23300
23260 REM GEFUNDEN, ALSO AUSGEBEN
23270 FOR J=1 TO 35:PRINT"-";:NEXT J
23280 PRINT:PRINT"POS. LOG.:";G3;"; POS. PHYS. :";S%(G3)
23290 PRINT N$:PRINT S$:PRINT W$:PRINT T$
23300 PRINT"ANZAHL LESEZUGRIFFE:";LZ
23310 RETURN
24000 REM EINLESEN UND EINBAU EINES NEUEN DATENSATZES
24001 LZ=0
24010 PRINT"NEUEINGABE"
24020 INPUT"NAME";N1$
24030 INPUT"STRASSE";S1$
24040 INPUT"WOHNORT";W1$
24050 INPUT"TELEFON";T1$
24060 GOSUB 25000: REM SUCHE FREIEN PLATZ AUF DISKETTE
24070 REM WENN GEFUNDEN, DANN WEGLEGEN
24080 R=P:N$=N1$:S$=S1$:W$=W1$:T$=T1$:GOSUB 13000
24090 REM AUF PLATTE ISSER ANGEKOMMEN
24100 REM JETZT WIRD'S KNACKIG: DER INDEX MUSS KORRIGIERT WERDEN
24110 REM DAZU VERWENDEN WIR DAS VERFAHREN VON DA OBEN
24130 REM GESCHACHTELTE SUCHE MIT INTERVALLHALBIERUNG
24140 G1=1:G2=MX:REM INITIALE BEREICHSGRENZEN
24145 IF MX=0 THEN P=1:G3=1:GOTO 24260
24150 REM SUCHSCHLEIFE
24160 G3=INT((G1+G2)/2):REM ZUGRIFF IMMER IN INTERVALLMITTE
24170 R=S%(G3):GOSUB 12000
24190 IF N1$=N$ THEN 24240
24200 IF N$<N1$ THEN G1=G3:GOTO 24230
24210 G2=G3
24230 IF G1<G2-1 THEN 24160
24231 PS=G1:R=S%(PS):GOSUB 12000
24232 IF (PS>MX) OR (N1$<N$) THEN 24234
24233 PS=PS+1:R=S%(PS):GOSUB 12000:GOTO 24232
24234 G3=PS
24240 REM NU HAMMER IHN
24250 REM PLATZ FUER INDEX SCHAFFEN
24260 FOR I=XD-1 TO G3 STEP -1
24270 S%(I+1)=S%(I)
24280 NEXT I
24290 S%(G3)=P
24291 MX=MX+1
24300 PRINT"ANZAHL LESEZUGRIFFE:";LZ
24310 RETURN
25000 REM SUCHE FREIEN PLATZ IN DER DATEI, DAMIT
25010 REM FRUEHER GELOESCHTE DATENSAETZE KEINE UNNOETIGEN 'LOECHER'
25020 REM HINTERLASSEN (ANDERNFALLS WUERDE DIE DATEI SUKZESSIVE WACHSEN
25030 REM UND IRGENDWANN DIE PLATTE SPRENGEN
25040 REM HIERFUER WERDEN ERSTMAL ALLE RECORDS DER DATEI FUER FREI ER
      KLAERT
25050 FOR I=0 TO XD
25060 S1%(I)=0
25070 NEXT I
```

```
25080 REM JETZT WERDEN AUS DIESER FREILISTE ALLE IM INDEXBEREICH VER-
25090 REM MERKTEN UND DAHER BELEGTEN RECORDS ANGEKREUZT
25100 REM WAS UEBERBLEIBT, IST MITHIN FREI ZUR VERFUEGUNG
25110 FOR I=1 TO MX
25120 S1%(S%(I))=1
25130 NEXT I
25140 REM ALLES RAUSGESTRICHEN. SUCHE JETZT DEN ERSTEN FREIEN RECORD
25150 P=-1
25160 FOR I=MX TO 1 STEP -1
25170 IF S1%(I)=0 THEN P=I
25180 NEXT I
25190 IF P>0 THEN 25230 : REM GEFUNDEN => FERTIG
25200 REM DATENBEREICH IST KOMPAKT UND DICHT
25210 REM ES MUSS ALSO AM ENDE ANGEFUEGT WERDEN
25220 P=MX+1
25230 RETURN
30000 REM DATEISYSTEM RUECKSETZEN
30001 PRINT"WIRKLICH ALLES LOESCHEN (^J = JA ) ?"
30002 GET C$:IF C$="" THEN 30002
30003 IF C$<>CHR$(10) THEN PRINT"*** ABORTED ***":RETURN
30004 PRINT"GANZ SICHER (CBM + J = JA ) ?"
30005 GET C$:IF C$="" THEN 30005
30006 IF C$<>CHR$(181) THEN PRINT"*** ABORTED ***":RETURN
30008 CLOSE9
30009 PRINT#1,"S0:ISAM.*":PRINT#1,"I0"
30010 OPEN 8,8,8,"0:ISAM.IDX,S,W"
30020 PRINT#8,0
30025 CLOSE8
30030 OPEN9,8,9,"0:ISAM.DTA,L,"+CHR$(130)
30050 MX=0:RETURN
READY.
```

10. Diskettenlaufwerke

"Auf der Frontplatte des Gerätes befindet sich ein Schlitz, in den die Diskette eingeführt wird. Durch das Herunterdrücken einer am Schlitz angebrachten Klappe oder eines Knebels wird die Diskette an die Laufwerksachse gekoppelt."

(Deutsche VC1541-Anleitung)

Wir riskieren einen Blick auf die Technik des Diskettenlaufwerks, lernen den "Steppermotor" kennen, der nur in Schritten bewegt werden kann, und einen "Bus", der nicht fahren kann, und beschäftigen uns zum Schluß ein wenig mit dem Verkehrssystem, in dem die Daten zwischen dem Computer und dem Laufwerk hin und her flitzen.

Wenn eine Diskette formatiert wird, entstehen magnetische Unterschiede auf der Diskettenoberfläche. Diese Unterschiede werden als binäre Information aufgefaßt und mit den Ziffern des Dualsystems interpretiert. Zum Beispiel kann man die beiden Polungen Nord- und Südpol mit 0 und 1 assoziieren. Damit wird nun auf die Diskette in Form von Spuren eine "leere Maske" aufgetragen, in die nachher die eigentlichen Daten eingetragen werden. Die Frage, die sofort auftaucht, ist: Wie erkennt man den Anfang des ersten Sektors, oder wie erkennt man überhaupt irgendeinen Sektoranfang?

Die Diskette besitzt ein sogenanntes Indexloch seitlich der zentralen Ausstanzung. Durch die Rotation läuft das Loch in der Scheibe in regelmäßigen Abständen am gleichen Loch in der Hülle vorbei. Mit einer Lichtschranke läßt sich ein Durchgang ermitteln und als Anfang auf der Spur festlegen. Aus der durch das Betriebssystem festgelegten Sektorlänge, der Anzahl pro Spur und der Drehgeschwindigkeit kann man die Zeit berechnen, wann jeder einzelne Sektor am Schreib-/Lesekopf vorbeisaust.

Ein Sektor wird beim Formatieren erst einmal in den Kopf oder Header und in das Datenfeld aufgeteilt. Eingerahmt werden diese beiden Felder eines Sektors von Synchronisationsmarken oder Sync-Marken. Die können statt des Indexlochs zur Erkennung von Sektoranfängen benutzt werden.

Ein weiteres Problem ist die Erkennung der einzelnen Spuren. Dieses Problem ist nur mit geeigneter Hardware zu lösen. Eingesetzt wird ein Steppermotor, der den Schreib-/Lesekopf über die Oberfläche der Scheibe steuert. Dieser Elekromotor heißt so, weil er nicht, wie gewohnt, normal "rund" läuft, sondern weil er sich in definierten kleinen Schritten bewegen kann. Je kleiner und genauer diese Schritte sind, um so besser ist die Qualität, da auch mehr Spuren pro Diskettenseite abgetastet werden können. Es ist Aufgabe eines Teils des Betriebssystems, diese analogen mechanischen Bewegungen (Diskettenrotation, Steppermotor des Tonarms) mit digitalen Informationen zu steuern. Dazu werden Wandler benutzt, die digitale Signale in analoge übersetzen und umgekehrt. Da solche Operationen sehr aufwendig sind, lagert man die kleinen Aufgaben (z.B. Datenänderungen im selben Sektor) solange es geht in den laufwerkseigenen Pufferspeicher aus. Das ist ein vom Computer unabhängiger Speicherbereich, in dem Daten, die zwischen Computer und Laufwerk ausgetauscht werden, zwischengespeichert sind.

Wenn man während einer Dateioperation die eingelegte Diskette gegen eine andere auswechselt, gehen die Daten, die man gerade bearbeitet hat, verloren. In der BAM, die ja im Speicher des Laufwerks steht, ist das Directory der alten Diskette vermerkt. Legt man eine neue Diskette während einer Schreiboperation in das Laufwerk, versucht das Laufwerk, mit den alten Daten im Speicher auf eine Diskette zuzugreifen, die eine völlig andere Sektorenbelegung hat. Dadurch werden die Daten auf der neuen Diskette zerstört, da das Laufwerk ja dachte, noch die alte Diskette vorzufinden. Bei einem programmgesteuerten Diskettenwechsel ist deswegen immer ein INITIALIZE durchzuführen.

Daß die Commodore-Laufwerke einen eigenen Prozessor besitzen, ist nicht selbstverständlich. Aus dieser Tatsache heraus ist es

möglich, dem Laufwerk eine Aufgabe zu übergeben und mit dem Programm im Computer fortzufahren, während das Laufwerk brav seine Aufgabe erfüllt. Beobachten kann man das sehr schön beim Formatieren. Sobald der Formatier-Befehl abgeschickt worden ist, blinkt der Cursor wieder, und sie können mit dem C64 weiterarbeiten, während das Laufwerk die Diskette formatiert. Würde der C64 selbst die Diskette formatieren, so müßten Sie solange warten, bis er damit fertig ist.

Eine Besonderheit bei Commodore-Laufwerken ist durchaus beachtenswert. Normalerweise machen es die Hersteller von Betriebssystemen sich und den Benutzern einfach und verpassen jeder Spur dieselbe Anzahl von Sektoren. Das aber ist Verschwendung; auf die äußeren Spuren passen nämlich mehr Sektoren als auf die inneren. Und gerade diese Tatsache wird bei Commodore-DOS ausgenutzt. Deswegen passen auf Commodore-Disketten immer etwas mehr Bytes als auf Laufwerke mit gleicher Spurzahl, die auf jeder Spur dieselbe Anzahl Sektoren besitzen. In der folgenden Tabelle habe ich die jeweiligen Sektorzahlen pro Spur, unabhängig von der Diskettenseite, aufgeführt:

Spurnummer	Sektoren/Spur
1 - 17	21
18 - 24	19
25 - 30	18
31 - 35	17

Der Puffer

Um mit dem Laufwerk Kontakt aufzunehmen, muß man einen Kanal dorthin öffnen. Die physikalische Verbindung besteht schon durch das Kabel, mit dem das Laufwerk an den Computer angeschlossen worden ist. Das Öffnen des Kanals ist also ein rein programmtechnisches Problem.

Das Verbindungskabel hat bestimmte elektrische und mechanische Eigenschaften. Diese Eigenschaften sind genormt, wenig-

stens in der Typenreihe, an die das Laufwerk angeschlossen werden kann. Diese Norm legt die Spannungen fest, die an den einzelnen Leitungen des Kabels anliegen dürfen, und sie setzt die Form und Belegung des Steckers und der Buchse fest, an die die Leitungen angeschlossen werden. Solch ein genormtes Steckverbindersystem heißt Bus. Über diesen Bus läuft der Datenaustausch zwischen Computer und den Peripheriegeräten, z.B. dem Floppylaufwerk.

Die einzelnen Peripheriegeräte haben feste Adressen, damit man die Daten eindeutig an ein bestimmtes Gerät senden kann. Das ist wie mit Briefen und der Post. Auch Briefe haben eine Adresse auf dem Umschlag, damit der Brief auch den Empfänger korrekt erreicht. Die Adresse des Gerätes "Diskettenlaufwerk" ist 8 für die erste Floppystation und 9 für die zweite Floppystation.

Damit man Daten in mehrere Dateien auf einem einzigen Gerät verschicken kann, ohne daß die Daten in heillosem Durcheinander untergehen, gibt es im Laufwerk so etwas wie Auffangbecken für die anfallende Datenflut. Es sind kleine Speicherbereiche, die vom Prozessor der Floppy verwaltet werden. Diese Speicher heißen Buffer oder Pufferspeicher. Sie "puffern" die unregelmäßig eintreffende Datenflut und ordnen sie so, daß jedes einzelne Byte in die richtige Datei findet. Es gibt 5 Pufferbereiche, die jeweils 256 Bytes groß sind. Sie werden mit Zahlen von 0 bis 4 durchnumeriert.

Wenn Sie einen OPEN-Befehl ausführen, wird einer dieser Pufferbereiche ausgewählt und der logischen Filenummer zugewiesen. Alle Daten, die mit dieser logischen Filenummer geschrieben oder gelesen werden, wandern in diesen Puffer. Mit dem CLOSE-Befehl werden alle restlichen, noch im Pufferbereich stehenden Daten, an den Empfänger gesendet und dann die Datei ordnungsgemäß wieder geschlossen. Schließt man den Fehler- und Befehlskanal (Sekundäradresse 15), so werden automatisch alle anderen noch offenen Kanäle ebenfalls geschlossen.

Mit einer besonderen Form des OPEN-Befehls kann man einen ganz bestimmten dieser Bereiche anwählen. Dazu gibt man dem OPEN-Befehl noch die Nummer des Pufferbereichs mit:

```
OPEN2,8,3,"#2"
```

Damit wird auf Gerät 8 (erste Floppystation) eine logische Datei (Filenummer 2) mit der Sekundäradresse 3 geöffnet. Durch "#2" wird Pufferbereich 2 für den Datenaustausch festgelegt.

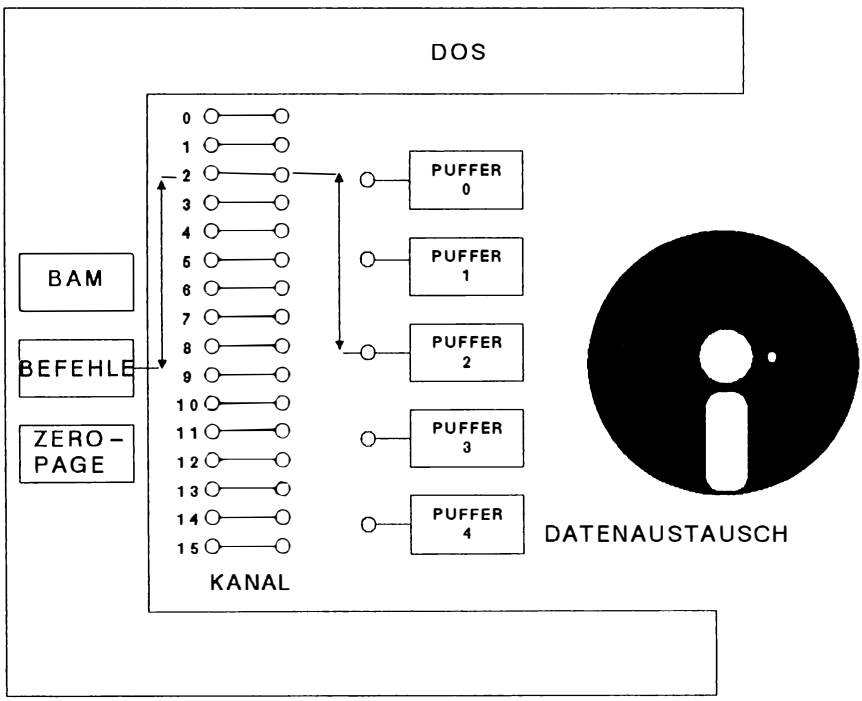


Abb. 2: Blockgraphik Floppybuffer

Da nur 5 Pufferbereiche zur Verfügung stehen, können leider nicht beliebig viele Dateien geöffnet werden.

Für eine relative Datei werden drei Pufferbereiche benötigt:

- Ein Pufferbereich wird vom Computer benötigt.
- Ein Pufferbereich verwaltet die anfallenden Daten.
- Ein Pufferbereich enthält den Side Sektor Block.

Sequentielle Dateien benötigen nur zwei Pufferbereiche:

- Ein Pufferbereich verwaltet die Ein- und Ausgabe.
- Ein Pufferbereich speichert einen Block der Datei.

Aus diesem Grund können höchstens zwei Dateien gleichzeitig offen sein, maximal eine relative und eine sequentielle Datei.

11. Blockbefehle, Direktzugriff

"Das Fliegen...ist eine Kunst oder vielmehr ein Trick. Der Trick besteht darin, daß man lernt, wie man sich auf den Boden schmeißt, aber daneben."

Douglas Adams[14]

Wir steigen in den "Keller" der BASIC-Programmierung, sehen uns einzelne Sektoren auf der Diskette an und basteln uns zwei feine, kleine Programme: Ein Diskettenkopierprogramm und einen "Diskettenmonitor".

Was wir hier kennenlernen, ist für einen BASIC-Dialekt schon außergewöhnlich: Man kann mit wenig Aufwand in einer höheren Programmiersprache auf ganze Sektoren einer Diskette zugreifen. Das ist sonst nur mit guten Kenntnissen in Assembler zu bewältigen. Um so einen Zugriff ausführen zu können, muß zunächst ein Pufferbereich festgelegt werden. Man kann zwar, wie wir oben gelernt haben, den Pufferbereich "von Hand" auswählen, aber wenn es nicht dringend notwendig ist, sollte man die Auswahl des Pufferbereichs dem DOS überlassen. Folgende Pufferbereiche stehen uns zur Verfügung:

Pufferbereich	Adresse hex	dezimal
0	$300 - $3FF	768 - 1023
1	$400 - $4FF	1024 - 1279
2	$500 - $5FF	1280 - 1535
3	$600 - $6FF	1536 - 1728
4	$700 - $7FF	1792 - 2047

Wundern Sie sich nicht, wenn vor jeder Zahl ein "$" steht; auch nicht, daß Buchstaben in den Zahlen vorkommen. Die Adressan-

14 Douglas Adams: Das Leben, das Universum und der ganze Rest, Ullstein 1988

gaben sind im für den Computerbereich üblichen Hexadezimalsystem angegeben. Das ist ein Zahlensystem mit der Basis 16. Zum Vergleich: unser alltäglich benutztes Zahlensystem (Dezimalsystem) hat die Basis 10. Ich habe deshalb auch die dezimalen Werte angegeben, obwohl diese nicht so "glatt" sind und sich deshalb nicht so gut merken lassen.

Von den Pufferbereichen sind nicht alle ohne Einschränkung benutzbar. Bereich 4 enthält üblicherweise die BAM. Bereich 3 enthält während der Arbeit mit "normalen" Dateien (SEQ, REL) noch das Directory. Also stehen uns noch die Bereiche 0 bis 2 zur freien Verfügung.

Mit dem Befehl

```
OPEN2,8,2,"#"
```

wird eine Direktzugriffsdatei (Filenummer 2, Gerät 8, Sekundäradresse 2) geöffnet. In dieser Form sucht das DOS sich selbstständig einen freien Pufferbereich. Damit man darüber informiert wird, welcher Bereich ausgewählt worden ist, gibt es die Möglichkeit, sich die Bereichsnummer ausgeben zu lassen.

Das erste Zeichen, das von einer solchen Datei eingelesen wird, ist die Nummer des ausgewählten Pufferbereichs! Mit einem kleinen Programm probieren wir das aus:

```
10 OPEN2,8,2,"#"
20 GET#2,P$
30 PN = ASC (P$ + CHR$ (0))
40 PRINT "AUSGEWAEHLTER PUFFERBEREICH IST NR.";PN
50 CLOSE2
60 END
```

In Verbindung mit dem Öffnen eines Direktzugriffkanals sollten Sie immer den Fehlerkanal abfragen, ob ein Bereich zur Verfügung gestanden hat. Diese Erweiterung bauen wir deswegen sofort zur Übung in das obige Beispiel ein:

```
10 OPEN15,8,15
20 OPEN2,8,2,"#"
```

```
30 GET#2,P$
40 PN = ASC (P$ + CHR$ (0))
50 PRINT "AUSGEWAEHLTER PUFFERBEREICH IST NR.";PN
60 GET#15,F$:PRINT F$;:IF (64 <> ST) THEN GOTO 60
70 PRINT
80 CLOSE2:CLOSE15
90 END
```

Falls kein Bereich mehr zur Verfügung steht, wird die Fehlermeldung

```
70 NO CHANNEL 00 00
```

ausgegeben. Das passiert aber meist nur, wenn Sie nebenher noch andere Dateien (SEQ, REL) offen haben.

Blockbefehle müssen immer über den Befehlskanal (Sekundäradresse 15) geschickt werden!

Wir lesen einen Block

In der Anleitung zum Diskettenlaufwerk stößt man sofort auf den Befehl "B-R" (als Abkürzung für Block-Read). Vergessen Sie ihn! Er arbeitet fehlerhaft, und so wollen wir nicht näher auf ihn eingehen. Aber es gibt gleichwertigen Ersatz, einen der sogenannten USER-Befehle. Und den wollen wir uns anschauen.

Allgemeine Form:

```
OPEN<lfn>,<ga>,<sad>,"#" PRINT#<lfn>,"U1 <sad> <dn> <sp> <sek>"
```

Beispiel:

```
OPEN1,8,15 OPEN2,8,2,"#" PRINT#1,"U1 2 0 17 5"
```

Damit wird ein Direktzugriffskanal (Nr. 2) auf Gerät 8 mit der logischen Filenummer 2 geöffnet. Zugegriffen wird auf Laufwerk 0 (Sie wissen, die Bezeichnung stammt von den Doppel-

laufwerken!), Spur 17, Sektor 5. Die Sekundäradresse beim Befehl ist die Adresse der Datei, die den Pufferbereich verwaltet (im Beispiel 2).

Nachdem Sie das ausgeführt haben, steht der komplette Sektor in dem vom DOS ausgewählten Pufferbereich. Mit Hilfe von GET# können Sie nun in aller Ruhe die Daten manipulieren: auf dem Bildschirm ausgeben, verändern,...

Wenn Sie Spur=18 und Sektor=0 benutzen, können Sie mit der beschriebenen Methode die BAM in den Speicher holen. Die ersten beiden Bytes bezeichnen dann den ersten Directory-Block. Eine Tabelle über den Aufbau von BAM und Directory ist im Anhang.

Da bei Dateien alle Sektoren miteinander verkettet sind (in einem Sektor steht immer die Adresse des folgenden Sektors), können Sie so recht einfach alle Sektoren ermitteln, die von einer Datei belegt sind. Folgendes kleine Programm ermöglicht es Ihnen, von einem beliebigen Sektor aus, alle folgenden Sektoren der Datei zu errechnen:

```
10 OPEN1,8,15
20 OPEN2,8,2,"#"
30 INPUT "SPUR UND SEKTOR";T,S
40 PRINT#1,"U1 2 0";T;S
50 GET#2,T$,S$
60 T = ASC (T$ + CHR$ (0)):S = ASC (S$ + CHR$ (0))
70 IF (T = 0) THEN GOTO 110
90 PRINT "SPUR ";T,"SEKTOR ";S
100 GOTO 40
110 CLOSE2:CLOSE1
```

In einer Direktzugriffsdatei ist es möglich, wie in relativen Dateien einen Zeiger zu bewegen, mit dem man jedes beliebige Byte ansprechen kann. Der Befehl heißt "B-P" (für "Buffer Pointer"). Er sollte vor dem ersten Zugriff mit GET# oder INPUT# benutzt werden:

```
PRINT#<lfn>,"B-P <sad> <pos>"
```

wobei <sad> Sekundäradresse des Puffers und <pos> Byteposition (0..255) bedeuten.

Wir schreiben einen Block zurück

Auch hierfür steht in der Anleitung ein Befehl, den ich hier nicht benutzen werde, denn er arbeitet ebenfalls fehlerhaft. Also nehmen wir wieder einen U-Befehl: U2.

Allgemeine Form:

```
PRINT#<lfn>,"U2 <sad> <dn> <sp> <sek>"
```

wobei

<sad> Sekundäradresse

<dn> Drivenumber (Laufwerksnummer)

<sp> Spur

<sek> Sektor

bedeuten.

Beispiel:

```
PRINT#1,"U2 2 0 17 5"
```

Damit schreiben wir den Block aus dem Pufferbereich mit der logischen Filenummer 2 wieder auf die Diskette, Spur 17, Sektor 5.

Sie fragen sich sicherlich, wie man denn nun die durch "U1" und GET# gewonnenen Daten wieder in den Puffer bekommt, damit man sie mit "U2" wieder zurückschreiben kann. Ganz einfach! Wir benutzen PRINT# und geben als logische Filenummer die des Direktzugriffkanals an.

Wir ändern den Diskettennamen

Eine kleine Übung schadet nicht. Also schreiben wir ein Programm, das den Diskettennamen ändert. Dazu muß man den Block 0 von Spur 18 in den Puffer lesen. Dort kann er stehen bleiben. Wir bereiten einen String vor, der den neuen Namen enthält. Allerdings muß der Name, wenn er kürzer ist als 16 Zeichen, bis zur 16. Stelle mit SHIFT-Blanks (Zeichen 160 = A0H) aufgefüllt werden.

```
10 OPEN1,8,15
20 OPEN2,8,2,"#"
30 PRINT#1,"U1 2 0 18 0"
40 PRINT#1,"B-P 2 144"
50 INPUT "NEUER NAME";N$
60 IF (LEN (N$) > 16) TEXT PRINT "+++ ZU LANG":GOTO 50
70 IF (LEN (N$) < 16) THEN N$ = N$ + CHR$ (160):GOTO 70
80 PRINT#2,N$;
90 PRINT#1,"U2 2 0 18 0"
100 CLOSE2
110 PRINT#1,"I"
120 CLOSE1
```

Ganz wichtig: das Semikolon ";" in Zeile 80! Würde es weggelassen, wäre der String nicht 16, sondern 17 Zeichen lang, denn es würde noch ein <CR> (Zeichen 13) übertragen werden. Das aber würde ein im Pufferbereich stehendes Byte (Byte Nr. 160) überschreiben. Das Byte hat aber eine Bedeutung, denn sonst würde es dort nicht stehen. Wird es mit dem ASCII-Zeichen 13 überschrieben, kann das ungeahnte Folgen haben. Wenn Sie eine Übungsdiskette benutzen, auf der es nicht schade ist, wenn eine Datei kaputt geht, dann versuchen Sie das einmal und experimentieren solange herum, bis Sie wissen, was dadurch verändert worden ist.

Bei einigem Nachdenken stößt man spätestens jetzt auf die bange Frage:

Wir haben ja jetzt die Möglichkeit, noch nicht belegte Sektoren mit einem (mehr oder weniger) sinnvollen Inhalt zu belegen. Wie, um alles in der Welt, verhindere ich, daß dieser Sektor beim nächsten Abspeichern einer Datei überschrieben wird???

Auch dafür gibt es Befehle:

- Block Allocate ("B-A") und
- Block Free ("B-F")

Die Syntax ist fast identisch:

```
PRINT#<lfn>,"B-A <dn> <sp> <sek>"
PRINT#<lfn>,"B-F <dn> <sp> <sek>"
```

wobei die Platzhalter die gewohnte Bedeutung haben. Ein kleines Programm als Beipiel:

```
10 OPEN1,8,15
20 INPUT "SPUR, SEKTOR";T,S
30 INPUT "1 = BELEGEN, 2 = FREIGEBEN";A
40 IF (A = 1) THEN PRINT#1,"B-A 0";T;S:GOTO 60
50 IF (A = 2) THEN PRINT#1,"B-F 0";T;S
60 CLOSE1 70 END
```

Natürlich kann es sein, daß ein Block, der belegt werden soll, schon belegt ist. In diesem Fall erfolgt die Fehlermeldung 65 ("NO BLOCK"), und es werden die nächsten freien Sektoren angegeben. Sind die Spur- und Sektornummer in der Fehlermeldung beide 0, dann gibt es keinen freien Block mehr, die Diskette ist voll. Es ist allerdings dazu zu sagen, daß auch dieser Befehl noch nicht ganz ausgereift ist. (Anscheinend haben die Programmierer des DOS gerade bei den interessanten Sachen ein wenig geschlafen.)

Versucht man einen freien Block zu belegen, arbeitet der Befehl einwandfrei. Ist der Block allerdings belegt, so werden nicht nur Spur und Sektor des nächsten freien Blocks ausgegeben, sondern auch alle Blöcke auf dieser Spur automatisch belegt.

Es gibt noch eine zweite Fehlermeldung, die auftauchen kann: Fehler 66 ("ILLEGAL TRACK OR SECTOR"). Sie tritt auf, wenn eine Track/Sector-Kombination aufgetaucht ist, die nicht existiert. So gibt es beispielsweise keinen Track 0 oder Track 40.

Denken Sie jedoch an den COLLECT-Befehl: Er gibt alle Blöcke, die so markiert worden sind, für erneute Benutzung frei. Denn COLLECT sucht nur Sektoren, die mit anderen verkettet sind. Welche das sind, ermittelt dieser Befehl aus der Directory. Dort stehen alle Sektoren, in denen eine Datei beginnt. Von dort hangelt er sich durch alle zugehörigen Dateien. Unsere Direktzugriffssektoren springen dabei über die Klinge. Das geschieht deswegen so radikal, weil nur dadurch "Karteileichen" aus der BAM entfernt werden können.

Ein Ausblick

Ich hatte schon oben erwähnt, daß es möglich ist, einen Sektor von der Diskette in den Laufwerksspeicher (Pufferbereich) zu schreiben. Das geht natürlich auch dann, wenn dieser Sektor ein ausführbares Maschinenprogramm enthält. Da das Laufwerk eine eigene CPU hat, kann man hier wunderschön kurze Unterprogramme oder parallele Programme zu Programmen im Computerspeicher basteln. Mit dem Befehl "B-E" kann man Maschinenprogramme im Pufferbereich des Laufwerks starten und damit ausführen lassen. Allerdings sind zwei Dinge zu beachten:

- Man muß einen Pufferbereich auswählen, da Maschinenprogramme meist auf absoluten Adressen arbeiten. Ein Programm im Laufwerkspufferbereich muß also für einen bestimmten Puffer geschrieben werden.

- Man sollte sich hervorragend mit Assembler, dem DOS und den Interna der benutzten Hardware auskennen.

Die allgemeine Form dieses Befehls lautet:

```
PRINT#<lfn>,"B-E <sad> <dn> <sp> <sek>"
```

Dabei bedeuten:

<sad> Kanalnummer (Puffer)

<dn> Laufwerksnummer (0/1)

<sp> Spur

<sek> Sektor

Die Laufwerksnummer stammt von den Doppellaufwerken und ist bei anderen Floppys stets 0. Die Kanalnummer bestimmt, in welchem Puffer das Programm gestartet werden soll. Spur und Sektor geben die Lage des Blocks auf der Diskette an, der im angegebenen Puffer ausgeführt werden soll.

Autostartdateien

Es gibt eine Methode, Dateien in einen oder mehrere Floppypuffer zu laden und von dort automatisch zu starten. Diese Dateien werden Autostartdateien genannt und unterscheiden sich nicht nur durch ihr Aufzeichnungsformat von normalen, ausführbaren Programmen.

Aufbau einer Autostartdatei:

Byte	Funktion
0,1	Startadresse im RAM (Low-/Highbyte)
2	Anzahl der Bytes des Programms (max. 255)
3..n	das Programm selbst (n <= 255)
n+1	Checksum (Prüfsumme) aus den Programmbytes

Autostartdateien sind wie normale Programmdateien sequentiell abgespeichert, müssen aber den Dateityp USR besitzen (geht nicht in BASIC 7.0).

Eine weitere Schwierigkeit ist, daß Autostartdateien in Blöcken abgespeichert sein müssen, wie sie in der obigen Tabelle dargestellt sind. Das bedeutet nicht, daß ein solches Programm nur 255 Bytes lang sein darf. Das Programm wird normal abgespeichert und von einem Konvertierungsprogramm in einzelne USR-Dateiblöcke aufgespalten. Diese Blöcke werden dann von der Floppy seperat abgearbeitet. Das folgende kleine Programm soll uns die zeitraubende Konvertierungsarbeit abnehmen:

```
10 REM ***** Konvertierungsprogramm
20 REM ***** PRG,USR -> Autostart
30 DIM B$(255)
40 INPUT"Programmname: ";PN$
50 INPUT"USER-Dateiname: ";UD$
60 OPEN1,8,0,PN$
70 OPEN8,8,8,UD$+",U,W"
90 GET#1,D$:D$=CHR$(ASC(D$+CHR$(0)))
100 ON SGN (ST) GOTO 290:A=ASC(D$)
110 GET#1,D$:D$=CHR$(ASC(D$+CHR$(0)))
120 ON SGN (ST) GOTO 290:A=A+ASC(D$)
130 PRINT#8,CHR$(A AND 255);CHR$(A/256);
140 P=0
150 FOR N=1 TO 255
160 GET#1,D$:D$=CHR$(ASC(D$+CHR$(0)))
170 P=(257*(P+ASC(D$))/256) AND 255
180 IF 64 AND ST THEN GOTO 220
190 IF SGN(ST) THEN GOTO 290
200 B$(N)=D$
210 NEXT N
220 PRINT#8,CHR$(N)
230 FOR M=1 TO N
240 PRINT#8,B$(M);
250 NEXT M
260 PRINT#8,CHR$(P);
270 A=A+N
280 ON (N/256+1) GOTO 300,140
290 PRINT"*** Fehler beim Diskettenzugriff!"
300 CLOSE1
310 CLOSE8
320 END
```

Wenn Sie mit diesem Programm aus einem "normalen" Programm eine Autostartdatei gemacht haben, können Sie diese mit dem Befehl

```
OPEN1,8,15,"&<dateiname>"
```

laden und automatisch starten. Heißt die erzeugte USR-Datei beispielsweise "TESTAUTO", dann sieht der Befehl so aus:

```
OPEN1,8,15,"&TESTAUTO"
```

Der erste Autostartblock wird dadurch in den Floppyspeicher geladen, und es wird ab der Adresse, die in diesem Block vermerkt ist, gestartet.

Ein kleines Kopierprogramm (1541/1571)

Um die Blockbefehle besser zu verstehen, werden wir hier ein kleines Kopierprogramm angehen; so bekommen wir auch gleich ein wenig Übung im Umgang mit den Blockbefehlen. Es bleibt wie immer Ihnen überlassen, das Programm nach Belieben zu verbessern und zu erweitern. Ich zeige Ihnen nur das notwendige Gerüst. Das Programm funktioniert nur mit bereits formatierten Disketten. Ihre Aufgabe ist es, als erstes Kommentare in das Programm zu setzen. Das Programm ist nicht schnell, im Gegenteil! Es dient jedoch in erster Linie dazu, den Ablauf zu verstehen.

```
10 REM ***** 1541/70 DISKETTENKOPIERPROGRAMM V1.0 7/88
20 REM *****
30 OPEN1,8,15,"I"
40 OPEN2,8,2,"#"
50 T=1:S=1:TC=35:F=-1
60 DIM B$(10000),SC(4),D$(1)
70 FOR I=1 TO TC
80 IF (I < 18) THEN SC(X)=21
90 IF (I >= 18) AND (I < 25) THEN SC(X)=19
100 IF (I >= 25) AND (I < 31) THEN SC(X)=18
110 IF (I >= 31) THEN SC(X)=17
120 NEXT I
130 D$(0)="DAS ORIGINAL":D$(1)="DIE KOPIE"
140 FOR T=1 TO TC
150 F= NOT F
160 PRINT "LEGEN SIE ";D$(ABS(F));" EIN:";
170 GET A$:IF (A$ = "") THEN GOTO 170
180 FOR S=1 TO SC(T)
190 PRINT#1,"U1 2 0";T;S
200 PRINT#1,"B-P 2 0"
210 FOR P=0 TO 255
220 GET#2,XB$:B$(256*(S-1)+P)=CHR$(ASC(XB$+CHR$(0)))
230 NEXT P
240 NEXT S
250 F = NOT F
260 PRINT "LEGEN SIE ";D$(ABS(F));" EIN:"
270 GET A$:IF (A$ = "") THEN GOTO 270
280 FOR S=1 TO SC(T)
290 PRINT#1,"B-P 2 0"
300 FOR P=0 TO 255
310 PRINT#2,B$(256*(S-1)+P);
320 NEXT P
330 PRINT#1,"U2 2 0";T;S
340 NEXT S
350 NEXT T
```

```
360 CLOSE2
370 CLOSE1
380 END
```

Daß man beim Kopieren mit diesem Programm die Disketten nach jeder Spur wechseln muß, ist sicherlich lästig. Versuchen Sie, das zu ändern! Versuchen Sie, aus dieser "lahmen Krücke" ein fähiges und komfortables Kopierprogramm zu basteln, das auch auf Diskettenfehler richtig reagiert.

Und noch eine Übung

Ein Diskettenmonitor: Das ist ein Programm, das genau einen Diskettensektor einliest, auf dem Bildschirm darstellt und auf Wunsch den nächsten holt.

Die Daten werden im folgenden Format dargestellt:

```
<byteadresse>: <byte 1> <byte 2> ..... <byte 7> <c 1><c 2><c 3>..
```

Mit folgenden Bedeutungen:

<byteadresse> Die Bytes eines Sektors werden von 0 bis 255 durchnumeriert.

<byte n> Das Byte an der Adresse n.

<c n> Das an der Adresse n als Zeichen interpretierte Byte.

```
1 REM ***** MINIDISKMONITOR V1.0 7/882 REM *****
3 DEF FN HN(X) = INT(X/16)
4 DEF FN LN(X) = X AND 15
5 DEF FN C(X) = X + 48 - (X > 9) * 7
6 DEF FN H(X) = FN C (FN HN (X))
7 DEF FN L(X) = FN C (FN LN (X))
10 OPEN1,8,15:OPEN8,8,8,"#"
20 INPUT "TRACK, SEKTOR";T,S:IF ((T*S) < 0) THEN GOTO 190
30 PRINT#1,"U1 ";8;0;T;S
40 PRINT#1,"B-P";8;0
50 FOR I=0 TO 31
60 HX$ = CHR$(FN H(I*8)) + CHR$(FN L(I*8))
70 HC$ = ""
80 FOR J = 0 TO 7
```

```
90 GET#8,XS:X=ASC(X$+CHR$(0)):X$=CHR$(X)
100 HX$=HX$ + " " + CHR$(FN H(X)) + CHR$(FN L(X))
110 C$ = "."
120 IF (X>31) AND (X<96) THEN C$=X$:GOTO 140
130 IF (X>128+63) THEN C$ = X$
140 HC$ = HC$ + C$
150 NEXT J
160 PRINT HX$;" ";HC$
170 NEXT I
180 GOTO 20
190 CLOSE8:CLOSE1:END
```

Nach dem Programmstart wird der Benutzer nach dem gewünschten Sektor gefragt. Man muß also zuerst die Spurnummer und dann die Sektornummer in dieser Spur angeben. Danach wird der Inhalt des Sektors eingelesen und auf dem Bildschirm ausgegeben. In jeder Zeile folgen 8 hexadezimal angegebene Bytes, dahinter steht die Interpretation des jeweiligen Bytes als ASCII-Zeichen. Nicht druckbare Zeichen werden als "." dargestellt. Mit diesem Diskmonitor können Sie nun zum Beispiel die Spur 18 (Directoryspur) untersuchen, sowie die BAM, die ja in Sektor 0 der Spur 18 steht. Prüfen Sie mit diesem Programm die Spur 18 auf ihren Inhalt, wie er in diesem Buch beschrieben wurde. Wenn Sie Lust haben, versuchen Sie das Programm so abzuändern, daß Sie Änderungen im Diskettensektor mit Hilfe des Diskmonitors vornehmen und den veränderten Sektor wieder auf die Diskette schreiben können.

Viel Spaß beim Ausprobieren!

12. Disketten - Wie werden Sie behandelt?

"Wir sehen die klügsten, verständigsten Menschen im gemeinen Leben Schritte tun, wozu wir den Kopf schütteln müssen."

Adolph Freiherr von Knigge[15]

Hier lernen wir die Grundregeln im Umgang mit den billigen kleinen schwarzen Scheiben kennen, die uns Computeranwendern so teuer sind. Außerdem zeigt uns der Autor, wie man, nachdem man leichtsinnig seine Daten ertränkt hat, die Information doch noch retten kann.

Disketten sind ein notwendiges Utensil beim Betrieb mit Micro- und sogar mit Minicomputern. Auch Großrechner brauchen mindestens eine Möglichkeit, Daten über irgendein Diskettenformat austauschen zu können. Disketten sind inzwischen so billig geworden, daß man im Umgang mit Ihnen vergißt, wie wertvoll die enthaltene Information oft ist. Eine Diskette ist schnell wieder gekauft, aber die Daten, die man im Schweiße seines Angesichts (oder im Schweiße des Angesichts seiner Angestellten) in den Computer eingegeben (oder hat eingeben lassen) und auf Diskette gespeichert hat, sind meistens nicht reproduzierbar und rettungslos verloren, wenn mit der Diskette ein Mißgeschick passiert. Eine Privatperson kann den Verlust eigener Daten oft noch verschmerzen, wenn auch zähneknirschend. Um die Arbeitszeit ist es trotzdem schade. Ein kleines Geschäft oder gar eine Firma können es sich aber auf keinen Fall leisten, die Kundendatei oder die Datei mit dem gesamten Lagerbestand zu verlieren, weil die Diskette falsch behandelt wurde.

Bevor wir auf die Regeln eingehen, die man im Umgang mit Disketten beachten sollte, will ich noch kurz über das sprechen,

---

15 Adolph Freiherr von Knigge: Über den Umgang mit den Menschen

was die Herstellerfirma von sich aus schon zum Schutz der Diskette tut. Dabei wollen wir nicht nur die 5¼"-Diskette besprechen, sondern auch ihren modernen Nachfolger, die 3½"-Diskette.

Die klassische 5¼"-Diskette

Diese Diskette besteht aus sechs Komponenten:

- Die eigentliche Diskette.
- Der Verstärkungsring
- Das Vlies
- Die Schutzhülle
- Das Etikett
- Die Schutztasche

Der eigentliche Datenträger besteht aus einer weichen, flexiblen Kunststoffscheibe, die mit magnetischem Material beschichtet ist. Das ist dasselbe Prinzip wie bei einem Tonband oder einer Cassette. Ein Schreib-/Lesekopf im Diskettenlaufwerk wird auf die beschichtete Oberfläche aufgesetzt und schleift über die Plattenoberfläche.

Ein Zapfen im Laufwerk, der durch das große Mittelloch in der Scheibe geführt ist, preßt mit seinem Rand die Diskette gegen den Antriebsmechanismus. Ein Elektromotor versetzt die Scheibe bei Bedarf in Rotation, und der Schreib-/Lesekopf kann die Information von der Diskette lesen oder auf die Diskette schreiben. Der Verstärkungsring, der auf den Rand des Mittellochs geklebt ist, verhindert, daß der Rand des Lochs ausleiert oder ausfranst. Heute hat jede Diskette einen Verstärkungsring, da die starke Beschleunigung beim Anfahren des Motors jede Diskette mit der Zeit stark beschädigen würde.

Um die weiche Scheibe herum ist das sogenannte Vlies angeordnet. Man könnte das Vlies als Innenhülle bezeichnen. Es ist

aus sehr weichem Zellstoff, der verhindern soll, daß sich die empfindliche Oberfläche der Scheibe bei der schnellen Rotation abnutzt. Ganz kann das natürlich nicht verhindert werden, aber die Scheibe wird sehr viel eher durch den Schreib-/Lesekopf zerstört als durch das Vlies.

Die Außenhülle besteht aus einem zwar flexiblen, aber stabilen Kunststoff. In die Hülle ist ein Mittelloch gestanzt, das etwas größer ist als das der Scheibe. Weiterhin geht nahe des Mittellochs bis hin zum äußeren Rand eine schmale Ausstanzung, die (bei Rotation) den Zugriff auf alle Spuren erlaubt, die auf die Diskette aufgetragen werden. Dieser Bereich auf der Scheibe ist besonders glattpoliert und er unterscheidet sich in der Oberflächenstruktur etwas von den Randbezirken. Auf der Rückseite der Schutzhülle befinden sich die Schweißpunkte, mit denen die Hülle zusammengeklebt wurde. Übrigens hat jede Herstellerfirma eigene Muster und eine eigene Anordnung für die Schweißpunkte. So kann man mit etwas Geschick sogenannte "No-Name-Produkte", Disketten ohne Firmenlabel, der wirklichen Herstellerfirma zuordnen. Eine kleine Einkerbung an der rechten Seite stellt die Schreibschutzkerbe dar. Ist sie offen, so kann auf der Diskette geschrieben werden. Ist sie mit einem der mitgelieferten Klebestreifen überklebt, so kann keine Information mehr auf die Diskette geschrieben werden, es können nur noch Daten gelesen werden.

Das Etikett ist nicht nur Werbeträger für das Firmensignet, sondern es enthält die Information über den Einsatzbereich der Diskette. Die Aufzeichnungsdichte, wieviel Seiten beschreibbar sind, wieviel Spuren, alles das ist auf dem Etikett vermerkt. Außerdem sollte sich mindestens ein Klebeetikett auf der Diskette befinden, auf dem vermerkt ist, welche Daten bzw. Programme sich auf ihr befinden. Und schließlich bezeichnet das Etikett ja auch den Bereich, an dem man eine Diskette ausschließlich anfassen sollte.

Die Schutztasche ist der einzige Aufbewahrungsort, der außerhalb des Laufwerk für die Diskette zugelassen ist. Die Schutztasche sollte immer an der Diskette verbleiben. Machen Sie es sich

zur Gewohnheit, eine Diskette sofort nach Gebrauch wieder in die Schutzhülle zu stecken. Dabei sind zwei Dinge zu beachten:

1. Die Diskette sollte immer so in der Tasche stecken, daß das Etikett zu sehen ist. D.h., eine Diskette wird immer mit der für den Schreib-/Lesekopf offenen Seite zuunterst in die Tasche gesteckt.

2. Sorgen Sie dafür, daß Sie eine Diskette immer in eine Tasche stecken, deren Firmenaufdruck dem der Herstellerfirma der Diskette entspricht. So behalten Sie die Übersicht. Man merkt sich sehr leicht, welche Daten man auf welcher Diskette hat. Wenn dann die Disketten jedesmal in anderen Taschen stecken, sucht man länger nach einer bestimmten Diskette.

Die moderne 3½"-Diskette

Wenn man eine solche Diskette zum ersten Mal in die Finger bekommt, nach langen Jahren als 5¼"-Diskettenbenutzer, wird man zuerst einmal vergeblich nach dem Schlitz suchen, über den der Schreib-/Lesekopf den Kontakt zur Scheibe bekommen soll. Auch die Hülle fühlt sich merkwürdig stabil an. Beinahe macht dieses "Ding" den Eindruck eines außerirdischen Sandwiches.

Die Details dieser Platte sind:

- Die eigentliche Diskette
- Die Blechscheibe
- Die Schutzhülle aus Hartkunststoff
- Der Schieber aus Metall, Federmechanismus
- Der Schreibschutzschalter

Die Scheibe der Diskette unterscheidet sich nicht wesentlich von der der 5¼"-Diskette, sie ist nur kleiner. Statt eines Verstär-

kungsringes ist auf der Scheibe eine metallene Scheibe angebracht, in die der Antriebszapfen einrastet und die Scheibe ohne Schlupf antreiben kann.

Die Schutzhülle besteht aus einem harten, kaum flexiblen Kunststoff. Ihre beiden großen Flächen haben einen konstanten Abstand, der wesentlich mehr Platz bietet, als die Scheibe dick ist. So erübrigt sich ein Vlies, da keine Reibung stattfindet.

Um an die für den Schreib-/Lesekopf wichtige Aussparung zu kommen, muß man den Schieber zur Seite drücken, der über einen Federmechanismus die Öffnung verschließt, solange sich die Diskette nicht im Laufwerk befindet. Wird die Diskette in das Laufwerk geschoben, sorgt eine Einrichtung dafür, daß der Schieber die Öffnung freigibt und der Schreib-/Lesekopf Zugang zur Scheibe hat.

Statt einer Schreibschutzkerbe gibt es einen Plastikschalter oder -schieber, der durch seine Stellung dem Laufwerk die Information "Nur Lesen" (geschlossen) oder "Schreiben/Lesen" (offen) mitteilt. Wegen der stabilen Verpackung brauchen diese Disketten keine Schutztaschen. Man kann sie bequem in Jackentaschen transportieren, was mit den älteren und größeren Verwandten nicht ging.

Wir wollen nun die Verhaltensregeln besprechen, die einen langfristigen und vor allem sicheren Gebrauch der Disketten ermöglichen. Einige Regeln gelten nur für die 5¼"-Disketten, aber es schadet sicher nicht, sie im Bedarfsfalle auch bei den kleineren anzuwenden.

Regel 1: Vorsicht vor Magneten!

Da die Information auf einer Diskette magnetisch gespeichert ist, kann auch jedes beliebige magnetische Feld diese Informationen beeinflussen oder gar löschen. Darum sollte man genügend Abstand zu allen Geräten haben, die magnetische Felder erzeugen:

- Natürlich alle Magneten selber
- Netzgeräte
- Fernsehgeräte
- Computermonitore (!)
- Lautsprecherboxen
- Stromleitungen

Regel 2: Bitte nicht knicken!

Diese Regel gilt besonders für die "Schlappscheiben" im 5¼"-Format. Normalerweise halten Disketten leichte "Biegeversuche" aus. Allerdings sollte man sich hüten, Knicke in der Scheibe zu produzieren, denn dann ist es dem Schreib-/Lesekopf nicht mehr möglich, Daten zu erkennen.

Regel 3: Nebeneinander ist besser als übereinander!

Versuchen Sie nicht, Höhenstapelrekorde aufzustellen. Disketten sollten am besten nebeneinander hochkant in staubgeschützten Kästen aufbewahrt werden. Sie haben die Wahl zwischen teuren, hochglanzpolierten, durchsichtigen Diskettenboxen oder billigen, unscheinbaren Karteikästen. Das ist aber Geschmackssache.

Regel 4: Vorsicht vor Büroartikeln!

Sie sollten auf Disketten keine Schreibübungen mit harten Stiften (Kugelschreiber, Bleistift,...) machen. Durch die harten Schreibspitzen dieser Geräte kann die Scheibe beschädigt werden. Auch Büroklammern und Ähnliches sollten in gehörigem Abstand zu den empfindlichen Scheiben gehalten werden.

Regel 5: Disketten sind häuslich!

Es gibt genau zwei Orte, an denen eine Diskette sein darf: in der Diskettenbox und im Laufwerk. Es gibt keine Ausnahmen.

Regel 6: Disketten sind gerne gut angezogen!

Nach Gebrauch sollten Sie Disketten immer in ihre Schutztasche zurückstecken. Da schon Staub oder Haare zwischen Vlies und Scheibe den Betrieb beeinträchtigen oder den Schreib-/Lesekopf beschädigen können, sollte man schon deswegen Disketten nicht offen herumliegen lassen.

Regel 7: Disketten sind Abstinenzler!

Nicht nur Alkohol sollten Sie von Disketten fernhalten. Alle Arten von Flüssigkeiten, die so im normalen Haushalt vorkommen, sollten nicht auf demselben Tisch sein wie Disketten. Eine Ausnahme bildet reiner Alkohol. Aber davon gleich mehr. Strengstens verboten ist alles, was krümelt oder schmilzt, also Brötchen, Kuchen, Schokolade etc.

Regel 8: Disketten lieben gemäßigtes Klima!

Wenn Sie Wert darauf legen, daß die Disketten Ihre Informationen sicher "behalten", sollten Sie immer dafür sorgen, daß sie keinen starken Temperaturwechseln unterworfen werden. Gerade im Winter ist es nicht gut, Disketten zu transportieren. Aus der warmen Wohnung in die Kälte ist es ein kurzer Schritt, und das nehmen Disketten häufig übel. Wenn Sie doch im Winter Disketten transportieren müssen, basteln Sie sich eine Diskettentransportbox aus Styropor. Der zulässige Temperaturbereich für Disketten ist +10° C bis +50° C. Die Arbeitstemperatur sollte am besten zwischen +15° C und +25° C liegen.

Wenn Sie alle diese Regeln beachten, kann Ihren Daten eigentlich nichts mehr passieren!

Der nächste Abschnitt beschäftigt sich damit, wie man sich verhält, wenn doch mal ein Malheur passiert ist.

Trotzdem kann mal was schiefgehen

Bei intensivem Betrieb mit dem Computer läßt es sich gar nicht vermeiden, daß mal Disketten auf dem Arbeitstisch herumliegen. Selbst wenn man eine Festplatte sein eigen nennt, braucht man ab und zu mal Informationen von Diskette. Und da viel Arbeit Appetit macht, kommt es dann auch vor, daß Eßwaren auf dem Tisch stehen, die ja dort nichts zu suchen haben.

So ist es auch mir eines schönen Tages passiert, daß mir eine Tüte Milch umkippte und sich der Inhalt über eine Diskette ergoß, auf der ich wertvolle Daten stehen hatte. Gerade Milch verklebt die Diskette völlig, und so eine Diskette sollte man selbstverständlich nicht mehr in das Diskettenlaufwerk stecken. Dummerweise hatte ich davon aber keine Sicherheitskopie, und da es private Daten waren, gab es keine Möglichkeit, sie vollständig zu reproduzieren.

Trotzdem habe ich die Daten gerettet, und die Methode, die primär für 5¼"-Disketten gilt, will ich Ihnen nicht vorenthalten. Für 3½"-Disketten funktioniert die Methode aber auch.

Wie man verschmutzte Disketten retten kann

Es ist passiert, Ihre Diskette ist verklebt von Ihrem Frühstück auf dem Tisch. Bewahren Sie Ruhe, und besorgen Sie sich folgende Utensilien:

- Eine alte Diskettenhülle, aus der die Scheibe herausgenommen worden ist und die am oberen Rand offen ist.
- Viel weiches Küchenpapier (keine Papiertaschentücher, die sind zu hart!)
- Reinen Isopropylalkohol aus der Apotheke

- Eine scharfe Schere oder ein scharfes Messer
- Eine neue Diskette
- Ein Diskettenkopierprogramm

Legen Sie zuerst eine Arbeitsfläche mit sauberem Küchenpapier aus. Dann waschen Sie gründlich Ihre Hände. Danach schneiden Sie mit einer scharfen Klinge vorsichtig und ohne die Scheibe zu beschädigen den oberen Rand auf oder brechen den Bördelrand der Schutzhülle auf. Dem Schneiden ist dabei der Vorzug zu geben.

Holen Sie vorsichtig die Scheibe aus der Hülle. Fassen Sie die Scheibe aber nie direkt an, sondern benutzen Sie ein trockenes Blatt Küchenpapier als Schutz. Legen Sie den "Patienten" (die schmutzige Scheibe) auf das Küchenpapier, das Sie als Unterlage vorbereitet haben. Zunächst saugen Sie die noch übriggebliebene Flüssigkeit mit einem Blatt Küchenpapier auf. Bitte jetzt noch nicht wischen! Sorgen Sie dafür, daß Sie genügend Küchenpapier bereit haben, denn Sie sollten verschwenderisch damit umgehen.

Tränken Sie jetzt ein Blatt Küchenpapier mit dem Alkohol, säubern Sie einen kleinen Teil der Scheibe mit diesem Tuch. Fahren Sie solange fort, bis die Scheibe auf der einen Seite einigermaßen sauber ist.

Nehmen Sie danach wieder ein frisches, mit Alkohol getränktes Blatt und versuchen Sie, die "Putzstreifen" auf der Scheibe wegzuwischen. Gehen Sie bei allen Arbeitsgängen sehr behutsam vor: minimaler Druck und sehr langsam wischen. Ist die eine Seite nach Sichtkontrolle so sauber, daß auf der Oberfläche kein Rückstand mehr zu sehen ist, wiederholen Sie den Putzvorgang für die Rückseite.

Ist die Scheibe vollständig sauber, nehmen Sie sie vorsichtig mit Ihrer durch Küchenpapier geschützten Hand und führen Sie sie vorsichtig in die vorbereitete leere Diskettenhülle ein. Starten Sie jetzt Ihren Computer, und laden Sie ein Kopierprogramm. Ko-

pieren Sie die gesäuberte Diskette auf die neue Diskette. Das dürfte bei richtiger Anwendung des Verfahrens keine Schwierigkeiten machen.

Wenn die Daten sicher auf der anderen Diskette sind, sollten Sie die gesäuberte Scheibe wegwerfen. Benutzen Sie ab jetzt nur die neue Kopie, da auf der von uns geretteten Scheibe sicherlich noch Mikroreste des Schmutzes sind, und so etwas nimmt uns unser Schreib-/Lesekopf im Laufwerk übel. Legen Sie am besten zur Sicherheit gleich eine weitere Kopie an, damit Sie sich beim nächsten Mißgeschick die Arbeit sparen können.

Teflon

Der Fortschritt geht immer weiter, und so hat die Industrie auch für das gerade beschriebene Problem eine Lösung gefunden: Teflon. Diese brandneue Methode nutzt denselben Effekt, wie man ihn von der Bratpfanne her kennt. Durch eine Teflonschicht wird die bisher so empfindliche Diskettenoberfläche geschützt, und Krümel-, Milch- und andere Verschmutzungen können jetzt einfach abgewischt werden. Daten sind so besser geschützt und man erspart sich die Arbeit bei der oben beschriebenen Methode, wenn die Diskette mal was abbekommen hat. Auch das ist wieder ein weiterer Schritt zur Datensicherung.

13. Kopierprogramme, -methoden, -schutz

"Es gibt eine Theorie, die besagt, wenn jemals irgendwer genau rausfindet, wozu das Universum da ist und warum es da ist, dann verschwindet es auf der Stelle und wird durch etwas noch Bizarreres und Unbegreiflicheres ersetzt. Es gibt eine andere Theorie, nach der das schon passiert ist."

Douglas Adams[16]

Wir hören die spannende Geschichte vom Kampf der Softwarehäuser gegen findige Programmierer, die es immer wieder schaffen, Kopierschutzmechanismen zu knacken. Dann hören wir ein wenig über Kopierprinzipien, und zum Schluß werden wir ganz kurz über die rechtlichen Fragen aufgeklärt.

Am Anfang war das Programm, und es gab weder Verbote noch Gebote. Jeder war glücklich, daß es überhaupt funktionierte, und keiner machte sich Gedanken darüber, wie leicht gute Software zu vervielfältigen war. Da die "Töne" auf einer Diskette nur zwei Zustände annehmen, kann man, wenn man den Zustand erkannt hat, den Ton, statt ihn vom Original zu übernehmen, neu erzeugen und ihn so auf der Kopie abspeichern. Deswegen gibt es, wie oft eine Diskette auch kopiert wird, nie irgendwelche Verluste.

Irgendwann blühte dann der Softwaremarkt auf, und man begann, mit dieser nicht faßbaren, abstrakten Ware Geld zu verdienen. Wo Geld zu verdienen ist, gibt es auch Leute, die ebenfalls mitverdienen wollen, die aber abwarten, bis andere Leute die Arbeit gemacht haben. Als die Microcomputer den Markt eroberten, wurden naturgemäß auch Programme verkauft. Die waren aber, wegen der geringen Nachfrage, noch sehr teuer. Wie

16 Douglas Adams: Das Restaurant am Ende des Universums, Ullstein 1988

hilft man sich also als gewitzter Programmierer? Man kopiert eben und benutzt das Programm ohne den Segen des Herstellers.

Die Softwarehäuser kamen schnell auf den Trichter, in die Programme einfache Schutzmechanismen einzubauen, die die Programmierer erst mal überraschten. Der erste Trick war wohl, eine bestimmte Spur nach dem Formatieren wieder zu löschen. Einfache Kopierprogramme merkten, daß da nichts steht und brachen ihre Arbeit einfach ab. Nachdem das bekannt war, gab es auch schnell Gegenmittel. Also fuhren die Hersteller ein weiteres Geschütz auf. Man manipulierte einfach mit den Daten auf der Diskette. Zum Beispiel legte man die Directoryspur auf eine andere als vom Betriebsystem geforderte Spur. Das Dumme an dieser Methode ist, daß man dem Programm auch gleich ein geändertes DOS mitgeben muß, da das normale nicht mehr mit der Diskette klarkommt.

Oder man benutzt eine vom DOS sonst nicht erreichbare Spur. Wenn das DOS normalerweise nur die Spuren von 1 bis 35 benutzt, legt man eben notwendige Information auf Spur 36. Aber auch hier das Problem: ein anderes DOS ist notwendig. Außerdem können manche Laufwerke solche artistischen Übungen nicht mitmachen. Als auch solche Sachen geknackt waren, änderte man einfach das Format, die Codierung oder die Prüfsummenberechnung. Es ging Schlag auf Schlag. Genauso schnell, wie ein neuer Kopierschutz auf dem Markt war, tauchte auch schon ein dafür passendes Kopierprogramm auf.

Übrigens wurden Kopierprogramme nicht nur von eifrigen Hackern geschrieben. Nein, es gab eine Firma auf dem Markt für Apple II+ Software, die grob geschätzt ca. 90% der für diesen Rechner existenten Superkopierprogramme geschrieben hat. Der Clou an einigen von diesen Programmen war, daß diese vor dem Kopieren nachschauten, ob das zu kopierende Programm von der eigenen Firma war oder nicht. Wenn ja, wurde das Programm irreparabel beschädigt. Wenn es von einer fremden Firma war, durfte es kopiert werden. Und es waren hervorragende Kopierprogramme!

Tja, die Ideen schossen wie Pilze aus dem Boden: Sektornummern wurden geändert, so daß das DOS mit dem Zählen durcheinander kam. Sync-Marken wurden anders gesetzt, oder es wurden einfach mehr Sync-Marken benutzt. Eine der tollsten Ideen war, die verschiedenen Spuren miteinander zu synchronisieren. Das funktioniert so: Man setzt auf einer Spur (z.B. Spur 1) irgendwann auf. Dann wartet man auf den Beginn der Spur (erster Sektor). Man liest die Information, und am Ende der Spur wechselt man in definierter Zeit auf die nächste, so daß man ganz bestimmte Daten (Sektornummer, Adressdaten im Sektor) suchen kann. Sind die beiden Spuren gegeneinander verschoben worden, was beim Kopieren zwangsläufig passiert, läuft das Programm einfach nicht. Das DOS erwartet dann vergeblich nach dem Spurwechsel den sofortigen Anfang einer Spur, und die Programme, die so programmiert sind, daß sie nur auf synchronisierten Spuren laufen, verweigern dann ihre Funktion.

Der verrückteste Schildbürgerstreich gegen die "Softwarepiraterie" gelang den Herstellern mit der physischen Beschädigung einer Diskette. Mit einem feinen Laserstrahl wurde auf einer formatierten Diskette ein ganz bestimmter Sektor irreparabel beschädigt. Das Programm prüfte das, indem es auf diesen Sektor vor Programmstart Daten schrieb und danach wieder zu lesen versuchte. Kam eine Fehlermeldung, so war es noch das Original, die Daten waren "durch das Loch gefallen". Auf einer Kopie aber war dieser Sektor zwangsläufig heil. Also: Programmabsturz oder Programm löschen. Dummerweise machte diese "Lochmethode" auch die Tonköpfe der Laufwerke kaputt. Und so verschwand die Methode schnell wieder vom Markt.

Zwei Kopierprinzipien

Ich möchte hier zwei verschiedene Methoden kurz besprechen: Den Dateikopierer und den Nibblekopierer.

Der Dateikopierer ist kein besonders schlaues Programm. Für so ein Programm muß alles in bester Ordnung sein. Das DOS, die Programme, kein Kopierschutz. Es sind nützliche Hilfen im täg-

lichen Alltag, da mit Ihnen ein großer Teil aller Programme kopiert werden kann. Ein Dateikopierprogramm verlangt als Eingabe nur Dateinamen. Diese werden gesucht und wie unter dem Kapitel "Sequentielle Dateien" besprochen, gelesen und auf eine andere Diskette kopiert. Wenn man auf diese Weise Disketten kopiert, hat man zwar denselben Inhalt, aber die Directories können sehr unterschiedlich aussehen. Sie sind nur logisch gleich.

Nibblekopierer gehen anders an die Arbeit. Am liebsten haben sie ganze Diskettenseiten ohne eine Einschränkung durch Dateinamen. Sie versuchen meist gar nicht erst, das DOS oder die Codierung zu erkennen. Sie gehen nach der Methode vor: die Daten sind nach einem bestimmten System auf die Diskette geschrieben worden, also muß es auch ein System geben, nach dem man die Daten wieder runterbekommt. Denn laufen soll das Programm ja schon (wenigstens auf den Originaldisketten). Sie versuchen einfach, sich irgendwie in den "Datenrythmus" einzuklinken und holen die Bytes Nibble für Nibble vom Original runter. Ein zuverlässiger und bekannter Nibblekopierer ist z.B. "Metacopy".

Zur Erklärung: ein Nibble ist ein halbes Byte, also 4 Bit. Das Betriebssystem des Diskettenlaufwerks wandelt jedes zu schreibende Byte in solche Halbbytes (Nibble) um, verändert sie dann noch nach einem speziellen Schema und schreibt sie dann Bit für Bit auf die Diskette. Eine Kopie mit einem Nibblekopierer ist fast eine getreue Kopie des Originals. Nibblekopierer, die synchronisieren können, machen das Ebenbild perfekt. Da ist es gerade so, als legten Sie Original und Duplikat übereinander, schlügen einmal mit der flachen Hand auf den Stapel, und alle Daten fallen geradewegs auf die Kopie.

Wann braucht man was? Zur täglichen Arbeit braucht man wenigstens einen Dateikopierer. Er hilft uns, die wichtigen Sicherheitskopien herzustellen, mit denen wir unsere Daten vor Verlust schützen. Manche Programmpakete oder Spiele können Sie damit nicht kopieren. Dafür gibt es Nibblekopierer.

Wollen Sie kopiergeschützte Software kopieren, brauchen Sie insbesondere bei Spielen Nibblekopierer. Aber denken Sie daran, daß Sie Kopien auf keinen Fall weitergeben dürfen.

Juristisches

Und schon sind wir bei den rechtlichen Fragen. Zunächst einmal eine Richtigstellung: Programme kann man nicht kaufen!

Man schließt mit der Herstellerfirma einen Nutzungsvertrag, bei dem eine einmalige Gebühr (der "Preis") fällig wird. Bei größeren Programmen unterschreiben Sie einen Nutzungsvertrag, schicken eine Ausfertigung an den Hersteller, der Sie in seiner Kundenkartei als rechtmäßigen Benutzer führt. Erst jetzt sind Sie berechtigt, das Programm zu benutzen. Die Disketten oder das Handbuch sind meist versiegelt. Brechen Sie das Siegel auf (Beschädigung reicht), haben Sie die Verpflichtung, das Programm abzunehmen.

Das bedeutet, daß Sie kein Eigentum an einem Programm erwerben können. Es bleibt immer geistiges Eigentum des Herstellers, genau wie man auch den Roman nicht kaufen kann, sondern nur das mit Buchstaben bedruckte Buch, das den Roman in dem uns bekannten Alphabet codiert (Deutsch, Englisch,...) enthält. Genauso erwerben wir nur die Diskette mit dem Code des Programms. Das Programm selber können wir nicht kaufen. Mit dem Erwerb der Nutzungsrechte gehen wir die Verpflichtung ein, das Programm (je nach Lizenz) nur auf einem Computer oder sogar nur auf einer Diskette zu nutzen. Meist sind aber 2 oder 3 Sicherheitskopien zur Datensicherung erlaubt. Einige Programme jedoch erwarten in einem Laufwerk mindestens die Originaldiskette, die sich nicht kopieren läßt und ohne die nichts läuft. Verleihen dürfen Sie das Programm meist nur, wenn Sie es in derselben Zeit nicht nutzen, so wie Sie ein Buch, das Sie verliehen haben, ebenfalls nicht lesen können. Software zu "tauschen" ist rechtlich mindestens bedenklich. Software zu kopieren ist verboten und kopierte Software kommerziell zu

verwerten oder zu verkaufen kann schon kriminell genannt werden. Aber es gibt trotz allem zwei Dinge, die einen bei allem Ernst schon wieder lächeln lassen.

Erstens gibt es sehr billige Programme. "Preiswert und gut" ist noch immer die beste Werbung, und es gibt viele alte Kopierhasen, die plötzlich Programme gekauft haben, weil sie billig und qualitativ hochwertig waren und man sich dadurch viel Ärger ersparen konnte.

Das zweite ist sogenannte Public Domain Software. Das sind Programme, die von wirklich edlen Charakteren geschrieben wurden und kostenlos oder zum Selbstkostenpreis verteilt werden. Es sind hochwertige Programme, in die die Programmierer sehr viel Mühe und Schweiß gesteckt haben. Trotzdem stellen sie ihre Programme kostenlos zur Verfügung. Einen herzlichen Dank an alle PD-Programmierer an dieser Stelle! Doch Vorsicht vor verseuchter PD-Software! Gerade unter Public Domain Software sind viele Viren, Trojanische Pferde, usw. zu finden, weil solche Programme sich am schnellsten verbreiten. Über das Thema Virus gibt es in der Literatur so viel, daß ich hier wieder auf den Anhang verweise.

Trojanische Pferde sind eine ziemlich miese Methode, Daten von Anwendern zu vernichten. Sie sehen aus wie bekannte Programme und heißen auch so, sie funktionieren aber anders. Beispielsweise weisen Sie auf dem Bildschirm darauf hin, daß jetzt alle zerstörten Sektoren einer Diskette repariert werden und zerstören in Wirklichkeit alle heilen Sektoren. Vor allem im PC-Bereich tummeln sich viele schwarze Schafe, die solche Programme mit Eifer in Umlauf bringen.

14. Tips und Tricks zur Floppy

Was wäre ein Floppybuch ohne Tips und Tricks zur Floppy. Das Ziel dieses Kapitels ist es nicht, Ihnen seitenlange Maschinensprachelistings vorzulegen, die dann etwas revolutionäres bewirken, sondern Ihnen vielmehr kleine und nützliche Tips und Tricks in die Hand zu geben, die Sie sofort ausprobieren können und auch verstehen. Also, fangen wir an!

14.1 Erweitertes DOS

In dem ersten Teil dieses Kapitels möchte ich Ihnen einige nützliche Routinen und Tricks vorstellen, die den Befehlssatz des DOS sozusagen erweitern (sofern das von BASIC aus geht).

14.1.1 Prüfung nach vorhandenem Schreibschutz

Nehmen wir einmal an, Sie wollen ein Programm schreiben, in dessen Verlauf Daten auf die Diskette geschrieben werden. Ihr Programm funktioniert wunderbar, doch wenn Sie einmal vergessen haben, von Ihrer Datendiskette den Schreibschutz zu entfernen, gibt das Programm einen Fehler aus und bricht ab. Dies kann sehr ärgerlich sein, wenn es sich um wichtige Daten handelt. Es wäre also besser, wenn das Programm herausfinden könnte, ob die Diskette schreibgeschützt ist oder nicht.

Hier zeige ich Ihnen eine Softwarelösung, die herausfindet, ob die Diskette schreibgeschützt ist oder nicht:

```
10 OPEN 1,8,15
20 PRINT# 1,"M-R";CHR$(0);CHR$(28)
30 GET# 1,SCHUTZ$
40 SCHUTZ = ASC(SCHUTZ$ + CHR$(0)) AND 16
50 IF SCHUTZ <> 0 THEN 100
60 PRINT "DIESE DISKETTE IST SCHREIBGESCHÜTZT, BITTE ENT-"
70 PRINT "FERNEN SIE ERST DEN SCHREIBSCHUTZ!!!"
```

```
80 GET A$ : IF A$ = "" THEN 80
90 PRINT "OK!"
100 REM hier wird das Programm fortgeführt
```

Das Programm öffnet den Floppykanal und überprüft dann, ob die Diskette schreibgeschützt ist. Ist dies der Fall, so enthält die Variable SCHUTZ einen Wert ungleich 0 und der Benutzer wird aufgefordert, den Schreibschutz zu entfernen. Hat er dies getan, so soll er eine Taste drücken, und das Programm kann fortfahren. War die Diskette nicht schreibgeschützt, so verzweigt das Programm zu Zeile 100, wo dann das Hauptprogramm folgen soll. Diese Routine kann überall dort eingesetzt werden, wo in Erfahrung gebracht werden muß, ob die Diskette schreibgeschützt ist oder nicht.

14.1.2 Schließen aller Kanäle

Haben Sie in einem Programm einmal mehrere Kanäle geöffnet, weil Sie beispielsweise mit der Floppy, dem Drucker und der Datasette gleichzeitig arbeiten wollen, so ist es sehr mühsam, diese Kanäle am Ende des Programms 'per Hand', also durch den Befehl

```
CLOSE Kanal
```

einzeln zu schließen. Haben Sie nämlich mitunter einmal 7 Kanäle geöffnet, so ist dies doch recht umständlich. Ich möchte Ihnen in diesem Kapitel drei verschiedene Methoden zeigen, wie Sie dieses Problem elegant lösen können.

Die erste Methode erinnert stark an das Schließen der Kanäle per Hand. Hier werden die CLOSE-Befehle jedoch nicht alle einzeln eingegeben, sondern in einer FOR-NEXT-Scheife abgearbeitet:

```
10 FOR KANAL = 1 TO 15
20 CLOSE KANAL
30 NEXT KANAL
```

Hierbei werden in einer Schleife, die 15 mal durchlaufen wird - denn es gibt nur 15 Kanäle - jeweils ein Kanal geschlossen.

Die zweite Methode besteht darin, einfach eine Routine ab Adresse 65511 mittels

```
SYS65511
```

aufzurufen, die jedoch nicht die Kanäle schließt, sondern Sie lediglich vergißt. Dies bewirkt zwar das gleiche, ist jedoch nicht ganz 'sauber' und es kann evtl. zu späteren Störungen kommen.

Die dritte - und damit letzte - Methode, die ich Ihnen zeigen möchte, basiert darauf, daß der C64 sich 'merkt', welche Kanäle im Moment geöffnet sind. Außerdem ist in der Speicherzelle 152 die Anzahl der gerade geöffneten Kanäle gespeichert. Folgende Routine durchläuft eine FOR-NEXT-Schleife so oft, wie Kanäle geöffnet sind. Innerhalb dieser Schleife werden dann die einzelnen Kanäle geschlossen:

```
10 FOR KANAL = 1 TO PEEK (152)
20 CLOSE PEEK (601)
30 NEXT KANAL
```

14.1.3 Floppyreset

Genau wie beim Computer, läßt sich auch an der Floppy ein Reset ausführen. Dies ist z.B. dann sinnvoll, wenn ein Fehler auftrat und nun die Floppylampe unentwegt blinkt. Soll nun ein Floppyreset druchgeführt werden, so muß der Befehl "UJ" an die Floppy geschickt werden. Dies geschieht durch folgende 3 Zeilen:

```
10 OPEN1,8,15
20 PRINT# 1,"UJ"
30 CLOSE 1
```

In Zeile 10 wird der Floppykanal "geöffnet", in Zeile 20 dann der Befehl "UJ" an die Floppy geschickt, und Zeile 30 schließt wieder den Floppykanal. Leider springt der Computer nach

Beendigung des Programmes nicht zurück in die Eingabe-Warteschleife. Drücken Sie dann bitte < RUN/STOP - RESTORE>.

Diese drei Zeilen lassen sich auch zu einer zusammenfassen, was den Vorteil hat, daß man diese Zeile dann im Direktmodus eingeben kann. Die Befehlsfolge sieht dann folgendermaßen aus:

```
OPEN1,8,15 : PRINT# 1,"UJ" : CLOSE1
```

oder einfach

```
OPEN1,8,15,"UJ" : CLOSE1
```

Sollte dieser Trick einmal nicht seine Wirkung zeigen, so ist die Floppy "abgestürzt", und es hilft nur noch ein Aus- und Anschalten.

14.1.4 Auslesen des Fehlerkanals

Wer kennt die Situation nicht: Man möchte gerade nochmal eben eine Floppyoperation durchführen, doch das Knacken bleibt aus und stattdessen blinkt das Floppylämpchen. Nach Prüfung auf Schreib- oder andere Flüchtigkeitsfehler weiß man nicht mehr weiter.

Doch warum sollte man nicht einfach die Floppy selbst befragen, warum sie nicht gehorcht. Dazu muß der Fehlerkanal der Floppy geöffnet werden und bestimmte Parameter eingelesen werden. Folgendes Programm erledigt das für Sie:

```
10 OPEN 1,8,15
20 INPUT#1,NR,FEHLER$,SPUR,SEKT
30 PRINT NR","FEHLER","SPUR","SEKT
40 CLOSE 1
```

Zeile 10 öffnet - wie schon erwähnt - den Fehlerkanal der Floppy. In Zeile 20 werden nun die verschiedenen Werte eingelesen. Dabei geben folgende Variablen folgende Werte an:

NR	Fehlernumer
FEHLER$	Fehlermeldung
SPUR	Spur
SEKT	Sektor

Vielleicht werden Sie sich fragen, warum ich für den Sektor die Variable SEKT, jedoch nicht SEKTOR benutzt habe. Benutzen Sie die Variable SEKTOR, so bricht der C64 mit einem Syntax Error ab, da im Wort SEKTOR das BASIC-Befehlswort TO versteckt ist, was nicht erlaubt ist.

Jede Fehlermeldung hat eine bestimmte Nummer, die hier in der Variablen NR abgespeichert wird. Sie können nun im Floppyhandbuch nachschlagen, was diese Nummer bedeutet. In der Variablen FEHLER$ steht die Fehlermeldung in Kurzform. Die Variablen SPUR und SEKTOR geben an, auf welcher Spur bzw. auf welchem Sektor der Fehler auftrat. Auf diese Art und Weise lassen sich Fehler leicht bestimmen und somit auch beheben.

14.1.5 Auslesen des Fehlerkanals im Direktmodus

Haben Sie gerade ein Programm im Speicher, welches Sie nicht abändern können, und tritt nun ein Floppyfehler auf, obwohl Sie nicht wissen warum, so muß der Fehlerkanal im Direktmodus ausgelesen werden.

Geben Sie dazu folgendes ein:

```
OPEN 1,8,15
FORA=0TO39:POKE781,1:SYS65478:SYS65487:PRINTCHR$(PEEK(780));:SYS65
484:IF ST=0 THEN NEXTA
```

Nachdem Sie die RETURN-Taste gedrückt haben, erscheint die Fehlermeldung in folgender Form:

```
FEHLERNUMMER,FEHLER,TRACK,SEKTOR
```

Wollten Sie z.B. etwas laden, obwohl sich keine Diskette im Laufwerk befand, so erscheint:

```
XY,DRIVE NOT READY,00,00
```

Dieser Trick erfordert einige Erklärungen:

Es ist von BASIC aus möglich, Maschinenspracheroutinen aufzurufen. Dies geschieht mit dem SYS-Befehl. Manche Routinen brauchen für ihre Arbeit bestimmte Werte, die in Registern abgelegt werden. Es gibt insgesamt vier Register:

Register	Adresse
Akkumulator	780
X-Register	781
Y-Register	782
Statusregister	783

Bei diesem Einzeiler wird eine Schleife höchstens vierzig mal durchlaufen, da eine Fehlermeldung höchstens vierzig Zeichen lang ist. In der Schleife erhält erst das X-Register den Wert eins, welcher die Filenummer angibt. Daraufhin wird eine Routine namens CHKIN, die das aktuelle Eingabegerät (Tastatur) auf das im X-Register angegebene Gerät umschaltet. In diesen Falle ist das die Floppy. Daraufhin wird ein Zeichen eingelesen, indem die Routine BASIN aufgerufen wird. Dieses Zeichen wird im Akkumulator (Adresse 780) abgelegt. Der Print-Befehl gibt dieses Zeichen auf den Bildschirm aus. Anschließend wird wieder die Tastatur als Eingabegerät und der Bildschirm als Ausgabegerät eingestellt. Danach wird das Status-Byte abgefragt. Ist es Null, so wird die Schleife noch einmal durchlaufen. Hat die Variable ST den Wert 64, so zeigt dies das Ende der Fehlermeldung an.

Vergessen Sie nicht, nach diesen Befehlen den Kanal durch

```
CLOSE 1
```

wieder zu schließen, denn sonst würde beim eventuellen zweiten Öffnen ein

```
FILE OPEN ERROR
```

ausgegeben.

14.1.6 Unscratch

Wußten Sie schon, daß ein Programm, welches nach Eingabe von

```
OPEN 1,8,15,"S:PRO.NAME"
CLOSE 1
```

gelöscht wurde, noch gerettet werden kann? Geben Sie dazu einfach folgenden Befehl ein:

```
LOAD "*",8
```

Hier zeigt sich wieder der Vorteil von Wildcard-Zeichen, denn bei der Eingabe von

```
LOAD "PRG.NAME",8
```

hätte der C64 einen

```
FILE NOT FOUND ERROR
```

ausgegeben. Ein Wildcard-Zeichen ist ein Zeichen, welches ein Ersatz für andere Zeichen ist. Das Sternchen steht stellvertretend für das zuletzt geladene Programm. Deshalb wird das eben gelöschte File noch einmal geladen.

Das Laden von gelöschten Programmen setzt jedoch zweierlei vorraus. Erstens darf nach dem ungewollten Löschen kein an-

deres Programm geladen werden und zweitens darf nach dem Löschen kein Floppy-Reset (siehe Kapitel 14.1.3) durchgeführt werden.

14.1.7 1328 Blocks Free

Sind Sie Besitzer einer Floppy 1571 und eines C64/C128? Haben Sie sich schon immmer geärgert, daß Sie - obwohl Ihr Laufwerk zwei Schreib-/Leseköpfe hat - Sie immer nur Ihre Disketten einseitig verwenden können? Dann sollten Sie sich nun dieses Kapitel durchlesen, denn danach ist Schluß mit einseitigen Disketten.

Wie Sie sich nun schon denken können, ist es möglich, mit der 1571 im C64 Modus die Diskette beidseitig zu formatieren und auch zu benutzen. Nehmen Sie sich bitte eine neue Diskette oder eine, die unbenötigte Daten enthält, denn wir wollen diese nun zweiseitig formatieren. Geben Sie dazu folgendes ein:

```
OPEN 1,8,15,"U0>M1"
PRINT# 1,"N:TIPS&TRICKS,AP"
CLOSE 1
```

Sollten Sie keine Diskette im Laufwerk haben, so gibt die Floppy nach dem ersten Befehl einen Fehler aus, den Sie aber ignorieren können. Spätestens vor der Eingabe des zweiten Befehles sollten Sie die zu formatierende Diskette ins Diskettenlaufwerk legen.

Das Laufwerk läuft nun an. Nach einiger Zeit hat es sich wieder beruhigt. Wenn Sie nun einmal das Directory laden, so wird Ihnen die Meldung

```
1328 BLOCKS FREE
```

entgegenstrahlen. Sie haben nun doppelt so viel Speicherplatz wie vorher zur Verfügung.

Der Trick ist eigentlich recht einfach: Mit dem ersten Befehl wird der Modus der 1571 umgeschaltet. Die 1571 hat nämlich einen C64 und einen C128 Modus. Arbeiten Sie im C64 Modus, so schaltet auch die Floppy automatisch in den C64 Modus. Man muß nun nur den C128 Modus der 1571 einschalten, und schon arbeitet das Diskettenlaufwerk zweiseitig. Das Umschalten in den C128 Modus geschieht durch die erste Zeile. Danach wird die Diskette durch den NEW-Befehl ganz normal zweiseitig formatiert, und der dritte Befehl schließt schließlich den Kanal 1.

14.1.8 Formatierung in einer Sekunde

Haben Sie eine volle Diskette und brauchen Sie die Daten nicht mehr, so ist es sinnvoll, diese Diskette neu zu formatieren, denn das einzelne Löschen der Programme ist recht zeitaufwendig und kompliziert. Der Formatierungsvorgang dauert jedoch ca. eine Minute, was bei mehreren Diskette schon recht nervend sein kann. Doch diese Zeit läßt sich stark verkürzen. Legen Sie einmal eine zu löschende Diskette ins Laufwerk und geben Sie folgendes ein (Achtung: Die Daten werden gelöscht!!!):

```
OPEN 1,8,15,"N:TIPS&TRICKS"
CLOSE 1
```

Das Diskettenlaufwerk läuft kurz an, doch mehr passiert auch nicht. Geben Sie nun einmal

```
LOAD "$",8
LIST
```

ein, und Sie sehen, daß die Diskette nun leer ist. Sie haben wieder 664 Blöcke frei.

Vielleicht ist Ihnen bei der Eingabe der kleine - aber feine - Unterschied zwischen dem normalen Formatierungsbefehl und dem abgeänderten gar nicht aufgefallen. Es wurde lediglich die ID-Angabe weggelassen, was bewirkt, daß lediglich das Inhalts-

verzeichnis der Diskette, nicht aber die Diskette selbst gelöscht wird. Es werden also die Daten, die sich auf der Diskette befinden, nicht gelöscht, sondern nur vergessen.

Sollten Sie einmal geheime Daten auf Ihren Disketten haben, die Sie nun nicht mehr benötigen, so sollten Sie diesen Trick nicht anwenden, da man sich die Daten noch mit einem Diskettenmonitor angucken kann. Also: Im Zweifelsfalle immer den normalen Formatierungsbefehl benutzen!

14.1.9 Ändern der Floppyadresse

Der C64 hat die Möglichkeit, 8 Floppys zu verwalten. Diese besetzen die Kanäle 8-15. Von Grund auf hat jede Floppy jedoch die Nummer 8. Wollen Sie diese ändern, so wird Ihnen dieses Programm dabei helfen:

```
10 OPEN 1,8,15
20 PRINT# 1,"M-W"CHR$(119);CHR$(0);CHR$(2);CHR$(NEUE_NUMMER + 32);
   CHR$(NEUE_NUMMER + 64)
30 CLOSE 1
```

Sie müssen hier lediglich für die Variable NEUE_NUMMER die von Ihnen gewünschte Geräteadresse angeben, z.B. 9.

Wollen Sie eine Floppy mit einer anderen Gerätenummer als 8 'umpolen', so müssen Sie die Zeile 10 folgendermaßen ändern:

```
10 OPEN 1,NUMMER,15
```

wobei Sie für Nummer die alte Geräteadresse angeben müssen. Dieser Trick hat besonders dann Sinn, wenn Sie mehrere Floppys besitzen.

Ein Nachteil läßt sich jedoch nicht abstreiten. Sobald Sie den Rechner ausschalten, werden alle Nummern wieder auf 8 gesetzt. Sie müssen also vor jedem neuen Arbeitsgang wieder die Nummern neu setzen. Dieser Nachteil kann mit Durchtrennen zweier

Drahtbrücken in der Floppy behoben werden. Wie das funktioniert, können Sie in einschlägiger Literatur nachlesen.

14.2 Laden und Speichern

Dieser Teil dieses Kapitels befaßt sich mit dem Speichern und dem Laden, denn dieses Thema kann sehr variantenreich sein.

14.2.1 Abspeichern eines Programmes als SEQ-File

Speichern Sie ein Programm mittels

```
SAVE "PRG.NAME",8
```

ab, so bekommt dieses File im Directory die Endung PRG.

Wollen Sie jedoch, daß Ihr Programm eine andere Endung bekommt, so ist es auch möglich, es als sequentielles File oder als User-File abzuspeichern. Dazu müssen Sie beim abspeichern folgendes eingeben:

```
SAVE "PRG.NAME,ENDUNG,STATUS",8
```

Hier muß nun die Variable Endung durch die Abkürzung für den Filetyp ersetzt werden. Hier nun eine Auflistung der möglichen Filetypen und deren Abkürzungen:

sequentielle Datei	- SEQ
User-Datei	- USR

Die Variable STATUS gibt an, ob das Programm gelesen oder geschrieben werden soll. Soll auf Diskette geschrieben werden, so muß an dessen Stelle ein W für WRITE stehen, soll gelesen werden, so muß an dessen Stelle ein R für READ stehen. Wollen Sie also ein Programm namens Programm1 als sequentielles File abspeichern, so müssen Sie folgendes eingeben:

```
SAVE "PROGRAMM1,S,W",8
```

Noch einmal zur Verdeutlichung: S gibt an, daß das Programm als sequentielles File abgespeichert wird, das W gibt an, daß das File geladen wird. Wollen Sie es nun wieder laden, so geben Sie bitte folgendes ein:

```
LOAD "PROGRAMM1",8
```

Sie werden Pech haben. Der Computer gibt einen

```
FILE NOT FOUND ERROR
```

aus, denn er sucht ja nach einem PRG-File, welches den Namen Programml hat. Dies existiert jedoch nicht, denn wir haben das Programm ja als sequentielles File abgespeichert. Es muß also auch wieder als sequentielles File geladen werden. Geben Sie deshalb bitte

```
LOAD "PROGRAMM1,S,R",8
```

ein, und das Programm wird geladen. Auch hier noch einmal die kurze Erklärung: Das S gibt an, daß ein sequentielles File geladen werden soll und das R gibt an, daß dieses File gelesen werden soll.

Wollen Sie ein Programm als User-File abspeichern, so geben Sie bitte

```
SAVE "PROGRAMM2,U,W",8
```

ein. Im Directory erscheint daraufhin

```
"PROGRAMM2"          USR
```

Wollen Sie es wieder laden, so müssen Sie

```
LOAD "PROGRAMM2,U,R",8
```

eingeben.

Nun, was hat das ganze für einen Sinn? Wollen Sie ein Programm ohne großen Umstand vor der Benutzung unbefugter schützen, so ist dies eine recht sichere Methode, vorausgesetzt der Trick ist dem Unbefugten Benutzer unbekannt. Denn wer vermutet schon hinter einem sequentiellen File ein BASIC-Programm? Und selbst wenn er es versucht normal zu laden, wird er durch einen

```
FILE NOT FOUND ERROR
```

darauf aufmerksam gemacht, daß das Programm im Grunde nicht existiert.

14.2.2 Abfrage eines Kennwortes

Wußten Sie schon, daß es bei der Floppy möglich ist, Programme nur dann starten zu lassen, wenn vorher beim Programmname eine bestimmte Kennung angegeben wurde? Im Gegensatz zur Datasette gibt es jedoch keinen 'Floppypuffer', in dem Daten zwischengespeichert werden. Es gibt jedoch einen Zeiger bei Addresse 187/188, der auf den zuletzt genutzten Programmnamen bei Floppy-Operationen zeigt. Sie können sich dies zu Nutzen machen und diesen Namen durch folgendes Programm abfragen:

```
10 A = PEEK (187) + PEEK (188)*256
20 FOR B = A TO 40959
30 C = PEEK (B)
40 NAME$ = NAME$ + CHR$(C)
50 NEXT B
60 PRINT NAME$
```

Das Prinzip ist recht einfach. Der Zeiger zeigt auf den Namen, der dann in einer Schleife Zeichen für Zeichen eingelesen und ausgegeben wird.

Das Prinzip einer Abfrage nach einer Kennung ist folgendes: Der Benutzer muß beim Laden an das Programm noch eine bestimmte Kennung anhängen, welche dann durch das Programm

abgefragt wird. Die einzige Bedingung ist, daß das Programm unter einem 16 Buchstaben langen Namen abgespeichert wird.

Hier nun die Abfrage:

```
10 ADR = PEEK (187) + PEEK (188)*256
20 FOR A = ADR+16 TO ADR+18
30 READ A$
40 ZEICHEN = ASC (A$)
50 IF ZEICHEN = PEEK (A) THEN 70
60 SYS64738
70 NEXT A
80 DATA O,K,!
```

Speichern Sie dieses Programm unbedingt unter einem Namen von 16 Zeichen Länge ab, und laden Sie es dann ohne Kennwort. Es kommt zu einen Reset. Laden Sie nun das Programm noch einmal und hängen Sie die Endung OK! an den Namen. Daraufhin arbeitet das Programm nach dem Laden normal. Kombiniert man diesen Trick mit einen Listschutz, so ist es für Unbefugte nicht einfach, dieses Programm zu starten.

14.2.3 Automatisches Laden und Starten von Programmen

Haben Sie bis jetzt auch immer Ihre Programme mit

```
LOAD "PRG.NAME",8
```

geladen und anschließen mittels

```
RUN
```

gestartet? Daß dies jedoch auch einfacher geht, zeigt Ihnen dieser Trick:

Bekanntlich kann ein Programm von der Datasette durch Drücken der <SHIFT/RUN-STOP> -Tasten geladen und automatisch gestartet werden. Solch eine Tastenkombination gibt es leider nicht für die Floppy. Geben Sie jedoch einmal folgende Befehlsreihe ein:

```
LOAD "PRG.NAME",8:
```

Drücken Sie jedoch noch nicht <RETURN>! Wenn sich der Cursor hinter dem Doppelpunkt befindet, so Drücken Sie nun die Tastenkombination <SHIFT/RUN-STOP>. Nun wird das Programm geladen und direkt danach wird es durch RUN automatisch gestartet.

Wie funktioniert dieser Trick? Nun, der LOAD-Befehl hat die Eigenschaft, eventuelle Befehle, die nach dem Doppelpunkt stehen, zu ignorieren. Das Drücken der Tasten <SHIFT/RUN-STOP> bewirkt nichts anderes als die Eingabe von

```
LOAD
RUN
```

Da der LOAD-Befehl nun den Befehl, der nach dem Doppelpunkt kommt, ignoriert, wird der 2. LOAD-Befehl einfach ignoriert. Danach folgt jedoch noch der Befehl

```
RUN
```

wodurch das Programm letztendlich noch gestartet wird.

Sie sehen: Dies ist wieder einer der recht kurzen und simplen Tricks, auf die man jedoch erst einmal kommen muß. Gewöhnen Sie es sich jedoch an, Ihre Programme immer auf diese Art zu Laden, so können Sie sich selbst eine Menge an Arbeit ersparen.

14.2.4 Ermitteln der Gerätenummer

Jedes Peripheriegerät besitzt eine Kennungsnummer, die sogenannte Gerätenummer. Diese Nummer ist dazu da, um z.B. den Drucker von der Floppy zu unterscheiden. Man kann sich ausmalen, welches Chaos entstünde, wenn bei einer Eingabe von SAVE der Computer z.B. den Drucker ansprechen würde.

Schreiben Sie ein Programm, welches mit verschiedenen Geräten (z.B. Floppy oder Datasette) arbeiten soll, so sollte das Programm

ermitteln können, was der Benutzer für Geräte angeschlossen hat. Dies wäre natürlich auch durch eine Abfrage im Programm an den Benutzer möglich, doch ist dies weniger elegant. Die Adresse 186 enthält die Gerätenummer des zuletzt benutzten Gerätes. Durch

```
PRINT PEEK (186)
```

können Sie diese erfahren. So könnten zum Beispiel die Befehle zum Nachladen eines Programmteiles folgendermaßen lauten:

```
10 OPEN 1,8,15
20 NUMMER = PEEK (186)
30 LOAD "PRG.",NUMMER
```

Ich habe hier die Zeile 10 eingefügt, damit auch gewährleistet ist, daß sich der nachfolgende Befehl auch auf die Floppy bezieht. Wenn Sie das Programm speichern, ist das nicht mehr nötig, denn dann ist gewährleistet, daß in der Speicherstelle 186 der Wert 8 steht. Würde Zeile 10 wegfallen und hätten Sie vorher noch nicht mit der Floppy gearbeitet, so ergäbe dies einen Fehler. Arbeiten Sie mit der Datasette, so müssen Sie die Zeile 10 entsprechend abändern. Auch das Öffnen eines Floppykanals - um z.B. eine sequentielle Datei zu öffnen - ist so möglich. Das Prinzip ist das gleiche:

```
10 OPEN 1,8,15
20 NUMMER = PEEK (186)
30 OPEN 1,NUMMER,15
```

Dementsprechend ist es möglich, daß ein Dateiverwaltungsprogramm sowohl mit der Floppy als auch mit der Datasette zusammenarbeitet. Wie oben schon einmal erwähnt, gibt diese Speicherstelle nur über das zuletzt benutzte Gerät Auskunft, jedoch nicht über alle angeschlossenen Geräte. Es ist also hier Vorsicht geboten, denn sonst kann es passieren, daß Sie ein Programm von Diskette nachladen wollen, vorher aber den Drucker benutzt haben, wodurch nun das Programm versuchen würde, etwas vom Drucker zu laden, was natürlich zu einem Fehler führen würde!

Die sicherste Methode ist, direkt am Programmanfang diese Nummer abzufragen. Das hat den Vorteil, daß diese dann nicht mehr verändert wird.

14.2.5 Vertauschen von SAVE und LOAD

Ein kleiner Kopierschutz läßt sich schon durch ein paar POKEs verwirklichen! Für viele Befehle oder Routinen gibt es Zeiger, welche auf die Anfangsadressen der Routinen zeigen. Auch für den Befehl SAVE und den Befehl LOAD gibt es solche Zeiger. Der Zeiger für SAVE befindet sich bei Adresse 818/819, der LOAD-Zeiger befindet sich bei 816/817. Es lassen sich diese beiden Zeiger nun vertauschen, so daß bei Eingabe von SAVE "Prg.name",8 nicht wie gewöhnlich gespeichert, sondern geVERIFYd wird. Auch läßt sich der LOAD-Zeiger so verändern, daß statt geladen gespeichert wird. Dieses Austauschen der Zeiger übernimmt folgendes Programm:

```
10 LL = PEEK (816)
20 LH = PEEK (817)
30 POKE 816,PEEK (818)
40 POKE 817,PEEK (819)
50 POKE 818,LL
60 POKE 819,LH
```

Die Funktionsweise des Programmes ist recht simpel. In Zeile 10 und 20 wird der Zeiger des SAVE-Befehls zwischengespeichert. Dann erhält der SAVE-Zeiger die Werte des LOAD-Zeigers, und der LOAD-Zeiger die vorher zwischengespeicherten Werte des SAVE-Zeigers. Sie können diese POKEs auch im Direktmodus eingeben. Die 'normalen' Werte des SAVE-Zeigers sind 165/244 ($F5ED) und die des LOAD-Zeigers sind 237/245 ($F4A5). Richtig zur Verzweiflung bringen kann man jemanden, wenn man den LOAD-Zeiger so läßt wie er ist, und den SAVE-Zeiger einfach auf den LOAD-Befehl zeigen läßt. Nun läßt sich überhaupt nichts mehr abspeichern.

Es lassen sich jedoch nicht nur diese beiden Zeiger untereinander vertauschen; man kann sie auch auf ganz andere Routinen

oder Befehle zeigen lassen. Folgende POKEs bewirken, daß der SAVE-Zeiger auf die Adresse 64738 zeigt, wodurch bei jedem Aufruf von SAVE ein RESET ausgeführt wird:

```
POKE 818,226
POKE 819,252
```

Wer es nicht ganz so radikal mag, (wer anderen eine Grube gräbt, fällt selbst hinein!) kann den SAVE-Zeiger ja einfach auf ein RTS zeigen lassen. RTS ist ein Maschinensprachebefehl und bewirkt nichts anderes als einen Rücksprung zu BASIC, was sich durch eine müde READY-Meldung äußert.

Hier nun die beiden Befehle:

```
POKE 818,26
POKE 819,167
```

Alles hier genannte läßt sich natürlich auch auf den LOAD-Zeiger anwenden!

14.2.6 Direktes Abspeichern von Maschinenprogrammen

Haben Sie für Ihre Maschinenroutinen auch immer BASIC-Loader geschrieben, der das Maschinenprogramm dann durch einen SYS-Befehl aufgerufen hat? Diese Methode ist nicht nur zeitaufwendiger, sondern verbraucht auch mehr Speicherplatz. Außerdem wird auf diese Weise immer das sich im Speicher befindliche BASIC-Programm zerstört.

Eine einfachere Methode ist es, den Speicherbereich direkt abzuspeichern. Dazu kurz einige Grundlagen zum SAVE-Befehl:

Der SAVE-Befehl ließt die Werte für den Programmanfang und das Programmende aus den Speicherstellen 43-46. Danach speichert er den so erhaltenen BASIC-Bereich ab. Wollen Sie nun einen beliebigen Speicherbereich abspeichern, so müssen Sie lediglich die Werte der Adressen 43-46 so abändern, daß der Zei-

ger in den Adressen 43/44 auf den Anfang des abzuspeichernden Bereichs zeigt und der Zeiger in den Adressen 45/46 auf das Ende des abzuspeichernden Bereiches.

Nehmen wir einmal an, Sie wollen den Speicherbereich von 4000-5000 abspeichern. Dies geschieht durch die Befehle

```
POKE 43,160
POKE 44,15
POKE 45,136
POKE 46,19
CLR
SAVE "PRG.NAME",8,1
```

Die eins hinter dem SAVE-Befehl bedeutet, daß das Programm absolut abgespeichert wird, das heißt, daß es beim Laden direkt an die richtige Adresse geladen wird. Ersetzen Sie beim SAVE-Befehl die acht durch eine eins, so funktioniert dieser Trick auch mit der Datasette.

Möchten Sie nach dem Abspeichern noch weiter in BASIC programmieren, so müssen Sie die Zeiger wieder richtig setzen. Dies geschieht durch folgende Befehle:

```
POKE 43,1
POKE 44,8
POKE 45,3
POKE 46,8
```

Durch diesen Trick lassen sich nicht nur Maschinenprogramme abspeichern; es ist auch möglich, verschiedene Bildschirme zu verwalten und diese auch nachzuladen. So läßt sich ein Bildschirm, der z.B. ab 15360 liegt, durch folgende Befehle abspeichern:

```
POKE 43,0
POKE 44,60
POKE 45,0
POKE 46,64
CLR
SAVE "BILDSCHIRM_2",8,1
```

Hier sei wieder angemerkt, daß dies nur bei neueren Betriebssystemversionen funktioniert.

Auch der Zeichensatz läßt sich so abspeichern, wodurch Ihre Programme um einiges flexibler werden, denn welche Programme arbeiten schon mit verschiedenen Zeichensätzen?

Hier nun noch ein zusätzlicher Trick zur Datasette:

Angenommen, Sie haben einen Bildschirm ab 15360 abgespeichert, Sie wollen ihn jedoch nun an einer anderen Stelle haben. Auch dies ist kein Problem für Datasettenbesitzer:

Geben Sie einfach den Befehl

```
SYS 63276
```

ein, und es erscheint die Meldung

```
PRESS PLAY ON TAPE
```

die Sie nun auch befolgen sollten. daraufhin wird jedoch nicht das ganze Programm, sondern nur der Tape-Header geladen. Im Tape-Header stehen die Werte, von wo bis wo das Programm im Speicher stehen soll. Sie können diese Werte durch folgende POKEs abändern:

```
POKE 829,ANFANG WERT1
POKE 830,ANFANG WERT2
POKE 831,ENDE WERT1 + 2
POKE 832,ENDE WERT2
```

Setzen wir einmal den Bildschirm an Adresse 8192-9191. Dazu müssen Sie folgende POKEs angeben:

```
POKE 829,0
POKE 830,32
POKE 831,0
POKE 832,34
```

Nun müssen Sie nur noch den Ladevorgang durch

```
SYS 62849
```

fortsetzen, und anschließend den Befehl

```
NEW
```

eingeben. Nun beginnt Ihr Bildschirm ab der Speicherstelle 8192.

14.3 Tips und Tricks zum Directory

Das Directory - immer da, doch kaum bemerkt. Was sich alles damit anfangen läßt, zeigt Ihnen dieses Kapitel.

14.3.1 Directory ohne Programmverlust

Wer kennt das nicht: Man hat gerade programmiert, hat das Programm noch nicht gespeichert. Doch ausgerechnet jetzt muß man wissen, ob ein bestimmtes File auf der Diskette ist. Man stieße hier auf das Problem, daß beim Laden des Directorys das BASIC-Programm im Speicher leider gelöscht würde. Man hätte nun natürlich die Möglichkeit, das Programm erst abzuspeichern, dann das Directory der Diskette einzuladen und anschließend wieder das Programm zu laden. Doch diese Methode ist recht umständlich. Einfacher geht dies, indem Sie

```
POKE 43,PEEK (45)
POKE 44,PEEK (46)+1
```

eingeben. Nun können Sie das Directory laden, anzeigen lassen und auch ausdrucken. Durch die Eingabe von

```
POKE 43,1
POKE 44,8
```

steht Ihnen wieder das BASIC-Programm zur Verfügung. Dieser Trick funktioniert ganz einfach. Da das Directory immer ab der

Adresse eingelesen wird, auf die der BASIC-Anfangzeiger zeigt, muß dieser nur so abgeändert werden, daß das Directory nicht das Programm im Speicher überschreibt. Dies geschieht durch die beiden Befehle

```
POKE 43,PEEK (45)
POKE 44,PEEK (46)+1
```

welche den BASIC-Anfang hinter das Programmende setzen. Wollen Sie nun wieder mit Ihrem Programm arbeiten, so müssen Sie den BASIC-Anfang wieder 'runtersetzen', und zwar auf die ursprünglichen Werte.

Da es nichts gibt, was nur Vorteile hat, müssen wir auch durch diesen Trick einen Nachteil hinnehmen: Da das Programm an das Ende des BASIC-Programmes gesetzt wird, werden die Variablen, die sich normalerweise hinter dem BASIC-Programm befinden, gelöscht. Dies können Sie jedoch umgehen, indem Sie den BASIC-Anfang hinter die Variablen legen. Dies geschieht durch

```
POKE 43,PEEK (49)
POKE 44,PEEK (50)
```

Auch dies kann durch die Befehle

```
POKE 43,1
POKE 44,8
```

wieder rückgängig gemacht werden.

14.3.2 Verstecktes Directory

Manche professionelle Programme haben ein geschütztes Directory, welches sich nicht auflisten läßt. Dies hat den Vorteil, daß sich die einzelnen Files nicht laden lassen, da man deren Namen nicht weiß. Dadurch ist es schwerer, Programme dieser Diskette zu kopieren.

Ich möchte Ihnen in diesem Kapitel eine Routine vorstellen, die das Directory vor unbefugten 'Einsehern' schützt. Um eine Diskette so zu schützen, geben Sie bitte dieses Programm ein, und starten Sie es. Am besten nehmen Sie erst einmal eine Diskette, die keine wichtigen Daten enthält, damit Sie die Funktion dieses Programm kennenlernen.

Hier nun das Listing:

```
10 OPEN 1,8,3,"#"
20 PRINT# 1,CHR$(20);CHR$(20);CHR$(20);
30 PRINT# 1,CHR$(0);CHR$(0);CHR$(0)
40 OPEN 2,8,15,"B-P3,144"
50 PRINT# 2,"U2:3,0,18"
60 PRINT# 2,"I"
70 CLOSE 1
80 CLOSE 2
```

Während des Auflistens des Directorys wird jedesmal überprüft, ob drei Nullen eingelesen wurden. Ist dies der Fall, so wird das Auflisten unterbrochen. Genau nach diesem Prinzip arbeitet dieses Programm auch. Es schreibt an den Anfang des Directorys drei Nullen, was dann beim Listen sofort zum Abbruch führt, denn der Rechner liest zuerst drei Nullen ein - die Kennung für das Ende des Directorys.

Dieses Programm ist jedoch auch nicht ganz ungefährlich und es sollte nur mit Obacht angewendet werden, denn es wird nicht nur das Directory vor dem Auflisten geschützt, sondern es wird auch die Diskette vor dem Beschreiben geschützt. Sie sollten dieses Programm also nie auf Datendisketten oder sonstige Disketten, die oft beschrieben werden, anwenden. Die Aufhebung dieses Programmes ist nur durch ein Neuformatieren möglich, womit die Daten jedoch restlos verloren gehen.

14.3.3 Schützen eines Teiles des Directorys

Möchten Sie nur einen Teil des Directorys löschen, so ist das Programm aus Kapitel 14.3.2 ungeeignet, da immer das ganze

Directory geschützt wird. Außerdem kann so auf die Diskette nichts mehr geschrieben werden, was Vor-, aber auch Nachteile haben kann.

Es kann aber nach dem gleichen Prinzip auch nur ein Teil des Directorys geschützt werden, indem Sie einfach einen Programmnamen so umbenennen, daß er mit drei Nullbytes endet. Nehmen wir einmal an, Sie haben folgendes Directory:

```
0  "TIPS&TRICKS"       AP
10 "PROGRAMM1"         PRG
10 "PROGRAMM2"         PRG
10 "PROGRAMM3"         PRG
10 "PROGRAMM4"         PRG
10 "PROGRAMM5"         PRG

596 BLOCKS FREE.
```

Wenn Sie nun Ihr Directory ab PROGRAMM3 schützen wollen, so müssen Sie an den Namen PROGRAMM2 drei Nullbytes anhängen. Dies können wir mit dem RENAME-Befehl erreichen:

```
OPEN 1,8,15,"R:PROGRAMM2"+CHR$(0)+CHR$(0)+CHR$(0)+"=PROGRAMM2"
CLOSE 1
```

Wird nun das Directory eingelesen und anschließend aufgelistet, so wird nach der Anzeige von PROGRAMM2 aufgehört, denn der Rechner erkennt, daß hier drei Nullbytes folgen, was für ihn das Ende des Directorys bedeutet.

Nach Eingabe von

```
LOAD "$",8
LIST
```

erscheint folgende Ausgabe:

```
0  "TIPS&TRICKS"       AP
10 "PROGRAMM1"         PRG
10 "PROGRAMM2"         PRG

596 BYTES FREE
```

Dieser Trick hat einige Vorteile gegenüber dem Trick aus Kapitel 14.3.2: Erstens ist die Diskette nicht schreibgeschützt, was mitunter ganz nützlich ist, und dieser Trick ist wieder umzukehren, d.h. Sie können Ihr Directory wieder 'entschützen'.

Um wieder das ganze Directory einsehen zu können, müssen die drei Nullbytes wieder aus dem abgeänderten Programmnamen 'verschwinden'. Dies geschieht wieder durch den RENAME-Befehl:

```
OPEN 1,8,15,"R:PROGRAMM2=PROGRAMM2"+CHR$(0)+CHR$(0)+CHR$(0)
CLOSE 1
```

Dadurch werden die Nullbytes aus dem PROGRAMM2 wieder entfernt, wodurch das Directory weiter aufgelistet wird.

Nach Eingabe von

```
LOAD"$",8
LIST
```

erscheint wieder das ganze Directory:

```
0  "TIPS&TRICKS"       AP
10 "PROGRAMM1"         PRG
10 "PROGRAMM2"         PRG
10 "PROGRAMM3"         PRG
10 "PROGRAMM4"         PRG
10 "PROGRAMM5"         PRG

596 BLOCKS FREE.
```

14.3.4 Steuercodes im Directory

Haben Sie schon einmal versucht, im Directory Steuercodes unterzubringen? So komisch das auch klingen mag, es ist möglich!

Speichern Sie bitte einmal ein Programm folgendermaßen ab:

```
SAVE CHR$(147)+CHR$(18)+CHR$(158)+"PRG.NAME",8
```

Wenn Sie es nun laden, wird nach der Meldung

```
SEARCHING
```

erst der Bildschirm gelöscht, dann der REVERS-Mode angeschaltet und anschließend die Meldung

```
LOADING PRG.NAME
```

in reverser Schrift ausgegeben.

Wie geht das? Nun, hat die Floppy das Programm 'gefunden', so gibt sie eine Meldung und den Programmnamen aus. Da der Programmname aber mit Steuercodes beginnt, werden diese erst ausgeführt, bevor dann der eigentliche Name erscheint.

Es ist auch beim Abspeichern möglich, die Steuercodes direkt durch Drücken bestimmter Tasten zu erzeugen, was etwas Tipparbeit spart. Geben Sie z.B. folgenden Befehl ein:

```
SAVE "<SHIFT/HOME><CRTL/1>PRG.NAME",8
```

Die in den Klammern stehenden Ausdrücke stellen hierbei die zu drückenden Tastenkombinationen dar. Beim nächsten Laden wird darauf erst der Bildschirm gelöscht und dann die Schriftfarbe auf schwarz gewechselt.

Ich habe Ihnen hier einmal eine Tabelle der wichtigsten Steuercodes aufgeführt:

CHR$(5)	WEIß
CHR$(8)	COMMODORE-TASTE BLOCKIERT
CHR$(9)	COMMODORE-TASTE ENTRIEGELT
CHR$(14)	UMSCHALTUNG AUF KLEINSCHRIFT
CHR$(18)	REVERSE-MODE ON
CHR$(19)	HOME
CHR$(20)	DELETE
CHR$(28)	ROT
CHR$(30)	GRÜN
CHR$(31)	BLAU
CHR$(142)	UMSCHALTUNG AUF GROßSCHRIFT

CHR$(144)	SCHWARZ
CHR$(147)	BILDSCHIRM LÖSCHEN
CHR$(156)	PURPUR
CHR$(158)	GELB
CHR$(159)	CYAN

Durch den Character-Code 20 ist es möglich, das Anzeigen des Programmnamens zu unterbinden. Es müssen nur beim Abspeichern hinter dem Programmnamen so viele CHR$(20) angehängt werden, wie der Programmname lang ist.

14.3.5 Sondereinträge im Directory

Normalerweise zeigt das Directory an, welche Files sich auf der Diskette befinden. Es werden deren Namen in Anführungsstrichen angezeigt. Es besteht jedoch auch die Möglichkeit, die Namen durch Bemerkungen zu erweitern. So ist es z.B. möglich, daß sich ein Programm unter folgenden Namen im Directory befindet:

```
10 "PROGRAMM1" V1.0    PRG
```

Das Besondere hierbei ist jedoch, daß die Erweiterung nicht beim Laden mit eingegeben werden muß. Das Programm läßt sich ganz normal durch

```
LOAD "PROGRAMM1",8
```

laden. Solche Erweiterungen im Programmnamen können mitunter recht nützlich sein, so kann die Endung DAT z.B. angeben, daß das Programm nur mit der Datasette läuft, oder die Endung PRT zeigt an, daß man für die Benutzung des Programmes einen Drucker braucht. Weiß der Benutzer das schon vor dem Programmstart, so kann er eventuellen Programmabstürzen vorbeugen.

Auch können alle Programme mit einer Endung versehen werden, die die Art des Programmes kennzeichnet. Anwenderprogramme können so z.B. die Endung ANW haben, während Spiele

die Endung SPIEL haben. (Vorraussetzung dafür ist jedoch, daß der Programmname und die Endung nicht länger als 16 Buchstaben sind). Auch die Kennzeichnung der aktuellen Programmversion ist auf diese Weise gut möglich, wenn man gerade ein Programm über eine längere Zeit programmiert. Der Trick ist recht einfach. Geben Sie einmal folgende Befehle ein:

```
OPEN 1,8,15,"R:PROGRAMM1"+CHR$(160)+" T&T"+"=PROGRAMM1"
CLOSE 1
```

Vorausgesetzt, es befand sich schon ein Programm mit dem Namen PROGRAMM1 auf der Diskette, so wurde es nun umbenannt und nach der Eingabe von:

```
LOAD "$",8
```

erscheint daraufhin:

```
10 "PROGRAMM1" T&T      PRG
```

Geladen wird das Programm jedoch ganz normal mittels

```
LOAD "PRGRAMM1",8
```

Wie funktioniert dies jedoch? Wie Sie am obigen Beispiel erkennen können, wurde der Programmname einfach umbenannt, und zwar wurde zwischen dem eigentlichen Namen und der Endung der Character Code 160 eingefügt. CHR$(160) steht für das geschützte Leerzeichen, SHIFT/SPACE. Dies zeigt dem Computer an, daß der Filename hier zu Ende ist. Deshalb gibt der Computer hier auch die zweiten Anführungsstriche an.

Der Name ist jedoch in Wirklichkeit noch nicht zu Ende, denn er wurde ja so umbenannt, daß hinter dem Anführungszeichen noch etwas kommt, nämlich die Endung, welche zwar auch abgespeichert, jedoch nicht mehr als Teil des Programmnamens angesehen wird. Es muß jedoch eine Einschränkung gemacht werden. Es darf leider nicht das Komma in der Endung erscheinen, da die Floppy dies nicht akzeptiert. Wollen Sie diese Änderung einmal rückgängig machen, so müssen die das Programm

lediglich wieder umbenennen. Um also unserem PROGRAMM1 wieder einen 'normalen' Namen zu geben, müssen Sie

```
OPEN 1,8,15,"R:PROGRAMM1=PROGRAMM1"+CHR$(160)+" T&T"
CLOSE 1
```

eingeben.

Wollen Sie eine Endung umbenennen, weil Sie z.B. die Programmversion 1.1 verbessert haben und diese nun die Versionsnummer 1.3 erhalten soll, so sollten Sie folgende Befehle eingeben:

```
OPEN 1,8,15,"R:PROGRAMM"+CHR$(160)+" V1.1"+"=PROGRAMM"+CHR$(160)+" V1.3"
CLOSE 1
```

Durch diesen Trick ist es recht einfach, die Directories so zu kennzeichnen, daß man sofort weiß, um welche Programme es sich handelt.

14.3.6 Steuerzeichen im Directory

Genau wie bei Programmnamen, ist es auch im Directory möglich, Steuerzeichen einzubauen. Dazu brauchen Sie lediglich eine leere Diskette zu formatieren und vor dem Diskettennamen die Steuerzeichen einzugeben.

Legen Sie sich bitte eine unformatierte Diskette ins Laufwerk und geben Sie einmal folgendes ein:

```
OPEN1,8,15,"N:"+CHR$(13)+CHR$(147)+"TIPS&TRICKS,AP"
CLOSE 1
```

Nach Eingabe von

```
LOAD"$",8
LIST
```

wird nun erst der Bildschirm gelöscht, bevor das Diskenteninhaltsverzeichnis eingeladen wird. Dieser Trick funktioniert ähn-

lich wie der aus Kapitel 14.2.1. Vor dem eigentlichen Namen werden beim Formatieren erst die Steuerzeichen angegeben. Das CHR$(147) wird Ihnen nun vielleicht schon bekannt vorkommen: es löscht den Bildschirm. Warum muß jedoch vorher ein CHR$(13) eingegeben werden? CHR$(13) ist der Character-Code für das Drücken der RETURN-Taste. Es muß erst ein RETURN ausgeführt werden, weil - bevor der Diskettenname ausgegeben wird - erst noch eine Null und die Anführungsstriche ausgegeben werden. Würde nun direkt das CHR$(147) folgen, so befände sich der Computer noch im Hochkomma-Modus, in dem Steuerzeichen nicht ausgegeben werden, sondern deren grafische Symbole. Dies können Sie einfach ausprobieren, indem Sie

```
PRINT "
```

eingeben. Der Computer befindet sich nun im Hochkomma Modus. Drücken Sie nun <CLR/HOME>, so wird ein reverses Herz ausgegeben - der Bildschirm wird jedoch nicht gelöscht. Um dieses Problem zu umgehen, wird also vor dem Steuerzeichen noch das Drücken der RETURN-Taste 'simuliert', so daß wieder eine Umschaltung in den normalen Darstellungsmodus erfolgt. Das Steuerzeichen wird nun direkt ausgeführt, was ein Löschen des Bildschirms bedeutet. Danach wird der Name der Diskette und das Inhaltsverzeichnis direkt ausgegeben.

Sie können selbstverständlich auch andere Steuerzeichen anstelle des CHR$(147) eingeben. Es gelten grundsätzlich die gleichen Steuerzeichen wie die aus Kapitel 14.3.4. Um Ihnen das lästige Umblättern zu ersparen, habe ich hier noch einmal die Liste für Sie aufgeführt:

CHR$(5)	WEIß
CHR$(8)	COMMODORE-TASTE BLOCKIERT
CHR$(9)	COMMODORE-TASTE ENTRIEGELT
CHR$(14)	UMSCHALTUNG AUF KLEINSCHRIFT
CHR$(18)	REVERSE-MODE ON
CHR$(19)	HOME
CHR$(20)	DELETE
CHR$(28)	ROT
CHR$(30)	GRÜN

CHR$(31)	BLAU
CHR$(142)	UMSCHALTUNG AUF GROßSCHRIFT
CHR$(144)	SCHWARZ
CHR$(147)	BILDSCHIRM LÖSCHEN
CHR$(156)	PURPUR
CHR$(158)G	ELB
CHR$(159)C	YAN

Hier nun noch einige Beispiele für persönliche Inhaltsverzeichnisse (Bedenken Sie bitte, daß beim Formatieren eventuelle Daten, die sich auf der Diskette befinden, verloren gehen!)

```
OPEN 1,8,15,"N:"+CHR$(13)+CHR$(5)+CHR$(14)+"TIPS&TRICKS,AP"
CLOSE 1
```

Hier wird erst die Schriftfarbe auf weiß gewechselt und dann auf Kleinschrift umgeschaltet, bevor das Directory ausgegeben wird.

```
OPEN 1,8,15,"N:"+CHR$(20)+CHR$(20)+"TIPS&TRICKS,AP"
CLOSE 1
```

Die Null und die ersten Anführungsstriche werden gelöscht, bevor das Directory ausgegeben wird.

```
OPEN 1,8,15,"N:"+CHR$(13)+CHR$(156)+CHR$(8)+"TIPS&TRICKS,AP"
CLOSE 1
```

Die Schriftfarbe wechselt auf Purpur und das Umschalten zwischen Klein- und Großschrift wird blockiert, bevor das Directory ausgegeben wird.

Ich habe Ihnen hier nun einige Beispiele gezeigt, wie Sie Ihr Directory umgestalten können. Probieren Sie dies ruhig ein wenig aus, denn probieren geht über studieren!

14.4 Sonstiges zur Floppy

Hier noch zwei kleine Tricks, die in keines der drei vorherigen Kapitel paßte, weshalb sie jedoch nicht minder nützlich sind. Hier sind Sie!

14.4.1 Löschen des Komma-Files

Ihnen ist sicherlich schon einmal ein File namens "," im Directory aufgefallen, welches dadurch entstanden ist, daß ein Programm, welches mit der Diskette arbeitet, einen kleinen Fehler hatte. Nun ist es jedoch nicht schön, immer ein unerwünschtes File auf der Diskette zu haben.

Versucht man nun, dieses File mittels

```
OPEN 1,8,15,"S:,"
CLOSE 1
```

zu löschen, so wird man sein blaues Wunder erleben, denn das File verschwindet nicht. Kommt man nun auf die Idee, dieses File mittels

```
OPEN 1,8,15,"R:NEUER_NAME=,"
CLOSE 1
```

umzubenennen, so wird man leider auch hier enttäuscht.

Es gibt jedoch einen Weg, solch ein Komma-File zu entfernen, der mitunter jedoch recht aufwendig sein kann. Benennen Sie einfach alle Files, die einen Namen von nur einem Buchstaben Länge haben, um. Dies geschieht mit folgendem Befehl:

```
OPEN 1,8,15,"R:NEUER_NAME=ALTER_NAME"
CLOSE 1
```

Hier müssen Sie natürlich noch die Variablen NEUER_NAME und ALTER_NAME durch die entsprechenden Namen ersetzen. Haben Sie dies getan, so sollte sich nur noch ein File auf der

Diskette befinden, dessen Name nur einen Buchstaben Länge hat. Überzeugen Sie sich davon, denn sonst können wichtige Daten verloren gehen. Geben Sie nun folgende Befehle ein:

```
OPEN 1,8,15,"S:?"
CLOSE 1
```

Das File mit dem Namen "," ist nun verschwunden, denn das Fragezeichen ist auch ein Wildcard-Zeichen. Das Fragezeichen steht stellvertretend für jedes Programm, welches nur einen Ein-Zeichen langen Namen hat. Da es nur das Komma-File gab, wurde dies also gelöscht. Dies ist auch der Grund, warum vorher alle Programme, die auch nur einen Ein-Zeichen langen Namen hatten, umbenannt werden mußten, denn sonst wären Sie auch alle gelöscht worden.

Übrigens lassen sich durch das Sternchen als Wildcard-Zeichen und dem SCRATCH-Befehl alle Programme auf der Diskette löschen: (probieren Sie das jedoch jetzt nicht aus!)

```
OPEN 1,8,15,"S:*"
CLOSE 1
```

14.4.2 Verkürzen der Floppyzugriffszeit

Es ist möglich, die Zugriffszeit der Floppy zu verkürzen. Unter Floppyzugriffszeit versteht man die Zeit, die der Schreib-/Lesekopf braucht, um eine bestimmte Stelle auf der Diskette zu erreichen, um dann auf die Daten 'zugreifen' zu können.

Während sich der Schreib-/Lesekopf zu einer Stelle auf der Diskette bewegt, werden jedoch mehrere Interrupts ausgelöst, in denen einige verschiedene Zustände überprüft werden, die uns hier nicht weiter interessieren sollen. Da dies jedoch kostbare Zeit kostet und außerdem zu diesem Zeitpunkt unwichtig ist, ist es sinnvoll, diese Interrupts (deutsch: Unterbrechungen) zu unterbinden. Dies geschieht durch folgende Befehle:

```
OPEN 1,8,15
PRINT# 1,"M-W"+CHR$(7)+CHR$(28)+CHR$(1)+CHR$(15)
CLOSE 1
```

Wenn Sie nun einmal durch den Befehl

```
LOAD "$",8
```

das Directory der Diskette einladen, oder durch

```
LOAD "PRG.NAME",8
```

ein Programm laden, so werden Sie feststellen, daß die Meldung

```
LOADING $
```

bzw.

```
LOAD "PRG.NAME"
```

wesentlich schneller als vorher auf dem Bildschirm erscheint. Dieser Trick ist wieder ein gutes Beispiel dafür, wie effektiv doch ein einfacher Befehl sein kann.

Anhang

Anhang A Logik und Bits - Eine kleine Einführung

"Wenn sowohl die wonkende Störpazgrejade als auch die köngrige Lukrogrejade den Orbengraub am tölligen Quolk vertrieseln, dann bingelt entweder die pelkernde Hinposgrejade oder die sanzende Weldruffgrejade ihr pommiges Mafferleutzchen."

Ausschnitt aus einer Logikaufgabe

Es folgt ein kleiner Überblick über das manchmal gar nicht so logisch erscheinende Gebiet der Logik und wie man Bits mit Hilfe der Logik erkennen kann.

"Wenn Monatsende ist, dann bin ich pleite!" ist für mich eine wahre Aussage. Einem Rockefeller oder Oppenheimer entlockt diese Aussage nicht einmal ein müdes Lächeln. Für diese ist die Aussage eindeutig falsch. Für einen afrikanischen Nomaden entbehrt dieser Satz jeglicher Logik. In diesem Fall gibt es keine Auswertung.

Logik ist also nicht gleich Logik!

Für die Sprache BASIC brauchen wir die Aussagenlogik, ein Teilgebiet der Logik. Und so können wir formulieren:

Die Grundstruktur der (Aussagen-) Logik ist die Aussage (wie sinnvoll sie auch sein mag!). Ausgewertet wird ihr Wahrheitsgehalt. Es können nur zwei Werte auftreten:

TRUE (= wahr) und FALSE (= falsch).

Zum Beispiel ist "Der Hahn ist ein Vogel" eine Aussage mit der Auswertung TRUE. Jedenfalls, wenn Sie kein Installateur sind.

Als Installateur würden Sie wahrscheinlich behaupten: "Der Hahn ist kein Vogel" (sondern eine Waschbeckenarmatur). Dann ist der Wahrheitsgehalt der ersten Aussage FALSE. Die Auswertung der zweiten Aussage ist dagegen TRUE!

Für Aussagen im Allgemeinen, nicht nur in der formalen Logik, gilt folgender wichtiger Satz:

Aussagen hängen mit den Umständen zusammen, unter denen sie gemacht werden. Aussagen sind auch nur dann sinnvoll, wenn klar ist, unter welchen Umständen sie gemacht werden! Statt "Umstände" kann man auch "Zusammenhang" sagen.

Selbstverständlich kann man die Ergebnisse solcher Auswertungen in Variablen abspeichern. In Pascal heißt der entsprechende Typ Boolean mit den zwei möglichen Konstanten TRUE und FALSE.

Beispiel in Pascal:

```
.
.
VAR statement: boolean; (Deklaration der Variablen)
BEGIN
.
.
statement := true; (Zuweisung)
.
.
END.
```

BASIC kennt diesen Typ nicht. Also muß man sich mit den vordefinierten Typen begnügen. Man nimmt daher numerische Variablen (Typ INTEGER) und definiert sich die entsprechenden Wahrheitswerte:

TRUE (wahr) entspricht -1 in BASIC oder 1 (je nach INTERPRETER)

FALSE (falsch) entspricht 0 in BASIC

Also wollen wir obiges Beispiel in BASIC übertragen. Da in BASIC die Typen vorgeschrieben und vordeklariert sind, darf hier keine Deklaration stehen. Somit müssen wir uns auf die Zuweisung beschränken:

```
.
.
10 statement = -1
.
.
```

Es gibt auch Verknüpfungen (Rechenvorschriften) für Aussagen, die den Verknüpfungen aus der numerischen Mathematik sehr ähneln. Zum Beispiel entspricht der Operator NOT dem negativen Vorzeichen '-'. Genauso, wie ein Minuszeichen aus einer positiven Zahl eine negative Zahl macht, so negiert NOT eine Aussage.

Zum Beispiel können wir auf die oben erwähnte "richtige" Aussage den Operator NOT anwenden, so daß eine "falsche" Aussage entsteht:

"Der Hahn ist kein Vogel"

oder formal:

NOT ("Der Hahn ist ein Vogel")

Ebenso gibt es Entsprechungen für '+' und '*': OR und AND. AND und OR werden also auf jeweils zwei Aussagen angewandt, NOT nur auf eine (Beispiel.: A AND B, D OR E, NOT C). Was jetzt noch fehlt, sind die genauen Bedeutungen der Operatoren NOT, OR und AND:

Der gesamte Wahrheitsgehalt zweier Aussagen bei Anwendung von AND ist TRUE (wahr), wenn beide Aussagen wahr sind, sonst falsch.

OR ist wahr, wenn entweder die eine oder die andere oder beide Aussagen richtig sind.

NOT negiert eine Aussage (s. oben).

Warum das alles? Wie wir an obigem Beispiel gesehen haben, haben Aussagen nur scheinbar einen allgemeingültigen Wahrheitsgehalt. Ändern sich die Umstände, unter denen eine Aussage gemacht worden ist, so kann sich auch ihr Wahrheitsgehalt ändern. Die Lösung eines Problems, welches wir mit dem Computer bearbeiten wollen, stellt verschiedene Aussagen zur Verfügung, deren Begleitumstände sich aber laufend ändern; und somit verändert sich auch ihr Wahheitsgehalt laufend. Deswegen muß man jedesmal, wenn sich die Umstände ändern, die Aussage auf ihren neuen Wahrheitsgehalt untersuchen. Ein wenig übersichtlicher wird es mit Hilfe einer sogenannten Wahrheitstafel.

Hat man zwei Aussagen A und B und läßt sie verschiedene Wahrheitswerte annehmen (die Umstände werden verändert), kann man das in einer Tabelle auflisten (-1 steht für TRUE, 0 steht für FALSE). In dieser Tafel sind alle Möglichkeiten festgehalten, die mit den verschiedenen Operatoren erreicht werden können.

A	B	NOT A	A OR B	A AND B
0	0	-1	0	0
-1	0	0	-1	0
0	-1	-1	-1	0
-1	-1	0	-1	-1

In eine solche Tabelle kann man auch die Ergebnisse einer Vergleichsoperation eintragen (Beispielfälle):

x	y	x=y	x>y	x>=y	x<y	x<=y	x<>y
2	3	0	0	0	-1	-1	-1
5	5	-1	0	-1	0	-1	0
3	2	0	-1	-1	0	0	-1

Überlegen Sie sich selbst Aufgabenstellungen mit verschiedenen Bedingungen, und versuchen Sie, die richtigen Verknüpfungen anhand der Wahrheitstafel herauszufinden. Eine gute Übung für

solche Zusammenhänge sind übrigens Logik-Rätsel, wie sie in vielen Magazinen und Zeitungen vorkommen, z.B. im "Zeitmagazin", der Beilage zur Wochenzeitung DIE ZEIT, unter der Rubrik "Logelei mit Zweistein".

Nachdem wir nun die Arbeitsweise von logischen Operatoren kennengelernt haben, wollen wir dieses Wissen von der hohen Ebene, auf der wir die Logik betrachtet haben, auf die niedrigere Ebene der Bits übertragen. Wie sieht nun zum Beispiel die AND-Operation, die uns ja besonders interessiert (siehe Statusvariable), auf der Bitebene aus?

Hier werden einfach einzelne Bits stellenweise miteinander verknüpft, ganz ähnlich der Additionsmethode auf einem Blatt Papier. Dabei gelten auch wieder die Wahrheitstafeln wie wir sie oben schon kennengelernt haben. Ein Beispiel:

Wir verknüpfen die Bytes 01000000 und 01001100. Das Zeichen "&" soll die Operation AND anzeigen, so wie das Zeichen "+" die Addition darstellt:

```
01000000 & 01001100 = 01000000
```

Wenn wir die Nullen als FALSE und die Einsen als TRUE interpretieren, dann kann man mit Hilfe der Wahrheitstafeln dieses Beispiel nachrechnen. Tun Sie das jetzt!

Nachdem Sie das Beispiel nachgerechnet haben, interpretieren wir die Bytes jetzt als Zahlen, wie es auch gegebenenfalls im Computer geschieht. Das erste Byte entspricht der Dezimalzahl 64, das zweite Byte entspricht der Zahl 76. Nach der AND-Operation ist das Ergebnis 64. Durch die Maske (erstes Byte) haben wir gefunden, daß in der Zahl 76 das Bit Nr. 6 enthalten (gesetzt) ist. (Bits werden von 0 beginnend von rechts nach links durchnumeriert!)

Ein zweites Beispiel:

```
00000100 & 01001100 = 00000100
```

Hier hat das erste Byte (Zahl 4) eine andere Form. Wir prüfen durch diese Maske, ob das Bit Nr. 2 gesetzt ist, was im Beispiel der Fall ist. Das Prinzip ist ganz einfach: An der Stelle im Byte, die wir auf ein gesetztes Bit überprüfen, setzen wir in der Maske ein Bit auf 1, der Rest wird auf 0 gesetzt. Dann wird die Maske mit dem gewünschten Byte mit AND verknüpft. Kommt als Ergebnis die Maske selber heraus, dann war das betreffende Bit gesetzt. Übertragen auf das Problem, ein Dateiende zu finden, gingen wir von folgenden Voraussetzungen aus:

a) Die Statusvariable enthält die Information über das Dateiende in Bit 6.

b) Wenn Bit 6 in ST gesetzt ist (auf 1), ist das Dateiende erreicht, ist es nicht gesetzt (auf 0), ist das Dateiende noch nicht erreicht.

c) Das Bit Nr. 6 bezeichnet die Stelle im Dualsystem (7. Position von rechts), die die (Dezimal-)Zahl 64 repräsentiert.

Also bedeutet die Zeile

```
EOF = 64 AND ST
```

nichts anderes als:

- Ist das Dateiende noch nicht erreicht, ist der Inhalt von EOF Null.
- Ist das Dateiende erreicht, enthält EOF die Zahl 64.

Da in BASIC logische Werte durch Zahlen dargestellt werden (0 entspricht FALSE, alle Zahlen ungleich Null entsprechen TRUE), ist es möglich, diese Variable als IF-THEN-Bedingung einzusetzen:

```
IF NOT EOF THEN...
```

Diese Formulierung führt den THEN-Teil solange aus, bis das Dateiende erreicht ist.

Anhang B Das Nähkästchen

Die Cornelius-Ecke

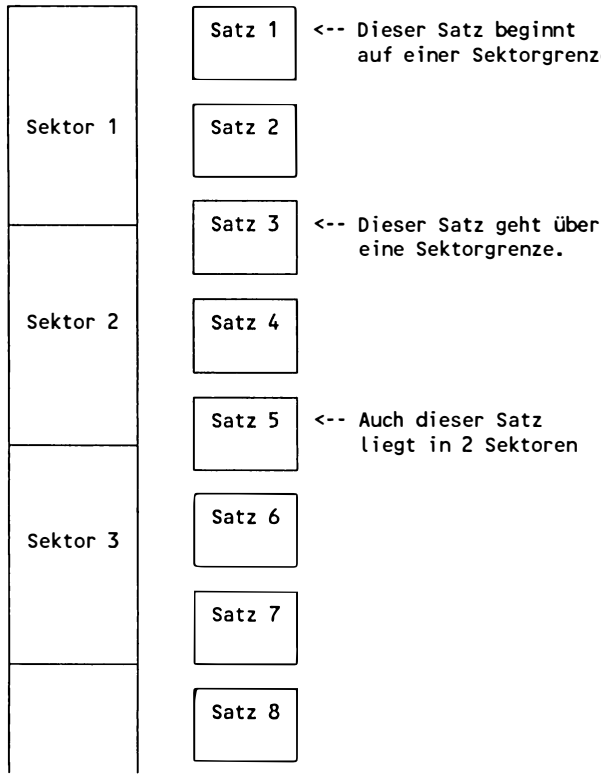
Ein Fehler in der Verwaltung von REL-Dateien oder "There is always one more Bug in the System." (v. C. Keck)

Ein Betriebsystem für Diskettenlaufwerke kann gut durchdacht und lange auf dem Markt sein, es wird trotzdem selten der Fall eintreten, daß dieses System vollständig fehlerfrei ist. Von dieser Tatsache bleiben auch Commodore-Laufwerke und die verschiedenen darauf laufenden DOS-Versionen nicht verschont. Ein zwar seltener, aber dafür sehr lästiger Fehler tritt beim Bearbeiten relativer Dateien (also Dateien des Typs REL) auf.

Der hier besprochene Fehler ist bei allen CBM-Diskettenlaufwerken zu finden; er erzeugt keine Fehlermeldungen und zerstört auch keine Daten in den jeweils betroffenen Dateien. Es kann jedoch während der Veränderung einer REL-Datei zu Fehlern beim Zurückschreiben der Datensätze in die Datei kommen. Die Datensätze werden dann nicht mehr vollständig zurückgeschrieben. Erinnern wir uns:

Mit jeder relativen Datei ist eine maximale Datensatzlänge verbunden. Einzelne Datensätze dürfen zwar kürzer, aber niemals länger als diese Satzlänge sein. Diese Satzlänge wird beim Neuanlegen von REL-Dateien einmal angegeben und kann danach nicht mehr verändert werden. Betroffen von dem hier zu besprechenden Fehler sind alle REL-Dateien, die eine beliebige maximale Satzlänge, außer 1, 2, 127 oder 254, aufweisen. Bei diesen speziellen Datensatzlängen kommt es nicht vor, daß sich mehrere Datensätze über eine Sektorgrenze hinweg erstrecken, da die Datensätze immer gerade in einen Sektor passen. Diese speziellen Datensatzlängen sind echte Teiler der Sektorgröße von 254 Byte.

Eine derartige Serie könnte nun so aussehen:



Eine relative Datei muß nur groß genug sein, um mindestens eine oder mehrere derartiger Serien von Datensätzen zu enthalten. Übereinstimmendes Merkmal dieser Serien ist, daß der erste Datensatz auf einer Sektorgrenze anfängt und alle anderen nicht. Das Vorhandensein einer solchen Serie ist eine Voraussetzung für unseren Fehler.

Eine weitere Vorbedingung ergibt sich aus der Tatsache, daß nicht alle Bytes eines Datensatzes gültige Information enthalten müssen. In der Praxis kann ein Datensatz oftmals kürzer sein, als

die maximale Satzlänge (die beim Anlegen der relativen Datei vereinbart wurde) zuläßt. Die relevante Information enthaltenen Bytes liegen kompakt am Anfang des Datensatzes.

Probleme gibt es jetzt beim Lesen von Datensätzen, die im 2. Sektor einer oben beschrieben Serie beginnen und im 3. Sektor dieser Serie enden. Wenn ein solcher Datensatz (im Beispiel Nr. 5) im 3. Sektor der Serie Daten enthält, werden beim Lesen dieses Satzes von den im 3. Sektor der Serie abgelegten Daten maximal so viele Bytes übernommen, wie der erste Datensatz der Serie insgesamt enthält. Die (angeblich fehlenden) Daten sind zwar auf der Diskette vorhanden und werden auch vom Diskettencontroller gelesen, aber aufgrund einer fehlerhaften Dateiverwaltung nicht als gültig erkannt und deswegen nicht an den Rechner übertragen.

Ein Beispielprogramm dazu:

```
10 print "demoprogramm für relfile-lesefehler bei cbm-"
20 print "diskettenlaufwerken"
30 input "max. datensatzlänge",ml
40 input "länge des kurzen satzes",lk
50 input "länge der langen sätze",ll
60 if (ml>254) or(lk>=ll) or (ll>ml) or (ml<10) or (ll>26) then 10
70 rem erzeuge füllstring
80 x$="":for l=1 to ll:x$=x$+chr$(64+ll):next l
100 k$=left$(x$,lk) : rem ein kurzer datensatz
110 l$=left$(x$,ll) : rem ein langer datensatz
120 open 1,8,15,"i" : rem kommandokanal zur floppy öffnen
125 print#1,"s0:dummy" : rem rel. datei evtl. löschen
130 open 8,8,8,"0:dummy,l,"+chr$(ml) : rem relative datei
140 rem mit maximaler satzlänge ml auf laufwerk 0 er-
150 rem öffnen
160 print#1,k$ : rem kurzen datensatz anlegen
170 for l=1 to 30 : rem lange datensätze anlegen
180 print#1,l$
190 next l
200 close 8:close 1:rem dateien schliessen
210 open 1,8,15,"i" : rem kommandokanal zur floppy öffnen
220 open 8,8,8,"0:dummy" : rem relative datei namens
230 rem dummy auf laufwerk 0 eröffnen. die datei ist vor-
240 rem handen, also können wir uns die längenangabe
250 rem schenken.
260 v$=l$: rem v$ brauchen wir zum vergleichen
270 rem sehen wir uns also die langen datensätze an. um den
280 rem ersten satz der datei brauchen wir uns nicht zu
```

```
290 rem kümmern, der ist sowieso etwas kürzer.
300 for l=2 to 30 : rem mehr datensätze gibt's nicht
310 print#1,"p"+chr$(96+8)+chr$(i)+chr$(0)+chr$(0)
320 rem für basicversionen > 2.0 kann man hier auch
330 rem schreiben : 310 record#8,(i)
340 input#8,l$ : rem lesen eines langen (?) datensatzes
350 if l$<>v$ then print l,l$,v$ : rem bei ungleichheit
360 rem gibt's hier etwas zu lesen.
370 next l
380 close 8:close 1:end
```

Der Fehler kann zwar nicht behoben werden (dazu wäre ein ziemlich aufwendiger Eingriff ins Betriebssystem des jeweiligen Laufwerks notwendig), aber er kann umgangen werden, indem man nach dem Lesen und Schreiben eines Sektors in eine relative Datei den Zeiger auf den gelesenen bzw. geschriebenen Datensatz erneut positioniert. Der Zeitbedarf hierfür ist minimal, da nur der Befehl über den Bus übertragen wird. Das Floppylaufwerk erkennt, daß es den betreffenden Sektor bereits im Speicher hat.

In unser Beispielprogramm müssen dazu folgende Zeilen eingefügt werden:

```
341 print#1,"p"+chr$(96+8)+chr$(i)+chr$(0)+chr$(0)
342 rem für basicversionen > 2.0 kann man hier auch
343 rem schreiben : 341 record#8,(i)
```

Side Sectors

Um auf die Daten von relativen Dateien sofort zugreifen zu können, muß man nicht nur die Datensätze abspeichern, sondern auch angeben, wo man diese abgelegt hat. Diese Information wird in den sogenannten Side Sectors untergebracht. Das DOS erledigt diese Aufgabe automatisch, so daß wir damit nicht behelligt werden. Um zu verstehen, wie das funktioniert, gehe ich kurz auf das Prinzip der Struktur von sequentiellen Dateien ein. Damit man sich von Zeichen zu Zeichen "hangeln" kann, muß irgendwo festgelegt sein, welche Sektoren von der Datei belegt sind und in welcher Reihenfolge sie benutzt werden. Man kann sich zwei Prinzipien überlegen:

- Track-Sektor-Liste: Man merkt sich in einer Liste, welche Sektoren von der Datei benutzt werden und schreibt sie in der Reihenfolge in die Liste, in der sie beschrieben worden sind.

- Sektorverkettung: Man merkt sich nur den ersten Sektor und schreibt in jeden weiteren, welcher Sektor als nächstes folgt.

Jedes Verfahren hat seine Vor- und Nachteile. Das erste Verfahren wurde beim altehrwürdigen APPLE II+ eingesetzt, das zweite Verfahren kommt bei den in diesem Buch besprochenen DOS-Versionen zum Einsatz.

Nun wieder zu den relativen Dateien! Da man eine feste Datensatzlänge hat, ist es möglich, sofort die Lage eines bestimmten Datensatzes innerhalb einer Datei zu berechnen. Das geschieht nach folgender Formel:

BDSn = DSL * n

wobei DSL die Datensatzlänge, BDSn das erste Byte und n die Nummer des gewünschten Datensatzes bezeichnen.

Aber um dorthin zu gelangen, müßte man sich wie in einer sequentiellen Datei durch die gesamte Datei lesen. Und das wäre nicht viel schneller. Wenn wir aber die Anzahl der Bytes bis zum gewünschten Sektor (das sind n * DSL) durch 254 teilen (das sind die Bytes eines Sektors), dann könnte man den Sektor ermitteln, in dem der Datensatz anfängt. In diesem Fall hätte man schon viel gewonnen. Der Rest der ganzzahligen Division (n * DSL) / 254 gibt an, an welcher Position (bei welchem Byte) der Datensatz im ermittelten Sektor beginnt. Jetzt müßte man nur noch irgendwo die Sektorverkettung notiert haben, dann könnte man sofort in den Sektor gehen, die Bytes abzählen - und schon wäre man da! Das ist der Punkt, an dem die Side Sektors ansetzen. Hier ist nämlich mitprotokolliert worden, wie die Sektoren verkettet worden sind! Das heißt, in den Side Sektors steht die Track-Sektor-Liste der Datei.

Die messerscharfe Schlußfolgerung: Eine relative Datei ist eigentlich nur eine sequentielle Datei, bei der zusätzlich eine Track-Sektor-Liste angelegt worden ist. Ein solcher Side Sektor kann genau 120 Sektoren verwalten, die ganze Liste besteht aus 6 Side Sektoren. Daraus ergibt sich nach kurzer Rechnung, daß eine relative Datei maximal aus 6 * 120 = 720 Sektoren bestehen darf. Das sind 720 * 254 = 182880 Bytes.

Aufgepaßt! Gerücht!

Man bekommt oft zu hören, daß eine relative Datei nur 720 Datensätze enthalten darf. Die Ursache für dieses Gerücht, denn mehr ist es nicht, ist wohl die amerikanische Originalanleitung der 1541 ("VIC-1541 User's Manual", Second Edition 1982, S. 33). Dort steht es mindestens mißverständlich (Zitat):

"...Each side sector can point up to 120 records, and there may be 6 side sectors in a file. There can be up to 720 records in a file, and each record can be up to 254 characters, so the file could be as large as the entire diskette..."

Das Wort "record" wird in der EDV gewöhnlich zur Bezeichnung von Datensätzen benutzt (siehe Pascal). Wenn man 'record' auf diese Weise übersetzt, kommt zwangsläufig ein völlig anderes Bild heraus, als es eigentlich den Tatsachen entspricht. In der deutschen Übersetzung der 1541-Anleitung wird 'record' mit 'Datenblöcke' übersetzt, was der Wahrheit schon näherkommt, aber immer noch mißverständlich ist. Mit 'record' ist in der besagten Anleitung aber eindeutig 'Sektor' gemeint! Bei den "720 records" handelt es sich also um Sektoren (Blöcke), nicht um Datensätze. Es können also auch auf der "kleinen" 1541 mehr als 720 Datensätze in eine relative Datei gepackt werden.

Wie sieht nun ein Side Sektor aus? Eine Tabelle der in einem Side Sektor benutzten Bytes folgt auf der nächsten Seite.

Bytenr.	Erklärung
0	Spur des nächsten Sektors
1	Sektornummer des nächsten Side Sektors
2	Laufende Nummer dieses Side Sektors (0..5)
3	Festgelegte Länge eines Datensatzes der zugehörigen relativen Datei
4	Spur des ersten Sektors
5	Sektor des ersten Side Sektors
6	Spur des zweiten Sektors
7	Sektor des zweiten Side Sektors
8	Spur des dritten Sektors
9	Sektor des dritten Side Sektors
10	Spur des vierten Sektors
11	Sektor des vierten Side Sektors
12	Spur des fünften Sektors
13	Sektor des fünften Side Sektors
14	Spur des sechsten Sektors
15	Sektor des sechsten Side Sektors
16	bis Track-Sektorliste der Datensektoren255 der relativen Datei

Nach diesem Wissen kann man das folgende Verfahren zur Errechnung des gewünschten Datensektors gut nachvollziehen:

Mit INT ((DSL * n) / 254) bekommt man die Anzahl der Sektoren bis zu dem Sektor, der den Datensatz enthält. Teilen wir diese Zahl durch 120, erfahren wir, in welchem Side Sektor sich die Nummer jenes Sektors befindet. Der Rest der Division ergibt nun die Position der Track-Sector-Angabe im Side Sector. Jetzt kennen wir den Sektor, in dem sich der Datensatz befindet. Mit dem Rest der ersten Division haben wir die Position des ersten Bytes des gewünschten Datensatzes im gefundenen Sektor. Wenn unglücklicherweise der Datensatz in den nächsten Sektor reicht, so ist das auch kein großes Problem, denn im vorigen Abschnitt wurde ja darauf hingewiesen, wie man sich in einem solchen Fall verhalten sollte.

Anhang C Directory, BAM - Aufbau

Directory:

Byte	Bedeutung
0	Dateityp Bit 0,1,2 = 0 DEL, gelöscht 1 SEQ 2 PRG 3 USR 4 REL 6 auf 1 = Read only 7 auf 1 = Datei geschlossen
1-2	Spur und Sektor des ersten Datensektors des Eintrags
3-18	Dateiname, max. 16 Zeichen, wird mit A0H aufgefüllt
19-20	Nur bei REL: Spur und Sektor des ersten Side Sektor Blocks
21	Nur bei REL: Länge eines Datensatzes
22-25	nicht benutzt
26-27	Nur bei SAVE@: Zwischenspeicher für Spur und Sektor des ersten Sektors der neuen Datei
28-29	Anzahl der von der Datei benutzten Blöcke im 16-Bit-Format, Lowbyte-Highbyte

BAM:

Byte	Erläuterung
0	Spur des ersten Directory Blocks, üblicherweise Spur 18
1	Sektor des ersten Directory Blocks, üblicherweise Sektor 1
2	Formatbezeichnung; Bei 1541/1570/1571 immer "A" (=65H). Wird ein falsches Formatkennzeichen erkannt, ist die Diskette automatisch schreibgeschützt.
3	Bit 7 ist Flag zur Umschaltung auf zwei Diskettenseiten: 0 = 1541/70, 1=1571
4-143	BAM-Verzeichnis für die Diskettenseite. Beginn bei Spur 1, 4 Bytes = 1 Spur
144-159	Name der Diskette, wird beim Formatieren vergeben. Max. 16 Zeichen lang, wird mit A0H aufgefüllt.
160-161	2 mal A0H
162-163	ID
164	Versionsnummer, nicht aktualisiert
165	Formatkennzeichen (s. Byte 3)
167-170	3 mal A0H
171-220	49 mal 0
221-255	Bei zweiseitigem Laufwerk (1571) Angabe über die auf der Rückseite verfügbaren Sektoren

Anhang D CBM-Laufwerksdaten

	1541/c	1551	1570	1571
Kapazität	170k	170k	170k	340k
Spuren	35	35	35	70
Sektoren/Spur	17-21	17-21	17-21	17-21
Bytes/Sektor	256	256	256	256
DOS-Version	2.6	3.0	3.0	3.0
Bus	ser.	ser.	ser.	ser.

Anhang E Das 3½"-Laufwerk 1581

Kapazität	1Mb (unformatiert) 808960 Byte (formatiert)
max. Dateigröße SEQ	802640 Byte
max. Dateigröße REL	ca. 800k
Datensätze/Datei	64k
Dateien/Diskette	296
freie Blöcke pro Diskette	3160
logische Byte/Sektor	256
physikalische Byte/Sektor	512

Das Laufwerk 1581 hat einige Spezialitäten, die sich nicht nur in der Abmessung des Gehäuses und in der Diskettengröße ausdrücken. Zunächst wirkt es sehr unscheinbar und zierlich, aber bei näherem Hinsehen merkt man, das man ein der Zeit angemessenes Laufwerk vor sich hat. Da fällt zuerst die Tatsache auf, daß sich die Zahl der physikalischen Sektoren von der Zahl der logischen Sektoren unterscheidet. Man hat einfach 2 Blöcke in einen Sektor gepackt. Um nämlich 1 MByte Daten auf der Platte verwalten zu können, müssen die Sektoren etwas länger sein, als die gewohnten 256 Byte Blöcke. Da man aber sehr viel Platz verschwenden würde, wenn man 512 Byte Blöcke als einen Sektor ansähe, legt man einfach zwei gewohnte 256 Byte Blöcke zusammen in einen großen 512 Byte Block. Sie werden dann bei Bedarf logisch getrennt.

Unterverzeichnisse, Partitionen

Eine weitere tolle Eigenschaft ist die der Partitionierung. So etwas kennt man sonst nur von Festplatten. Aber bei 1 MByte Speicherkapazität ist diese Eigenschaft schon sinnvoll. Beim Partitionieren wird die physikalische Platte in mehrere logische Teile untergliedert. Man kann nun zwei verschiedene Betriebssysteme "fahren", beispielsweise Commodore-DOS und CP/M, und das alles auf einer Diskette.

Eine andere Möglichkeit ist, verschiedene Benutzerbereiche zu definieren, die voreinander geschützt sind. In Anlehnung an die Eigenschaften von UNIX und MS-DOS heißen solche Bereiche

Subdirectories oder Unterverzeichnisse. Diese Unterverzeichnisse bekommen eigene Namen, ja man könnte sogar sagen, daß diese Subdirectories (wenigstens bei Commodore) "Platten auf der Platte" sind. Sie müssen mit HEADER gesondert formatiert werden, bevor sie benutzt werden können. Einen Bereich richtet man mit dem Befehl

```
PRINT#<lfn>,"/0:<bereichsname>,"+CHR$(<ta>)+CHR$(<as>)+CHR$(<zdshi
>)+CHR$(<zdslo>)+",C"
```

ein, wobei

<lfn>	logische Filenummer
<bereichsname>	Name des Unterbereichs
<ta>	Anfangsspur
<as>	Anfangssektor
<zdshi>	Highbyte der Anzahl der Sektoren
<zdslo>	Lowbyte

bedeuten.

Beispiel:

```
10 OPEN1,8,15
20 PRINT#1,"/0:DATEN,"+CHR$(1)+CHR$(0)+CHR$(1)+CHR$(144)+",C"
30 CLOSE1
40 ...
```

Damit wird der Unterbereich namens "DATEN" ab Spur 1, Sektor 0 mit einer Länge von 400 Blöcken (Highbyte=1, Lowbyte=144) eingerichtet.

Um aus einer Partition ein Subdirectory machen zu können, müssen folgende 4 Punkte eingehalten werden:

- Der Bereich darf Spur 40 nicht beinhalten, da das die Systemspur ist.

- Anfangssektor muß 0 sein.
- Endsektor muß ein Vielfaches von 40 sein
- Der Bereich muß mindestens 120 Sektoren lang sein

Innerhalb eines Bereiches dürfen wieder Bereiche eingerichtet werden, und auch in diesen können weitere Unterbereiche angelegt werden. So kann man eine ähnliche Baumstruktur auf seine Diskette bringen wie es aus UNIX und MS-DOS bekannt ist. Angewählt wird ein solcher Unterbereich mit dem Befehl

```
PRINT#<lfn>,"/0:<bereichsname>"
```

wobei <lfn> die Filenummer des Befehlskanals sein muß. Nach dem Anwählen muß der Bereich gesondert formatiert werden. Nur nach dem erfolgreichen Anwählen des Unterverzeichnises darf formatiert werden, da sonst das Verzeichnis formatiert wird, in dem Sie sich gerade befinden. Die erfolgreiche Anwahl kann durch den Fehlerkanal oder die Fehlervariablen geprüft werden. Wenn die Meldung 02 ("SELECTED PARTITION") angezeigt wird, war die Anwahl erfolgreich. Ansonsten erscheint Meldung 77 ("SELECTED PARTITION ILLEGAL"). Man kann, anders als in UNIX oder MS-DOS, immer nur in Einzelschritten vorgehen. Will man in ein Unterverzeichnis 2. Ordnung muß man zunächst das dazwischenliegende Unterverzeichnis anwählen, dann erst kann man in das gewünschte Subdirectory schalten. Wollen Sie von einem Subdirectory in ein anderes, das in einem anderen Pfad liegt, müssen Sie den mühsamen Weg über die Rootdirectory (engl.: root = Wurzel) gehen. In das Wurzelverzeichnis gelangen Sie sofort mit

```
PRINT#<lfn>,"/"
```

wobei <lfn> wiederum die Filenummer des Befehlskanals sein muß. Wie in solchen Baumstrukturen üblich, können Dateien, die außerhalb der eigenen Directories liegen, nicht erreicht werden. Jedes Unterverzeichnis hat sein eigenes Directory in der ersten verfügbaren Spur des Verzeichnisses. Das Directory und die BAM werden dort im selben Format gespeichert wie die In-

formationen auf der Systemspur (Spur 40). Diese Unterverzeichnisdirectories müssen vom Programmierer geschützt werden; das DOS schützt diese Informationen nicht!

Bereiche können wie Dateien mit SCRATCH gelöscht werden. Ist der Bereich ein Subdirectory, gehen alle Dateien darin verloren.

Der letzte Punkt ist die Fähigkeit, das Laufwerk 1581 in den 1541-Modus zu schalten, so daß sich dieses 3½"-Laufwerk wie die gute alte C64-Floppy verhält. Sogar an den alten VC20 kann das Ding angeschlossen werden! In der seriellen Datenübertragung erkennt das Laufwerk automatisch die verschieden schnellen Betriebsarten des C64 und des C128. Wollen Sie das 1581 im 1541-Modus betreiben, schalten Sie es einfach ein. Nach dem Einschalten befindet sich die Floppy zunächst im 1541-Modus. Erst wenn beim Rechner eine hohe serielle Datenübertragungsrate festgestellt wird (z.B. C128), schaltet das Gerät auf den 1581-Modus. An einem C64 wird es also im 1541-Modus verharren. Man kann das allerdings beeinflussen:

C64 Den 1581-Modus erreicht man durch OPEN1,8,15, "U0[gfac]/>M1". Der C64 greift dadurch nicht schneller auf das Laufwerk zu, man hat aber mehr Platz auf der Diskette.

C128 Umschalten in den 1541-Modus durch OPEN1,8,15, "U0[gfac]/>M0[gfac]/".

Zusammenfassend kann man sagen, daß zwar die moderne Struktur übernommen worden ist, nicht aber die Bequemlichkeiten und Sicherheiten. Vergleicht man die Größe der Betriebssysteme, kann man jedoch getrost sagen, daß hier für wenig Geld in einem kleinen Betriebssystem viel geboten wird. Die vielen Features machen das Gerät zu einem standesgemäßen neuen Mitglied in der Commodore-Familie.

Anhang F ASCII-Tabelle

Die Tabelle der ASCII-Zeichen ist aufgebaut wie folgt:

Die erste Zahl ist der Code, danach kommt die Bezeichnung des Zeichens und die Bezeichnung der Taste, evtl. noch seine Wirkung (bei Steuerzeichen). Die Liste geht nur bis 127, da ab diesem Punkt viele Rechner voneinander abweichen. Steuerzeichen sind alle Zeichen von 0 bis 31 ihre Bezeichnung wird nicht erläutert, sie ist aber in der Fachliteratur nachzulesen. Steuerzeichen sind über die Taste <CTRL> zu erreichen. Sie wird wie <SHIFT> bedient. 'CTRL' steht für 'Control', und so werden die Steuerzeichen auch bezeichnet: Control-Codes (z.B.: CTRL-A gesprochen "Control A" wird durch gleichzeitiges Betätigen der Tasten <CTRL> und "A" erreicht). In der Liste wird ein Control-Zeichen durch einen vorangestellten Hochpfeil "^" dargestellt.

Die Bezeichnungen der Steuerzeichen stammen aus der Zeit, als man vornehmlich mit dem Fernschreiber arbeitete. Deswegen die Bezeichnungen für "Wagenrücklauf" (Carriage Return, CR) oder "Zeilenvorschub" (Line Feed, LF). Auch die "Glocke", die das Zeilenende ankündigte, war vertreten (Bell). Daneben gibt es noch Backspace (BS), Horizontal Tab (HT) usw. Die Bedeutung der Zeichen hat sich im Laufe der Zeit etwas verschoben, aber die Namen sind geblieben.

Für die ASCII-Zeichen [,\,],{,|,},~,@ stehen im deutschen Zeichensatz: Ä,Ö,Ü,ä,ö,ü,ß,§

00	NULL	^@:	32	SPACE:	64	@:	96	`
01	SOH	^A:	33	!:	65	A:	97	a
02	STX	^B:	34	":	66	B:	98	b
03	ETX	^C:	35	#:	67	C:	99	c
04	ET	^D:	36	$:	68	D:	100	d
05	ENQ	^E:	37	%:	69	E:	101	e
06	ACK	^F:	38	&:	70	F:	102	f
07	BEL	^G (Beep):	39	':	71	G:	103	g
08	BS	^H:	40	(:	72	H:	104	h
09	HT	^I (Tab):	41	):	73	I:	105	i

10	LF	^J (Linefeed):	42	*:	74	J:	106	j
11	VT	^K:	43	+:	75	K:	107	k
12	FF	^L (Formfeed):	44	,:	76	L:	108	l
13	CR	^M (Carriage Return):	45	-:	77	M:	109	m
14	SO	^N:	46	.:	78	N:	110	n
15	SI	^O:	47	/:	79	O:	111	o
16	DLE	^P:	48	0:	80	P:	112	p
17	DC1	^Q:	49	1:	81	Q:	113	q
18	DC2	^R:	50	2:	82	R:	114	r
19	DC3	^S:	51	3:	83	S:	115	s
20	DC4	^T:	52	4:	84	T:	116	t
21	NAK	^U:	53	5:	85	U:	117	u
22	SYN	^V:	54	6:	86	V:	118	v
23	ETB	^W:	55	7:	87	W:	119	w
24	CAN	^X:	56	8:	88	X:	120	x
25	EM	^Y:	57	9:	89	Y:	121	y
26	SUB	^Z:	58	::	90	Z:	122	z
27	ESC	ESC:	59	:	91	[:	123	{
28	FS	^\:	60	<:	92	\:	124	|
29	GS	^]:	61	=:	93	]:	125	}
30	RS	^^:	62	>:	94	^:	126	~
31	US	^_:	63	?:	95	SPC:	127	DEL

Anhang G Floppy-Fehlermeldungen

Nummer	Fehlerbedeutung
00	OK-Meldung
01	FILES SCRATCHED,XX Rückmeldung nach dem Löschen XX gibt die Zahl der gelöschten Dateien an Die Angaben TT und SS sind die Spur- und Sektornummer des Datenblocks, wo der Fehler auftrat.
20	READ ERROR,TT,SS Der Sektorheader eines Blocks wurde nicht gefunden. Dabei handelt es sich um eine unformatierte oder beschädigte Diskette.
21	READ ERROR,TT,SS Die Sync-Markierung wurde nicht gefunden. Entweder ist die Diskette nicht formatiert oder es liegt ein Fehler am Laufwerk, beispiels-weise ein dejustierter Lesekopf, vor.
22	READ ERROR,TT,SS Der Datenblock eines Sektors wurde nicht gefunden.
23	READ ERROR,TT,SS Es wurde ein Prüfsummenfehler festgestellt. In diesem Fall müssen Sie versuchen, den Sektor noch zu retten, indem Sie mit den Direkt-zugriffsbefehlen den Sektor mehrmals lesen, bis der Fehler eventuell nicht mehr auftritt. Ansonsten müssen Sie den Sektor in den Floppy-Puffer einlesen und neu zurückschreiben. Dabei wird die Prüfsumme neu berechnet, allerdings kann der Inhalt des Sektors fehlerhaft sein
25	WRITE ERROR,TT,SS Beim Schreiben eines Sektors wurde beim nachträglichen Verify ein Fehler fest-gestellt. Sie sollten in diesem Fall eine neue Diskette verwenden.

26 WRITE PROTECT ON,TT,SS
Die Diskette ist durch eine Schreibschutz-marke geschützt.

27 READ ERROR,TT,SS
Im Sektorheader wurde ein Prüfsummenfehler festgestellt.

29 DISK ID MISMATCH,TT,SS
Die ID des Sektorheaders stimmt nicht mit der zuletzt gelesenen ID überein. Maßnamen: Diskette initialisieren oder neu formatieren.

30 SYNTAX ERROR
Die C1570/71 kennt den Befehl, der über den Befehlskanal gesendet wurde nicht.

31 SYNTAX ERROR
Der Befehl kann von der C1570/71 nicht aus-geführt werden. (z.B. Backup)

32 SYNTAX ERROR
Der über den Befehlskanal gesendete Befehl ist länger als 41 Zeichen und der Eingabepuffer der Floppy ist gefüllt.

33 SYNTAX ERROR
Der Joker wurde beim Schreiben als Dateiname verwendet.

34 SYNTAX ERROR
Der Dateiname wurde nicht gefunden. Sie haben eventuell den Doppelpunkt nach dem Befehls-buchstaben vergessen.

39 FILE NOT FOUND
Die angegebene Autostart-Datei wurde auf der Diskette nicht gefunden.

50 RECORD NOT PRESENT
Der angesprochene Datensatz einer relativen Datei ist nicht angelegt. Beim ersten Schreiben des Datensatzes kann diese Meldung ignoriert werden. Beim Lesen weist sie darauf hin, daß der gewünschte Datensatz nicht existiert.

51 OVERFLOW IN RECORD
Die an die Floppy übertragenen Daten sind länger als der Datensatz, wobei die überzähligen Zeichen ignoriert werden.

52 FILE TOO LARGE
Die Nummer des letzen Datensatzes ist zu groß, da eine derartige Datei nicht mehr auf die Diskette passen würde.

60 WRITE FILE OPEN
Es wurde versucht auf eine nicht ordnungsgemäß geschlossene Datei zuzugreifen. Diese kann nur im 'Modify'-Modus eröffnet werden.

61 FILE NOT OPEN
Es wurde versucht eine Datei zu benutzten, die nicht geöffnet wurde.

62 FILE NOT FOUND
Das angegebene Programm oder die Datei wurden nicht gefunden.

63 FILE EXISTS
Die neue Datei ist auf der Diskette bereits vorhanden

64 FILE TYPE MISMATCH
Der beim Eröffnen angegebene Dateityp stimmt nicht mit dem im Directory verzeichneten Dateityp überein.

65 NO BLOCK,TT,SS
er bei Block-Allocate angegeben Block ist bereits belegt. TT und SS geben die Spur- und Sektornummer des nächsten freien Blocks der Spur an. Haben TT und SS den Wert 0, dann ist kein freier Sektor mehr vorhanden. Bitte beachten Sie das in Kapitel 2.1.3 angegebene Fehlverhalten des Block-Allocate-Befehls.

66 ILLEGAL TRACK OR SECTOR,TT,SS
Die beim Direktzugriffsbefehl angegebenen Sektorparameter sind fehlerhaft.

67 ILLEGAL TRACK OR SECTOR,TT,SS
Die Sektorverkettung zeigt auf einen Sektor, der nicht vorhanden ist.

70 NO CHANNEL
Es ist kein weiterer Kanal mehr verfügbar. Sie müssen zuerst eine noch offene Datei wiederschließen, damit wieder ein Kanal zur Verfügung steht.

71 DIR ERROR,TT,SS
Das BAM-Verzeichnis im Floppy-Speicher stimmt nicht mit dem BAM-Verzeichnis auf der Diskette überein. Sie müssen in diesem Fall die Diskette initialisieren

72 DISK FULL
Die Kapazität der Diskette ist erschöpft und es sind weniger als drei Blocks frei.

73 Einschaltmeldung
Es wurde versucht eine Diskette zu beschreiben, die unter einem anderen DOS formatiert wurde.

74 DRIVE NOT READY
Es ist keine formatierte Diskette eingelegt.

Anhang H 1581 Fehlermeldungen

Die Floppy 1581 hat einige Besonderheiten, die einige andere Fehlermeldungen zur Folge haben. Diese Fehler, die zusätzlich zu denen in der obigen Liste benutzt werden, werden hier aufgelistet. Es sind vier:

Nummer	Fehlerbedeutung
02	PARTITION SELECTED Kein Fehler. Die geforderte Diskettenpartition (Bereich, Subdirectory) ist erfolgreich angewählt.
75	FORMAT ERROR Das Formatieren einer Diskette ist nicht erfolgreich abgeschlossen worden.
76	CONTROLLER ERROR Der Diskettencontroller (Baustein WD177x) im Laufwerk 1581 ist defekt.
77	SELECTED PARTITION ILLEGAL Bei der Anwahl einer Partition oder einer Subdirectory wurde auf einen Bereich zugegriffen, der nicht die geforderten Eigenschaften einer Partition besitzt.

Um mehr Informationen über die großartige Fähigkeit der Partitionierung eines Diskettenlaufwerks zu erfahren, sei auf das Handbuch[17] des 1581 verwiesen, das zu der seltenen Gattung der kompakten, aber reichhaltigen Anleitungen gehört. Kompliment an die Verfasser!

17 Commodore Büromaschinen GMBH: Commodore Diskettenlaufwerk 1581, Bedienungshandbuch, Frankfurt 1986

Anhang I Kurzübersicht der Befehle

Bei den Befehlen der BASIC-Versionen bis 2.0 ist der Aufbau des Befehls fast immer gleich. Eingeleitet wird ein solcher Befehl mit

```
OPEN 1,<ga>,15,<befehlssequenz>.
```

Deshalb wird in dieser Tabelle nur die Befehlssequenz aufgeführt. In diesem Fall ist der Befehl mit (+) gekennzeichnet. Alle anderen 2.0-Befehle und die Befehle ab 3.0 werden vollständig aufgelistet.

BASIC Version bis 2.0 ab 3.0

Formatieren einer Diskette:

```
(+)
"N:<dateiname>,<id>"HEADER "<dateiname>",I<id>,U<ga>
```

Beispiel:

```
"N:MEINE DISKETTE,A1"HEADER "MEINE DISKETTE",IA1,U8
```

Laden eines BASIC-Programms:

```
LOAD "<dateiname>",<ga>DLOAD "<dateiname>"{,U<ga>}
```

Beispiel:

```
LOAD "MEINPROGRAMM",8DLOAD "MEINPROGRAMM"
```

Speichern eines BASIC-Programms:

```
SAVE "{@:}<dateiname>",<ga>DSAVE "MEINPROGRAMM"{,<ga>}
```

Beispiel:

```
SAVE "MEINPROGRAMM",8DSAVE "MEINPROGRAMM"
SAVE "@:MEINPROGRAMM",8DSAVE "@MEINPROGRAMM"
```

Programme überprüfen:

```
VERIFY "<dateiname>",<ga>DVERIFY "<dateiname>"{,U<ga>}
```

Beispiel:

```
VERIFY "MEINPROGRAMM",8DVERIFY "MEINPROGRAMM"
```

Speicherbereiche laden:

```
LOAD "<dateiname>,<ga>,1BLOAD "<dateiname>" ON B<b>,P<s>
```

Beispiel:

```
LOAD "DUMP",8,1BLOAD "DUMP" ON B0,P3072
```

Speicherbereich speichern:

```
(nicht vorhanden)BSAVE "DUMP",U<ga> ON B<b>,P<s>
TO P<e>
```

Beispiel:

```
BSAVE "DUMP",U8 ON B0,P0 TO P100
```

Directory:

```
LOAD "$",<ga>DIRECTORY
LIST
```

Löschen einer Datei:

```
(+)
"S:<dateiname>"SCRATCH "<dateiname>",U<ga>
```

Beispiel:

```
"S:ALTESPROGRAMM"SCRATCH "ALTESPROGRAMM",U8
"S:ALT,GANZALT,URALT"SCRATCH "ALT,GANZALT,URALT",U8
```

Ordnen der Diskette:

```
(+)
"V"COLLECT ON U<ga>
```

Umbenennen einer Datei:

```
(+)
"R:<neuname>=<altname>"RENAME "<altname>" TO "<neuname>"
```

Beispiel:

```
"R:NEUDATEI=ALTDATEI"RENAME "ALTDATEI" TO "NEUDATEI"
```

Dateien verketten, kopieren:

```
(+)
"C:<ziel>=<d1>,<d2>,..."CONCAT "<d1>" TO "<ziel>" ON U<ga>
COPY "<quelle>" TO "<ziel>" ON <ga>
```

Beispiel:

```
"C:GROSS=KLEIN1,KLEIN2"CONCAT "KLEIN" TO "GROSS" ON U8
"C:TEST.BAK=TEST"COPY "TEST" TO "TEST.BAK" ON U8
```

Schließen aller Kanäle:

```
(nicht vorhanden)DCLEAR ON U<ga>
```

Anhang J Glossar

Access Zugriff

Adresse Nummer zur physikalischen Kennzeichnung eines Speicherplatzes in einem Rechner

AI Artificial Intelligence (= Künstliche Intelligenz, KI)

Algorithmus Eindeutige Verfahrensvorschrift

Alphanumerische Zeichen
Buchstaben, Ziffern und Sonderzeichen

Arithmethic Logical Unit
Arithmetisch-Logische Einheit, Rechenwerk eines Prozessors

ASCII American Standard Code of Information Interchange (amerikanische Norm der Zeichencodierung)

BASIC Beginners All Purpose Symbolic Instruction Code

Baud bit/s, Geschwindigkeit der Datenübertragung

BCD Binary Coded Decimal Code zur Darstellung von Zahlen: Jede Ziffer wird durch 4 Bit (=1 Nibble) dargestellt. Solchermaßen codierte Zahlen weisen bei Berechnungen keine Rechenungenauigkeiten auf, sind aber aufwendiger in Rechenoperationen zu handhaben. Berechnungen im BCD sind zwar genauer, aber auch langsamer.

Bit kleinste Einheit einer Information, kann zwei definierte Werte annehmen: 0 oder 1 (bzw. L (low) oder H (high))

Buffer Pufferspeicher, Zwischenspeicher

Bug Wanze, Fehler im Programm

Bus Mehradrige Sammelleitung, auf der alle Signale des Prozessors abgegriffen werden können. Um Kompatibilität zwischen Geräten unterschiedlicher Hersteller zu erreichen, gibt es verschiedene BUS-Normen (ECB, Apple, VME,...)

Byte nächstgrößere Einheit: 8 Bit = 1 Byte (1k Byte = 1 kilobyte = 1024 byte, 1M Byte = 1 Megabyte = 1024 kByte)

CPU Central Processing Unit = Zentrale Rechnereinheit

DOS Diskette Operating System, Programm zur Steuerung des Datenflusses von und zur Diskette

Drucker Matrixdrucker: Verbreitetste Druckerart. Buchstaben und Zeichen werden aus einer Matrix von Punkten zusammengesetzt. Ein Druckkopf mit 9, 18 oder 24 Nadeln, die untereinander angeordnet sind, fährt diese Matrix (elektronisch im Drucker festgelegt) ab. Je mehr Nadeln, um so bessere Druckqualität. Drucker mit schlechter Qualität sind heute nur noch selten. Matrixdrucker sind sehr gut graphikfähig! Geschwindigkeit: 30 bis 400 Zeichen pro Sekunde. Typenraddrucker: Schönschriftdrucker arbeiten mit einem Schreibmaschinentypenrad, nur eingeschränkt graphikfähig. Geschwindigkeit: 10 bis 80 Zeichen pro Sekunde. Tintenstrahldrucker: sog. "Tintenspritzer" sind sehr leise, da sie statt Nadeln wie der Matrixdrucker mit einer Tintendüse arbeiten. Die

Zeichen werden ebenfalls aus einer Punktmatrix zusammengesetzt. Keine Durchschläge möglich! Graphikfähig. Geschwindigkeit: wie Matrixdrucker. Laserdrucker: L. werden heute immer billiger und geraten in den Preisbereich guter Typenraddrucker (ca. 5000,- DM). L. arbeiten wie Fotokopierer, allerdings wird die Vorlage nicht wie beim Kopierer optisch, sondern elektronisch aus dem Computer aufbereitet. Keine Durchschläge möglich. Extrem gut graphikfähig. Geschwindigkeit: Mehrere Seiten pro Minute (variiert sehr stark, je nach Preis).

Bei allen Druckern kann man unter verschiedenen Zeichensätzen und Schriftarten wählen. Der Typenraddrucker ist hier jedoch eine Ausnahme: Man ist immer auf den Zeichensatz des benutzten Typenrades beschränkt (oder fleißig wechseln!).

EBCDIC Extended BCD Interchange Code 8-Bit Code, der von IBM und Siemens (!) häufig verwendet wird. In diesem Code hat die Buchstabenliste A..Z leider Lücken.

EPROM Erasable PROM = löschbares PROM

Festplatte Bezeichnung für ein Speicherplattensystem. Die Platte kann aber nicht gewechselt werden. Da Festplattensysteme meist aus einem Plattenstapel von mehreren Platten bestehen, ist meist genügend Platz vorhanden, so daß ein Wechseln nur in Ausnahmefällen notwendig ist.

Harddisk (siehe Festplatte)

Firmware sind Programme, die Teile der Hardware softwaretechnisch ersetzen. (manchmal auch in EPROMs geliefertes Betriebssystem des Rechners)

Hardcopy 1:1-Wiedergabe des Bildschirminhalts auf den Drucker.

Hardware ist der physikalische Teil eines Rechners (Platine, alle Bauteile, Drucker, Laufwerke usw.)

Interface Schnittstelle, genormte Steckverbindung. Alle Steckverbindungen einer Norm passen (oder sollten passen) zueinander. Normen sind: Centronics (parallel, Drucker), RS-232 (seriell), V24 (seriell),...

I/O Input/Output = Eingabe und Ausgabe (E/A)

kompatibel kompatibel sind Geräte zu einem Gerät eines anderen Herstellers, wenn die Programme, die auf dem einen Rechner laufen, auch auf dem anderen laufen. Auch Peripheriegeräte können kompatibel zum Hauptgerät sein, wenn sie ohne Änderung daran laufen.

PROM Programmable ROM = programmierbarer ROM

QWERTY Amerikanische Tastaturbelegung, im Gegensatz zur deutschen: QWERTZ. Das Y und das Z sind dabei vertauscht. Auch fehlen die Umlaute und "ß". Dafür sind Ä,Ü,Ö,ä,ü,ö,ß vorhanden.

ROM Read Only Memory = Lese-Speicher

RAM Random Access Memory = Schreib-Lese-Speicher

Software sind alle Programme (Betriebssystem, Compiler, Anwenderprogramme usw.)

Spaghetticode nicht sehr übersichtlicher Programmierstil, der besonders in BASIC und FORTRAN leicht zu erreichen ist.

Streamer Bandgerät dient zur Datensicherung in Verbindung mit einem (->) Winchesterlaufwerk.

Subroutine = Unterprogramm

TRACE Fehlerverfolgung, in BASIC-Interpretern meist als Befehl eingebaut. Listet die durchlaufenen Zeilen.

Anhang K Warenschutz

Apple II+ ist geschütztes Warenzeichen von Apple Computer, Inc.

CP/M ist geschütztes Warenzeichen von Digital Research, Inc.

IBM ist geschütztes Warenzeichen von International Business Machines Corp.

MS ist geschütztes Warenzeichen von Microsoft Corp.

UNIX ist geschütztes Warenzeichen von Bell Labs.

6502 ist geschütztes Warenzeichen von MOS Technology

Z80 ist geschütztes Warenzeichen von Zilog Corp.

8080, 8088 sind geschützte Warenzeichen von Intel Corp.

Anhang L Literaturverzeichnis

Brauch: Programmierung mit BASIC, Stuttgart 1981, Teubner Studienskripten

Burger: Das große Computer-Viren Buch, Düsseldorf 1987, DATA BECKER

Commodore Büromaschinen GMBH: Commodore Diskettenlaufwerk 1581, Bedienungshandbuch, Frankfurt 1986

Commodore Electronic Ltd.: 1551 disk drive User's Guide

Ellinger: Commodore 1571 & 1570, Das große Floppybuch, Düsseldorf 1986, Data Becker (2.Auflage)

Ellinger, Englisch, Gelfand, Szczepanowiski: Das große Floppy Buch zur 1541; Düsseldorf 1986, DATA BECKER,

Herbert: The Tragedy of Hamlet, Reference Manual

Jensen/Wirth: Pascal User Manual and Report, Springer 1985 (3. Edition)

Schönleber: Hitchhacker's Guide to BASIC, Kiel 1987, Verlag Claus Schönleber

Vallee: Computernetze, Träume und Alpträume von einer neuen Welt, Rowohlt 1984

Wirth: Systematisches Programmieren, Stuttgart 1985, 3.Auflage

Wirth: Algorithmen und Datenstrukturen, Stuttgart 1983, 3. Auflage

Stichwortverzeichnis

DAS STEHT DRIN:

Nutzen Sie alle Möglichkeiten, die Ihnen die Commodore-Floppies zur Verfügung stellen. Dabei ist es völlig gleichgültig, ob Sie die 1541, 1541-II, 1570/71 oder die 1581 besitzen. In diesem Buch lernen Sie, Ihre Floppystation optimal einzusetzen. Anhand vieler Beipiele werden die verschiedenen Anwendungsmöglichkeiten gezeigt, die nicht nur für die Profis unter Ihnen interessant sind. Ein umfangreiches Kapitel mit Tips & Tricks zu Ihrer Floppy rundet das Buch ab.

Aus dem Inhalt:

- Einführung für Einsteiger
- Die unterschiedlichen Dateitypen
- Programmierbetrieb in BASIC
- Sequentielle und relative Dateiverwaltung zum Abtippen
- Die ISAM-Datei
- Blockbefehle und direkter Diskettenzugriff am Beispiel eines einfachen Kopierprogramms und eines Diskettenmonitors
- Daten von defekten Disketten retten
- Viele Tips & Tricks zur Floppy

UND GESCHRIEBEN HAT DIESES BUCH:

Claus Schönleber ist Student der Mathematik und Informatik mit großer Programmiererfahrung auf den verschiedensten Computersystemen. Durch seine langjährige Mitarbeit an der Kieler Volkshochschule, wo er unter anderem BASIC unterrichtet, ist er in der Lage, auch schwierige Themen leichtverständlich darzustellen.

ISB N 3-89011-269-2 DM +029.00

DM 29,–
ÖS 226,–
sFr 27,–

DATA BECKER

9 783890 112695 02900