David Lawrence
Mark England

Commodore Sachbuchreihe Band 16

FLOPPY-PROGRAMMIERUNG MIT DEM COMMODORE 64

Ein Leitfaden für die erfolgreiche Programmierung auf der Floppy Disk 1541.

INHALT

Kapitel 1 11
Disketten und Diskettenlaufwerke
Einleitung – Der Aufbau einer Diskette – Das Diskettenlaufwerk

Kapitel 2 19
Die Inbetriebnahme
Der Anschluß der Laufwerkes – Das Einschalten – Der Umgang mit Disketten – Das Einlegen und Herausnehmen von Disketten – Probleme während des Betriebs – Das Ausschalten

Kapitel 3 27
Das Abspeichern und Laden von Programmen
Wie oft sollten Programme abgespeichert werden – Der SAVE- und LOAD-Befehl mit einer Floppy Disk – Das Abspeichern und Laden mit mehreren Laufwerken – Eine einfache Methode zur Vereinfachung des Abspeicherns eines Programms – Der Gebrauch von VERIFY – Das Überschreiben von Dateien mit " @ 0:" – Die Sicherung des Abgespeicherten

Kapitel 4 33
Die System-Befehle
Einleitung – OPEN – CLOSE – PRINT# – NEW – SCRATCH – RENAME – COPY – INITIALIZE – VALIDATE

Kapitel 5 61
Pattern Matching
Was ist "pattern matching" – Zeichenfolgen mit "*" – Zeichenfolgen mit "?" – Die Kombination von "*" und "?" – Die Anwendung des "pattern matching" mit Befehlen

Kapitel 6 65
Der Fehlerkanal
Was ist der Fehlerkanal – Das Lesen des Fehlerkanals – Der Gebrauch des Fehlerkanals – Zusammenfassung des Fehlerkanals

Kapitel 7 71
Sequentielle und User-Dateien
Was ist eine sequentielle Datei – Das Öffnen einer sequentiellen Datei mit OPEN – Die Ein- und Ausgabe von Daten: PRINT# und INPUT# – GET# – Das Herausfinden des Dateiendes – Das Beenden der Aus- oder Eingabe mit CLOSE – Anwendungsbeispiele für die sequentielle Datei

Kapitel 8 91
Programm-Dateien
Was ist eine Programm-Datei – Der Aufbau einer BASIC-Programm-Datei – Der Gebrauch von Programm-Dateien für andere Zwecke – Die Ausgabe von Programm-Dateien zum Drucker – Das Aneinanderhängen von Programmen mit Hilfe von Programm-Dateien auf der Diskette – Das Neunumerieren einer Programm-Datei auf der Diskette

Kapitel 9 113
Relative Dateien
Einleitung – Das Erstellen einer relativen Datei – Das Öffnen einer relativen Datei – Positionsbestimmung in einer relativen Datei – Das Schreiben in eine relative Datei – Das Lesen aus einer relativen Datei – Das Schließen einer relativen Datei – Die Anwendung relativer Dateien

Kapitel 10 127
Direktzugriffs-Dateien
Das Öffnen einer Direktzugriffs-Datei mit OPEN – Die Lage von Daten im Puffer – Das Schreiben von Daten in den Puffer – Das Schreiben von Daten auf die Diskette – Das Laden von Daten von der Diskette in den Puffer – Das Zurückholen von Daten in den C 64 – Das Belegen und Freisetzen von Sektoren auf der Diskette – Das Ausführen von Maschinensprache von der Diskette – Zwei Dienstleistungsprogramme für die Direktzugriffs-Dateien

Kapitel 11 143
Das Directory der Diskette
Das Format des Directorys – Das Lesen des Directorys – Die Anwendung einer Operation auf mehrere Dateien

Kapitel 12 153
Befehle für die Programmierung in Maschinensprache
Das Lesen des Laufwerkspeichers – Das Hineinschreiben in den Speicher – Die Anwendung von Maschinensprache im Laufwerkspeicher

Kapitel 13 161
Das Ändern der Gerätenummer

Anhang A: Die Fehlermeldungen der Floppy Disk 163

Anhang B: Zusätzliche Maschinensprachbefehle 166

Anhang C: Die DOS-5.1-Befehle 167

BEMERKUNGEN ZU DEN PROGRAMMLISTINGS

Die in eckigen Klammern beschriebenen Steuerzeichen in den Programmzeilen werden wie folgt eingegeben:

BEZEICHNUNG	SYMBOL	TASTEN	ODER	CHR$(XXX)
[SCHIRM NEU]	[CS]" "	=	<SHIFT>+<CLR>	CHR$(147)
[HOME]	" "	=	<CLR>	CHR$(19)
[CRSR HOCH]	[CU]" "	=	<SHIFT>+<CRSR>	CHR$(145)
[CRSR RUNTER]	[CD]" "	=	<CRSR>	CHR$(17)
[CRSR LINKS]	[CL]" "	=	<SHIFT>+<CRSR>	CHR$(157)
[CRSR RECHTS]	[CR]" "	=	<CRSR>	CHR$(29)
[SCHWARZ]	[BLK]" "	=	<CTRL>+<1>	CHR$(144)
[WEISS]	[WHT]" "	=	<CTRL>+<2>	CHR$(5)
[ROT]	[RED]" "	=	<CTRL>+<3>	CHR$(28)
[CYAN]	[CYN]" "	=	<CTRL>+<4>	CHR$(159)
[PURPUR]	[PUR]" "	=	<CTRL>+<5>	CHR$(156)
[GRUEN]	[GRN]" "	=	<CTRL>+<6>	CHR$(30)
[BLAU]	[BLU]" "	=	<CTRL>+<7>	CHR$(31)
[GELB]	[YEL]" "	=	<CTRL>+<8>	CHR$(158)
[REVERS EIN]	[REV]" "	=	<CTRL>+<9>	CHR$(18)
[REVERS AUS]	[RVO]" "	=	<CTRL>+<0>	CHR$(146)
[ORANGE]	" "	=	<C=>+<1>	CHR$(129)
[BRAUN]	[BROWN]" "	=	<C=>+<2>	CHR$(149)
[HELLROT]	[LT.RED]" "	=	<C=>+<3>	CHR$(150)
[GRAU 1]	[GREY 1]" "	=	<C=>+<4>	CHR$(151)
[GRAU 2]	[GREY 2]" "	=	<C=>+<5>	CHR$(152)
[HELLGRU]	[LT.GRN]" "	=	<C=>+<6>	CHR$(153)
[HELLBLAU]	[LT.BL]" "	=	<C=>+<7>	CHR$(154)
[GRAU 3]	[GREY 3]" "	=	<C=>+<8>	CHR$(155)

<CTRL> BEDEUTET 'CONTROL-TASTE'.
<C=> BEDEUTET 'COMMODORE-TASTE'.

EINLEITUNG

Das Erscheinen von billigen, zuverlässigen Diskettenlaufwerken für den Hausgebrauch gleicht einer Revolution für die Anwendungsmöglichkeiten eines Heimcomputers. Jeder Computer, ob groß oder klein, wird sehr viel leistungsfähiger, sobald er Daten schneller erfassen kann, als es mit dem Schneckentempo eines Kassettenlaufwerks möglich ist. Ohne die schnelle Massenspeicherung würden umfangreiche Programme zu ermüdenden Lasten, die bis zu 15 Minuten Ladezeit brauchten. Der Umgang mit den Daten würde wegen der ständigen Notwendigkeit, Kassetten zurückzuspulen oder auszuwechseln, und des Wartens während die Daten langsam dem Speicher zugeführt werden, zu einem Alptraum. Kurz, der beste Computer ist nur so gut wie das Gerät, auf dem er Daten speichert. Das Diskettenlaufwerk ist kein Luxus, es ist eine wichtige Komponente eines effektiven Mikrocomputersystems.

Nirgendwo trifft das mehr zu, als beim Commodore 64. Der Erfolg des C 64 als ein Rechner, der nicht nur eine Spielmaschine oder ein Computerspielzeug ist, beruht zu einem großen Teil darauf, daß Commodore ein Diskettenlaufwerk produzierte, das den höchsten Ansprüchen genügt und trotzdem in einer Preislage liegt, die jeder, der seine Computerei ernst nimmt, akzeptieren kann.

Aber einfach eine Floppy Disk zu kaufen, ist noch nicht die Lösung des Problems, den Anwendungsbereich zu vergrößern. Genau wie jedes andere Laufwerk, hat auch das 1541-Laufwerk seine eigene Arbeitsweise und Eigenarten, die, vom Benutzer nicht beachtet, einige Risiken bergen. Schlecht durchdachtes Umgehen mit der Diskette bedeutet oft, daß ein großer Teil ihres Leistungsvermögens ungenutzt bleibt. Unachtsamkeit oder mangelndes Verstehen kann den unwiederbringlichen Verlust wertvoller Daten und Programme zur Folge haben. Und dennoch ist die 'Floppy' 1541 ein gut durchdachtes Gerät, das voller Eigenschaften steckt, von denen viele Disk-Drive-Besitzer nur träumen können.

Dieses Buch ist ein Versuch unsererseits, einige der vielen Entdeckungen, die wir bei unseren Floppy Disks gemacht haben, die Freude, die beim Umgang mit ihr aufkommt und die Leistungsfähigkeit, die sie bietet, wenn die Floppy Disk mit Geschick eingesetzt wird, mit Ihnen zu teilen.

ANMERKUNG BEZÜGLICH DER BENUTZUNG DIESES BUCHES

Dieses Buch wurde speziell für Commodore-64-Besitzer geschrieben, die mit einem oder mehreren 1541-Laufwerken arbeiten. Die meisten der in diesem Buch erwähnten Befehle und Techniken können jedoch auch von Benutzern des älteren 1540-Laufwerkes und/oder des Computers VC 20 angewendet werden. Es muß aber gesagt werden, daß das hier präsentierte Material nicht auf den gerade genannten Geräten ausprobiert wurde. Es werden auch Stellen vorkommen, wo die beschriebenen Einrichtungen für diese nicht verfügbar sind.
Was den VC 20 betrifft, muß, wie im 1541-Bedienungshandbuch beschrieben, die Geschwindigkeit, mit der die 1541 läuft, mit dem "UI-"-Befehl angepaßt werden. Benutzer des 1540-Drives werden, laut 1540-Bedienungshandbuch, die in Kapitel 9 beschriebenen Techniken für relative Dateien nicht einsetzen können.

KAPITEL 1

DISKETTEN UND DISKETTENLAUFWERKE

1. *Einleitung*
2. *Der Aufbau einer Diskette*
3. *Das Diskettenlaufwerk*

1. GRUNDLAGEN DER MAGNETISCHEN PLATTENSPEICHERUNG

Die Plattenspeicherung beruht wie die Bandspeicherung auf der Tatsache, daß die dünne Schicht einer ferromagnetischen Verbindung geeignet ist, magnetisiert und entmagnetisiert zu werden. Wird eine solche Verbindung durch Annäherung an ein magnetisches Feld magnetisiert, ist es in der Lage, diesen magnetisierten Zustand aufrechtzuerhalten.

In der Praxis ist es unwichtig, daß Magnetismus angewendet wird (außer, Sie haben die Angewohnheit, Magnete auf Ihre Disketten zu legen). Es ist jedoch wichtig, daß solche Eisenverbindungen auf die eine oder andere Art registrieren können, daß etwas auf sie eingewirkt hat – sie können Informationen aufzeichnen. Werden kleine Mengen dieser Verbindung sehr flach ausgebreitet, haben sie die Fähigkeit, z. B. das Vorbeiführen eines Elektromagneten aufzuzeichnen, die Stärke seines Magnetfeldes und die Richtung des Stromflusses in ihm. Weil ein Elektromagnet die Eigenschaft besitzt, einen elektrischen Stromfluß zu erzeugen, wenn er durch ein magnetisches Feld geführt wird, kann der dünne Film, einmal magnetisiert, von einem Elektromagneten, durch den kein Strom fließt, gelesen werden – sogar dieses winzige, von einem dünnen Film einer Eisenverbindung gespeicherte Feld. Vorausgesetzt, daß man einen Elektromagneten nahe genug über die Schicht hinwegführen kann und daß man den Zustand des Elektromagneten steuern kann, und vorausgesetzt, daß dies mit hinreichender Genauigkeit geschieht, so daß man zu ein und derselben Position immer wieder zurückkehren kann, dann können die magnetischen Eigenschaften der dünnen Schicht einer Eisenverbindung zur Speicherung von Informationen genutzt werden.

Bei einem Computer-Diskettenlaufwerk befindet sich der Film mit der Eisenverbindung auf der Oberfläche einer 5¼ Zoll großen, dünnen, biegsamen Plastikscheibe. Im Laufwerk befindet sich der Elektromagnet in Form eines sehr kleinen Aufnahmekopfes, der in der Lage ist, mit hoher Genauigkeit auf einer geraden Linie zwischen

dem Zentrum der Scheibe und deren Randgebieten hin- und herzufahren. Die Bewegung des Kopfes über den Film wird durch das Drehen der Scheibe gewährleistet.
Im wesentlichen besteht das Diskettensystem aus einer sich drehenden Scheibe (der Diskette) und einem Elektromagneten, der diese von innen nach außen und umgekehrt abtastet, während sie sich dreht.
Der Vorteil dieses Systems gegenüber dem Band liegt nicht nur in der Geschwindigkeit, mit der ein Informationsblock gespeichert werden kann – einige Bandsysteme arbeiten mit ähnlich hohen Geschwindigkeiten. Nein, die wirkliche Stärke des Diskettensystems liegt in der Geschwindigkeit, in der es eine Information oder den Platz, an der sie gespeichert werden soll, *finden* kann. Man kann diesen Unterschied zwischen Diskette und Band auch am Beispiel des Vergleichs zwischen einer Tonbandkassette und einer Langspielplatte demonstrieren. Vorausgesetzt, daß Sie in beiden Fällen wissen, wo die von Ihnen gewünschte Information gespeichert ist, d. h. welches Stück eines Albums Sie abspielen möchten, können Sie dieses bei der Schallplatte viel schneller finden, weil Sie die Nadel sofort zu der betreffenden Stelle führen. Einmal dort, ermöglicht Ihnen eine andere Bewegungsart, nämlich die Drehung der Schallplatte, Ihr Stück zu hören. Beim Bandsystem aber haben Sie nur eine Bewegungsart zur Verfügung. Sie können nur auf den Schnell-Vorwärts-Spulknopf drücken, bis die richtige Stelle erreicht ist.

2. DER AUFBAU EINER DISKETTE

Im Gegensatz zu einer LP haben die für die 1541 verwendbaren Disketten keine permanenten, individuellen Spuren. Die empfindliche Schicht ist, oder sollte über die ganze Diskettenoberfläche von gleichmäßiger Beständigkeit sein. Das Unterteilen der Diskette in leicht zu identifizierende "Spuren" für die Speicherung und das Lesen von Daten wird von dem Laufwerk in dem "Formatieren" genannten Vorgang selbst vorgenommen.
Der Zweck des Formatierens ist, die Diskette magnetisch mit einer Reihe von Sektoren zu kennzeichnen, die ungefähr ein ¾ Zoll lang sind. Genau wie bei der Langspielplatte, sind die Sektoren in Ringe unterteilt, die Spuren genannt werden. Es sind zusammen 35, wobei die Anzahl der Sektoren von der Entfernung der Spur zum Zentrum der Diskette abhängt – je weiter vom Zentrum entfernt, desto länger die Spur und desto mehr Sektoren auf der Spur.
Diese simplen Vorgänge werden noch von einigen komplizierten begleitet, die es dem sehr fein abgestimmten Laufwerk ermöglichen, eine Stelle auf der Diskette zu identifizieren und den Aufnahmekopf dorthin zu bewegen. Jeder Sektor besitzt 256 Bytes für die Datenspeicherung, enthält aber zusätzlich noch eine andere Information, z. B. die Identifikationsnummer der Diskette, die Nummer des Sektors innerhalb der Spur und noch einige Standarddaten, die die Floppy Disk später gebrauchen wird, um die Synchronisation der rotierenden Diskette zu überprüfen.

Abgesehen von den leeren Sektoren, die für die Aufnahme von Daten vorgesehen sind, gibt es noch eine andere Stelle auf der Diskette (die Spur 18), die für das Directory[1]) oder die Liste der Dateien, die sich auf der Diskette befinden werden, zuständig ist.
Beim ersten Formatieren der Diskette werden nur die ersten beiden Sektoren der Spur 18 für diesen Zweck gebraucht – andere Sektoren werden dann belegt, wenn die Programme hinzugefügt werden. Innerhalb des Directorys gibt es einen Bereich für Verwaltungsinformationen, die sogenannte "Block Allocation Map". Der Zweck der BAM ist es, für jeden Sektor der Diskette festzuhalten, ob dieser für die Speicherung von Informationen zur Verfügung steht, oder ob er von einer bereits existierenden Datei belegt ist.
Die BAM liegt im ersten Sektor (Sektor 0) der Spur 18 und besteht aus 140 Bytes Diskettenspeicherplatz. Dieser Platz ist noch einmal in 35 Blöcke zu 4 Bytes unterteilt. Das erste Byte der Gruppe zeigt die Anzahl der Sektoren an, die in einer der 35 Spuren der Diskette zur Verfügung stehen. Die nächsten drei Bytes zeichnen den individuellen Zustand der Sektoren 0–7, 8–16 und 17–23 der entsprechenden Spur auf. Ist z. B. der Sektor Null einer bestimmten Spur für die Speicherung frei, wird das Bit 0 des im zweiten der vier Bytes gespeicherten Wertes gesetzt (also gleich 1 anstelle von 0). Ist der Wert des ganzen Bytes (d. h. 8 Bits) 0, so daß also kein Bit gesetzt ist, würde das bedeuten, daß alle acht Sektoren, die es kontrolliert, von einer bestehenden Datei belegt sind. Behalten Sie bitte im Auge, daß die BAM Vorkehrungen zur Aufnahme von 24 Sektoren (0–23) pro Spur trifft, obwohl es höchstens 21 auf den längeren äußeren Spuren gibt. Die BAM meistert diese Schwierigkeit, indem sie die nicht-existierenden Sektoren als nicht verfügbar registriert, wenn die Diskette formatiert wird.
Aus **Tabelle 1.1** geht hervor, daß noch ein Sektor der Spur 18 zu dem Directory hinzugefügt werden muß, wenn sich mehr als acht Dateien auf der Diskette befinden. Dieser neue Sektor hat das gleiche Format wie der Sektor 1 in der Tabelle. Der letzte Sektor im Directory wird dadurch gekennzeichnet, daß die ersten zwei Bytes, die normalerweise die Adresse des folgenden Sektors anzeigen, auf Spur 0, Sektor 255 zeigen, der nicht existiert.

DIE STRUKTUR EINER DATEI AUF DER DISKETTE

Nachdem die Diskettenstruktur und das Ausgangs-Directory erstellt sind, ist die Diskette nun für die Speicherung von Daten in sogenannten "Files" oder Dateien bereit. Die zwei meistgebrauchten Dateiarten sind die Programm-Datei, welche erstellt wird, wenn ein Programm geSAVEt wird, und die sequentielle Datei, die

[1]) Auch *die* Directory ist gebräuchlich. Wir haben uns jedoch für *das* Directory entschieden.

erstellt wird, wenn eine noch nicht bestehende Datei mit OPEN für die Speicherung von Datenfeldern geöffnet wird.

Tabelle 1.1 Aufbau der Block Allocation Map

BYTE	ANMERKUNGEN
1	Zeigt die Anzahl der in dieser Spur verfügbaren Blocks an.
2	Sektoren 0–7; Bit 0 repräsentiert Sektor 0
3	Sektoren 8–15; Bit 0 repräsentiert Sektor 8
4	Sektoren 16–23; Bit 0 repräsentiert Sektor 16

Tabelle 1.2 Aufbau der Directory-Spur (Spur 18)

Spur 18, Sektor 0

BYTE	ANMERKUNGEN
0	Spur des nächsten Directory-Blocks (immer Spur 18)
1	Sektor des nächsten Directory-Blocks (Sektor 1)
2	Steht hier der Wert 65, wurde die Diskette für die 1541 formatiert
3	Unbenutzt; enthält normalerweise den Wert 0
4	Erstes Byte der BAM; enthält die Anzahl der in Spur 1 verfügbaren Sektoren
5	Spur 1, Sektoren 0–7 der BAM
6	Spur 1, Sektoren 8–16 der BAM
7	Spur 1, Sektoren 17–23 der BAM
8	Anzahl der in Spur 2 verfügbaren Sektoren
9	Spur 2, Sektoren 0–7 der BAM
10	Spur 2, Sektoren 8–16 der BAM
11	Spur 2, Sektoren 17–23 der BAM
	. . . . .
143	Spur 35, Sektoren 17–23 der BAM (dies ist das letzte Byte der BAM)
144–161	Name der Diskette ergänzt mit "SHIFT SPACEs" (CHR$(160))
162	Erstes Byte der Disketten-ID
163	Zweites Byte der Disketten-ID
164	Unbenutzt; enthält normalerweise den Wert 160
165	Zeichen "2" zeigt die DOS-Version an
166	Zeichen "A" zeigt die DOS-Version an
167	Unbenutzt; enthält normalerweise den Wert 160
168–170	Unbenutzt; Wert unbestimmt

Spur 18, Sektor 1

BYTE	ANMERKUNGEN
0	Spur des nächsten Directory-Sektors (normalerweise 18, aber 0 wenn das Ende des Directories erreicht ist)
1	Sektor des nächsten Directory-Sektors (255 bedeutet das Ende des Directories)
2–31	Eintrag der 1. Datei – Details in Kapitel 11
32	Unbenutzt
33	Unbenutzt
34–63	Eintrag der 2. Datei
64	Unbenutzt
65	Unbenutzt
66–95	Eintrag der 3. Datei
96	Unbenutzt
97	Unbenutzt
98–127	Eintrag der 4. Datei
128	Unbenutzt
129	Unbenutzt
130–159	Eintrag der 5. Datei
160	Unbenutzt
161	Unbenutzt
162–191	Eintrag der 6. Datei
192	Unbenutzt
193	Unbenutzt
194–223	Eintrag der 7. Datei
224	Unbenutzt
225	Unbenutzt
226–255	Eintrag der 8. Datei

Beide Arten werden auf die gleiche Art und Weise auf der Diskette gespeichert, weswegen wir das SAVEn einer normalen Programm-Datei als Beispiel nehmen.

DIE REIHENFOLGE DES VORGEHENS BEIM SAVEN EINER PROGRAMM-DATEI:

1. Der SAVE-Befehl wird vom Benutzer eingegeben, worauf der C 64 das Laufwerk instruiert, eine Programm-Datei mit diesem Namen zu öffnen.

2. Das Laufwerk überprüft das Directory, um zu sehen, ob nicht schon eine Datei mit diesem Namen existiert.

3. Vorausgesetzt, daß es keine Datei mit demselben Namen gibt, nimmt die Floppy Disk den Dateinamen mit der Ausgangsspur und dem Sektor 0, 255 – d. h. in eine nicht existierende Spur – in das Directory auf.

4. Mit Hilfe der BAM, welche sich ständig im Laufwerkspeicher befindet, beginnt das Laufwerk nach der der Directory-Spur folgenden Spur zu suchen, die einen freien Sektor hat – und zwar entweder nach außen zum Rand der Diskette hin oder nach innen zur Mitte und kennzeichnet diesen Sektor als der BAM zugeteilt.

5. Sobald der SEKTOR 1 gefunden ist, zeichnet die Floppy Disk dessen Position auf und ist dann bereit, 254 Bytes des Programms vom C 64 zu empfangen, die es in einem Puffer des Laufwerkspeichers ablegt.

6. Nun wird die Suche nach einem weiteren freien Sektor (SEKTOR 2) aufgenommen.

7. Die Adresse des SEKTOR 2 (aus Schritt 6) wird in die ersten beiden Bytes des im Schritt 5 erstellten Puffers geschrieben.

8. Der gesamte Inhalt des Puffers wird nun in den SEKTOR 1 (aus Schritt 4) geschrieben.

9. Der SEKTOR 2 wird jetzt als SEKTOR 1 betrachtet. Der Vorgang wird von Schritt 5 an wiederholt, bis der C 64 das Laufwerk informiert, die Datei zu schließen. Dies geschieht, nachdem das ganze Programm übertragen wurde.

10. Für die letzte Portion Daten wird die Adresse des nächsten Sektors (die ersten beiden Bytes) in die Spur Null, Sektor 255 geschrieben, um das Ende der Datei anzuzeigen.

11. Der Directory-Eintrag für die neue Datei wird geändert, um den für den Anfang der Datei und für die Anzahl der in der Datei beinhalteten Bytes gebrauchten Sektor aufzunehmen.

Tabelle 1.3 Belegung der Sektoren auf einer typischen Spur

Spur = 15		
0	SEQ. F	,PRGBLOCK 2
1	EOF. F	,PRGBLOCK 2
2	LIST T&S. F	,PRGBLOCK 14
3	APRICOT. F. WOW	,SEQBLOCK 1

4	LIST T&S. F	,PRGBLOCK 12
5	SEQ ARRAYS. F	,PRGBLOCK 4
6	TEST1	,SEQBLOCK 1
7	SCREEN SAVE	,PRGBLOCK 1
8	PROG READ	,PRGBLOCK 1
9	SCREEN	,PRGBLOCK 1
10	SEQ. F	,PRGBLOCK 3
11	EOF. F	,PRGBLOCK 3
12	SEQ ARRAYS. F	,PRGBLOCK 3
13	SEQ ARRAYS. F	,PRGBLOCK 5
14	LIST T&S. F	,PRGBLOCK 13
15	SCREEN	,PRGBLOCK 3
16	LIST T&S. F	,PRGBLOCK 11
17	SCREEN	,PRGBLOCK 4
18	PROG READ	,PRGBLOCK 2
19	SCREEN	,PRGBLOCK 2
20	SEQ. F	,PRGBLOCK 4

Wenn im Laufe der Zeit Programme auf die Diskette geschrieben, wieder entfernt oder überschrieben werden, erscheint der Aufbau der Diskette dem menschlichen Auge bestimmt recht verworren. Sie enthält auf jeder Seite einen Sektoren-Mischmasch, der aus einer Vielzahl von Programmen besteht. Vorausgesetzt, daß nichts passiert ist, was das Directory oder die zwei am Anfang eines jeden Sektors stehenden Bytes, die die Position des nächsten Sektors der Datei enthalten, stört, wird die Floppy Disk einen Dateianfang immer finden und die Datei Sektor für Sektor ohne Schwierigkeit lesen können. Ein Beispiel für den Aufbau einer vielgebrauchten Diskette wird in **Tabelle 1.3** dargestellt. Es ist der Ausdruck eines Programms, daß in Kapitel 10 dieses Buches zu finden ist. Die Tabelle stellt den Inhalt einer einzigen Spur auf einer der im Verlauf dieses Buches angelegten Disketten dar.

3. DAS DISKETTENLAUFWERK

Bis jetzt haben wir uns nur die Disketten und deren Aufbau angeschaut, wobei wir die Tätigkeit der 1541 als selbstverständlich betrachteten. Es wäre falsch, dieses Kapitel zu beschließen, ohne daran zu erinnern, daß die 1541 ein sehr kompliziertes und leistungsfähiges Gerät ist. Es wird von einem 6502-Mikroprozessor betrieben, und das eigene interne Disk Operating System (DOS)-Programm ist so umfangreich wie der ROM des C 64. Der Vorteil ist, daß, anders als bei der Mehrzahl der Mikrocomputer, für den Betrieb der 1541 kein Speicherplatz verlorengeht. Die 1541 muß sich nicht darauf verlassen, vom C 64 detaillierte Anweisungen für die Ausfüh-

rung der Aufgaben zu erhalten, sondern braucht nur den Namen des Befehls, den sie ausführen soll, mitgeteilt zu bekommen. Sie wird dann ohne weitere Hilfe fortfahren, einen der komplexen Vorgänge auszuführen. Aus diesem Grunde gehört die 1541 zu den sogenannten "intelligenten" Diskettenlaufwerken.

KAPITEL 2

DIE INBETRIEBNAHME

1. *Der Anschluß des Laufwerkes*
2. *Das Einschalten*
3. *Der Umgang mit Disketten*
4. *Das Einlegen und Herausnehmen von Disketten*
5. *Probleme während des Betriebs*
6. *Das Ausschalten*

1. EINE ANLEITUNG ZUM ANSCHLUSS DES LAUFWERKS FÜR ANFÄNGER

Um ein Diskettensystem an Ihren Computer anzuschließen und zu betreiben, brauchen Sie mindestens ein 1541-Laufwerk und das dazugehörende Verbindungskabel. Außerdem benötigen Sie ein paar Disketten. Diese kauft man normalerweise im Zehnerpack; sie sind aber auch einzeln erhältlich. Sie sollten nicht vergessen, eine zusätzliche Steckdose für die Stromversorgung Ihres Laufwerkes bereitzuhalten.
Vorausgesetzt, daß Ihr C 64, sein Netzteil und der Fernseher/Monitor, den Sie üblicherweise benutzen, korrekt miteinander verbunden und alle Geräte ausgeschaltet sind, folgen Sie dieser Anleitung:

1. Drehen Sie den C 64 so herum, daß Sie die Rückseite vor sich haben.

2. Von rechts nach links schauend, finden Sie zwei rechteckige Schlitze, durch die Sie den Rand der Platine des C 64 erkennen können.

3. Links von diesen beiden Schlitzen befinden sich zwei runde Steckbuchsen. Die rechte ist eine 6polige Buchse, der sogenannte "Serielle Bus". Dies ist die Einrichtung, über die der C 64 mit externen Geräten wie dem Drucker und dem Laufwerk kommuniziert.

4. Ist an dieser Buchse bereits ein Drucker angeschlossen, so ziehen Sie erstmal den Anschlußstecker zum Drucker heraus.

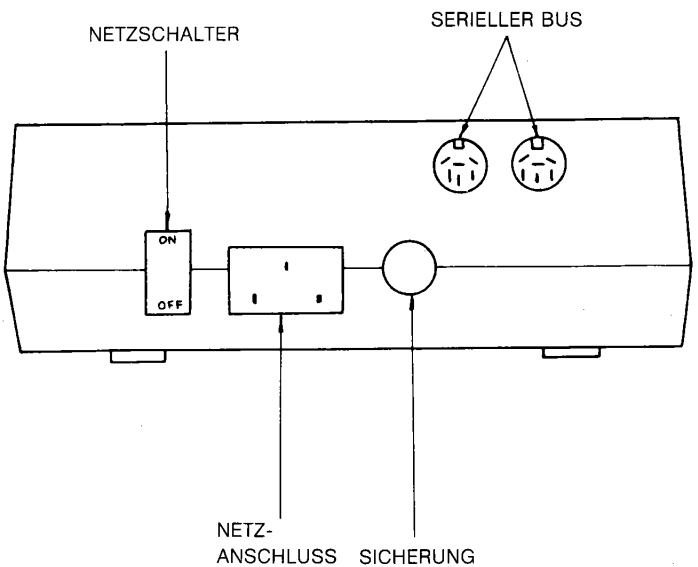
5. Stecken Sie jetzt den DIN-Stecker der Leitung, die vom Laufwerk kommt, in die Buchse des seriellen Busses.

6. Nun können Sie den C 64 herumdrehen, so daß Sie wieder auf die Tastatur blicken.

7. Nehmen Sie das Laufwerk, und stellen Sie es so neben den C 64, daß Sie auf die Rückseite schauen.

8. Auf der Rückseite der 1541 finden Sie die in **Abbildung 2.1** dargestellten Buchsen.

Abbildung 2.1 Die Rückseite der 1541 Floppy Disk



9. Nehmen Sie nun das andere Ende des Verbindungskabels, und stecken Sie es in eine der beiden, in der Illustration mit "SERIELLER BUS" bezeichneten Buchsen.

10. Besitzen Sie ein zweites Laufwerk, stecken Sie dessen Verbindungskabel in die andere Buchse. (Wollen Sie mehr als eine Floppy Disk betreiben, sollten Sie zusätzlich das Kapitel 13 lesen, in dem beschrieben wird, wie man die Gerätenummer eines der beiden Laufwerke ändert, falls diese nicht schon modifiziert wurden.) Auf diese Weise kann eine ganze Reihe von Laufwerken hintereinandergeschaltet werden.

11. Besitzen Sie einen Commodore-kompatiblen Drucker, können Sie diesen an der freien Buchse für den seriellen Bus des letzten Laufwerkes anschließen.

12. Vergewissern Sie sich, daß die 1541, der C 64, der Fernseher/Monitor (und der eventuell angeschlossene Drucker) alle ausgeschaltet sind.
Stecken Sie nun das andere Ende des Stromversorgungskabels in die Buchse (NETZ-ANSCHLUSS) auf der Rückseite des Laufwerkes und das andere Ende in die Steckdose. Schalten Sie aber zu diesem Zeitpunkt noch keines der Geräte ein.

13. Drehen Sie die 1541 herum, so daß Sie die Vorderseite anschauen. Dabei sollten Sie beachten, keine Kabel unter der Floppy Disk einzuklemmen.

14. An der Vorderseite der 1541 sehen Sie einen kleinen Riegel hervorstehen. Dieser ist in einer der Positionen:

a) etwas unterhalb des quer über die Front des Laufwerkes verlaufenden Schlitzes, oder

b) etwas oberhalb.

15. Ist der Riegel in Position a), drücken Sie ihn vorsichtig mit einem Finger herunter und erlauben ihm, sich langsam, von seiner Feder gezogen, nach oben zu bewegen.

16. Nun ist die Klappe der Floppy Disk geöffnet. Um absolut sicher zu gehen, daß sich weder eine Diskette, noch das quadratische Stück Pappe, das zum Schutz der Mechanik während des Transportes dient, in dem Laufwerk befindet, schließen Sie die Klappe durch sanftes Herunterdrücken des Riegels bis zum Einrasten, dann öffnen Sie sie wieder.

17. Kommt eine Diskette oder die Pappe zum Vorschein, ziehen Sie sie vorsichtig heraus.

18. Schalten Sie die Floppy Disk (und den eventuell angeschlossenen Drucker) ein. Sowohl das grüne als auch das rote Lämpchen auf der Vorderseite der Floppy

Disk werden aufleuchten, das Laufwerk wird ungefähr eine Sekunde lang surren, dann wird das Surren aufhören und das rote Lämpchen ausgehen. Erlischt das rote Lämpchen nicht, schalten Sie das Laufwerk aus, prüfen die Verbindungen und wiederholen dann diesen Schritt noch einmal. Leuchtet das rote Lämpchen immer noch, sollten Sie Ihren Händler aufsuchen.

19. Schalten Sie Ihren C 64 ein. (Sie sollten dieselben Vorgänge feststellen, die unter Schritt 18 beschrieben wurden.)

20. Schalten Sie Ihren Fernseher/Monitor ein, und stimmen Sie ihn auf den Ausgang des C 64 ab.

21. Jetzt können Sie damit beginnen, Ihr Commodore-64-Disketten-System zu benutzen.

2. DAS EINSCHALTEN DES SYSTEMS, NACHDEM ES INSTALLIERT WURDE

Die empfohlene Reihenfolge des Einschaltens für das Diskettensystem, nachdem es einmal richtig installiert wurde, lautet:

Drucker→ Floppy Disk→Computer

3. DER UMGANG MIT DISKETTEN

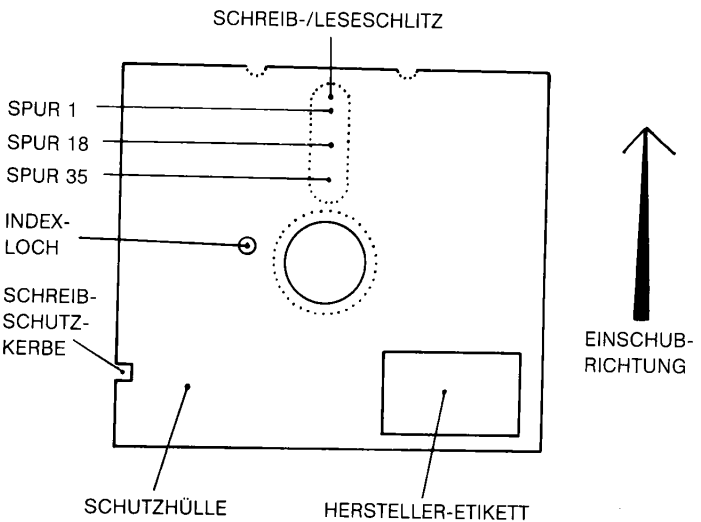
Abbildung 2.2 stellt den Aufbau einer typischen 5¼-Zoll-Diskette dar.

Solche Disketten werden Ihnen manchen Dienst erweisen, wenn Sie ein paar einfache Regeln befolgen:

1. Die Diskette darf niemals aus ihrem Schutzumschlag herausgenommen werden. Einwandfreie Rotation ist auch innerhalb dieses Schutzumschlages gewährleistet. Die Diskette wird durch den Schreib-Lese-Schlitz gelesen.

2. Disketten sind nicht dazu geschaffen, vorsätzlich verbogen oder geworfen zu werden. Wird eine Diskette einmal aus Versehen für einen Moment verbogen, sollte sie dies eigentlich überleben; es gibt jedoch keine Garantie. Beim Einlegen und Entnehmen der Diskette aus dem Laufwerk wird sie immer leicht verbogen. Davor brauchen Sie keine Angst zu haben, Sie sollten es bloß nicht übertreiben.

Abbildung 2.2 Eine 5¼-Zoll-Diskette



3. Ihre Diskette hat außerdem noch eine Papierhülle, die den größten Teil der Diskette bedeckt und den Schreib-Lese-Schlitz schützt. Nach Gebrauch sollten Sie Ihre Disketten immer in die Papierhülle zurückstecken. Bewahren Sie Ihre Disketten am besten in einer (nicht magnetischen) Plastik-Box auf, die es speziell für diesen Zweck zu kaufen gibt. Lassen Sie eine Diskette nie ohne ihre Papierhülle herumliegen.
Oft lassen Programmierer Disketten in dem Glauben ungeschützt herumliegen, daß solange man den Schreib-Lese-Schlitz auf der Oberseite der Diskette (die mit den Klebeetiketten) nicht berührt, alles o.k. sei. Da die 1541 die Disketten aber von der Unterseite liest und schreibt, kann dies ein folgenschwerer Irrtum sein.

4. Disketten sollten nie in die Nähe magnetischer Felder geraten, d. h. auch, daß man sie nicht auf dem Laufwerk oder dem Fernseher/Monitor liegen lassen darf.

5. Disketten sollten nie extremen Temperaturen oder Feuchtigkeit ausgesetzt werden, dies schließt auch das Liegenlassen in direktem Sonnenlicht ein.

6. Beschreiben Sie die Etiketten entweder *bevor* Sie sie auf die Disketten kleben, oder gebrauchen Sie einen Filzschreiber, der keinen Druck durch den Schutzumschlag hindurch auf die Diskette ausübt.

7. Eigentlich sollte man nicht mehr erwähnen müssen, daß Sie unter keinen Umständen den Schreib-Lese-Schlitz der Diskette berühren dürfen.

8. Billige Disketten können genauso wie billige Bänder zum Desaster führen. Nur Sie selbst können entscheiden, welchen Wert Ihre Programme und Daten für Sie darstellen.

4. DAS EINLEGEN UND ENTNEHMEN DER DISKETTEN

Bevor Sie eine Diskette in das Laufwerk einlegen, versichern Sie sich, ob die Klappe geöffnet ist. Dann halten Sie die Diskette so, daß der Markenaufkleber oben ist und das Ende des Schreib-Lese-Schlitzes zu der 1541 hinzeigt. Drücken Sie die Diskette vorsichtig in den horizontalen Schlitz auf der Vorderseite des Laufwerkes hinein. Bleibt die Diskette unterwegs hängen, wenden Sie keine Gewalt an, sondern ziehen Sie sie wieder heraus und probieren es noch einmal. Sollten Sie wieder Probleme haben, überprüfen Sie, ob nicht eine Diskette im Laufwerk klemmt.
Die Diskette sollte soweit hineingeschoben werden, bis nichts mehr herausschaut und sie ohne Zuhilfenahme der Finger drinbleibt. Zum Schluß schließen Sie die Klappe – die Floppy Disk kann, bevor dies nicht geschehen ist, nicht vom C 64 angesprochen werden.
Das Entnehmen geschieht einfach, indem man die Klappe öffnet. Die Diskette rutscht dann ungefähr 3 cm heraus. Sollte die Diskette nicht erscheinen, schließen Sie die Klappe und öffnen sie dann erneut. Disketten, die sich sträuben, aus der 1541 herauszukommen, können mit den Finger herausgezogen werden, wobei auch keine Gewalt angewendet werden darf.
Sollte eine Diskette dennoch nicht herauszubekommen sein, liegt dies entweder an der Diskette (vielleicht hat sich der Aufkleber gelöst) oder am Laufwerk. Sie sollten jedoch nie mit einem Werkzeug in dem Laufwerk herumhantieren, um die Diskette zu befreien – besonders dann nicht, wenn der Netzstecker noch steckt.
Die Klappe sollte niemals geöffnet werden, während das rote Lämpchen brennt und der Motor läuft; andernfalls könnte die Diskette beschädigt werden. Bemerkenswert ist auch, daß einige Ausgaben des Laufwerk-Handbuches fälschlicherweise behaupten, daß keine Diskette entnommen werden darf, solange das grüne Lämpchen brennt. Das grüne Lämpchen ist die Netzanzeige der Floppy Disk, und der einzige Weg es zu löschen ist, das Laufwerk auszuschalten. Sie sollten dies wiederum auf gar keinen Fall tun, bevor Sie nicht die Diskette herausgenommen haben.

5. PROBLEME WÄHREND DES BETRIEBS

Genau wie bei jedem anderen kompliziert aufgebauten Gerät, kann es auch bei der 1541 trotz ihrer hohen Zuverlässigkeit Pannen geben. Zum Beispiel können Fehler beim Lesen oder Schreiben eines Programms auftreten, oder es kommt zu anderen Problemen, die die Ausführung eines Disk-Befehls verhindern. In diesem Fall wird das rote Lämpchen blinken und Sie sollten, wenn Sie sicher sind, daß nicht Sie etwas falsch gemacht haben, den Vorgang, der zu dem Fehler geführt hat, wiederholen – siehe Kapitel 6.

Es gibt auch Situationen, in denen der C 64 und das Laufwerk schlicht verweigern, Informationen untereinander auszutauschen. Dem empfohlenen Einschalt-Verfahren folgend, wird zunächst die Diskette entnommen, der C 64 und die Floppy Disk ausgeschaltet und dann in der richtigen Reihenfolge wieder eingeschaltet.

Enthält der C 64 ein wertvolles Programm, genügt es unserer Erfahrung nach, die Diskette herauszunehmen und die 1541 (sowie die anderen an den seriellen Bus angeschlossenen Geräte, wie z. B. den Drucker) auszuschalten. Nach dem Wiedereinschalten ist das Problem fast immer beseitigt. Man sollte sich jedoch, falls alles andere fehlschlägt, daran erinnern, daß der Datassette-Rekorder die Situation retten kann, indem man das Programm auf ihm speichert, bis das Laufwerk wieder funktionsfähig ist.

Es ist sehr unwahrscheinlich, daß Ihnen die 1541 Schwierigkeiten bereiten wird, wenn Sie beachten, daß es ein präzis hergestelltes Gerät ist, welches, anders als der C 64, bewegliche Teile enthält, die weder Erschütterungen noch plötzliche Stöße oder extreme Hitze (einschließlich direktem Sonnenlicht) vertragen.

6. DAS AUSSCHALTEN

Obwohl es ein kleiner Punkt ist, dem wir hier einen ganzen Abschnitt widmen, sollten Sie sich immer vergewissern, daß die Klappe der Floppy Disk geöffnet und das Laufwerk leer ist, bevor Sie es ausschalten. Eigentlich werden Disketten nur selten dadurch beschädigt, daß man Sie beim Ausschalten im Laufwerk vergißt, aber es *kann* passieren.

KAPITEL 3

DAS ABSPEICHERN UND LADEN VON PROGRAMMEN

1. *Wie oft sollten Programme abgespeichert werden?*
2. *Der SAVE- oder LOAD-Befehl mit einer Floppy Disk*
3. *Das Abspeichern und Laden mit mehreren Laufwerken*
4. *Eine einfache Methode zur Vereinfachung des Programmabspeicherns*
5. *Der Gebrauch von VERIFY*
6. *Das Überschreiben von Daten mit "@0:"*
7. *Die Sicherung des Abgespeicherten*

Das Speichern von Programmen ist meistens das erste, wozu das Laufwerk gebraucht wird. Wenn Ihnen das Programmieren Spaß macht und Sie Ihren C 64 öfter gebrauchen, dürfte es überhaupt keine Zweifel daran geben, daß der Geschwindigkeitsunterschied, mit dem Sie auf Programme zugreifen können, die höheren Kosten eines Diskettenlaufwerkes, verglichen mit einem Datassette-Rekorder, wettmacht. Trotzdem ist es immer wieder erstaunlich, wie sorglos manche mit ihren Programmen umgehen, für deren Entwicklung sie viel Zeit geopfert haben. Sie versäumen beim Entwickeln, regelmäßig einzelne Programmabschnitte abzuspeichern, überprüfen nicht, ob ein Programm richtig abgespeichert wurde, erstellen keine Kopie von wichtigen Programmen und mißbrauchen ihre Disketten, indem sie sie herumliegen lassen und den Naturelementen aussetzen. Nachfolgend sind einige einfache Regeln aufgeführt, die das Abspeichern von Programmen betreffen.

1. WIE OFT SOLLTEN PROGRAMME ABGESPEICHERT WERDEN?

Speichern Sie während der Entwicklung von Programmen regelmäßig einzelne Abschnitte ab. Wie jeder andere Mikrocomputer, kann auch der C 64 Programme verlieren, wenn es einen plötzlichen Spannungsstoß im Stromnetz gibt, jemand gegen den Netzstecker tritt oder Sie den C 64 beim Programmieren aus dem Gleichgewicht bringen. Wieviel Arbeit Sie dann zunichte machen, hängt davon ab, wie lange es her ist, seit Sie den letzten Abschnitt abgespeichert haben. Wird ein

Programm schnell eingegeben, sollten Sie nie länger als 15 Minuten tippen, ohne das Programm abzuspeichern. Werden nur Fehler in einem Programm beseitigt, also relativ wenig Veränderungen vorgenommen, können Sie den Zeitraum ruhig auf eine halbe Stunde ausdehnen.
Es kommt darauf an, wieviel Sie bereit sind zu verlieren, aber Sie können sich darauf verlassen, daß Sie früher oder später ein wichtiges Programm, dessen Eingabe Sie sehr viel Zeit gekostet hat, verlieren, wenn Sie Programme nicht regelmäßig abspeichern.

2. DER SAVE- UND LOAD-BEFEHL MIT EINER FLOPPY DISK

Um sich auf das regelmäßige Abspeichern von Programmen einlassen zu können, müssen Sie zuerst einmal den Befehl, der ein Programm auf dem Laufwerk abspeichert, kennenlernen. Haben Sie vorher mit einem Datassette-Rekorder gearbeitet, dann sind Sie sicher gewöhnt an

```
SAVE "<PROGRAMMNAME>"
```

zum Abspeichern eines Programms oder an

```
LOAD "<PROGRAMMNAME>"
```

zum Einladen eines Programms in den Speicher.
Jetzt, wo Sie eine Floppy Disk installiert haben, ändert sich die Situation etwas. Obwohl der C 64 sehr gut mit einer 1541 arbeiten kann, nahm man bei seinem Entwurf an, daß er mit dem Datassette-Rekorder betrieben wird. Diese Tatsache erlaubt dem Besitzer eines Datassette-Rekorders, ohne die Angabe einer sehr wichtigen Information auszukommen, nämlich der Nummer des Gerätes, auf dem das Programm gespeichert werden soll. Die Gerätenummer des Datassette-Rekorders ist 1. Gibt man die Anweisung

```
SAVE "PROGRAMM"
```

ein, nimmt der C 64 an, daß eigentlich

```
SAVE "PROGRAMM",1
```

gemeint ist. Benutzt man aber eine Floppy Disk, kann sich der Programmierer nicht mehr darauf verlassen, daß der C 64 diese Information für ihn hinzufügt. Also muß das Format des SAVE-Befehls normalerweise

SAVE "<PROGRAMMNAME>",8

sein und das des LOAD-Befehls

LOAD "<PROGRAMMNAME>",8

3. DAS ABSPEICHERN UND LADEN MIT MEHREREN LAUFWERKEN

Wird ein Einzeldiskettenlaufwerk beim Händler gekauft, ist es intern so eingestellt, daß es von sich denkt, es sei das Gerät 8. Es wird also allen Anweisungen, so auch den SAVE- und LOAD-Befehlen aus dem vorigen Abschnitt, folgeleisten, die an ein Gerät mit dieser Nummer gerichtet sind. Eine zunehmende Anzahl von Leuten entdeckt aber den Vorteil, mehr als ein Laufwerk zu betreiben. Betreibt man aber mehr als ein Laufwerk, stellt sich das Problem, daß die Laufwerke wissen *müssen*, welches von ihnen zu einem bestimmten Zeitpunkt gerade angesprochen wird.
Um diese Probleme zu überwinden, wurden die 1541-Laufwerke so gebaut, daß man ihre Gerätenummer ändern kann. Ein Befehl kann dann z. B.

SAVE "PROGRAMM",9

lauten. So wird nur eines der beiden Laufwerke angesprochen, und das Gerät 8 bleibt völlig unberührt. Es gibt sowohl eine Handware- als auch eine Softwarelösung für die Änderung der Gerätenummern, d. h. Sie können entweder das Laufwerk selbst verändern, oder Sie gebrauchen ein Programm, daß eine vorübergehende Änderung vornimmt. Werden Sie ständig mehr als eine Floppy Disk betreiben, ist die Handwarelösung sicherlich die eleganteste. Diese erfordert normalerweise nur die Durchtrennung einer Leiterbahn auf der Platine im Inneren der 1541. Da die Details zu dieser Lösung zwar im 1541-Handbuch stehen, aber nicht sehr verständlich beschrieben sind, empfehlen wir Ihnen, diese Zwei-Minuten-Arbeit direkt beim Kauf von Ihrem Händler ausführen zu lassen. Sollte sich Ihr Händler vor dieser Arbeit scheuen, suchen Sie sich am besten einen, der sein Handwerk versteht.
Die Gerätenummer einer Floppy Disk mit einem Programm zu verändern, ist nicht sehr schwer (siehe Kapitel 13), aber es kann lästig werden, da es jedesmal gemacht werden muß, wenn man die Floppy Disk einschaltet. Trotzdem, sollten Sie sich die zweite Floppy Disk nur für einen Tag von Ihrem Freund geliehen haben, ist es besser, die Softwarelösung anzuwenden, als das Laufwerk auseinanderzunehmen.

4. EINE EINFACHE METHODE ZUR VEREINFACHUNG DES PROGRAMMABSPEICHERNS

Um das Abspeichern eines Programms zu vereinfachen und Sie zu ermutigen, dies auch zu *tun,* können Sie in jedes Programm, daß Sie schreiben, eine Programmabspeichereinrichtung einbauen. Diese könnte wie folgt aussehen:

```
1 GOTO 3
2 SAVE "@0:PROGRAMMNAME",8 : VERIFY "PRO
GRAMMNAME",8 : STOP
3 REM
```

Das Einfügen einer solchen Routine in ein Programm hat den Vorteil, daß Sie das Programm nur sehr unwahrscheinlich wegen eines Tippfehlers unter dem falschen Namen abspeichern. Das Abspeichern wird durch die Eingabe von GOTO 2 eingeleitet. Mit GOTO 1 als zusätzlichem Bonus können Sie alle Programme starten, um RUN zu umgehen, das ja bekanntlich alle gespeicherten Variablen löscht.
Zwei Merkmale dieser Routine erfordern einige Erklärung: der VERIFY-Befehl und der Modifizierer "@0:" am Anfang des Programmnamens.

5. DER GEBRAUCH VON VERIFY

Einer der Hauptgründe, warum die SAVE-Routine in das Programm eingebaut wurde, ist, daß man sie anschließend mit dem VERIFY-Befehl kombinieren kann. Der Zweck des VERIFY-Befehls ist die Überprüfung, ob das auf einem bestimmten Laufwerk abgespeicherte Programm auch wirklich mit dem sich augenblicklich im Speicher des C 64 befindenden Programm übereinstimmt, d. h. ob ein Programm richtig abgespeichert wurde. Das Format des VERIFY-Befehls ist:

VERIFY "<PROGRAMMNAME>",<GERÄTENUMMER>

wobei PROGRAMMNAME für den Namen des auf dem Laufwerk abgespeicherten Programms steht. Es ist nicht wichtig, daß der Name des Programms auf der Diskette mit dem Namen übereinstimmt, den Sie dem Programm im Speicher gegeben haben. Der Programmname wird zwar in dem Directory der Diskette gespeichert, aber nicht mit dem Programm selbst. Es wird auch kein Name für das aktuelle Programm im Speicher des C 64 abgelegt. Der Floppy Disk die Anweisung zu geben, eine bestimmte Datei zu finden, ist alles, was Sie hier machen.
Anders als der Datassette-Rekorder, erfordert die Floppy Disk keine Arbeit vom Benutzer, wenn der VERIFY-Befehl angewendet wird. In der kurzen SAVE-Routine aus dem vorigen Abschnitt wird das Laufwerk automatisch das abgespeicherte Programm heraussuchen, ohne daß der Benutzer eingreift.

6. DAS ÜBERSCHREIBEN VON DATEIEN

In einem Punkt ist die Floppy Disk nicht so einfach einzusetzen wie der Datassette-Rekorder: Wollen Sie ein Programm zum zweiten Mal auf einem Band abspeichern, brauchen Sie nur zurückzuspulen und den SAVE-Befehl einzugeben – das vorige Programm wird überschrieben. Nicht so mit der Floppy Disk, weil sie so gebaut wurde, daß Sie ein Programm nicht unabsichtlich mit einem desselben Namens überschreiben können. Das ist in den meisten Fällen auch wünschenswert. Will man aber aufeinanderfolgende Versionen eines Programms abspeichern, kann dies störend werden. Mit dem Modifizierer (oder 'Klammeraffen') "@0:" liefert das DOS (Disk Operating System) eine Einrichtung, die dieses Problem angeht.
Er steht vor dem Namen der Datei (entweder einer Programm-Datei oder einer anderen, später beschriebenen Datei – mit Ausnahme der relativen Datei).
Begegnet das DOS einem Dateinamen, der mit "@0:" beginnt, fängt es sofort an, auf der Diskette nach einem Programm mit demselben Namen wie dem des angegebenen Dateinamen *ohne* dem vorangestellten "@0:" zu suchen. Gibt es keines, wird das Programm wie üblich abgespeichert. *Gibt* es aber ein Programm mit demselben Namen, ersetzt das abgespeicherte Programm das auf der Diskette – die vorherige Version ist unwiderbringlich gelöscht.
Eines sollte man beim Gebrauch des "@0:" beachten, weil die Routine auch einen Nachteil hat. Bei Disketten, die fast voll sind, wird mit "@0:" zwar der Programmname vollständig abgespeichert, andere Daten auf der Diskette aber werden zerstört. Der Grund hierfür ist, daß "@0:" in einigen Fällen zu versäumen scheint, das richtige Bild der Sektoren auf der Disk, die sie belegt oder gelöscht hat, in die Block Allocation Map (BAM) hineinzuschreiben. So passiert es, daß nachfolgende Dateien an Stellen abgelegt werden, wo sie nicht hingehören.

Zu diesem Problem gibt es mehrere Lösungsmöglichkeiten:

1. Sie fügen einen VALIDATE-Befehl (siehe Kapitel 4) in die Zeile 2 der kurzen SAVE-Routine ein. Dies ordnet die BAM neu und sorgt dafür, daß es keine Störungen gibt. Der einzige Nachteil ist, daß es länger dauern kann, den VALIDATE-Befehl auszuführen als eine Diskette zu formatieren.

2. Beginnen Sie, indem Sie das Programm TEST01 oder ähnlich nennen und ändern Sie jedesmal, nachdem Sie das Programm abgespeichert und mit LIST die Zeile 2 auf den Bildschirm geholt haben, die Zahl am Ende des Programmnamens. Dies ist zwar sehr einfach, verbraucht aber während der Entwicklung des Programms ziemlich viel Platz auf der Diskette.

3. Sie ignorieren den Nachteil der Routine – er wird Sie sowieso nur sehr selten in Ihrer Arbeit beeinflussen.

4. Am besten aber, Sie verwenden die Befehle RENAME und SCRATCH, zwei System-Befehle, die in Kapitel 4 beschrieben werden und eine viel sichere und elegantere Methode darstellen, ein Programm oder aber Dateitypen abzuspeichern.

7. DIE SICHERUNG DES ABGESPEICHERTEN

Der Vorgang, ein wertvolles Programm sicher aufzubewahren, endet nicht mit der Abspeicherung auf Disketten. Disketten können jederzeit beschädigt werden. Haben Sie ein wichtiges Programm, sollten Sie sich *immer* eine Kopie machen und diese an einem sicheren Ort verwahren.
Zusätzlich sollten Sie nicht vergessen, daß Sie von wichtigem Material auch eine Kopie auf der relativ sicheren und verläßlichen Kassette machen können. Ein ernster Fehler im Laufwerk kann sehr frustrierend werden, wenn Sie von dem erwünschten Programm nur Kopien auf Disketten besitzen. Am Anfang ignorieren die meisten diesen Tip, aber nur bis sie die Erfahrung lehrt, wie schlimm es sein kann, ein Programm, an dem sie wochenlang gearbeitet haben, plötzlich zu verlieren.

KAPITEL 4

DIE DISKETTEN-SYSTEM-BEFEHLE

1. *Einleitung*
2. *OPEN*
3. *CLOSE*
4. *PRINT#*
5. *NEW*
6. *SCRATCH*
7. *RENAME*
8. *COPY*
9. *INITIALIZE*
10. *VALIDATE*

1. EINLEITUNG

Wir haben bislang gesehen, daß das Laufwerk zum schnellen Abspeichern und Laden von Programmen dienen kann. Wird es auf diese Art gebraucht, hat der Benutzer keine andere Möglichkeit zur Einflußnahme auf die Disketten oder deren Inhalt, als ein Programm abzuspeichern oder zu laden. Eine effektive Nutzung des Laufwerkes, auch zur Abspeicherung von Programmen, beinhaltet die Fähigkeit, mit der Floppy Disk zu kommunizieren und diese Kommunikation zu gebrauchen, um die Art, *wie* Dinge auf der Diskette gespeichert werden, zu steuern. In diesem Kapitel betrachten wir eine Reihe von Befehlen, die weniger mit der Art zu tun haben, in der das Laufwerk Informationen aufnimmt, als damit, wie mit Dateien umgegangen wird, nachdem sie auf einer Diskette gespeichert wurden.

DIE KOMMUNIKATION MIT DER FLOPPY DISK – EINE EINFACHE ANALOGIE

Um die Probleme zu verstehen, die sich bei der Kommunikation zwischen dem C 64 und der Floppy Disk ergeben, stellen Sie sich vor, daß Sie über eine Schalttafel mit 15 schwarzen Telefonen und einer unbegrenzten Anzahl von Notizblöcken mit einer anderen Person kommunizieren wollen. Ihre Aufgabe hierbei ist es, Informationen für die Person am anderen Ende der Leitungen zu speichern und

sie auf Verlangen zurückzuschicken. Die zu speichernden Informationen sind ziemlich kompliziert. Es handelt sich z. B. um Aktien und Warenpreise an der Börse. Die Informationen erreichen Sie in verschiedenen Formen.
Natürlich erhält sowohl die Person von Ihnen, als auch Sie erhalten von der Person Informationen. Zu allem Überfluß müssen die verschiedenen Informationsarten an unterschiedlichen Stellen, die von der jeweiligen Art abhängig sind, gespeichert werden. Wie werden Sie damit fertig?
Ein Weg wäre, jede Informationsart einem Telefon und einem extra Notizblock zuzuordnen. Klingelt Telefon Nummer 4, wissen Sie, daß, wenn die Stimme am anderen Ende "567" sagt, der momentane Goldpreis gemeint ist, und Sie notieren ihn auf dem dazugehörenden Notizblock, der mit "Gold" beschriftet ist. Wäre die Botschaft über das Telefon 5 gekommen, hätte dies bedeutet, daß der Preis einer bestimmten Aktie gemeint war, der auf einem anderen Notizblock vermerkt werden müßte. Analog dazu würde das Klingeln des Telefon 8 bedeuten, daß Sie den letzten Stand des Silberpreises durchgeben sollen – die Person am anderen Ende würde genau wissen, was gemeint ist, da das Telefon 8 benutzt wird.
In späteren Kapiteln werden wir noch genauer darauf eingehen, wie die Floppy Disk dazu gebraucht werden kann, Informationen auf eine Art zu speichern und auf Befehl wieder abzuliefern, die unserem imaginären Telefonsystem entspricht. Mit anderen Worten: Wie sollen Sie wissen, daß der mit "GOLD" beschriftete Notizblock neben dem Telefon 4 liegen muß? Die Person am anderen Ende muß Ihnen *mitteilen,* daß die Nummer 4 für diesen Zweck gebraucht wird. Sie kann dies aber nicht über eines der 15 normalen Telefone machen, weil Sie sich bewußt dumm anstellen und sich weigern, die Telefone zu bedienen, bevor Sie nicht wissen, für welche Art von Informationen sie gebraucht werden. Um dies zu umgehen, bekommen sie ein zusätzliches Telefon, diesmal ein rotes. Wann immer das rote Telefon klingelt, beantworten Sie es sofort, und Sie werden ungefähr folgendes hören:

"Nehmen Sie den Block mit der Aufschrift GOLD vom Telefon 4 weg, und legen Sie es an einen sicheren Ort. Ersetzen Sie ihn durch den mit der Aufschrift Aktien des Unternehmens XYZ."
oder
"Das Telefon Nummer 4 wird bis auf weiteres nicht mehr gebraucht, legen Sie den GOLD-Block fort."
oder
"Legen Sie den SILBER-Notizblock neben das Telefon Nummer 5, das momentan nicht in Gebrauch ist."

Immer wenn Sie so eine Anweisung über das rote Telefon erhalten, befolgen Sie sie sofort. Das Resultat ist, daß Sie nun immer wissen, welches Telefon für was gebraucht wird.

Die zweite Bedingung, auf die wir bis jetzt noch nicht eingegangen sind, bezieht sich auf die Ablage der Notizblöcke. Wir haben zu Beginn gesagt, daß Sie über eine unbegrenzte Anzahl Notizblöcke verfügen können. Obwohl dies zutrifft, stellt sich das Problem, daß Sie nur für eine begrenzte Anzahl, sagen wir 144, genügend sicheren Stauraum haben. Die Person am anderen Ende legt sehr großen Wert auf die in die Notizblöcke eingetragenen Informationen und weiß, daß Sie nur einen begrenzten Stauraum haben. Deswegen gibt sie Ihnen von Zeit zu Zeit Anweisungen, wie Sie Ihren Vorrat an Notizblöcken handhaben sollen. Dies geschieht, indem er Sie zunächst über das rote Telefon anruft und Ihnen sagt: "Die nächste Mitteilung über Telefon Nummer 15 wird eine Anweisung sein, etwas mit einem oder mehreren Notizblöcken anzustellen." Falls sie weiß, was sie macht, wird kurz darauf Telefon Nummer 15 klingeln, und Sie werden eine Mitteilung hören, die so klingen könnte:

"Wir brauchen die Silberpreise nicht mehr. Sie können alles im SILBER-Notizblock Aufgeschriebene ausradieren und das Etikett entfernen."
oder
"Ich möchte die gestrigen Goldpreise von den heutigen getrennt haben, also benennen Sie den GOLD-Block in GOLD 1 um, und fangen Sie einen neuen mit der Aufschrift GOLD 2 an."
oder
"Ich habe Angst um die in den Notizblöcken aufgeschriebenen Informationen. Bitte fertigen Sie von allen Kopien an."

Dies alles sieht vielleicht sehr trivial aus, aber wenn Sie sich die Mühe machen, sich in das Problem hineinzuversetzen, werden Sie dem Ziel, die Vorgänge bei der Zusammenarbeit des C 64 mit der 1541 zu verstehen, ein schönes Stück näher gekommen sein. In den folgenden Abschnitten werden wir uns zunächst einmal mit dem OPEN-Befehl beschäftigen, der dem roten Telefon in unserem Beispiel entspricht. Wir werden aber auch noch eine Vielzahl anderer Befehle untersuchen, die alle "Verwaltungsaufgaben" für Dateien ausführen. Diese sind von der Sorte, die in dem Telefonbeispiel den über Telefon Nummer 15 empfangenen Anweisungen entsprechen.

2. OPEN

FUNKTION: OPEN ordnet einem Kanal eine Filenummer zu, so daß Informationen zu dieser Datei gesendet und von der Floppy Disk empfangen werden können. Zusätzlich wird OPEN zur Bestimmung von Filenamen und der Gebrauchsart eines Files gebraucht.

Wird ein Programm geladen oder abgespeichert, führt der BASIC-Interpreter des C 64 eine Reihe von Operationen durch, die dem Benutzer verborgen bleiben. Die wichtigste ist hierbei, abgesehen vom Übertragen und Abfragen von Daten von der Diskette, das Öffnen eines Kommunikations-Kanals mit der Floppy Disk. Dies entspricht dem Gebrauch des roten Telefons in dem obigen Beispiel. Im Falle des Abspeicherns, zum Beispiel, muß dem Laufwerk mitgeteilt werden, daß es Daten erhält und wohin es diese stecken soll. Im Falle des Ladens mit "LOAD" und des Abspeicherns mit "SAVE" wird diese Operation automatisch ausgeführt, ohne daß der Benutzer etwas anderes zu tun hat, als das richtige Schlüsselwort einzugeben. Wäre die 1541 *nur* zum Speichern und Abrufen von Programmen da, würde man lediglich zwischen dem SAVE- und dem LOAD-Modus hin- und herschalten zu brauchen. In Wirklichkeit ist die 1541 fähig, sowohl eine Vielzahl verschiedener Informationstypen als auch Programme zur Speicherung zu akzeptieren. Zusätzlich kann sie auf Instruktionen und Befehle hören und weiß, daß dies *keine* Dateien sind, die für die Abspeicherung gedacht sind. Sie ist imstande, ganze Datenketten zur Speicherung aufzunehmen und jede einzelne, entsprechend den Anweisungen des Programmierers, an einer anderen Stelle abzulegen. Sie ist fähig, andere Informationen von einer anderen Stelle zu holen, um kurz darauf wieder zu der Stelle zurückzukehren, an der die letzten Informationen gespeichert wurden, um diesen noch einige hinzuzufügen. Kurz, die 1541 kann all jene Aufgaben erfüllen, die auch die 15 Telefone in dem oben beschriebenem Beispiel schafften.
Genau wie das Telefonsystem, können auch der C 64 und 1541 auf verschiedenen "Leitungen" (die genaue Anzahl hängt vom Filetyp ab) miteinander kommunizieren. Die Felder auf der Diskette, in denen Informationen gespeichert werden können, werden Files oder Dateien genannt. Sie entsprechen den Notizblöcken in unserem Beispiel. Beim Übertragen und Empfangen von Informationen können verschiedene Leitungen für die Verbindung gebraucht werden. Dies sind die sogenannten "Filenamen", wobei jede mit einer anderen File korrespondieren muß. Es sind keine physikalisch voneinander getrennten Leitungen zwischen den zwei Geräten, sondern einfache Zahlen, die es dem C 64 ermöglichen, in einer speziellen Datei Informationen zu speichern oder von ihr abzufragen.
Um Informationen von der 1541 abzufragen, oder zu ihr zu übersenden, müssen zunächst einmal mit Hilfe des OPEN-Befehls die folgenden Punkte angegeben werden:

1. Die Zahl, die der Datei zugeteilt werden soll (die 'Telefonnummer').

2. Die Tatsache, daß diese Zahl für die Kommunikation zwischen dem C 64 und der 1541 gebraucht wird und nicht zwischen dem C 64 und irgendeinem anderen Gerät, wie z. B. dem Datassette-Rekorder oder dem Drucker.

3. Ob diese Datei als Informationsquelle gebraucht wird, oder um in ihr Informationen zu speichern, oder ob das, was ausgetauscht werden soll, Verwaltungsinformationen sind.
Manchmal, aber noch nicht für die Zwecke dieses Kapitels, ist noch eine zusätzliche Angabe nötig, nämlich:

4. Der Name der Datei (des 'Notizblockes') auf der Diskette, der der oben erwähnten Filenummer zugeordnet werden soll.

Im Moment möchten wir jedoch erstmal erfahren, wie uns der OPEN-Befehl ermöglicht, die in dem Telefonbeispiel erklärten Systembefehle anzuwenden. Es gibt noch viele andere Anwendungsmöglichkeiten für den OPEN-Befehl, die in den folgenden Kapiteln untersucht werden, aber für die Zwecke dieses Kapitels wird das Format des OPEN-Befehls so aussehen:

OPEN<**FILENUMMER**>,<**GERÄTENUMMER**>,**15** z. B. OPEN 15,8,15

1. **FILENUMMER** – steht für eine Zahl zwischen 1 und 127. Normalerweise – für die Verwaltungsaufgaben jedenfalls – bleiben Programmierer am besten bei einer Filenummer, damit sie erkennen, daß sie nur für diesen Zweck gebraucht wird. Der Grund dafür ist, daß die Systembefehle sehr einflußreich sind und sogar zum Verlust von Informationen auf der Diskette führen können. Da die Zahl am Ende des OPEN-Befehls, wie Sie weiter unten sehen werden, auch 15 heißt, bevorzugen es viele Programmierer, die Zahl 15 nur für Dateiverwaltungszwecke zu verwenden.

2. **GERÄTENUMMER** – eine Zahl, die dem C 64 anzeigt, mit welchem Gerät er in Verbindung tritt. Spricht man die Floppy Disk an, ist dies normalerweise die Zahl 8, obwohl es auch andere Zahlen sein können, wie in Kapitel 13 beschrieben.

3. **15,** die Zahl am Ende – die sogenannte "Sekundäradresse" oder der "Kanal". Eine Zahl an dieser Stelle wird verwendet, um dem angegebenen Gerät mitzuteilen, daß die folgende Information auf eine spezielle Art behandelt werden muß. Wenn Sie an das Telefonbeispiel zurückdenken, erinnern Sie sich sicherlich daran, daß bevor irgendeine Verwaltungsinformation ausgetauscht wurde, man Ihnen mitteilte, daß zu diesem Zweck ein spezielles Telefon gebraucht wurde. Im Format des OPEN-Befehls beinhaltet eine Sekundäradresse 15 die Botschaft: "Alles, was dieser Datei zugesandt wird, soll bis auf weiteres wie eine Verwaltungsinformation behandelt werden."
Sie werden in den folgenden Kapiteln noch sehen, daß das, was wir hier den "Verwaltungsmodus" nennen, eigentlich noch viel mehr ist.
Man nennt den Kanal 15 auch den "Fehlerkanal". Dieser verbindet uns direkt mit dem komplizierten Programm, das in die 1541 eingebaut ist und sie betreibt. Der

Fehlerkanal ermöglicht uns, sonst unerfüllbare Arbeiten zu erledigen. Sie werden noch sehen, daß es noch mehr über den OPEN-Befehl zu sagen gibt. Für den Moment aber wird

```
OPEN 15,8,15
```

ausreichen, um in den Verwaltungsmodus zu gelangen und die Befehle, die zu ihm gehören, zu erklären.

ZUSAMMENFASSUNG VON OPEN

1. Eine Information kann weder auf die 1541 geschrieben werden noch von ihr gelesen werden, ohne daß vorher mit OPEN ein File geöffnet wurde.

2. Wird ein File geöffnet, müssen immer die Filenummer, die Gerätenummer und die Kanalnummer angegeben werden. Für einige Zwecke müssen der Filename, der Filetyp und die Verwendungsart (ob die Datei zum Schreiben oder Lesen gebraucht wird) angegeben werden. Die letzten drei Angaben werden in anderen Kapiteln beschrieben.

3. Die Filenummer liegt zwischen 1 und 127.

4. Benutzen Sie nur ein Laufwerk, so ist die Gerätenummer normalerweise 8, sie kann sich aber ändern, falls noch andere Laufwerke in Gebrauch sind.

5. Für die meisten Anwendungen liegt die Kanalnummer zwischen 2 und 14. Sie ist 15, wenn eine Datei für den Fehlerkanal geöffnet wird.

6. Das Format des OPEN-Befehls ist:

```
OPEN<FILENUMMER>,<GERÄTENUMMER>,<KANALNUMMER>
```

3. CLOSE

FUNKTION: CLOSE setzt einen Terminator an das Ende einer Datei und schreibt die Adresse und die Länge in das Disketten-Directory.

In dem Telefonbeispiel gab es, Sie erinnern sich vielleicht, eine über das rote Telefon übermittelte Anweisung, die Sie aufforderte, einen bestimmten Notizblock,

oder gar ein bestimmtes Telefon, bis auf weiteres nicht mehr zu gebrauchen. Die Parallele dieser Anweisung ist der CLOSE-Befehl, der dem Laufwerk mitteilt, daß es für eine bestimmte Datei im Moment keine weitere Verwendung gibt, so daß sie sicher und ordentlich irgendwo auf der Diskette gespeichert werden kann.
In nachfolgenden Kapiteln werden Sie sehen, daß es für CLOSE, genau wie für OPEN, sehr viele Anwendungsmöglichkeiten gibt. Zur Zeit aber ist alles, was uns interessiert, der Verwaltungsmodus, und wir wollen uns darauf beschränken, herauszubekommen, wie wir mit diesem umgehen.

BEISPIEL ZUR VERANSCHAULICHUNG DER NOTWENDIGKEIT VON CLOSE

1. Schalten Sie den C 64 und die Floppy Disk ein, und legen Sie eine Diskette in das Laufwerk.

2. Tippen Sie:

   OPEN 15,8,15[RETURN]

3. Tippen Sie:

   OPEN 15,8,15[RETURN]

4. Nun sollten Sie die Fehlermeldung

   ?FILE OPEN ERROR

sehen.

5. Tippen Sie:

   CLOSE 15[RETURN]

6. Tippen Sie:

   OPEN 15,8,15[RETURN]

7. Dieses Mal wird keine Fehlermeldung hervorgerufen.

8. Wenn Sie sehr sorgfältig arbeiten wollen, geben Sie ein zweites Mal CLOSE 15[RETURN] ein.

Die Bedeutung des Ganzen ist sehr einfach. Haben Sie dem System einmal mitgeteilt, daß eine bestimmte Filenummer für einen bestimmten Zweck gebraucht wird, bleibt diese auch diesem Zweck zugeordnet, bis Sie sie wieder freisetzen. Der erste OPEN-Befehl weist dem Fehlerkanal der Floppy Disk die Filenummer 15 zu. Sie steht erst wieder zur Verfügung, nachdem die erste Zuweisung abgeschlossen wurde. Dies ist die Funktion des CLOSE-Befehls. Sie sollten sich deshalb angewöhnen, eine Datei, die Sie für eine Weile nicht mehr gebrauchen werden, zu schließen. Dies gilt besonders für den Fehlerkanal, da seine Befehle sehr wichtig sein können. Ist die Datei einmal geschlossen, besteht keine Gefahr mehr, daß Sie ihr unabsichtlich eine Anweisung übermitteln.
Haben Sie einen der in diesem Kapitel beschriebenen Systembefehle angewendet, sollten Sie daran denken, den Fehlerkanal zu schließen, bevor Sie fortfahren.
Obwohl es noch eine Menge über das Öffnen (mit OPEN) und das Schließen (mit CLOSE) des Fehlerkanals zu sagen gibt, würde dies nicht viel zu Ihrem Verständnis der Systembefehle in diesem Kapitel beitragen. Eines sollten Sie jedoch beachten, wenn Sie die Verwaltungsbefehle in ein Programm einbauen wollen, bevor Sie das Kapitel über den Fehlerkanal gelesen haben: Den Fehlerkanal zu schließen bewirkt, daß alle Dateien auf der Diskette geschlossen werden, ohne daß der C 64 darauf aufmerksam gemacht wird. Wenn Sie andere Dateien gebrauchen, um Informationen zu speichern, schließen Sie sie entweder, bevor Sie den Fehlerkanal öffnen, oder öffnen Sie den Fehlerkanal vor allen anderen Dateien und schließen Sie ihn auch nach ihnen.

ZUSAMMENFASSUNG VON CLOSE

1. Filenummern, die durch den OPEN-Befehl einer Datei zugeordnet wurden, stehen so lange nicht für andere Anwendungen zur Verfügung, bis diese Datei (mit CLOSE) geschlossen wurde.

2. Dateien, die nicht korrekt (mit CLOSE) geschlossen wurden, bevor die Verbindung zwischem dem C 64 und der 1541 abgebrochen wird, werden nicht vollständig aufgezeichnet, und ihr Inhalt geht verloren.

3. Das Schließen des Fehlerkanals (mit CLOSE) bewirkt, daß alle in diesem Moment offenen Dateien auf der 1541 geschlossen werden, obwohl sie der C 64 immer noch als offen betrachtet.

4. Das Format des CLOSE-Befehls ist:

CLOSE<FILENUMMER>

4. PRINT#

FUNKTION: PRINT# übermittelt einer bestimmten Datei Informationen in Form von String- oder numerischen Variablen.

Bisher haben wir nur gelernt, wie wir eine bestimmte Datei (mit OPEN) öffnen und (mit CLOSE) schließen können, aber nicht was wir wirklich mit einer Datei anfangen können. In diesem Abschnitt werden wir uns zum ersten Mal ansehen, wie Informationen, die zur Speicherung in einer Datei bestimmt sind, zur Floppy Disk übermittelt werden können. Noch einmal: Für die Zwecke dieses Kapitels sind wir nur daran interessiert, wie wir, wenn der Fehlerkanal einmal geöffnet ist, Systemanweisungen zu der 1541 übermitteln können. Später wird noch genauer auf den Gebrauch des Print# eingegangen.

ANWENDUNGSBEISPIEL FÜR DEN PRINT#-BEFEHL

1. Geben Sie das folgende kurze Programm ein, und starten Sie es mit RUN:

```
10 PRINT "□"
20 OPEN 1,3
30 PRINT#1,"DIES IST EIN BEISPIEL FUER E
INE AUSGABE"
35 PRINT#1,"AUF EIN BESTIMMTES GERAET"
40 CLOSE 1
```

Sie sollten die in Zeile 30 angegebenen Wörter auf dem Bildschirm dargestellt sehen – was ist passiert? PRINT# ist eine Art, Informationen in ein File zu übertragen. In diesem Fall wurde (mit OPEN) eine Datei mit der Nummer 1 auf dem Gerät mit der Nummer 3, dem Bildschirm selbst, geöffnet. Nachdem die Datei geöffnet wurde, erfüllte PRINT# die Aufgabe, die genannten Wörter zu dem bestimmten Gerät zu übertragen.
In den folgenden Abschnitten wird PRINT# dazu verwendet, eine Botschaft über den offenen Fehlerkanal zu übertragen, die dem DOS mitteilt, welche Systemoperationen ausgeführt werden sollen.
Vorsicht ist jedoch auch bei der Anwendung von PRINT# (und den vergleichbaren Befehlen INPUT# und GET#) geboten, die Sie in den folgenden Kapiteln finden werden. Viele C-64-Besitzer haben sich angewöhnt, beim Eingeben eines BASIC-Schlüsselwortes die Abkürzung zu benutzen (z. B. L gefolgt von SHIFT I für LIST). Es sollte daran erinnert werden, daß keiner der gerade erwähnten Befehle eingegeben werden kann, indem man die Abkürzung des normalen Schlüsselwortes gebraucht und dann "#" anfügt. Gibt man "?#" in eine Programmzeile ein, zeigt

das LISTING zwar "PRINT#", aber jedesmal, wenn die Zeile ausgeführt werden soll, erscheint ein SYNTAX ERROR. Die korrekten Abkürzungen für die drei Befehle, die das "#" enthalten, sind:

PRINT# – P SHIFT/R

INPUT# – I SHIFT/N

GET# – Keine Abkürzung

ZUSAMMENFASSUNG VON PRINT#

1. PRINT# wird zur Übermittlung von Informationen zu einer Datei gebraucht.

2. Die richtige Abkürzung für PRINT# ist *nicht* "?#", sondern "P SHIFT/R.

3. Für die Übersendung von Systembefehlen zu der 1541 ist das Format des PRINT#-Statements:

PRINT#<FILENUMMER>,<BEFEHLSSTRING>

4. Die maximale Länge des Befehlsstrings, der über den Fehlerkanal geschickt werden kann, ist 41 Zeichen und nicht, wie im Handbuch angegeben, 58 Zeichen.

5. PRINT# spielt noch in vielen anderen Anwendungen eine Rolle, die in anderen Kapiteln erörtert werden.

WIE MAN PRINT# IN SYSTEMBEFEHLEN VERMEIDET

Wer schon einige Erfahrung beim Umgang mit der Floppy Disk hat, wird sicherlich schon wissen, daß es eine kürzere Form für die Übertragung von Befehlen gibt als die Anwendung von PRINT#: Man kann den "Befehlsstring" (d. h. die Wörter in Anführungszeichen in Zeile 30 des oben erwähnten Beispielprogramms) als Teil des OPEN-Befehls auffassen, z. B. würde aus

OPEN 15,8,15

gefolgt von

PRINT# 15,"BEFEHL"

OPEN 15,8,15,"BEFEHL"

Dies ist allerdings nur Mitteilungen für den Fehlerkanal vorbehalten. Beispiele finden Sie in den Zusammenfassungen der einzelnen Befehle. Wir haben uns dazu entschlossen, die Anwendung der Befehle im allgemeinen mit Hilfe des PRINT#-Befehls darzustellen, weil dieses unkomplizierte Format am wenigsten von der Hauptsache, nämlich dem Befehl selbst, ablenkt.

5. NEW

FUNKTION: Um eine neue Diskette für die Aufnahme von Daten vorzubereiten, gibt es einen Vorgang, das sogenannte "Formatieren". Die Diskette wird in Spuren und Sektoren aufgeteilt und ein Inhaltsverzeichnis (das Directory) und die BAM (Block Allocation Map) angelegt. Eine weniger drastische Form des NEW-Befehls ändert den Titel einer Diskette, ohne den momentanen Inhalt zu löschen.

In der Beschreibung des Aufbaus einer Diskette in Kapitel 1 haben wir bereits gehört, daß eine Diskette vorbehandelt werden muß, bevor sie für die Speicherung von Programmen und Daten gebraucht werden kann. Die 1541 kann diesen Vorgang auf einem von zwei Wegen ausführen:

1. Sie löscht alles, was sich auf der Diskette befindet und formatiert sie neu, *oder*

2. Ist die Diskette schon formatiert, so ist es möglich, das Directory der Diskette zu löschen, ohne sie wirklich neu zu formatieren. Dateien, die nachfolgend auf der Diskette gespeichert werden, überschreiben einfach die Programme, die sich zur Zeit auf ihr befinden, weil diese für die 1541 durch das Löschen des Directorys unsichtbar geworden sind. Der Vorteil dieser zweiten Methode ist, daß es sehr viel weniger Zeit beansprucht als eine Diskette vollständig neu zu formatieren.

In beiden oben genannten Fällen gehen alle auf der Diskette gespeicherten Daten verloren. Im Fall der ersten Methode ist die Situation nicht wieder wettzumachen, da die Daten durch die magnetischen Schreib-Lese-Köpfe des Laufwerkes physikalisch gelöscht werden. Wird die zweite Methode angewendet, ist es eventuell möglich, die verlorenen Daten bei Umgehung des Directorys von der Diskette zu lesen, obwohl dies alles andere als einfach ist. Die Lehre hieraus ist, daß Sie sehr vorsichtig vorgehen sollten, wenn Sie eine Diskette formatieren. Haben Sie irgendwelche Zweifel, untersuchen Sie erst das Directory der Diskette, um völlig sicher zu gehen, daß sie nicht noch Material enthält, das Sie behalten wollen.
Nun, da wir etwas über die Gefahren gesprochen haben, lernen Sie den Vorgang des Formatierens am besten kennen, indem Sie ihn anwenden.

ANWENDUNGSBEISPIEL FÜR NEW

1. Schalten Sie den C 64 und die Floppy Disk ein.

2. Nehmen Sie eine neue, unformatierte Diskette oder eine, welche nichts Wichtiges enthält, und schieben Sie sie in das Laufwerk.

3. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"NEW0:TEST,01" [RETURN]
```

4. Warten Sie, bis das rote Lämpchen an der Floppy erlischt.

5. Tippen Sie:

```
CLOSE15 [RETURN]
LOAD "$",8 [RETURN]
LIST [RETURN]
```

6. Sie sollten nun folgendes auf dem Bildschirm sehen (obwohl die Überschrift in inversen Zeichen dargestellt sein wird):

```
0 "TEST                    " 01 2A
664 BLOCKS FREE.
```

7. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"NEW0:TEST2" [RETURN]
```

8. Warten Sie, bis das rote Lämpchen am Laufwerk erlischt.

9. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST
```

10. Sie sollten folgendes auf dem Bildschirm sehen:

```
0 "TEST2                   " 01 2A
664 BLOCKS FREE.
```

Nun haben Sie die Diskette zweimal formatiert, einmal komplett und ein zweites Mal, indem Sie einfach den Namen geändert und das Directory gelöscht haben (es stand sowieso nichts im Directory). Sie haben sicherlich bemerkt, daß die zweite Methode wesentlich schneller vonstatten ging als die erste.

DAS FORMAT DES NEW-BEFEHLS

Bei der Untersuchung der obigen Schritte fällt sehr schnell auf, daß zwei Befehle das Formatieren und das Neuformatieren bewirken:

1. PRINT#15, "NEW0:TEST1,01"

und

2. PRINT#15,"NEW0:TEST2"

Dies läßt sich wie folgt erklären:

a) **NEW:** der Befehl selbst. Alle Systembefehle beginnen mit dem Schlüsselwort des Befehls oder dessen Abkürzung. Für NEW ist die Abkürzung "N", so daß der Befehl auch

PRINT#15,"N0:TEST,01"

hätte lauten können.

b) **"0:":** bei Doppellaufwerken gibt dieses Zeichen das Laufwerk an, auf dem die Operation ausgeführt werden soll. Bei den normalerweise benutzten Einzellaufwerken ist es eine Null, die man dann auch ganz weglassen kann.

c) **TEST1** (oder TEST2): der Name der Diskette. Der Name wird einzig und allein dazu gebraucht, um die Disketten beim Ausdrucken des Directorys identifizieren zu können.

d) **"01":** die Identifikationszahl oder ID der Diskette. Sie besteht aus zwei alphanumerischen Zeichen zwischen 00 und ZZ. Die ID erfüllt zwei Aufgaben. Erstens erlaubt sie es, einer Reihe von Disketten denselben Namen zu geben, die dann anhand der ID unterschieden werden können. Zweitens wird die ID beim Formatieren der Diskette in jeden Block geschrieben. Sie wird vom Laufwerk gelesen, das,

wenn es einen Block liest, überprüft, ob die Diskette verändert wurde, seit das Laufwerk das letzte Mal adressiert worden ist. Dies ist allerdings keine effektive Überprüfung, da die meisten Programmierer ohnehin dazu neigen, alle ihre Disketten mit "01" zu numerieren. Wird eine Diskette mit derselben ID wie die der vorigen in das Laufwerk eingelegt, ist es möglich, daß die Floppy Disk weiterhin auf der Basis des Directorys der vorigen Diskette arbeitet und nicht wahrnimmt, daß ein Wechsel vorgenommen wurde. Für das Laden von Dateien mag dies nicht so wichtig sein, doch kann es sich verhängnisvoll auf das Abspeichern von Programmen oder Dateien auswirken, da bereits existierende Informationen überschrieben werden können. Um dieses Problem zu umgehen, sollten Sie das Laufwerk jedesmal, wenn Sie Disketten auswechseln, neu initialisieren (siehe den INITIALIZE-Befehl weiter unten in diesem Kapitel).
Sie haben bestimmt bemerkt, daß beim zweiten Beispiel zum NEW-Befehl keine ID angegeben wurde. In diesem Fall wird die Diskette so belassen, wie sie ursprünglich formatiert wurde, und nur der Titel und das Directory geändert. Weil es nicht nötig ist, die Blockmarkierung zusammen mit der ID in jedem Block auf der gesamten Diskette neu hineinzuschreiben, beansprucht die zweite Methode weniger Zeit als die erste.

ZUSAMMENFASSUNG VON NEW

1. NEW muß immer angewendet werden, wenn eine gerade gekaufte Diskette zum ersten Mal gebraucht werden soll.

2. Bei schon formatierten Disketten spart die Version des NEW-Befehls ohne ID Zeit.

3. NEW löscht alle auf der Diskette vorhandenen Daten und muß deshalb mit äußerster Vorsicht angewendet werden.

4. NEW kann jeweils eines der folgenden Formate haben:

   PRINT#15,"NEW0:<FILENAME>,<ID>" – vollständiges Reformatieren einer Diskette.

   PRINT#15,"N0:<FILENAME>,<ID>" – wie oben, nur mit abgekürztem Schlüsselwort

   PRINT#15,"NEW0:<FILENAME>" – Diskette wird umbenannt und das Directory gelöscht

PRINT#15,"N0:<FILENAME>" – wie oben, nur mit abgekürztem Schlüsselwort

5. Wendet man die verkürzte Form mit OPEN an, könnte dies ein typisches Format sein:

OPEN15,8,15"N0:TEST1,01"

6. SCRATCH

FUNKTION: SCRATCH löscht eine Datei von der Diskette, indem es sie im Directory als "scratched file" (gelöschtes File) kennzeichnet. Obwohl gelöschte Dateien auf der Diskette erhalten bleiben, wird der Platz, den sie einnehmen, für nachfolgende Dateien bereitgestellt und, falls weitere Files geöffnet werden, überschrieben.

Wollen Sie Ihre Floppy Disk wirkungsvoll einsetzen, werden Sie sich wohl öfters entscheiden müssen, welche Files Sie auf einer Diskette behalten wollen und welche nicht. Disketten sind zwar nicht sehr teuer, aber dennoch können die Kosten schnell eskalieren, wenn Sie Ihre Disketten mit frühen Versionen von Programmen und Daten vollpacken, die Sie nie wieder gebrauchen werden. Hinzu kommt, daß Disketten mit einer großen Menge unnützen Materials die Aufgabe, ein wichtiges Programm zu finden, sehr erschweren. Was Sie normalerweise brauchen, ist eine Diskette extra für jedes Projekt an dem Sie zur Zeit arbeiten. Jede dieser Disketten enthält eine Menge halbfertiger Programme, die Sie nicht mehr lange behalten wollen. Sind Sie einmal über das Entwicklungsstadium hinaus, sollten Sie sich eine Diskettensammlung anlegen, die nur brauchbares Material enthält. Am besten speichern Sie gleichartige Programme auf einer Diskette ab, so daß Sie immer gleich wissen, welche Diskette Sie heraussuchen müssen, um ein bestimmtes Problem zu lösen.
Um eine Sammlung von Disketten aufzubauen, die das enthalten, was Sie *wollen* und nicht das, was gerade da ist, müssen Sie nicht nur in der Lage sein, Dateien zu speichern, sondern auch zu löschen. Und dies ermöglicht Ihnen der SCRATCH-Befehl.

ANWENDUNGSBEISPIEL FÜR SCRATCH

1. Vergewissern Sie sich, daß der C 64 und die Floppy Disk eingeschaltet sind, und die im vorigen Abschnitt formatierte Diskette TEST2 korrekt im Laufwerk sitzt.

2. Geben Sie das folgende einzeilige Programm ein:

```
10 REM DIES IST EIN TESTPROGRAMM
```

3. Tippen Sie:

```
SAVE "TEST1",8 [RETURN]
SAVE "TEST2",8 [RETURN]
```

4. Nachdem das Abspeichern beendet ist, tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

5. Nun sollten Sie folgendes auf dem Bildschirm sehen:

```
0 "TEST2              " 01 2A
1     "TEST1"             PRG
1     "TEST2"             PRG
662 BLOCKS FREE.
```

6. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"SCRATCH0:TEST1" [RETURN]
CLOSE15 [RETURN]
```

7. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

Auf dem Bildschirm erscheint:

```
0 "TEST2              " 01 2A
1     "TEST2"             PRG
663 BLOCKS FREE.
```

Folgendes ist passiert: Von den beiden Files, TEST1 und TEST2, die Sie auf der Diskette plaziert haben, wurde eine, TEST1, gelöscht und deren Platz für anderweitigen Gebrauch freigestellt. Nach den Warnungen bezüglich des NEW-Befehls

sollte man nicht mehr zu sagen brauchen, daß auch der SCRATCH-Befehl nur sehr vorsichtig angewendet werden darf, weil Filenamen nur zu leicht verwechselt werden und man dann versehentlich die falsche Datei löscht. Haben Sie irgendwelche Zweifel, so untersuchen Sie den Inhalt einer Datei, bevor Sie sie löschen.
Im folgenden Kapitel über "pattern matching" (zu deutsch etwa Zeichenfolgen vergleichen) werden Sie herausfinden, daß SCRATCH , auf der Basis von Ähnlichkeiten in Filenamen, sogar mehr als eine Datei beeinflussen kann. Im Prinzip unterscheidet sich die spätere Abwendungsmöglichkeit aber nicht von dieser.

ZUSAMMENFASSUNG VON SCRATCH

1. Der Systembefehl SCRATCH entfernt eine bestimmte Datei von der Diskette im Laufwerk.

2. SCRATCH ist ein wichtiger Befehl zur Verwaltung Ihrer Diskettensammlung, der angewendet wird, um zu vermeiden, daß wertvoller Platz durch unnützes Material verschwendet wird.

3. SCRATCH sollte mit Vorsicht angewendet werden, da gelöschte Files normalerweise nicht wiederhergestellt werden können.

4. Das Format des SCRATCH-Befehls ist:

```
PRINT#15,"SCRATCH0:<FILENAME>"
```

oder

```
PRINT#15,"S0:<FILENAME>"
```

5. Gebraucht man die verkürzte Version von OPEN, ist das Format:

```
OPEN15,8,15,"S0:<FILENAME>"
```

6. Vorausgesetzt, daß die mit SCRATCH gelöschten Files noch nicht überschrieben worden sind, ist es möglich, sie eventuell mit dem UNSCRATCH-Programm aus Kapitel 10 zurückzuholen.

7. RENAME

FUNKTION: RENAME ändert den Namen einer Datei auf der Diskette, vorausgesetzt, daß der neue Name, welcher der Datei gegeben werden soll, noch nicht im Directory vorhanden ist. Der Inhalt der Datei wird dabei nicht verändert.

Der RENAME-Befehl ermöglicht es Ihnen, den Namen einer Datei zu korrigieren, ohne den Rest des Dateiinhalts in irgendeiner Weise zu ändern. Nur wenige Programmierer machen sich den RENAME-Befehl zunutze, weil sie keinen Grund erkennen, Änderungen an Dateinamen vorzunehmen. Dies ist schade; denn der RENAME-Befehl ist in Verbindung mit dem SCRATCH-Befehl ein leistungsfähiges und effektives Werkzeug bei der Verwaltung von Dateien.
Das Format des RENAME-Befehls ist (bei Verwendung nur einer Floppy kann '0:' natürlich jeweils entfallen):

(RENAME:<NEUER FILENAME>=<ALTER FILENAME>)

ANWENDUNGSBEISPIEL FÜR DEN RENAME-BEFEHL

1. Vergewissern Sie sich, daß der C 64 und die Floppy Disk eingeschaltet sind und das Laufwerk die früher formatierte Diskette TEST2 enthält. Die Diskette TEST2 sollte nur das eine Programm TEST2 enthalten, welches während des Ausprobierens des SCRATCH-Befehls abgespeichert wurde.

2. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"RENAME0:TEST1=0:TEST2" [RETURN]
CLOSE15 [RETURN]
```

3. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

4. Sie sollten folgendes sehen:

```
0 "TEST2           " 01 2A
1    "TEST1"           PRG
663 BLOCKS FREE.
```

5. Tippen Sie:

```
LOAD "TEST1",8 [RETURN]
LIST [RETURN]
```

6. Es erscheint:

```
10 REM THIS IS A TEST PROGRAM
```

Die Diskette, die vorher das Programm TEST2 enthielt, enthält nun das Programm TEST1, obwohl die Inhalte der Datei nicht verändert wurden. RENAME kann noch zu vielen anderen Gelegenheiten benutzt werden.

BEISPIEL FÜR DIE ERSTELLUNG VON ERSATZDATEIEN MIT HILFE DES SCRATCH- UND RENAME-BEFEHLS

1. Geben Sie dieses kurze Programm ein:

```
1 GOTO 10
2 SAVE "TEMPORARY",8 : VERIFY "TEMPORARY
",8
3 OPEN 15,8,15
4 PRINT#15,"SCRATCH0:TEST1.BAK"
5 PRINT#15,"RENAME0:TEST1.BAK=0:TEST1"
6 PRINT#15,"RENAME0:TEST1=0:TEMPORARY"
7 CLOSE 15
10 REM DAS IST VERSION 1
```

2. Starten Sie das Programm mit RUN, und warten Sie, bis das Wort READY erscheint und der blinkende Cursor zurückkehrt.

3. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

4. Im Directory sollten Sie erkennen, daß es zwei Files gibt, TEST1 und TEST1.BAK.

5. Tippen Sie:

```
LOAD "TEST1",8 [RETURN]
```

Dann ändern Sie Zeile 10 wie folgt um:

```
10 REM DAS IST VERSION 2
```

6. Starten Sie das Programm mit RUN, und warten Sie, bis der blinkende Cursor wiederkehrt.

7. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

8. Sie sollten aus dem Directory herauslesen können, daß es immer noch zwei Files sind: TEST1 und TEST1.BAK, nur daß TEST1.BAK nun zuvorderst steht.

9. Tippen Sie:

```
LOAD "TEST1.BAK",8 [RETURN]
LIST [RETURN]
```

und Sie werden erkennen, daß die Version 1 des Programms geladen wurde.

10. Tippen Sie:

```
LOAD "TEST1",8 [RETURN]
LIST [RETURN]
```

und nun sehen Sie, daß die Version 2 des Programms geladen wurde.

Hier haben Sie also das abzuspeichernde Programm zuerst unter dem Namen "TEMPORARY" (zu deutsch "vorübergehend") auf der Diskette gespeichert. Falls Sie sich diese Methode zu eigen machen wollen, sollten Sie darauf achten, daß der Filename "TEMPORARY" immer nur als vorläufiger Name für ein momentan abgespeichertes Programm gebraucht wird. Es kann natürlich für alle Ihre abzuspeichernden Programme eingesetzt werden, da er sowieso nur für einige Augenblicke auf der Diskette existiert.

Nachdem "TEMPORARY" abgespeichert wurde, sucht das Programm auf der Diskette nach dem Programm "FILENAME.BAK" (in unserem Fall TEST1.BAK) und löscht es. Ist dieses Programm nicht vorhanden, wie es der Fall ist, wenn etwas zum ersten oder zweiten Mal nach dieser Methode abgespeichert wurde, entstehen keine Probleme – die Anweisung wird einfach ignoriert.
Der dritte Schritt ist das Programm FILENAME (in unserem Fall TEST1) in FILENAME.BAK umzubenennen, wobei ".BAK" die Abkürzung für "backup" (zu deutsch "Ersatz-") ist. Deswegen wird die momentane Version des Programms, an der noch gearbeitet wird, nicht gelöscht, sondern einfach umbenannt.
Zum Schluß wird die Datei "TEMPORARY" wieder in FILENAME (in unserem Fall TEST1) umbenannt.
Die letzte Version des Programms heißt jetzt FILENAME und die letzte aller vorherigen Fassungen FILENAME.BAK. Man muß jedoch beachten, daß das Anfügen von ".BAK" an das Ende des Dateinamens bedeutet, daß die maximalmögliche Länge des Dateinamens von 16 auf 12 Buchstaben reduziert wird.
Ein Vorteil ist jedoch, daß Sie trotzdem eine Kopie (nämlich die vorherige) behalten, sollten Sie einmal aus Versehen ein Programm verderben und dann abspeichern. Ein zweiter Vorteil ist, daß der Fehler, der bei der Methode mit "@0:" auftreten kann, hier vermieden wird – es kann nichts überschrieben werden. Außerdem kann die Floppy Disk, genau wie jedes andere Gerät, einmal ausfallen. Entweder zerstört dies eine Datei, während sie abgespeichert wird, oder ein Spannungsstoß im Netz führt zur Unterbrechung des SAVE-Vorganges, oder was noch schlimmer ist, das Programm stürzt im C 64 ab. In diesem Fall besitzen Sie nur noch ein zerstückeltes Programm auf der Diskette und noch nicht einmal mehr eines, das Sie aus dem Speicher des C 64 retten könnten. Führen Sie aber die obige Prozedur aus, kann diese Situation sehr einfach vermieden werden, weil sich zu keinem Zeitpunkt eine Version des Programms auf der Diskette befindet, die nicht verloren gehen darf – es wird immer noch eine vollständige Version vorhanden sein.

ZUSAMMENFASSUNG VON RENAME

1. Die Funktion des RENAME-Befehls ist, einer Datei auf der Diskette einen neuen Filenamen zu geben, ohne den übrigen Inhalt der Diskette zu verändern.

2. RENAME, in Verbindung mit SCRATCH, ist eine wertvolle Einrichtung zur Erstellung von Ersatzkopien von Informationen, die andernfalls überschrieben würden.

3. RENAME hat, genau wie SAVE, einige Einschränkungen, z. B. wird der RENAME-Befehl keine Datei eröffnen, wenn schon eine mit demselben Namen existiert.

4. Das Format des RENAME-Befehls ist:

```
PRINT#15,"RENAME0:<NEUER FILENAME> =0:<ALTER FILENAME>"
```

oder

```
PRINT#15,"R0:<NEUER FILENAME> =0:<ALTER FILENAME>"
```

5. Das Format des mit Hilfe des abgekürzten OPEN-Befehls angewendeten RENAME-Befehls ist:

```
OPEN15,8,15,"R0:<NEUER FILENAME> =0: <ALTER FILENAME>"
```

8. COPY

FUNKTION: COPY kopiert eine existierende Datei unter einem anderen Namen. Der COPY-Befehl unterscheidet sich vom RENAME-Befehl darin, daß die Originaldatei erhalten bleibt. Mit COPY lassen sich Dateien an das Ende anderer Dateien anfügen.

Der Copy-Befehl wird gebraucht, um von irgendeiner Datei auf der Diskette ein Duplikat mit anderem Namen, aber gleichem Inhalt herzustellen. Beachten Sie, daß der Unterschied zum RENAME-Befehl darin liegt, daß die Originalversion der Datei auf der Diskette bleibt. Auf der 1541 verwendet man COPY hauptsächlich zur Erstellung von Ersatzkopien einer Datei auf derselben Diskette. Offen gesagt, ist COPY auf einem Einzellaufwerk wie der 1541 viel weniger brauchbar, als es auf einem Doppellaufwerk wie der CBM 4040 oder der CBM 8050 der Fall wäre. Auf einem Doppellaufwerk können Dateien von den zwei Laufwerken *untereinander* kopiert werden, so daß Ersatzkopien auf einer seperaten Diskette angelegt werden können.

DAS ANEINANDERKETTEN VON DATEIEN

COPY ermöglicht es, Files einer Diskette "aneinanderzuketten". Man kann zwischen zwei und vier Dateien zu einer neuen Datei zusammenfassen – auch hier bleiben die Originaldateien unverändert. Sie haben vielleicht bemerkt, daß diese Möglichkeit im Handbuch der 1541 zwar erwähnt, aber kein Anwendungsbeispiel dargestellt wird. Ein Grund hierfür wäre, daß Dateien nur selten aneinandergekettet werden müssen. Im Kapitel über sequentielle Dateien zeigen wir Ihnen, wie Dateien

brauchbar verbunden werden können. Aneinandergekettete Programm-Dateien machen zwar beim Laden keine Schwierigkeiten, jedoch wird dem Programmierer jeweils nur das erste Programm zur Verfügung stehen, während der Rest den Speicher verstopft. Der Grund hierfür ist, daß jede Programm-Datei mit zwei Bytes beginnt, die dem C 64 mitteilen, wohin im Speicher das jeweilige Programm geladen werden soll und mit zwei Nullen aufhört, die das Ende der Datei anzeigen. An den richtigen Stellen sind diese Kennzeichen sehr wichtig; werden sie jedoch in der Mitte der neuen zusammengefaßten Datei eingeschlossen, machen Sie es dem C 64 unmöglich, das vollständige Programm zu erkennen, das in den Speicher eingeladen wurde.

ANWENDUNGSBEISPIEL FÜR DEN COPY-BEFEHL

1. Vergewissern Sie sich, daß die Floppy Disk und der C 64 eingeschaltet sind, und die TEST2-Diskette richtig im Laufwerk sitzt.

2. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"C0:TEST2=0:TEST1" [RETURN]
CLOSE15 [RETURN]
```

3. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

4. Im Directory sehen Sie jetzt, daß zu den Programmen TEST1 und TEST1.BAK das Programm TEST2 hinzugekommen ist.

5. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"C0:TEST3=0:TEST1,0:TEST2" [RETURN]
CLOSE15 [RETURN]
```

6. Tippen Sie:

```
LOAD "$",8 [RETURN]
LIST [RETURN]
```

7. Sie sollten nun aus dem Directory herauslesen können, daß noch ein weiteres Programm hinzugekommen ist, nämlich TEST3, und daß dieses neue Programm zwei Blöcke lang ist.

Von der Datei TEST1 wurde unter dem Namen TEST2 ein Duplikat angefertigt, welches zwei Dateien auf der Diskette ergibt. Die zwei Dateien wurden dann zu TEST3 zusammengefügt. Wenn Sie wollen, können Sie das Programm TEST3 mit LOAD in den Speicher laden. Sie werden aber herausfinden, daß, obwohl das Programm auf der Diskette doppelt so lang erscheint, es mit dem vorigen Programm TEST1 übereinstimmt, welches wir schon im Abschnitt über RENAME untersucht haben. Tatsächlich geht ein Duplikat des Programms irgendwo im Speicher des C 64 verloren, was die längere Version auf der Diskette begründet.

ZUSAMMENFASSUNG VON COPY

1. COPY dupliziert eine Datei unter einem anderen Namen, aber mit demselben Inhalt.

2. COPY kann auch verwendet werden, um bis zu vier Dateien aneinanderzuketten, obwohl diese Anwendungsmöglichkeit beschränkt ist (siehe Kapitel 7).

3. Das Format des COPY-Befehls bei der Anwendung zum Vervielfachen von Dateien ist:

```
PRINT#15,"COPY0:<ZWEITER FILENAME> =0: <ERSTER FILENAME>"
```

oder

```
PRINT#15,C0:<ZWEITER FILENAME> =0:<ERSTER FILENAME>"
```

4. Das Format des COPY-Befehls bei der Aneinanderkettung von Dateien ist:

```
PRINT#15,"COPY0:<NAME4> =0:<NAME1>,0:<NAME2>,0:<NAME3>"
```

oder

```
PRINT#15,"C0:<NAME4>, =0:<NAME1>,0:<NAME2>,0:<NAME3>"
```

Es ist wichtig, daran zu denken, daß der Befehlsstring, der über den Fehlerkanal übermittelt wird, nicht länger als 41 Zeichen sein darf.

5. Verwendet man die abgekürzte Form des OPEN-Befehls, sind die zwei Formate:

OPEN15,8,15,"C0:<ZWEITER FILENAME> =0:<ERSTER FILENAME>

oder

OPEN15,8,15,"C0:<NAME4> =0:<NAME1>,0:<NAME2>,0:<NAME3>"

9. INITIALIZE

FUNKTION: INITIALIZE zwingt die Floppy Disk, die BAM von der Diskette in ihren Speicher zu schreiben. So wird sichergestellt, daß die Floppy Disk immer mit der neuesten Version der BAM arbeitet. Man vermeidet Probleme, die während der Ausführung des VALIDATE-Befehls oder beim Austausch der Diskette gegen eine andere mit demselben Identifizierungsmerkmal (ID) auftreten können.

In der allgemeinen Einführung zur Arbeitsweise der Floppy Disk erwähnten wir, daß das DOS eine Art Belegungsplan der Diskette, die Block Allocation Map (BAM), gebraucht, wenn die Floppy Daten liest oder schreibt. Manchmal passiert es, wie wir im Kapitel über SAVE gesehen haben, daß das Laufwerk den richtigen Zustand der BAM nicht erkennt und Daten in Sektoren der Diskette schreibt, die schon belegt sind, also existierende Dateien zerstört.
Diese Verwirrung tritt auf, weil die Kopie der BAM auf der Diskette nur dann auf den neuesten Stand gebracht wird, wenn eine Datei (mit CLOSE) geschlossen wird.
Wenn das Laufwerk angeschaltet wird und auf eine Diskette zum ersten Mal zugegriffen wird, schreibt die Floppy Disk die BAM in ihren eigenen Speicher. Veränderungen, wie das Schreiben von Daten in eine Datei, werden zwar von der im Laufwerkspeicher existierenden Version der BAM registriert, aber *nicht* von der Kopie der BAM auf der Diskette (bis eine Datei, wie schon erwähnt, geschlossen wird). Es ist möglich, daß irgendein Diskettenfehler zur Zerstörung der BAM im Speicher führt, so daß die Floppy Disk mit einem falschen Bild der Diskette arbeitet. Dies kann besonders bei der Ausführung des VALIDATE-Befehls, aber auch beim Austauschen der Diskette gegen eine andere mit derselben ID (siehe NEW) passieren. In diesem Fall wird die Floppy Disk so lange mit der BAM der ersten Diskette arbeiten, bis eine Datei (mit CLOSE) geschlossen wird; dann ist es allerdings meistens zu spät.
Die Funktion des INITIALIZE-Befehls ist es, das Laufwerk zu zwingen, die BAM von der Diskette einzulesen, um so das Problem von zerstörten BAMs zu umgehen. INITIALIZE sollte immer dann angewendet werden, wenn man bei der Ausführung des VALIDATE-Befehls (siehe nächsten Abschnitt) auf ein Problem stößt, oder man

sich nicht absolut sicher ist, daß sich die ID der neuen Diskette von der der alten unterscheidet. Die Anwendung des INITIALIZE-Befehls erzeugt keine sichtbaren Veränderungen auf der Diskette. Der einzige Beweis für das Funktionieren des Befehls wird die schrumpfende Menge zerstörter Disketten sein.

ZUSAMMENFASSUNG VON INITIALIZE

1. Die Funktion des INITIALIZE-Befehls ist es, sicherzustellen, daß die Block Allocation Map, auf die sich die Floppy Disk bei der Arbeit beruft, ein genaues Abbild des momentanen Diskettenzustandes ist.

2. Der INITIALIZE-Befehl sollte bei Problemen mit dem VALIDATE-Befehl oder beim Austauschen einer Diskette gegen eine andere, die vielleicht dieselbe ID hat, angewendet werden.

3. Das Format des INITIALIZE-Befehls ist:

```
PRINT#15,"INITIALIZE"
```

oder

```
PRINT#15,"I"
```

4. Wendet man die abgekürzte Version des OPEN-Befehls an, ist das Format:

```
OPEN15,8,15,"I"
```

10. VALIDATE

FUNKTION: VALIDATE erneuert die BAM und geht dabei von dem wirklich auf der Diskette Gespeicherten aus.
Nicht ordnungsgemäß einer Datei zugeordnete oder mit einer Datei verknüpfte Sektoren, die entweder gelöscht oder nicht korrekt (mit CLOSE) geschlossen wurden, werden wieder freigesetzt.

Die Funktion des VALIDATE-Befehls ist, eine Diskette "aufzuräumen" und Platz für die Datenspeicherung zu schaffen. Der Grund, warum dies nötig werden kann, sind die verschiedenen Systembefehle, wie SCRATCH und das "@0:" zum Überschreiben von existierenden Dateien, die nicht immer alle Blöcke auf der Diskette

löschen, die mit einer überschriebenen oder mit SCRATCH gelöschten Datei verbunden sind. Weil aber Dateien auf einer Experimentier-Diskette ständig gelöscht oder überschrieben werden, kann die Situation eintreten, in der eine große Anzahl Blöcke laut der BAM irgend etwas zugeordnet ist, aber sonst keine Funktion erfüllt.
Der VALIDATE-Befehl liest nun jede Datei auf der Diskette durch, Block für Block, vom Anfang bis zum Ende. Wenn er Blöcken begegnet, die entweder nicht in die richtige Struktur einer Datei fallen, oder einer nicht mehr existenten Datei zugeordnet sind, werden diese Blöcke in der BAM wieder als frei gekennzeichnet. Das Ergebnis dieses Vorgangs ist eine völlig neue BAM, die auf dem beruht, was wirklich auf der Diskette ist. Es ist nicht möglich, die Anwendung des VALIDATE-Befehls mit einer frischen Diskette, wie z. B. unserer TEST01-Diskette, zu demonstrieren, aber wenn Sie ihn auf einer Diskette anwenden wollen, die Sie schon sehr häufig gebraucht haben, speziell für umfangreiche Programme, kann die Anzahl freigesetzter Blöcke manchmal recht hoch sein.
Mit VALIDATE können unter anderem auch die Einschränkungen des im Kapitel 3 erwähnten "@0:" umgangen werden. Das Problem bei "@0:" ist, daß es die durch Löschen der vorigen Version freigesetzten Blöcke oder die beim Speichern der neuen Version beanspruchten Blöcke nicht immer richtig in die BAM schreibt. Das behindert zwar nicht die Speicherung der momentanen Datei, bedeutet aber, daß zukünftige Dateien eventuell anhand der unrichtigen BAM abgespeichert werden, und dies mit allen Konsequenzen. Wird der VALIDATE-Befehl aber sofort nach dem Abspeichern einer Datei mit "@0:" angewendet, kann dieses Problem umgangen werden. Aber offen gesagt ist die Methode mit SCRATCH und RENAME zum Überschreiben von Dateien, beschrieben im Abschnitt über RENAME, die weitaus praktischere, weil VALIDATE viel mehr Zeit braucht.

ZUSAMMENFASSUNG VON VALIDATE

1. Die Funktion von VALIDATE ist, nicht ordnungsgemäß zugeordnete Blöcke auf der Diskette freizusetzen.

2. Das Format des VALIDATE-Befehls ist:

```
PRINT#15,"VALIDATE"
```

oder

```
PRINT#15,"V"
```

3. Wendet man die abgekürzte Form des OPEN-Befehls an, ist das Format:

OPEN15,8,15,"V"

4. Arbeiten Sie mit Direkt-Zugriffs-Daten (siehe Kapitel 10), so sollte der VALIDATE-Befehl nur unter Vorbehalt angewendet werden, da er diese wahrscheinlich löscht.

KAPITEL 5

PATTERN MATCHING

1. *Was ist "pattern matching"?*
2. *Zeichenfolgen mit "*"*
3. *Zeichenfolgen mit "?"*
4. *Die Kombination von "*" und "?"*
5. *Die Anwendung des "pattern matching" mit Befehlen*

1. WAS IST "PATTERN MATCHING"?

Der im letzten Kapitel beschriebene SCRATCH-Befehl kann zusammen mit LOAD und VERIFY durch eine einfache Technik, das "pattern matching" (zu deutsch etwa: Zeichenfolgen vergleichen), in seiner Wirkungsweise verbessert werden.
Im wesentlichen legt man beim "pattern matching" eine Art Schablone über die Dateinamen auf der Diskette. Die Schablone hat an einigen Stellen Buchstaben eingestanzt, durch die hindurch man das darunterliegende Papier sehen kann, während an anderen Stellen das Papier verdeckt bleibt. Liest man auf der Schablone zum Beispiel "PROG--------" (wobei die Bindestriche undurchsichtige Zeichenpositionen darstellen) und bewegt sie über einen Text mit Buchstaben derselben Größe wie die der ausgeschnittenen, so wird man, wenn die Buchstabenfolge "PROG" erscheint, diese Buchstaben deutlich lesen können, während andere nur ein belangloses Durcheinander ergeben. "Pattern matching" wendet diese Technik bei den Dateinamen im Directory der Floppy Disk an, indem der Benutzer eine Zeichenfolge oder Schablone angibt, um eine Operation mit allen Dateien, die in dieses Muster passen, durchzuführen.

2. ZEICHENFOLGEN MIT "*"

Die am einfachsten zu verstehenden Zeichenfolgen sind die, die das Sternchen "*" enthalten. Immer wenn ein Sternchen am Ende eines Dateinamens steht, nimmt das DOS an, daß es dem Benutzer egal ist, welches oder welche Zeichen hinter dem Sternchen folgen, nur die Zeichen vor dem Stern müssen stimmen. Untenste-

hend finden Sie Beispiele für eine Zeichenfolge mit "*" und den akzeptablen Dateinamen:

ZEICHENFOLGE	AKZEPTABLE NAMEN	UNAKZEPTABLE NAMEN
SMITH*	SMITHSON	SMIT, SMITT
	SMITHERS	SMYTHE

3. ZEICHENFOLGEN MIT "?"

Das andere Zeichen, das zur Erstellung einer Zeichenfolge benutzt werden kann, ist das Fragezeichen "?". Wenn ein Fragezeichen auftaucht, interpretiert es das DOS ungefähr so: "Es muß ein einzelnes Zeichen an dieser Stelle sein, aber es kommt nicht darauf an, welches: Mehrere Fragezeichen können verwendet werden, um eine bestimmte Anzahl Zeichen innerhalb eines Namens zu kennzeichnen, wobei die eigentlichen Zeichen irrelevant sind." Im Gegensatz zum "*" kann das Fragezeichen entweder am Anfang oder in der Mitte einer Zeichenfolge stehen.

ZEICHENFOLGE	AKZEPTABLE	UNAKZEPTABLE
SM??H	SMITH, SMYTH	SMITHSON, SMIT
	SMASH	SMART

4. DIE KOMBINATION VON "*" UND "?"

Die beiden Zeichen können miteinander kombiniert werden, um eine ganze Reihe von Effekten zu erzielen. Sie müssen aber dennoch mit Vorsicht angewendet werden, weil man manchmal nicht sofort erkennen kann, welche Folgen eine bestimmte Kombination haben wird. Schlimm wird es, wenn die Zeichenfolge mit einem SCRATCH-Befehl die falsche Datei löscht:

ZEICHENFOLGE	AKZEPTABLE	UNAKZEPTABLE
SM??*	SMITH, SMART	SMI, SMIT
	SMITHERS	
???T*	SMITH, SMITHERS	SMART, SMIT
??I?*	SMITH, SMITHERS	SIMS, SMART, SMIT
	PRICE	

5. DIE ANWENDUNG DES "PATTERN MATCHING" MIT BEFEHLEN

Das "pattern matching" kann mit den Befehlen SCRATCH, LOAD und VERIFY verwendet werden, indem man das jeweilige Symbol innerhalb der Anführungszeichen setzt, als ob es ein Teil des Filenamen ist.
Zum Beispiel:

```
"S0:T*"
```

als Botschaft über den Fehlerkanal würde jede mit "T" beginnende Datei durch SCRATCH löschen, während

```
"S0:*"
```

alle Dateien löscht.
Mit LOAD und VERIFY wird die Schablone dazu gebraucht, die erste übereinstimmende Zeichenfolge im Directory der Diskette zu finden. Diese Technik kann das Laden von Programmen erheblich verkürzen. LOAD "*" lädt die erste Datei aus dem Directory, eine Methode, die man häufig bei kommerziellen Disketten findet.

KAPITEL 6

DER FEHLERKANAL

1. *Was ist ein Fehlerkanal?*
2. *Das Lesen des Fehlerkanals*
3. *Der Gebrauch des Fehlerkanals*
4. *Zusammenfassung des Fehlerkanals*

In den vorigen Kapiteln haben wir den Fehlerkanal schon einige Male in Anspruch genommen. Der Ausdruck Fehlerkanal wird im Handbuch der 1541 für die mit der Sekundäradresse 15 geöffnete Datei oder den Kanal angewendet. Im wesentlichen haben wir diesen Kanal bis jetzt immer zur Übermittlung von Instruktionen zwischen dem C 64 und dem DOS gebraucht. In diesem Kapitel wollen wir uns die Art näher betrachten, in der die 1541 den C 64 über Probleme informiert, die eventuell beim Umgang mit Files auftreten können.

1. WAS IST DER FEHLERKANAL?

Probleme können bei der Floppy Disk in allen Situationen auftreten. Selbst beim Abspeichern (mit SAVE) oder Einladen (mit LOAD) kann das Blinken des roten Lämpchens anzeigen, daß aus irgendeinem Grund der Befehl nicht ausgeführt werden kann. Dies ist zwar sehr frustrierend, es gibt aber noch schlimmere Fehler. Wir werden in den Kapiteln 7 bis 10 noch sehen, daß bei der Datenspeicherung, anders als bei der Programmspeicherung, die Steuerung der Diskette durch das im C 64 laufende Programm sehr wichtig ist. Während man bei sorgfältiger Vorgehensweise erwarten kann, daß dieser Ablauf die meiste Zeit über reibungslos verläuft, wird es unvermeidlich auch Augenblicke geben, wo fehlerhafte Disketten, ein plötzliches Problem bei der 1541 oder mangelhaftes Programmieren zu Schwierigkeiten bei der Speicherung und Abfrage von Daten führen wird. So eine Situation ist dann weniger verhängnisvoll, wenn das die Floppy Disk steuernde Programm einen Fehler bemerkt und in der Lage ist, entweder selbst einzugreifen oder wenigstens den Benutzer zu informieren, daß irgend etwas getan werden muß. Wenn das Programm jedoch, ohne auf das Problem zu achten, weiterläuft, kann der Verlust oder die Zerstörung wertvoller Daten die Folge sein.

Glücklicherweise enthält 1541 in Form des Fehlerkanals eine mehr als ausreichende Einrichtung zur Aufdeckung von Problemen beim Umgang mit Disketten. Diese Direktverbindung zum 1541-DOS kann nicht nur für die Übermittlung von Instruktionen von Nutzen sein. Sie ist außerdem in der Lage, mehr als 50 verschiedene Fehlermeldungen über jedes auftretende Problem zu erzeugen und sie so zu übertragen, daß entweder der C 64 oder der Benutzer die Situation bereinigen können. Die Fehlermeldungen werden in vier getrennten Informationen geliefert, die das 1541-DOS auf Abruf bereit hat, um sie jedesmal über den Fehlerkanal zu schicken, wenn die Diskette angesprochen wird. In den meisten Fällen werden diese Informationen jedoch nicht gesendet, weil der Fehlerkanal nicht geöffnet ist – sie sind aber trotzdem immer verfügbar und können abgerufen werden, um die Sicherheit und Effektivität der Diskettenhandhabung deutlich zu verbessern.

2. DAS LESEN DES FEHLERKANALS

Nachstehend ist der Gebrauch des Fehlerkanals auf einfache Weise beschrieben.

BEISPIEL ZUM LESEN DES FEHLERKANALS

1. Geben Sie das folgende kurze Programm ein:

```
10 OPEN 15,8,15
20 INPUT#15,A,B$,C,D
30 PRINT A,B$,C,D
40 CLOSE 15
50 END
```

2. Wenn Sie sicher sind, daß das Laufwerk leer ist, schalten Sie es aus.

3. Schalten Sie die Floppy Disk wieder ein.

4. Starten Sie das Programm mit RUN. Sie sollten

```
 73              CBM DOS V2. 6 1541      0              0
```

angezeigt sehen.

5. Immer noch ohne Diskette im Laufwerk, geben Sie

```
SAVE"T",8[RETURN]
```

ein.

6. Starten Sie das Programm mit RUN. Sie sollten

```
74                DRIVE NOT READY      0            0
```

angezeigt sehen.

Sie haben jetzt direkt mit dem internen Prozessor der 1541 kommuniziert und ihn aufgefordert, Ihnen genau zu sagen, welches Problem Sie von der Ausführung des eingegebenen Befehls abhält. Der Prozessor liefert ihnen vier Angaben – in dieser Reihenfolge: Zahl, String, Zahl, Zahl – und sie haben folgende Bedeutung:

1. **Die Fehlernummer:** Obwohl auch eine verbale Meldung gemacht wird, sollte die Fehlernummer nicht ignoriert werden. Die Fehlermeldungen selbst sind in Gruppen unterteilt, die alle dieselbe verbale Botschaft haben. In diesen Fällen ist der einzige Weg, die möglichen Ursachen der Fehlermeldung unterscheiden zu können, die Fehlernummer für weitere Informationen nachzuschlagen. Die Fehlernummern werden im Handbuch der 1541 genau beschrieben.

2. **Die Fehlermeldung:** Diese zwei oder drei Wörter lange Botschaft ist eine allgemeine Beschreibung des aufgetretenen Problems (siehe "Die Fehlernummer").

3. **Die erste Zahl:** Die Spur der Diskette, auf der das Problem aufgetreten ist. War der zuletzt eingegebene Befehl SCRATCH, und es trat kein Fehler bei der Ausführung auf, wird die Fehlernummer "eins" heißen und die Zahl in der "Spur-Position" wird die Anzahl der gelöschten Files anzeigen.

4. **Die zweite Zahl:** Der Sektor innerhalb der Spur, in der das Problem aufgetreten ist. In den zwei obengenannten Beispielen zeigen die Nullen in den Positionen 3 und 4 an, daß der Fehler nicht bei der Diskette zu suchen ist, sondern beim Laufwerk. In den meisten Fällen helfen die Spur- und Sektor-Informationen bei der Suche nach der Fehlerursache jedoch nicht weiter. Wird ein READ- oder WRITE-Fehler in Spur 18, Sektor 0 angezeigt, haben Sie höchstwahrscheinlich eine unformatierte Diskette verwendet – in Spur 18 befindet sich normalerweise das Directory. Bei der Arbeit mit Direkt-Zugriffs-Dateien (siehe Kapitel 10) kann die

Spur- und Sektor-Information für die Identifizierung des Fehlers lebenswichtig sein, weil solche Dateien vom Programm aus auf einzelne Sektoren zugreifen.

Beachten Sie: Obwohl es möglich ist, den Fehlerkanal im Direktmodus (eine Anweisung ohne Zeilennummer) zu öffnen, ist es nicht möglich, die Fehlermeldung im Direktmodus über den Fehlerkanal zu erhalten, weil INPUT# (wie INPUT) eine ILLEGAL DIREKT ERROR-Meldung hervorruft. Der Fehlerkanal kann nur mit in Programmzeilen geschriebenen Anweisungen gelesen werden.

3. DIE BENUTZUNG DES FEHLERKANALS

Das oben erwähnte, fünfzeilige Testprogramm ist ein Beispiel, wie die vom Fehlerkanal erzeugten Meldungen auf den Bildschirm gebracht werden können. Viele Programmierer hängen diese Zeile als eine separate Routine an das Ende ihres Programms an. Falls dann ein Diskettenfehler auftritt, dessen Ursache nicht sofort auffällt, kann das Programm gestoppt und GOTO <Startzeile der Fehlerroutine> eingegeben werden. Dies ist zwar eine recht brauchbare Methode, aber ist das Programm einmal fertiggeschrieben und laufbereit, ist sie nicht mehr so zufriedenstellend. Gebraucht wird vielmehr eine Methode, die es ermöglicht, den Fehlerkanal während des Programmdurchlaufs abzufragen, so daß entweder das Programm selbst Schritte einleiten kann oder zumindest der Benutzer über das Problem informiert wird.
Es folgt eine kurze Subroutine, die in ein Programm eingesetzt werden kann, um den Fehlerkanal zu öffnen und die betreffenden Fehlermeldungen zu lesen.

EINE SUBROUTINE ZUM ABFRAGEN DES FEHLERKANALS ÜBER DAS PROGRAMM

```
5000 REM*******************************
5010 REM DISK FEHLER-STATUS
5020 REM*******************************
5030 INPUT#15,EN,EM$,ET,ES
5040 IF EN=73 THEN 5030
5050 IF EN<20 THEN RETURN
5060 PRINT "■■■■■ ***********************
****************"
5070 PRINT "■■                DISK FEHLER"
5080 PRINT "■■        FEHLER -" EN " " EM$
```

```
5090 PRINT "■       AUF SPUR -" ET "    UN
D SEKTOR -"ES
5100 PRINT "■■      PROGRAMMAUSFUEHRUNG U
NTERBROCHEN"
5110 PRINT "■■■■ ************************
**************"
5120 CLOSE 15
5130 SYS 65511
5140 END
```

DIE ERKLÄRUNG DER FEHLERKANAL-SUBROUTINE

Die Hauptmerkmale des Unterprogramms sollten schon beim ersten Betrachten klar sein. Die Subroutine arbeitet unter der Voraussetzung, daß der Fehlerkanal mit einem Befehl wie dem folgenden geöffnet wurde, bevor das Programm zum ersten Mal initialisiert wird:

```
OPEN 15,8,15
```

Falls während des Programmablaufs keine Fehler auftreten, sollte das Programm den Fehlerkanal von selbst schließen (mit CLOSE). Denken Sie daran, daß der Fehlerkanal immer *nach* den anderen Dateien geschlossen wird, weil das Schließen des Fehlerkanals das Schließen aller anderen Dateien auf der Diskette bewirkt.
Vorausgesetzt, daß der Fehlerkanal geöffnet wurde, kann die Subroutine von einem Datenfile-Modul z. B. immer dann aufgerufen werden, wenn eine Datei geöffnet wird, oder, will man absolut sichergehen, immer dann, wenn etwas in eine Datei geschrieben oder von ihr gelesen wird. Der Umgang mit Dateien wird aber durch diese letzte Methode verlangsamt. In den meisten Fällen wird das Abfragen keinen sichtbaren Effekt haben, weil die über den Fehlerkanal erhaltene Meldung "0, OK, 0, 0" sein wird, was bedeutet, daß alles in Ordnung ist. Wenn der Fehlerkanal aber ein Problem bemerkt, wird die Fehlermeldung sofort von der 1541 weitergegeben. Daraufhin folgt ein CLOSE 15, welches alle anderen Dateien auf der Diskette schließt und ein SYS 65511, der Aufruf einer Kernal-Routine, die alle Dateien vom C 64 aus schließt. Schließlich bricht die Subroutine den Programmablauf ab.
Diese Anwendungsmöglichkeit der Subroutine ist nur eine von vielen. Es ist selbstverständlich auch möglich, das Unterprogramm so aufzubauen, daß es bestimmte Fehlermeldungen nur für eine kurze Zeit anzeigt und dann zum File-Modul des Programmes oder zum Hauptmenü, von dem aus das File-Modul aufgerufen wurde, zurückkehrt. Bei einem Programm, welches dem Benutzer erlaubt, einen Filenamen für abzuspeichernde Daten einzugeben, wollen Sie viel-

leicht das Überschreiben von bereits existenten Files vermeiden. In diesem Fall würde es das Datenspeicherungs-Modul des Programms erlauben, einen Filenamen anzugeben, eine Datei zu öffnen versuchen, um dann die Subroutine aufzurufen, die die Meldung "63, FILE EXISTS, 0, 0" über den Fehlerkanal erhalten würde. Dies würde dem Benutzer als Hinweis dafür angezeigt werden, daß die Daten *nicht* abgespeichert wurden. Der Ablauf würde zum Hauptprogramm zurückkehren, wo dem Benutzer die Möglichkeit angeboten würde, die Datei unter einem anderen Namen abzuspeichern. In anderen Fällen, wenn z. B. komplizierte Daten von einer Datei zur anderen übertragen werden sollen, könnte das Versäumnis eine der Dateien zu öffnen, den sofortigen Abbruch des Programms bewirken, um das Zerstören von wertvollen Daten zu vermeiden.

4. ZUSAMMENFASSUNG ZUM FEHLERKANAL

1. Der Fehlerkanal, d. h. eine mit einer Sekundäradresse 15 geöffnete Datei, bietet eine praktische Methode, die Ursache von Problemen aufzudecken, die beim Umgang mit Disketten auftreten können.

2. Fehlermeldungen werden vom DOS der 1541 in Form einer Gruppe mit vier Informationen erzeugt: ZAHL 1, STRING, ZAHL 2, ZAHL 3.
Die ZAHL1 bezieht sich auf die Fehlernummer, mit der man die genauere Bedeutung eines Fehlers im Handbuch der Floppy Disk nachschlagen kann. Der STRING liefert eine kurze Beschreibung der Fehlerart in Worten. Die ZAHL2 und die ZAHL3 beziehen sich auf die Spur und den Sektor, wo der Fehler aufgetreten ist oder, falls der letzte eingegebene Befehl SCRATCH war, auf die Anzahl der gelöschten Files. Die Fehlernummer ist dann "eins", z. B. "1, FILES SCRATCHED, 2, 0,", was bedeutet, daß zwei Files gelöscht wurden.

3. Die Fehlermeldungen können *nur* innerhalb von Programmzeilen über den Fehlerkanal abgefragt werden; im Direktmodus ist es nicht möglich.

4. Der Fehlerkanal (mit CLOSE) zu schließen, heißt alle Dateien zu schließen. Soll der Fehlerkanal innerhalb eines Programms abgefragt werden, ist es im allgemeinen besser, ihn am Anfang des Programms zu öffnen und am Ende zu schließen.

KAPITEL 7

SEQUENTIELLE UND USER-DATEIEN

1. *Was ist eine sequentielle Datei?*
2. *Das Öffnen einer sequentiellen Datei mit OPEN*
3. *Die Ein- und Ausgabe von Daten PRINT# und INPUT#*
4. *GET#*
5. *Das Herausfinden des Dateiendes*
6. *Das Beenden der Aus- und Eingabe mit CLOSE*
7. *Anwendungsbeispiele für die sequentielle Datei*

In diesem Kapitel wollen wir einige Methoden für die Speicherung und das Abfragen von Daten innerhalb eines Programms untersuchen. Solche Methoden sind eine wichtige Hilfe für Programme, die eine große Datenmenge für den zukünftigen Gebrauch abspeichern sollen oder mehr Speicherplatz erfordern, als der C 64 zur Verfügung hat. Tatsächlich erhält man bei richtigem Einsatz der 1541 zur Speicherung von Daten einen enormen Speicherzuwachs beim C 64-Speicher, der beim Ausschalten des Gerätes nicht gelöscht wird.
Dieses Kapitel bezieht sich auf zwei Filetypen, die für diesen Zweck sehr geeignet sind, die sequentiellen und die User-Dateien. Der *einzige* Unterschied zwischen diesen beiden Typen ist, daß der eine mit dem Kennzeichen "S" eröffnet wird, während der andere das Kennzeichen "U" verwendet und daß die sequentielle Datei im Directory der Diskette mit dem Anhängsel "SEQ" erscheint, während User-Dateien mit "USR" gekennzeichnet sind. In diesem Kapitel gelten sämtliche Hinweise über die sequentielle Datei ohne Einschränkung auch für die User-Dateien, nur daß an Stellen, wo ein "S" steht, dieses durch ein "U" ersetzt werden muß. Eine gleich lautende Zusammenfassung der User-Datei ist zum einfacheren Nachschlagen am Ende des Kapitels zu finden.

1. WAS IST EINE SEQUENTIELLE DATEI?

Im wesentlichen ist eine sequentielle Datei nicht mehr als eine einfache Liste mit Strings oder Zahlen oder beidem vermischt, die auf der Diskette in der Reihenfolge, in der sie gespeichert wurden, geführt wird. Die Datenposten werden in Form einzelner Bytes, Zeichen für Zeichen, gespeichert. Innerhalb der Datei wird zwi-

schen den als String oder den als numerische Variable gespeicherten Zeichen kein Unterschied gemacht.
Die Lage der Zeichen innerhalb der Datei wird allein durch die Reihenfolge, in der sie gespeichert wurden, und die beim Schreiben in die Datei angewendete Zeichensetzung bestimmt. Die in der Datei gespeicherte Information kann normalerweise nur in der Reihenfolge gelesen werden, in der sie gespeichert wurde. Die Floppy Disk arbeitet mit einem internen Zeiger, der auf den Anfang der Datei deutet, wenn diese (mit OPEN) eröffnet wird. Werden Zeichen aus der Datei gelesen, wird der interne Zeiger dazu gebracht, die Spur, den Sektor und die Position innerhalb des Sektors des nächsten Zeichens anzuzeigen. Zur Anwendung einer sequentiellen Datei müssen Sie nur:

1. die Datei für das Schreiben oder Lesen öffnen,

2. Informationen in der richtigen Reihenfolge in die Datei schreiben oder von ihr lesen,

3. die Datei schließen.

2. DAS ÖFFNEN EINER SEQUENTIELLEN DATEI (MIT OPEN)

Wir haben im Kapitel über die Systembefehle schon einige Anwendungsmöglichkeiten für den OPEN-Befehl untersucht. Es ist der Befehl, der in unserem Telefonbeispiel dem roten Telefon entsprach, das uns Anweisungen für den Gebrauch der verschiedenen Verbindungsleitungen übermittelte.
Um eine sequentielle Datei (mit OPEN) zu öffnen, benötigt man fünf Informationen:

1. Die Filenummer, die der jeweiligen Gebrauchsart zugeordnet wird. Diese Nummer wird der im OPEN-Befehl genannten Datei bis zum Schließen zugeordnet bleiben. Der zulässige Bereich für eine Filenummer liegt unter normalen Umständen zwischen 1 und 127.

2. Die Gerätenummer des Laufwerkes. Sie ist normalerweise 8, kann aber, falls mehrere Laufwerke in Gebrauch sind, zwischen 4 und 31 liegen.

3. Die Kanalnummer oder Sekundäradresse, die für die sequentielle Datei zwischen 2 und 14 liegen muß. Anders als beim Datassettensystem, gibt es hier, genau wie bei der Sekundäradresse für das Schreiben oder Lesen von Daten, keine speziellen Regeln. Jedoch dürfen niemals zwei Files zur gleichen Zeit dieselbe Sekundäradresse beanspruchen.

4. Den Namen der Datei, auf die zugegriffen werden soll. Es muß eine sequentielle Datei sein.

5. Ein Hinweis darüber, ob die Datei für die Eingabe oder Ausgabe von Informationen verwendet wird. Ein und die gleiche Datei darf nicht gleichzeitig für die Ein- und Ausgabe geöffnet sein.

Sind alle diese Punkte bestimmt, lautet das Format des OPEN-Befehls für eine sequentielle Datei wie folgt:

OPEN<FILENUMMER>,<GERÄTENUMMER>,<KANALNUMMER>,"<FILENAME>,S[,Roder,W]"

Die FILENUMMER, GERÄTENUMMER, KANALNUMMER und FILENAME sollten nach der obigen Erklärung klar sein. Das "S" hinter dem Filenamen zeigt an, daß es sich um eine sequentielle Datei handelt. Danach folgt eine Auwahlmöglichkeit, die angibt, ob die Datei für die Speicherung von Daten ist, ",W" steht für "write file", oder für die Ausgabe ",R" steht für "read file". Wird diese Angabe völlig weggelassen, nimmt die Floppy Disk an, daß die Datei nur für die Ausgabe geöffnet wurde. Beispiele für den OPEN-Befehl sind unter anderem:

OPEN 1,8,2,"DATASTORE,S,W" Daten können in die Datei "DATASTORE" geschrieben werden.

OPEN 1,8,2,"DATASTORE,S,R" Daten können von der Datei „DATASTORE" gelesen werden.

OPEN 1,8,2,"DATASTORE,S" bedeutet dasselbe wie das vorige Format.

3. DIE EIN- UND AUSGABE VON DATEN – PRINT# und INPUT#

Die Speicherung und das Abfragen von Daten auf der Diskette wird durch die beiden Befehle PRINT# und INPUT# ermöglicht. Der PRINT#-Befehl legt eine oder mehrere Informationen auf der Diskette ab, der INPUT#-Befehl ist das Gegenteil, er lädt eine oder mehrere Informationen von der Diskette. In diesem Abschnitt wollen wir verschiedene Beispiele zu den zwei Befehlen und die Art, in der sie sich beeinflussen, darstellen.

DAS FORMAT VON PRINT# UND INPUT#

Das Format der beiden Befehle ist:

PRINT#<FILENUMMER>,LISTE MIT VARIABLEN

und

INPUT#<FILENUMMER>,LISTE MIT VARIABLEN

DIE ZEICHENSETZUNG BEI PRINT# UND INPUT#

Von den beiden Befehlen ist INPUT# einfacher zu erklären, weil seine Anwendung davon abhängig ist, wie die Informationen vorher mit PRINT# auf der Diskette gespeichert wurden.
Die einzelnen Posten der Variablenlisten hinter dem INPUT#-Statement sollten, genau wie beim normalen INPUT#-Befehl in BASIC, immer durch ein Komma getrennt werden.
Abgesehen von der Tatsache, daß es keinen speziellen Befehl dafür gibt, unterscheidet sich die Eingabe in eine Datei nur geringfügig von dem normalen Schreiben auf dem Bildschirm. Die Art, in der Daten auf der Diskette gespeichert werden, hängt von der Reihenfolge der Eingabe, von der angewendeten Zeichensetzung (oder -nichtsetzung) bei der Eingabe und von der An- bzw. Abwesenheit von "carriage return"-Zeichen ab ("carriage return" = "Wagenrücklauf" bei der Schreibmaschine).
Als Beispiel für die Probleme, die auftreten können, geben Sie das folgende Programm ein und starten es mit RUN:

```
10 PRINT "ABC" "DEF";"GHI","JKL";
20 PRINT "MNO";CHR$(13);"PQR"
```

Was Sie auf dem Bildschirm sehen werden, ist folgendes:

```
ABCDEFGHI          JKLMNO

PQR
```

Bei der Untersuchung des zweizeiligen Programms und des Ausdrucks sehen wir, daß zwei Posten direkt (ohne Zwischenraum) aufeinander folgen, wenn Sie ohne Zeichensetzung eingegeben werden. Dasselbe passiert, wenn ein Semikolon zwischen zwei Posten eingefügt wird. Dabei ist es belanglos, ob der zweite Posten

bereits zum zweiten PRINT#-Statement gehört. Ein Komma rückt Cursor-rechts-Zeichen (nicht Leerzeichen) zwischen die zwei Posten, obwohl dies auf dem Bildschirm nicht sichtbar ist. Schließlich bewirkt die Eingabe des Wagenrücklauf-Zeichens (Code 13), daß die Posten in eine neue Zeile geschrieben werden, d. h. von den übrigen völlig getrennt. Dies sind zwar alles recht brauchbare Informationen für das Schreiben auf dem Bildschirm, aber sie können, wenn sie nicht ständig überwacht werden, sehr große Probleme bereiten, wenn man Daten in eine sequentielle Datei schreiben will.
Um zu sehen, welchen Effekt die verschiedenen Interpunktionszeichen haben, geben Sie das folgende Programm ein und starten es mit RUN.

PRINT#-INTERPUNKTIONS-TESTPROGRAMM

```
10 A$ = "AAA" : B$ = "BBB" : C$ = "CCC"
 : A = 100 : B = 200 : C = 300
20 R$ = CHR$(13)
1000 REM**********************************
1010 OPEN 1,8,10,"SEQTEST,S,W"
1020 PRINT#1,A$B$C$
1030 PRINT#1,A$;B$;C$
1040 PRINT#1,A$,B$,C$
1050 PRINT#1,A$R$B$R$C$
1060 PRINT#1,A$ R$ B$ R$ C$
1070 PRINT#1,A$;R$;B$;R$;C$
1080 PRINT#1,A$,R$,B$
1090 PRINT#1,A;B;C
1100 PRINT#1,A,B,C
1110 PRINT#1,AR$,BR$,C
1120 PRINT#1,A R$ B R$ C
1130 PRINT#1,A;R$ B;R$ C
1140 CLOSE 1
2000 REM**********************************
2010 OPEN 1,8,6,"SEQTEST"
2020 INPUT#1,A$ : PRINT ">" A$ "<"
2030 INPUT#1,A$ : PRINT ">" A$ "<"
2040 INPUT#1,A$ : PRINT ">" A$ "<"
2050 INPUT#1,A$,B$,C$ : PRINT ">" A$
 "<>" B$ "<>" C$ "<"
2060 INPUT#1,A$,B$,C$ : PRINT ">" A$
```

```
 "<>" B$ "<>" C$ "<"
2070 INPUT#1,A$,B$,C$ : PRINT ">" A$
 "<>" B$ "<>" C$ "<"
2080 INPUT#1,A$,B$ : PRINT ">" A$ "<>"
 B$ "<"
2090 INPUT#1,A : PRINT ">" A "<"
2100 INPUT#1,A : PRINT ">" A "<"
2110 INPUT#1,A : PRINT ">" A "<"
2120 INPUT#1,A : PRINT ">" A "<"
2130 INPUT#1,A,B,C : PRINT ">" A "<>"
 B "<>" C "<"
2140 CLOSE 1
```

Der Ausdruck des PRINT#-Interpunktionstests

```
>AAABBBCCC<
>AAABBBCCC<
>AAA            BBB            CCC<
>AAA<>BBB<>CCC<
>AAA<>BBB<>CCC<
>AAA<>BBB<>CCC<
>AAA            <>BBB<
> 100200300 <
> 100200300 <
> 300 <
> 300 <
> 100 <> 200 <> 300 <
```

ERKLÄRUNG DER AUSGABE DES PRINT#-PROGRAMMS

Die Untersuchung der Programmausgabe wird Ihnen einige der mit der Ein- und Ausgabe von Daten in bzw. von sequentiellen Files verbundenen Fehlerquellen deutlicher machen. Jede Zeile entspricht dem, was durch eine der PRINT#-Zeilen in der ersten Hälfte des Programms in die Datei geschrieben wurde. Jeder wirkliche String, der geschrieben oder gelesen wird, ist durch ein ">" am Anfang und ein "<" am Ende gekennzeichnet. In der zweiten Hälfte des Programms liest jede INPUT#-Zeile das von einer PRINT#-Zeile geschriebene wieder ein. Trotzdem erfordert das Warum hinter den Übereinstimmungen manchmal einiges Nachdenken:

Zeilen 1020 und 2020: Da es keine Interpunktion gibt, werden die drei Strings als ein String gespeichert.

Zeilen 1030 und 2030: Die Semikolons bewirken dasselbe wie oben.

Zeilen 1040 und 2040: Die Kommas ergeben wieder ein String, nur daß die Elemente dieses Mal durch Leerzeichen getrennt sind.

Zeilen 1050 und 2050: Die Trennung der zur Ausgabe bestimmten Posten durch das Wagenrücklauf-Zeichen CHR$(13) bewirkt, daß sie als getrennte Posten gespeichert werden.

Zeilen 1060 und 2060; 1070 und 2070: Vorausgesetzt, daß das Wagenrücklauf-Zeichen mit eingegeben wird, unterscheidet sich die Trennung mit Leerzeichen oder Semikolons nicht von den Ergebnissen in 1050/2050.

Zeilen 1080 und 2080: Wird das Wagenrücklauf-Zeichen mit eingegeben, werden aber die einzelnen Posten durch Kommas getrennt, bewirkt dies, daß die Posten separat abgespeichert werden, A$ jedoch als A$ plus dem Leerraum abgespeichert wird, der durch das Komma vor dem den String beendenden Wagenrücklauf-Zeichen zustande kommt. Beachten Sie, daß dem letzten String, B$, *kein* Leerraum durch das auf den Wagenrücklauf-Zeichen (R$) folgende Komma vorangestellt ist. Der Leerraum, der durch dieses Komma erzeugt wird, *wurde* zwar auf der Diskette gespeichert, jedoch streicht das Betriebssystem des C 64 beim INPUT#-Befehl automatisch den einem String vorangestellten Leerraum.

Zeilen 1090 und 2090: Numerische Werte werden bei der Verwendung von Semikolons wie die Strings behandelt – derselbe Effekt würde auch bei Leerräumen oder gar keiner Interpunktion zu beachten sein.

Zeilen 1100 und 2100: Gebraucht man Kommas bei numerischen Variablen, treten bei der Abfrage interessanterweise dieselben Effekte auf, wie in der vorigen Zeile.

Zeilen 1110 und 2110: Das Ergebnis dieser Zeile verdient einige Aufmerksamkeit, da es ein wichtiges Merkmal der Diskettenspeicherung illustriert, nämlich daß die Diskette keine Leerstrings speichert. Die zwei nicht-existenten Strings AR$ und BR$ stehen im PRINT#-Statement *vor* der numerischen Variablen C. Wenn wir jedoch den INPUT#-Befehl gebrauchen, um die mit A bezeichneten numerischen Variablen wieder einzulesen, finden wir anstelle eines TYPE MISMATCH ERRORs (der anzeigt, daß wir versucht haben, eine Zahl einzugeben, wo der nächste Posten

eigentlich ein String sein mußte), daß die Strings AR$ und BR$ nicht auf die Diskette aufgezeichnet wurden.

Zeilen 1120 und 2120: Auch wenn man die numerischen Werte mit Leerzeichen von den Wagenrücklauf-Zeichen trennt, ändert sich nichts, da die Leerzeichen von der ROM-Routine, die die Zeile für den BASIC-Interpreter analysiert, nicht akzeptiert wird. Das Ergebnis ist dasselbe wie in der letzten Zeile.

Zeilen 1130 und 2130: Diese Zeilen zeigen, daß korrekt voneinander getrennte numerische Werte auch getrennt auf die Diskette geschrieben werden. Beachten Sie, daß die Trennung entweder durch ein Interpunktionszeichen oder, im Falle von Strings, durch das "$"-Zeichen, welches den Variablennamen abschließt, vorgenommen werden kann.

ANDERE REGELN FÜR PRINT# UND INPUT#

1. Zwischen dem Namen, mit dem ein Posten abgespeichert wird, und dem Namen, mit dem er gelesen wird, gibt es keine Verbindung. Ein Posten, der z. B. mit A$ abgespeichert wurde, kann mit jedem zulässigen Stringvariablennamen wieder eingelesen werden.

2. Posten, die als numerische Variablen gespeichert wurden, können jederzeit als Strings eingelesen werden. Als String gespeicherte Posten können dann als numerische Variable abgerufen werden, wenn der String ein zulässiges numerisches Format hat, z. B. "1234".

3. Der längste String, der mit PRINT# auf eine Diskette geschrieben werden kann, darf höchstens 255 Zeichen lang sein.

4. Der längste String, der mit INPUT# von der Diskette eingelesen werden kann, darf höchstens 88 Zeichen lang sein. Längere Strings müssen mit GET# (siehe nächsten Abschnitt) eingelesen werden.

5. Leerstrings werden *nicht* auf der Diskette aufgezeichnet. Die Diskette speichert weder die Art, in der die Posten getrennt sind, noch die Namen der Posten.
All dies übernimmt das BASIC des C 64. Versucht man der Diskette ein Leerstring zu übermitteln, wird kein Zeichen übertragen und kein Zeichen gespeichert.

6. Zwar können Stringvariablen mit vorangestellten Leerzeichen auf die Diskette geschrieben werden, doch werden diese beim Wiedereinlesen vom C 64 gestri-

chen. Diese Eigenschaft kann auch ein Vorteil sein, um die Einschränkungen aus Punkt 5 zu umgehen. Sollte es wichtig sein, eine leere Stringvariable in einer Variablenkette unterzubringen, kann dies mit einem Leerzeichen geschehen. Es wird korrekt gespeichert und kann neu geladen werden, um wieder als Leerstring erkennbar zu sein.

7. Verschiedene Zeichen des Zeichensatzes, der auf der Diskette gespeichert werden kann, können nicht mit INPUT# geladen werden. Es sind unter anderem die Zeichen mit den folgenden ASCII-Codes:

0 / 13 / 32 (Falls in Form eines vorangestellten Leerzeichens) / 34 /44 / 58

Diese Einschränkung macht es fast unmöglich, Maschinensprache-Anweisungen zu speichern oder numerische Daten in Stringform mit INPUT# abzurufen. Im allgemeinen können alle Zeichen, die sichtbar gemacht werden können, einschließlich der Farb- und Cursorsteuer-Zeichen (also alle Zeichen, die normalerweise zum Schreiben auf dem Bildschirm verwendet werden), auf die Diskette geschrieben und dann durch INPUT# gelesen werden.

4. GET#

Genau wie PRINT und INPUT einen entsprechenden Befehl für den Umgang mit der Floppy Disk haben, hat auch der BASIC-Befehl GET, der ein einzelnes Zeichen von der Tastatur liest, ohne RETURN zu benötigen, einen entsprechenden Diskettenbefehl, nämlich GET#. Die Aufgabe von GET# ist, das nächste Zeichen von der Stelle der Diskette einzulesen, die der interne Zeiger des Laufwerks anzeigt. Nachdem das Zeichen eingelesen wurde, wird der Zeiger um eins erhöht. Genau wie mit GET können einstellige Zahlen mit GET# gelesen werden, obwohl dies kein Vorteil gegenüber INPUT# darstellt. Ein wesentlicher Nachteil aber ist, daß die Anweisung, mit GET# eine numerische Variable zu lesen, ein SYNTAX ERROR hervorruft, wenn diese einem nicht-numerischen Zeichen begegnet. Im allgemeinen wird GET#, wie auch GET, für das Lesen von Zeichen gebraucht, die wie Stringvariablen behandelt werden sollen.
Die Vorteile des GET#-Befehls sind, daß es mit ihm möglich ist, Zeichen von der Diskette zu lesen, die vom normalen INPUT#-Befehl abgelehnt werden. Eine Ausnahme ist CHR$(0). Außerdem können auf diese Weise auch Strings, die länger als 88 Zeichen sind, von der Diskette gelesen werden. Das folgende Programm demonstriert einige Unterscheide zwischen INPUT# und GET#.

AUSDRUCK DES GET#-TESTPROGRAMMS

```
10 OPEN 8,8,15,"S0:GET TEST" : CLOSE 8
1000 REM**********************************
1010 OPEN1,8,2,"GET TEST,S,W"
1020 PRINT#1,"    AAA"
1030 PRINT#1,"AAA♣♣♣BBB"
1040 PRINT#1,"AAA:BBB"
1050 PRINT#1,"123"
1060 PRINT#1,123
1070 A$ = "AAAA" : A$ = A$+A$ : A$ = A$+
A$ : A$ = A$+A$ : A$ = A$+A$ : A$ = A$+A
$
1080 PRINT#1,A$
1090 CLOSE 1
2000 REM**********************************
2005 PRINT "■LADEN DER DATEN MIT GET#■"
2010 OPEN 1,8,2,"GET TEST,S"
2020 A$ = ""
2030 GET#1,T$ : IF T$<>CHR$(13) THEN A$
= A$+T$ : GOTO 2030
2040 PRINT ">" A$ "<"
2050 A$ = ""
2060 GET#1,T$ : IF T$<>CHR$(13) THEN A$
= A$+T$ : GOTO 2060
2070 PRINT ">" A$ "<"
2080 A$ = ""
2090 GET#1,T$ : IF T$<>CHR$(13) THEN A$
= A$+T$ : GOTO 2090
2100 PRINT ">" A$ "<"
2110 GET#1,A : PRINT ">" A "<"
2120 GET#1,A : PRINT ">" A "<"
2130 GET#1,A : PRINT ">" A "<" : GET#1,A
$
2140 GET#1,A$ : GET#1,A : PRINT ">" A "<
" : INPUT#1,A$
2150 A$ = ""
2160 GET#1,T$ : IF T$<>CHR$(13) THEN A$
= A$+T$ : GOTO 2160
2170 PRINT ">" A$ "<"
```

```
2180 CLOSE 1
3000 REM**********************************
3005 PRINT "■LADEN DER DATEN MIT INPUT#■
"
3010 OPEN 1,8,2,"GET TEST,S"
3020 INPUT#1,A$ : PRINT ">" A$ "<"
3030 INPUT#1,A$ : PRINT ">" A$ "<"
3040 INPUT#1,A$ : PRINT ">" A$ "<"
3050 INPUT#1,A : PRINT ">" A "<"
3060 INPUT#1,A : PRINT ">" A "<"
3070 INPUT#1,A$ : PRINT ">" A$ "<"
3080 CLOSE 1
3090 END
```

An dieser Stelle stoppt das Programm mit einer STRING TOO LONG ERROR IN 2070 Fehlermeldung. Sie sollten CLOSE 1 [RETURN] eingeben, um die Datei zu schließen.

ERKLÄRUNGEN ZUM AUSDRUCK DES GET#-TESTPROGRAMMS

Der Zweck dieses Programms war, einige Eigenschaften des GET# und die Unterschiede zwischen INPUT# und GET# darzustellen. Wie beim vorigen Programm Interpunktions-Test, wollen wir auch hier jedes Zeilenpaar der einzelnen PRINT#–, INPUT# und GET#-Anweisungen kommentieren.

Zeilen 1020; 2020–2040; 3020: Zunächst fällt einmal auf, daß das Lesen eines Postens mit GET# relativ kompliziert ist. Es erfordert, im Vergleich zu der einen einfachen Anweisung für INPUT#, gleich zwei Programmzeilen.
Die Aufgabe der GET#-Zeilen ist es, so lange Zeichen von der Diskette zu lesen und sie an einen vormals leeren A$ anzufügen, bis ein Wagenrücklauf-Zeichen (CHR$(13)) auftaucht. Im Gegenstaz zu INPUT#, akzeptiert GET# die zum A$ gehörenden Leerzeichen.

Zeilen 1030; 2050; 2070; 3030: Bei keiner Methode ergeben sich Probleme, nomale Grafik-Zeichen zu lesen.

Zeilen 1040; 2080; 2100; 3040: Bei GET# gibt es keine Probleme, nur INPUT# akzeptiert den Stringteil nicht, der hinter dem Doppelpunkt steht (dies würde auch bei einem Komma passieren).

Zeilen 1050; 2110–2130; 3050: Obwohl sie als Stringvariable gespeichert war, wird die numerische Variable A von INPUT# gelesen – ein weiterer Beweis dafür, daß auf der Diskette selbst keine Unterschiede gemacht werden. Will man aber mit GET# lesen, muß jedes Zeichen des Strings einzeln eingelesen werden, obwohl es wie eine Zahl behandelt wurde. Beachten Sie das GET#1, A$ am Ende der dritten Zeile (2130). Dieses muß unbedingt hinzugefügt werden, um das Wagenrücklauf-Zeichen am Ende des Strings auf der Diskette zu berücksichtigen.

Zeilen 1060; 2140; 3060: Wo ein Posten als numerische Variable auf der Diskette gespeichert wurde, gibt es keinen Unterschied zur vorigen Methode, wo er als String gespeichert wurde. INPUT# liest die Zahl in einem Arbeitsgang, während GET# immer nur ein Zeichen der Zahl auf einmal aufnimmt.

Zeilen 1070–1080; 2150–2170; 3070: Der PRINT#-Abschnitt des Programms erzeugt einen 128 Zeichen langen String und schreibt ihn auf die Diskette, GET# liest diesen String, aber INPUT# ruft eine STRING TOO LONG ERROR Fehlermeldung hervor, weil der String länger als 88 Zeichen ist.

GET# UND DIE SPEICHERUNG VON ZAHLEN IN STRINGS

Eine weitere Anwendungsmöglichkeit für den GET#-Befehl sollte noch erwähnt werden, weil er zur Reduzierung des für die Speicherung von numerischen Daten auf der Diskette beanspruchten Platzes von einiger Wichtigkeit sein kann. Er wird daher häufig zur Einsparung von Speicherplatz verwendet, der von numerischen Variablen ausgefüllt ist. Ein-Byte-Werte, das sind Zahlen zwischen 0 und 255, können auf sehr einfache Art in Form von Stringzeichen gespeichert werden, wobei jedes einzelne Zeichen des Strings, basierend auf dem ASCII-Code des Zeichens, einen Wert im Bereich von 0 bis 255 darstellt. Werte, die mehr als ein Byte erfordern, können durch mehrere Zeichen ausgedrückt werden.
Wie schon erwähnt, hat GET# gegenüber INPUT# den Vorteil, daß es eine größere Anzahl Zeichen von der Diskette akzeptiert. Eine einzige Ausnahme ist CHR$(0). Werden Zahlen in Form von Strings gespeichert, wird das Unvermögen mit CHR$(0) umzugehen, zu einem Hauptnachteil. Eine Zeile wie die folgende überwindet dieses Problem.

```
100 GET#1, T$ : T$=LEFT$(T$+CHR$(0),1)
```

Sobald ein Zeichen von der Diskette geholt wird, wird das CHR$(0) hinzugefügt, um in den meisten Fällen ein Zwei-Zeichen-String zu bilden. Das äußerst linke Zeichen dieses Strings wird dann für das von der Diskette geholte Zeichen gehalten. In der

Mehrzahl der Fälle ändert dies nichts an den von der Diskette geholten Zeichen. Im Falle von CHR$(0) aber liest GET# nur ein Leerstring. Wird aber das CHR$(0) angehängt und das äußerst linke Zeichen herausgezogen, ist das Ergebnis das CHR$(0), das an den Leerstring angehängt wurde. Das Resultat ist auch CHR$(0), wenn GET# auf der Diskette gewesen sein muß. Die Anwendung dieser Methode wird ausführlich im Beispielprogramm am Ende des Kapitels demonstriert.

5. DAS HERAUSFINDEN DES DATEIENDES

Beim Einlesen von auf der Diskette gespeicherten Daten ist es natürlich auch nötig zu wissen, wann das Ende der Daten erreicht wurde. Eine Methode, dies herauszubekommen, ist die wirkliche Datenmenge festzustellen, bevor Sie beginnen. Eine zweite ist, sich von der Floppy Disk mitteilen zu lassen, wenn man Gefahr läuft, über das Ende hinauszulesen.

PROGRAMM ZUM HERAUSFINDEN DES DATEIENDES

```
10 OPEN 8,8,15,"S0:ENDE DER DATEI" : CLO
SE 8
1000 REM*********************************
1010 OPEN 1,8,2,"ENDE DER DATEI,S,W"
1020 ITEMS = 20 : PRINT#1,ITEMS
1030 FOR I = 0 TO ITEMS-1
1040 PRINT#1,I
1050 NEXT I
1060 CLOSE 1
2000 REM*********************************
2010 PRINT "■■DATEIENDE-FESTSTELLUNG DUR
CH FESTGELEGTE";
2015 PRINT "DATENANZAHL■■"
2020 OPEN 1,8,2,"ENDE DER DATEI,S"
2030 INPUT#1,ITEMS
2040 FOR I = 0 TO ITEMS-1
2050 INPUT#1,X : PRINT X ; "/" ;
2060 NEXT
2070 CLOSE 1
2080 PRINT
```

```
3000 REM**********************************
3010 PRINT "■■DATEIENDE-FESTSTELLUNG DUR
CH ABFRAGE"
3015 PRINT "DER STATUSVARIABLEN ST■■"
3020 OPEN 1,8,2,"ENDE DER DATEI,S"
3030 INPUT#1,ITEMS
3040 INPUT#1,X : SS = ST : PRINT X ; "/"
 ;
3050 IF SS<>64 THEN 3040
3060 CLOSE 1
3070 PRINT
```

Das erste Modul des Programms schreibt die Zahl 20 und dann eine Reihe von zwanzig Zahlen in die Datei. Die Schleife, die zum Schreiben der Werte verwendet wird, beginnt bei 0 und endet bei dem Wert ITEMS-1 (zu deutsch Posten-1), weil die zu schreibenden Werte in den meisten Fällen aus einem Feld (z. B. PRINT#, ARRAY(I)) genommen werden, wobei 20 Posten die Positionen 0 bis 19 im Feld beanspruchen.
Das zweite Modul arbeitet, indem es zuerst die ITEMS-Werte zurück in die Variable und dann die angegebene Anzahl von Zahlen von der Diskette einliest. In einem funktionstüchtigen Programm würden die Werte normalerweise in ein Feld, z. B. in INPUT#1, ARRAY(I), zurückgelesen.
Das dritte Modul arbeitet, indem INPUT# die Posten aufnimmt, und nachdem jeder Posten gelesen ist, den Wert der Systemvariable ST abfragt. Ist der von der Floppy Disk gelesene Wert von ST gleich 64, ist der zuletzt eingegebene Posten der letzte der Datei. Im allgemeinen ist es besser, ST sofort nach INPUT# (oder GET#) zu überprüfen, um sicherzugehen, daß deren Wert nicht durch eine andere Operation verändert wurde, bevor sie abgefragt wird. Beachten Sie, daß in diesem dritten Modul der Wert von ITEMS durch die Anwendung von INPUT# umgangen werden muß. Normalerweise wäre, falls die Methode mit ST angewendet worden wäre, ITEMS gar nicht erst auf die Diskette geschrieben worden.
Der Einsatz von ST anstelle einer vom Programm erzeugten Variable zum Herausfinden des Dateiendes wird dann wichtig, wenn der COPY-Systembefehl verwendet wird, um sequentielle Dateien aneinander zu ketten. Sequentielle Dateien können mit COPY ohne weiteres verbunden und die erzeugte Datei normal gelesen werden. Deutlich ist, daß es nicht wünschenswert sein kann, eine Variable wie ITEMS in der Mitte so einer Datei stecken zu haben. Deswegen gibt es in solchen Fällen keine andere Wahl, als die ST-Methode zum Herausfinden des Dateiendes zu verwenden.

6. DAS BEENDEN DER AUS- ODER EINGABE MIT CLOSE

Es ist wichtig, nach Beendigung der Aus- oder Eingabe, die Datei mit CLOSE zu schließen.

Je nachdem, ob die Datei für die Aus- oder Eingabe bestimmt ist, bindet das Offenlassen die der Datei zugeordnete Filenummer, und jeder Versuch, eine andere Datei mit derselben Nummer zu öffnen, ruft eine FILE-OPEN-Fehlermeldung hervor. Viel wichtiger ist, daß, wenn eine Ausgabe-Datei nicht sofort geschlossen wird, die Chance besteht, daß das Programm, vielleicht wegen eines Fehlers im BASIC-Programm, mit der immer noch offenen Datei abstürzt. In diesem Fall wird die Datei auf der Diskette als ungeschlossen gekennzeichnet und bei der nächsten Ausführung des VALIDATE-Befehls gelöscht.

Um herauszubekommen, ob Sie irgendwelche ungeschlossenen Dateien auf der Diskette haben, brauchen Sie einfach nur das Directory zu laden und den Filetyp für jede Datei zu untersuchen. Vor jedem Filetyp einer nicht korrekt geschlossenen Datei steht ein Sternchen (*). Eine Datei in diesem Zustand kann kaum noch gerettet werden.

7. ANWENDUNGSBEISPIELE FÜR DIE SEQUENTIELLE DATEI

In diesem Abschnitt wollen wir einige Beispiele dafür vorstellen, wie man sequentielle Dateien zur Abspeicherung verschiedener Datenarten benutzt. Bedienen Sie sich der in dem unten abgedruckten Programm dargestellten Methoden. So sollte es Ihnen nicht sehr schwerfallen, die sequentiellen Dateien zur Datenspeicherung in einem Programm für normale Anwendungen einzubauen.

TESTPROGRAMM FÜR SEQUENTIELLE DATEIEN

```
10 OPEN 8,8,15,"S0:TESTDATEI" : CLOSE 8
1000 REM**********************************
1010 REM INITIALISIEREN DER ARRAYS
1020 CLR
1030 R$ = CHR$(13)
1040 DIM ARRAY$(100),ARRAY(100)
1050 A$ = ""
1060 FOR I = 0 TO 100
1070 IF I/5=INT(I/5) THEN A$ = A$+"A"
```

```
1080 ARRAY$(I)=A$
1090 NEXT I
1100 FOR I = 0 TO 100
1110 ARRAY(I) = I
1120 NEXT I
1130 NUMBER$ = "9876543210123456789"
1140 CODE$ = ""
1150 FOR I = 0 TO 100
1160 CODE$ = CODE$+CHR$(I)
1170 NEXT I
1180 V1 = 1111
1190 V2 = 222
1200 V3 = 33
1210 VB$ = "BBBB"
1220 VC$ = "CCCCCCC"
1230 VD$ = "DDD"
1240 ITEMS = 101
2000 REM#********************************
2010 OPEN 1,8,2,"TESTDATE I,S,W"
2020 PRINT#1,ITEMS;R$;V1;R$;VB$
2030 FOR I = 0 TO ITEMS-1
2040 PRINT#1,ARRAY$(I)
2050 NEXT I
2060 PRINT#1,V2;R$;VC$
2070 FOR I = 0 TO ITEMS-1
2080 PRINT#1,ARRAY(I)
2090 NEXT I
2100 PRINT#1,NUMBER$
2110 PRINT#1,CODE$
2120 PRINT#1,V3;R$;VD$
2130 CLOSE 1
3000 REM#********************************
3010 OPEN 1,8,2,"TESTDATE I,S"
3020 INPUT#1,ITEMS,V1,VB$
3030 FOR I = 0 TO ITEMS-1
3040 INPUT#1,ARRAY$(I)
3050 NEXT I
3060 INPUT#1,V2,VC$
3070 FOR I = 0 TO ITEMS-1
3080 INPUT#1,ARRAY(I)
3090 NEXT I
```

```
3100 INPUT#1,NUMBER$
3110 CODE$ = ""
3120 FOR I = 0 TO ITEMS-1
3130 GET#1,T$ : T$ = LEFT$(T$+CHR$(0),1)

3140 CODE$ = CODE$+T$
3150 NEXT
3160 INPUT#1,V3,VD$
3170 CLOSE 1
4000 REM##############################
4010 FOR I = 0 TO ITEMS-1
4020 PRINT ">" ARRAY$(I) "<"
4030 NEXT
4040 INPUT"■BITTE [REVERS EIN]RETURN[REV
ERS AUS] DRUECKEN ";T$
4050 PRINT
4060 FOR I = ITEMS-1 TO 0 STEP -1
4070 PRINT ARRAY(I) "/" ;
4080 NEXT
4090 PRINT
4100 INPUT"■BITTE [REVERS EIN]RETURN[REV
ERS AUS] DRUECKEN ";T$
4110 PRINT
4120 FOR I = 1 TO LEN(CODE$)
4130 PRINT ASC(MID$(CODE$,I,1))"/" ;
4140 NEXT I
4150 PRINT
4160 INPUT"■BITTE [REVERS EIN]RETURN[REV
ERS AUS] DRUECKEN ";T$
4170 PRINT
4180 PRINT "NUMBER$ = " NUMBER$
4190 PRINT "V1 =" V1
4200 PRINT "V2 =" V2
4210 PRINT "V3 =" V3
4220 PRINT "VB$ =" VB$
4230 PRINT "VC$ =" VC$
4240 PRINT "VD$ =" VD$
4250 END
```

ERKLÄRUNG DES TESTPROGRAMMS FÜR DIE SEQUENTIELLEN DATEIEN

Die Aufgabe dieses Programms ist, verschiedene Daten von Feldern, die auch in Programmen vorkommen könnten, auf die Diskette zu schreiben, diese Daten dann von der Diskette zu lesen und sie in die Felder zurückzuschreiben.
Das erste Modul, die Zeilen 1000–1240, erstellt die Felder. Das Feld ARRAY$ (ARRAY, englisch für Feld) enthält 101 Strings, die wiederum aus einer zunehmenden Anzahl von As bestehen, wobei die Anzahl von As nach jeweils fünf abgespeicherten Strings um eins zunimmt. ARRAY enthält die Zahlen 0–101. Der String NUMBER$ enthält die Zeichen "9876543210123456789". Der String CODE$ enthält 101 Zeichen, deren ASCII-Codes 0–100 sind. Der Zweck von CODE$ ist, die Methode der Speicherung von Ein-Byte-Zahlen in Strings zu demonstrieren; eine Methode, die sehr ökonomisch mit dem Speicher umgeht. Jedes Zeichen aus CODE$ wird wichtig sein, nicht so sehr als Zeichen, sondern wegen seines ASCII-Codes, der einen abzuspeichernden Wert repräsentiert. Zusätzlich gibt es noch sieben andere Variablen, deren Wert in den Zeilen 1180–1240 stehen. Eine der Variablen ist ITEMS, dessen Wert 101 die Anzahl der in jedem der Felder abzuspeichernden Datenposten wiedergibt. Der Wert von ITEMS würde normalerweise vom Programm selbst bestimmt, wenn neue Datenposten hinzugefügt oder gelöscht werden.
Das zweite Modul, die Zeilen 2000–2130, schreibt den Inhalt der Felder und Werte auf die Diskette.
Das komplizierteste Modul ist ohne Zweifel das dritte, die Zeilen 3000–3170. Die Aufgabe dieses Moduls ist, mit der in diesem Kapitel entwickelten Methode die Daten von der Diskette zu lesen und sie zurück in die verschiedenen Felder zu schreiben.
Das vierte Modul, die Zeilen von 4000 an aufwärts, hat die einfache Aufgabe, die gelesenen Daten auf den Bildschirm zu schreiben, um den Erfolg des Vorgangs zu veranschaulichen.

ZUSAMMENFASSUNG DER USER-DATEIEN

1. User-Dateien sind ein wichtiges Werkzeug für das erste Programmieren, welches fast immer die Notwendigkeit einschließt, mehr oder weniger komplexe Daten von einem zum anderen Programmablauf abzuspeichern.

2. User-Dateien müssen mit einem Befehl wie dem folgenden eröffnet werden:

```
OPEN 1,8,2,"USERFILE,U,W"
```

welches folgende Angaben einschließt:

a) eine nur einmal vorkommende Filenummer

b) die Gerätenummer (normalerweise 8)

c) eine nur einmal vorkommende Kanalnummer (2–14)

d) den Namen der Datei, einschließlich der Endung ",U", um die User-Datei zu kennzeichnen.

e) die Angabe, ob die Datei für die Aus- oder Eingabe bestimmt ist (",R" oder ",W").

3. Daten werden mit Hilfe des PRINT#-Statements in die Datei geschrieben.

4. Daten werden von den User-Dateien entweder mit INPUT# oder mit GET# gelesen.

5. User-Dateien müssen mit CLOSE geschlossen werden, wenn sie nicht mehr benötigt werden, sonst kann es passieren, daß sie auf der Diskette unbrauchbar gemacht werden und die Daten, die sie enthalten, verloren gehen.

ZUSAMMENFASSUNG DER SEQUENTIELLEN DATEIEN

1. Sequentielle Dateien sind ein wichtiges Werkzeug für das erste Programmieren, welches fast immer die Notwendigkeit einschießt, mehr oder weniger komplexe Daten von einem anderen Programmablauf abzuspeichern.

2. Sequentielle Dateien müssen mit einem Befehl wie dem folgenden geöffnet werden:

```
OPEN1,8,2,"SEQFILE,S,W"
```

welches folgende Angaben einschließt:

a) eine nur einmal vorkommende Filenummer

b) die Gerätenummer (normalerweise 8)

c) eine nur einmal vorkommende Kanalnummer (2–14)

d) den Namen der Datei, einschließlich der Endung ",S", um die sequentielle Datei zu kennzeichnen.

e) die Angabe, ob die Datei für die Aus- oder Eingabe bestimmt ist (",R" oder ",W")

3. Daten werden mit Hilfe des PRINT#-Statements in die Datei geschrieben.

4. Daten werden von der sequentiellen Datei entweder mit INPUT# oder mit GET# gelesen.

5. Sequentielle Dateien müssen mit CLOSE geschlossen werden, wenn sie nicht mehr benötigt werden, sonst kann es passieren, daß sie auf der Diskette unbrauchbar gemacht werden und die Daten, die sie enthalten, verloren gehen.

KAPITEL 8

PROGRAMM-DATEIEN

1. *Was ist eine Programm-Datei?*
2. *Der Aufbau einer BASIC-Programm-Datei*
3. *Der Gebrauch von Programm-Dateien für andere Zwecke*
4. *Die Ausgabe von Programm-Dateien zum Drucker*
5. *Das Aneinanderhängen von Programmen mit Hilfe von Programm-Dateien auf Diskette*
6. *Das Neunumerieren einer Programm-Datei auf der Diskette*

Das Directory einer Diskette weist meist mindestens zwei Datei-Typen aus, sequentielle und Programm-Dateien. Bis jetzt haben wir über den einfachen Vorgang des Ladens (mit LOAD) und des Abspeicherns (mit SAVE) gesprochen und die sequentielle Datei etwas näher untersucht. Es gibt jedoch neben LOAD und SAVE noch viele andere sehr brauchbare Operationen, die mit Programm-Dateien ausgeführt werden können. Aber bevor Sie diese verstehen können, müssen wir Ihnen zuerst einmal die Eigenarten einer Programm-Datei, wie sie auf der Diskette gespeichert wird, erklären.
Was sind die Hauptunterschiede zwischen der Programm- und der sequentiellen Datei? Die Antwort ist recht einfach: Der Unterschied zwischen "PRG"-Dateien und "SEQ"-Dateien ist, daß die eine "PRG" und die andere "SEQ" heißt. Probieren Sie das folgende Experiment:

1. Geben Sie dieses einzeilige Programm ein:

```
10 REM DAS IST EIN SEQUENTIELLES PROGRAM
M
```

2. Tippen Sie:

```
SAVE "SEQPROG,S",8 [RETURN]
```

3. Wenn Sie nun das Directory der Diskette untersuchen, sehen Sie, daß es eine Datei mit dem Namen SEQPROG auf der Diskette gibt und daß es eine sequentielle Datei ist.

4. Tippen Sie:

```
NEW [RETURN]
```

um das Testprogramm aus dem Speicher zu löschen.

5. Tippen Sie:

```
LOAD "SEQPROG,S",8 [RETURN]
```

6. LISTen Sie das Programm, und Sie werden sehen, daß Sie gerade erfolgreich ein auf der Diskette als sequentielle Datei gespeichertes Programm geladen haben.

1. WAS IST EINE PROGRAMM-DATEI?

Wir haben ja schon deutlich demonstriert, daß ein Programm in Form einer sequentiellen Datei abgespeichert werden kann. Nun stellt sich aber die Frage: Was ist eine Programm-Datei? Die einfache Antwort ist, daß "PRG", das Kennzeichen einer Programm-Datei auf der Diskette, ein Markierungszeichen ist, welches *vom C 64* und nicht im Laufwerk selbst verwendet wird, um dem C 64 anzuzeigen, daß dies eine Datei ist, in der er normalerweise ein abgespeichertes Programm erwarten würde. Der C 64 erstellt immer eine "PRG"-Datei, wenn ihm befohlen wird, ein Programm (mit SAVE) abzuspeichern, außer es wird ein anderer Datei-Typ angegeben. Dann schreibt er die momentane BASIC-Startadresse und den Inhalt des Programmspeichers Byte für Byte in diese Datei, obwohl sich diese nicht von einer sequentiellen Datei unterscheidet, mit Ausnahme des Kennzeichens "PRG". Ähnlich sucht der C 64, wenn er angewiesen wird, ein Program (mit LOAD) zu laden, nach einer Programm-Datei mit dem angegebenen Namen und lädt deren Inhalt zurück in den Programmspeicher.
Zusammenfassend läßt sich sagen, daß eine Programm-Datei eine sequentielle Datei ist, die den Inhalt des BASIC-Speichers enthält und die spezielle Bezeichnung "PRG" trägt, welche den Datei-Typ anzeigt, auf den der BASIC-Interpreter beim Laden und Abspeichern (mit LOAD und SAVE) zugreift.

2. DER AUFBAU EINER BASIC-PROGRAMM-DATEI

Programm-Dateien geben den Aufbau von Programmen wieder. Um diesen Aufbau zu verstehen, müssen wir erst einmal etwas über die Art, in der Programme im Speicher abgelegt werden, sagen.

BASIC-Programme nehmen normalerweie einen bei Adresse 2049 beginnenden Bereich ein. Dort ist jede Zeile des Programms in der richtigen Reihenfolge repräsentiert. Der Bereich läßt sich in vier Abschnitte unterteilen:

1. Zwei Bytes, die "link bytes". Diese Zeichen enthalten die Startadresse der *nächsten* Programmzeile, d. h. das dem Ende der gegenwärtigen Zeile folgende Byte.

2. Zwei Bytes, die die Zeilennummer enthalten.

3. Eine unbestimmte Anzahl Bytes, die den Text der Zeile wiedergibt.

4. Ein Byte mit dem Wert Null, das das Ende der Zeile anzeigt.

Am Ende des Programms, hinter der die letzte Zeile abschließenden Null, stehen zwei weitere Bytes mit dem Wert Null, die dem BASIC-Interpreter als Kennzeichen dienen.
Wird ein Programm auf der Diskette abgespeichert, ist das Format fast genau dasselbe wie das im Speicher. Die einzige Ausnahme ist, daß, weil ein BASIC-Programm fast überall im Speicher anfangen kann, die Datei mit zwei Bytes beginnt, die aufzeichnen, wo der Start des Programms lag, als es abgespeichert wurde. Unten steht das Listing eines Programms, welches eine Programm-Datei von der Diskette liest, als wäre es eine normale sequentielle Datei. Im Moment ist es so aufgebaut, daß es seine eigene Programm-Datei liest, muß also zuerst eingegeben, (mit SAVE) abgespeichert werden, um dann die Diskette zu lesen, auf der es aubgespeichert wurde.

DER PROGRAMMLESER

```
10 OPEN 8,8,8,"PROG READ"
15 GET#8,T$ : GET#8,T$
20 GET#8,T$ : GET#8,T$
30 IF T$="" THEN 110
40 GET#8,T$ : T = ASC(T$+CHR$(0))
50 GET#8,T$ : T = ASC(T$+CHR$(0))*256+T
60 PRINT T " ";
80 GET#8,T$ : IF T$>"Z" THEN T$ = "["+MI
D$(STR$(ASC(T$)),2)+"]"
90 IF T$<>"" THEN PRINT T$ ; : GOTO 80
95 PRINT
```

```
100 GOTO 20
110 CLOSE 8
120 END
```

ERKLÄRUNG DES PROGRAMMLESERS

Zeile 15: Liest die ersten beiden Bytes der Datei – es passiert nichts mit ihnen.

Zeilen 20–30: Lesen ein Paar "link bytes. Ist das zweite Byte, welches das "high Byte" (höheres Byte) sein sollte, Null, muß das Ende der Datei erreicht worden sein.

Zeilen 40–60: Lesen und drucken die Zeilennummer aus.

Zeilen 80–90: Die Bytes, die die Zeile bilden, werden gelesen und einzeln ausgedruckt.

Einige von ihnen werden Ein-Byte-Tokens für Schlüsselwörter sein, die nicht druckfähig sind. Diese werden durch ihre ASCII-Werte in eckigen Klammern repräsentiert.

DER AUSDRUCK DES PROGRAMMLESERS

```
 15   [161]#8,T$ : [161]#8,T$
 20   [161]#8,T$ : [161]#8,T$
 30   [139] T$[178]"" [167] 110
 40   [161]#8,T$ : T [178] [198](T$[170][
199](0))
 50   [161]#8,T$ : T [178] [198](T$[170][
199](0))[172]256[170]T
 60   [153] T " ";
 80   [161]#8,T$ : [139] T$[177]"Z" [167]
 T$ [178] "[91]"[170][202]([196]([198](T
$)),2)[170]"[93]"
 90   [139] T$[179][177]"" [167] [153] T$
 ; : [137] 80
 95   [153]
 100  [137] 20
```

```
110   [160] 8
120   [128]
```

Der Programmausdruck ist nicht abgedruckt worden, um ihn im Detail zu analysieren, sondern um den groben Aufbau einer Programm-Datei darzustellen. Die Werte in eckigen Klammern sind die ASCII-Codes der "Tokens", die Abkürzungen der im Programm enthaltenen BASIC-Schlüsselwörter. Abgesehen von den Tokens können Sie sehen, daß der Rest des Programminhaltes in einem sehr unkomplizierten Format abgespeichert ist, fast so, wie Sie es auf dem Bildschirm sehen.

3. DER GEBRAUCH VON PROGRAMM-DATEIEN FÜR ANDERE ZWECKE

Genau wie sequentielle Dateien für das Abspeichern von Programmen verwendet werden können, können auch Programm-Dateien für andere Zwecke gebraucht werden. Es ist durchaus möglich, eine Programm-Datei zur Datenspeicherung einzusetzen. Der Befehl dazu könnte lauten:

```
OPEN 1,8,2,"DATASTORE,P,W"
```

Die Daten werden mit

```
OPEN 1,8,2,"DATASTORE,P,R"
```

gelesen.

Wurde die Datei korrekt eröffnet, kann sie mit allen im Kapitel über sequentielle Dateien beschriebenen Methoden beeinflußt werden.

Üblicherweise werden Programm-Dateien jedoch zur Abspeicherung von Speicherbereichen gebraucht, die zum Beispiel Programme in Maschinensprache oder die Bildschirmanzeige enthalten. Verglichen mit dem sehr mühsamen Vorgang, die Posten einzeln auf die Diskette zu schreiben und von ihr zu lesen, ist dies eine schnellere Methode zum Abspeichern und Einlesen von Daten. Nachstehend finden Sie eine kurze Routine, die den Inhalt des Bildschirmspeichers als Programm-Datei auf der Diskette abspeichert.

ROUTINE ZUM ABSPEICHERN DES BILDSCHIRMSPEICHERS ALS PROGRAMM-DATEI

```
14040 IF A=1 THEN END
14050 FOR I=0 TO 3 : A%(I)=PEEK(43+I) :
NEXT
14060 POKE 43,0 : POKE 44,4 : POKE 45,0
: POKE 46,8
14070 SAVE "SCREEN",8
14080 POKE 43,A%(0) : POKE 44,A%(1) : PO
KE 45,A%(2) : POKE 46,A%(3)
14090 PRINT "■"
15000 FOR I = 55296 TO 56319 : POKE I,1
: NEXT
15005 A = 1
15010 LOAD "SCREEN",8,1
```

Lassen Sie das Programmlisting auf dem Bildschirm, und starten Sie das Programm mit RUN (mit einer Diskette im Laufwerk). Das Laufwerk sollte nun anfangen zu laufen und den Bildschirmspeicher abspeichern. Dann sollte der Bildschirm gelöscht werden und das Programmlisting in Weiß zurückkehren. Schließlich endet das Programm, und der Cursor blinkt am oberen Rand des Bildschirms.

ERKLÄRUNG DER ROUTINE:

Zeilen 14050–14060: Zuerst werden die Register aufgezeichnet, die den Beginn und das Ende des BASIC-Programms enthalten, dann werden sie dahingehend geändert, daß sie den Beginn und das Ende des abzuspeichernden Speicherbereichs anzeigen – in diesem Fall den Bildschirmspeicher.

Zeile 14070: Eine einfache SAVE-Anweisung reicht nun aus, um den festgelegten Speicherbereich auf der Diskette als Programm abzuspeichern.

Zeile 14080: Die in Zeile 14050 aufgezeichneten Ausgangswerte der Register werden wieder eingesetzt, damit das System weiß, wo das BASIC-Programm wirklich ist. Beachten Sie, daß dies nicht mit einer Schleife geschehen kann, weil das System bei der Speicherstelle des Wertes der Schleifenvariable durch die vorübergehende Änderung des BASIC-Endes durcheinander gebracht würde. Mit Hilfe eines Feldes, in dem die Werte gespeichert werden, wird dieses Problem

umgangen, da wir das Register, das den Start der Felder im Speicher anzeigt, nicht verändert haben.

Zeilen 14090–15000: Nun wird der Bildschirm gelöscht. Weil wir den Inhalt des Farbspeichers nicht mit abspeichern, wird der Wert 1 in alle 1024 Bytes des Farbspeichers gePOKEt. Dies legt fest, daß alle Zeichen im Farbspeicher in Weiß erscheinen. Würde dies nicht geschehen, würde der wieder eingeladene Bildschirm unsichtbar sein.

Zeile 15010: Nun wird die Programm-Datei mit dem Bildschirm als Inhalt geladen. Der einzige Unterschied zwischen dieser Anweisung und dem normalen LOAD ist, daß hier eine Sekundäradresse 1 an das Ende gehängt wurde. Dies bewirkt, daß die Programm-Datei in genau denselben Bereich eingeladen wird, aus dem sie stammte. Beachten Sie, daß die Programm-Datei vom Programm aus eingeladen wird und nicht durch eine Anweisung im Direktmodus. Der Beginn und das Ende der BASIC-Zeiger werden nicht verändert, um die Position und den Umfang des Wiedereingeladenen wiederzugeben. Mit anderen Worten: Obwohl ein Programm mit dem Namen SCREEN (mit LOAD) eingeladen wird, bleibt das System dabei, *das* Programm als maßgeblich zu betrachten, welches das Einladen von SCREEN befohlen hat.

Zeilen 14040 und 15005: Diese zwei Zeilen scheinen nicht in die oben umrissene Entwicklung des Programms zu passen. Wenn Programm-Dateien durch ein Programm (mit LOAD) geladen werden, startet der C 64 automatisch das BASIC-Programm noch einmal, obwohl die Variablen hierbei nicht gelöscht werden. Um nicht in einer Endlosschleife gefangen zu werden, wird die Variable A auf eins gesetzt, nachdem der Bildschirm wieder geladen worden ist. Der Wert dieser Variablen wird als Markierung benutzt, um das System zu informieren, daß es nicht nötig ist, das Programm noch einmal zu informieren, daß es nicht nötig ist, das Programm noch einmal zu starten. Würde die Routine in ein größeres Programm eingebaut, würde die Zeile 14040 normalerweise mit der Ausführung zu der Stelle hinter dem Wiedereinladen von SCREEN zurückkehren, anstatt das Programm einfach zu beENDen.

Obwohl das Abspeichern des Bildschirms ein übersichtlicher und brauchbarer Vorgang sein kann, wird die Möglichkeit, Speicherbereiche abzuspeichern, bestimmt öfter dazu eingesetzt, Programme in Maschinensprache für den schnellen Zugriff auf Diskette zu speichern.

DIE AUSGABE VON PROGRAMM-DATEIEN ZUM DRUCKER

Wurden die Eigenschaften von Programm-Dateien einmal verstanden, wird es möglich, sie auf verschiedene Arten zu gebrauchen und zu beeinflussen – oft sogar weitaus bequemer, als dies mit einem Programm im Speicher geschehen könnte. Ein Beispiel wäre das Vorbereiten eines Programms auf den Ausdruck des Listings. Es gibt keinen Zweifel, daß Programmlistings, in denen die Steuerzeichen durch Abkürzungen in eckigen Klammern dargestellt werden, anstatt durch inverse Grafikzeichen, viel einfacher zu lesen sind. Diese Form wurde für die Programme in diesem Buch und in vielen anderen Commodore Programmierbüchern gewählt. Es ist möglich, eine Routine an ein Programm im Speicher zu hängen, um es in diesem Format zu listen, aber noch einfacher ist es, die in dem folgenden Programm verwendete Methode zu übernehmen, die das Programm von der Diskette liest und es auch von dort listet.

PROGRAMM FÜR DAS LISTEN MIT ECKIGEN KLAMMERN

```
10 GOTO 16000
5000 REM#******************************
5010 REM DISK FEHLER-STATUS
5020 REM******************************
5030 INPUT#15,EN,EM$,ET,ES
5040 IF EN=73 THEN 5030
5050 IF EN<20 THEN RETURN
5060 PRINT "■■■■ **********************
****************"
5070 PRINT "■■              DISK FEHLER"
5080 PRINT "■■      FEHLER -" EN " " EM$
5090 PRINT "■      AUF SPUR -" ET "UND SE
KTOR -"ES
5100 PRINT "■■   PROGRAMMAUSFUEHRUNG UNT
ERBROCHEN"
5110 PRINT "■■■ ************************
**************"
5120 CLOSE 15
5130 CLR
5140 END
13000 REM#******************************
13010 REM INITIALISIERUNG DES PROGRAMMS
13020 REM******************************
```

```
13030 DIM C$(255,1)
13040 FOR I = 0 TO 255 : C$(I,0) = CHR$(
I) : C$(I,1) = CHR$(I) : NEXT
13050 AD = 41118 : T1 = 128
13060 T$ = ""
13070 T = PEEK(AD) : AD = AD+1
13080 T$ = T$+CHR$(T AND 127)
13090 IF T<128 THEN 13070
13100 C$(T1,0) = T$ : T1 = T1+1
13110 IF PEEK(AD) THEN 13060
13120 RESTORE
13130 READ T$ : IF T$<>"FUER LIST" THEN
13130
13140 READ T
13150 IF T<0 THEN RETURN
13160 READ T$ : C$(T,1) = "["+T$+"]"
13170 GOTO 13140
14000 REM**********************************
14010 DATA "FUER LIST"
14020 REM**********************************
14030 DATA 5,WEISS,17,CRSR RUNTER,18,REV
ERS EIN,19,HOME
14040 DATA 20,DEL,28,ROT,29,RETURN,30,GR
UEN
14050 DATA 31,BLAU,129,ORANGE,133,F1
14060 DATA 134,F3,135,F5,136,F7,137,F2
14070 DATA 138,F4,139,F6,140,F8,144,SCHW
ARZ
14080 DATA 145,CRSR HOCH,146,REVERS AN,1
47,SCHIRM NEU
14090 DATA 148,INST,149,BRAUN,150,HELLRO
T
14100 DATA 151,GRAU 1,152,GRAU 2
14110 DATA 153,HELLGRUEN,154,HELLBLAU
14120 DATA 155,GRAU 3,156,PURPUR,157,CRS
C LINKS
14130 DATA 158,GELB,159,CYAN
14140 DATA -1
15000 REM**********************************
15010 REM AUFLISTEN
```

```
15020 REM*****************************
15030 OPEN 3,DEV,3,NR$ : GOSUB 5000
15040 OPEN 4,4
15050 GET#3,T$ : GET#3,T$
15060 GET#3,T$ : GET#3,T$
15070 QU = 0
15080 IF T$="" THEN CLOSE 3 : CLOSE 4 :
RETURN
15090 GET#3,T$ : T = ASC(T$+CHR$(0))
15100 GET#3,T$ : T = ASC(T$+CHR$(0))*256
+T
15110 PRINT#4,MID$(STR$(T),2) " " ;
15120 GET#3,T$
15130 IF T$=CHR$(34) THEN QU = 1-QU
15140 IF T$<>"" THEN PRINT#4,C$(ASC(T$),
QU) ; : GOTO 15120
15150 PRINT#4
15160 GOTO 15060
16000 REM*****************************
16010 REM EINLADEN DER DATEN
16020 REM*****************************
16030 GOSUB 13000
16040 INPUT "GERAETENUMMER ? 8■■■■";DEV
16050 OPEN 15,DEV,15
16060 INPUT "NAME DES PROGRAMMS ";NR$
16070 GOSUB 15000
16080 INPUT "WEITERE PROGRAMME AUSDRUCKE
N (J/N) ? N";T$
16090 IF T$="N" THEN CLOSE 15 : END
16100 IF T$<>"J" THEN PRINT "□□□" : GOTO
 16080
16110 GOTO 16060
```

ERKLÄRUNG DES PROGRAMMS FÜR DAS LISTEN MIT ECKIGEN KLAMMERN

Zeilen 16000–16110: Der Zweck dieses Moduls ist es, dem Besitzer zu ermöglichen, die Gerätenummer des Laufwerkes, auf dem das Programm gespeichert ist, anzugeben, den Namen der Datei zu nennen und den Fehlerkanal zu diesem Gerät zu öffnen. Wenn die Ausgabe des Programms zum Drucker beendet ist, wird dem Benutzer die Möglichkeit eingeräumt, eine weitere Datei ausdrucken zu lassen.

Zeilen 13000–14140: Das erste dieser beiden Module initialisiert das Programm, wobei es hauptsächlich Daten aus dem zweiten Modul benutzt.

Zeilen 13030–13040: Zuerst wird ein 256zeiliges Feld aufgebaut, in dem jede Zeile zwei Elemente enthält. Am Anfang werden beide Elemente mit dem Zeichen gleichgesetzt, das den gleichen ASCII-Wert besitzt wie die Zeilennummer im Feld (0–255).

Zeile 13050: AD ist die Startadresse der Tabelle mit BASIC-Schlüsselwörtern im Speicher. T1 wird auf 128 gesetzt. Dies ist der Wert des ersten Zeichens aus dem Zeichensatz, der die Tokens für die BASIC-Schlüsselwörter liefert.

Zeilen 13060–13090: Zeichen werden von der BASIC-Schlüsselwört-Tabelle gePEEKt und zu einem vorübergehenden String addiert, bis das Programm auf ein Zeichen mit einem Wert größer als 127 stößt. In der Schlüsselwort-Tabelle selbst wird das Ende eines jeden Schlüsselwortes durch den mit AND 128 verknüpften ASCII-Wert des letzten Zeichens kennzeichnet.

Zeile 13100: Beginnend in Zeile 128 des Feldes, genau wie die Schlüsselwort-Tokens bei CHR$(128) beginnen, wird das Schlüsselwort im Element Null der relevanten Zeile von C$ gespeichert.

Zeile 13110: Der Vorgang wird fortgeführt, bis ein Byte mit dem Wert Null gefunden wird, welches das Ende der Schlüsselwort-Tabelle markiert.

Zeilen 13120–13130: Der Anfang des DATA-Moduls ist mit dem Statement DATA FOR LIST gekennzeichnet. Diese Zeilen lesen und legen alle Daten bis zu diesem Punkt ab. In diesem Programm gibt es *keine* DATA-Statements vor Zeile 13130, aber Sie können die zwei Module dennoch in einem größeren Programm unterbringen.

Zeilen 13140–13170: Die Abkürzungen für die verschiedenen Steuerzeichen, die das Programm reformatieren soll, werden von den DATA-Statements gelesen und in das Element 1 der relevanten Zeile des Feldes C$ geschrieben. Bis das Modul seine Arbeit beendet hat, enthalten Null-Elemente von C$ schon die Schlüsselwörter, die den ASCII-Werten der Zeichen, die als Token eingesetzt wurden, entsprechen, so wie die "1"-Elemente die Darstellungen in eckigen Klammern der einzelnen Zeichen enthalten, die benutzt wurden, um die Steuerzeichen zu repräsentieren. Die zwei Arten müssen in zwei verschiedenen Hälften des Feldes aufgezeichnet werden, weil dieselben Zeichen, abhängig davon, ob sie in Anführungszeichen stehen oder nicht, verschiedenen Zwecken dienen können. So repräsentiert CHR$(137) die F2-Taste, wenn sie in einem String steht, und GOTO, wenn nicht.

Zeilen 15000–15160: Dieses Modul führt die Aufgabe aus, das Programm Zeile für Zeile von der Diskette zu lesen und es auszudrucken. Es ähnelt im Aufbau dem Programm PROG READ aus dem zweiten Abschnitt dieses Kapitels.

Zeilen 15030–15040: Eine Datei wird dem angegebenen Gerät unter dem angegebenen Programmnamen geöffnet. Eine zweite Datei wird dem Drucker geöffnet, um die Ausgabe zu empfangen.

Zeile 15050: Die ersten beiden Bytes der Datei, die die Startadresse des Programms im BASIC enthalten, werden gelesen und abgelegt.

Zeilen 15060–15080: Diese Zeilen sollen die "link bytes" am Anfang der Zeile aufnehmen und prüfen, ob es nicht die Enden der Dateikennzeichen sind. Die Variable QU wird gebraucht, um aufzuzeichnen, ob die aufgenommenen Zeichen in Anführungszeichen stehen – sie wird am Anfang der Zeile auf Null gesetzt.

Zeilen 15090–15100: Die zwei Bytes mit der Zeilennummer werden von der Diskette eingelesen und die Zeilennummer ausgedruckt.

Zeilen 15120–15140: Die Zeichen der Zeile werden einzeln von der Diskette eingelesen. Ist das aufgenommene Zeichen ein Anführungszeichen, so liegt der Wert QU zwischen eins und Null. Abhängig vom QU-Wert werden für andere Zeichen die Inhalte der einen oder anderen Seite einer Zeile im Feld C$ ausgedruckt. Die meisten Zeichen im Feld C$ sind normale Buchstaben oder Grafik-Zeichen, aber einige wurden auch, wie wir schon sahen, durch Schlüsselwörter oder Darstellungen in eckigen Klammern ersetzt, um den Zeichen, die als Schlüsselwort-Token oder Steuerzeichen gebraucht wurden, zu entsprechen.

Zeilen 5000–5140: Die schon vorher untersuchte Diskettenzustandsroutine.

DIE ANWENDUNG DES PROGRAMMS

Um das Programm zu benutzen, müssen Sie es zuerst (mit LOAD) in den Speicher laden. Vergewissern Sie sich, daß die Floppy Disk eingeschaltet ist und daß sie das Programm enthält, welches Sie ausdrucken wollen. Der Drucker muß auch eingeschaltet sein. Starten Sie das Programm, und geben Sie auf Befehl die Gerätenummer und den Programmnamen ein. Die Liste mit den Abkürzungen in eckigen Klammern, mit denen das Programm die Steuerzeichen ersetzt, finden Sie in den Anmerkungen über die Programmlistings am Anfang des Buches.

5. DAS ANEINANDERHÄNGEN VON PROGRAMMEN MIT HILFE VON PROGRAMM-DATEIEN (MERGE)

Die Möglichkeit, Programme aneinanderhängen zu können (englisch: to merge), d. h. sie beide gleichzeitig in den Speicher zu laden, so daß sie zu einem Programm werden, ist zwar sehr nützlich, aber leider nicht in den C 64 eingebaut. Das folgende Programm benutzt die Fähigkeit, Programm-Dateien sequentiell zu lesen, um zwei BASIC-Programm-Daten aneinanderzuhängen, so daß sie eine dritte bilden, welche aus den Zeilen beider Programme besteht. Gibt es in beiden Files Zeilen mit der gleichen Zeilennummer, werden die Zeilen in der zuerst angegebenen Datei von denen aus der zweiten überschrieben.

MERGE-PROGRAMM

```
10 GOTO 7000
5000 REM#*******************************
5010 REM DISK FEHLER-STATUS
5020 REM*******************************
5030 INPUT#15,EN,EM$,ET,ES
5040 IF EN=73 THEN 5030
5050 IF EN<20 THEN RETURN
5060 PRINT "■■■■■ **********************
****************"
5070 PRINT "■■                DISK FEHLER"
5080 PRINT "■■      FEHLER -" EN " " EM$
5090 PRINT "■      AUF SPUR -" ET " UND S
EKTOR -"ES
5100 PRINT "■■     PROGRAMMAUSFUEHRUNG UN
TERBROCHEN"
5110 PRINT "■■■ ************************
**************"
5120 CLOSE 15
5130 CLR
5140 END
7000 REM#*******************************
7010 REM ZWEI PROGRAMME ZUSAMMENFUEGEN
7020 REM*******************************
7030 INPUT "NAME VON PROGRAMM 1";N1$
```

```
7040 INPUT "NAME VON PROGRAMM 2";N2$
7050 INPUT "NAME DES NEUEN PROGRAMMS";N3
$
7060 INPUT "GERAETENUMMER ? 8";DEV :
OPEN 15,DEV,15
7070 PRINT "ZUSAMMENFUEGUNG VON " N1$ "
UND " N2$
7080 PRINT "ERGEBNISPROGRAMM IST " N3$
7090 GOSUB 8000
7100 CLOSE 15
7110 END
8000 REM*********************************
8010 REM ZUSAMMENFUEGEN
8020 REM*********************************
8030 L3$ = "" : L4$ = ""
8040 OPEN 3,DEV,3,N1$ : GOSUB 5000
8050 OPEN 4,DEV,4,N2$ : GOSUB 5000
8060 OPEN 5,DEV,5,N3$+",P,W" : GOSUB 500
0
8070 FI = 3
8080 T$ = ""
8090 GOSUB 10000: GOSUB 10000: PRINT#5,T
$ ;
8100 FI = 4 : GOSUB 10000: GOSUB 10000
8110 GOSUB 11000
8120 GOSUB 12000
8130 IF L3<=L4 THEN 8160
8140 IF LEN(L4$)>2 THEN PRINT#5,L4$;
8150 GOTO 8120
8160 IF LEN(L3$)>2 THEN PRINT#5,L3$;
8170 IF L3=L4 AND L3<65536 THEN 8110
8180 GOSUB 11000
8190 IF L3<>65536 OR L4<>65536 THEN 8130

8200 PRINT#5,CHR$(0) CHR$(0) ;
8210 CLOSE 3
8220 CLOSE 4
8230 CLOSE 5
8240 RETURN
9000 REM*********************************
9010 REM EINE ZEILE IN T$ HOLEN.
```

```
 9011 REM ZEILENNUMMER IN T
 9020 REM*********************************
 9030 T$ = ""
 9040 GOSUB 10000: GOSUB10000
 9050 IF T1$=CHR$(0) THEN 9100
 9060 GOSUB 10000: T = ASC(T1$)
 9070 GOSUB 10000: T = ASC(T1$)*256+T
 9080 GOSUB 10000
 9090 IF T1$<>CHR$(0) THEN 9080
 9100 RETURN
 10000 REM#********************************
 10010 REM EIN EINZELNES ZEICHEN LADEN
 10020 REM*********************************
 10030 GET#FI,T1$
 10040 T1$ = LEFT$(T1$+CHR$(0),1)
 10050 T$ = T$+T1$
 10060 RETURN
 11000 REM#********************************
 11010 REM EINE ZEILE AUS PROG. 3 LESEN
 11020 REM*********************************
 11030 IF LEN(L3$)=2 THEN L3 = 65536 : GO
TO 11080
 11040 FI = 3
 11050 GOSUB 9000
 11060 L3$ = T$
 11070 L3 = T
 11080 RETURN
 12000 REM#********************************
 12010 REM EINE ZEILE AUS PROG. 4 LESEN
 12020 REM*********************************
 12030 IF LEN(L4$)=2 THEN L4 = 65536 : GO
TO 12080
 12040 FI = 4
 12050 GOSUB 9000
 12060 L4$ = T$
 12070 L4 = T
 12080 RETURN
```

ERKLÄRUNG DES MERGE-PROGRAMMS

Zeilen 7000–7110: Dieses Modul erlaubt es dem Benutzer, die Namen der aneinanderzuhängenden Programme und den Namen der zu erzeugenden Programm-Datei anzugeben.

Zeilen 10000–10080: Dieses Modul, das andere Module innerhalb des Programms beliefert, verwendet GET#, um ein einzelnes Byte aus einer Datei zu lesen, dessen Nummer in der Variable F1 angegeben ist. Das im Byte enthaltene Zeichen wird vorübergehend in einen String T$ geschrieben, um von anderen Modulen gelesen werden zu können.

Zeilen 9000–9100: Dieses Modul gleicht genau den Modulen, die wir schon vorher in diesem Kapitel untersucht haben und die den Inhalt einer Programmzeile ausdrucken. Eine Erklärung der Methode finden Sie im zweiten Abschnitt dieses Kapitels.

Zeilen 11000–11080 und 12000–12080: Diese beiden Subroutinen speichern die von den vorigen Modulen gelesenen Zeilen zusammen mit den erreichten Zeilennummern in getrennten Variablen für die beiden Dateien. Ist die in der Variable L4$ oder L3$ belassene Zeilenlänge zwei, ist aufgrund einer früheren Untersuchung der relevanten Datei das Ende des Datei-Kennzeichens gefunden. In diesem Fall wird nicht weiter aus der Datei gelesen.

Zeilen 8000–8240: Das Steuermodul, das die Arbeit auf den Rest des Programms verteilt.

Zeilen 8040–8060: Die zwei Dateien, die miteinander verknüpft werden sollen, werden geöffnet. Die Ausgabe-Datei wird mit dem Suffix ",P,W" geöffnet, damit in sie hineingeschrieben werden kann.

Zeilen 8070–8090: Die beiden Bytes, die die Startadresse des Programms im Speicher enthalten, werden von der Datei 3 gelesen und in die neue Programm-Datei geschrieben. Deshalb wird das verknüpfte Programm dieselbe Startadresse haben wie das Programm 1.

Zeile 8100: Die Startadresse des Programm 2 wird gelesen und gelöscht.

Zeilen 8110–8180: Dieser Abschnitt nimmt Zeilen von den beiden Programm-Dateien auf und untersucht sie, um herauszufinden, welche Zeilennummer zuerst kommt. Diese Zeile wird dann in die Ausgabe-Datei geschrieben und eine weitere Zeile von der Datei genommen, welche die gerade geschriebene lieferte.

Haben die beiden Zeilen die gleiche Nummer, wird die Zeile des Programm 2 geschrieben und die entsprechende Zeile des Programm 1 gelöscht. In jedem Fall werden Zeilen nur geschrieben, wenn die Variablen L3$, L3, L4$ und L4 nicht anzeigen, daß in dem betreffenden Programm bereits das Dateiende erreicht wurde.

Zeilen 81900–8230: Wird das Dateiende beider Programm erreicht, werden zwei Nullen an das Ende der Ausgabe-Datei gehängt und alle Dateien geschlossen.

DIE ANWENDUNG DES MERGE-PROGRAMMS

Ist das Programm einmal im Speicher, muß die Diskette in der Floppy Disk ein, die *beide* miteinander zu verknüpfende Programme und genügend Freiraum enthält, um die Erstellung des verknüpften Programms zu ermöglichen. Nach dem Starten mit RUN wird der Benutzer aufgefordert, den Namen des ersten und zweiten miteinander zu verknüpfenden Programms zusammen mit dem Namen der Datei, die erstellt werden soll, einzugeben. Es sollte darauf geachtet werden, daß, falls zwei Zeilen die gleiche Nummer haben, die Zeilen der zuletzt genannten Datei die der ersten überschreiben.

6. DAS NEUNUMERIEREN EINER PROGRAMM-DATEI AUF DISKETTE (RENUMBER)

Ein Programm problemlos neu zu numerieren, ist ein oft gehegter Wunsch des Programmierers, der seine Programme attraktiv aufgemacht und einfach zu lesen wünscht. Das Programm in diesem Abschnitt ermöglicht es, eine Programm-Datei auf der Diskette neu zu numerieren, indem es sich der Möglichkeit bedient, von einer Datei in eine andere lesen zu können.

RENUMBER-PROGRAMM

```
1000 REM**********************************
1010 REM NEUNUMERIERUNG EINES PROGRAMMS
1011 REM AUF DISKETTE
1020 REM**********************************
1030 DEFFNHI(X) = INT(X/256) : DEFFNLO(X
) = X-FNHI(X)*256
1040 INPUT "PROGRAMMNAME ";NR$
```

```
1050 NW$ = "TEMPORARY"
1060 INPUT "GERAETENUMMER ? 8■■■■";DEV
1070 N$ = CHR$(0)
1080 INPUT "NEUNUMERIEREN AB ZEILE ? 10■■
";SL
1090 INPUT "SCHRITTWEITE ? 10■■■■■";SP
1100 LMAX = 2500 : DIM LN(LMAX,1)
1110 OPEN 15,DEV,15 : PRINT "■" : GOSUB
5000
1120 PRINT#15,"S0:TEMPORARY" : GOSUB 500
0
1130 GOSUB 2000
1140 GOSUB 3000
1150 PRINT#15,"S0:"+NR$ : GOSUB 5000
1160 PRINT#15,"R0:"+NR$+"=0:TEMPORARY"
1170 CLOSE 15
1180 GOSUB 4000
1190 END
2000 REM*******************************
2010 REM DURCHLAUFPHASE 1
2020 REM*******************************
2030 PTR = 0 : NL = SL
2040 OPEN 3,DEV,3,NR$ : GOSUB 5000
2050 GET#3,T$ : GET#3,T$
2060 GET#3,T$ : GET#3,T$
2070 IF T$="" THEN CLOSE 3 : RETURN
2080 GET#3,T$ : T = ASC(T$+CHR$(0))
2090 GET#3,T$ : T = ASC(T$+CHR$(0))*256+
T
2100 PRINT "■■■■■■■■■■■■BERECHNUNG DER NEU
EN ZEILENNR. FUER" T "          "
2110 GET#3,T$
2120 IF T$=CHR$(143) THEN GET#3,T$ : IF
T$= "#" THEN NL = INT(1+NL/1000)*1000
2130 IF T$<>"" THEN 2110
2140 LN(PTR,1) = NL
2150 LB(PTR,0) = T
2160 NL = NL+SP
2170 IF NL>63999 THEN T$ = "SCHRITTWEITE
 ZU GROSS" : GOTO 6000
```

```
2180 PTR = PTR+1
2190 IF PTR<=LMAX THEN 2060
2200 T$ = "ZU VIELE ZEILEN"
2210 GOTO 6000
2220 RETURN
3000 REM*********************************
3010 REM DURCHLAUFPHASE 2
3020 REM*********************************
3030 OPEN 3,DEV,3,NR$ : GOSUB 5000
3040 OPEN 4,DEV,4,NW$+",P,W" : GOSUB 500
0
3050 GET#3,T$:PRINT#4,LEFT$(T$+N$,1); :G
ET#3,T$:PRINT#4,LEFT$(T$+N$,1);
3060 FOR J = 0 TO PTR-1
3070 QU = 0
3080 GET#3,T$:PRINT#4,LEFT$(T$+N$,1); :G
ET#3,T$:PRINT#4,LEFT$(T$+N$,1);
3090 GET#3,T$
3100 GET#3,T$
3110 PRINT "[redacted] ZEILE "MID$(
STR$(LN(J,0))+"       ",2,6);
3120 PRINT " WIRD ZU ZEILE" LEFT$(STR$(A
BS(LN(J,1)))+"       ",6)
3130 T = ABS(LN(J,1))
3140 PRINT#4,CHR$(FNLO(T)) CHR$(FNHI(T))
 ;
3150 GET#3,T$
3160 IF T$=CHR$(34) THEN QU = NOT QU
3170 PRINT#4,LEFT$(T$+N$,1) ;
3180 IF(T$=CHR$(137)ORT$=CHR$(141)ORT$=C
HR$(167)ORT$=CHR$(138))ANDNOTQUTHEN3240
3190 IF T$<>"" THEN 3150
3200 NEXT J
3210 PRINT#4,N$ N$ ;
3220 CLOSE 3 : CLOSE 4
3230 RETURN
3240 GET#3,T$ : IF T$=" " THEN PRINT#4,T
$; : GOTO 3240
3250 IF T$<"0" OR T$>"9" THEN 3170
3260 T1$ = T$
3270 GET#3,T$
```

```
3280 IF T$=" " THEN 3270
3290 IF T$>="0" AND T$<="9" THEN T1$ = T
1$+T$ : GOTO 3270
3300 T = VAL(T1$)
3310 T1 = -1
3320 FOR I = 0 TO PTR-1
3330 IF LN(I,0)=T THEN T1 = I
3340 NEXT
3350 T1$ = "????"
3360 IF T1>0 THEN T1$ = MID$(STR$(LN(T1,
1)),2) : GOTO 3380
3370 LN(J,I) = -ABS(LN(J,I))
3380 PRINT#4,T1$ ;
3390 IF T$="," THEN PRINT#4,T$; : GOTO 3
240
3400 PRINT#4,LEFT$(T$+N$,1) ;
3410 GOTO 3190
4000 REM#################################
4010 REM UNDEFINIERTE ZEILENNUMMERN
4020 REM#################################
4030 PRINT "■■■UNDEFINIERTE ZEILENNUMMER
N IN DEN"
4040 PRINT "■    FOLGENDEN NEUNUMERIERTE
N ZEILEN:"
4050 UL = 0
4060 FOR I = 0 TO PTR-1
4070 IF LN(I,1)<0 THEN PRINT -LN(I,1) ,
: UL = UL+1
4080 NEXT
4090 IF UL=0 THEN PRINT SPC(17) "KEINE"
4100 PRINT "■■■■"
4110 RETURN
5000 REM#################################
5010 REM DISK FEHLERABFRAGE
5020 REM#################################
5030 INPUT#15,EN,EM$,ET,ES
5040 IF EN=73 THEN 5030
5050 IF EN<20 THEN RETURN
5060 PRINT "■■■■■■ **********************
***************"
```

```
5070 PRINT "■■■                DISK FEHLER"
5080 PRINT "■■■      FEHLER-" EN " " EM$
5090 PRINT "■■      AUF SPUR -" ET " UND S
EKTOR"ES
5100 PRINT "■■■  PROGRAMMAUSFUEHRUNG UNTE
RBROCHEN"
5110 PRINT "■■■■ ************************
**************"
5120 CLOSE 15
5130 CLR
5140 END
6000 REM#################################
6010 REM PROGRAMMFEHLER
6020 REM#################################
6030 PRINT "■■■■■ ***********************
*****************"
6040 PRINT "■■■■                PROGRAMMFEHL
ER"
6050 PRINT "■■■■      FEHLER - " T$
6060 GOTO 5100
```

ERKLÄRUNG DES RENUMBER-PROGRAMMS

Zeilen 1000—1190: Das Steuermodul, welches dem Benutzer ermöglicht, die neu zu numerierende Datei anzugeben und die Arbeit auf den Rest des Programms verteilt.

Zeile 1030: Die beiden Funktionen werden gebraucht, um den Dezimalwert einer Zeilennummer in einen Zwei-Byte-Wert für die Speicherung auf Diskette umzuwandeln.

Zeilen 1110—1120: Im Gegensatz zum vorigen Programm Merge löscht dieses die Ursprungsdatei, nachdem es sie neu numeriert hat. Während des Neunumerierens wird die Ausgabe-Datei "TEMPORARY" (zu deutsch: vorübergehend) genannt. Diese Zeilen öffnen den Fehlerkanal und sorgen dafür, daß alle Dateien mit demselben Namen gelöscht werden.

Zeilen 1150—1160: Diese Zeilen löschen die Ursprungsdatei, nachdem das Neunumerieren beendet ist und geben der Datei "TEMPORARY" den Namen der Ursprungsdatei.

Zeilen 2000–2220: Dieses Modul führt den sogenannten "first pass", den ersten Durchlauf, durch das Programm aus. Während dieses Vorgangs wird die ganze Datei zwar untersucht, es werden aber noch keine Veränderungen vorgenommen. Der Zweck des Durchlaufs 1 ist, die neuen Veränderungen zu berechnen, die am Ende eingesetzt werden sollen, und nach Fehlermöglichkeiten dafür zu suchen, daß das Programm abstürzt, während die neuen Zeilennummern in die Datei geschrieben werden.

Zeilen 2050–2100: Die bereits bekannten Zeilen, die die ersten beiden Bytes der Datei aufnehmen und sie löschen und dann, für jede Zeile einzeln, die "link bytes" und die Zeilennummer-Bytes aufnehmen. Der einzige Unterschied ist, daß die Zeilen zusätzlich den Benutzer darüber informieren, welche Zeile gerade bearbeitet wird.

Zeile 2110–2130: Die Zeile wird durchsucht, bis das das Zeilenende anzeigende Null-Byte entdeckt wird. In diesem Vorgang wird auch Ausschau nach dem CHR$(143), dem Token für REM, gehalten. Wird dies gefolgt von einem "#" gefunden, gilt es als Anzeichen dafür, daß die Zeilennummer der nächsten Zeile auf die nächsthöhere Tausenderstelle angehoben werden soll.

KAPITEL 9

RELATIVE DATEIEN

1. *Einleitung*
2. *Das Erstellen einer relativen Datei*
3. *Das Eröffnen einer relativen Datei*
4. *Positionsbestimmung in einer relativen Datei*
5. *Das Schreiben in eine relative Datei*
6. *Das Lesen aus einer relativen Datei*
7. *Das Schließen einer relativen Datei*
8. *Die Anwendung relativer Dateien*

1. EINLEITUNG

Bis jetzt haben uns ausschließlich mit Dateien beschäftigt, die sequentiell gelesen werden, das heißt ein Posten nach dem anderen, vom Beginn der Datei an. Obwohl die Geschwindigkeit der Floppy Disk die Zeit, die dadurch verschwendet wird, vermindern hilft, ändert dies jedoch nichts an der Tatsache, daß Zeit verschwendet wird. Die Anwendung der Dateien auf diese Art gleicht der Anwendung eines Feldes in BASIC, nur daß jedes Element vom Beginn an gelesen werden muß, um ein bestimmtes Element herausgreifen zu können.

Was wir brauchten, wäre ein Dateityp auf der Diskette, der sich genauso wie ein BASIC-Feld verhält, nämlich dem Benutzer erlaubt, eine Position festzulegen und von dieser direkt Informationen zu lesen. Ein solcher Dateityp existiert tatsächlich. Es ist die sogenannte "Relative Datei".

Daß es einen Dateityp "Relative Datei" gibt, stimmt nicht ganz. Die Arbeit mit relativen Dateien ist eigentlich eine Methode, zwei getrennte Dateien miteinander arbeiten zu lassen. Die erste der beiden Dateien ist die, in der die eigentlichen Daten gespeichert sind. Der Aufbau dieser Datei auf der Diskette ähnelt dem einer normalen sequentiellen Datei, obwohl sie nicht als solche im Directory der Diskette auftaucht. Zusätzlich zu dieser Hauptdatei gibt es jedoch noch eine zweite, die die Spur und den Sektor für jeden Datenposten, der in der Hauptdatei gespeichert ist, enthält.

Um nun auf eine relative Datei zugreifen zu können, wird die Nummer des Datenpostens in der Hauptdatei angegeben und die sekundäre "Zeiger"-Datei benutzt, um herauszufinden, wo auf der Diskette sich der Posten befindet, damit er direkt von dieser Position gelesen werden kann.

2. DAS ERSTELLEN EINER RELATIVEN DATEI

Im Gegensatz zur sequentiellen Datei, die nur einmal zum Schreiben geöffnet werden kann, läßt sich eine relative Datei immer wieder öffnen und ist nach jedem Öffnen bereit, sowohl von der Diskette zu lesen als auch auf sie zu schreiben. Dies ist möglich, weil im Fall der relativen Datei, die Datei *nicht* durch den normalen OPEN-Befehl erstellt wird. Um eine relative Datei zu erstellen, muß eine spezielle Form des OPEN-Befehls angewendet werden, dessen Format wie folgt lautet:

```
OPEN<FILENUMMER>,<GERÄTENUMMER>,<KANALNUMMER>,"
<FILENAME>,L," + CHR$(<RECLEN>)
```

Das einzig Neue gegenüber dem, was wir schon vorher einmal hatten, ist:

1. **L** – der Buchstabe L, das Datei-Kennzeichen für die relative Datei.

2. **RECLEN** – obwohl eine relative Datei mehr einem auf der Diskette gespeicherten Stringfeld ähnelt als einem im Speicher gespeicherten, unterscheidet es sich von einem BASIC-Stringfeld darin, daß jedes Element der Datei eine festgelegte Länge hat, die bestimmt werden muß, wenn die Datei erstellt wird. RECLEN (für record length, zu deutsch: Datensatzlänge) kann einen Wert zwischen 1 und 254 annehmen – die maximale Länge eines Datensatzes oder Records entspricht einem vollen Sektor auf der Diskette, d. h. 256 Bytes abzüglich der zwei "link bytes" für den Sektor (siehe Kapitel 1).
Beachten Sie, daß die "@0:"-Einrichtung zum Überschreiben einer bereits existierenden Datei bei der relativen Datei *nicht* funktioniert..

3. DAS ÖFFNEN EINER RELATIVEN DATEI

Vorausgesetzt, daß eine relative Datei vorher mit Hilfe der im vorigen Abschnitt beschriebenen Form des OPEN-Befehls erstellt wurde, *muß* sie danach mit dem OPEN-Befehl im folgenden Format geöffnet werden:

```
OPEN<FILENUMMER>,<GERÄTENUMMER>,<KANALNUMMER>,
<FILENAME>"
```

Beachten Sie, daß keine Notwendigkeit besteht, ein Kennzeichen für den Dateityp einzuschließen, da die Floppy Disk in der Lage ist, eine relative Datei zu erkennen, sobald sie einer begegnet.

4. POSITIONSBESTIMMUNG IN EINER RELATIVEN DATEI

Das Wichtigste bei der relativen Datei ist, daß es dem Benutzer ermöglicht wird, die Stelle in der Datei zu bestimmen, an die ein Posten geschrieben oder von der ein Posten gelesen werden soll. Bevor die Techniken zum Lesen oder Schreiben von einer bzw. in eine Datei untersucht werden können, ist es zuerst einmal nötig, die Anwendung des Positionsbefehls zu erklären, der es erlaubt, die Nummer des Datensatzes der Datei festzulegen.
Das Format des Positionsbefehls lautet wie folgt:

PRINT#< **FILENUMMER**>, **"P"** CHR$(<**CHAN**>) CHR$(<**LO**>)
CHR$(<**HI**>) CHR$(<**POS**>)

1. **FILENUMMER** – als erstes müssen Sie sich darüber im Klaren sein, daß der Positionierbefehl nicht etwas ist, was selbst in die relative Datei geschrieben wird, sondern ein an den *Fehlerkanal* gerichteter Befehl. Deshalb muß die eingegebene Filenummer die Nummer der Datei sein, die vorher (mit OPEN) in Form von OPEN FILENUMMER, GERÄTENUMMER, 15 für den Fehlerkanal geöffnet wurde.

2. **"P"** – dieses Kennzeichen zeigt dem Fehlerkanal an, daß der Positionierbefehl übermittelt wurde.

3. **CHAN** – die Nummer des Kanals (englisch CHANel), welcher der relatvien Datei zugeordnet wurde, als diese das letzte Mal geöffnet wurde.

4. **LO und HI** – die beiden Werte LO und HI geben die Datensatznummer im sogenannten Zwei-Byte-Format an. Der Wert dieser Kennzeichen kann für eine Zahl (X) auf die folgende Art berechnet werden:

HI = INT(X/256)

LO = X – 256*HI

5. **POS** – diese nicht unbedingt erfolgerliche Angabe kann an den Befehl angehängt werden, um den Zeiger in der Datei auf irgendein Zeichen innerhalb des durch LO und HI bestimmten Datensatzes zu setzen. Unter normalen Umständen werden die Records vom Anfang an gelesen und der Wert POS wäre dann 1. Der Positionierbefehl nimmt in jedem Fall den Wert 1 an, sollte die Eingabe CHR$ (POS) versäumt worden sein.
Beachten Sie: Wenn ein Positionierbefehl eine Datensatznummer enthält, die größer als die Position des gegenwärtig letzten Records ist, füllt die 1541 den

Leerraum auf der Diskette sofort mit Leer-Records auf, um die Anzahl der verfügbaren Datensätze an die Nummer, die für die neue Position angegeben wurde, anzugleichen. Dieser Vorgang kann einige Zeit beanspruchen. Damit so etwas nicht während der Dateieingabe passiert, ist es ratsam, die Datei bereits beim ersten Öffnen auf die maximal erforderliche Größe zu setzen. Dies geschieht, indem man z. B. die Position mit 1000 angibt, nachdem die Datei erstellt wurde. Darauf werden 1000 Leereinträge auf die Diskette geschrieben, wodurch Sie später, beim Zugriff auf Einträge zwischen 1 und 1000, weniger Zeit verlieren werden.

5. DAS SCHREIBEN IN EINE RELATIVE DATEI

Nachdem Sie die Floppy Disk über die Position des Eintrags, den Sie in die Datei schreiben wollen, informiert haben, ist es nun möglich, einen Datenposten in den Eintrag zu schreiben. Das Format des Print#-Befehls ist:

PRINT#<**FILENUMMER**>,**LISTE MIT VARIABLEN**

1. **FILENUMMER** – die Filenummer, unter der die relative Datei zuletzt geöffnet wurde.

2. **LISTE MIT VARIABLEN** – die zu schreibenden Daten, entweder numerisch oder als String, oder beides (siehe Interpunktion unten).

DIE INTERPUNKTION BEIM SCHREIBEN IN EINE RELATIVE DATEI

Werden Datenposten in eine relative Datei geschrieben, muß man auch auf die Interpunktion achten. Genau wie bei der sequentiellen Datei beeinflußt die Zeichensetzung beim Schreiben einer Variablenreihe in einen Record auch hier die Art, in der die Daten gespeichert werden (siehe Kapitel 7). Werden Datenposten durch Kommas getrennt in eine Datei geschrieben, bewirkt dies, daß in der Datei Leerzeichen zwischen die einzelnen Datenposten gesetzt werden. Man muß auf die Interpunktion achten, weil die Länge des Records, in den Daten geschrieben werden, begrenzt ist. Falls unvorsichtiges Zeichensetzen zu einer Überschreitung der Datensatzlänge führt, werden die Daten beschnitten und die Fehlermeldung OVERFLOW IN RECORD hervorgerufen, obwohl das Programm nicht stoppt.
Sogar wenn nur CHR$(13) gebraucht werden, um Datenposten zu trennen, muß die Anzahl der verwendeten Wagenrücklauf-Zeichen berücksichtigt werden, weil auch diese zur Länge der geschriebenen Daten beitragen.

Am meisten Platz wird gespart, wenn die Datenposten mit Hilfe des Semikolons getrennt werden oder gar kein Trennungszeichen benutzt wird. Dies aber verlangt, daß das Programm genau weiß, wie lang jeder Posten in einem Datensatz ist, wenn Daten aus der Datei gelesen werden, denn der Eintrag selbst enthält dann kein Kennzeichen für den Beginn oder das Ende eines Datenpostens.

6. DAS LESEN AUS EINER RELATIVEN DATEI

Unter der Voraussetzung, daß der Positionierbefehl angewendet worden ist, um den Datensatz, aus dem gelesen werden soll, zu bestimmen, kann entweder INPUT# oder GET# zum Lesen von Daten aus der Datei benutzt werden. Das Format für die beiden Befehle ist dasselbe wie für die sequentielle Datei. Die Form, in der Daten gelesen werden, hängt von den bei der Speicherung verwendeten Interpunktionszeichen ab. Wurden Wagenrücklauf-Zeichen zwischen jeden Posten des Datensatzes gesetzt, kann INPUT# benutzt werden, um die Datenposten einzeln einzulesen. Die Einschränkungen für die Anwendung von INPUT# und GET# sind dieselben wie die für die sequentielle Datei (siehe Kapitel 7).
Zusätzlich sollte noch einmal betont werden, daß jeder Datensatz eine festgelegte Länge hat und daß, sobald alle Posten eines Datensatzes gelesen wurden, eine Fehlermeldung RECORD OVERFLOW im Fehlerkanal erzeugt wird. Der Lesevorgang wird *nicht* fortgesetzt, solange nicht ein neuer Positionierbefehl eingegeben wurde.
Eine Zusatzmöglichkeit, die sich beim Lesen von einer relativen Datei mit INPUT# und GET# bietet, ist, über den Positionierbefehl das *Zeichen* festzulegen, von dem aus das Lesen oder Schreiben stattfinden soll. Würde also ein 100-Zeichen-Record 10 Posten (oder Felder) enthalten, wobei jeder 9 Zeichen lang und durch ein Wagenrücklauf-Zeichen begrenzt ist, so wäre es möglich, daß die Zeichenposition korrekt mit einem Positionierbefehl übermittelt wurde.

7. DAS SCHLIESSEN EINER RELATIVEN DATEI

Das Format des CLOSE-Befehls in Beziehung auf die relativen Dateien unterscheidet sich von dem der sequentiellen Dateien nicht:

CLOSE<FILENUMMER>

8. DIE ANWENDUNG RELATIVER DATEIEN

Bis jetzt haben wir die für die Handhabung der relativen Dateien notwendigen Befehle besprochen. Wir wollen uns nun einem praktischen Beispiel für die Anwendung relativer Datein in Form eines einfachen Daten-Bank-Programms zuwenden, welches "Diskbase" (zu deutsch etwa Diskettenbank) heißt.

LISTING DES PROGRAMMS DATENBANK

```
10 REM DATENBANK
20 DEV = 8
30 DIM F$(10),F%(10),DA$(10)
40 DEF FNHI(X) = INT(X/256)
50 DEF FNLO(X) = X-INT(X/256)*256
60 PAD$ = "                    ": PAD$ = PAD
$+PAD$
70 PAD$ = PAD$+PAD$ : PAD$ = PAD$+PAD$ :
 PAD$ = LEFT$(PAD$,127)+PAD$
80 GOSUB 9000
90 GOSUB 10000
100 GOTO 80
1000 REM##############################
1010 REM AENDERN DER GERAETENUMMER
1020 REM##############################
1030 PRINT "■       NEUE GERAETENUMMER ?"
 DEV "■■■■■";
1040 IF DEV>9 THEN PRINT "■" ;
1050 INPUT T
1060 IF T<4 OR T>31 THEN PRINT "□□" : GO
TO 1000
1070 DEV = T
1080 RETURN
2000 REM##############################
2010 REM DATEI INITIALISIEREN
2020 REM##############################
2030 INPUT "■■DATEINAME ";NA$
2040 IF LEN(NA$)>12 OR LEN(NA$)<1 THEN 2
030
2050 INPUT "■ANZAHL FELDER (1-10) ";NF
```

```
2060 IF NF<1 OR NF>10 THEN PRINT "□□□" :
 GOTO 2050
2070 FS = 0
2080 FOR I = 1 TO NF
2090 PRINT "■NAME VON FELD" I ;
2100 INPUT F$(I-1)
2110 F$(I-1) = LEFT$(F$(I-1),30)
2120 PRINT "LAENGE VON FELD" I;
2130 INPUT F%(I-1)
2140 FS = FS+F%(I-1)
2150 NEXT
2160 IF FS>254 THEN PRINT "ZU VIELE FELD
ER ODER FELDER ZU LANG" : GOTO 2050
2170 PRINT "■DATEINAME - " NA$
2180 FOR I = 0 TO NF-1
2190 FOR I = 0 TO NF-1
2200 PRINT "■" F$(I) SPC(30-LEN(F$(I)))
F%(I)
2210 NEXT
2220 INPUT "■ALLE EINGABEN RICHTIG (J/N)
 ";T$
2230 IF T$="N" THEN 2000
2240 IF T$<>"J" THEN PRINT "□□□□□" : GOT
O 2220
2250 IT = 0
2260 PRINT#15,"S0:"+NA$+".D"
2270 GOSUB 4000
2280 OPEN8,DEV,8,NA$+".D,L,"+CHR$(FS)
2290 PRINT#15,"P" CHR$(8) CHR$(100) CHR$
(0) CHR$(1)
2300 INPUT#15,A,B$,C,D
2310 CLOSE 8
2320 RETURN
3000 REM#################################
3010 REM LADEN DER USER-DATEI
3020 REM#################################
3030 INPUT "■          DATEINAME ";NA$
3040 IF LEN(NA$)>12 OR LEN(NA$)<1 THEN 3
030
3050 OPEN 8,DEV,8,NA$+".C,U"
3060 FS = 0
```

```
3070 INPUT#15,EN,EM$,ET,ES
3080 IF EN<19 THEN 3120
3090 PRINT "■       DISK FEHLER " EM$
3100 FOR I = 0 TO 2000 : NEXT : CLOSE 8
3110 GOTO 3170
3120 INPUT#8,IT,NF
3130 FOR I = 0 TO NF-1
3140 INPUT#8,F$(I),F%(I)
3150 FS = FS+F%(I)
3160 NEXT
3170 CLOSE 8
3180 RETURN
4000 REM#################################
4010 REM SPEICHERN DER USER-DATEI
4020 REM#################################
4030 CLOSE 8
4040 OPEN 8,DEV,8,"@0:"+NA$+".C,U,W"
4050 PRINT#8,IT
4060 PRINT#8,NF
4070 FOR I = 0 TO NF-1
4080 PRINT#8,F$(I)
4090 PRINT#8,F%(I)
4100 NEXT
4110 CLOSE 8
4120 RETURN
5000 REM#################################
5010 REM LADEN DER DATEN
5020 REM#################################
5030 PRINT#15,"P" CHR$(8) CHR$(FNLO(T+1)
) CHR$(FNHI(T+1)) CHR$(1)
5040 FOR I = 0 TO NF-1
5050 DA$(I) = ""
5060 FOR J = 1 TO F%(I)
5070 GET#8,T$
5080 T1 = ASC(T$) AND 127
5090 IF T1<31 THEN T$ = ""
5100 DA$(I) = DA$(I)+LEFT$(T$+CHR$(0),1)

5110 NEXT J,I
5120 RETURN
```

```
6000 REM********************************
6010 REM SPEICHERN DER DATEN
6020 REM********************************
6030 PRINT#15,"P" CHR$(8) CHR$(FNLO(T+1)
) CHR$(FNHI(T+1)) CHR$(1)
6040 T$ = ""
6050 FOR I = 0 TO NF-1
6060 T$ = T$+LEFT$(DA$(I)+LEFT$(PAD$,F%(
I)-LEN(DA$(I))),F%(I))
6070 NEXT I
6080 PRINT#8,T$ ;
6090 RETURN
7000 REM********************************
7010 REM AENDERN DER DATEN
7020 REM********************************
7030 PRINT "■" IT " FELDER IN DER DATEI
[REVERS EIN]" NA$ ""
7040 INPUT "NUMMER DES DATENSATZES ? #■■■
";T$
7050 T = VAL(T$) : IF T$="#" THEN T = IT

7060 IF T<0 OR T>IT THEN PRINT "□□" : GO
TO 7040
7070 PRINT
7080 FOR I = 0 TO NF : DA$(I) = "" : NEX
T
7090 IF IT<>T THEN GOSUB 5000
7100 FOR I = 0 TO NF-1
7110 PRINT F$(I) " ? " DA$(I)
7120 PRINT "□" F$(I) " " ;
7130 INPUT T$
7140 IF LEN(T$)>F%(I) THEN PRINT "□□" :
GOTO 7110
7150 DA$(I) = T$
7160 NEXT I
7170 PRINT "■■HINZUFUEGEN VON DATEN"
7180 GOSUB 6000
7190 IF T=IT THEN IT = IT+1
7200 INPUT "■■WEITERE FELDER (J/N) ? J■■■
";T$
```

```
7210 IF T$="J" THEN 7000
7220 IF T$<>"N" THEN PRINT "□□□□" : GOTO
 7200
7230 RETURN
8000 REM#################################
8010 REM AUFLISTEN DER DATEN
8020 REM#################################
8030 LN = 0
8040 PRINT "□" IT "FELDER IN DER DATEI [
REVERS EIN]" NA$ ""
8050 PRINT "NUMMER DES DATENSATZES ?" LN

8060 IFIT=0THENPRINT "■■■■■ KEINE FELDER
 DEFINIERT" :FOR I=0 TO 2000:NEXT:RETURN

8070 INPUT "□NUMMER DES DATENSATZES ";T
8080 IF T<0 OR T>=IT THEN 8040
8090 PRINT
8100 FOR I = 0 TO NF : DA$(I) = "" : NEX
T
8110 GOSUB 5000
8120 FOR I = 0 TO NF-1
8130 PRINT F$(I) " = " DA$(I)
8140 NEXT I
8150 LN = LN+1 : IF LN>=IT THEN LN=0
8160 INPUT "■■WEITERE FELDER (J/N) ? J■■■
";T$
8170 IF T$="J" THEN 8040
8180 IF T$<>"N" THEN PRINT "□□□□" : GOTO
 8160
8190 RETURN
9000 REM#################################
9010 REM EROEFFNUNGSMENUE
9020 REM#################################
9030 OPEN 15,DEV,15
9040 PRINT "□"
9050 PRINT SPC(12) "1541 DATENBANK"
9060 PRINT SPC(11) "■EROEFFNUNGSMENUE
9070 PRINT SPC(8) "■■■■■1) NEUE DATEI ERS
TELLEN"
```

```
9080 PRINT SPC(8) "■2) BESTEHENDE DATEI
OEFFNEN"
9090 PRINT SPC(8) "■3) AENDERN DER GERAE
TENUMMER"
9100 PRINT SPC(8) "■4) ZURUECK ZU BASIC■
"
9110 INPUT "       BEFEHL (1-4) ? 2■■■■";T
$
9120 CO = VAL(T$)
9130 IF CO<1 OR CO>4 THEN PRINT "□□" : G
OTO 9110
9140 ON CO GOSUB 2000,3000,1000,9180
9150 CLOSE 15
9160 IF CO=3 OR EN>19 THEN 9000
9170 RETURN
9180 PRINT "■"
9190 CLOSE 15
9200 END
10000 REM#############################
10010 REM HAUPTMENUE
10020 REM#############################
10030 OPEN 15,DEV,15
10040 OPEN 8,DEV,8,NA$+",D"
10050 PRINT "■"
10060 PRINT SPC(12) "1541 DATENBANK"
10070 PRINT SPC(14) "HAUPTMENUE"
10080 PRINT SPC(8) "■■■■■1) DATEN HINZUFU
EGEN/EDITIEREN"
10090 PRINT SPC(8) "■2) DATEN AUFLISTEN"

10100 PRINT SPC(8) "■3) ZURUECK ZUM EROE
FFNUNGSMENUE"
10110 INPUT "       BEFEHL (1-3) ? 2■■■■";
T$
10120 T = VAL(T$)
10130 IF T<1 OR T>3 THEN PRINT "□□" : GO
TO 10110
10140 IF T=3 THEN 10180
10150 ON T GOSUB 7000,8000
10160 CLOSE 8
```

```
10170 GOTO 10040
10180 CLOSE 8
10190 CLOSE 15
10200 RETURN
```

ERKLÄRUNG DES PROGRAMMS DATENBANK

Zeilen 9000–9200: Das eröffnende Menü des Programms.

Zeilen 10000–10200: Das Hauptmenü wird aufgerufen, nachdem der Benutzer eine Datei eröffnet hat.

Zeilen 2000–2320: Dieser Abschnitt erlaubt dem Benutzer das Aussehen der zu erstellenden Datei zu bestimmen, d. h. die Anzahl der Felder, die jeder Datensatz enthält, deren Namen und ihre Länge. In Bezug auf die relativen Dateien selbst, sind der einzige interessante Teil die Zeilen 2260–2320. Diese Zeilen löschen jede bereits mit diesem Namen existierende Datei, öffnen die relative Datei in Zeile 2280 und erteilen den Positionierbefehl für Datensatz 100, d. h. einhundert Leer-Datensätze werden auf die Diskette geschrieben.

Zeillen 4000–4120: Eine zweite Datei, dieses Mal eine User-Datei, wird damit beauftragt, die Details des Feldnamens, die Größe usw. zu speichern. Wenn später einmal auf die schon erstellte Hauptdatei zugegriffen wird, wird das Programm zu allererst die Details der Felder aus der User-Datei lesen, so daß es den Feldern innerhalb des Records die richtigen Namen und Längen zuordnen kann.

Zeilen 3000–3180: Wie schon im vorigen Abschnitt erwähnt, liest das Programm, wenn es auf eine existierende Datei zugreift, die Feldnamen und -größen aus einer User-Datei, bevor es beginnt, Daten abzurufen. Dieser Programmabschnitt wird zuerst aufgerufen, wenn der Benutzer bestimmt hat, eine bereits existierende Datei zu lesen. Der Dateiname wird vom Benutzer eingegeben und die Informationen von der User-Datei, in den Zeilen 3120–3170, werden gelesen. Die User-Datei selbst wird denselben Namen tragen wie die Daten-Files, außer daß deren Name mit einem ".c" für "control" endet, anstelle des ".d" für "data". So wird also, wenn der Benutzer den Filenamen "DATASTORE" eingibt, die Kontrolldatei DATASTORE.C heißen und die Daten-File DATASTORE.D.

Zeilen 5000—5120: Dieser Programmabschnitt liest einen einzelnen Record aus der Daten-File. Die Nummer des zu lesenden Datensatzes steht an einer anderen Stelle im Programm und ist in der Variablen T enthalten. Die beiden Funktionen FNLO und FNHI, die ganz am Anfang des Programms aufgestellt wurden, berechnen das LOW- und das High-Byte der die Nummer des Datensatzes repräsentierenden Zwei-Byte-Zahl. Die Zeilen von 5060 und 5110 lesen die Datenposten mit Hilfe des GET# einzeln aus dem Record. Während des Lesens wird überprüft, ob nicht Steuer- oder Leerzeichen auf die Diskette das laufende Programm unterbrechen. Jeder aus den Record gelesene Datenposten wird in dem Feld DA$ gespeichert.

Zeilen 8000—8190: Diese Zeilen erlauben dem Benutzer, die Nummer eines aufzurufenden Datensatzes anzugeben, und rufen dann das Modul in Zeile 5000 auf, welches die Daten liest.

Zeilen 6000—6090: Diese Zeilen erteilen den Positionierbefehl an einen vom Benutzer bestimmten Record und schreiben dann die vom Modul in Zeile 7010 eingegebenen Datenposten in diesen hinein.

Zeilen 7000—7230: Dieser Abschnitt ermöglicht dem Benutzer, eine Recordnummer anzugeben. Die im Moment an diesem Punkt gespeicherten Daten werden zuerst gelesen, worauf dem Benutzer freigestellt wird, die existierenden Datenposten zurückzuschreiben oder sie zu ändern. Die schließlich vom Benutzer eingegebenen Datenposten werden mit Hilfe des Moduls in Zeile 6000 in die Datei geschrieben.

Das Programm ist nicht sonderlich anspruchsvoll, aber es stellt trotzdem ein brauchbares Beispiel dar, das deutlich macht, was mit einer relativen Datei, besonders in Bezug auf die Geschwindigkeit, mit der Daten gelesen oder geschrieben werden können, erreicht werden kann. Es liefert eine gute Basis für weitere Entwicklungen solcher Dateien.

KAPITEL 10

DIREKTZUGRIFFS-DATEI

1. *Das Öffnen einer Direktzugriffs-Datei mit OPEN*
2. *Die Lage von Daten im Puffer*
3. *Das Schreiben von Daten in den Puffer*
4. *Das Schreiben von Daten auf die Diskette*
5. *Das Laden von Daten von der Diskette in den Puffer*
6. *Das Zurückholen von Daten in den C 64*
7. *Das Belegen und Freisetzen von Sektoren auf der Diskette*
8. *Das Ausführen von Maschinensprache von der Diskette*
9. *Zwei Dienstleistungsprogramme für die Direktzugriffs-Dateien*

Bis jetzt haben wir uns nur mit Dateien beschäftigt, bei denen die Hauptarbeit beim Lesen und Schreiben vom hochentwickelten und umfangreichen DOS-Programmen erledigt wurde. Wir haben die zu speichernden Daten und manchmal sogar die Stelle, die sie in einer Datei einnehmen sollten, eingegeben, aber es war immer das DOS, das den Diskettenplatz verwaltet hat. Es suchte den besten Speicherplatz aus und zeichnete das komplizierte System der Sektoren auf, aus denen eine Datei in der BAM "Block Allocations Map" besteht. Es gibt jedoch einen Dateityp, der es erlaubt, wenigstens einige DOS-Funktionen zu umgehen und die Art und die Stelle, an der Daten auf der Diskette gespeichert werden sollen, selbst zu bestimmen. Es sind die sogenannten "Direktzugriffs-Dateien" (engl. random files).
Es gibt keinen besonderen Grund dafür, daß die Direktzugriffs-Dateien nicht zu den auf der 1541 verfügbaren Einrichtungen gehört. Dies liegt daran, daß es der Grundtyp einer Datei ist, mit deren Hilfe das DOS die Dateien erstellt, die wir bis jetzt untersucht haben. Als Datei sind die Direktzugriffs-Dateien sehr wahrscheinlich am wenigsten brauchbar, aber gerade weil sie so einfach in ihrem Aufbau und ihrer Verwaltung sind, können sie oft dann eingesetzt werden, wenn der Benutzer in unerwarteter Weise auf die Diskette zugreifen muß.
Sie können Direktzugriffs-Dateien gebrauchen, um jeden anderen Dateityp zu simulieren, den wir bis jetzt betrachtet haben. Jedoch würde das Endergebnis solcher Anstrengungen kaum das erreichen, was Commodore bereits in das DOS eingebaut hat, also wäre auch der erhebliche Programmieraufwand kaum der Mühe wert. In diesem Kapitel untersuchen wir lediglich die Grundbegriffe für den Umgang mit Direktzugriffs-Dateien, wollen jedoch die Befehle, mit denen solche Dateien

beeinflußt werden, anhand von ein oder zwei (unüblichen) Anwendungsbeispielen darstellen.

1. DAS ÖFFNEN EINER DIREKTZUGRIFFS-DATEI

Direktzugriffs-Dateien werden angewendet für den Datenaustausch und die Erteilung von Befehlen. Zum Beispiel werden Daten, die in eine Direktzugriffs-Datei geschickt werden, nicht automatisch auf die Diskette geschrieben. Dies geschieht erst, wenn ein eigener Befehl über den Fehlerkanal gesendet wird, der angibt, wo und wie die Datei gespeichert werden soll. Wird also eine Direktzugriffs-Datei (mit OPEN) geöffnet, wird keine direkte Leitung vom C 64 zur Diskette selbst geschaltet, sondern eine Leitung in einen Speicherbereich des Laufwerkes, den sogenannten Puffer, der bis zu 256 Bytes an Daten aufnehmen kann. Weil die Floppy Disk mehrere Bereiche für diesen Zweck bereitstellt, muß der OPEN-Befehl für eine Direktzugriffs-Datei nicht nur die Filenummer, die Geräte- und die Kanalnummer enthalten, sondern auch die Puffernummer.
Das Format des OPEN-Befehls für die Direktzugriffs-Datei ist:

OPEN<FILENUMMER>,<GERÄTENUMMER>,<KANALNUMMER>1,"# [<**PUFFER**>]

1. # – das "#"-Zeichen steht hier für den Filenamen und ist eine Anzeige für das Laufwerk, daß die zu öffnende Datei eine Direktzugriffs-Datei ist.

2. **PUFFER** – der Wert PUFFER kann eingesetzt werden, um anzugeben, welcher der laufwerks-eigenen Puffer der Filenummer zugeordnet werden soll. Die Nummern der Puffer sind die Zahlen 0 bis 5. Jedoch werden von diesen mindestens zwei schon von der Floppy Disk beansprucht, und dies sogar, bevor Sie Dateien öffnen, die noch weitere Puffer in Beschlag nehmen. Weil es sehr unsicher ist, die verfügbaren Puffer zu bestimmen, wird der nicht unbedingt notwendige PUFFER-Wert kaum gebraucht. Wird er weggelassen, ordnet das Laufwerk der Datei einen freien Puffer zu (falls einer zur Verfügung steht). Für das Programmieren ist es nicht nötig, welcher Puffer benutzt wird, es sein denn, es muß ein bestimmter Puffer verwendet werden, um in Maschinensprache Programme in den DOS-RAM hineinzuschreiben.

2. DIE LAGE VON DATEN IM PUFFER

Nachdem der Datei ein Puffer zugeordnet wurde, ist die nächste Aufgabe (je nachdem, ob Daten vom C 64 eingelesen werden sollen), das Laufwerk zu informieren, an welcher Stelle im Puffer ein spezieller Zeiger, nämlich der sogenannte Puffer-Zeiger, gesetzt werden soll. Nachfolgende Anweisungen, vom Puffer zu lesen oder in ihn hineinzuschreiben, werden dann von der Stelle an ausgeführt, die vom Puffer-Zeiger angezeigt wurde, und nicht unbedingt vom Anfang an.
Das Format für den Puffer-Zeiger-Befehl ist:

PRINT#<**FILENUMMER**>,"**B-P:**"<**KANALNUMMER**>,<**CHAR**>

1. **FILENUMMER** – wie alle Direktzugriffs-Befehle, ist dies eine Anweisung, die über den Fehlerkanal geschickt wird: Also ist die Filenummer die einer vorher dem Kanal 15 geöffneten Datei.

2. **B-P:** – die Abkürzung des Befehls "BUFFER-POINTER:" – das vollständige Format wird zwar akzeptiert, ist aber etwas umständlich.

3. **KANALNUMMER** – die angegebene KANALNUMMER wird die sein, die vorher in einem OPEN-Befehl der Direktzugriffs-Datei zugeordnet wurde, mit der nun gearbeitet werden soll.

4. **CHAR** – der Wert des Bytes innerhalb des Puffers, von dem aus der nachfolgende Schreib- bzw. Lesevorgang stattfinden soll; der Wert liegt zwischen 0 und 255.

3. DAS SCHREIBEN VON DATEN IN DEN PUFFER

Zum Schreiben von Daten in den Puffer wird der normale PRINT#-Befehl verwendet, dessen Format

PRINT#<FILENUMMER>,LISTE MIT VARIABLEN

ist, wobei die FILENUMMER die Nummer ist, die der Direktzugriffs-Datei zugeordnet worden ist, als sie geöffnet wurde.
Wie bei den relativen Dateien, ist bei den Direktzugriffs-Dateien der Puffer auf genau 256 Bytes (0–255) begrenzt, so daß die überschüssigen Daten verloren gehen, wenn mehr Daten in den Puffer geschrieben werden, als dieser aufnehmen kann.

4. DAS SCHREIBEN VON DATEN AUF DIE DISKETTE

In den Puffer geschriebene Daten gehen beim Ausschalten des Laufwerkes verloren, wenn nichts weiter mit ihnen getan wird. Demnach gibt es zwei Befehle, die benutzt werden können, um den Inhalt eines Puffers auf die Diskette zu schreiben. Der erste Befehl, "BLOCK-WRITE", wird angewendet, um den Teil des Puffer-Inhaltes zwischen 0 und der Stelle des Puffer-Zeigers auf die Diskette zu schreiben. (*Anmerkung:* Obwohl nur ein Teil des *Puffers* geschrieben wird, wird der ganze Sektor, in den die Daten plaziert werden, geändert.) Der zweite Befehl heißt "U2:" und wird angewendet, um den *vollständigen* Puffer-Inhalt in eine bestimmte Spur und einen bestimmten Sektor zu schreiben. Da es selten einen Grund gibt, nur einen Teil des Puffers auf die Diskette zu schreiben, ist "U2:" der öfter angewandte Befehl bei Direktzugriffs-Dateien.

PRINT#<**FILENUMMER**>,**"B-W:"**<**KANALNUMMER**>,<**GERÄTE-NUMMER**>,<**SPUR**>,<**SEKTOR**>

oder

PRINT#<FILENUMMER>,**"U2:"**<KANALNUMMER>,<GERÄTE-NUMMER>,<SPUR>,<SEKTOR>

1. **FILENUMMER** – die Nummer der vorher dem Fehlerkanal geöffneten Datei.

2. **"B-W:"** – die vollständige Form "BLOCK-WRITE:" kann auch eingegebenen werden.

3. **KANALNUMMER** – die der Direktzugriffs-Datei im OPEN-Befehl zugeordnete Datei des Kanals.

4. **GERÄTENUMMER** – immer null, wenn die 1541 verwendet wird.

5. **ILLEGAL TRACK AND SECTOR** Fehlermeldung – wird im Fehlerkanal hervorgerufen wenn ein nicht-existierender Teil der Diskette angegeben wird.

6. **"U2:"** – eine alternative Form, "UB:", kann auch eingegeben werden.

Anmerkung: Da beim Einsatz von Direktzugriffs-Dateien die Schutzvorrichtungen des DOS umgangen werden, gibt es bei diesen Befehlen keinen Schutz gegen das Überschreiben von wichtigen Daten, einschließlich der im Directory.

5. DAS LADEN VON DATEN VON DER DISKETTE IN DEN PUFFER

Analog zu den beiden Befehlen im vorigen Abschnitt gibt es zwei Befehle, "B-R:" und "U1:", die es erlauben, eine Spur und einen Sektor anzugeben, und diese zum Lesen von der Diskette *in* den Puffer zu laden. Das Format der beiden Befehle ist jeweils ein Spiegelbild des Formats der beiden Schreibbefehle,d. h.:

PRINT#<FILENUMMER>,**"B-R:"**<KANALNUMMER>,<GERÄTE-NUMMER>,<SPUR>,<SEKTOR>

oder

PRINT#<FILENUMMER>,**"U1:"**<KANALNUMMER>,<GERÄTE-NUMMER>,<SPUR>,<SEKTOR>

2. **"U1:"** – die alternative Form "UA:" kann auch eingegeben werden.

6. DAS ZURÜCKHOLEN VON DATEN IN DEN C 64

Sind die erwünschten Daten erst einmal von der Diskette gelesen und im Puffer plaziert, ist es sehr einfach, sie mit Hilfe der Befehle INPUT# und GET# zurück in den C 64 zu laden. Der Puffer-Zeiger-Befehl kann benutzt werden, um zu bestimmen, wo im Puffer GET# oder INPUT# mit dem Lesen anfangen sollen. Sonst gibt es keinen Unterschied zu den Vorgängen bei anderen Dateitypen.
Das Format für INPUT# und GET# ist:

INPUT#<FILENUMMER>,LISTE MIT VARIABLEN

oder

GET#<FILENUMMER>,LISTE MIT VARIABLEN

wobei die FILENUMMER die der Direktzugriffs-Datei zugeordnete Zahl ist.

7. DAS BELEGEN UND FREISETZEN VON SEKTOREN AUF DER DISKETTE

Ein Problem muß noch gelöst werden. Wenn Daten in Form einer Direktzugriffs-Datei auf die Diskette geschrieben werden, wird die Belegung der Sektoren für die

Zwecke der Datei nicht in der BAM aufgezeichnet. Es besteht daher die Gefahr, daß das DOS, welches sich bei der Arbeit auf die BAM stützt, die der Direktzugriffs-Datei zugeordneten Sektoren einfach überschreibt. Die Lösung dieses Problems liegt im "BLOCK-ALLOCATE:"-Befehl (und dessen Gegenstück "BLOCK-FREE:").
Die Aufgabe des "B-A:" ist, in die BAM einzutragen, daß ein bestimmter Sektor belegt ist. Ist dies einmal geschehen, wird das DOS diesem Sektor nicht mehr mit anderem Material überschreiben. "B-F:" macht das Gegenteil: Es trägt in die BAM ein, daß ein vorher als belegt gekennzeichneter Sektor nun freigesetzt ist. Das Format der beiden Befehle ist:

```
PRINT#<FILENUMMER>,"B-A:"<GERÄTENUMMER>,<SPUR>,<SEK-
TOR>
```

und

```
PRINT#<FILENUMMER>,"B-F:"<GERÄTENUMMER>,<SPUR>,<SEK-
TOR>
```

Eigentlich wird "BLOCK-ALLOCATE:" normalerweise angewendet, *bevor* ein Sektor beschrieben wird. Schreibt man den Befehl in den Fehlerkanal, ruft dies die Fehlermeldung NO BLOCK hervor, falls der Sektor schon einer Datei zugeordnet war.
Zusammen mit dem Fehlermeldungstext geben die Spur- und Sektor-Zahlen den nächst höheren *freien* Sektor an – in Spuren und Sektoren, deren Zahl niedriger als die von Ihnen angegebene ist, wird nicht nach freiem Platz gesucht. Daher beginnen Direktzugriffs-Dateien meist in Spur 1, Sektor 0, so daß sie die vom "BLOCK-ALLOCATE:"-Befehl gebotenen Leistungen voll ausnutzen können.

8. DAS AUSFÜHREN VON MASCHINENSPRACHE VON DER DISKETTE

Ein weiterer Block-Befehl bleibt noch übrig, obwohl er selten angewendet wird. Es ist der "BLOCK-EXECUTE:"-Befehl, und er hat das Format:

```
PRINT#<FILENUMMER>,"B-E:"<GERÄTENUMMER>,<SPUR>,
<SEKTOR>
```

Er bewirkt, daß der angegebene Sektor in den Puffer geladen und dann innerhalb des Laufwerkes als Maschinensprache-Programm gestartet wird, d. h. das Pro-

gramm beeinflußt den 6502-Chip, der die 1541 steuert und nicht den 6510-Chip, die CPU des C 64. Die Floppy Disk in dieser Weise mit einem Maschinensprache-Programm zu betreiben, sollte nur mit größter Vorsicht geschehen; denn das Ergebnis falscher Laufwerksteuerung können den Verlust von Daten oder sogar die Beschädigung des Mechanismus selbst sein.

9. ZWEI DIENSTLEISTUNGSPROGRAMME FÜR DIE DIREKT-ZUGRIFFS-DATEIEN

In diesem Abschnitt wollen wir zwei interessante Programme präsentieren, die sich die Leistungsfähigkeit der Direktzugriffs-Dateien zunutze machen. Das erste der beiden Programm heißt UNSCRATCH, und seine Aufgabe ist, die Auswirkungen des SCRATCH-Befehls rückgängig zu machen. Das Sektor-Programm LIST TRACK AND SECTOR leistet den Inhalt einer Diskette auf den Bildschirm auf und zeigt dabei die Belegung aller Sektoren Spur für Spur an.

```
10 GOTO 27000
27000 REM**********************************
27010 REM WIEDERHERSTELLUNG EINES
27015 REM GELOESCHTEN FILES (UNSCRATCH)
27020 REM**********************************
27030 INPUT "GERAETENUMMER ? 8";DEV
27040 OPEN 8,DEV,8,"#"
27050 OPEN 15,DEV,15
27060 N$ = CHR$(0)
27070 S$ = CHR$(160)+CHR$(160)
27080 S$ = S$+S$ : S$ = S$+S$
27090 S$ = S$+S$
27100 INPUT "NAME DES GELOESCHTEN FILES
";NA$
27110 INPUT "FILETYP ";TY$
27120 GOSUB 30000
27130 IF CO THEN 27150
27140 PRINT "FILE " NA$ " KANN NICHT"
27145 PRINT "ZURUECKGEHOLT WERDEN" : GOT
O 27180
27150 IF T>128 THEN 27180
27160 PRINT "FILE KANN ZURUECKGEHOLT WER
DEN, ABER"
```

```
27170 PRINT "FILETYP " TY$ " IST NICHT B
EKANNT"
27180 PRINT#15,"V0"
27190 CLOSE 8 : CLOSE 15
27200 END
28000 REM*******************************
28010 REM NACH DEM GELOESCHTEN FILE
28015 REM SUCHEN
28020 REM*******************************
28030 CO = 0
28040 NT = 18 : NS = 0
28050 GOSUB 29000
28060 FI = 0
28070 PRINT#15,"B-P:"8,FI*32+2
28080 GET#8,T$
28090 T = ASC(T$+N$) AND 127
28100 IF T<>0 THEN 28170
28110 GET#8,T$ : GET#8,T$
28120 T1$ = ""
28130 FOR I = 0 TO 15
28140 GET#8,T$ : T1$ = T1$+LEFT$(T$+N$,1
)
28150 NEXT
28160 IF LEFT$(NA$+S$,16)=T1$ THEN CO=-1
 : GOTO 28190
28170 IF FI<7 THEN FI = FI+1 : GOTO 2807
0
28180 IF NT<>0 THEN 28050
28190 RETURN
29000 REM*******************************
29010 REM NAECHSTE SPUR UND SEKTOR
29015 REM LESEN
29020 REM*******************************
29030 TR = NT : S = NS
29040 PRINT#15,"U1:"8;0;TR;S
29050 PRINT#15,"B-P:"8;0
29060 GET#8,T$ : NT = ASC(T$+N$)
29070 GET#8,T$ : NS = ASC(T$+N$)
29090 RETURN
30000 REM*******************************
```

```
30010 REM UNSCRATCH HAUPTPROGRAMM
30020 REM*****************************
30030 GOSUB 28000
30040 IF NOT CO THEN 30140
30050 PRINT#15,"B-P"8;FI*32+2
30060 T = 0
30070 IF TY$="SEQ" THEN T=1
30080 IF TY$="PRG" THEN T=2
30090 IF TY$="USR" THEN T=3
30100 IF TY$="REL" THEN T=4
30110 T = T+128
30120 PRINT#8,CHR$(T) ;
30130 PRINT#15,"U2:"8;0;TR;S
30140 RETURN
```

ERKLÄRUNG DES UNSCRATCH-PROGRAMMS

Zeilen 27000–27200: Dieser Abschnitt erlaubt es dem Benutzer, den Namen der gelöschten Datei und den Dateityp, dem die Datei zugeordnet werden soll, wenn sie wieder hergestellt ist, anzugeben. Am Ende des Moduls kommt der VALIDATE-Befehl zum Einsatz, nachdem der Rest des Programms die Datei wieder in das Directory geschrieben hat, um die BAM auf den neuesten Stand zu bringen, welche die von der Datei beanspruchten Sektoren als frei gekennzeichnet hatte.
Anmerkung: Dieses Programm ist nicht imstande, eine Datei wieder herzustellen, wenn schon anderes Material in die Sektoren geschrieben wurde, die vormals von der mit SCRATCH gelöschten Datei belegt waren, oder der Directory-Eintrag überschrieben wurde.

Zeilen 29000–29080: Diese Zeilen lesen den gegenwärtigen Sektor von der Directory-Spur und speichern die Zeiger zu nächsten Spur und zum nächsten Sektor in den Variablen NT und NS. Jedesmal, wenn dieses Modul aufgerufen wird, werden die nächste Spur und der nächste Sektor mit "U1:" in den Puffer gelesen und die Spur- und Sektor-Zeiger auf den Stand des folgenden Sektors gesetzt.

Zeilen 28000–28190: Die Aufgabe dieses Moduls ist, die Directory-Spur nach gelöschten Dateien zu durchsuchen und die Namen der gefundenen mit dem der wieder herzustellenden Datei zu vergleichen. Dies geschieht, indem ein Sektor des Directorys mit Hilfe des Moduls in Zeile 29000 in den Puffer gelesen wird. Dann wird

der Puffer-Zeiger in 32-Byte-Schritten durch den Puffer bewegt – ein einzelner Directory-Eintrag ist 30 Bytes lang, und mit den zwei Bytes zwischen jeweils zwei Einträgen werden es 32.
Bei jedem Schritt wird das erste Byte jedes Eintrags überprüft, ob es null ist oder 128, dem Zeichen für die mit SCRATCH gelöschten Dateien. *Wird* eine gelöschte Datei gefunden, wird deren Name aus dem Puffer herausgeholt und mit dem Namen des angegebenen Programms verglichen. Gleichen sie sich, wird die Variable CO auf minus eins gesetzt; falls nicht, wird die Suche fortgesetzt, bis das Ende des Directorys erreicht ist.

Zeilen 30000–30140: Zeigt das Flag CO an, daß die richtige Datei gefunden wurde, wird die entsprechende Nummer des Dateityps, dem nach der Wiederherstellung durch UNSCRATCH zugeordnet werden soll, in das erste Zeichen des Directory-Eintrags im Puffer geschrieben. Dann wird der Puffer zurück auf die Diskette geschrieben, womit der Zustand der Datei im Directory wieder geändert ist. Falls der angegebene Dateityp unrichtig ist, teilt das Programm dies dem Benutzer mit und versieht die Datei mit dem Kennzeichen "DEL" oder "deleted" (zu deutsch "gelöscht"). Dies wird im Directory erscheinen und kann nun mit UNSCRATCH gegen einen gültigen Dateityp ausgetauscht werden.

LISTING DES PROGRAMMS LIST TRACK AND SECTORS

```
21000 REM##############################
21010 REM SPUR UND SEKTOR LESEN
21020 REM##############################
21030 DIM D$(35,20),DI$(144),TY$(3)
21040 TY$(0) = "SEQ"
21050 TY$(1) = "PRG"
21060 TY$(2) = "USR"
21070 TY$(3) = "REL"
21080 INPUT "GERAETENUMMER ? 8";DEV
21090 OPEN 15,DEV,15
21100 OPEN 8,DEV,8,"#"
21110 PRINT "       EINLESEN DER DAT
EN"
21120 GOSUB 24000
21130 CLOSE 8 : CLOSE 15
21140 GOSUB 26000
21150 END
```

```
22000 REM#################################
22010 DATA "SPUR-UND SEKTORGROESSEN"
22020 REM#################################
22030 DATA 1,17,20,18,24,18,25,30,17,31,
35,16
23000 REM#################################
23010 REM FILE LESEN (START BEI TR & S)
23020 REM#################################
23030 BL = 1
23040 D$(TR,S) = N$+" BLOCK"+STR$(BL)
23050 BL = BL+1
23060 PRINT#15,"U1:"8;0;TR;S
23070 PRINT#15,"B-P:"8;0
23080 GET#8,T$ : TR = ASC(T$+CHR$(0))
23090 GET#8,T$ : S = ASC(T$+CHR$(0))
23100 IF TR>0 THEN 23040
23110 RETURN
24000 REM#################################
24010 REM EINLESEN DER DATEN DER DISK
24020 REM#################################
24030 RESTORE
24040 READ T$ : IF T$<>"SPUR-UND SEKTORG
ROESSEN" THEN 21070
24050 FOR I = 1 TO 4
24060 READ T1,T2,S1
24070 FOR TR = T1 TO T2
24080 FOR S = 0 TO S1
24090 D$(TR,S) = "UNUSED"
24100 NEXT S,TR,I
24110 REM ----- EINLESEN DER BAM -----
24120 PRINT#15,"U1:"8;0;18;0
24130 PRINT#15,"B-P:"8;4
24140 T1$ = ""
24150 FOR I = 0 TO 143 : GET#8,T$ : T1$
= T1$+LEFT$(T$+CHR$(0),1) : NEXT
24160 T = 0
24170 RESTORE
24180 READ T$ : IF T$<>"SPUR-UND SEKTORG
ROESSEN" THEN 24180
24190 FOR I = 1 TO 4
```

```
24200 READ T1,T2,S1
24210 FOR TR = T1 TO T2
24220 FOR S = 0 TO S1
24230 T = TR*32+S-24
24240 T3 = INT(T/8)+1
24250 T4 = 2↑(T-(T3-1)*8)
24260 T = ASC(MID$(T1$,T3,1))
24270 IF (T4 AND T) = 0 THEN D$(TR,S) =
"RANDOM FILE"
24280 NEXT S,TR,I
24290 REM -- EINLESEN DER DIRECTORY --
24300 N$ = "DIRECTORY"
24310 TR = 18 : S = 0
24320 GOSUB 23000
24330 REM ---- LESEN DER FILENAMEN ----
24340 GOTO 25000
24350 IF DP<1 THEN RETURN
24360 FOR I = 1 TO DP
24370 T = ASC(DI$(I))
24380 IF T<129 OR T>132 THEN 24430
24390 TR = ASC(MID$(DI$(I),2,1))
24400 S = ASC(MID$(DI$(I),3,1))
24410 N$ = MID$(DI$(I),4,16)+","+TY$(T-1
29)
24420 GOSUB 23000
24430 NEXT I
24440 RETURN
25000 REM*******************************
25010 REM DIRECTORY IN DI$ EINLADEN
25020 REM*******************************
25030 DP = -1 : NT = 18 : NS = 0
25040 TR = NT : S = NS
25050 PRINT#15,"U1:" 8;0;TR;S
25060 PRINT#15,"B-P:" 8;0
25070 GET#8,T$ : NT = ASC(T$+CHR$(0))
25080 GET#8,T$ : NS = ASC(T$+CHR$(0))
25090 IF TR=18 AND S=0 THEN 25040
25100 PRINT#15,"B-P:" 8;0
25110 FOR I = 0 TO 7
25120 GET#8,T$ : GET#8,T$
```

```
25130 DP = DP+1
25140 DI$(DP) = ""
25150 FOR J = 0 TO 29
25160 GET#8,T$
25170 DI$(DP) = DI$(DP)+LEFT$(T$+CHR$(0)
,1)
25180 NEXT J,I
25190 IF TR>0 THEN 25040
25200 RETURN
26000 REM*******************************
26010 REM ANZEIGE VON SPUR UND SEKTOR
26020 REM*******************************
26030 TR = 1 : S = 0
26040 PRINT "□"
26050 PRINT "■SPUR =     ■■■■■"TR"   "
26060 RESTORE
26070 READ T$ : IF T$<>"SPUR-UND SEKTORG
ROESSEN" THEN 26070
26080 READ T1,T2,S1
26090 IF TR>T2 THEN 26080
26100 FOR S = 0 TO S1
26110 PRINT LEFT$(STR$(S)+"          ",8)
;
26120 PRINT LEFT$(D$(TR,S)+"
                    ",31)
26125 REM OBEN 8 UND UNTEN 31 LEERSTELLE
N
26130 NEXT
26140 FOR T = S1 TO 20
26150 PRINT "
          " : REM 38 LEERSTELLEN
26160 NEXT T
26170 PRINT "F1 = WEITER,F3 = ZURUECK,F5
 = NEUE SPUR"
26175 PRINT "F7 = PROGRAMM BEENDEN";
26180 GET T$
26190 T = ASC(T$+CHR$(0))-132
26200 IF T<1 OR T>4 THEN 26180
26210 ON T GOTO 26220,26260,26290,26410
26220 TR = TR+1
```

```
26230 IF TR>35 THEN TR = 1
26240 PRINT "[redacted]" ;

26245 PRINT "
            "
26246 PRINT "
   ";
26250 GOTO 26050
26260 TR = TR-1
26270 IF TR<1 THEN TR=35
26280 GOTO 26240
26290 PRINT "SPUR =
          SPUR =   " ;
26300 T1$ = ""
26310 GET T$ : IF T$<>CHR$(20) AND T$<>C
HR$(13) AND (T$<"0" OR T$>"9") THEN 2631
0
26320 IF T$<>CHR$(13) THEN 26360
26330 T=VAL(T1$)
26340 IF T>=1 AND T<=35 THEN TR=T : GOTO
 26240
26350 GOTO 26290
26360 IF T$=CHR$(20) AND T1$<>"" THEN T1
$ = LEFT$(T1$,LEN(T1$)-1) : GOTO 26390
26370 IF LEN(T1$)>=2 THEN 26310
26380 T1$ = T1$+T$
26390 PRINT "■" T$ " ";
26400 GOTO 26310
26410 PRINT "■"
26420 RETURN
```

ERKLÄRUNG VON LIST TRACK AND SECTORS

Zeilen 21000–21150: Dieses Modul stellt die Felder auf, mit deren Hilfe das Programm den Inhalt der einzelnen Sektoren auf der Diskette und die Directory-Einträge speichert. Es erstellt dann eine Datei für den Fehlerkanal und eine Direktzugriffs-Datei für das angegebene Gerät.

Zeilen 23000–23110: Diesem Modul werden jedesmal, wenn es aufgerufen wird, zwei Informationen übermittelt: der Name einer Datei und die Adresse des ersten

Sektors dieser Datei auf der Diskette. Die Aufgabe dieses Moduls ist, mit Hilfe der Spur- und Sektor-Zeiger am Anfang eines jeden Sektors die Datei auf der Diskette zu durchforsten. Dabei schreibt es den Namen der Datei, zu dem der Sektor gehört, und die Nummer des Sektors innerhalb der Datei in das dazugehörige Element des Feldes D$.

Zeilen 24000–24100: Basierend auf der Data-Zeile 22030, wird das Feld D$ mit dem Wort "UNUSED" (zu deutsch "unbenutzt") an den Stellen aufgefüllt, die der Spur- und Sektornummer auf der Diskette entsprechen. So werden D$ (1,0) bis D$ (1,20) anfänglich als unbenutzt gekennzeichnet, da es 21 Sektoren in der Spur 1 gibt. Die Zeilen des Feldes D$, die zu den weiter innen liegenden Spuren gehören, werden daher weniger Elemente haben.

Zeilen 24110–24280: Nachdem alle möglichen Sektoren im Feld als unbenutzt gekennzeichnet sind, liest der zweite Abschnitt des Moduls die Spur 18, Sektor 0 in den Puffer und kopiert dann die BAM in die Stringvariable T1$. Sich auf die Spur- und Sektorziffern aus Zeile 22030 beziehend, untersuchen die Zeilen 24210–24280 jedes einzelne Bit des Bytes innerhalb von T1$, welches die Belegung der Sektoren enthält. Für jedes ungesetzte (Null-)Bit wird das Kennzeichen eines belegten Sektors, also das passende Element des Feldes D$, dahingehend verändert, daß es dann RANDOM FILE (Direktzugriffs-Datei) heißt. Spätere Teile dieses Programms werden die meisten dieser Einträge wegen der im Directory enthaltenen Informationen umschreiben. Aber jeder von der BAM als belegt dargestellte Eintrag, der zusätzlich dazu nicht zu einer Directory-Datei gehört, bleibt Teil einer Direktzugriffs-Datei.

Zeilen 24290–24320: Diese Zeilen rufen das Modul in Zeile 23000 auf, um die Sektoren, die das Directory darstellen, zu durchforsten und die einzelnen Sektorpositionen im Feld D$ aufzuzeichnen.

Zeilen 24330–24440: Nachdem das Modul in Zeile 25000, welches jeden Directory-Eintrag in das Feld DI$ liest, aufgerufen wurde, übermitteln diese Zeilen dem Modul in Zeile 23000 die Informationen über den Beginn der einzelnen Dateien. Das Modul in Zeile 23000 durchsucht die Datei Sektor für Sektor und schreibt den Namen der Datei in das relevante Element des Feldes D$.

Zeilen 25000–25200: Dieses Modul liest den Inhalt des Directorys in das Feld DI$. Von jedem Eintrag werden die zwei übrigen Bytes gelesen und abgelegt (Zeile 25120). Dann werden 30 Bytes, die Länge eines Directory-Eintrags, in das Feld DI$ gelesen und der Zähler DP erhöht. Der Vorgang läuft so lange, bis ein Spurzeiger

am Anfang eines Directory-Sektors mit dem Wert Null gefunden wird. Dies zeigt das Ende des Directorys an. Eine detaillierte Erklärung finden Sie in Kapitel 11.

Zeilen 26000–26420: Dies ist das Anzeige-Modul, welches den Inhalt einer einzelnen Spur auf dem Bildschirm auflistet und dem Benutzer ermöglicht, eine Spur für die Anzeige zu bestimmen.

Anmerkung: Wegen der zahlreichen Stringfelder wird das Programm von Zeit zu Zeit anhalten. Dann wird eine String-Aufräum-Funktion, die sogenannte "garbage collection" (zu deutsch etwa "Müll einsammeln") vom C 64 durchgeführt.

KAPITEL 11

DAS DIRECTORY DER DISKETTE

1. *Das Format des Directorys*
2. *Das Lesen des Directorys*
3. *Die Anwendung einer Operatioon auf mehrer Dateien*

Das Directory der Diskette wurde schon einmal kurz im ersten Kapitel dieses Buches angesprochen. Seitdem betrachten wir die Arbeit des Directorys als selbstverständlich, also die Möglichkeit, den Inhalt einer Diskette zu untersuchen oder dem DOS bei der Suche nach einer bestimmten Datei zu helfen. In diesem Kapitel wollen wir uns das Directory, dessen Aufbau und die Art, wie man direkt auf das Directory zugreifen kann, noch einmal anschauen.

1. DAS FORMAT DES DIRECTORYS

In Tabelle 1.2 (Seite 14) ist der gesamte Aufbau der Spuren, die für das Directory zuständig sind, dargestellt. Das Directory liegt in der Spur 18 und beginnt dort in Sektor 0. Der erste Directory-Sektor steht der BAM zur Verfügung, der Rest der Spur 18 ist jedoch für die einzelnen Dateien auf der Diskette reserviert. Der zweite Abschnitt der Tabelle 1.2 zeigt, wie jeder dieser Sektoren in der Lage ist, die Details von acht Dateien aufzunehmen. Es gibt 17 Sektoren in der Spur 18 der Diskette. Also ist maximale Anzahl Dateien, die auf der Diskette gespeichert werden kann, 16*8 oder 144, egal wieviel Platz auf der Diskette ist.
Das Format eines Eintrags für eine einzelne Datei ist innerhalb des gesamten Directorys immer gleich und wird in Tabelle 11.1 dargestellt.
In der Tat wird Ihnen das meiste in dieser Tabelle schon aus früheren Kapiteln bekannt sein. Die Dateiarten sind im Byte 0 des Eintrags gespeichert, den wird uns schon im UNSCRATCH-Programm aus Kapitel 10 zunutze gemacht haben, wo Dateitypen verändert wurden, um Dateien wiederherzustellen, die im Directory als gelöscht registriert waren.
Die Bytes der ersten Spur und des ersten Sektors und der Dateiname selbst wurden vom LIST TRACK AND SECTOR-Programm benutzt, um die jeder einzelnen Datei zugeordneten Sektoren zu durchforsten und dann den Namen der

relevanten Datei gegenüber jedem Sektor der Diskette darzustellen. Bei normalen Anwendungen sollen diese Bytes dem DOS ermöglichen, das Directory nach bestimmten Dateinamen zu durchsuchen und den Anfang der Datei, auf die es zugreifen sollte, zu finden.

Tabelle 11.1. Format eines einzelnen Directory-Eintrags

BYTE	BEMERKUNG
0	Der benutzte Dateityp
	0 = Unbenutzte oder gelöschte Datei (DEL)
	1 = Ungeschlossene SEQuentielle Datei
	2 = Ungeschlossene PRoGramm-Datei
	3 = Ungeschlossene USeR-Datei
	4 = Ungeschlossene RELative Datei
	128 = Geschlossene gelöschte Datei (DEL)
	129 = Geschlossene SEQuentielle Datei
	130 = Geschlossene PRoGramm-Datei
	131 = Geschlossene USeR-Datei
	132 = Geschlossene RELative Datei
1	Die Spur des ersten Dateiblocks
2	Der Sektor des ersten Dateiblocks
3–18	Filename ergänzt durch "SHIFT SPACEs" (CHR$(160))
19	Relative Dateien – Spur des ersten Side-Sector-Blocks
	Andere Dateitypen – Unbenutzt
20	Relative Dateien – Sektor des ersten Side-Sector-Blocks
	Andere Dateitypen – Unbenutzt
21	Relative Dateien – Recordlänge
	Andere Dateitypen – Unbenutzt
22–25	Unbenutzt
26–27	Wird nur gesetzt, wenn mit SAVE eine Datei abgespeichert oder mit "@0:" eine Datei geöffnet wird
28–29	Die Anzahl der Blöcke in dieser Datei

In Kapitel 9 haben wir gesehen, daß relative Dateien eigentlich aus zwei getrennten Teilen bestehen. Einer enthält die Daten, während der andere aufzeichnet, wo sich die Sektoren, die die Daten gespeichert haben, auf der Diskette befinden. Die Tabelle zeigt, daß die Anfangsadresse des zweiten Teils der relativen Datei in den Bytes 19 und 20 festgehalten wird. Im Gegensatz dazu wird die festgelegte Länge eines jeden Eintrags in einer relativen Datei im Byte 21 festgehalten.
Die Bytes 26 und 27 sind neu, doch ist deren Gebrauch recht einfach. Wird eine Datei mit Hilfe des "@0:" abgespeichert oder geöffnet, um zu bestimmen, daß jede

vorherige Datei mit demselben Namen und vom selben Dateityp überschrieben wird, haben diese Bytes die Aufgabe, die Startspur und den Startsektor festzuhalten, bis die neue Datei erstellt ist.
Wenn schließlich das Directory angezeigt wird, wird die Größe der Datei in Form von belegten Sektoren mit angezeigt. Diese Zahl wird in den Bytes 28 und 29 des Dateieintrags gespeichert.
Im ganzen beansprucht ein Dateiantrag im Directory bis zu 30 Bytes (0–29) auf der Diskette. Um die acht möglichen Einträge innerhalb der 256 Bytes eines Sektors regulär unterzubringen, werden noch zwei Extra-Bytes an das Ende der ersten sieben Einträge gehängt. Diese Bytes enthalten keine brauchbaren Informationen, sondern sind allein dazu da, dem DOS zu ermöglichen, das Directory in Schritten zu 32 Bytes abzutasten.

2. DAS LESEN DES DIRECTORYS

Das Directory kann auf zwei Wegen gelesen werden:

1. Es kann mit dem Befehl LOAD "$",<DEV> in den Speicher geladen werden, wobei DEV die Gerätenummer (von engl. DEVice number) des benutzten Laufwerkes ist. Wird das Directory auf diese Art geladen, wird es fast genauso behandelt wie eine Programm-Datei, und alle momentan im Speicher befindlichen Programme, werden gelöscht. Das Laden ist möglich, weil das "$" dem DOS anzeigt, daß es das Directory in eine Programm-Datei übersetzen muß, wobei es jeden Eintrag wie eine Programmzeile behandelt. Es liefert die Null-Bytes, die das Ende einer Zeile anzeigen und den Platz für die "link bytes" (Verbindungs-Bytes). Mit anderen Worten: Das dem C 64 übermittelte Format unterscheidet sich, wenn der Befehl LOAD "$" eingegeben wird, völlig von dem Format des Directorys auf der Diskette.

2. Das Directory kann auch innerhalb eines Programms von der Diskette gelesen werden. Die "DOS 5.1"-Software, die den neueren 1541-Laufwerken beiliegt, enthält eine nette Routine, die dies ermöglicht und den Inhalt des Directorys auf den Bildschirm bringt, ohne daß das laufende Programm gestört wird (siehe Anhang C). Es ist jedoch auch möglich, das Directory vom BASIC aus zu lesen. Ein Beispiel ist das LIST TRACK AND SECTOR-Programm in Kapitel 10. Nachfolgend finden Sie zwei kurze Programme, die das Directory in ein Feld lesen. Das erste liest die Directory-Datei ungefähr wie eine Programm-Datei (siehe Kapitel 8), das zweite liest die Diskette unmittelbar.

EIN PROGRAMM, DAS DAS DIRECTORY VON DER DATEI "$" IN EIN FELD LIEST

```
10 DIM DI$(150) : DEV = 8
20 GOSUB 1000
30 FOR I = 0 TO DP-1
40 PRINT CHR$(34) DI$(I)
50 NEXT
60 END
1000 REM*********************************
1010 REM EINLADEN DES DIRECTORYS
1020 REM*********************************
1030 DP = 0
1040 OPEN 8,DEV,0,"$"
1050 GET#8,T$ : GET#8,T$
1060 GET#8,T$ : GET#8,T$ : IF T$="" THEN
 CLOSE 8 : RETURN
1070 GET#8,T$ : GET#8,T$
1080 GET#8,T$
1090 IF T$<>CHR$(34) AND T$<>"" THEN 108
0
1100 IF T$="" THEN 1060
1110 T1$ = ""
1120 GET#8,T$
1130 IF T$<>"" THEN T1$ = T1$+T$ : GOTO
1120
1140 DI$(DP) = T1$ : DP = DP+1
1150 GOTO 1060
```

ERKLÄRUNG DES PROGRAMMS

Wenn Sie das Kapitel 8 über die Programm-Dateien gelesen haben, sollten Sie kaum Schwierigkeiten haben, die hier angewandten Techniken zu erkennen. Das DOS liefert das Directory in Form einer Programm-Datei, wobei jeder Filename in eine extra Zeile geschrieben und das Ganze noch mit den "link bytes" und dem anderen Zubehör versehen wird. Weil es überhaupt keine Übereinstimmung zwischen den beiden gibt, braucht man erst gar nicht zu versuchen, das mit den GET#-

Statements gelesene mit dem Inhalt der Tabelle am Anfang dieses Kapitels zu vergleichen. Was hier gelesen wird, ist nicht das Directory, sondern die übersetzte, vom DOS gelieferte Version des Directorys.

PROGRAMM ZUM DIREKTEN LESEN DES DIRECTORYS VON DER DISKETTE IN EIN FELD

```
10 OPEN 15,8,15 : OPEN 8,8,8,"#"
20 DIM DI$(150)
30 GOSUB 1000
40 FOR I = 0 TO DP
50 T = ASC(DI$(I)) : IF T>=129 AND T<=13
0 THEN PRINT MID$(DI$(I),4,16)
60 NEXT
70 CLOSE 8 : CLOSE 15
80 END
1000 REM#################################
1010 REM DIREKTES EINLESEN DES
1015 REM DIRECTORYS IN DI$
1020 REM#################################
1030 DP = -1 : NT = 18 : NS = 0
1040 TR = NT : S = NS
1050 PRINT#15,"U1:" 8;0;TR;S
1060 PRINT#15,"B-P:"8;0
1070 GET#8,T$ : NT = ASC(T$+CHR$(0))
1080 GET#8,T$ : NS = ASC(T$+CHR$(0))
1090 IF TR=18 AND S=0 THEN 1040
1100 PRINT#15,"B-P:" 8;0
1110 FOR I = 0 TO 7
1120 GET#8,T$ : GET#8,T$
1130 DP = DP+1
1140 DI$(DP) = ""
1150 FOR J = 0 TO 29
1160 GET#8,T$
1170 DI$(DP) = DI$(DP)+LEFT$(T$+CHR$(0),
1)
1180 NEXT J,I
1190 IF NT>0 THEN 1040
1200 RETURN
```

Zeilen 10–80: Dieser Abschnitt steuert den Programmablauf. Seine drei Hauptaufgaben sind den Fehlerkanal zu öffnen und die Belegung eines Laufwerkspeicher-Puffers anzumelden, das nächste Modul aufzurufen und dann die gewählten Datein aus dem Feld DI$ auszudrucken.

Zeilen 1000–1200: Diese Zeilen sind uns schon einmal im LIST TRACK AND SECTOR-Programm aus Kapitel 10 begegnet. Ihre Aufgabe ist es, den Inhalt des Directorys in das Feld DI$ zu lesen.

Zeilen 1050–1060: Der Inhalt eines einzelnen Sektors wird in den Puffer gelesen und der Puffer-Zeiger auf den Anfang des Puffers gesetzt. Der erste zu lesende Sektor ist Spur 18, Sektor 0.

Zeilen 1070–1090: Die ersten zwei Bytes des Sektors, beides Zeiger, werden gelesen und in den zwei Variablen NT und NS für Next Track (nächste Spur) und Next Sector (nächster Sektor) gespeichert. Beim ersten Durchlauf durch das Modul wird die BAM aufgenommen, deshalb springt das Programm gleich zum nächsten Sektor.

Zeilen 1100–1180: Der Puffer-Zeiger wird zurück auf den Anfang des Blocks gesetzt, dann werden die acht im Sektor enthaltenen Dateieinträge nacheinander gelesen. Dafür müssen zunächst die zwei unbenutzten vorangestellten Bytes gelöscht und dann die nächsten 30 Zeichen gelesen werden. Dieser 30-Zeichen-Eintrag wird anschließend in einer Zeile des Feldes DI$ gespeichert.

Zeilen 1190–1200: Zeigt der nächste Spurzeiger an diesem Punkt auf die Spur 0, ist dies ein Zeichen dafür, daß der gerade bearbeitete Sektor der letzte des Directorys war.

3. DIE ANWENDUNG EINER OPERATION AUF MEHRERE DATEIEN

Weil die Methode, mit LOAD "$" und mit der "DOS 5.1"-Einrichtung das Directory auszudrucken, recht flexibel ist, gibt es nur wenige Anlässe, das Directory direkt zu lesen. Eine Notwendigkeit könnte jedoch bestehen, wenn eine Operation auf mehrere Dateien angewendet werden soll. In Kapitel 5 erwähnten wir, daß es nur ein paar Befehle gibt, die mit dem "pattern matching" der 1541 verwendet werden können. Mit geringem Programmieraufwand ist es allerdings möglich, einige Routinen zu entwickeln, die eine Operation auf eine ganze Reihe von Dateien, welche in ein bestimmtes Schema passen, wirken lassen. Dies aber hängt von der Fähigkeit ab, die im Directory enthaltene Information lesen und auswerten zu können.

PROGRAMM ZUR ANWENDUNG EINER OPERATION AUF MEHRERE DATEIEN

```
30900 GOTO 33000
31000 REM#******************************
31010 REM EINLESEN DER 1541 DIRECTORY
31020 REM******************************
31030 DP = 0
31040 OPEN 8,DEV,0,"$"
31050 GET#8,T$ : GET#8,T$
31060 GET#8,T$ : GET#8,T$ : IF T$ = "" T
HEN CLOSE 8 : RETURN
31070 GET#8,T$ : GET#8,T$
31080 GET#8,T$
31090 IF T$<>CHR$(34) AND T$<>"" THEN 31
080
31100 IF T$="" THEN 31060
31110 T1$ = ""
31120 GET#8,T$
31130 IF T$<>"" THEN T1$ = T1$+T$ : GOTO
 31120
31140 DI$(DP) = T1$ : DP = DP+1
31150 GOTO 31060
32000 REM#******************************
32010 REM VERGLEICH VON N$ MIT PA$
32020 REM******************************
32030 SAME = -1
32040 T = LEN(N$)
32050 IF LEN(PA$)>T THEN T = LEN(PA$)
32060 FOR I = 1 TO T
32070 T1 = (MID$(PA$,I,1)=MID$(N$,I,1))
OR (MID$(PA$,I,1)="?")
32080 SAME = SAME AND ((MID$(PA$,I,1)="*
") OR T1)
32090 IF MID$(PA$,I,1)<>"*" THEN NEXT I
32100 RETURN
33000 REM#******************************
33010 REM WIEDERHOLUNG FUER ALLE FILES
33020 REM******************************
33030 DIM DI$(150)
```

```
33040 INPUT "GERAETENUMMER ? 8[■■■]";DEV
33050 INPUT "PROGRAMMNAME ? *[■■■]";PA$
33060 GOSUB 31000
33070 IF DP<2 THEN 33200
33080 FOR I0 = 1 TO DP-1
33090 T$ = DI$(I0) : T = -1
33100 FOR I1 = 1 TO LEN(T$)
33110 IF MID$(T$,I1,1)=CHR$(34) THEN T =
 I1-1
33120 NEXT I1
33130 IF T<1 THEN 33190
33140 N$ = LEFT$(T$,T)
33150 TY$ = MID$(T$,19,3)
33160 GOSUB 32000
33170 IF NOT SAME THEN 33190
33180 GOSUB XXXXX : REM HIER MUSS DIE ZE
ILENNUMMER DER ROUTINE, DIE BENUTZT
33185 REM WERDEN SOLL, EINGESETZT WERDEN

33190 NEXT I0
33200 END
```

ERKLÄRUNG DES PROGRAMMS

Zeilen 31000–31150: Dieses Modul liest mit Hilfe der ersten der beiden im zweiten Abschnitt dieses Kapitels erklärten Methode die Dateinamen aus dem Directory.

Zeilen 32000–32100: Diese Zeilen vergleichen zwei Strings. Der eine ist der Dateiname aus dem Directory, der andere ist der im nächsten Modul eingelesene String, der die Zeichenfolge darstellt, mit der alle Dateien der Diskette verglichen werden. Die Zeichenfolge kann, wie in Kapitel 5 beschrieben, mit dem "*"-Zeichen und dem "?"-Zeichen aufgebaut sein. Das einzige wichtige Produkt dieses Moduls ist der Wert der Variable SAME. Paßt der von dem Modul betrachtete Dateiname zu der Zeichenfolge, bleibt der Wert von SAME bei minus eins, andernfalls wird es nach Durchlaufende null sein.

Zeilen 33000–33200: Dieser Abschnitt enthält das Hauptsteuermodul. Dies ruft das Modul in Zeile 31000 auf, um das Directory in das Feld DI$ zu lesen und

übermittelt dann dem vorigen Modul aufeinanderfolgende Filenamen, die für den Vergleich mit der vom Benutzer eingegebenen Zeichenfolge bestimmt sind. Die Erstellung von TY$, welche den Dateityp aufzeichnet, stellt eine Extra-Einrichtung dar. Von dieser wird zwar im vorliegenden Programm kein Gebrauch gemacht, aber es könnte sein, daß Sie diese Einrichtung einsetzen wollen, um bestimmte Fileytypen, ungeachtet deren Namen von einer Operation auszuschließen.
Beim tatsächlichen Gebrauch würde noch ein weiteres Modul nötig sein, das angibt, was genau mit einer Datei, die zu der Zeichenfolge paßt, getan werden soll. Dieser Extra-Abschnitt würde in eine weitere Subroutine geschrieben und durch das GOSUB in Zeile 33180 aufgerufen. *Anmerkung:* Da die Programmzeile 33180 keine funktionstüchtige Zeile darstellt, kann die Routine in dieser Form nicht gestartet werden – Sie *müssen* erst einen Abschnitt hinzufügen, der die durchzuführende Operation bestimmt. Weiter unten finden Sie ein Anwendungsbeispiel, welches den Einsatz der REPEAT-Einrichtung näher darstellt.

ANWENDUNGSBEISPIEL ZUM EINSATZ VON REPEAT

1. Geben Sie das obige REPEAT-Programm ein, und speichern Sie es mit SAVE ab.

2. Nehmen Sie eine Diskette, die keine wertvollen Dateien enthält (es könnte ja etwas schiefgehen), oder formatieren Sie eine neue Diskette, und speichern Sie drei Dateien mit unterschiedlichen Namen auf ihr ab – der Inhalt der Dateien ist unwichtig, nur sollten die Dateinamen *weniger* als 16 Zeichen lang sein.

3. Laden Sie die Wiederholungseinrichtung mit LOAD, und berichtigen Sie sie, indem Sie die folgenden abgeänderten bzw. neuen Zeilen eingeben:

```
38180 GOSUB 34000

34000 REM*******************************
34010 REM ALLE FILES MIT RENAME
34015 REM UMBENENNEN
34020 REM*******************************
34030 OPEN 15,DEV,15
34040 COM$="RENAME0:Z" + N$ + "=0:" + N$

34050 PRINT#15,COM$
34060 CLOSE 15
34070 RETURN
```

4. Speichern Sie das berichtigte Programm unter dem Namen REPEAT 2 mit SAVE ab.

5. Starten Sie das Programm mit RUN, und wenn Sie gebeten werden, eine Zeichenfolge einzugeben, drücken Sie einfach die RETURN-Taste. Dieses bewirkt die Eingabe eines einzelnen Sternchens, welches anzeigt, daß jeder Filename für den Vergleich akzeptiert wird.

6. Wenn das Programm unterbricht, laden Sie das Directory, und Sie werden sehen, daß jede Datei auf der Diskette ein "Z" am Anfang des Dateinamens stehen hat. Sie haben somit erfolgreich einen Vorgang ausgeführt, der mit dem normalen "pattern matching" nicht möglich gewesen wäre.

KAPITEL 12

BEFEHLE FÜR DIE PROGRAMMIERUNG IN MASCHINENSPRACHE

1. *Das Lesen des Laufwerkspeichers*
2. *Das Hineinschreiben in den Speicher*
3. *Die Anwendung von Maschinensprache im Laufwerkspeicher*

Trotz der großen Leistungsfähigkeit und Flexibilität des Disk Operating Programms, welches 16 K im ROM der 1541 beansprucht, kann es Situationen geben, in denen der erfahrene Programmierer seine eigenen Maschinenspracheroutinen schreiben will, um die Floppy Disk zu steuern. Es muß aber noch einmal gesagt werden, daß dies keine Entscheidung ist, die auf die leichte Schulter genommen werden kann. Obwohl es unwahrscheinlich ist, ist es dennoch *möglich*, das Laufwerk durch Betreiben mit schlecht durchdachten Programmen zu beschädigen, da die 1541, anders als der C 64, bewegliche Teile besitzt, die von ihrem 6502-Mikroprozessor gesteuert werden.

1. DAS LESEN DES LAUFWERKSPEICHERS

Das Lesen der Inhalte des RAMs und des ROMs der Floppy Disk, die zwischen Adresse $0 und 7FF bzw. $C000 und FFFF liegen, wird mit Hilfe des MEMORY-READ-Befehls möglich. Das Format ist:

PRINT#<**FILENUMMER**>,"**M-R**"CHR$(**LO**)CHR$(**HI**)[CHR$(**CHARS**)]

1. **FILENUMMER** – die Nummer der vorher für den Fehlerkanal geöffneten Datei.

2. **"M-R"** – die Abkürzung von MEMORY-READ. Die vollständige Version wird nicht akzeptiert. *Anmerkung:* Das Fehlen des Doppelpunktes ":" am Ende des Befehls ist *kein* Fehler. Einige Ausgaben des 1541-Handbuches, die vor diesem Buch erschienen sind, schließen den Doppelpunkt fälschlicherweise in das Format des Befehls ein. Keiner der "M-"-Befehle wird von der 1541 akzeptiert, wenn der Doppelpunkt dabei steht.

3. **LO und HI** – werden wie folgt berechnet: Ist X die zu lesende Adresse, dann ist HI = INT(X/256) und LO = X–256*HI

4. **CHARS** – die Anzahl der zu lesenden Bytes – siehe unten über die Anwendung von GET# zum Lesen der Daten.

Der M-R-Befehl selbst liest den Speicher jedoch nicht. Vielmehr liest er ein einzelnes Byte von der angegebenen Speicherstelle in den Fehlerkanal-Puffer, es sei denn, das nicht unbedingt notwendige CHARS wird hinzugefügt. Dort wartet es darauf, in den C 64 geladen zu werden. Wird dann eine GET#-Anweisung übermittelt, so wird das einzelne Byte aus der angegebenen Adresse gelesen. Die bis zum Erscheinen dieses Buches im 1541-Handbüchern erwähnte Methode, wo der M-R-Befehl ohne das CHARS eingesetzt wird, um dann mit Hilfe des GET# eine Bytekette zu lesen, scheint nicht zu wirken. Unserer Erfahrung nach werden damit nach dem ersten Byte nur noch die Zeichen der vorliegenden Fehlermeldung gelesen. Also muß wohl, entgegen den Informationen im Handbuch, entweder ein neuer M-R-Befehl übermittelt oder die Anzahl der aufzunehmenden Bytes angegeben werden, wobei CHR$ (0) die maximale Anzahl von 256 Bytes darstellt, ohne daß ein neuer M-R-Befehl nötig ist. Beachten Sie: Wann immer weniger Bytes als angegeben gelesen werden, muß die Reihe mit Hilfe des INITIALIZE-Befehls abgeschlossen werden, da bis dahin *nur* Bytes aus dem Speicher aus dem Fehlerkanal gelesen werden.
Das untenstehende Programm zeigt, wie der M-R-Befehl dazu verwendet werden kann, einen bestimmten Bereich des Laufwerkspeichers auszudrucken. Wer unsere anderen Bücher kennt, wird feststellen, daß das Programm aus den sogenannten "memory dump"-Routinen (zu deutsch etwa "Speicherauszug") vom Mastercode Assembler besteht, die so geändert wurden, daß sie anstelle des internen Speichers des C 64 den Diskettenspeicher lesen und formatieren.

PROGRAMM, WELCHES MIT HILFE VON M-R DEN INHALT DES DISKETTENSPEICHERS AUSDRUCKT

```
17000 REM#******************************
17010 REM SPEICHERINHALTE DER
17015 REM DISKETTENSTATION AUSLESEN
17020 REM******************************
17030 DEV = 8
17040 OPEN 15,DEV,15
17050 DEFFNHI(X) = INT ((X-INT(X/65536)*
```

```
65536)/256)
17060 DEFFNLO(X) = X-INT(X/256)*256
17070 INPUT " STARTADRESSE ";T$
17080 IF LEFT$(T$,1)="$" THEN T$ = MID$(
T$,2) : GOTO 17140
17090 AD = VAL(T$)
17100 IF AD<>0 THEN 17170
17110 FOR I = 1 TO LEN(T$) : T1$ = MID$(
T$,I,1)
17120 IF T1$>"9" OR T1$<"0" THEN PRINT "
" : GOTO 17070
17130 NEXT : GOTO 17170
17140 GOSUB 20000
17150 IF ERR THEN PRINT "□□" : GOTO 1707
0
17160 AD = T
17170 IF AD>65536 OR AD<0 THEN PRINT "□□
" : GOTO 17070
17180 INPUT " AUSGABE AUF DEN DRUCKER (J
/N) ? N";T$
17190 OP = T$="J"
17200 IF NOT OP AND T$<>"N" THEN PRINT "
" : GOTO 17180
17210 T = 3
17220 IF OP THEN T = 4
17230 OPEN 1,T
17240 PRINT "□"
17250 FOR I = 0 TO 20
17260 GOSUB 18000
17270 PRINT#1,HE$
17280 NEXT
17290 INPUT "■ WEITER (J/N) ? J■■■";T$
17300 IF T$="J" THEN 17240
17310 IF T$<>"N" THEN PRINT "□□□" : GOTO
 17290
17320 PRINT#15,"I0"
17330 CLOSE 1
17340 CLOSE 15
17350 END
18000 REM*******************************
```

```
18010 REM 8 BYTES VON DER DISK LADEN
18020 REM*******************************
18030 O1$ = "" : O2$ = ""
18040 T1$ = "" : T2$ = ""
18045 PRINT#15,"M-R" CHR$(FNLO(AD)) CHR$
(FNHI(AD)) CHR$(0)
18050 FOR J = 0 TO 7
18070 GET#15,T$ : T$ = LEFT$(T$+CHR$(0),
1)
18080 T = ASC(T$)
18090 GOSUB 19000
18100 T1$ = T1$+RIGHT$("00"+HE$,2)+" "
18110 T3$ = "."
18120 IF T$>CHR$(31) AND T$<CHR$(128) TH
EN T3$ = T$
18130 T2$ = T2$+T3$
18140 NEXT
18150 T = AD
18160 GOSUB 19000
18170 HE$ = RIGHT$("0000"+HE$,4)+" "+T1$
+" "+T2$
18180 AD = AD+8
18190 RETURN
19000 REM#******************************
19010 REM T IN HEX-ZAHL UMWANDELN
19020 REM*******************************
19030 HE$ = ""
19040 T1 = T - INT(T/16)*16
19050 T = INT(T/16)
19060 HE$ = CHR$(T1+48-(T1>9)*7)+HE$
19070 IF T>0 THEN 19040
19080 RETURN
20000 REM#******************************
20010 REM HEX-ZAHL IN DEZIMAL UMWANDELN
20020 REM*******************************
20030 T = 0 : ERR = 0
20040 FOR I = 1 TO LEN(T$)
20050 T1 = ASC(MID$(T$,I,1))-48
20060 IF T1>9 THEN T1 = T1-7+(T1>22)*15
20070 T = T*16+T1
```

```
20080 ERR = T1>15 OR T1<0 OR ERR
20090 NEXT
20100 RETURN
```

ERKLÄRUNG DES PROGRAMMS

Zeilen 17000–17350: Dieser Abschnitt ermöglicht es dem Benutzer, die Startadresse anzugeben, von der aus der Speicher ausgedruckt werden soll. Die Adresse kann sowohl in Dezimalschreibweise als auch, wenn durch ein "$" gekennzeichnet, in Hexadezimalschreibweise eingegeben werden. Die Darstellung der Ausdruckergebnisse geschieht normalerweise auf dem Bildschirm; es ist jedoch auch über den Drucker möglich. Nachdem 20 Zeilen zu acht Bytes dargestellt worden sind, erhält der Benutzer die Möglichkeit, das Programm weiterlaufen zu lassen oder abzubrechen. Beachten Sie, daß beim Abbruch auch die Floppy Disk initialisiert wird, um sicherzugehen, daß sie für nachfolgende Befehle bereit ist. Falls sich keine Diskette im Laufwerk befindet, wird eine Fehlermeldung hervorgerufen.

Zeilen 19000–19080: Diese Zeilen ändern eine in dezimaler Schreibweise eingegebene Zahl in eine äquivalente hexadezimale um.

Zeilen 20000–20100: Analog zum obigen wandeln diese Zeilen eine hexadezimale Zahl in eine dezimale um. Zusätzlich wird die Fehlervariable ERR (für ERRor) auf minus eins gesetzt, falls die Hexadezimalzahl falsch eingegeben wurde.

Zeilen 18000–18190: Dieser Abschnitt liest eine acht Byte lange Zeile aus dem Laufwerkspeicher, übersetzt jedes Byte in eine Hexadezimalzahl, indem er das Modul in Zeile 19000 aufruft, steckt dann den die Hexadezimalzahl repräsentierenden String in T$ und gegebenenfalls das ASCII-Zeichen mit demselben Code in den String T2$. Diese werden dann zu dem auszudruckenden String HE$ Zeile für Zeile von dem Modul in Zeile 17000 zusammengefügt. Das sich ergebende Display zeigt auf der linken Seite des Bildschirms hexadezimale Form von acht Bytes und auf der rechten Seite die Anwesenheit der eventuell dazugehörenden Gruppe ASCII-Zeichen.

2. DAS HINEINSCHREIBEN IN DEN SPEICHER

Daten können in Blöcken von bis zu 34 Bytes in den Laufwerkspeicher hineingeschrieben werden. Dies geschieht mit Hilfe des MEMORY-WRITE-Befehls, dessen Format wie folgt lautet:

PRINT#<**FILENUMMER**>,**"M-W"**CHR$**(LO)**CHR$**(HI)**CHR$**(CHARS)**
R<**BYTES**>

1. **FILENUMMER** – die Nummer einer voher für den Fehlerkanal geöffneten Datei.

2. **"M-W"** – die Abkürzung von MEMORY-WRITE – die vollständige Form wird nicht akzeptiert.

3. **LO und HI** – siehe dieselben Werte im vorigen Abschnitt.

4. **CHARS** – die Anzahl der zu übersendenen Bytes; bis zu 35 in einem einzigen Befehl.

5. **BYTES** – die zu übermittelnden Bytes; entweder in Form von CHR$ (BYTE 1), CHR$ (BYTE 2) usw. oder – wenn geeignet – in Form eines Strings mit ASCII-Zeichen. Sollte die Anzahl der angegebenen Zeichen die im String enthaltenen und zum Fehlerkanal übermittelten überschreiten, wird das Wagenrücklaufzeichen, ASCII-Code 13, am Ende der in den Speicher geschriebenen Zeichen gespeichert.

ANWENDUNGSBEISPIEL ZUR ILLUSTRATION VON M-W

1. Geben Sie das oben stehende MEMORY-DUMP-Programm ein, und speichern Sie es mit SAVE ab.

2. Zeigen Sie mit dem MEMORY-DUMP-Programm im Speicher und einer (nicht allzu wichtigen) Diskette im Laufwerk den bei $400 beginnenden Speicherinhalt an. Wenn Sie nicht schon im Laufwerkspeicher herumgespielt haben, sollte der Inhalt aus Nullen bestehen.

3. Fügen Sie die folgenden Zeilen an das Programm an:

```
21000 OPEN 15,8,15
21010 PRINT#15,"M-W" CHR$(0) CHR$(4) CHR
$(26) "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
21020 CLOSE 15
```

4. Tippen Sie:

RUN21000[RETURN]

5. Nach einer kleinen Pause wird der Cursor zurückkehren, dann starten Sie das Hauptprogramm. Wenn Sie gebeten werden, eine Adresse anzugeben, geben Sie noch einmal $400 ein. Sie sollten sehen, daß die Buchstaben des Alphabets erfolgreich von $400 an aufwärts gespeichert worden sind.

3. DIE ANWENDUNG VON MASCHINENSPRACHE IM LAUFWERKSPEICHER

Wie schon erwähnt, können Maschinenspracheprogramme angewendet werden, um den 6502-Chip zu steuern, der die Floppy Disk betreibt. Solche Programme können aus Routinen des ROMs der 1541 oder aus vom Benutzer geschriebenen Routinen bestehen. Die Anwendung dieser Programme wird mit den MEMORY-EXECUTE-Befehl durchgeführt, dessen Format wie folgt lautet:

PRINT#<**FILENUMMER**>,CHR$**(LO)**CHR$**(HI)**

1. **FILENUMMER** – Nummer einer vorher für den Fehlerkanal geöffneten Datei.

2. **LO und HI** – die Startadresse der Maschinenspracheroutine mit den zwei im ersten Abschnitt dieses Kapitels berechneten Bytes.

KAPITEL 13

DAS ÄNDERN DER GERÄTENUMMER

Wie in Kapitel 1 erwähnt, hängt das Betreiben mehrerer Laufwerke von der Möglichkeit ab, die Gerätenummer eines oder mehrerer Laufwerke ändern zu können, weil Laufwerke mit derselben Gerätenummer des System durcheinander bringen. Zwar würden Daten, die *zu* einer von zwei Laufwerken geteilten Gerätenummer übermittelt werden sollen, korrekt auf beiden abgespeichert, jedoch würde es zum Chaos kommen, wenn zwei Geräte angewiesen würden, Informationen zum C 64 zu senden. Es gibt zwei Methoden, um die Gerätenummer zu ändern. Die eine, indem man eine kleine Veränderung an der Platine der 1541 vornimmt, und die andere durch einen Speicherbefehl. Wir empfehlen Ihnen, die Hardware-Methode von einem fähigen Händler ausführen zu lassen; dagegen können Sie die Software-Lösung mit einem Befehl selbst durchführen. Der Befehl hat folgendes Format:

PRINT#<FILENUMMER>,"M-W"CHR$(119)CHR$(0)CHR$(2)
CHR$(DEV+32)CHR$(DEV+64)

1. **FILENUMMER** – die Nummer einer vorher für den Fehlerkanal geöffneten Datei.

2. **"M-W"** – der MEMORY-WRITE-Befehl.

3. **119 und 0** – die Adresse eines Registers im Diskettenspeicher, d. h. 119+256*0=119.

4. **2** – die Anzahl der in den Speicher geschriebenen Bytes.

5. **DEV** – dies ist die Gerätenummer (von engl. DEVice number), die Sie dem Laufwerk zu geben wünschen.

Folgen Sie dieser Anleitung, um die Gerätenummer eines Laufwerks mit der Nummer 8 so zu ändern, daß es auf die Gerätenummer 9 hört:

1. Vergewissern Sie sich, daß alle anderen Laufwerke mit einer Gerätenummer 8 ausgeschaltet sind.

2. Tippen Sie:

```
OPEN 15,8,15 [RETURN]
PRINT#15,"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$
(41)CHR$(73) [RETURN]
CLOSE 15
```

3. Nun können Sie die anderen Geräte einschalten.

Auf eines sollten Sie achten: Da das Laufwerk wieder seine ursprüngliche Gerätenummer annimmt, wenn es aus irgendeinem Grund ausgeschaltet wird, muß dieser Vorgang beim erneuten Einschalten wiederholt werden.

ANHANG A

DIE FEHLERMELDUNGEN DER FLOPPY DISK

NUMMER	MELDUNG
0	Kein Fehler aufgetreten
1	Der SCRATCH-Befehl wurde ausgeführt. Die Zahl in der Spur-Position steht für die Anzahl der gelöschten Files.
2–19	Diese Zahlen kommen nicht vor.
20	Der "Header" (Kopf) des zuletzt angegebenen Sektors kann nicht gefunden werden.
21	Das Laufwerk ist nicht in der Lage, die auf jeder Spur vorhandene Synchron-Markierung zu finden. Passiert dies auch bei anderen Disketten, sollten Sie das Laufwerk justieren lassen.
22	Die Floppy Disk wurde angewiesen, einen unauffindbaren Sektor zu lesen.
23	Die Prüfsumme oder Zahl, die in jedem Sektor gespeichert ist, um einer Störung vorzubeugen, zeigt an, daß Daten gelesen wurden.
24	Eine allgemeine Meldung, die anzeigt, daß Daten unvollständig von der Diskette gelesen wurden.
25	Die auf die Diskette geschriebenen Daten stimmen nicht mit dem überein, was geschrieben werden sollte.
26	Es wurde versucht, auf eine Diskette zu schreiben, die einen Schreibschutz-Aufkleber aufgeklebt hat.
27	Der "Header" (Kopf) eines Sektors stimmt nicht mit der Prüfsumme überein.
28	Das Laufwerk kann den Anfang eines Blocks nicht ermitteln, der auf den gerade geschriebenen folgt. Die Diskette muß normalerweise neu formatiert werden.
29	Die ID der Diskette stimmt nicht mit der ID im Laufwerkspeicher überein. Die Diskette muß, wenn sie ausgewechselt wurde, mit INITIALIZE initialisiert werden. Es könnte auch auf eine fehlerhafte Diskette zurückzuführen sein.
30	Der letzte zum Laufwerk geschickte Befehl kann nicht interpretiert werden. Dies liegt sehr wahrscheinlich an einem ungültigen Format.
31	Der letzte zum Laufwerk geschickte Befehl kann nicht interpretiert werden. Dies liegt sehr wahrscheinlich an einem ungültigen Schlüsselwort.

32	Der letzte zum Laufwerk geschickte Befehl war zu lang.
33	Es wurde ein ungültiger Filename angegeben.
34	Das Laufwerk kann den Filenamen in einem Befehl nicht finden. Kommt vor, wenn z. B. der Doppelpunkt hinter der das Gerät bestimmenden Null vergessen wird.
39	Der letzte zum Laufwerk geschickte Befehl kann nicht interpretiert werden.
50	Das Laufwerk wurde angewiesen, über das Dateiende hinweg Daten zu lesen.
51	Daten, die in eine relative Datei geschrieben werden sollen, sind länger als die angegebene Länge des Datensatzes in der Datei.
52	Es wurde eine Stelle innerhalb einer relativen Datei angegeben, die über die Diskettenkapazität hinausgehen würde.
60	Es wurde versucht, eine Datei zum Lesen zu öffnen, die vorher schon zum Schreiben geöffnet worden ist und nicht (mit CLOSE) geschlossen wurde.
61	Das Laufwerk wurde angewiesen, auf eine Datei zuzugreifen, die nicht (mit OPEN) geöffnet war.
62	Das Laufwerk soll eine Datei finden, die nicht auf der Diskette existiert.
63	Das Laufwerk wurde instruiert, eine Datei mit einem Namen zu erstellen, die schon auf der Diskette existiert.
64	Das Laufwerk soll eine Datei wie die eines anderen Typs behandeln.
65	Der in einem BLOCK-ALLOCATE-Befehl angegebene Sektor ist belegt. Die Spur- und Sektor-Ziffern weisen auf den nächst höheren freien Block. Eine Null an beiden Positionen bedeutet, daß kein höherer Block mehr frei ist.
66	Der in der letzten Operation angegebene Block existiert nicht. Dies ist entweder auf einen Programmierfehler oder auf die Verteilung des auf den nächsten Block einer Datei zeigenden Zeigers zurückzuführen.
67	Zeigt eine ungültige Spur-Sektor-Kombination an.
70	Entweder ist nur der angegebene Kanal *oder* aber es sind sämtliche Kanäle belegt.
71	Die Block Allocation Map (BAM) stimmt nicht mit dem Inhalt der Diskette überein. Die Diskette sollte (mit INITIALIZE) initialisiert werden, obwohl es keine Garantie dafür gibt, daß einige Dateien schon zerstört worden sind.

72 Entweder gibt es schon zu viele Dateien auf der Diskette, so daß keine mehr in das Directory geschrieben werden kann, oder es ist kein Platz mehr auf der Diskette.

73 Es wurde versucht, auf eine Diskette zu schreiben, die mit einem anderen Disk Operating System (DOS) formatiert wurde.

74 Das Laufwerk wurde angesprochen, ohne daß sich eine Diskette im Laufwerk befindet.

ANHANG B

ZUSÄTZLICHE MASCHINENSPRACHBEFEHLE

Zusätzlich zu den in Kapitel 12 erwähnten Hauptbefehlen gibt es noch eine Reihe von sogenannten "U"-Befehlen:

U3 (UC)	Sprung nach $0500
U4 (UD)	Sprung nach $0503
U5 (UE)	Sprung nach $0506
U6 (UF)	Sprung nach $0509
U7 (UG)	Sprung nach $050C
U8 (UH)	Sprung nach $050F
U9 (UI)	Sprung nach $FFFA
U; (UJ)	Einschaltet-Reset
UI+	paßt die Laufwerkgeschwindigkeit an die des C 64 an
UI−	paßt die Laufwerkgeschwindigkeit an die des VC 20 an

Beachten Sie, daß der Hauptbestandteil des Sprungbefehls jeweils drei Bytes in der Adresse beansprucht. Dies, um Ihnen zu ermöglichen, bei $0500 im Diskettenspeicher eine Sprungtabelle anzulegen.

ANHANG C

DIE DOS-5.1-BEFEHLE

Neuere 1541-Laufwerke wurden mit dem überaus brauchbaren Programm "DOS 5.1" ausgeliefert, das es ermöglicht, die Systembefehle im Direktmodus (ohne Zeilennummern) einzugeben. Um das "DOS-5.1"-Programm in den Speicher zu laden, nehmen Sie das C-64-WEDGE-Programm, das auf der mit dem Laufwerk gelieferten Diskette mit Dienstprogrammen zu finden ist. Dieses lädt und startet das Maschinenspracheprogramm, welches das BASIC des C 64 erweitert.
Die Befehle, die über das DOS 5.1 zur Verfügung stehen, lauten:

1. @<Befehlsstring> oder > <Befehlsstring>
2. @ oder >
3. / <Filename>

1. Der erwähnte Befehlsstring ist irgendein gültiger Befehl, der über den vom DOS 5.1 automatisch geöffneten Fehler- oder Befehlskanal geschickt wird. Die Eingabe von "$" oder "$0" als Befehl bewirkt den Ausdruck des Directorys auf dem Bildschirm, ohne daß das momentan im Speicher befindliche Programm gelöscht wird. Das Listen des Directorys kann durch Drücken der SPACE-Taste gestoppt und durch Drücken irgendeiner andern Taste wieder gestartet werden. Das Drücken der RUN/STOP-Taste bricht den LIST-Vorgang ab.

2. Diese Befehlsform liest die vorliegende Fehlermeldung, befreit Sie also davon, den Fehlerkanal öffnen zu müssen, ihn zu lesen und dann wieder zu schließen.

3. Lädt eine Datei, ohne daß LOAD eingegeben werden oder der Dateiname in Anführungszeichen stehen muß. Das Directory kann durch Eingabe von "/$" geladen werden.

Wollen Sie mehrere Laufwerke betreiben und das DOS 5.1 auch auf anderen Geräten als dem mit der Nummer 8 verwenden, müssen Sie es zuerst vom Gerät 8 laden und dann

OPEN15,<NEUE GERÄTENUMMER>,15:CLOSE15:SYS52224

eingeben. Daraufhin wird die Ausgangsmeldung des DOS 5.1 angezeigt und das Programm für die neue Gerätenummer zur Verfügung stehen.

Es ist wichtig, sich daran zu erinnern, daß die DOS-5.1-Befehle *nicht* in ein Programm eingebaut werden können, sondern *nur* im Direktmodus eingegeben werden dürfen.

Dieser Band ist der unverzichtbare Leitfaden für alle, die optimal mit ihrer "Floppy Disk" 1541 arbeiten wollen.

Das Buch gibt eine einfache Einführung in die Grundlagen der Diskettenspeicherung und der Arbeitsweise dieses Diskettenlaufwerks. Es führt den Leser schließlich zum Verständnis auch fortgeschrittener Anwendungstechniken auf der 1541.

Sehr gründlich werden alle unterschiedlichen Arten der Datenspeicherung erklärt, einschließlich des Einsatzes von Programm-Dateien und deren einfaches Speichern und Laden, das Arbeiten mit sequentiellen Dateien, relativen Dateien und Direktzugriffs-Dateien. Andere Kapitel erklären die Benutzung des Fehler-Kanals, die Handhabung vom Directory und den richtigen Zugang zum Betriebssystem der 1541 für ein weites Feld von Anwendungen.

Die "Floppyprogrammierung auf dem Commodore 64" enthält u.a. Programme für schnelle Datenspeicherung, Wiederherstellung von mit dem Scratch-Befehl getilgten Daten, Auslisten von Programmen auf einer Diskette, Renumber- und Merge-Programme und eine Programm-Anzeige des internen Speichers des DOS.

David Lawrence ist einer der erfolgreichsten Computerbuch-Autoren in England. Neben seiner Autorentätigkeit kommentiert er regelmäßig im Rundfunk und schreibt Beiträge für "Popular Computing Weekly". Mark England ist mehrfach als Co-Autor von David Lawrence aufgetreten, u.a. in den Bänden 12 und 13 zur Maschinencode-Programmierung auf dem C 64.

Commodore

Commodore GmbH
Lyoner Straße 38
D-6000 Frankfurt/M. 71

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Commodore GmbH
Kinskygasse 40–44
A-1232 Wien

Nachdruck, auch auszugsweise, nur mit schriftlicher Genehmigung von Commodore.
Artikel-Nr. 556428/7.85 Änderungen vorbehalten. ISBN 3-89133-016-2