

Dateiverwaltung komplett

Wenn der Speicherplatz im Computer nicht mehr reicht, muß man auf das Diskettenlaufwerk ausweichen. Wir zeigen, wie man Daten sinnvoll und effektiv verwalten kann.

Die einfachste Methode, Dateien innerhalb eines Basic-Programmes zu verwalten, ist, den Text (im Fall einer Datei für Texte) einfach in DATA-Zeilen beziehungsweise dimensionierten Stringfeldern abzulegen.

Zunächst wollen wir einmal von einem ganz einfachen Beispiel ausgehen: Wie auch in späteren Kapiteln dieses Kurses stellen wir uns hierzu eine Aufgabe beziehungsweise geben uns eine Datei vor, an der wir anschließend arbeiten werden.

Gehen wir einmal von folgender Datei aus:

```
01 BANANE
02 ORANGE
03 TRAUBE
04 APFEL
05 BIRNE
06 COCA COLA
07 ERDBEERE
08 MANDARINE
09
10 HIMBEERE
11 PFLAUME
12 BIRNE
13 BROMBEERE
```

Zugegeben, dies ist keine besonders glückliche Datei (nicht etwa, weil sie 13 Elemente enthält), sondern vielmehr eine ungeordnete Liste von Früchten. Einige von Ihnen wundern sich bestimmt auch schon, was Amerikas bestverkauftes Getränk in dieser Liste zu suchen hat – dazu jedoch später.

Lassen Sie mich einmal kurz aufzählen, was für eine ideale Datei charakteristisch ist:

1. Die Datei bezieht sich auf genau einen Bereich, beziehungsweise ist in einzelne Gruppen unterteilt. Das heißt, in einer Datei, mit der man zum Beispiel Videokassetten ordnen und verwalten will, haben Elemente wie Stromabrechnung oder Zeitschrifteninhaltsverzeichnisse nichts zu suchen. Genauso ist Coca Cola in unserer Beispielsdatei, mit der wir arbeiten wollen und die sich auf das Verwalten von Früchten bezieht, total fehl am Platz.
2. Eine Datei kann in einzelne Gruppen unterteilt werden. Wenn man zum Beispiel eine Datei zum Verwalten eines Haushaltes eröffnen möchte, kann man sie in die Gruppen Stromabrechnung, Wasserabrechnung, Telefongebühren etc. unterteilen. Diese einzelnen Gruppen stehen dann in festgelegten Bereichen der Haushaltsdatei und dürfen sich nicht überschneiden.
3. Es bietet sich an, die Dateien nach bestimmten Kriterien wie zum Beispiel alphabetisch zu sortieren.

Unsere Beispielsdatei sieht auf den ersten Blick zwar wie eine relativ vernünftige Aufzählung von Früchten aus, ist jedoch noch lange keine ideale Datei in bezug auf die obigen drei Definitionen.

Aufbau einer Datei

Hier einige Betrachtungen zum Aufbau unserer Früchtedatei:
– Die Datei besteht aus 13 SÄTZEN. Unter einem SATZ verstehen wir eine Information (zum Beispiel APFEL in unserer Früchtedatei), die an einer bestimmten Stellen der Datei zu finden ist. Entsprechend unserer Beispielsdatei könnte man also sagen »SATZ Nummer 1 heißt BANANE« oder »BANANE

steht in SATZ 1 der Früchtedatei« beziehungsweise »BANANE steht an erster STELLE in der Früchtedatei«.

– In der Früchtedatei sind drei Beerenarten vertreten – ERDBEERE, HIMBEERE und BROMBEERE.

– Satz 9 der Datei ist leer beziehungsweise an Stelle 9 der Früchtedatei ist eine Lücke.

– In SATZ 6 steht ein Element (COCA COLA), das nicht in diese Datei gehört (da COCA COLA ein Getränk ist und in unserer Früchtedatei nur Früchte vorkommen sollen).

– Die Datei ist weder alphabetisch noch nach einem anderen erkennbaren Kriterium sortiert beziehungsweise geordnet.

Bevor wir mit unserer Datei arbeiten können, müssen wir sie zunächst einmal in den Computer bringen.

Wir legen die »Früchte« hierzu in DATA-Zeilen ab und lesen sie dann in ein dimensioniertes Stringfeld D\$ ein (Listing 1).

Mit der Anweisung GOSUB 10000 wird unsere Früchtedatei definiert und kann dann bearbeitet werden.

```
10000 REM FRUECHTE-DATEI DEFFINIEREN -----
-----
10010 SZ=13      :REM ANZAHL DER SAETZE      <026>
10020 DIM D$(SZ):REM STRING-FELD            <133>
10030 FOR I=1 TO SZ                          <163>
10040   : READ D$(I)                          <073>
10050 NEXT I                                  <228>
10100 DATA BANANE                            <123>
10105 DATA ORANGE                           <023>
10110 DATA TRAUBE                           <132>
10115 DATA APFEL                            <045>
10120 DATA BIRNE                            <009>
10125 DATA COCA COLA                       <060>
10130 DATA ERDBEERE                         <245>
10135 DATA MANDARINE                       <050>
10140 DATA                                     <248>
10145 DATA HIMBEERE                         <172>
10150 DATA PFLAUME                         <066>
10155 DATA BIRNE                           <044>
10160 DATA BROMBEERE                       <226>
10170 RETURN                                <086>
```

Listing 1. Definition der Früchtedatei

```
10 GOSUB 10000: REM DATEI DEFFINIEREN      <103>
100 REM STEUERPROGRAMM -----
-----
110 INPUT"WELCHEN SATZ ";SA                 <020>
120 IF SA<0 OR SA>SZ THEN PRINT"DIESER SATZ <054>
    EXISTIERT NICHT!":GOTO 110              <098>
130 PRINT:PRINT D$(SA)                     <112>
140 GOTO 100                               <068>
```

Listing 2. Lesen eines Datensatzes

```
10 GOSUB 10000: REM DATEI DEFFINIEREN      <103>
100 REM STEUERPROGRAMM -----
-----
110 X$="":INPUT"WELCHEN BEGRIFF SOLL ICH S <020>
    UCHEN ";X$                              <254>
120 FOR SA=1 TO SZ                          <039>
130   : IF X$=D$(SA) THEN PRINT"DER GESUCHTE <008>
        BEGRIFF STEHT IN SATZ ";SA          <068>
140 NEXT SA                                <094>
150 GOTO 110
```

Listing 3. Suche nach einem Datensatz

Erste Gehversuche

Wir wollen nun einige Experimente mit unserer Datei durchführen.

Das Unterprogramm zur Definition ist immer das gleiche. Sie müssen lediglich jeweils die anderen Programmzeilen anpassen.

SATZ lesen

Unser erstes »Experiment« ist so einfach, daß sich jegliche Erklärungen erübrigen. Bitte ergänzen Sie nun um die Zeilen aus Listing 2.

Dieses Steuerprogramm verlangt also die Eingabe von Satzzahlen und gibt daraufhin den entsprechenden Satz aus.

SATZ suchen

Hier geht es darum, einen bestimmten SATZ innerhalb der Datei zu finden. Man gibt als Suchbegriff zum Beispiel ERDBEERE ein und erhält als Antwort SATZ 7 – also der gesuchte Begriff ERDBEERE steht in SATZ 7 der Früchtedatei. Das Steuerprogramm hierfür finden Sie in Listing 3.

Auch dieses Programm ist noch sehr einfach. Nach Eingabe des gesuchten Begriffs als X\$ durchläuft eine Schleife SA die gesamte Datei D\$(1) bis D\$(SZ) – wobei SZ der höchste SATZ der Datei ist – und überprüft bei jedem Schritt, ob X\$ gleich D\$(SA) ist. Wird eine Übereinstimmung gefunden, so erhält man den entsprechenden Satzwert SA.

```

10 GOSUB 10000: REM DATEI DEFFINIEREN      <103>
100 REM STEUERPROGRAMM -----            <020>
-----
110 X$="": INPUT "WELCHEN BEGRIFF SOLL ICH S  <254>
    UCHEN "; X$                               <168>
120 IC=0: FOR SA=1 TO SZ
130 : IF X$=D$(SA) THEN PRINT "DER GESUCHTE
    BEGRIFF STEHT IN SATZ "; SA: IC=IC+1    <101>
140 NEXT SA                                  <068>
142 IF IC=0 THEN PRINT "DIESER BEGRIFF IST
    IN DER FRUECHTEDATEI NICHT ENTHALTEN!" <045>
144 IF IC>1 THEN PRINT "ACHTUNG, DER BEGRIF
    F "; X$; " KOMMT "; IC; "MAL VOR!"      <087>
150 GOTO 110                                <094>

```

Listing 4. Suchen eines Datensatzes

```

10 I$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"      <098>
20 X$="": PRINT {CLR}                      <230>
30 X$=X$+"BEI DER BUCHSTABENFOLGE "       <044>
40 X$=X$+LEFT$(I$,INT(23*RND(1))+1)        <076>
50 X$=X$+" HANDELT ES SICH UM EINEN TEIL D
    ES ALPHABETS."                         <071>
60 PRINT X$                                <166>
70 PRINT "{2DOWN}BITTE EINE TASTE DRUECKEN!
    "                                       <212>
80 POKE 198,0: WAIT 198,1                 <052>
90 GOTO 20                                  <020>

```

Listing 5. Das PRINT-Problem

```

20000 REM FORMATIERTE TEXTAUSGABE -----
-----
20010 IF X$="" THEN 20090                  <251>
20020 IF LEN(X$)<40 THEN PRINT X$: GOTO 200
    90                                       <204>
20030 IF LEN(X$)=40 THEN PRINT X$; GOTO 20
    90                                       <239>
20040 I$=LEFT$(X$,41): IC=0: FOR I=1 TO LEN(
    I$): IF MID$(I$,I,1)=" " THEN IC=I      <125>
20050 NEXT I: IF IC=0 THEN PRINT X$: GOTO 20
    90                                       <247>
20060 I$=LEFT$(I$,IC-1): IF LEN(I$)=40 THEN
    PRINT I$;                               <062>
20070 IF LEN(I$)<40 THEN PRINT I$           <051>
20080 X$=RIGHT$(X$,LEN(X$)-IC): GOTO 20010 <121>
20090 RETURN                               <243>
20090 RETURN                               <080>

```

Listing 6. Formatierte Textausgabe

Im Falle einer idealen Datei würde dieses Steuerprogramm optimal funktionieren. Bei unserer momentan noch etwas chaotischen Datei kann es jedoch noch zu Schwierigkeiten kommen. Betrachten wir einmal folgende Fälle:

- Suchen Sie doch einmal den Begriff »PFIRSICH«. Selbstverständlich werden Sie keine Antwort erhalten, da diese Frucht in unserer Früchtedatei nicht aufgeführt wird. Unser Steuerprogramm kann also insofern verbessert werden, daß es mit »Dieser Begriff ist in der Früchtedatei nicht enthalten!« antwortet. Diese Änderung läßt sich noch einfach vornehmen, aber es gilt auch noch ein anderes Problem zu lösen:
- Geben wir als Suchbegriff »BIRNE« ein, so erhalten wir zwei Antworten. Einmal, daß der Begriff in SATZ 5 steht und einmal in SATZ 12. Diese Antwort des Steuerprogramms ist völlig korrekt, da der Begriff beziehungsweise die Frucht BIRNE in unserer Früchtedatei zweimal auftritt – nämlich in den SÄTZEN 5 und 12.

Allerdings ist es ziemlich unsinnig, bei unserer Früchtedatei eine Frucht zweimal aufzuführen. Warum habe ich die Frucht BIRNE eigentlich zweimal aufgeführt, sollten Sie sich jetzt fragen. Ganz einfach – wenn man sehr lange Listen eintippen muß (zum Beispiel Ersatzteile-Bestandsliste für einen Kfz-Betrieb) kann es sehr leicht passieren, daß man einen Satz versehentlich mehrmals eintippt. Spätestens das Dateiverwaltungs-Programm sollte dann soviel »Intelligenz« (Anführungszeichen deshalb, da dies noch lange nichts mit KI (künstliche Intelligenz) zu tun hat) haben, um den Fehler zu erkennen und auf ihn aufmerksam zu machen. Es gibt selbstverständlich auch Dateiverwaltungen, bei denen es sinnvoll ist, Begriffe doppelt auszuführen (zum Beispiel Objekt- und Wortschatztabelle für Abenteuerspiele) – dazu jedoch später.

Wir wollen unser derzeitiges Steuerprogramm also noch so verbessern, daß es einen Hinweis gibt, wenn ein Begriff mehrmals auftaucht.

Versuchen Sie zur Übung einmal diese beiden Verbesserungen am Steuerprogramm selbst durchzuführen, bevor Sie die Lösung abtippen. Die Lösung finden Sie in Listing 4.

Dieses Steuerprogramm ist schon relativ brauchbar. Es kann Begriffe innerhalb unserer Datei suchen, erkennt, wenn ein gesuchter Begriff in der Datei überhaupt nicht vorkommt und es teilt uns mit, wenn ein Begriff in der Datei mehrmals enthalten ist (zum Beispiel BIRNE).

Bisher haben wir bei unseren Beispielprogrammen noch keinen Wert auf eine schöne Darstellung beziehungsweise Formatierung der Textausgaben auf dem Bildschirm geachtet. Im Laufe des Kurses wollen wir uns jedoch auch diesem Problem zuwenden.

Wenn man bei unserer Früchtedatei den Begriff BIRNE sucht, so erhalten wir die Antwort »Achtung, der Begriff kommt 2mal vor!«.

Auf dem Monitor sieht dieser Satz allerdings nicht sehr gut dargestellt aus, wenn man einmal das Wort »vor« am Ende des Satzes betrachtet. Das »v« des Wörtchens steht am Ende einer Bildschirmzeile, während das »or« zu Beginn der nächsten Zeile steht.

Viel deutlicher noch wird das Problem einer formatierten Textausgabe im Beispiel von Listing 5.

Dieses Programm macht nichts anderes, als einfach einen zufällig erzeugten Text auf dem Bildschirm auszugeben. Das heißt, Anfang und Ende des Textes sind immer gleich, der mittlere Teil stellt jedoch eine zufällige, aus dem Alphabet ausgewählte Buchstabenfolge dar. Durch die zufällige Auswahl des mittleren Textteils variiert die Gesamtlänge des Textes stark – es kommt so zu einer unkontrollierten, unformatierten Bildschirmausgabe. Einige Computer haben in ihrem Basic PRINT-Anweisungen mit »Intelligent Space« zur Verfügung, mit deren Hilfe es sehr einfach ist, Text formatiert auf den Bildschirm zu bringen. Eine derartige Routine ist im Basic des

Commodore 64 leider nicht enthalten. Sie läßt sich jedoch mit nur geringem Aufwand als Basic-Unterprogramm programmieren, wie es Listing 6 zeigt.

Nun wollen wir dieses Unterprogramm in unser letztes Beispielsprogramm einbauen. Hierzu müssen lediglich folgende Änderungen durchgeführt werden:

60 GOSUB 20000

Wenn Sie das Programm nun erneut starten, werden alle Textausgaben formatiert auf den Bildschirm gebracht. Was die formatierte Textausgabe anbelangt, so wollen wir es zunächst bei diesem Unterprogramm belassen.

Eine MERGE-Routine

Wie Sie sicherlich bereits bemerkt haben, kommen in diesem Kurs Routinen vor, auf die wir ständig zurückgreifen werden. Um sich viel Tipparbeit zu ersparen, bietet es sich daher an, die einzelnen Routinen einzeln abzuspeichern, um sie bei Bedarf wieder laden zu können. Hierbei ist es nützlich, wenn Sie mit einem MERGE-Befehl arbeiten, der es erlaubt, einzelne Programmteile nacheinander zusammen »zu laden«. Sie können hierzu Basic-Erweiterungen à la G-Basic, Simons Basic oder auch Toolkits à la KFC Super 64 verwenden. Mit fortlaufendem Kurs werden wir jedoch soviel Speicherplatz benötigen beziehungsweise das Betriebssystem des C64 derartig »umkrepeln« und erweitern, daß es sich mit keiner dieser Basic-Erweiterungen mehr verträgt. Es bietet sich sodann die folgende MERGE-Routine, die Sie im Direktmodus durchführen können, an (sowohl für Disklaufwerk als auch Datensette).

Dieser Direktmodus-MERGE geht so:

- Programm A laden. Dann notieren Sie sich den Inhalt der Speicherzellen 43 und 44 – also PEEK(43) und PEEK(44).
- Geben Sie nun im Direktmodus ein:
POKE 43,(PEEK(45)+256*PEEK(46)-2)AND255
(RETURN)
POKE 44,(PEEK(45)+256*PEEK(46)-2/256)
(RETURN)
- Programm B laden. Dann die notierten Werte wieder in die Speicherzellen 43 und 44 POKEn.

Fertig – beide Programme sind jetzt gemeinsam im Speicher.

Bei diesem Vorgang ist zu beachten, daß zuerst das Programm mit den niedrigeren Zeilennummern geladen wird (Programm A) – ansonsten kann es zu Schwierigkeiten kommen beziehungsweise das MERGEEn klappt einfach nicht.

Wir werden deshalb schon sehr bald auf die beste Dateiverwaltungsmethode zu sprechen kommen, die auf dem C64 überhaupt möglich ist: Relative Dateien mit Hilfe des Diskettenlaufwerks 1541.

Würde man auf Disk eine Datei mit 2000 Sätzen anlegen und in dieser dann einen Begriff suchen, der an letzter Stelle der Datei steht, so würde man bei Verwenden unserer bisherigen Suchroutine eine viel größere Suchzeit als 14 Sekunden in Kauf nehmen müssen. Auch eine Programmierung in Assembler würde keine befriedigenden Suchzeiten ergeben, da bei relativen Dateien auf Disk noch ein anderer Faktor eine wesentliche Rolle spielt: Die Zugriffszeit des Floppy-

```

10 DIM X$(2000) <005>
20 X$(2000)="MIKE" <114>
30 REM SUCHROUTINE <014>
35 TI$="000000":REM STOPPUHR AUF NULL <110>
40 FOR I=1 TO 2000 <091>
50 :IF X$(I)="MIKE" THEN PRINT"MIKE STEHT I
N SATZ";I:PRINT"BENÖTIGTE ZEIT:";TI$ <165>
60 NEXT I <144>

```

Listing 7. Suchvorgang gestoppt

Laufwerks – also die Zeit, die das Disketten-Laufwerk braucht, um auf bestimmte Sätze einer relativen Datei zuzugreifen. Diese Zugriffszeit, die vom Floppy-Laufwerk beansprucht wird, läßt sich auch mit Assembler nicht beeinflussen. Auch Hypra-Load-Fans werden enttäuscht sein, da Hypra-Load wie auch alle anderen ähnlichen Programme (Ausnahme: Parallelübertragung à la SpeedDos etc.) die Zugriffszeit auf relative Dateien in keinsten Weise beeinflussen. Hier auch gleich eine Anmerkung für alle, die ihr altes Betriebssystem durch ein Betriebssystem mit Hypra-Load ersetzt haben (beziehungsweise Umschaltplatine im Computer haben): Hypra-Load beansprucht den Datensetten-Puffer von 828-1019. Diesen Bereich werde ich ab und zu für kleine MP-Routinen beanspruchen. Es ist daher unbedingt erforderlich, daß Sie dann das alte Betriebssystem im Computer haben beziehungsweise auf dieses umschalten!

Nun aber zurück zum Sortieren.

Bevor wir unsere erste Sortierroutine programmieren, müssen wir uns noch einiges an Theorie zu Gemüte führen.

Betrachten wir hierzu noch einmal unsere Früchtedatei. Bisher haben wir immer nur einzelne Begriffe gesucht. Was aber nun, wenn wir aus unserer Früchtedatei alle Früchte herausuchen wollen, die einer Beersorte angehören? Wir müßten als Antwort dann ERDBEERE, HIMBEERE und BROMBEERE erhalten. Woher in aller Welt soll unser C64 aber wissen, was Beeren sind beziehungsweise welche Früchte als Beeren bezeichnet werden können?

Die einfachste Lösung wäre die folgende:

Alle Beerenarten in unserer Datei haben die Endung BEERE gemeinsam. Es liegt nun nahe, beim Suchvorgang einfach zu überprüfen, ob die letzten 5 Zeichen jedes Früchtenamens gleich »BEERE« ist; also:

```
IF RIGHT$(D$,5)=X$ THEN
```

wobei X\$="BEERE" ist.

Um die Suchroutine (Listing 4) entsprechend zu ändern, müssen Sie lediglich Zeile 130 modifizieren;

```
130 : IF RIGHT$(D$(SA),LEN(X$))=X$ THEN PRINT D$(SA):
IC=IC+1
```

Die erste Hürde bei der Einführung von Dateiverwaltungsprogrammierung wäre nun geschafft. Sie wissen jetzt, wie man einfache Dateien innerhalb von Basic tabellenartig verwaltet, und Sie haben die einfachste und schlechteste Suchroutine kennengelernt, die es gibt.

Ohne Sortieren geht es nicht

Bisher haben wir unsere Früchtedatei von Anfang bis Ende durchsucht, um schließlich eine bestimmte Frucht zu finden oder zu erfahren, daß sie in der Datei gar nicht beziehungsweise mehrmals auftritt. Das ging bisher auch ziemlich

```

30000 REM QUICKSORT -----
-----
30001 DIM LG(100),RG(100):Z=0:LG(1)=1:RG(1)=SZ <087>
1)=SZ <177>
30010 Z=Z+1:IF LG(Z)>=RG(Z) THEN 30120 <166>
30020 X=LG(Z):Y=RG(Z) <011>
30030 VG=D$(INT((X+Y)/2)) <188>
30040 IF X>Y THEN 30100 <205>
30050 IF D$(X)<VG THEN X=X+1:GOTO 30050 <214>
30060 IF D$(Y)>VG THEN Y=Y-1:GOTO 30060 <064>
30070 IF X>Y THEN 30100 <235>
30080 S=D$(X):D$(X)=D$(Y):D$(Y)=S <130>
30090 X=X+1:Y=Y-1:GOTO 30040 <099>
30100 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 30010 <184>
30110 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 30010 <104>
30120 Z=Z-1:RETURN <161>

```

Listing 8. Der legendäre Quicksort

schnell, in Bruchteilen von Sekunden. Was aber nun, wenn wir riesige Dateien mit mehreren tausend Sätzen verwalten wollen?

Die »Schwäche« unserer bisherigen Suchroutine wird bei dem Demo-Programm in Listing 7 besonders deutlich:

Hier wird in einer Datei mit 2000 Sätzen der Name »Mike« gesucht, der unglücklicherweise an letzter Stelle der Datei steht. Das Programm sucht 14 Sekunden, bis es den Namen endlich gefunden hat. Ein kleiner Tip am Rande: SYS 65499 bewirkt das gleiche wie der Befehl TIS= "000000".

Diese Zeit ist sogar noch akzeptabel, wenn man bedenkt, daß sich auf unsere Früchtedatei keine andere vernünftige beziehungsweise schnellere Suchroutine als die obig vorgestellte anwenden läßt.

Warum? Ganz einfach – vernünftige Suchroutinen lassen sich nur bei Dateien anwenden, die nach einem bestimmten Kriterium sortiert sind (zum Beispiel nach der alphabetischen Grundordnung).

Wir müssen unsere Früchtedatei also zunächst sortieren, bevor wir eine bessere Suchroutine programmieren können.

»Warum programmieren wir die obige Suchroutine nicht einfach in Maschinensprache?« könnte jetzt einer rufen! Er hätte auch recht – die obige Routine wäre in Assembler derartig schnell, daß wir uns über die Programmierung von Suchroutinen keine Gedanken machen müßten.

Es muß also noch andere Gründe geben, warum ich das Zeitproblem nicht einfach mit einer Maschinensprache-Routine unter den Tisch kehre und stattdessen einen Umweg über die Programmierung von Sortieralgorithmen mache. Und der ist wie folgt: Sie müssen sich lediglich vor Augen halten, daß wir es im Moment nur mit einfachen Dateien innerhalb von Basic zu tun haben. Diese Dateien sind zwar sehr gut zu handhaben, aber ihre Verwaltungskapazität ist zu stark eingeschränkt – nämlich auf 38 KByte Basic-RAM beim C64.

Wenn Sie das Programm nun starten und als Suchbegriff »BEEREN« eingeben, so gibt die Suchroutine alle Sätze aus, die mit »BEEREN« enden – also alle Beerensorten. Genauso gut können Sie als Suchbegriff »COLA« eingeben und erhalten dann alle in der Datei vorkommenden Cola-Sorten aufgelistet. Diese Suchmethode funktioniert im Moment zwar ganz gut, ist jedoch viel zu sehr eingeschränkt, da sich Dateien nur in ganz seltenen Fällen nach ihren Satzendungen sortieren lassen. Angenommen, Sie möchten eine Datei programmieren, in der Computerprogramme verwaltet werden (zum Beispiel für ein Softwarehaus). Man müßte dann nach verschiedenen Kategorien einteilen: Zum Beispiel Programmart, Preislage, Computertyp etc.

Ein solches Programm für einen Software-Händler müßte folgende Bedingungen aufweisen:

1. Feststellen, ob ein bestimmtes Programm geführt wird.
2. Datei nach bestimmten Kriterien sortieren:
 - Programmart
 - Preislage etc.
3. Lagerbestand kontrollieren – also wieviele Exemplare eines Programmes noch vorrätig sind etc.

Man müßte die Dateiverwaltung in Form einer Tabelle aufbauen, die so aussehen könnte:

1. Schlüssel Programmname	2. Schlüssel Art	3. Schlüssel Preis	4. Schlüssel Vorrat
vizawrite	1	200.00	24
wordpro	1	1155.50	13
space invaders	2	39.00	67
jumpman	2	249.00	34
simons basic	3	99.00	65
g-basic	3	3140.00	45
gordon saga	4	39.00	89
deadline	4	99.00	34

Die einzelnen Spalten beziehungsweise Kategorien einer solchen Tabelle/Datei bezeichnet man als Schlüssel. Wenn ich die Software-Datei zum Beispiel nach dem 3. Schlüssel sortiere, so sortiere ich nach dem Kriterium Preis. Beim 1. Schlüssel – dem Programmnamen – würde ich alphabetisch sortieren.

Bevor man nun zur Programmierung übergeht, muß man sich noch einige andere Gedanken über den Aufbau der Tabelle machen. Man muß so zum Beispiel genau festlegen, aus wieviel Zeichen der Programmname maximal bestehen darf. Auch für alle anderen »Schlüssel« muß genau festgelegt werden, welche Länge sie einnehmen dürfen. Betrachten wir einmal den zweiten Schlüssel: Um zwischen 9 Programmarten unterscheiden zu können, benötige ich ein Zeichen Platz. Lege ich jedoch 2 Zeichen als maximale Länge fest, so kann ich bis zu 99 (also zweistellige Zahlen) Arten unterscheiden. Unsinnig wäre es sicherlich, 4 Stellen zu reservieren, da man wohl kaum seine Programme in 9999 verschiedene Arten einteilen will. Auch wenn ich momentan nur 9 Programmarten unterscheide (also nur 1 Zeichen Platz benötige), entscheide ich mich für 2 Zeichen Reservierung, damit meine Datei möglichst ausbaufähig bleibt.

Für den 2. Schlüssel muß ich noch eine weitere Tabelle anlegen:

- 1 = Textverarbeitungsprogramm
- 2 = Actionsspiel
- 3 = Basic-Erweiterung
- 4 = Abenteuerspiel

Nach einigen Überlegungen lege ich schließlich folgendes fest:

- Programmnamenlänge : max. 15 Zeichen
 2. Schlüssel : max. 2 Zeichen
 3. Schlüssel : max. 6 Zeichen
 4. Schlüssel : max. 2 Zeichen

Die Tabelle, wie sie später im Computer sein soll, ist damit festgelegt wie in Bild 1.

1. SCHLÜSSEL														2. SCHLÜSSEL				3. SCHLÜSSEL				4. SCHLÜSSEL			
V	I	Z	A	W	R	I	T	E						0	1	2	0	0	.	0	0	2	4		
W	O	R	D	P	R	O								0	1	1	S	S	.	S	0	1	3		
S	P	A	C	E		I	N	V	A	D	E	R	S	0	2		3	9	.	0	0	6	7		
J	U	M	P	M	A	N								0	2		4	9	.	0	0	3	4		
S	I	M	O	N	S		B	A	S	I	C			0	3		9	9	.	0	0	6	5		
G	-	B	A	S	I	C								0	3	1	4	0	.	0	0	4	5		
G	O	R	D	O	N		S	A	G	A				0	4		3	9	.	0	0	8	9		
D	E	A	D	L	I	N	E							0	4		9	9	.	0	0	3	4		

Bild 1. Aufbau der Software-Datei

Lassen Sie mich noch einmal kurz zu unserer Früchtedatei zurückkehren. Diese Datei läßt sich besonders einfach sortieren, da sie aus nur einem Schlüssel besteht (dem Frucht-namen).

Zum Sortieren braucht man einen möglichst guten Sortier-Algorithmus – am besten in Form eines Basic-Unterprogramms, das Sie später einfach in Ihrem eigenen Programm einbauen können.

Ich halte es nicht für sinnvoll, in diesem Kurs lange und breit auf die Funktion der einzelnen Sortieralgorithmen einzugehen beziehungsweise diese vorzustellen. Der schnellste Sortieralgorithmus ist der Quicksort. Ihn wollen wir immer dann im Kurs anwenden, wenn irgendetwas sortiert werden muß. Wer sich für die Grundlagen der Sortieralgorithmen interessiert, den möchte ich auf den Kurs »Effektives Programmieren« hinweisen, der im Jahre 1985 in der Zeitschrift 64'er veröffentlicht worden ist. Dort erfahren Sie alles Wis-

senswerte über die Programmierung und Funktion dieser Algorithmen.

Listing 8 zeigt das Unterprogramm zum Quicksort-Algorithmus.

Der Quicksort ist wie bereits gesagt der schnellste, aber auch der am schwierigsten verständliche Algorithmus zum Sortieren. Für den Verlauf dieses Kurses ist es lediglich notwendig, daß Sie wissen, wie das Quicksort-Unterprogramm angewendet wird beziehungsweise welche Parameter man ihm übergeben muß.

Beachten Sie bitte die beiden DIMs in Zeile 30001. Wenn Sie innerhalb eines Programmes den Quicksort mehrmals verwenden wollen, also das Unterprogramm mit GOSUB30000 aufrufen, müssen Sie diese beiden DIMs an einer anderen Stelle im Programm (am besten am Anfang) unterbringen, daß es nicht zu einem REDIM'D ARRAY ERROR kommt.

Das Unterprogramm benötigt folgende Parameter:

- das Feld, das sortiert werden soll, muß D\$(1)-D\$(?) sein. Sie können im Unterprogramm natürlich auch D\$ durch etwas anderes ersetzen!

- In der Variablen SZ steht die Anzahl der Sätze der Datei/des Feldes.

- LG(1) und RG(1) geben Anfang und Ende des Bereiches des Feldes, das sortiert werden soll, an.

Ist LG(1)=1 und RG(1)=SZ, so heißt das, daß das gesamte Feld D\$ sortiert werden soll. Ist LG(1)=10 und RG(1)=40, so wird vom 10ten bis zum 40sten Satz sortiert etc.

Das Beispielprogramm in Listing 9 zeigt, wie sich unsere Früchtedatei mit Hilfe des Quicksort-Unterprogramms alphabetisch sortieren läßt.

An diesem Programm braucht nicht viel erklärt werden - einfache Abtippen und RUN.

Interessant ist vielleicht, was mit unserem Satz Nummer 9 der Früchtedatei passiert ist (der leere Satz). Er steht nach dem Sortiervorgang an erster Stelle der Datei. Wenn Sie möchten, dann können Sie leere Sätze als eine Art NOPs betrachten - einfache Platzhalter, in die man später noch etwas einbauen kann. Bei Assembler-Programmierung sind solche Platzhalter unentbehrlich, bei Dateiverwaltung jedoch relativ überflüssig, da gute Dateiverwaltungen Befehle enthalten, innerhalb einer Datei automatisch zwischen zwei Sätzen einen Leerraum zu schaffen. Vor dem Sortieren bietet es sich folglich an, die leeren Sätze aus der Datei zu eliminieren.

Man durchläuft die gesamte Datei hierbei mit einer Schleife und schreibt immer dann, wenn man auf einen leeren Satz findet, den letzten Satz dieser Datei in den leeren Satz. Danach wird SZ, die Variable, die angibt, wieviele Sätze die Datei enthält, um den Wert 1 erniedrigt.

Das geht so wie in Listing 10.

Sie müssen zum Aufruf des Unterprogramms lediglich Listing 9 um folgende Zeile ergänzen:

```
11 GOSUB20000:REM LEERE SAETZE WEG
```

Nun werden vor dem Sortiervorgang alle leeren Sätze aus der Datei entfernt. Wir sind nun in der Lage, jede (sich komplett im Computer befindliche) Datei zu sortieren. Daher eröffnen sich uns auch völlig neue Suchroutinen:

Die binäre und die index-sequentielle Suchmethode.

Für unsere folgenden Experimente gibt die Früchtedatei nicht mehr genug her. Wir wenden uns deshalb nun der Software-Datei zu, die über mehrere Schlüssel verfügt.

Zunächst wird die Datei wieder in einem Unterprogramm ab 10000 definiert, wie es Listing 11 zeigt.

Eigentlich haben wir unsere Dateien bisher immer doppelt im Speicher stehen: einmal innerhalb von DATA-Zeilen und zum anderen in einem dimensionierten Stringfeld. Wir gehen bisher also ziemlich verschwenderisch mit dem geringen Speicher unseres C64 um (gering in Anbetracht der Tatsache, daß es heute schon Computer zu Homecomputer-

```
10 GOSUB 10000:REM TABELLE DEFFINIEREN <247>
20 PRINT<CLR>TABELLE VOR DEM SORTIEREN: <042>
30 FOR I=1 TO SZ <235>
40 :PRINT I,D$(I) <118>
50 NEXT I <134>
60 PRINT<DOWN>[RETURN]":POKE 198,0:WAIT 1
  98,1 <220>
65 PRINT"BITTE WARTEN!" <000>
70 GOSUB 30000:REM D$(1)-D$(SZ) SORTIEREN <009>
80 PRINT<CLR>TABELLE NACH DEM SORTIEREN: <218>
90 FOR I=1 TO SZ <039>
100 :PRINT I,D$(I) <178>
110 NEXT I <194>
120 END <122>
```

Listing 9. Steuerprogramm zu Quicksort

```
20000 REM LEERE SAETZE AUS EINER DATEI ELI
      MINIEREN ----- <088>
-- <039>
20020 I=0: REM ZAEHLER AUF NULL <163>
20030 I=I+1: IF I=SZ THEN RETURN <073>
20040 : IF D$(I)="" THEN D$(I)=D$(SZ):SZ=SZ- <222>
      1:GOTO 40 <146>
20050 GOTO 20030
```

Listing 10. Aufräumen einer Datei

```
10000 REM FRUECHTE-DATEI DEFFINIEREN -----
-- <026>
-- <038>
10010 SZ=8 :REM ANZAHL DER SAETZE <163>
10020 DIM D$(SZ):REM STRING-FELD <073>
10030 FOR I=1 TO SZ <073>
10040 : READ D$(I) <228>
10050 NEXT I <086>
10100 DATA"VIZAWRITE{6SPACE}01200.0024" <036>
10105 DATA"WORDPRO{8SPACE}01155.5013" <175>
10110 DATA"SPACE INVADERS 02 39.0067" <111>
10115 DATA"JUMPMAN{8SPACE}02 49.0034" <230>
10120 DATA"SIMONS BASIC{3SPACE}03 99.0065" <186>
10125 DATA"G-BASIC{8SPACE}03140.0045" <093>
10130 DATA"GORDON SAGA{4SPACE}04 39.0089" <132>
10135 DATA"DEADLINE{7SPACE}04 99.0034" <036>
10140 RETURN
```

Listing 11. Dateidefinition einmal anders

```
10 REM D$(1)-D$(SZ) ALS SEQ-FILE ----- <061>
-- <001>
20 GOSUB 10000: REM TABELLE DEFFINIEREN <242>
30 OPEN 2,8,2,"DATEN-FILE,S,W" <245>
40 FOR I=1 TO SZ <007>
50 :PRINT#2,D$(I) <144>
60 NEXT I <084>
65 CLOSE 2 <079>
70 PRINT"*** FERTIG ***" <082>
80 END
```

Listing 12. So speichert man die Datei auf Disk

```
10000 REM FRUECHTE-DATEI DEFFINIEREN -----
-- <026>
-- <038>
10010 SZ=8 :REM ANZAHL DER SAETZE <163>
10020 DIM D$(SZ):REM STRING-FELD <206>
10030 OPEN 2,8,2,"DATEN-FILE,S,R" <083>
10040 FOR I=1 TO SZ <197>
10050 :INPUT#2,D$(I) <238>
10060 NEXT I <183>
10070 CLOSE 2 <232>
10080 RETURN
```

Listing 13. Lesen einer Datei von Disk

Preisen gibt, die auf 512 KByte RAM aufgestockt werden können!). Es wird also höchste Zeit, sich um eine bessere Unterbringung unserer Tabellen zu bemühen.

In der Tat - sind die DATA-Zeilen erst einmal in ein Stringfeld eingelesen, werden sie nicht mehr weiter benötigt und blockieren lediglich freien Speicherplatz.

Oh, soeben habe ich einen kleinen Fehler in Listing 11 entdeckt: Die REM-Zeile 10000 darf nicht länger Früchte-Datei heißen, sondern muß jetzt Software-Datei lauten. Wir müssen

Zeile 10000 deshalb folgendermaßen abändern:

10000 REM SOFTWARE-DATEI DEFINIEREN

Vielleicht werden sich einige (mit gutem Recht) wundern, warum ich hier lange und breit auf diesen lächerlichen »Fehler« eingehe und nicht einfach mein Manuskript verbessere..... kein Kommentar, denn jetzt kommt wieder ein kleiner (aber äußerst interessanter) Tip am Rande:

Ärger beim »Replace«'n

Selbstverständlich gehe ich jetzt davon aus, daß Sie alle Listings, die Sie im Rahmen dieses Kurses abtippen, auf Diskette speichern, um diese später wieder verwenden zu können. Nehmen wir einmal an, daß Sie auch das falsche Listing 11 bereits gespeichert haben. Worauf ich hinaus will, ist das folgende: Wenn Sie das »falsche« Listing bereits gespeichert haben, so werden Sie es sicherlich verbessern und anschließend mit dem Befehl

SAVE"@:LISTING 11",8

wieder speichern. Wenn Ihnen nicht klar ist, was das »@« im Filename soll, so sind Sie noch nicht »reif« genug für den folgenden Tip – lesen Sie schleunigst Ihr Bedienerhandbuch zur VC 1541!

Wer die Anweisung kennt und schon des öfteren angewandt hat, hat sicherlich auch schon unerklärbare Probleme damit gehabt. So kann es zum Beispiel vorkommen, daß der Befehl einfach nicht funktioniert und die rote LED am Laufwerk ähnlich einer extrem miserablen 1-Kanal-Lichtorgel im ¼-Takt zu blinken beginnt. Die Erklärung dieses Fehlers im Betriebssystem ist sehr kompliziert. Für den Anwender ist nur eines wichtig: Niemals den @ verwenden! Sicherer ist es, mit »Scratch« zu löschen und dann noch einmal zu speichern. Also: Niemals SAVE " @..."

Nun aber schnell zurück zu unserem aktuellen Problem. Laden Sie bitte Listing 11, falls Sie es nicht mehr im Speicher haben. Nach Eingabe von CLR und PRINT FRE(0)+65000 erhalten Sie die Antwort, daß 37928 Byte Basic RAM frei sind. Nach Starten des Programmes mit GOSUB 10000 im Direktmodus und Eingabe von PRINT FRE(0) erhalten Sie die Antwort, daß jetzt nur noch 37880 Bytes Basic RAM frei sind. Es wird also Basic-RAM benötigt, um ein Stringfeld zu füllen. Es macht bei unserem Programm zwar nur ein paar Bytes aus, aber unsere Tabelle ist ja extrem klein. Bei großen Dateien kann ein sehr großer Unterschied vorkommen.

Es bieten sich nun mehr Lösungsmöglichkeiten an.

Man kann so zum Beispiel darauf verzichten, die DATAs in ein Stringfeld einzulesen. Man benötigt dann eine modifizierte RESTORE-Routine. Im Basic des C64 wird mittels RESTORE der DATA-Zeiger einfach auf die erste DATA-Zeile zurückgesetzt. Unter modifizierte RESTORE-Funktion versteht man nun die Möglichkeit, den DATA-Zeiger auf eine bestimmte DATA-Zeile zu setzen – RESTORE(X), wobei X für die DATA-Zeile steht, auf die der Zeiger gesetzt werden soll. Von dieser Möglichkeit, die sich hervorragend zum Programmieren von Adventures anbietet, die komplett im Basic-RAM stehen (worauf Datasetten-Benutzer angewiesen sind), habe ich in meinem ersten Adventure-Programmierkurs im zweiten 64'er Sonderheft regen Gebrauch gemacht. Das entsprechende Maschinenprogramm für die modifizierte RESTORE-Routine finden Sie in diesem Heft.

Für Dateiverwaltung, KI-Programme und disketten-orientierte Adventures ist die RESTORE-Methode jedoch absolut ungeeignet. Die Tabelle kann innerhalb von Basic nicht ausgebaut werden – und gerade dies ist bei Dateiverwaltung und selbstlernenden KI-Programmen unvermeidlich (schon gut, es gibt Basic-Zeilengeneratoren, aber wir wollen uns mit effektvolleren Methoden vertraut machen!).

Eine wesentlich effektvollere Methode besteht darin, die DATA-Zeilen durch sequentielle Dateien zu ersetzen.

Wir ergänzen deshalb Listing 11 nun um ein Steuerprogramm, das D\$(1) bis D\$(SZ) als sequentielle Datei auf Disk ablegt (Listing 12).

Die Software-Datei ist jetzt unter dem Filenamen »Daten-File« als SEQ-File auf Diskette gespeichert. Unter einem sequentiellen File versteht man eine Datenkette, die auf Diskette gespeichert ist – also ein String beziehungsweise Element nach dem anderen. Eine so gespeicherte Tabelle kann ganz einfach wieder in den Computer in ein dimensioniertes Feld geladen werden (Listing 13).

Die hier vorgestellten Beispielprogramme sollen uns zunächst genügen. Es gibt noch einige weitere, wesentlich bessere Methoden, Dateien und Tabellen als sequentielle Files zu speichern. Tatsächlich ist es auch nicht besonders sinnvoll, jeden einzelnen Satz einer Tabelle mit mehreren Schlüsseln als einfachen String zu speichern. Wie ich Ihnen bereits in der Einleitung zum Kurs mitteilte, möchte ich das Thema Dateiverwaltung speziell auf die Bereiche KI und fortgeschrittene Adventure-Programmierung beziehen. SEQ-Files sind auch mit der Datasette möglich. Beim Programmieren von Adventures und KI sind sie jedoch nur äußerst begrenzt sinnvoll einsetzbar. Wie es der Name schon sagt, werden sequentielle Dateien kettenförmig gespeichert. Es ist somit nicht möglich, einen bestimmten Teil der Datei so mir nichts, dir nichts zu verändern beziehungsweise in der Mitte der Datei eine Lücke zu schaffen, um nachträglich etwas einzutragen. Zur Bearbeitung muß eine SEQ-Datei immer komplett in den Computer geladen werden. Dann, nach Beendigung der Arbeit, wird sie wieder auf Diskette gespeichert. Eine sequentielle Datei läßt sich nur dann effektiv modifizieren, wenn sie so klein (relativ betrachtet) ist, daß sie auf einmal in den Speicher des C64 paßt. Dadurch ist die maximale Kapazität einer solchen Datei sehr eingeschränkt.

Für kleinere Dateien sind sequentielle Dateien jedoch empfehlenswert, da sie sehr einfach zu handhaben sind. Für die späteren KI-Programme und Adventures werden SEQ-Files nur eine kleine Nebenrolle spielen. Ich möchte daher nicht auf alle Möglichkeiten der sequentiellen Datenspeicherung eingehen, zumal zu diesem Thema auch genügend geschrieben worden ist.

Jetzt wird es aber höchste Zeit, einige weitere Experimente zum Sortieren und Suchen an unserer neuen Datei durchzuführen. Als Ausgangsprogramm verwenden Sie bitte immer Listing 11. Für unsere Beispiele lohnt es sich nicht, die Datei jedesmal als SEQ-file laden zu müssen.

Sortieren mit dem Quicksort

Wie wir anhand des Demoprogramms zur Suchzeit (Listing 7) festgestellt haben, ist unsere erste Suchmethode – Datei vom ersten bis zum Zielelement durchsuchen – viel zu langsam, wenn eine große Datei vorliegt, und das gesuchte Element ziemlich am Ende der Datei steht. Wir waren uns auch einig, daß bessere Suchmethoden nur dann sinnvoll sind, wenn die Datei alphabetisch beziehungsweise nach einem anderen Kriterium sortiert ist. Besonders deutlich kommen die verschiedenen Sortierkriterien anhand einer Personen-datei zum Vorschein. Wenn ich zum Beispiel wissen will, ob eine bestimmte Person in der Datei aufgeführt ist, so muß ich die Datei alphabetisch ordnen (Namen), um ein schnelles Suchprogramm schreiben zu können. Will ich jedoch wissen, wer an einem bestimmten Tag alles Geburtstag hat, so muß ich die Datei nach den Geburtstagen sortieren. Das Sortieren von Geburtstagen ist schon etwas komplizierter als das Sortieren nach dem Alphabet.

Beispiel:

Tobi	21.11.66
Borgsi	30.05.67
Olaf	20.03.67

Als String lassen sich die Geburtstage nicht so ohne weiteres sortieren. Ein kleiner Trick hilft jedoch:

Tobi 661121
Borgsi 670530
Olaf 670320

Die Geburtstage werden durch einfaches »Umdrehen« in sechsstellige Zahlen verwandelt. Die höchste Zahl kommt jetzt dem jüngsten der drei zu, die niedrigste dem ältesten.

Jetzt können die »Geburtstage« einfach als Strings mit dem Quicksort oder einem anderen Sortieralgorithmus sortiert werden.

Zurück zur Software-Datei. Wie wir bereits festgestellt haben, sind verbesserte Suchmethoden nur dann möglich, wenn die Datei irgendwie (nach irgendeinem Schlüssel) sortiert ist. Bei unserer Software-Datei bietet sich der 1. Schlüssel – also der Programmname – an. Listing 14 zeigt, wie die Datei mit dem Quicksort alphabetisch geordnet wird (1. Schlüssel).

Bitte beachten Sie unbedingt die Änderungen des Quicksort-Unterprogramms in den Zeilen 30030, 30050 und 30060. Diese Zeilen müssen immer dann modifiziert werden, wenn nicht ein ganzer String, sondern nur ein Teilbereich des Strings als Sortierargument gelten soll. Beim 1. Schlüssel unserer Software-Datei sind dies die jeweils ersten 15 Zeichen vom Gesamtstring.

Wir stellen uns nun folgende Aufgabe: Nach Eingabe des Programmnamens gibt das Programm Art, Preis und Lagerbestand zu dem Programm aus.

Eine einfache Lösung wäre es wieder, die ganze Datei nach dem Programmnamen zu durchsuchen. Da unsere Programmnamen jedoch alphabetisch geordnet sind, können wir mit einer anderen Suchmethode arbeiten, mit der binären Suchmethode.

Die binäre Suchmethode

Betrachten wir einmal den geordneten Schlüssel der Software-Datei:

- 1 DEADLINE
- 2 G-BASIC
- 3 GORDON SAGA
- 4 JUMPMAN
- 5 SIMONS BASIC
- 6 SPACE INVADERS
- 7 VIZAWRITE
- 8 WORDPRO

Wir wollen nun das Programm GORDON SAGA suchen, um festzustellen, in welchem Satz (Satznummer) es steht.

Wir setzen zunächst einen Zeiger auf den mittleren Satz der Datei. Die Satzanzahl SZ beträgt 8. Der mittlere Satz SA errechnet sich als $SA = \text{INT}(SZ/2)$. Im ersten Durchgang wird der Zeiger also auf 4 gesetzt.

Nun untersuchen wir, ob unser Suchbegriff GORDON SAGA gleich Satz 4, kleiner als Satz 4 oder größer als Satz 4 ist. Wir kommen zu dem Ergebnis, daß der Suchsatz kleiner als Satz 4 ist. Der Suchsatz muß also in der ersten Hälfte der Datei stehen (im Bereich Satz 1 – Satz 3).

Wir halbieren den Restbereich und haben als neue Mitte Satz 2. GORDON SAGA ist jedoch größer als Satz 2. Unser Suchsatz kann also nur noch im Bereich von Satz 3 – Satz 3 stehen – also in Satz 3. Damit haben wir ihn gefunden.

Die binäre Suche besteht also darin, Bereiche nach oben beziehungsweise nach unten hin zu halbieren, bis der Suchsatz gefunden ist. Das Schema in Bild 2 macht die binäre Suchmethode noch viel deutlicher. SU\$ ist der Begriff (String), der gesucht werden soll. D\$(SA) ist der Inhalt (String) des Satzes mit der Nummer SA beziehungsweise des Satzes der Datei.

```

10 GOSUB 10000:REM DATEI DEFFINIEREN <103>
20 PRINT"DATEI UNGEORDNET:" <035>
30 FOR I=1 TO SZ:PRINT D$(I):NEXT I <120>
40 GOSUB 30000:REM 1.SCHLUESSEL SORTIEREN <078>
50 PRINT"DATEI NACH 1.SCHLUESSEL SORTIERT:" <137>
   " <150>
60 FOR I=1 TO SZ:PRINT D$(I):NEXT I <072>
70 END
10000 REM FRUECHTE-DATEI DEFFINIEREN -----
   -- <026>
10010 SZ=8 :REM ANZAHL DER SAETZE <038>
10020 DIM D$(SZ):REM STRING-FELD <163>
10030 FOR I=1 TO SZ <073>
10040 : READ D$(I) <073>
10050 NEXT I <228>
10100 DATA"VIZAWRITE<6SPACE>01200.0024" <086>
10105 DATA"WORDPRO<8SPACE>01155.5013" <036>
10110 DATA"SPACE INVADERS 02 39.0067" <175>
10115 DATA"JUMPMAN<8SPACE>02 49.0034" <111>
10120 DATA"SIMONS BASIC<3SPACE>03 99.0065" <230>
10125 DATA"G-BASIC<8SPACE>03140.0045" <186>
10130 DATA"GORDON SAGA<4SPACE>04 39.0089" <093>
10135 DATA"DEADLINE<7SPACE>04 99.0034" <132>
10140 RETURN <036>
30000 REM QUICKSORT -----
   -- <087>
30001 DIM LG(100),RG(100):Z=0:LG(1)=1:RG(1)=SZ <177>
30010 Z=Z+1:IF LG(Z)>=RG(Z)THEN 30120 <166>
30020 X=LG(Z):Y=RG(Z) <011>
30030 VG$=LEFT$(D$(INT((X+Y)/2)),15) <071>
30040 IF X>Y THEN 30100 <205>
30050 IF LEFT$(D$(X),15)<VG$THEN X=X+1:GOTO 30050 <015>
30060 IF LEFT$(D$(Y),15)>VG$THEN Y=Y-1:GOTO 30060 <034>
30070 IF X>Y THEN 30100 <235>
30080 S$=D$(X):D$(X)=D$(Y):D$(Y)=S$ <130>
30090 X=X+1:Y=Y-1:GOTO 30040 <099>
30100 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 30010 <184>
30110 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 30010 <104>
30120 Z=Z-1:RETURN <161>

```

Listing 14. Sortieren einer Datei

```

1110 : IF SU$=MID$(D$(SA),1,LEN(SU$))THEN
   PRINT"SU$ STEHT IN SATZ ";SA:GOTO 200 <160>
1120 : IF SU$<MID$(D$(SA),1,LEN(SU$))THEN
   SA=SA-(2↑(N-1)):GOTO 1200 <250>
2000 REM SU$ IST GEFUNDEN - AUSWERTUNG ---
   -- <163>
2010 PRINT" {CLR,RVSON}";LEFT$(D$(SA),15);"
   {RVOFF}":PRINT <221>
2020 PRINT"PROGRAMMART: "; <201>
2030 I=VAL(MID$(D$(SA),16,2)) <176>
2040 IF I=1 THEN PRINT"TEXTVERARBEITUNG" <235>
2050 IF I=2 THEN PRINT"ACTIONSPIEL" <055>
2060 IF I=3 THEN PRINT"BASIC-ERWEITERUNG" <246>
2070 IF I=4 THEN PRINT"ADVENTURE" <171>
2080 PRINT"PREIS: ";MID$(D$(SA),18,6);" DM
   " <111>
2085 PRINT"BESTAND: ";VAL(RIGHT$(D$(SA),2))
   "; "STUECK" <056>
2090 GOTO 1000 <254>

```

Listing 14b. Ergänzung zu Listing 14.

Man setzt also jeweils einen Zeiger in die Mitte des Bereiches, in dem sich der gesuchte Begriff befinden muß. Am Anfang des Suchvorgangs ist dies logischerweise die Mitte der gesamten Datei. Danach wird untersucht, ob er Begriff gefunden wurde beziehungsweise ob der sich im Bereich unterhalb oder oberhalb des Zeigers befindet. Stellt man wie in unserem Bild beim 1. Vergleich fest, daß sich der gesuchte Begriff unterhalb des Zeigers befindet, so wird der neue Zeiger auf die Mitte des unteren Bereiches etc. gesetzt. Was jedoch, wenn ein Suchbegriff überhaupt nicht in einer Datei vorkommt?

Ganz einfach, man kann bei einer Datei mit bekannter Satzanzahl SZ genau berechnen, wieviele Vergleiche maximal

gemacht werden müssen, um einen Begriff zu finden, der in der Datei enthalten ist. Die binäre Suchmethode beruht schließlich auf dem Halbieren von Bereichen.

Gehen wir wieder einmal von der Software-Datei aus. Sie enthält genau 8 Elemente. Nach dem ersten Suchversuch wird die Datei halbiert. Es kommen dann nur noch 4 Elemente zur Auswahl, in deren Mitte der zweite Zeiger positioniert ist. Nach dem zweiten Versuch kommen nur noch 2 Elemente in Frage und spätestens nach dem 4. Versuch schließlich nur noch eine.

Spätestens beim 4. Versuch (Halbierung) muß ein Suchbegriff somit in der Software-Datei gefunden werden – andernfalls kommt er nicht vor.

Die Formel, mit der sich die maximal nötige Suchvorgangs-Anzahl berechnen läßt, lautet so:

$$N = \text{INT}(\text{LOG}(\text{SZ}) / \text{LOG}(2)) + 1$$

Nach N-Versuchen muß ein Begriff also gefunden sein, wenn er in einer Datei mit SZ-Sätzen enthalten ist.

Am besten läßt sich die binäre Suchroutine anhand eines entsprechenden Unterprogrammes (Listing 15) erklären.

Das Unterprogramm setzt folgendes voraus:

Das Feld, in dem gesucht werden soll, steht in D\$(0)–D\$(SZ), wobei SZ die Nummer des höchsten Satzes beziehungsweise die Anzahl der Sätze darstellt.

Dokumentation:

1010 Eingabe des Suchbegriffs SU\$.

1020 Berechnung der maximal nötigen Suchvorgänge: N

1030 SA, der Zeiger auf die Datei wird zu Beginn des Suchvorgangs in die Mitte der Datei gesetzt. Warum berechnet sich die Mitte nicht einfach mit $SA = \text{SZ} / 2$? Ganz einfach: Zum Halbieren »der Restbereiche« verwenden wir Potenzen von 2 (wird gleich erläutert). Probleme würde es geben, wenn eine Datei aus einer ungeraden Anzahl von Sätzen bestünde. Man könnte dann schlecht halbieren. Eine »ungerade Datei« einfach mit einem »Pseudozusatzsatz« zu versehen, würde auch nichts bringen – früher oder später kämen beim Halbieren ungerade SA-Positionen zustande – und die darf's nicht geben. Die Formel $SA = (2 \times a^N) / 2$ gewährleistet jedoch, daß Nmal halbiert werden kann, wobei die Restbereiche stets aus einer geraden Anzahl von Elementen bestehen.

1040 Nach jedem Suchvorgang (Halbierung) wird die Variable N um 1 erniedrigt. Ist sie kleiner als 0, so kommt der Suchbegriff in der Datei nicht vor.

1050 – 1100 Halbierungsvorgang

1060 Prüfen, ob D\$(SA) gleich SU\$ ist. Wenn ja, so ist der Begriff gefunden worden und die Suche wird abgebrochen.

1070 Wenn SU\$ kleiner als D\$(SA) ist, so heißt dies, daß sich SU\$ (falls es vorkommt) nur in dem Bereich über dem Zeiger SA befinden kann. Der Zeiger muß also auf die Mitte dieses Bereiches gesetzt werden. Das Versetzen geschieht mit der Potenzfunktion: $SA = SA - (2 \times a^{(N-1)})$.

Dadurch, daß N nach jedem Suchbegriff um 1 verkleinert wird, werden die Restbereiche und somit die Verschiebungsdistanz des Zeigers auch immer kleiner.

1080–1090 Diese Zeilen treten dann in Kraft, wenn SU\$ größer als D\$(SA) ist. Die Verschiebungsformel lautet dann: $SA = SA + (2 \times a^{(N-1)})$, da der Zeiger jetzt in den unteren Bereich verschoben werden muß. Um einen Überlauf zu verhindern, ist Zeile 1090 nötig. In Zeile 1030 idealisierten wir unsere Datei so, daß nie ungerade Zeigerwerte auftreten können. Die Datei wird also größer angenommen, als sie eigentlich ist. Das Vergrößern der Datei ist die einzige Möglichkeit zum Idealisieren. Würde man sie verkleinern, so gingen Sätze verloren beziehungsweise würden beim Suchen außer acht gelassen werden. Die Zeile 1080 ruft solange rekursiv auf, bis der Zeiger SA sich wieder im erlaubten Bereich von 0 bis SZ befindet – falls er diesen Bereich verlassen hat. Es ist erlaubt, den Zeiger außerhalb des erlaubten Bereiches zu positionieren; es ist jedoch klar, daß ein Suchbegriff außer-

```

1000 REM BINAERE SUCHROUTINE -----
-----
1005 INPUT SU$ :REM SUCHBEGRIFF                <046>
1010 N=INT(LOG(SZ)/LOG(2))+1                    <209>
      :REM MAXIMAL-FORMEL                      <137>
1040 SA=(2*N)/2:REM MITTE DER GES.DAT          <071>
1050 N=N-1 :REM 1.ABFRAGE                      <086>
1100 :                                          <060>
1110 : IF SU$=D$(SA) THEN PRINT"SU$ STEHT I
N SATZ ";SA:END:REM SU$ GEFUNDEN              <033>
1120 : IF SU$<D$(SA) THEN SA=SA-(2*(N-1)):G
OTO 1200 :REM ''HALBIEREN''                  <030>
1125 : IF SA+(2*(N-1))>SZ AND N>-1 THEN N=
N-1:GOTO 1125 :REM ''UEBERLAUF''             <147>
1130 : SA=SA+(2*(N-1))
      :REM ''HALBIEREN''                      <246>
1200 :                                          <160>
1210 : N=N-1 :REM NAECHST. HALBIEREN          <149>
1220 : IF N<0 THEN PRINT"SU$ GIBT'S NICHT!
IN 1220":END :REM SU$ NICHT IN DAT          <159>
1225 PRINT"SA=";SA;" N=";N
      :REM UNNOETIG !!                        <036>
1230 :GOTO 1100 :REM NAECHST. SUCH-VERS.     <123>

```

Listing 15. Die binäre Suchmethode

halb der Datei gefunden werden kann. Auch kann es nur im Fall SU > D(SA) zu einer Positionierung außerhalb des Datenbereiches kommen.

1110 Nächsten Suchvorgang vorbereiten

1120 Ist N kleiner als 0, so befindet sich SU\$ nicht in der Datei.

1130 kann jederzeit entfernt werden.

Dies hört sich alles schön und gut an, ich weiß jedoch, daß sich viele schwer tun, wenn sie das erste Mal mit binären Suchsystemen konfrontiert werden. Ich rate Ihnen deshalb, falls Sie Schwierigkeiten haben, sich das obige Programm

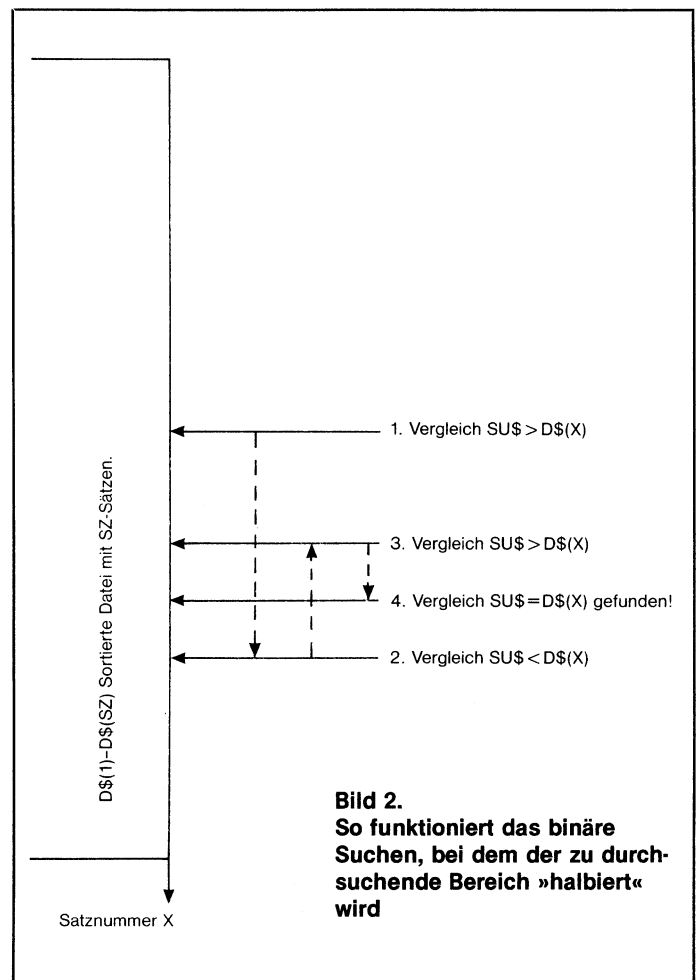


Bild 2.
So funktioniert das binäre Suchen, bei dem der zu durchsuchende Bereich »halbiert« wird

einmal Schritt für Schritt durchzudenken. Nehmen Sie gegebenenfalls ein Blatt Papier und eine kleine Datei her und schreiben Sie den gesamten Programmablauf (wo sich der Zeiger nach jedem Suchvorgang befindet) in Form von einfachen Schemen auf. Hat man es erstmal verstanden, dann ist es ganz einfach.

Bitte ergänzen Sie jetzt Listing 14 mit Listing 15. Außerdem sind folgende Änderungen vorzunehmen:

Zeile 70 löschen.

Ergänzen beziehungsweise ändern Sie auch die Zeilen, wie in Listing 14b angegeben.

Die binäre Suchmethode ist sehr schnell. Um eine Datei von 1000 Sätzen nach einem Begriff zu durchsuchen, sind maximal 10 Halbierungen notwendig, um festzustellen, in welchem Satz der Begriff steht, beziehungsweise daß er nicht vorkommt.

Erhöhen kann man die Suchgeschwindigkeit noch durch Einsatz von index-sequentiellen Dateien, wie wir sie bei Parser 4.0 kennenlernen werden.

Das Programm, das Sie jetzt im Computer haben, simuliert eine einfache Dateiverwaltung für Software.

Damit haben Sie den ersten Einstieg in die Dateiverwaltung auch schon geschafft. Wir können uns nun Dateien auf Disk zuwenden, die für die Programmierung von Adventures und KI ein großes Hilfsmittel darstellen.

Deutscher Zeichensatz im Commodore 64

Bevor ich mich der Programmierung von relativen Dateien widme, möchte ich schnell noch dieses Kapitel einfügen. Wer professionelle deutsche Text-Adventure schreiben will (und das können Sie am Ende dieses Kurses!), sollte keineswegs auf einen deutschen Zeichensatz verzichten.

Auch bei allen anderen Programmen mit deutschen Texten macht ein deutscher Zeichensatz einen besseren Eindruck. Natürlich gehen bei einer Zeichensatzänderung immer ein paar KByte-RAM flöten (der Zeichensatz muß schließlich vom ROM ins RAM kopiert werden), aufgrund guter Programmieretechniken, die sehr platzsparend sind, können wir diesen Verlust jedoch getrost verschmerzen.

Auch bei relativen Dateien bringt ein neuer Zeichensatz erhebliche Vorteile, wie wir schon bald sehen werden.

Zunächst müssen wir festlegen, wo im RAM der neue Zeichensatz liegen soll.

Bild 3 zeigt den Speicherbelegungsplan des C64, wie er Ihnen wahrscheinlich bereits von anderen Kursen aus dem 64'er-Magazin her bekannt ist. Uns interessieren nur der linierte Bereich und das Basic-RAM. Im Laufe des Kurses werden wir den gesamten linierten RAM-Bereich hernehmen (für Maschinenprogramme, Dateien etc.). Den Basic-RAM

versuche ich weitgehend für Ihre eigenen Ideen und Routinen zur freien Verfügung zu halten. Sie sollten sich nicht wundern, warum bestimmte Routinen (zum Beispiel Zeichensatz) in bestimmte Speicherbereiche geschrieben werden, denn alle Routinen sollen sich am Ende des Kurses bei vollausgelastetem RAM miteinander vertragen. Grübeln Sie also nicht lange nach, wenn ich eine Routine in einen zunächst ungeeignet anmutenden RAM-Bereich schreibe.

Unseren deutschen Zeichensatz bringen wir im Bereich 57344 – 61440 unter. Da der VIC jedoch nur ganze RAM-Blöcke von 16 KByte verwalten kann, müssen wir auch den Videobereich (Bildschirm) verschieben. Ihn legen wir nach 52223 – 53248. Bitte beachten Sie, daß hierbei auch die Sprite-Pointer (2040 – 2047) nach 53239 – 53246 verschoben werden (Bild 4)!

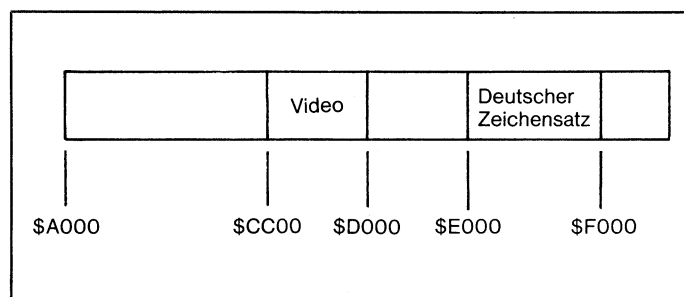


Bild 4. Hier bringen wir den deutschen Zeichensatz unter

Bei dem Listing 16 handelt es sich um einen menüfreundlichen Zeichensatzeditor, der speziell für den Kurs abgerichtet ist (also für RAM-Bereich 57344–61440).

Dokumentation zum Listing 16

Der Zeichensatzeditor ist lediglich ein Utility zum Kurs. Ich beschränke mich daher auf das Wesentliche.

Interessant ist im Prinzip nur das Abändern des Listings zum Erstellen für Zeichensätze in andere RAM-Bereiche als die Vorgeschlagenen. Ich bin mir jedoch ziemlich sicher, daß der Zeichensatz im Bereich von 57344 – 61440 sehr gut untergebracht ist, wie ich Ihnen schon bald zeigen werde.

Um den Editor für andere Bereiche umzuschrauben, müssen lediglich folgende Zeilen abgewandelt werden:

55 IZ = Startadresse im RAM (ist momentan 57344)
122 siehe Listing
124 siehe Listing
201 VIDEO = 52223
1030 und 1040 siehe Listing
2010 siehe Listing

Es müssen also nur Anfangs- und Endadressen vom ZS-Bereich und gegebenenfalls vom Video modifiziert werden.

Bedienungsanleitung

Nach dem Programmstart meldet sich das Hauptmenü mit drei Auswahlmöglichkeiten:

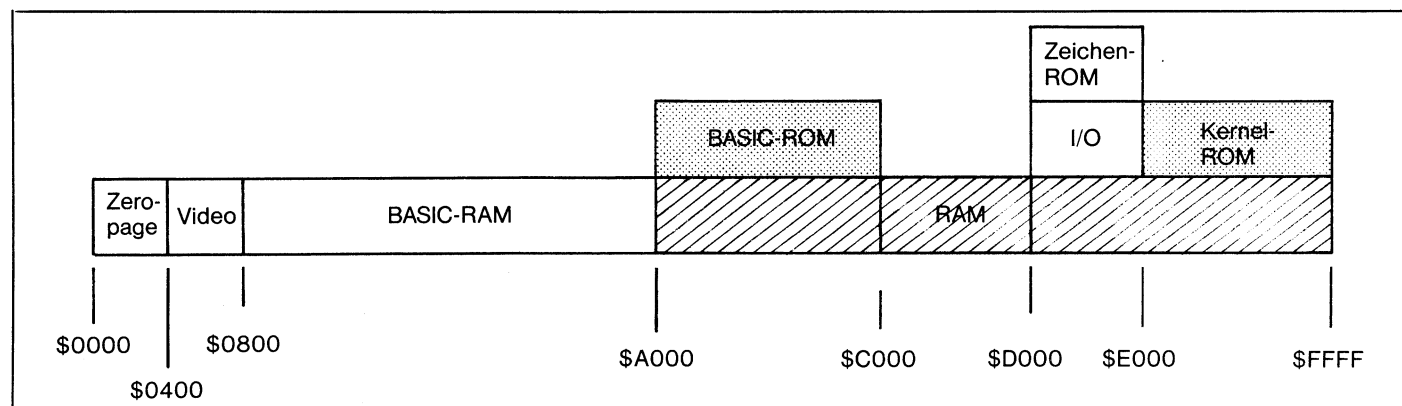
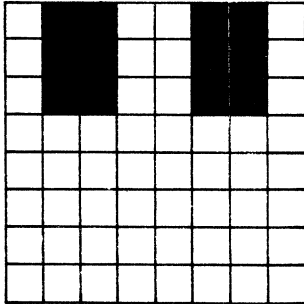
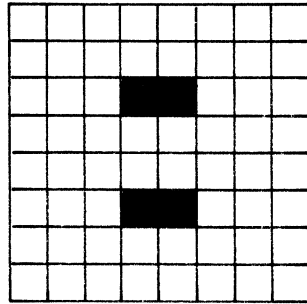


Bild 3. So sieht die Speicherbelegung des C64 aus

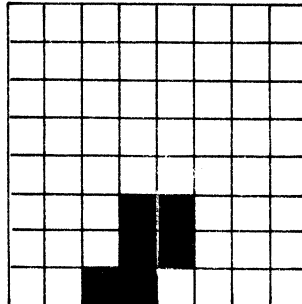
CHR\$ (64)



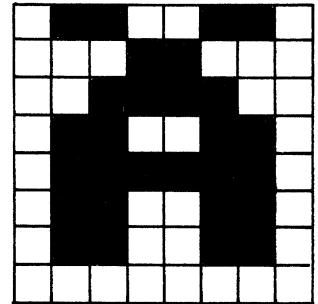
CHR\$ (94)



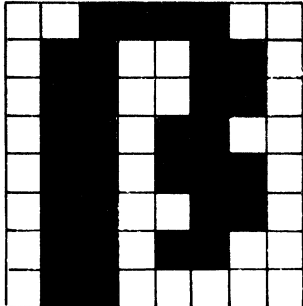
CHR\$ (95)



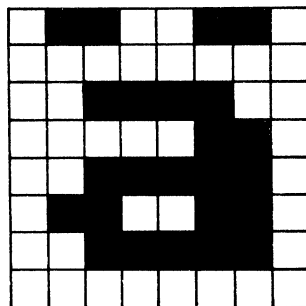
CHR\$ (173)



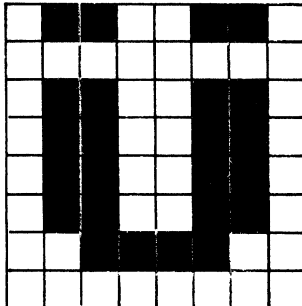
CHR\$ (174)



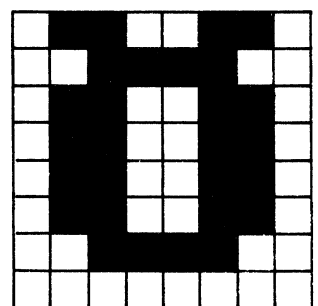
CHR\$ (176)



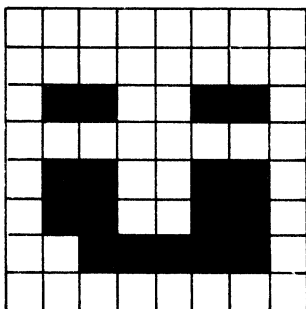
CHR\$ (181)



CHR\$ (182)



CHR\$ (184)



CHR\$ (185)

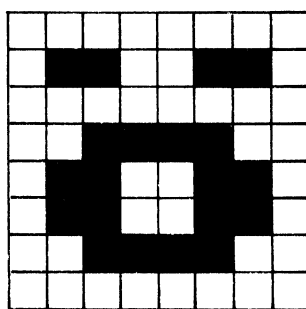


Bild 5. So sehen die deutschen Zeichen aus

Geänderte Zeichen

ä = € + a Ä = € + Z ll = @
 ö = € + o Ö = € + L : = ↑
 ü = € + u Ü = € + J / = —

B = C + []

NEUEN Z.SATZ ERSTELLEN: Durch diese Wahl wird der Zeichensatz vom ROM in den RAM-Bereich ab 57344 kopiert.

ALTEN Z.SATZ ÄNDERN: Ein bereits bestehender Zeichensatz kann geladen werden, um weiter verändert zu werden.

DEUTSCHEN Z.SATZ GENERIEREN: Der Editor erstellt hier automatisch einen deutschen Zeichensatz, den Sie anschließend speichern können. Bild 5 zeigt Ihnen, welche Zeichen wie geändert wurden. Ich bin mir sicher, daß Sie sich schnell an die neue Tastaturbelegung gewöhnen werden. Daß auch Zeichen wie „ und : neu definiert wurden, hat seinen Grund: den erfahren Sie im nächsten Kapitel!

Nach der Wahl von einem der drei Hauptmenüpunkten gelangen Sie ins nächste Menü.

```

0 REM ***** <012>
1 REM ZEICHENSATZ-EDITOR (C) 1985 <146>
2 REM ***** <014>
3 IF I=1 THEN 100 <221>
4 VIDEO=52223:REM START BILDSCHIRM <012>
5 POKE 53280,0:POKE 53281,0:PRINT "YELLOW" <235>
" <054>
10 REM MODIFIZIERTE PEEK-FUNKTION ----- <054>
11 FOR I=704 TO 751:READ X:POKE I,X:NEXT:P <009>
   OKE 785,192:POKE 786,2:REM USR-VEKTOR <009>
12 DATA 165,20,72,165,21,72,32,247,183,165 <079>
   ,1,72,165,21,201,208,144,7,201,224 <079>
13 DATA 176,3,169,49,44,169,52,120,133,1,1 <228>
   60,0,177,20,168,104,133,1,88,104,133 <191>
14 DATA 21,104,133,20,76,162,179 <009>
30 REM MENUE ----- <009>
31 PRINT "CLR,WHITE,RVSON}ZEICHENSATZ-EDIT <037>
   OR I(RVOFF)" <006>
32 PRINT "DOWN}VON MICHAEL NICKLES. <216>
34 PRINT "3DOWN,5SPACE,RVSON}N(RVOFF}EUEEN <076>
   ZEICHENSATZ ERSTELLEN" <243>
35 PRINT "DOWN,5SPACE,RVSON}A(RVOFF}LTEN Z <009>
   EICHENSATZ AENDERN(2SPACE)" <009>
36 PRINT "DOWN,5SPACE,RVSON}D(RVOFF}EUTSCH <009>
   EN ZEICHENSATZ GENERIEREN(2SPACE)" <009>
37 GET I$:IF I$<>"N"AND I$<>"A"AND I$<>"D" <009>
   THEN 37

```

```

38 IF I$="N"THEN 50 <171>
39 IF I$="D"THEN 50 <167>
40 REM DEUTSCHEN ZEICHENSATZ LADEN ----- <178>
41 INPUT "3DOWN}FILENAME ";I$ <206>
42 IF I<>1 THEN I=1:LOAD I$,8,1 <074>
44 POKE 648,204:POKE 53272,56:POKE 56576,0 <069>
   :PRINT "CLR" <247>
50 REM Z.S. VOM ROM INS RAM AB 57344 --- <080>
51 PRINT "DOWN}BITTE WARTEN!" <161>
53 POKE 56334,PEEK(56334)AND 254:POKE 1,PE <001>
   EK(1)AND 251:REM INTERRUPT AUS <033>
55 IZ=57344 <139>
60 FOR I=53248 TO 57344 <248>
62 :POKE IZ,PEEK(I) <150>
64 :IZ=IZ+1 <223>
66 NEXT I <209>
70 POKE 1,PEEK(1)OR 4:POKE 56334,PEEK(5633 <226>
   4)OR 1 <169>
80 IF I$="D"THEN 3000 <223>
100 REM START ----- <169>
105 POKE 648,204:POKE 53272,56:POKE 56576, <223>
   0:PRINT "CLR,WHITE"}; <169>
106 PRINT "BITTE WAEHLEN SIE: <223>
110 PRINT "2DOWN,5SPACE}GROSS-GRAFIK Z.SAT

```

Listing 16. Verzichten Sie in deutschen Adventures nicht auf den deutschen Zeichensatz

```

Z(2SPACE,RVSON)1(RVOFF)" <021>
115 PRINT"(DOWN,5SPACE)GROSS-KLEIN(2SPACE) <026>
Z.SATZ(2SPACE,RVSON)2(RVOFF)"
116 PRINT"(DOWN,5SPACE)Z.SATZ ABSPEICHERN( <210>
3SPACE,RVSON)3(RVOFF)
118 PRINT"(DOWN,5SPACE)CRAZY CHARACTERS(5S <080>
PACE,RVSON)4(RVOFF)
120 GET I$:IF I$<>"1"AND I$<>"2"AND I$<>"3 <091>
"AND I$<>"4"THEN 120
122 IF I$="1"THEN AN=57344 :PRINT CHR <132>
$(142)
124 IF I$="2"THEN AN=57344+256*8:PRINT CHR <051>
$(14)
125 IF I$="3"THEN 1000 <109>
127 IF I$="4"THEN 2000 <244>
130 PRINT"(2DOWN)WELCHES ZEICHEN WOLLEN SI <082>
E AENDERN?"
131 PRINT"ANGABE BITTE IM BILDSCHIRMCODE! <190>
132 INPUT"(DOWN)ZEICHEN :;NR:Z=NR <234>
200 REM MASKE AUSGEBEN ----- <013>
201 VIDEO=52223 <169>
205 PRINT"(CLR)ZEICHEN: ":POKE VI+10,Z <144>
210 Z=AN+Z*8 <160>
229 PRINT"(2DOWN,4SPACE)"; <180>
230 FOR I1=Z TO Z+7 <018>
235 PRINT I1;"(3SPACE)"; <233>
240 :B=USR(I1) <060>
245 : FOR I2=7 TO 0 STEP-1 <036>
250 : IF B AND 2*I2 THEN PRINT"*";:GOTO <166>
260
255 : PRINT". "; <191>
260 : NEXT I2 <220>
270 :PRINT"(3SPACE)"B:PRINT"(4SPACE)"; <021>
275 NEXT I1 <124>
300 REM EDITOR-PROGRAMM ----- <045>
310 XX=VI+135:XP=PEEK(XX):POKE XX,32 <130>
330 JOY=56320:REM PORT 2 <032>
331 TA=203:T1=653 <206>
335 IF PEEK(JOY)=127 AND PEEK(TA)=64 THEN <140>
W=0
340 IF PEEK(JOY)=126 OR PEEK(TA)=7 AND PEE <054>
K(T1)=1 THEN W=-40
342 IF PEEK(JOY)=125 OR PEEK(TA)=7 AND PEE <023>
K(T1)=0 THEN W=+40
344 IF PEEK(JOY)=123 OR PEEK(TA)=2 AND PEE <244>
K(T1)=1 THEN W=-1
346 IF PEEK(JOY)=119 OR PEEK(TA)=2 AND PEE <061>
K(T1)=0 THEN W=+1
350 IF PEEK(JOY)>111 AND PEEK(TA)<>60 THEN <201>
360
352 IF XP=46 THEN XP=42:GOTO 355 <078>
354 IF XP=42 THEN XP=46:GOTO 355 <206>
355 POKE XX,XP:IF PEEK(JOY)<=111 OR PEEK(T <204>
A)=60 THEN 355
360 :IF PEEK(TA)=1 THEN POKE XX,XP:GOTO 40 <084>
0
370 POKE XX,XP <220>
375 XX=XX+W:IF PEEK(XX)=32 THEN XX=XX-W <020>
380 XP=PEEK(XX):POKE XX,32 <206>
390 GOTO 330 <128>
400 REM NEUES ZEICHEN BERECHNEN <026>
405 PRINT"(DOWN)BITTE WARTEN!" <180>
410 XX=VI+135 <198>
430 FOR I1=XX TO XX+7*40 STEP 40 <179>
435 :IM=0:IZ=7 <186>
445 : FOR I2=I1 TO I1+7 <149>
450 : IF RV=0 THEN IF PEEK(I2)=42 THEN I <155>
M=IM+2*I2
451 : IF RV=1 THEN IF PEEK(I2)=46 THEN I <036>
M=IM+2*I2
455 : IZ=IZ-1 <130>
460 : NEXT I2 <164>
470 :POKE Z,IM:Z=Z+1 <248>
475 NEXT I1 <068>
476 IF RV=1 THEN RV=0:GOTO 100 <041>
479 IF NR>127 THEN 100 <015>
480 PRINT"REVERS-AENDERUNG ERWUNSCHT? (J/ <210>
N)
482 GET I$:IF I$<>"J"AND I$<>"N"THEN 482 <197>
484 IF I$="N"THEN 100 <094>
485 Z=AN+(NR+128)*8:RV=1:GOTO 400 <013>
490 GOTO 100 <164>
1000 REM Z.SATZ ABSPEICHERN ----- <196>
1000 REM SPEICHERN AUF DISK ----- <092>
1005 PRINT"(CLR,RVSON)ZEICHENSATZ AUF DISK <178>
SPEICHERN(RVOFF)":POKE 198,0
1006 INPUT"(2DOWN)FILENAME";I$ <206>
1010 PRINT"(DOWN)BITTE DISKETTE EINLEGEN! <153>

```

```

1020 PRINT"[RETURN]":POKE 198,0:WAIT 198,1 <000>
1030 AA=57344 <240>
1040 EA=61439 <051>
1050 OPEN 1,8,1,I$ <227>
1060 HB=INT(AA/256):LB=AA-HB*256 <006>
1070 PRINT#1,CHR$(LB);CHR$(HB); <179>
1080 FOR I=AA TO EA <076>
1090 :PRINT#1,CHR$(USR(I)); <001>
1100 NEXT I <168>
1110 CLOSE 1 <105>
1120 PRINT"(3DOWN)*** FERTIG ***:RUN <251>
2000 REM CRAZY CHARACTERS ----- <203>
2010 FOR I=57344 TO 61439 <079>
2020 :X=USR(I):M=0 <213>
2030 : FOR I1=0 TO 7 <128>
2040 : IF X AND 2*I1 THEN M=M+2*(7-I1) <044>
2050 : NEXT I1 <200>
2060 :POKE I,M <044>
2070 :NEXT I <182>
2080 GOTO 100 <099>
3000 REM DEUTSCHEN ZEICHENSATZ AUTOMATISCH <243>
GENERIEREN -----
3010 DATA 60264,102,24,60,102,126,102,102, <124>
0 :REM GROSSES AE
3015 DATA 60336,102,60,102,102,102,102,60, <143>
0 :REM GROSSES OE
3020 DATA 60328,102,0,102,102,102,102,60,0 <202>
:REM GROSSES UE
3025 DATA 60288,102,0,60,6,62,102,62,0 <091>
:REM KLEINES AE
3030 DATA 60360,0,102,0,60,102,102,60,0 <068>
:REM KLEINES OE
3035 DATA 60352,0,102,0,102,102,102,62,0 <048>
:REM KLEINES UE
3040 DATA 60272,60,102,102,108,110,102,108 <124>
,96:REM SCHARFES S
3045 DATA 59392,102,102,102,0,0,0,0 <042>
:REM ANFUEHRUNGSZEICHEN
3050 DATA 59640,0,0,0,0,0,24,24,48 <006>
:REM KOMMA
3055 DATA 59632,0,0,24,0,0,24,0,0 <218>
:REM DOPPELPUNKT
3060 DATA-1 <148>
3100 READ X:IF X=-1 THEN 100 <212>
3110 FOR I=X TO X+7 <049>
3120 : READ A:POKE I,A :REM NORMAL <066>
3130 : POKE I+2048,A:REM REVERS <110>
3140 NEXT I <176>
3150 GOTO 3100 <074>

```

Listing 16. Ein einfacher Zeichensatzeditor

programm :	rueckwaerts	c400	c4bb
c400 :	a9 c0 8d 88 02 20 44 e5	7c	
c408 :	a2 15 a0 c4 78 8e 14 03	48	
c410 :	8c 15 03 58 60 a2 00 a0	4f	
c418 :	27 bd 00 c0 99 00 cc bd	7e	
c420 :	28 c0 99 28 cc bd 50 c0	91	
c428 :	99 50 cc bd 78 c0 99 78	b9	
c430 :	cc bd a0 c0 99 a0 cc bd	68	
c438 :	c8 c0 99 c8 cc bd c0 c0	e0	
c440 :	99 f0 cc bd 18 c1 99 18	62	
c448 :	cd bd 40 c1 99 40 cd bd	8a	
c450 :	68 c1 99 68 cd bd 90 c1	9d	
c458 :	99 90 cd bd b8 c1 99 b8	d6	
c460 :	cd bd e0 c1 99 e0 cd bd	cf	
c468 :	08 c2 99 08 ce bd 30 c2	5a	
c470 :	99 30 ce bd 58 c2 99 58	3f	
c478 :	ce bd 80 c2 99 80 ce bd	f2	
c480 :	a8 c2 99 a8 ce bd d0 c2	a8	
c488 :	99 d0 ce bd f8 c2 99 f8	f3	
c490 :	ce bd 20 c3 99 20 cf bd	13	
c498 :	48 c3 99 48 cf bd 70 c3	65	
c4a0 :	99 70 cf bd 98 c3 99 98	5c	
c4a8 :	cf bd c0 c3 99 c0 cf e8	af	
c4b0 :	88 e0 28 f0 03 4c 19 c4	51	
c4b8 :	4c 31 ea 00 00 00 00 00	58	

Listing 17. Erster Teil des Überraschungs-Programms

Sie haben nun folgende Möglichkeiten:

1: GROSS-GRAFIK Z.SATZ: Dies müssen Sie wählen, wenn Sie ein Zeichen des GROSS/GRAFIK-Zeichensatzes ändern wollen.

2: GROSS-KLEIN Z.SATZ: Wie oben, jedoch GROSS/KLEIN-Zeichensatz.

Nach Wahl von Punkt 1 oder 2 müssen Sie den Bildschirmcode des Zeichens angeben, das Sie ändern wollen (Tabelle siehe C64-Handbuch). Nun kann das Zeichen mittels Joystick oder Cursor-Tasten + SPACE editiert werden. Sind Sie mit der Änderung fertig, so muß die RETURN-Taste gedrückt werden. Sie werden dann gefragt, ob auch das entsprechende Revers-Zeichen geändert werden soll. Dann gelangen Sie wieder zurück ins Menü.

3: Z.SATZ Speichern: Der neue Zeichensatz wird auf Disk gespeichert.

4: Crazy Characters: Eine Spezialfunktion des Editors (benötigt viel Rechenzeit!). Speichern Sie den mit Crazy-Characters erzeugten Zeichensatz einmal unter dem Filenamen »CRAZY« ab. Wenn Sie nun das Listing17 und 18 eingeben und starten, werden Sie schnell verstehen, warum ein Spiegel zum unentbehrlichen Programmierhilfsmittel werden kann!

Bitte erzeugen Sie nun mittels dem Menüpunkt »DEUTSCHEN Z.SATZ GENERIEREN« einen deutschen Zeichensatz und speichern Sie diesen unter dem Filenamen »D.Z.SATZ« auf Disk – wir werden ihn später benötigen!

Laden des Zeichensatzes

So wird der deutsche Zeichensatz innerhalb von Basic-Programmen geladen;

```
10 IF I=0 THEN I=1:LOAD"D.Z.SATZ",8,1
20 POKE648,204:POKE53272,56:POKE56576,0
30 END
```

Ist ein neuer Zeichensatz aktiviert und ein Programm wird mittels RUN/STOP + RESTORE unterbrochen, so stürzt das Programm ab, der Cursor ist verschwunden. Der Cursor ist allerdings noch da – Sie können ihn momentan nur nicht sehen! Geben Sie nun bitte »blind« folgendes im Direktmodus ein:

```
POKE 648,204 (RETURN)
POKE 53272,56 (RETURN)
POKE 56576,0 (RETURN)
```

Der Cursor ist wieder da! Am besten, Sie schreiben sich diese POKEs gleich auf einen Aufkleber und bringen diesen am C64 an.

Relative Dateien auf der 1541

Um es gleich zu sagen: Dies ist KEIN Kurs für Disketten-Laufwerke. Ich werde in diesem Kapitel nicht erklären, wie das Floppy-Laufwerk 1541 seine Daten speichert, wie es in Ihrem RAM/ROM aussieht oder gar wie man den Schreib-Lesekopf im $\frac{1}{4}$ -Takt hin- und herschlagen läßt (als 4. Musikstimme, wie es bei einigen verrückten Programmen schon vorgekommen sein soll). Es geht mir in diesem Kapitel nur darum, daß Sie lernen, spielend leicht mit relativen Dateien umzugehen. Durch den Umgang mit solchen Dateien eröffnen sich völlig neue Programmiermöglichkeiten. Wer erst einmal mit relativen Dateien gearbeitet hat, wird schon bald nicht mehr auf sie verzichten wollen. In der Tat, erst durch relative Dateien ist es möglich, die Speicherkapazität der 1541 voll und ganz auszureizen. Im Bedienungshandbuch für das Dis-

kettenlaufwerk wird kaum auf relative Dateien eingegangen. Auch die großen Bücher über Floppy-Laufwerke geben nicht alle Tips und Tricks zum Umgang mit relativen Dateien preis, sondern handeln alles ganz simpel und logisch ab. Der Anfänger schaut jedoch spätestens dann dumm aus der Wäsche (auch mir ging dies anfangs so), wenn ein FILE DATA ERROR auftaucht oder Satzteile in einer relativen Datei beim Einlesen plötzlich nicht mehr auftauchen. Im Prinzip ist der Umgang mit relativen Dateien jedoch sehr einfach, wenn man erst mal durchgeblickt hat; es gibt nur sehr wenige Kommandos, die nötig sind, um mit relativen Dateien umzugehen: positionieren, lesen und schreiben.

Was ist eine relative Datei?

Am ehesten kann man eine relative Datei mit einem dimensionierten Stringfeld auf Disk vergleichen. Ein Stringfeld im Computer vom Format X\$(1)-X\$(N) entspricht in etwa einer relativen Datei mit N-Sätzen. Genauso wie ein Stringfeld erst einmal mittels DIM definiert werden muß, muß eine relative Datei erst einmal auf Disk angelegt werden, bevor man mit ihr arbeiten kann.

Anlegen einer relativen Datei

Beim Öffnen einer relativen Datei kommt es lediglich auf die Angabe von drei Parametern an: Dateiname, Satzlänge und Satzanzahl der Datei. Listing 19 zeigt, wie eine relative Datei eröffnet wird.

Und so geht das vor sich:

```
20 - 40 Festlegen der 3 Parameter
50 Die Satzanzahl SZ wird in Lower und Higher Byte unterteilt.
60 Öffnen des Floppy-Befehlskanals
70 Initialisieren der Disk. Dies empfiehlt sich immer dann, wenn eine neue Disk eingelegt beziehungsweise eine Seite gewechselt wird.
80 Eine relative Datei mit Namen FS und der Satzlänge SL wird geöffnet.
90 Der relative Datei-Zeiger wird auf den höchsten Satz der Datei positioniert.
100 In den letzten beziehungsweise höchsten Satz der Datei wird CHR$(255) geschrieben.
110 Floppy-Datenkanal und relative Datei werden geschlossen.
```

```
10 REM CRAZY-LOADER <125>
30 IF I2=0 THEN I2=1:LOAD"CRAZY",8,1 <047>
40 IF I2=1 THEN I2=2:POKE 648,204:POKE 53272,56:POKE 56576,0 <082>
60 IF I1=0 THEN I1=1:LOAD"RUECKWAERTS",8,1 <084>
70 IF I1=1 THEN I1=2:SYS 50176:SYS 64764 <100>
```

Listing 18. Zweiter Teil des Überraschungs-Programms

```
10 REM EROEFFNEN EINER REL. DATEI ----- <197>
----- <197>
20 F$="TEST" :REM DATEINAME <088>
30 SL=40 :REM SATZLAENGE <148>
40 SZ=50 :REM SATZANZAHL <172>
50 HB=INT(SZ/255):LB=SZ-HB*255 <042>
60 OPEN 15,8,15 :REM FLOPPYKANAL OEFFNE N <158>
70 PRINT#15,"I0" :REM INITIALISIEREN <229>
80 OPEN 1,8,3,F$+"",L,"+CHR$(SL) :REM DATEI OEFFNEN <246>
90 PRINT#15,"P";CHR$(3);CHR$(LB);CHR$(HB);CHR$(1) <105>
100 PRINT#1,CHR$(255) <111>
110 CLOSE 1:CLOSE 15 :REM KANAELE SCHLIESS EN <096>
```

Listing 19. Öffnen einer relativen Datei

Nach dem Programmstart kann es eine Weile dauern - dies hängt natürlich von der Größe der Datei ab. Am Ende blinkt die rote LED des Diskettenlaufwerks. Dies zeigt einen Fehler Nr. 50 - »RECORD NOT PRESENT« an. Das Auftauchen dieses Fehlers nach dem Anlegen einer relativen Datei (unmittelbar danach) ist okay. Sie brauchen sich darum keine Sorgen zu machen.

Sie haben bestimmt schon entdeckt, worin sich relative Dateien von dimensionierten Stringfeldern im Speicher unterscheiden: Die Satzlänge, also die maximale Zeichenzahl, die ein String haben darf, muß genau festgelegt sein. Bei dimensionierten Stringfeldern kann jeder String bis zu 255 Zeichen lang sein. Dies geht natürlich auch bei relativen Dateien, wenn man als maximale Satzlänge SL den Wert 255 angibt. Dies ist jedoch nur selten empfehlenswert, denn: Je kürzer beziehungsweise kleiner die maximal erlaubte Satzlänge SL ist, umso mehr Sätze lassen sich auf einer Disk-Seite unterbringen.

Man sollte sich also am Anfang genau überlegen, wie groß jeder Satz maximal sein muß. Bei der Software-Datei hätte man sich schließlich für die Satzlänge SL=25 entschieden. Man muß auch von Anfang an wissen, wieviele Sätze man benötigt. Dies stimmt nicht ganz, denn man kann relative Dateien prinzipiell auch offen lassen - indem man beim Anlegen der Datei auf den letztmöglichen Satz positioniert und in ihn den Wert 255 schreibt. Davon ist jedoch abzuraten, da offene relative Dateien viel längere Zugriffszeiten erfordern - im Programmablauf also viel mehr Zeit kosten.

Seien Sie beim Anlegen der Dateien stets großzügig. Hat man eine Datei fertig bearbeitet, so kann man leicht eine neue Datei eröffnen, deren Satzlänge und Satzgröße nur so groß ist wie erforderlich. Dann müssen nur noch die Daten aus der ersten Datei in diese Datei übertragen werden. Dies geht ganz einfach mit dem Editor, den Sie in diesem Kapitel noch finden werden.

Lesen/Beschreiben einer relativen Datei

Bevor man eine relative Datei beschreiben kann, muß natürlich erst einmal eine vorhanden sein. Bitte legen Sie eine leere Disk beziehungsweise Disk mit mindestens 10 freien Blöcken ins Laufwerk und starten Sie dann Listing 19 mittels RUN. Es wird sodann (bitte etwas Geduld) eine relative Datei mit dem Filenamen »TEST« und 50 Sätzen mit einer Länge von je 40 Zeichen angelegt.

Will man auf beziehungsweise von einer relativen Datei schreiben oder lesen, so sind folgende Punkte notwendig: Positionieren, Lesen und Schreiben.

Am deutlichsten wird die, wenn man relative Dateien einmal mit Stringfeldern innerhalb von Basic (X\$(1)-X\$(N)) vergleicht.

Ein Stringfeld innerhalb von Basic wird einfach mittels DIM X\$(N)

angelegt, wobei N für die Anzahl der Sätze steht. Jeder Satz darf maximal 255 Zeichen umfassen.

Eine relative Datei hingegen muß erst etwas umständlicher angelegt werden. Wie beim Stringfeld muß auch hierbei die Anzahl der Sätze und außerdem auch noch die maximale Länge jedes Satzes angegeben werden.

Will man einen Satz SA von einem Stringfeld lesen beziehungsweise wissen, was im SAten Satz steht, so geht man ganz einfach so vor:

```
PRINT X$(SA)
```

Bei relativen Dateien geht dies nicht so einfach. Hier muß erst auf den gewünschten Satz SA positioniert werden, bevor er gelesen werden kann. Also:

Positioniere Zeiger auf Satz SA. Lese Satz SA.

Analog verhält es sich natürlich mit dem Schreiben. Es bietet sich an, die Routinen Schreiben, Lesen und Positionieren als Unterprogramme zu gestalten (Listing 20).

Wie funktionieren die drei Unterprogramme?

Mittels GOSUB 9000 wird der Zeiger auf den Satz SA der relativen Datei positioniert. Die Syntax zum Positionieren des Zeigers ab zum Beispiel Satz 23 einer relativen Datei würde so lauten:

```
SA=23:GOSUB 9000
```

9010 Hier wird SA in High- und Lowbyte zerlegt. Es ist also möglich, relative Dateien mit bis zu 65000 Sätzen (Satzlänge entsprechend klein) anzulegen.

9020 Über den Floppy-Befehlskanal wird der Positionierbefehl der Syntax

```
PRINT #15, "P";CHR$(Kanal);CHR$(LOWBYTE);CHR$(HIGHBYTE);CHR$(Zeichen)
```

dem Floppy-Laufwerk mitgeteilt. »Zeichen« gibt an, auf welches beziehungsweise wievielles Zeichen im Satz positioniert werden soll. Will man den ganzen Satz einlesen (was meist der Fall ist), so muß man auf das erste Satzzeichen positionieren, also CHR\$(1). Danach wird der Fehlerkanal abgefragt mittels

```
INPUT #15,A
```

um festzustellen, ob der Satz SA überhaupt vorhanden ist. Nimmt A den Wert 50 an, so bedeutet dies einen RECORD NOT PRESENT ERROR. Dies ist zum Beispiel dann der Fall, wenn wir bei unserer Test-Datei Satz Nummer 70 lesen wollten (die TEST-Datei besteht nämlich nur aus 50 Sätzen).

Mittels GOSUB 9100 wird der Inhalt des Satzes in AS eingelesen, auf den der Zeiger gerade deutet. Will man zum Beispiel wissen, was im 20ten Satz einer Datei steht, so gilt die folgende Syntax:

```
SA=20 : GOSUB 9000 : GOSUB 9100
```

Das Lesen geschieht mittels INPUT # 1,A\$, wenn der Satz beziehungsweise die maximal erlaubte Satzlänge der relativen Datei 88 Zeichen nicht überschreitet (Größe des INPUT-Puffers). Wird mit Sätzen gearbeitet, die größer als 88 Zeichen sind, so muß Zeichen für Zeichen mittels GET-Befehl eingelesen werden. Das Lesen funktioniert also wie bei sequentiellen Dateien und hat auch die gleichen Einschränkungen.

Mittels GOSUB 9200 wird der String A\$ in den Satz SA der relativen Datei geschrieben. Will man zum Beispiel »SUPERMIKE« in den 34ten Satz einer relativen Datei schreiben, dann geht das so:

```

9000 REM ZEIGER POSITIONIEREN -----
-
-
9001 :
9010 HB%=SA/256:LB%=SA-HB%*256
9020 PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$(HB%);CHR$(1)
9030 INPUT#15,A:IF A=50 THEN PRINT"SATZNUMMER ZU HOCH"
9040 RETURN
9100 REM SATZ LESEN -----
-
-
9101 :
9110 IF A=50 THEN RETURN
9120 INPUT#1,A$
9130 RETURN
9200 REM SATZ SCHREIBEN -----
-
-
9201 :
9210 IF A=50 THEN RETURN
9220 PRINT#1,A$
9230 RETURN

```

Listing 20. Wichtige Routinen für relative Dateien

A\$="SUPERMIKE" : SA=34 : GOSUB 9000 : GOSUB 9200

Das Listing 21 stellt in Kombination mit den drei Unterprogrammen einen ganz einfachen Editor für relative Dateien dar.

Dieser Editor ist zum Arbeiten mit der Test-Datei gedacht. Eine Dokumentation ist überflüssig, da alles bereits erklärt worden ist.

Schreiben Sie nun einmal ein paar Sätze in die Test-Datei beziehungsweise lesen Sie einmal aus der Test-Datei. Sie werden schnell feststellen, daß beim Lesen oder Schreiben nicht immer auf Disk zugegriffen wird – das Diskettenlaufwerk verhält sich still. Dies hat seinen guten Grund: Die Daten werden nicht augenblicklich auf die Diskette geschrieben, sondern zunächst erst in einem festgelegten RAM-Bereich innerhalb der 1541 gepuffert. Erst zum Beispiel bei Überlauf wird dieser Puffer auf Disk gespeichert. Bei sehr kleinen Dateien wie unserer Test-Datei kann es also passieren, daß sich alles nur im Puffer abspielt. Deshalb ist es unbedingt notwendig, nach Beendigung des gesamten Schreib-Lesevorgangs beziehungsweise dem Arbeiten mit der relativen Datei, die Datei zu schließen – dann wird nämlich der Puffer gespeichert. Ansonsten kann es passieren, daß Daten verloren gehen. Zum Schließen der relativen Datei genügt CLOSE 15. Dies kann auch im Direktmodus durchgeführt werden – zum Beispiel dann, wenn sie mittels RUN/STOP den Editor abgebrochen haben. Also nie vergessen: Nach dem Beenden des Schreibens auf eine relative Datei IMMER die Datei mittels CLOSE 15 (beziehungsweise Floppykanal) schließen. Beim Schließen des Befehlskanals des Diskettenlaufwerks werden übrigens auch alle anderen Kanäle geschlossen. Die Datei muß also nicht extra geschlossen werden – das Schließen des Floppy-Befehlskanals (den man immer mit OPEN 15,8,15 öffnet und mit CLOSE 15 schließt) reicht vollkommen aus.

Probleme mit relativen Dateien

Bisher hörte sich alles ganz logisch und einfach an, ich möchte es jedoch keinesfalls versäumen, Sie auch auf die Probleme aufmerksam zu machen, die beim Arbeiten mit relativen Dateien immer wieder auftauchen und denen Anfänger oft hilflos gegenüberstehen. Besonders traurig ist die Tatsache, daß bei den folgenden Problemen auch die großen Bücher zur 1541 in keinsten Weise helfen!

```

10 OPEN 15,8,15 : REM FLOPPYKANAL OEFFNEN <108>
15 PRINT#15,"I0" <114>
20 OPEN 1,8,3,"TEST": REM RELATIVE DATEI O
   EFFNEN <027>
100 PRINT"LESEN, SCHREIBEN, ENDE ?" <070>
110 GET I$: IF I$<>"L" AND I$<>"S" AND I$<>"E"
   THEN 110 <188>
120 IF I$="L" THEN 1000 <245>
130 IF I$="S" THEN 2000 <134>
140 IF I$="E" THEN 3000 <141>
1000 REM LESEN----- <100>
1010 INPUT "SATZNUMMER ";SA <237>
1020 GOSUB 9000: REM POSITIONIEREN <244>
1030 GOSUB 9100: REM LESEN <066>
1040 PRINT A$ : REM ANZEIGEN <164>
1050 GOTO 100 <218>
2000 REM SCHREIBEN----- <121>
2010 INPUT "SATZNUMMER ";SA <221>
2020 INPUT A$ : REM EINGABESATZ <116>
2030 GOSUB 9000: REM POSITIONIEREN <238>
2040 GOSUB 9200: REM SCHREIBEN <252>
2050 GOTO 100 <202>
3000 REM ENDE----- <041>
3010 CLOSE 1:CLOSE 15 <128>
3020 END <228>

```

Listing 21. Einfacher Relativ-Editor

Besitzer des C128 haben es beim Umgang mit relativen Dateien schon etwas einfacher, denn ihnen steht ein komfortableres Basic als das des C64 zur Verfügung. Es gibt zwar Basic-Erweiterungen für den C64 (zum Beispiel DISK-Basic vom 5. Sonderheft der 64'er), die jedoch viel zu viel Speicher in Anspruch nehmen. Wenn man die einzelnen Probleme der relativen Dateiprogrammierung jedoch kennt, so kommt man auch ganz gut mit den drei Unterprogrammen von Listing 20 zurecht. Nun zu den Problemen beziehungsweise Schwierigkeiten:

Der Dateizeiger

Was beim Programmieren oft mißachtet wird, ist die Tatsache, daß der Dateizeiger (der auf jeweils einen Satz positioniert werden kann) der relativen Datei nicht ruht – er versetzt sich nach jedem Zugriff automatisch eine weiter.

Beispiel: Satz 5 soll geändert werden.

Zunächst wollen wir wissen, was eigentlich in Satz 5 steht. Also:

```
SA=5 : GOSUB 9000 : GOSUB 9100 : PRINT A$.
```

So schön, so gut, wir wissen jetzt, was in Satz 5 steht. Wo steht jedoch jetzt der Zeiger? Er hat sich automatisch um 1 erhöht. Das heißt, er zeigt jetzt auf Satz 6. Würde man nun erneut GOSUB 9100 : PRINT A\$ eingeben, so erhielte man den Inhalt von Satz 6 und der Zeiger würde auf Satz 7 verschoben werden.

Wenn wir also den Inhalt von Satz 5 ändern wollen, so genügt es NICHT

```
A$="neuer Inhalt" : GOSUB 9100
einzugeben.
```

Vielmehr muß es

```
A$="neuer Inhalt" : SA=5 : GOSUB 9000 : GOSUB 9200
lauten.
```

Es muß also vor dem Schreiben erst wieder positioniert werden. Denken Sie also stets daran, vor dem Schreiben immer auf den Satz zu positionieren, in den geschrieben werden soll.

Eine andere Aufgabe: Die Sätze 30 – 40 einer relativen Datei sollen gelesen und mittels PRINT ausgegeben werden. Die Lösung ist ganz einfach:

```

FOR SA=30 TO 40
: GOSUB 9000
: GOSUB 9100
: PRINT A$
NEXT SA

```

```

50000 REM INPUT-ROUTINE ----- <137>
50001 : <192>
50002 SL=40 <069>
50005 SL=40 <072>
50020 PRINT "{DOWN}": BE$="": POKE 198,0: POKE
   211,0: POKE 214,22: SYS 58732: PRINT "{
   YELLOW}" <156>
50030 GET X$: IF PEEK(203)=1 THEN 50120 <226>
50040 IF X$="" THEN 50030 <171>
50050 IF LEN(BE$)=0 AND ASC(X$)=20 THEN 5
   0030 <148>
50060 I=ASC(X$): IF I<32 OR I>133 AND I<159
   THEN IF I<>20 THEN 50030 <131>
50070 IF LEN(BE$)=SL AND I<>20 THEN 50030 <128>
50075 IF X$=" " THEN X$="{RIGHT}" <012>
50080 BE$=BE$+X$ <031>
50090 PRINT CHR$(20); X$; " "; <108>
50100 IF I=20 THEN BE$=LEFT$(BE$,LEN(BE$)-
   2): GOTO 50030 <101>
50110 GOTO 50030 <001>
50120 PRINT "{LEFT,SPACE}" <227>
50140 RETURN <157>

```

Listing 22. Mit diesem Unterprogramm wird der INPUT-Befehl kräftig aufgemöbelt

Diese Lösung funktioniert zwar perfekt, es geht jedoch auch viel einfacher und schneller:

```
SA=30 : GOSUB 9000
FOR I=30 TO 40
: GOSUB 9100
: PRINT A$
NEXT I
```

Die zweite Lösung, die vom automatischen Vorrücken des Dateizeigers Nutzen zieht, benötigt nur halb soviel Zeit, um die 10 Sätze auszugeben wie die erste Lösung. Es lohnt sich also stets, das automatische Vorrücken des Zeigers beim Programmieren zu beachten.

Steuerzeichen

Versuchen Sie einmal mittels dem Editor in den 10. Satz der TEST-Datei den Satz beziehungsweise den Inhalt »LOLO, PETER UND MIKE GINGEN IN DEN WALD« zu schreiben. Sie werden schnell merken, daß dies nicht geht, da die INPUT-Routine des C64 keine Kommas zuläßt – es kommt ein EXTRA IGNORED. Hier hilft nur eine verbesserte INPUT-Routine, die das Listing 22 darstellt:

Die Routine wird einfach mittels GOSUB 50000 aufgerufen. Im String BE\$ steht dann der Eingabesatz. Interessant im Listing ist lediglich Zeile 50020.

Mittels

POKE 211,SPALTE:POKE 214,ZEILE:SYS 58732
kann der Cursor an eine beliebige Stelle des Bildschirms positioniert werden.

In Zeile 50005 muß mittels der Variablen SL festgelegt werden, wie groß die maximale Länge des Eingabestrings sein darf.

Erweitern Sie bitte den Editor (Listing 21) um Listing 22.

Außerdem muß noch Zeile 2020 so geändert werden, daß die neue INPUT-Routine aufgerufen wird:

```
2020 GOSUB 50000 : A$=BE$
```

Jetzt können Sie »LOLO, PETER UND MIKE IN DEN WALD« in Satz 10 schreiben. Enttäuschung wird jedoch breit, wenn Sie anschließend Satz 10 wieder lesen wollen.

Sie erhalten beim Lesen nur »LOLO«. Alles, was nach dem Komma steht, einschließlich dem Komma ist verschwunden. Das Komma läßt sich innerhalb relativer Dateien also nicht so einfach verwenden, wie wir es gerne hätten. Wir können jedoch auch nicht auf das Komma verzichten, wenn wir Texte schreiben wollen.

Das gleiche Problem taucht bei Verwendung eines Doppelpunktes auf. Ganz zu schweigen von den ewigen Problemen mit den Gänsefüßchen beziehungsweise Anführungszeichen innerhalb von Strings.

Wie bei den meisten Problemen gibt es allerdings auch hier eine einfache Lösung. Der Editor wird einfach um folgende Zeile ergänzt:

```
2035 A$=CHR$(34)+A$+CHR$(34)
```

Bevor A\$ abgespeichert wird, wird es also einfach in Anführungszeichen gesetzt. Jetzt können alle Sonderzeichen verwendet werden. Auch beim Einlesen bereiten die beiden Anführungszeichen zu Beginn und am Ende von A\$ keine Schwierigkeiten – sie tauchen nicht auf. Die Sache hat wie immer natürlich ihren Haken: Die maximal erlaubte Satzlänge beträgt jetzt nicht mehr 40, sondern nur noch 38 Zeichen, da die Anführungszeichen zwei Zeichen Platz benötigen. Es wird LOLO, PETER UND MIKE also wieder nicht gelingen, endlich in den Wald zu gelangen. Abhilfe würde hier ganz einfach das Anlegen einer neuen TEST-Datei mit 42 Zeichen Satzlänge schaffen.

Hier sind jedoch auch Überlegungen dem Tun voranzustellen. Nehmen wir einmal an, auf einer Diskette wäre Platz für maximal 1000 Sätze mit je 40 Zeichen. Da wir jedoch 42 brauchen, um Strings mit 40 Zeichen darzustellen, würden wir nur noch zirka 952 Sätze auf der Diskette unterbringen. Unsere jetzige Lösung ist somit zwar einfach, jedoch nicht

sehr effektiv, da sie viel zu verschwenderisch mit dem Disk-Speichervolumen umgeht.

Sicherlich werden sich einige von Ihnen daran erinnern, daß ich im letzten Kapitel behauptete, daß ein deutscher Zeichensatz auch enorme Vorteile beim Programmieren von relativen Dateien mit sich bringt. Einige haben sich sicherlich auch schon gewundert, warum ich neben den deutschen Umlauten und »ß« auch noch »"«, »«, »:« und »;« nochmals definiert habe.

Komma ist nicht gleich Komma

Wenn wir unseren deutschen Zeichensatz im Computer haben, so können wir am Aussehen des Kommas auf dem Bildschirm nicht unterscheiden, ob es sich um CHR\$(44) (dem Originalkomma) oder um CHR\$(95) (Pfeil nach links, der als Komma umdefiniert wurde) handelt. Unser Computer hingegen kann es.

Geben Sie im Direktmodus einmal folgendes ein:

```
PRINT " " " " und drücken RETURN
```

Als Antwort erscheint nichts.

Geben Sie nun

```
PRINT " " " " ein.
```

Das mittlere Anführungszeichen müssen Sie allerdings durch Drücken der @-Taste eingeben (also das definierte Anführungszeichen).

Diesmal erscheint als Antwort tatsächlich ein Anführungszeichen. Der umdefinierte Zeichensatz macht's möglich! Erweitern Sie den Editor bitte um folgende Zeilen:

```
1 IFI=0THENI=1:LOAD"D.Z.SATZ",8,1
```

```
2 POKE648,204:POKE53272,56:POKE56576,0:
```

Jetzt wird zu Beginn des Programmes der deutsche Zeichensatz (den Sie hoffentlich bereits mit dem Zeicheneditor automatisch erzeugt und als D.Z.SATZ auf Diskette gespeichert haben!) geladen.

Auf Zeile 2035 können wir nun verzichten – wir brauchen die Anführungszeichen links und rechts von A\$ nicht mehr. Das heißt, in einer Datei mit Satzlänge von 40 Zeichen können jetzt auch wirklich Sätze mit 40 und nicht mit 38 Zeichen untergebracht werden.

Jetzt können LOLO, PETER UND MIKE auch endlich in den Wald gehen (natürlich ohne das Bier zu vergessen). Auf die Frage, was LOLO, PETER UND MIKE eigentlich überhaupt im Wald wollen beziehungsweise wozu sie das Bier mitgenommen haben, werde ich später noch zu sprechen kommen!

Bei Eingabe von Sätzen dürfen Sie von nun an jedoch nicht vergessen, daß die Tastatur in Bezug auf Komma, Doppelpunkt und Anführungszeichen nicht mehr stimmt – wenn Sie Texte für relative Dateien schreiben wollen. Anstelle der Komma-Taste muß jetzt die Pfeil-links-Taste betätigt werden, wenn ein »Pseudokomma« gewünscht wird etc. Sie werden sich sicher schnell mit der neuen Tastaturbelegung und dem deutschen Zeichensatz anfreunden.

Durch die enormen Vorteile, die sich uns jetzt bieten, gibt es wirklich kein Argument mehr, das gegen die Verwendung eines deutschen Zeichensatzes spricht.

Wenn man sich für die Verwendung des deutschen Zeichensatzes anstelle der Lösung mit den beiden Anführungszeichen entscheidet, müssen noch einige Dinge berücksichtigt werden:

SPACES

Spaces – Leerräume am Anfang eines Satzes sind nicht ohne weiteres möglich. Angenommen, wir schreiben in Satz 30 den String »HALLO«. Lesen wir den Satz 30 anschließend, so erhalten wir die Antwort »HALLO«. Die 10 Spaces werden vom Computer zu Beginn des Satzes einfach ignoriert. Es ist jedoch unsinnig, ein neues Space zu definieren, auch wenn dies eine Lösung wäre. Wir verwenden anstelle

von Space (CHR\$(32) und CHR\$(160)) einfach CHR\$(29) – Cursor nach rechts. Hierfür sorgt Zeile 50075 im Editor. Zu beachten ist hierbei, daß, wenn später mit Strings gearbeitet werden soll, die aus einer mit dieser Methode geführten relativen Datei stammen, die man zum Auffinden von Wortzwischenräumen nicht nach »«, sondern nach CHR\$(29) suchen muß!!!

Im Gegensatz zu »«, »:« und »"« kann zum Erzeugen eines Spaces jedoch ruhig die Space-Taste gedrückt werden (und nicht die Cursor-nach-rechts-Taste). Verwirrt? Probieren Sie einfach mal alles am Computer aus, dann wird alles schnell klar!

Leersätze

Versuchen Sie doch mal einfach, einen Leersatz zu schreiben. Also einfach RETURN drücken. Soll der Satz wieder gelesen werden, so steigt der Computer aus – er findet nichts mehr und verhält sich still. Es ist nicht möglich, in relative Dateien Leersätze " " zu schreiben, die nicht in Anführungszeichen stehen. Da wir uns jedoch für die modifizierte Zeichensatz-Lösung entschieden haben, müssen wir erneut einen kleinen Kunstgriff machen: Wir schreiben anstelle von Leersätzen einfach Sätze, die mit einem Doppelpunkt beginnen, denn wir haben ja erfahren, daß alles, was nach einem Komma beziehungsweise Doppelpunkt steht, als nichtig interpretiert wird. Wir müssen den Editor nur um folgende Zeile ergänzen:

```
50130 IF BE$="" THEN BE$=":"
```

Damit wäre auch schon das Wichtigste über relative Dateien gesagt. Wer die Vorzüge dieser Dateien erst einmal schätzen gelernt hat, wird sie schon bald für unentbehrlich halten.

Einschränkung

Das Disketten-Laufwerk 1541 hat nur drei Datenkanäle zur Verfügung. Eine relative Datei beansprucht zwei dieser Kanäle, das heißt, es kann jeweils nur eine relative Datei geöffnet sein. Will man mit zwei oder mehreren relativen Dateien arbeiten, so muß beachtet werden, daß immer die eine Datei zuerst geschlossen werden muß, wenn man eine andere öffnen will. Die Datenübertragung von einer relativen Datei zu einer anderen ist somit nur mittels eines Puffers möglich.

Neben einer relativen Datei kann auch noch eine sequentielle Datei geöffnet werden, da diese Dateiart nur einen Kanal benötigt. Man kann also maximal drei sequentielle Dateien oder eine relative und eine sequentielle gleichzeitig geöffnet haben.

Der Relative-Dateien-Editor

Im folgenden stelle ich Ihnen meinen Relative-Dateien-Editor vor, der uns für den Rest des Kurses als unentbehrliches Werkzeug dienen wird.

Der Editor hat sich im Laufe der Monate allmählich aus dem Editor entwickelt. Er ist keineswegs so komfortabel wie ein Textverarbeitungssystem, was bei Relative-Datei-Editoren auch nie der Fall sein wird beim C64. Ich habe großen Wert auf Anwenderfreundlichkeit des Programmes gelegt: Der Editor hat nur eine einzige Menüzeile, von der sich jederzeit alle nur erdenklichen Zwischenmenüs aufrufen lassen. Der Editor eignet sich hervorragend zum Erstellen von Texten von Adventure-Spielen. Er eignet sich auch prima zum Erstellen von Wortschätzen – wichtig für Parsertechnik, wie wir bald sehen werden. Der Editor arbeitet auch zusammen mit dem deutschen Zeichensatz. Am besten ist es, wenn Sie den Editor gemeinsam mit dem Zeichensatzeditor und dem deutschen Zeichensatz auf eine Diskette speichern. Im Laufe des Kurses werde ich Ihnen noch weitere Programme für diese Utility-Disk vorstellen. Zunächst sollten Sie jedoch erst einmal das Listing 23 zum Editor abtippen:

```
0 REM ***** <131>
1 REM * TEXTEDITOR V. III.4 * <052>
2 REM * <C> 1985 * <177>
3 REM * VON MICHAEL NICKLES * <213>
4 REM ***** <135>
5 IF IM=1 THEN POKE 648,204:POKE 53272,56: <217>
   POKE 56576,0:PRINT "{CLR}":GOTO 20
6 POKE 53280,0:POKE 53281,0:PRINT "{CLR,WHI <053>
   TE}";
7 MZ=300:DIM M$(MZ),M(MZ):REM PUFFER <065>
8 DIM LG(100),RG(100):REM Q.SORT <062>
9 REM ANFANGSMENUE ----- <094>
10 PRINT "{CLR,RVSON,8SPACE}TEXTEDITOR - VE <159>
   RSION 3(10SPACE,RVOFF)";
11 PRINT "{RVSON,4SPACE}<C> 1985 VON MICHAEL <196>
   NICKLES(8SPACE,RVOFF)";
12 PRINT "{DOWN,5SPACE}ALTE DATEI EDITIERE <080>
   N(2SPACE)-(2SPACE)'A'";
13 PRINT "{5SPACE}NEUE DATEI EROEFFNEN(2SPA <188>
   CE)-(2SPACE)'N'";
14 PRINT "{DOWN,5SPACE}D.Z.SATZ LADEN(8SPAC <241>
   E)-(2SPACE)'D'";
15 GET I$:IF PEEK(203)=1 THEN I$="A" <101>
16 IF I$<>"A"AND I$<>"N"AND I$<>"D"THEN 25 <098>
17 IF I$="D"THEN IM=1:LOAD"D.Z.SATZ",8,1 <110>
18 POKE 198,0:INPUT "{DOWN}DATEINAME(2SPAC <087>
   E)*{3LEFT}";D$:POKE 198,0
19 INPUT"SATZLAENGE(2SPACE)40(4LEFT)";SL:I <200>
   F SL<1 OR SL>254 THEN PRINT"GEHT NICHT <024>
   !":GOTO 30
20 IF I$="A"THEN 40
21 INPUT"SATZANZAHL ";SN:HB=INT(SN/255):LB <094>
   =SN-HB*255
22 PRINT"BITTE LEERE DISK EINLEGEN (RETURN <092>
   )":POKE 198,0:WAIT 198,1
23 OPEN 15,8,15 <191>
24 PRINT#15,"I0":OPEN 1,8,3,D$:"+L,"+CHR$( <237>
   SL)
25 PRINT#15,"P";CHR$(3);CHR$(LB);CHR$(HB); <051>
   CHR$(1)
26 PRINT#1,CHR$(255) <048>
27 CLOSE 1:CLOSE 15 <206>
28 OPEN 15,8,15 <198>
29 PRINT#15,"I0":OPEN 1,8,3,D$:PRINT <191>
30 REM ZWISCHENMENUE ----- <253>
31 PRINT "{RVSON}S / L / V / O / D / I / Z <011>
   / F / ENDE(4SPACE,RVOFF)";
32 GET I$:IF I$=""THEN 52 <225>
33 IF I$="E"THEN PRINT "{CLR}":GOTO 9900 <034>
34 IF I$="V"THEN PRINT "{CLR}":GOTO 6000 <130>
35 IF I$="O"THEN PRINT "{CLR}":GOTO 10000 <251>
36 IF I$="D"THEN PRINT "{CLR}":GOTO 7000 <125>
37 IF I$="I"THEN PRINT "{CLR}":GOTO 30000 <254>
38 IF I$="Z"THEN PRINT "{CLR}":GOTO 8000 <140>
39 IF I$="F"THEN PRINT "{CLR}":GOTO 61000 <137>
40 IF I$<>"S"AND I$<>"L"THEN 52 <183>
41 INPUT"SATZNR";SA:IF SA<0 OR SA>63000 TH <064>
   EN 60
42 IF I$="L"THEN PRINT "{CLR}":GOTO 2000 <127>
43 IF I$="S"THEN PRINT "{CLR}":MO=1:GOTO 3 <089>
   000
44 GOTO 50 <024>
45 REM LESEN ----- <193>
46 - <148>
47 2001:PRINT "{CLR,RVSON}";SA;" {LEFT}-";STR$ <247>
   (SA+20);" {SPACE,RVOFF}"
48 2005 IA=SA <154>
49 2007 FOR SA=IA TO IA+20 <196>
50 2010 :GOSUB 9000:GOSUB 9100 <162>
51 2025 :IF A=50 THEN SA=IA+21:GOTO 2045 <211>
52 2030 :IF LEN(A$)<40 THEN PRINT A$ <118>
53 2035 :IF LEN(A$)>=40 THEN PRINT A$; <127>
54 2045 NEXT:PRINT
55 2046 IF A=50 THEN PRINT"SATZNUMMER ZU HOCH <243>
   !"
56 2050 GOTO 50 <228>
57 3000 REM SCHREIBEN ----- <214>
58 - <236>
59 3005 PRINT"EINGABESATZ(2SPACE)//=KEINE EIN <202>
   GABE"
60 3010 GOSUB 9000 :REM POSITIONIEREN
61 Listing 23. Der Relative-Dateien-Editor
```

```

3020 IF A=50 THEN PRINT "SATZNUMMER ZU HOCH
!":GOTO 50 :REM FEHLERKAN <130>
ALABFR.
3030 GOSUB 50000:A$=BE$ :REM INPUT-ROUT. <181>
3040 IF A$="//" THEN MO=0:GOTO 50:REM ENDE <241>
3050 GOSUB 9200 :REM A$ SCHREIBEN <083>
3060 SA=SA+1 :REM NAECHSTER SATZ <225>
3070 GOTO 3010 :REM NEXT (LOOP) <190>
6000 REM VERSETZEN -----
- <154>
6002 PRINT {CLR,RVSON}VERSETZEN{RVOFF}" <195>
6010 PRINT {DOWN}BITTE GEBEN SIE ANFANG UN
D ENDE DES BE- REICHES AN, DEN SIE "; <209>
6015 PRINT "VERSETZEN WOLLEN !" <239>
6020 POKE 198,0:INPUT {DOWN}ANFANG";AN <211>
6021 POKE 198,0:INPUT "ENDE";EN:IF AN>EN TH
EN PRINT "DAS IST UNSINN !":GOTO 6020 <113>
6023 SA=AN:GOSUB 9000:IF A=50 THEN PRINT "A
NFANGSSATZ EXISTIERT NICHT!":GOTO 50 <229>
6024 SA=EN:GOSUB 9000:IF A=50 THEN PRINT "E
NDSATZ ZU HOCH!":GOTO 50 <152>
6026 PRINT {DOWN}QUELDATEI = ";D$ <007>
6028 POKE 198,0:INPUT "ZIELDATEI ";Z$:IF Z$
="" THEN Z$=D$:PRINT {UP,13RIGHT}";D$ <098>
6029 PRINT {DOWN}BITTE ANFANG DES NEUEN BE
REICHS ANGEBEN!";POKE 198,0:INPUT NB
:PRINT <193>
6030 SA=NB:GOSUB 9000:IF A=50 THEN PRINT "N
EUER ANFANGSSATZ ZU HOCH!":GOTO 50 <198>
6031 SA=NB+EN-AN:GOSUB 9000:IF A=50 THEN P
RINT "BEREICH ZU GROSS!":GOTO 50 <240>
6032 PRINT "BITTE WARTEN!" <125>
6036 SA=AN:IM=0:IF AN=EN THEN PRINT "*** FE
RTIG ***":POKE 198,0:WAIT 198,1:GOTO
50 <113>
6038 GOSUB 9000:GOSUB 9100:IF A=50 THEN 61
00 <203>
6040 M$(SA-AN)=A$ <219>
6045 IF SA=EN THEN 6100 <208>
6050 IF SA-AN=M$ THEN IM=1:GOTO 6100:REM * <108>
6060 SA=SA+1:GOTO 6038 <094>
6100 REM TRANSPORT <068>
6110 PRINT "BITTE DISK MIT ZIELDATEI EINLEG
EN !":POKE 198,0:WAIT 198,1 <204>
6120 CLOSE 1:PRINT#15,"I0":OPEN 1,8,3,Z$ <182>
6200 SA=NB:GOSUB 9000:IF A=50 THEN 50 <222>
6205 II=EN-AN:IF IM=1 THEN II=M$:REM * <163>
6215 FOR I=0 TO II <014>
6220 A$=M$(I):IF A$="" THEN A$=":" <108>
6225 GOSUB 9200:REM SCHREIBEN <119>
6240 SA=SA+1:NEXT I:IF IM<0 THEN 6260 <135>
6250 PRINT "BITTE DISK MIT QUELDATEI EINLE
GEN !":POKE 198,0:WAIT 198,1:GOTO 50 <185>
6260 AN=AN+M$:NB=NB+M$:PRINT "BITTE DISK MI
T QUELDATEI EINLEGEN !" <248>
6270 POKE 198,0:WAIT 198,1:CLOSE 1:PRINT#1
5,"I0":OPEN 1,8,3,D$:GOTO 6036:REM * <096>
7000 REM PRINTER-ROUTINE -----
- <065>
7001 PRINT {CLR,RVSON}PRINTER-ROUTINE MPS
801{RVOFF} <097>
7002 INPUT {DOWN}ANFANG";AN:INPUT "ENDE";EN
:IF AN>EN THEN 7000 <176>
7005 SA=AN:GOSUB 9000:IF A=50 THEN PRINT "A
NFANGSSATZ EXISTIERT NICHT!":GOTO 50 <195>
7006 SA=EN:GOSUB 9000:IF A=50 THEN PRINT "E
NDSATZ ZU HOCH!":GOTO 50 <118>
7009 AN=AN-1:PRINT {DOWN}BITTE WAEHLEN SIE.
: {DOWN} <025>
7010 PRINT,"NORMALE AUSGABE ..1 <248>
7011 PRINT,"D.Z.S. AUSGABE ...2 <165>
7012 PRINT,"EIGENE ROUTINE ...3 <072>
7013 GET I$:IF VAL(I$)<1 OR VAL(I$)>3 THEN
7013 <176>
7014 IF I$="2" THEN 7100 <044>
7015 IF I$="3" THEN 7500 <205>
7020 REM NORMALE AUSGABE ----- <016>
7021 OPEN 3,4,7:FOR SA=AN TO EN:GOSUB 9000
:GOSUB 9100 <049>
7025 IF A=50 THEN PRINT "SATZNUMMER ZU HOCH
!":SA=EN+1:GOTO 7045 <175>
7035 PRINT#3,RIGHT$(" {3SPACE}" +STR$(SA),3)
;" {4SPACE}"; <085>

```

```

7036 FOR I=1 TO LEN(A$) <224>
7037 :IF MID$(A$,I,1)="(RIGHT)" THEN PRINT#
3," ";:GOTO 7040 <137>
7038 :PRINT#3,MID$(A$,I,1); <048>
7040 NEXT I:PRINT#3 <140>
7045 NEXT SA:CLOSE 3:PRINT:GOTO 50 <117>
7100 REM D.Z. SATZ AUSGABE ----- <230>
7101 OPEN 3,4,7:FOR SA=AN TO EN:GOSUB 9000
:GOSUB 9100 <129>
7102 IF A=50 THEN SA=EN+1:PRINT "SATZNUMMER
ZU HOCH!":GOTO 7170 <022>
7110 PRINT#3,RIGHT$(" {4SPACE}" +STR$(SA),4)
;" {4SPACE}"; <161>
7120 FOR I=1 TO LEN(A$):IF A$="" THEN 7160 <163>
7125 IM=ASC(MID$(A$,I,1)) <201>
7126 IF IM>64 AND IM<91 OR IM>96 AND IM<12
3 THEN 7160 <065>
7130 IF IM=64 THEN PRINT#3,CHR$(34);:GOTO
7170 <022>
7135 IF IM=173 THEN PRINT#3,"BE";:GOTO 717
0 <125>
7140 IF IM=174 THEN PRINT#3,"SS";:GOTO 717
0 <238>
7145 IF IM=176 THEN PRINT#3,"AE";:GOTO 717
0 <106>
7150 IF IM=181 THEN PRINT#3,"UE";:GOTO 717
0 <016>
7152 IF IM=182 THEN PRINT#3,"DE";:GOTO 717
0 <145>
7154 IF IM=184 THEN PRINT#3,"UE";:GOTO 717
0 <247>
7156 IF IM=185 THEN PRINT#3,"OE";:GOTO 717
0 <120>
7158 IF IM=94 THEN PRINT#3,":"; :GOTO 717
0 <244>
7159 IF IM=95 THEN PRINT#3,","; :GOTO 717
0 <179>
7160 PRINT#3,CHR$(IM); <028>
7170 NEXT I:PRINT#3,CHR$(13); <147>
7175 NEXT SA:CLOSE 3:PRINT:GOTO 50 <249>
7500 REM EIGENE ROUTINE ----- <183>
7510 PRINT {DOWN}SIE HABEN NOCH KEINE EIGE
NE PRINTER-{4SPACE}ROUTINE DEFFINIERT
!":GOTO 50 <032>
8000 REM TEXT ZENTRIEREN -----
- <058>
8001 I$="":FOR I=1 TO 10:I$=I$+" {20RIGHT}"
:NEXT I <067>
8002 REM SL=27 <234>
8005 REM DIMM$(300):GOTO 8100 <009>
8010 PRINT {RVSON}ZENTRIEREN{RVOFF,DOWN}" <040>
8015 PRINT "BITTE GEBEN SIE ANFANG UND ENDE
DES{5SPACE}BEREICHES AN, DER ZENTRIE
RT"; <197>
8017 PRINT "WERDEN SOLL!" <039>
8020 INPUT "ANFANG";AN:INPUT "ENDE";EN:IF AN
>EN THEN 50 <157>
8025 SA=AN:GOSUB 9000:IF A=50 THEN PRINT "A
NFANGSSATZ EXISTIERT NICHT!":GOTO 50 <199>
8030 SA=EN:GOSUB 9000:IF A=50 THEN PRINT "E
NDSATZ ZU HOCH!":GOTO 50 <126>
8040 PRINT {DOWN}LINKS-RECHTS BUENDIG ZENT
RIEREN{2SPACE}- '1' <098>
8045 PRINT {DOWN}IN DIE MITTE ZENTRIEREN{1
0SPACE}- '2' <045>
8050 GET II$:IF II$<>"1" AND II$<>"2" THEN 8
050 <115>
8060 IF II$="2" THEN 8600 <081>
8099 REM LINKS-RECHTS BUENDIG ----- <089>
8100 PRINT {DOWN}BITTE WARTEN!" <255>
8105 FOR SA=AN TO EN <207>
8110 :GOSUB 9000:GOSUB 9100:PRINT A$:PRIN
T LEN(A$) <043>
8120 :IF RIGHT$(A$,1)=CHR$(29) THEN A$=LEF
T$(A$,LEN(A$)-1):GOTO 8120 <192>
8122 :IF RIGHT$(A$,1)=CHR$(32) THEN A$=LEF
T$(A$,LEN(A$)-1):GOTO 8122 <128>
8123 :IF RIGHT$(A$,1)=CHR$(160) THEN A$=LE
FT$(A$,LEN(A$)-1):GOTO 8123 <078>
8130 :IF LEN(A$)=SL THEN 8520 <059>
8200 :LU=SL-LEN(A$):PRINT "LU=";LU <233>
8220 :I=1:IC=0:REM BEGIN LOOP ----- <026>
8222 : <146>
8230 :IF MID$(A$,I,1)="(OR MID$(A$,I+1,
1)=" THEN 8530 <222>

```

```

8231 : IM=ASC(MID$(A$,I,1)) <089>
8232 : IN=ASC(MID$(A$,I+1,1)) <178>
8240 : IF IM<>29 AND IM<>32 AND IM<>160 <208>
      THEN 8500 <056>
8250 : IF IN=29 OR IN=32 OR IN=160 THEN <164>
      8500 <058>
8260 : A$=LEFT$(A$,I)+" "+RIGHT$(A$,LEN <028>
      (A$)-I) <167>
8261 : IF MID$(A$,I,1)=" "OR MID$(A$,I,1 <177>
      )=" "RIGHT$ THEN I=I+1:GOTO 8261 <068>
8265 : LU=LU-1:IF LU=0 THEN 8510 <101>
8266 : IC=IC+1 <186>
8270 : I=I+1 <035>
8400 : <006>
8500 I=I+1 <076>
8505 IF I<LEN(A$) THEN 8230 <112>
8510 IF I=LEN(A$) AND LU>0 AND IC<>0 THEN B <133>
      220 <001>
8520 GOSUB 9000:GOSUB 9200 <214>
8530 NEXT SA <212>
8540 GOTO 50 <146>
8600 REM IN DIE MITTE ----- <128>
8610 PRINT"(DOWN)BITTE WARTEN!" <234>
8620 FOR SA=AN TO EN <247>
8630 :GOSUB 9000:GOSUB 9100 <004>
8635 IF RIGHT$(A$,1)=CHR$(29) THEN A$=LEFT$ <226>
      (A$,LEN(A$)-1):GOTO 8635 <006>
8636 IF RIGHT$(A$,1)=CHR$(32) THEN A$=LEFT$ <130>
      (A$,LEN(A$)-1):GOTO 8636 <068>
8637 IF RIGHT$(A$,1)=CHR$(160) THEN A$=LEFT <005>
      $(A$,LEN(A$)-1):GOTO 8637 <079>
8640 : IF LEN(A$)=SL THEN 8680 <208>
8650 : AS=SL-LEN(A$): REM ANZAHL SPACES <042>
8660 : A$=LEFT$(I$,AS/2)+A$ <154>
8670 :GOSUB 9000:GOSUB 9200 <083>
8680 NEXT SA <013>
8690 GOTO 50 <106>
9000 REM POSITIONIEREN ***** <182>
9010 HB%=SA/256:LB%=SA-HB%*256 <205>
9020 PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$(H <042>
      B%);CHR$(1) <154>
9030 INPUT#15,A <083>
9040 RETURN <013>
9100 REM SATZ EINLESEN ***** <106>
9110 IF A=50 THEN RETURN <182>
9120 INPUT#1,A$ <205>
9140 RETURN <042>
9200 REM SATZ SCHREIBEN ***** <154>
9210 IF A=50 THEN RETURN <083>
9211 IF A$="" THEN A$="":REM ZUR SICHERHEI <013>
      T !!! <106>
9220 PRINT#1,A$ <182>
9240 RETURN <205>
9900 REM BEENDEN ***** <042>
9910 CLOSE 1 <154>
9920 CLOSE 15 <083>
9930 PRINT "++ ENDE ++" <013>
9940 END <106>
10000 REM SORT-ALGORITHMUS ----- <182>
      -- <205>
10010 PRINT"(CLR,RVSON)DATEI SORTIEREN(RVO <042>
      FF)" <154>
10100 PRINT"BITTE GEBEN SIE ANFANG UND END <083>
      E DES ZU(2SPACE)SORTIERENDEN BEREICH <013>
      S AN !" <106>
10110 INPUT"ANFANG";AN <182>
10120 INPUT"ENDE"; EN <205>
10122 SA=AN:GOSUB 9000:IF A=50 THEN PRINT" <042>
      ANFANGSSATZ EXISTIERT NICHT!":GOTO 5 <154>
      0 <083>
10124 SA=EN:GOSUB 9000:IF A=50 THEN PRINT" <013>
      ENDSATZ ZU HOCH!":GOTO 50 <106>
10130 IF EN-AN>M$ THEN PRINT"BEREICH ZU GRO <182>
      SS!":GOTO 50 <205>
10135 PRINT"(DOWN)BITTE MARKIEREN SIE DEN <042>
      BEREICH INNER- <154>
10136 PRINT"HALB DER STRINGS NACH DEM DIE <083>
      SORTIERUNG"; <013>
10137 PRINT"ERFOLGEN SOLL!" <106>
10138 SA=AN:GOSUB 9000:GOSUB 9100:IF A=50 <182>
      THEN PRINT"SATZ EXISTIERT NICHT!":GO <205>
      TO 50 <106>
10139 IG=-1:GOSUB 60000:IG=0 <182>
10140 IC=0:FOR SA=AN TO EN:IC=IC+1 <205>

```

```

10145 :GOSUB 9000:GOSUB 9100:IF A=50 THEN <048>
      PRINT"SATZNUMMER ZU HOCH!":SA=EN+1:G <172>
      OTO 1015 <053>
10146 :M$(IC)=A$ <047>
10150 NEXT SA <023>
10180 Z=0:LG(1)=1:RG(1)=EN-AN+1 <084>
10190 GOSUB 10200:GOTO 10400 <234>
10200 REM QUICKSORT ----- <069>
10210 Z=Z+1:IF LG(Z)>=RG(Z) THEN 10350 <095>
10220 X=LG(Z):Y=RG(Z) <008>
10230 VG$=MID$(M$(INT((X+Y)/2)),MA(1),MB(1 <143>
      )-MA(1)+1) <116>
10240 IF X>Y THEN 10330 <020>
10250 IF MID$(M$(X),MA(1),MB(1)-MA(1)+1)<V <196>
      G$ THEN X=X+1:GOTO 10250 <205>
10260 IF MID$(M$(Y),MA(1),MB(1)-MA(1)+1)>V <132>
      G$ THEN Y=Y-1:GOTO 10260 <253>
10270 IF X>Y THEN 10330 <226>
10280 S$=M$(X):M$(X)=M$(Y):M$(Y)=S$ <198>
10290 X=X+1:Y=Y-1:GOTO 10240 <040>
10330 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 10210 <195>
10340 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 10210 <010>
10350 Z=Z-1:RETURN <162>
10400 REM ABSPEICHERN <008>
10410 IC=0:FOR SA=AN TO EN:IC=IC+1 <143>
10420 :GOSUB 9000:A$=M$(IC):GOSUB 9200 <116>
10430 NEXT SA <020>
10500 GOTO 50 <196>
20000 REM KORREKTUR-ROUTINE ----- <205>
      -- <132>
20001 FOR I=0 TO LEN(BE$):PRINT CHR$(20);: <253>
      NEXT I:BE$="" <226>
20010 SA=SA+SC : REM TEXTZEIGER <198>
20015 IF SA<0 THEN SA=0 : REM SPERRE <040>
20020 GOSUB 9000 : REM POSITIONIEREN <196>
20025 IF A=50 THEN SA=SA-SC:GOTO 20020 <223>
20040 INPUT#1,A$ <114>
20060 RETURN <050>
30000 REM DIRECTORY EINLESEN ----- <161>
      -- <199>
30010 PRINT"(CLR,RVSON)INHALTSVERZEICHNIS: <007>
      (DOWN,RVOFF)" <221>
30110 PRINT#15,"10":OPEN 2,8,2,"#" <021>
30120 T=18:S=1 <214>
30130 PRINT#15,"U1";2;0:T;S <054>
30140 PRINT#15,"B-P";2;0 <098>
30150 GET#2,X$:IF X$="" THEN X$=CHR$(0) <074>
30160 T=ASC(X$) <114>
30170 GET#2,X$:IF X$="" THEN X$=CHR$(0) <149>
30180 S=ASC(X$) <060>
30190 FOR X=0 TO 7 <098>
30192 PRINT#15,"B-P";2;X*32+2 <092>
30194 GET#2,X$:IF X$="" THEN X$=CHR$(0) <106>
30196 IF ASC(X$)=0 THEN 30290 <239>
30200 PRINT#15,"B-P";2;X*32+5 <136>
30210 FF$="" <191>
30220 FOR Y=0 TO 15 <168>
30230 GET#2,X$:IF X$="" THEN X$=CHR$(0) <244>
30240 IF ASC(X$)=160 THEN 30270 <166>
30250 FF$=FF$+X$ <012>
30260 NEXT Y <010>
30270 IF A=0 THEN A=1:PRINT FF$;:GOTO 3029 <091>
      0 <059>
30280 A=0:PRINT TAB(19);"R ";FF$ <056>
30290 NEXT X <106>
30300 IF T<>0 THEN 30130 <097>
30310 CLOSE 2:PRINT <085>
30330 GOTO 50 <168>
50000 REM TEXTEINGABE ----- <069>
      -- <204>
50002 IF AR=1 THEN AR=0:A$="(RVSON)SATZNUM <173>
      MER ZU HOCH! (RVOFF)":GOTO 50007 <045>
50006 GOSUB 9000:GOSUB 9100 <097>
50007 PRINT"(HOME,40SPACE)";:PRINT"(HOME,R <085>
      VSON)SATZ";SA;"(RVOFF)" <168>
50008 PRINT"(40SPACE,UP)";A$ <069>
50009 PRINT"TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT <204>
      TTTTTTTTTT"; <173>
50010 PRINT"(DOWN)" <045>
50020 BE$="":POKE 198,0:POKE 211,0:POKE 21 <045>
      4,22:SYS 58732:PRINT"(YELLOW)";

```

Listing 23. Der Relative-Dateien-Editor (Fortsetzung)

```

50030 GET X$:IF PEEK(203)=1 THEN 50120 <226>
50035 IF PEEK(203)=7 AND PEEK(653)=0 THEN
SC=1 : GOSUB 20000 : GOTO 50000 <023>
50037 IF PEEK(203)=7 AND PEEK(653)=1 THEN
SC=-1 : GOSUB 20000 : GOTO 50000 <031>
50040 IF X$="" THEN 50030 <171>
50050 IF LEN(BE$)=0 AND ASC(X$)=20 THEN 50
030 <148>
50060 I=ASC(X$):IF I<32 OR I>133 AND I<159
THEN IF I<>20 THEN 50030 <131>
50070 IF LEN(BE$)=SL AND I<>20 THEN 50030 <128>
50075 IF X$=" " THEN X$="(RIGHT)" <012>
50080 BE$=BE$+X$ <031>
50090 PRINT CHR$(20);X$;" "; <108>
50100 IF I=20 THEN BE$=LEFT$(BE$,LEN(BE$)-
2):GOTO 50030 <101>
50110 GOTO 50030 <001>
50120 PRINT "{LEFT,SPACE}":GOSUB 9000 <017>
50130 IF BE$="" THEN BE$=":" <049>
50140 PRINT "{HOME,24DOWN}":RETURN <030>
60000 REM MARKIEREN WIE DATIERT WIRD -----
-- <071>
60010 I$=A$:FOR I=1 TO LEN(I$):M$(I)="0":N
EXT <131>
60020 IM=1:PRINT <131>
60025 PRINT I$:FOR I=1 TO LEN(I$):PRINT M$
(I);:NEXT:PRINT <223>
60030 PRINT "{DOWN}EDIT {3SPACE}>";MID$(I$,I
M,1);"< {4SPACE,RVSON}"IM {RVOFF}" <232>
60035 I=PEEK(203):IF I=64 THEN 60035 <110>
60040 IF I=2 AND PEEK(653)=0 AND IM<>LEN(I
$) THEN IM=IM+1:GOTO 60070 <174>
60042 IF I=2 AND PEEK(653)=1 AND IM<>1 THE
N IM=IM-1:GOTO 60070 <119>

60045 IF I=56 THEN M$(IM)="1":GOTO 60070 <041>
60055 IF I=35 THEN M$(IM)="0":GOTO 60070 <178>
60056 IF I=1 THEN 60100 <013>
60060 GOTO 60035 <183>
60070 PRINT "{UP,25SPACE,25LEFT,3UP}";:GOTO
60025 <139>
60100 REM UMRECHNEN <252>
60102 FOR I=1 TO 3:MA(I)=0:MB(I)=0:NEXT I <148>
60105 FOR II=1 TO 3 <169>
60110 :FOR I=1 TO LEN(I$):IF MA(II)=0 AND
M$(I)=RIGHT$(STR$(II),1) THEN MA(II)=
I <059>
60113 NEXT I <250>
60115 :IF MA(II)=0 THEN 60150 <245>
60120 : MB(II)=MA(II) <124>
60125 : MB(II)=MB(II)+1:REM PRINT,MB(II) <069>
60130 : IF M$(MB(II))<>RIGHT$(STR$(II),1
) THEN MB(II)=MB(II)-1:GOTO 60150 <217>
60135 : IF MB(II)=LEN(I$) THEN 60150 <074>
60140 : GOTO 60125 <248>
60150 NEXT II <179>
60170 PRINT "DIE SORTIERUNG ERFOLGT NACH FO
LGEM {3SPACE}STRINGBEREICH:" <088>
60180 PRINT "{RVSON}"MID$(I$,MA(1),MB(1)-MA
(1)+1)" {RVOFF}" <168>
60190 RETURN <047>
61000 REM FLOPPY-FEHLERKANAL LESEN -----
-- <254>
61005 PRINT "{CLR,RVSON}FLOPPY-FEHLERKANALA
BFRAGE {RVOFF}" <208>
61010 INPUT#15,A,B$,C,D <166>
61020 PRINT A,B$,C,D <126>
61040 GOTO 50 <031>

```

Listing 23. Der Relative-Dateien-Editor (Schluß)

Dokumentation zum Listing 23

Wichtigste Variablen:

M%	Größe des Stringpuffers
M\$(1)-M\$(M%), M(1)-M(M\$)	Stringpuffer
LG, RG	werden für Quicksort-Unterprogramme benötigt
A\$	von relativer Datei eingelesener String beziehungsweise String, der auf relative Datei geschrieben wird.
SA	Satz, auf den der Zeiger der relativen Datei positioniert ist beziehungsweise werden soll.
D\$	Filename der Datei
LB, HB	Lowbyte, Highbyte
AN, EN	Anfang beziehungsweise Ende eines Bereiches der relativen Datei, der sortiert beziehungsweise ausgedruckt werden soll.

Variablen beziehungsweise Stringnamen, die mit I beginnen, sind Hilfsvariablen beziehungsweise Strings, die sich während dem Programmablauf ständig ändern – zum Beispiel Schleifen, Prüfvariablen etc.

0 - 15	Vorbereitungen, Felder dimensionieren etc.
19 - 30	Anfangsmenü
31 - 41	Anlegen einer neuen Datei.
50 - 70	Zwischenmenü beziehungsweise Menüzeile. Dies ist der Hauptteil des Programmes. Von hier aus werden alle möglichen Routinen angewählt.
2000 - 2050	Routine zum Lesen von Sätzen einer relativen Datei.
3000 - 3070	Routine zum Schreiben von Sätzen in eine relative Datei.
6000 - 6270	Routine zum Versetzen bestimmter Bereiche einer relativen Datei innerhalb derselben oder auf andere relative Dateien. Die

7000 - 7510

7500 - 7999

8000 - 8690

9000 - 9140

10000 - 10500

20000 - 20060

30000 - 30330

50000 - 50140

60000 - 60190

61000 - 61040

Daten werden jeweils zuerst von der Quelldatei in den Puffer M% gelesen. Dann wird die Quelldatei geschlossen und die Zieldatei geöffnet. Sodann wird der Pufferinhalt in die Zieldatei übertragen etc.

Printer-Routine zum Ausdrucken von relativen Dateien.

In diesem Bereich können Sie eine eigene Printer-Routine unterbringen. So kann zum Beispiel der deutsche Zeichensatz Ihrem Drucker angepaßt werden.

Verschiedene Routinen zur formatierten Textausgabe einer relativen Datei.

Die bereits vorgestellten Unterprogramme zum Lesen, Schreiben, Positionieren und Schließen einer Datei.

Routine zum Sortieren eines bestimmten Bereiches innerhalb der Datei. Der gewünschte Bereich, der aus maximal M% Sätzen bestehen darf, wird in den Stringpuffer eingelesen. Nun wird der Stringpuffer mittels einem Quicksort-Algorithmus (10200 - 10350) sortiert und anschließend wieder in die Datei auf Disk zurückgeschrieben.

Korrekturroutine – »Blättern« in einer relativen Datei mittels den Cursor-Tasten. Diese Routine wird von der Texteingaberoutine ab 50000 aufgerufen.

Unterprogramm zum Einlesen des Disk-Directories ohne Programmverlust.

Modifizierte INPUT-Routine.

Unterprogramm zum Markieren des Stringbereiches, der beim Sortieren als Kriterium gelten soll.

Unterprogramm zur Abfrage des Fehlerkanals des Floppy-Laufwerks.

Die Dokumentation ist sehr oberflächlich; ich glaube nicht, daß am Editor noch viel verbessert werden kann. Mit dem Edi-

tor bekommen Sie relative Dateien in den Griff. Mit Hilfe der drei Unterprogramme zum Lesen, Schreiben und Positionieren können Sie sich leicht die unterschiedlichsten Dateiverwaltungen mittels kurzer Basic-Programme zusammenbasteln. Dazu müssen Sie jedoch wissen, wie mit dem Editor umzugehen ist:

Bedienungsanleitung zum Texteditor

Nach dem Programmstart mittels RUN erscheint das Grundmenü. Zum einen kann von hier aus ein alternativer Zeichensatz geladen werden, der mit dem Zeichensatz-Editor erstellt werden beziehungsweise ab hexadezimal E000 liegen muß. Der Zeichensatz muß den Namen »D.Z.SATZ« haben (beziehungsweise Name im Listing Zeile 27 umändern). Es bietet sich also an, mit dem Zeichensatz-Editor einen deutschen Zeichensatz automatisch generieren zu lassen und ihn dann als D.Z.SATZ auf die Disk, auf der sich auch der Editor befindet, zu speichern. Zum Laden des Alternativ-Zeichensatzes muß einfach die Taste »D« gedrückt werden. Wenn das Disketten-Laufwerk wieder zum Stillstand kommt, befindet sich der neue Zeichensatz im Computer. Natürlich hätte ich nach dem Filenamen des Zeichensatzes fragen können, nachdem »D« gedrückt wurde.

Ich habe mich jedoch für diese Lösung entschieden, da sich dadurch viel schneller mit dem Texteditor arbeiten läßt: Man muß so nicht erst mehrere Parameter eingeben, bevor man mit dem Arbeiten beginnen kann.

Mittels »N« kann eine neue Datei angelegt werden. Sie müssen lediglich den Dateinamen (Filenamen), maximal erlaubte Satzlänge und die Satzanzahl der Datei angeben. Dann eine Disk mit genügend freiem Platz einlegen und RETURN drücken. Je nach Größe der Datei kann es einige Zeit in Anspruch nehmen, bis der Computer hiermit fertig ist. Das Anlegen einer relativen Datei wird so zum Kinderspiel. Mittels »A« wird eine alte Datei zum weiteren Bearbeiten geöffnet. Es ist hierzu erforderlich, daß sich die Disk, auf der sich die Datei befindet, im Laufwerk befindet (logisch). Ganz Eilige können auch einfach mehrmals die RETURN-Taste drücken. Dann wird einfach das erste Programm der Disk, die sich im Laufwerk befindet, geladen und die maximale Zeichenzahl auf 40 festgelegt.

Achtung: Beim Anlegen einer relativen Datei ist immer folgendes zu beachten: Soll die Satzlänge einer Datei 40 Zeichen betragen, so muß man beim Anlegen der Datei als maximale Satzlänge 41 angeben – also immer eins mehr als die Satzlänge später tatsächlich sein soll. Dieser eine Platz beziehungsweise Zeichen wird nämlich für das CHR\$(13) (Carriage Return) benötigt, das am Satzende jeweils angehängt wird! Es ist in bedingten Fällen auch möglich, auf das CHR\$(13) zu verzichten. Man muß dann jedoch »ganz cool« über die zahlreich auftretenden FILE DATA beziehungsweise OVERFLOW IN RECORD ERRORS hinwegsehen!

Ich rate Ihnen jedenfalls, nicht an diesem einen Zeichen zu sparen!

Nach Anwahl des ersten oder zweiten Grundmenü-Punktes (alte Datei eröffnen beziehungsweise neue anlegen) gelangt man ins Hauptmenü, das aus einer einzigen Zeile besteht:

s/l/v/o/d/i/z/f/ende

Die einzelnen Buchstaben beziehungsweise Tasten stehen für folgende Funktionen:

s	- schreiben
l	- lesen
v	- versetzen
o	- ordnen
d	- drucken
i	- Inhaltsverzeichnis
z	- zentrieren
f	- Fehlerkanal abfragen
ende	- na, was wohl?

Die Menüpunkte im einzelnen

Schreiben

Zuerst muß die Satznummer eingegeben werden, ab der man schreiben will. Danach erscheint das Satzeingabe-Bild. In der obersten Bildschirmzeile sind Satznummer sowie momentaner Inhalt des Satzes zu sehen. Der Cursor steht in der untersten Zeile des Bildschirms. Dort kann ein neuer Satzinhalt eingegeben werden. Die Cursor-Tasten sind hierbei in ihrer Funktion zum Bewegen des eigentlichen Cursors außer Gefecht gesetzt. Auch alle anderen Steuertasten (zum Beispiel Farbwahl und CLR), außer der DEL-Funktion, funktionieren nicht mehr. Wie ich anfangs bereits sagte, kann bei einem Texteditor für relative Dateien keineswegs an die Vorzüge eines Textverarbeitungssystems gedacht werden. Sie können lediglich gesamte Sätze schreiben und diese mit RETURN absenden, wobei Sie automatisch in die nächste Zeile gelangen. Mit den Tasten CURSOR UP und CURSOR DOWN kann der Text nach oben beziehungsweise unten gescrollt werden.

Durch Eingabe von // wird die Texteingabe beendet – das Hauptmenü erscheint wieder.

Einen gewaltigen Vorteil hat ein Relative-Dateien-Editor allerdings gegenüber einem Textverarbeitungssystem: Bei einem Stromausfall gehen nur die Daten verloren, die sich gerade im Puffer des Floppy-Laufwerks befinden (meist nur 2 – 3 Sätze bei 40 Zeichen Satzlänge)!

Lesen

Auch hier muß nur die Satznummer des Satzes eingegeben werden, ab der gelesen werden soll. Auf dem Bildschirm erscheinen sodann die 20 Sätze ab dem eingegebenen Wert. Wenn Sie nach der Satznummer gefragt werden und einfach die RETURN-Taste drücken, ohne eine Zahl einzugeben, so wird ab dort weitergelesen beziehungsweise geschrieben, wo zuletzt aufgehört worden ist – also, wo der Zeiger gerade steht.

Versetzen

Diese Funktion erlaubt es, Teilbereiche (auch ganze relative Dateien) innerhalb einer Datei zu versetzen beziehungsweise auf eine andere Datei zu übertragen. Zunächst müssen Anfang und Ende des Bereiches angegeben werden, der versetzt werden soll. Danach muß die Zieldatei angegeben werden. Soll innerhalb der relativen Datei versetzt werden, so muß als Zieldatei einfach der Name der Quelldatei (= Datei, mit der gerade gearbeitet wird) angegeben werden.

Letztendlich muß nur noch der Anfang des neuen Bereiches angegeben werden. Beachten Sie bitte, daß die Satzlänge der Zieldatei entweder größer oder mindestens genauso groß wie die der Quelldatei sein muß!

Ordnen

Heißt nichts anderes als Sortieren einer Datei. Zunächst wird der Bereich angegeben, der sortiert werden soll (darf nicht größer als 300 Sätze sein!). Dann muß markiert werden, welcher Teilbereich beziehungsweise welcher Schlüssel der Datei als Sortierkriterium gelten soll. Mittels den Cursortasten LEFT–RIGHT wird der Bereich markiert. Als Kriterium gilt der Bereich, unter dem Einser stehen. Mit den Tasten 1 und 0 können solche Einser gesetzt beziehungsweise gelöscht werden. Beispiel:

```
001BETRACHTEN 56
0011111111111100
```

Nun gilt zum Beispiel bei einer Verben-Tabelle, wie wir sie schon bald kennenlernen werden, das Verb als Kriterium.

Zur Sortieroutine, die im Editor integriert ist, ist zu bemerken, daß sie nur für kleine Bereiche geeignet ist, deren Satzlänge kleiner als 40 Zeichen ist. In einem anderen Kapitel dieses Kurses finden Sie noch ein Sortierprogramm für riesige Dateien mit Satzlängen bis zu 255 Zeichen!

Drucken

Durch Aufruf dieser Routine können relative Dateien beziehungsweise Teilbereiche einer Datei mit einem Drucker ausgegeben werden. Das Druckermenü bietet dann folgende Auswahlmöglichkeiten:

Normale Ausgabe: Der Text wird so ausgegeben, wie er mit dem normalen Zeichensatz des C64 aussieht.

D.z.s.-Ausgabe: Ein Text, der mit Hilfe des deutschen Zeichensatzes erstellt wurde, wird umgerechnet ausgegeben. Das heißt, anstelle von ß wird ss, anstelle von ä ein ae ausgedruckt etc. Dies ist nötig, da die Druckerroutine für einen MPS-801 »Billigdrucker« geschrieben wurde.

Eigene Routine: Wenn Sie einen besseren Drucker haben (Neid!!!), können Sie sich eine eigene Druckroutine schreiben (siehe Listingsdokumentation). Wer will, kann mittels Bit-Gefummel den deutschen Zeichensatz auch auf einem MPS-801 darstellen.

Inhaltsverzeichnis

Hier wird das Directory ohne Programmverlust ausgegeben.

Zentrieren

Durch diese Routine können Dateibereiche zentriert werden. Es ist so möglich, den Text formatiert auszugeben. Es gibt zwei Formatierungsmöglichkeiten: Links-rechtsbündig zentriert und in die Mitte zentrieren. Bei ersterem werden einfach so viele SPACES eingefügt, bis der Text am linken und rechten Rand anstößt. Beim in die Mitte zentrieren wird jeder Satz so dargestellt, daß links und rechts gleich viele SPACES sind.

Fehlerkanalabfrage:

Hiermit kann der Fehlerkanal des Floppy-Laufwerks abgefragt werden.

Ende

Muß unbedingt betätigt werden, wenn man die Arbeit an einem Text beenden will. Nur dann wird gewährleistet, daß der Puffer des Diskettenlaufwerks auf Disk gespeichert wird und so keine Daten verloren gehen können.

Sie können den Editor auch jederzeit mittels RUN/STOP-Taste verlassen. Mittels GOTO 50 im Direktmodus kommen Sie dann wieder ins Hauptmenü.

Dies ist auch schon alles, was Sie zum Umgang mit dem Editor wissen müssen. Wir werden im Rahmen des Kurses den Editor noch oft benötigen. Ich rate Ihnen deshalb, sich schon jetzt mit seinen einzelnen Funktionen gründlich vertraut zu machen. Im nächsten Kapitel finden Sie ein weiteres Utility, das ebenfalls (wie der Zeichensatz-Editor) direkt auf den Text-Editor für relative Dateien anwendbar ist: Ein Sortierprogramm für relative Dateien.

Das Sortieren von riesigen Datenfeldern (relativen Dateien)

Bisher haben wir immer nur kleine Tabellen sortiert. Also Dateien, die immer komplett im Computer standen. Unser Texteditor für relative Dateien ist zwar in der Lage, riesige Dateien in den Griff zu bekommen, was jedoch das Sortieren von Datenfeldern anbelangt, so bedient er sich eines kleinen Kunstgriffes. Der Dateiabschnitt, der sortiert werden soll, wird komplett in ein Puffer-Stringfeld innerhalb des Basic-RAMs geladen. Dieser Puffer wird sodann mit Hilfe eines Quicksort-Unterprogramms sortiert. Anschließend wird der Pufferinhalt wieder auf die relative Datei auf Disk übertragen. Die »Sortierkapazität« unseres Texteditors ist also auf die Größe des Puffers und somit auf den Speicher des C64 begrenzt. Dieser Puffer kann nicht als besonders groß bezeichnet werden. Sie werden mit Sicherheit sehr bald auf seine Grenzen stoßen. Spätestens dann wird es kritisch, wenn man den Wortschatz zu einem Adventure sortieren will.

Wir haben schließlich vor, Wortschätze mit über 1000 Worten zu verwalten – und die wollen erst einmal sortiert sein.

Wir brauchen also ein Sortierprogramm, das in der Lage ist, Dateien von Disk zu sortieren, die nicht auf einmal in den Speicher unseres C64 geladen werden können.

Das Programmieren von solchen Datei-Sortier-Programmen ist eine äußerst haarige Sache. Auch die besten Programmierer müssen sich vor dem Programmieren lange und genaue Überlegungen und Skizzen auf Papier zeichnen, um genauen Durchblick zu behalten. Die Schwierigkeit liegt weniger bei der komplizierten Programmieretechnik, die erforderlich ist, sondern vielmehr bei den Grundüberlegungen, die gemacht werden müssen. Ähnlich wie bei den einzelnen Sortieralgorithmen (Vgl. Kurs über effektives Programmieren in der Zeitschrift 64'er) gibt es auch hier zahlreiche Möglichkeiten zum Lösen des Problems. Die Grundroutinen sind zwar immer die gleichen, es gibt jedoch unendlich viele Möglichkeiten, sie zu kombinieren. Auch die besten Sortiermethoden für Dateien, die komplett auf einmal im Speicher sind, setzen sich aus mehreren Sortieralgorithmen zusammen (zum Beispiel Quicksort und Bubblesort kombiniert), um eine möglichst schnelle Ausführungszeit zu erreichen. Bei diskorientierten Sortierprogrammen spielt neben dem Zeitaufwand, der zum Sortieren gebraucht wird, auch noch ein anderer Faktor eine dominierende Rolle: Die Zahl der Zugriffe auf Disk. Selbstverständlich ist es möglich, unser Quicksort-Unterprogramm (Listing 8) mit wenig Aufwand so zu modifizieren, daß es in der Lage ist, eine riesige Datei auf Disk zu sortieren. Dabei wären bei großen Dateien zehntausende, ja bis zu über 100 000 Schreib-Lese-Operationen mit dem Diskettenlaufwerk notwendig. Sie werden mir sicherlich zustimmen, wenn ich nun behaupte, daß eine solche Sortiermethode eine unglaublich große Strapaze für unser Floppy-Laufwerk (ganz zu schweigen von der Diskette, die dieser Tortur ausgesetzt wird) wäre.

Eine andere Methode wäre es zum Beispiel, mehrere Disketten zu verwenden. Zum Beispiel die Quelldisk, auf der sich die Tabelle befindet und eine weitere Disk, auf der das Sortierprogramm zwischenspeichern kann. Dies wäre zumindest schonend für die Diskette.

Beide Methoden sind jedoch äußerst zeitaufwendig. Eine Methode, möglichst schnell zu sortieren, ist es, im Speicher des C64 nur die Satznummern zu puffern und diese zu sortieren, um dann gemäß dieser sortierten Satznummern die ganze Datei umzuordnen. Während man mit der allereinfachsten Methode (gesamte Datei auf einmal im Speicher) etwa 1000 Sätze (mittlerer Länge) sortieren kann, erlaubt die zuletzt vorgestellte Methode schon ein Vielfaches dieser Kapazität.

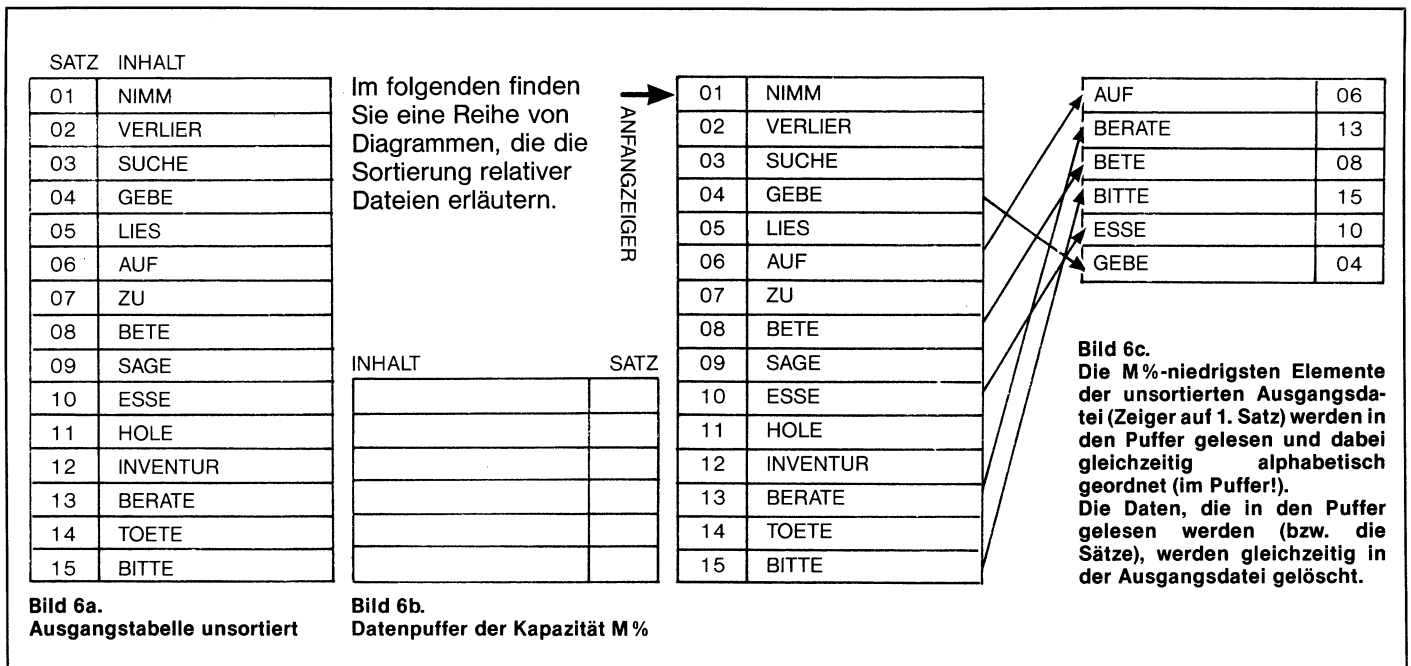
Die Methode hat jedoch wieder eine Einschränkung: Man kann eine bestimmte Satzanzahl (Größe des Satznummern-Puffers im Speicher) nicht überschreiten.

Wir wollen keine halben Sachen machen und ein absolut endgültiges Sortierprogramm schreiben, das folgende Eigenschaften haben soll:

- Es kann jede Datei, die sich auf einer Diskettenseite in Form einer relativen Datei unterbringen läßt, sortiert werden.
- Die Sortierung muß möglichst Disketten- und Floppy-Laufwerk-schonend ablaufen. Auf schnelle Ausführungszeit wird deshalb weniger Wert gelegt.

Auf den folgenden Seiten werde ich Ihnen anhand von Ablaufplänen und Schemen ausführlich erläutern, wie man eine solche Sortierung programmiert. Diese Grundlagen sind übrigens in keinsten Weise auf den C64 beschränkt – sie gelten auch für jedes andere Computersystem, mit dem relative Dateien verwaltet werden können.

Lehnen Sie sich also gemütlich in Ihren Sessel, besorgen Sie sich was zu knabbern und/oder trinken und studieren Sie ausführlich die folgenden Seiten!



Anhand der folgenden Skizzen wollen wir uns zunächst einmal grundlegende Gedanken über das Sortieren von riesigen Dateien machen. Das Verstehen der Programmieretechnik wird Ihnen dann viel leichter fallen. Zunächst benötigen wir wieder eine Ausgangstabelle. Um es möglichst einfach zu halten, beschränke ich mich hier auf eine Datei mit nur einem Schlüssel: Eine Verben-Datei, wie sie uns im Laufe des Kurses noch oft in ähnlicher Form begegnen wird. Diese Beispielsdatei sehen Sie in Bild 6a. Die »Ausgangstabelle« besteht aus 15 Sätzen – Verben in total ungeordneter Reihenfolge. Wir wollen einmal annehmen, daß der Speicher unseres C 64 viel zu klein ist, um diese Datei auf einmal fassen zu können – beziehungsweise die Datei zu groß ist, um komplett in den Computer geladen werden zu können. Wir benötigen also einen Puffer – ein Stringfeld, in das wir Teile der riesigen Datei laden können, um sie dort zu verändern. Das folgende Beispiel macht besonders leicht deutlich, wie so ein »Puffer« arbeitet.

Herr Helm hat zwei große Schwimmbecken – eines leer und eines voll mit Wasser. Er möchte das leere Schwimmbecken nun mit dem Wasser des vollen Schwimmbeckens füllen. Da er das volle Becken jedoch nicht einfach hochheben, zum anderen Becken hintragen und hineinleeren kann, braucht er ein Hilfsmittel. Zum Beispiel einen Eimer, der maximal so groß sein darf, daß Herr Helm ihn im gefüllten Zustand noch gut tragen kann. Nun kann er den Eimer im vollen Becken mit Wasser vollschöpfen, zum leeren Becken gehen und den Eimer dort hineinleeren. Herr Helm muß also mehrmals zwischen den Becken hin und her laufen, bis das volle Becken leer und das vorher leere Becken voll ist. Je kleiner der Eimer ist, umso öfter muß Herr Helm zwischen den Becken hin und her gehen...

Der Eimer ist als »Puffer« beziehungsweise »Datenpuffer« zu sehen. Den »Eimer«, den wir als Puffer zum Sortieren benutzen werden, sehen Sie in Bild 6b. Er faßt bis zu 6 Sätze. In Wirklichkeit ist der Speicher des C 64 natürlich groß genug, um mehrere hundert Sätze puffern zu können. Der Einfachheit halber wollen wir uns jedoch auf einen kleinen Puffer beschränken. Unser Puffer besteht aus zwei Teilen. Im einen (M\$(1)–M\$(%)) werden die Sätze gespeichert. M% ist die Größe des Puffers – also 6. In der anderen Hälfte des Puffers (M\$(1)–M\$(M%)) wird gespeichert, an welcher Stelle die Sätze in der relativen Datei stehen – also die Nummern der Sätze.

Als Ziel des gesamten Sortiervorgangs soll die Verben-Datei alphabetisch geordnet vorliegen. Wie beginnen wir nun den Sortiervorgang? Das einzige Feld, das wir sortieren können, ist der Puffer. Wir müssen also zuerst Sätze von der Verben-Datei in den Puffer lesen, um sie dort verarbeiten zu können. Unsinnig wäre es, einfach die ersten 6 Sätze der Datei in den Puffer zu lesen und zu sortieren – das bringt nichts.

Wir durchlaufen zunächst die ganze Verben-Datei einmal von Anfang bis Ende und suchen die niedrigsten 6 Sätze heraus. Diese Sätze schreiben wir in den Puffer (Schema Bild 6c).

Das Suchschema läuft so ab, daß die 6 Sätze am Ende bereits alphabetisch im Puffer abgelegt sind. Gleichzeitig werden alle Sätze, die in den Puffer geschrieben werden, auf der Ausgangstabelle gelöscht. Nachdem der Puffer voll ist, sieht die relative Datei auf Disk wie in Bild 6d aus:

Betrachten wir nun einmal die ersten M%-Sätze der jetzigen Datei. An genau dieser Stelle müssen am Ende des Sortiervorgangs die 6 niedrigsten Sätze stehen. Diese Sätze befinden sich in unserem Puffer. Allerdings können wir den Puffer jetzt nicht gleich dorthin übertragen, da sich in ihm noch volle Sätze befinden, die beim Zurückschreiben des Puffers überschrieben werden würden – also verloren gingen.

Zählen wir doch einmal, wie viele volle Sätze noch im oberen Bereich stehen: genau 4 (Satz 1, 2, 3 und 5). Wir müssen also 4 Sätze »in Sicherheit« bringen, bevor der Pufferinhalt in den oberen Bereich geschrieben werden darf.

Aber wohin denn nun mit den 4 Sätzen? Sicherlich werden Sie schon festgestellt haben, daß sich im unbetroffenen Teil der Datei, der beim Übertragen des Puffers nicht überschrieben wird, genau 4 leere Stellen befinden. 4 Stellen, an denen vorher Sätze standen, die sich jetzt gerade im Puffer befinden und die später in den oberen Bereich geschrieben werden. Es bietet sich also an, die 4 Sätze im oberen Bereich in die 4 Leersätze des unteren Bereiches zu schreiben (s. Bild 6e).

Wenn Sie nicht ganz mitgekommen sind, lesen Sie bitte den Abschnitt noch mal durch. Es gibt nun verschiedene Methoden, den oberen Bereich zu räumen. Die einfachste ist, die gesamte Datei mit Schleifen zu durchlaufen. Dies erfordert jedoch wieder zahlreiche Lesezugriffe per Disk und wir haben uns vorgenommen, ein möglichst Disketten- und

01	NIMM
02	VERLIER
03	SUCHE
04	
05	LIES
06	
07	ZU
08	
09	SAGE
10	
11	HOLE
12	INVENTUR
13	
14	TOETE
15	

Bild 6d. Ausgangsdatei nach dem 1. »Füllen« des Puffers.

01	
02	
03	
04	
05	
06	
07	NIMM
08	VERLIER
09	SUCHE
10	LIES
11	ZU
12	SAGE
13	HOLE
14	INVENTUR
15	TOETE

Bild 6e. Unser nächstes Ziel: Platz schaffen, um den Puffer wieder in die Ausgangsdatei zurückschreiben zu können.

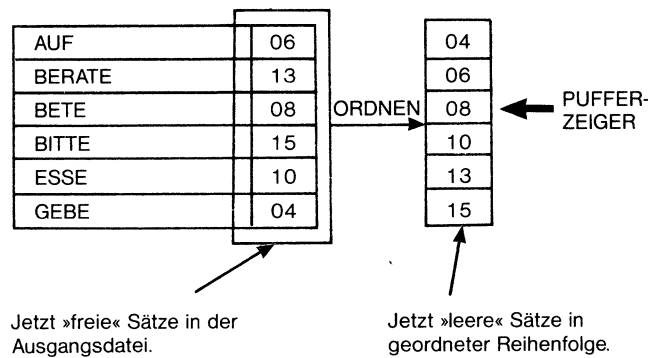


Bild 6f. Hier sehen Sie, wie nach dem Aussortieren der ersten Sätze Platz zum Zurückschreiben geschaffen wird.

Laufwerk-schonendes zu entwickeln. Wir entscheiden uns deshalb für einen komplizierteren Weg, der jedoch nur sehr wenige Zugriffe mit dem Floppy-Laufwerk erfordert.

Wie ich bereits sagte, ist der Puffer in zwei Hälften unterteilt – eine für die Sätze selbst und eine für die Satznummern. Werfen Sie bitte einmal einen Blick auf Bild 6f. Hier wird gezeigt, wie die Pufferhälfte, in der die Satznummern stehen, mit der kleinsten Satzzahl beginnend sortiert wird. Hierfür verwenden wir unser Quicksort-Unterprogramm, das aus Geschwindigkeitsgründen jedoch auf Zahlen-Sortieren modifiziert wird.

Nachdem die Nummern-Pufferhälfte sortiert ist, wird ein Pufferzeiger gesetzt – der genau auf den ersten leeren Satz zeigt, der sich nicht mehr innerhalb des oberen Bereiches befindet (in unserem Beispiel SATZ 8). Dies ist nötig, damit versehentlich nicht Daten, die aus dem oberen Bereich »gerettet« werden sollen, in einen leeren Satz innerhalb des oberen Bereiches geschrieben werden.

Bild 6g zeigt schließlich, wie die 4 Sätze aus dem oberen Bereich auf die 4 leeren Stellen des unteren Bereiches verteilt werden. Nun kann der Pufferinhalt wieder in die Verben-Datei zurückgeschrieben werden (Bild 6h). Die ersten M%-Sätze der Datei sind damit alphabetisch geordnet.

Nun wird der Zeiger der Verben-Datei, der bisher an erster Stelle stand (weil wir die ganze Datei und nicht nur einen bestimmten Bereich sortieren wollen) um den Wert M% erhöht. Er zeigt sodann auf die Stelle, an der der noch unsortierte Bereich der Datei beginnt (Bild 6i). Der nächste Sortierabschnitt kann nun beginnen.

Sind alle Sortierabschnitte beendet – also jeweils Puffer mit den M% niedrigsten Elementen vom unsortierten Bereich füllen, Sätze aus dem Pufferbereich in Sicherheit bringen, Puffer zurückschreiben etc. – liegt die komplette Verben-Datei sortiert vor (Bild 6j).

Eine Frage ist noch offen ...

Woher das BIER in SATZ 16 kommt? Nun, das haben Sie sich nach diesem Brocken Theorie mit Sicherheit verdient (und ich auch!)...

Anschließend finden Sie das Listing 24 zum Sortierprogramm. Bitte studieren Sie auch die Dokumentation ausführlich, damit Sie später in der Lage sind, das Programm eigenen Vorstellungen entsprechend zu ergänzen beziehungsweise abzuändern.

Dokumentation zum Super-Sort-Listing

100 – 130

Hier werden die Felder für die beiden Puffer-teile und den Quicksort definiert. Durch Ändern der Variable M% kann die Größe des Puffers geändert werden (momentaner Wert 300).

200 – 295

Anfangsmenü und Eingabe aller Parameter. AN – EN, Bereich innerhalb der Datei NS, der sortiert werden soll. ZL, ZR begrenzen den Bereich innerhalb der Sätze, nach dem sortiert werden soll. Beispiel: MID\$(A\$,ZL,ZR) Vorbereitung für den ersten Sortiervorgang (Sortieren der M% niedrigsten Sätze, beginnend ab dem Zeiger auf die Ausgangsdatei). ZD, Zeiger in Datei. Um die M% niedrigsten Sätze zu finden, wird die gesamte Datei von Satz ZD bis zum letzten Satz durchlaufen. VG\$ wird als Vergleichsargument bestimmt. Hier wird mittels einer Schleife die Datei ab ZD bis Ende nach den M% niedrigsten Sätzen durchsucht. Dabei wird jeweils der Vergleichsstring VG\$ mit einem Satz des Suchbereiches verglichen. Wird ein Satz gefunden, der kleiner als VG\$ ist, so nimmt VG\$ dessen »Wert« an. In der Variablen MM wird gespeichert, an welcher Stelle der Datei der momentane »Wert2«, der in VG\$ gespeichert ist, steht. Ist die Datei einmal durchlaufen, so steht in VG\$ der kleinste Satz des durchsuchten Bereiches.

400 – 470

Der Satz MM, also die zugehörige Satznummer zum im Durchlauf gefundenen, niedrigsten Wert VG\$, wird in der Datei gelöscht. Löschen ist eigentlich eine falsche Bezeichnung. In den Satz MM wird vielmehr ein »%« geschrieben. Dies ist notwendig, da ein leerer Satz, also A\$ = "" schließlich auch ein Satz ist. Leere Sätze sind kleiner als alles andere. Hat eine Datei, die sortiert wird, leere Sätze, so stehen diese nach dem Sortiervorgang immer an den ersten Stellen in der geordneten Datei. In der Datei, die sortiert werden soll, darf somit nie in einem Satz ein einzelnes »%« stehen, da es dann zu

500 – 570

01	NIMM
02	VERLIER
03	SUCHE
04	
05	LIES
06	
07	ZU
08	
09	SAGE
10	
11	HOLE
12	INVENTUR
13	
14	TOETE
15	

Bild 6g. Die Sätze, die noch in dem Bereich stehen, in den der Puffer übertragen werden soll, werden in »Sicherheit« gebracht (in den unteren Teil der Ausgangsdatei).

01	AUF
02	BERATE
03	BETE
04	BITTE
05	ESSE
06	GEBE
07	ZU
08	NIMM
09	SAGE
10	VERLIER
11	HOLE
12	INVENTUR
13	SUCHE
14	TOETE
15	LIES

Bild 6h. Der Puffer ist in die Ausgangsdatei zurückgeschrieben worden. Die ersten M%-Elemente (Sätze) sind jetzt alphabetisch geordnet. Die restliche Datei ist noch unsortiert.

Zeiger →

01	AUF
02	BERATE
03	BETE
04	BITTE
05	ESSE
06	GEBE
07	ZU
08	NIMM
09	SAGE
10	VERLIER
11	HOLE
12	INVENTUR
13	SUCHE
14	TOETE
15	LIES
16	BIER

Bild 6i. Natürlich müssen jetzt auch noch die restlichen Elemente sortiert werden. Dazu wird einfach der Zeiger um den Wert M% erhöht. Die nächste Sortierrunde kann losgehen.

Schwierigkeiten beim Lückenfüllen kommen kann. Wer auf »%« in seiner Datei nicht verzichten kann, kann im Listing das »%« auch durch ein anderes Zeichen ersetzen, das dann nicht verwendet werden kann (in der Datei). Nun wird geprüft, ob der Puffer schon voll ist (Zeile 540). ZP ist der Pufferzeiger. Sein Wert wird immer dann um 1 erhöht, wenn ein Satz in den Puffer geschrieben worden ist. Ist noch Platz im Puffer, geht es so weiter: Ein neues Vergleichselement VG\$ muß bestimmt werden. Dabei muß aufgepaßt werden, daß das neue VG\$ niemals " " sein darf. Hierfür sorgt Zeile 560. Nun folgt ein Rücksprung zu 400 – es wird wieder das kleinste Element gesucht. Der Puffer ist jetzt voll. Die Pufferhälfte MS(1)–MS(M%), in der die Satznummern gespeichert sind, wird nun mittels dem Quicksort-Unterprogramm sortiert (vergleiche Bild 6). Anschließend wird der Pufferzeiger ZP auf den ersten Satz gesetzt (Leersatz beziehungsweise Satz mit »%«), der im unteren Bereich liegt (Zeile 650 – 670).

600 – 670

700 – 760

Hier werden die vollen Sätze, die noch immer im oberen Bereich stehen, evakuiert – also in die leeren Sätze des unteren Bereiches geschrieben. Hierzu wird der obere Bereich mit einer Schleife durchlaufen. Wird ein voller Satz gefunden, so wird er in den Leersatz geschrieben, auf den der Pufferzeiger gerade deutet. Der Pufferzeiger wird sodann um den Wert 1 erhöht.

800 – 860

Der Puffer wird mittels Schleife in den oberen Bereich ab ZD (Zeiger auf Datei) geschrieben. Nun wird der nächste Sortierabschnitt vorbereitet. Der Dateizeiger ZD wird um den Wert M% erhöht – also auf den Satz gestellt, bei dem der noch unsortierte Bereich der Datei beginnt. Ein neues Vergleichselement wird bestimmt. Der Pufferzeiger wird wieder auf den Wert 1 zurückgesetzt.

900 – 950

```

0 REM ***** <050>
1 REM *   SORTIERPROGRAMM FUER   * <250>
2 REM *   RELATIVE DATEIEN      * <250>
3 REM *   <C> 1985 BEI          * <122>
4 REM *   MICHAEL NICKLES       * <121>
5 REM ***** <055>
100 M%=300 :REM GROESSE DES PUFFERS <173>
105 N%=M% :REM WERT VON M% SPEICHERN <235>
110 DIM M$(M%) :REM PUFFER FUER SAETZE <198>
120 DIM MS(M%) :REM PUFFER FUER ZAHLEN <024>
130 DIM LG(100),RG(100):Z=0:LG(1)=1:RG(1)=M% :REM FUER QUICKSORT <088>
200 REM ANFANGSMENUE ----- <075>
201 : <177>
205 (POKE 53280,0:POKE 53281,0:PRINT" (CLR,W <249>
HITE)");
210 PRINT" (RVSON,6SPACE)REL.DATEI SORTIERP <216>
ROGRAMM(9SPACE,RVOFF)";
215 PRINT" (RVSON,9SPACE)VON MICHAEL NICKLE <040>
S(12SPACE,RVOFF)";
220 INPUT" (2DOWN)DATEINAME ";N$ <226>
230 OPEN 15,8,15:REM FLOPPYKANAL AUF <169>
235 PRINT#15,"I0":REM INITIALISIEREN <138>
240 OPEN 1,8,3,N$ :REM REL.DATEI OEFFNEN <174>
250 INPUT#15,A,B$:REM FEHLERABFRAGE <235>
254 IF A<>0 THEN PRINT" (RVSON)ERROR (RVOFF, <189>
2SPACE)";B$:END
270 PRINT" (2DOWN,8SPACE,RVSON)G (RVOFF)ANZE <172>
DATEI SORTIEREN
275 PRINT" (DOWN,8SPACE,RVSON)T (RVOFF)EILBE <060>
REICH SORTIEREN
277 PRINT" (DOWN,8SPACE,RVSON)L (RVOFF)ESEN <175>
280 GET I$:IF I$<>"G"AND I$<>"T"AND I$<>"L <032>
THEN 280
281 IF I$="L"THEN 10000 <164>
285 IF I$="G"THEN AN=1:GOTO 290 <073>
286 PRINT" (DOWN)BITTE GEBEN SIE ANFANG UND <006>
ENDE DES(5SPACE)BEREICHES AN, DER SOR
TIERT";
287 PRINT" WERDEN SOLL!" <185>
288 INPUT"ANFANG ";AN <038>
289 INPUT"ENDE(3SPACE)";EN:IF AN>EN THEN P <126>
RINT"UNSINN!";GOTO 295
290 PRINT" (DOWN)BITTE GEBEN SIE AN, NACH W <180>
ELCHEM STRING-BEREICH DIE SORTIERUNG
291 PRINT"ERFOLGEM SOLL!" <135>
292 INPUT" (DOWN)VON ZEICHEN ";ZL <053>
293 INPUT" BIS ZEICHEN ";ZR <002>
295 ZR=ZR-ZL+1 <040>
300 REM SORTIERVORGANG ANFANG ----- <202>
301 : <023>
305 TI$="000000" :REM ZEIT AUF 0 <133>
310 ZD=AN :REM ZEIGER SETZEN <037>

```

Listing 24. Das Sortierprogramm für relative Dateien ►

01	AUF
02	BERATE
03	BETE
04	BITTE
05	ESSE
06	GEBE
07	HOLE
08	INVENTUR
09	LIES
10	NIMM
11	SAGE
12	SUCHE
13	TOETE
14	VERLIER
15	ZU
16	BIER

Bild 6j.
Nach einer langen Tour-de-Force ist unsere Datei sortiert. Bitte beachten Sie Datensatz 16. Die Erklärung finden Sie auf Seite 34.

Die komplett sortierte Datei.

```

320 ZP=1 :REM PUFFERZEIGER <108>
325 SA=ZD:GOSUB 9000:GOSUB 9100
      :REM 1.SATZ LESEN <041>
330 VG$=A$ :REM VERGL.SATZ <228>
400 REM DIE M% KLEINESTEN ELEMENTE VON SAT
    Z ZD BIS LETZTEN SATZ SUCHEN----- <065>
401 : <123>
402 PRINT" (DOWN,RVSON)";ZP;" (LEFT). DURCHL
    AUF (4SPACE,RVOFF,DOWN)" <027>
410 SA=ZD+1:REM ----- BEGIN LOOP ----- <203>
411 PRINT SA,A$ :REM SUPERMIKE <195>
430 :GOSUB 9000 :REM POSITIONIEREN <134>
431 IF A=50 THEN M%=SA-ZD:EN=SA:GOTO 500
      :REM END
      LOOP <058>
432 IF SA=EN+1 THEN 500
      :REM END LOOP <082>
435 :GOSUB 9100 :REM A$ LESEN <149>
440 : IF MID$(A$,ZL,ZR)>MID$(VG$,ZL,ZR) OR A
    $="@" THEN 460 <071>
445 : VG$=A$ :REM NEUER VG$ <209>
450 : MM=SA :REM SATZNR. MERKEN <014>
460 SA=SA+1 :REM NEXT SATZ <035>
465 GOTO 411:REM ----- CONT LOOP ----- <181>
470 : <192>
500 REM KLEINSTES ELEMENT IM BEREICH ZD BI
    S ZD+M% = VG$ WURDE GEFUNDEN ----- <173>
501 : <223>
510 SA=MM:GOSUB 9000:A$="@" :GOSUB 9200
      :REM SATZ MM LOESCH <002>
      EN
520 M$(ZP)=VG$:MS(ZP)=MM
      :REM PUFFER FUELLEN <230>
530 ZP=ZP+1 :REM PUFFER-ZEIGER+1 <087>
540 IF ZP>M% THEN 600 :REM PUFFER VOLL <230>
550 REM NEUEN VG$ SUCHEN <029>
555 SA=ZD <013>
560 GOSUB 9000:GOSUB 9100:IF A$="@" THEN SA
    =SA+1:GOTO 560 <176>
565 VG$=A$:MM=SA <193>
570 GOTO 400 <014>
600 REM PUFFER IST JETZT VOLL ----- <008>
      ----- <069>
601 :
610 PRINT" (DOWN,RVSON)PUFFERINHALT (RVOFF,D
    OWN)";FOR I=1 TO M%:PRINT M$(I),MS(I):
    NEXT I:REM SUPERMIKE <158>
620 REM MS(1)-MS(M%) WIRD JETZT MIT QUICKS
    ORT SORTIERT ----- <203>
621 : <089>
625 GOSUB 30000:REM AUFRUF QUICKS.UP <124>
630 PRINT" (DOWN,RVSON)SORTIERTER NUMMERPUF
    FER (RVOFF,DOWN)";FOR I=1 TO M%:PRINT M
    S(I):NEXT:REM SUPERMIKE <192>
650 REM NEUEN PUFFERZEIGER SETZEN -----
      ----- <174>
651 : <119>
655 ZP=1 <207>
660 IF MS(ZP)<ZD+M% THEN ZP=ZP+1:GOTO 660 <089>
670 PRINT"ZP=";ZP :REM SUPERMIKE <204>

```

```

700 REM PLATZ FUER PUFFER SCHAFFEN ----- <004>
      ----- <169>
701 : <252>
710 FOR I=ZD TO ZD+M%-1
720 SA=I:GOSUB 9000:GOSUB 9100
      :REM A$ LESEN <155>
729 PRINT A$ :REM SUPERMIKE <111>
730 : IF A$="@" THEN 760 <187>
740 : SA=MS(ZP):GOSUB 9000:GOSUB 9200
      :REM LUECKE FUELLEN <155>
745 : ZP=ZP+1 :REM P.ZEIGER +1 <136>
760 NEXT I <080>
800 REM PUFFER ZURUECKSCHREIBEN -----
      ----- <099>
801 : <015>
810 SA=ZD <014>
820 : FOR I=1 TO M% <077>
830 : A$=M$(I) <145>
831 : PRINT A$ :REM SUPERMIKE <088>
840 : GOSUB 9000:GOSUB 9200 <050>
850 : SA=SA+1 <196>
860 : NEXT I <242>
900 REM NAECHSTEN SORTIERVORGANG VORBEREIT
    EN----- <052>
901 : <115>
910 IF M%<>N%OR O% THEN PRINT"SORTIERZEIT:
    ";TI$:PRINT" (2DOWN)*** FERTIG ***":CLO
    SE 15:END <106>
920 ZD=ZD+M%:REM DATEIZEIGER ERHOEHEN <217>
924 ZP=1 :REM PUFFERZEIGER AUF 1 <179>
925 SA=ZD <129>
926 GOSUB 9000:GOSUB 9100:IF A$="@" THEN SA
    =SA+1:GOTO 560 <032>
927 VG$=A$:MM=SA <045>
930 IF M%>EN-ZD THEN M%=EN-ZD <227>
935 IF M%=EN-ZD THEN O%=1 <238>
950 GOTO 400 :REM NAECHSTE SORTIERRUNDE <134>
9000 REM POSITIONIEREN ***** <109>
9001 : <087>
9010 : HB%=SA/256:LB%=SA-HB%*256 <005>
9020 : PRINT#15,"P";CHR$(3);CHR$(LB%);CHR$
    (HB%);CHR$(1) <125>
9030 : INPUT#15,A:REM FEHLERABFRAGE <220>
9040 RETURN <208>
9100 REM SATZ LESEN ***** <031>
9101 : <187>
9110 : IF A=50 THEN RETURN:REM FEHLER <068>
9120 : INPUT#1,A$ <236>
9130 RETURN <042>
9200 REM SATZ SCHREIBEN ***** <224>
9201 : <031>
9210 : PRINT#1,A$ <230>
9230 RETURN <144>
10000 REM DATEI LESEN -----
      ----- <086>
      ----- <071>
10001 : <114>
10010 PRINT" (RVSON)LESEN / ENDE (RVOFF)" <189>
10015 GET I$:IF I$<>"L"AND I$<>"E" THEN 100
    15 <195>
10020 IF I$="E" THEN PRINT" (CLR)":GOTO 270 <103>
10030 INPUT"SATZNUMMER ";IA <047>
10034 PRINT" (RVSON)SATZ";IA;" (LEFT)-";IA+2
    0;" (RVOFF)" <054>
10035 FOR SA=IA TO IA+20 <098>
10040 : GOSUB 9000:GOSUB 9100 <103>
10050 : IF LEN(A$)<40 THEN PRINT A$ <015>
10060 : IF LEN(A$)>40 THEN PRINT A$; <092>
10070 NEXT SA <202>
10080 GOTO 10010
30000 REM QUICKSORT -----
      ----- <087>
      ----- <128>
30001 RG(1)=M% <166>
30010 Z=Z+1:IF LG(Z)>=RG(Z) THEN 30120 <011>
30020 X=LG(Z):Y=RG(Z) <011>
30030 VG=MS(INT((X+Y)/2)) <205>
30040 IF X>Y THEN 30100 <153>
30050 IF MS(X)<VG THEN X=X+1:GOTO 30050 <241>
30060 IF MS(Y)>VG THEN Y=Y-1:GOTO 30060 <235>
30070 IF X>Y THEN 30100 <214>
30080 S=MS(X):MS(X)=MS(Y):MS(Y)=S <099>
30090 X=X+1:Y=Y-1:GOTO 30040 <184>
30100 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 30010 <104>
30110 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 30010 <161>
30120 Z=Z-1:RETURN

```

Listing 24. Das Sortierprogramm für relative Dateien

Sodann wird überprüft, ob der Puffer größer ist als der Bereich, der noch sortiert werden muß. Ist dies der Fall, so wird der Puffer so verkleinert (M% in Zeile 930), daß der Bereich, der noch sortiert werden muß, genau in ihn hineinpaßt. In Zeile 910 wird solch eine Puffergrößenveränderung festgestellt – die Sortierung ist beendet. Die Variable erfüllt den gleichen Zweck dann, wenn alles glatt aufgeht, und M% nicht erniedrigt wird.

9000 – 9230 Unterprogramme zum Positionieren, Lesen und Schreiben eines Satzes auf eine relative Datei.

10000 – 10080 Kleine Lese-Routine zum Orientieren vor dem Sortieren.

30000 – 30120 Quicksort-Unterprogramm

Ich möchte an dieser Stelle nochmals darauf hinweisen, daß das hier vorgestellte Sortierprogramm keineswegs das bestmögliche ist. Verbesserungen sind insbesondere in punkto Sätze aus dem oberen Bereich in die Leersätze übertragen, möglich. Bevor Sie anhand eigener Ideen Verbesserungen vornehmen, sollten Sie jedoch stets bedenken, ob diese Verbesserungen Disketten und Floppy-Laufwerk schonend genug behandeln – also möglichst wenige Schreib-Lese-Operationen ausführen.

Arbeiten mit dem Super-Sort-Programm

Nach dem Programmstart müssen Sie den Namen der Datei angeben, die Sie sortieren möchten (Disk einlegen!).

Danach haben Sie die Auswahlmöglichkeit unter den drei Menüpunkten »Ganze Datei sortieren«, »Teilbereich sortieren« und »Lesen«. »Lesen« funktioniert wie beim relativen Dateien-Editor, und dient hier lediglich zur Orientierung für Sie. Wählen Sie »Ganze Datei sortieren«, so wird die gesamte Datei sortiert. Das Satzende wird automatisch durch Abfragen des Floppykanals nach Fehler 50 ermittelt.

»Teilbereich ermitteln« erlaubt es, einen bestimmten Bereich der Datei zu ordnen. Vor dem Sortieren müssen Sie noch angeben, nach welchem Stringbereich die Sortierung erfolgen soll – also welcher Stringbereich verglichen wird. Diese Funktion ist besonders wichtig beim Sortieren von Dateien mit mehr als einem Schlüssel.

Alle Zeilen im Programm, die mit REM SUPERMIKE gekennzeichnet sind, können weggelassen werden (zur Geschwindigkeitserhöhung). Sie ermöglichen es, den ganzen Sortiervorgang auf dem Monitor mitzuverfolgen.

Bedenken Sie, daß das Sortieren von Dateien mit mehreren 1000 Sätzen sehr viel Zeit in Anspruch nimmt. Es lohnt sich dann kaum, beim Computer zu warten.

Geheimtip: Sortiervorgang vor dem Schlafengehen starten – am nächsten Morgen ist alles vorbei!

Wer Angst vor einer zu hohen Stromrechnung hat, kann ein Schalter-Interface an den User-Port anschließen und das Programm so modifizieren, daß sich der Computer samt Peripherie am Ende des Sortiervorganges selbst abschaltet!

Eine bessere Lösung

Unser Super-Port funktioniert einwandfrei. Wer nur ab und zu sortieren will, ist mit ihm gut bedient. Die folgende Sortiermöglichkeit ist eine wesentliche Verbesserung gegenüber unserer ersten. Wer die Handhabung von Puffern und Zeigern verstanden hat, wird auch mit der folgenden Theorie keine Probleme haben.

Unser Super-Sort 1 benötigt noch ziemlich viele Disk-Zugriffe, um den Puffer zu füllen. Es gibt jedoch eine wesentlich schnellere und schonendere Lösung:

Theorie zum Super-Sort 2

Auch hier wird wieder ein Puffer der Größe M% verwendet. Zu Beginn des Sortiervorganges wird der Puffer einfach mit

```

10 REM ***** <060>
15 REM * PROGRAMM ZUM AENDERN VON * <000>
20 REM * DISKNAME UND ID * <035>
25 REM ***** <075>
30 PRINT "CLR,RVSON)DISK-UTILITY 1(26SPACE
   ,RVOFF)"; <024>
35 PRINT " (2DOWN,5SPACE)DISK-NAME AENDERN(3
   SPACE)-(2SPACE)'1' <166>
40 PRINT " (5SPACE)DISK-ID(3SPACE)AENDERN(3S
   PACE)-(2SPACE)'2' <017>
42 PRINT " (5SPACE)ENDE(16SPACE)-(2SPACE)'3' <241>
45 GET X$: IF X$ <> "1" AND X$ <> "2" AND X$ <> "3"
   THEN 45 <205>
46 IF X$="3" THEN PRINT " (2DOWN)*** FERTIG *
   ***:END <033>
50 OPEN 15,8,15:REM FLOPPYKANAL OEFFNEN <148>
55 PRINT#15,"I0":OPEN 1,8,3,"#":PRINT#15,"
   U1:3'0",18,0": IF X$="1" THEN 90 <205>
60 PRINT#15,"B-P:3,162":GET#1,A$,B$,C$,D$,
   E$:PRINT " (DOWN)ALTER DISK-ID IST (SPACE,
   RVSON,SPACE)";A$; <070>
65 PRINT B$;C$;D$;E$:INPUT " (DOWN)NEUER DIS
   K-ID ";F$ <070>
70 IF LEN(F$)<>4 THEN F$=LEFT$(F$+" (6SPACE
   3",5) <097>
75 PRINT#15,"B-P:3,162":PRINT#1,F$;:PRINT#
   15,"U2:3'DV",18,0 <118>
80 I$="I"+MID$(STR$(DV),2):PRINT#15,I$ <072>
85 RUN <127>
90 PRINT#15,"B-P:3,144":R$="":FOR L=1 TO 1
   6:GET#1,A$:R$=R$+A$:NEXT L <029>
95 PRINT " (DOWN)ALTER DISK-NAME IST "; " (RVS
   ON)"R$ " (RVOFF)" <089>
100 INPUT " (DOWN)NEUER DISK-NAME ";NE$ <148>
105 NE$=LEFT$(NE$+" (14SPACE)",16) <248>
110 PRINT#15,"B-P:3,144":PRINT#1,NE$;:PRIN
   T#15,"U2:3'DV",18,0":GOTO 80 <098>

```

Listing 25. Disk-Utility

```

51200 REM UP - DISK EINLEGEN ----- <123>
51210 PRINT " (RVSON)BITTE DISK-SEITE";SIDE;
   " (LEFT,SPACE)EINLEGEN ! (RVOFF)":POKE
   198,0:WAIT 198,1 <174>
51240 OPEN 2,8,2,"#":PRINT#15,"U1 2 0 18 0
   ":PRINT#15,"B-P=";2;162:GET#2,I$ <089>
51250 CLOSE 2: IF SIDE<>VAL(I$) THEN 51200 <090>
51260 RETURN <007>

```

Listing 26. Ein nützliches Unterprogramm

den ersten M%-Sätzen der Datei gefüllt. Wir suchen also **nicht** wie bei Super-Sort 1 die M% niedrigsten Sätze heraus.

Nun wird der gesamte Pufferinhalt mit dem Quicksort sortiert. Der nächste Schritt besteht darin, die gesamte Datei vom Zeiger beginnend bis zum Ende zu durchlaufen. Der Zeiger steht beim ersten Sortierabschnitt bei Satz M%, beim zweiten Durchlauf auf 2mal M% etc. Bei jedem eingelesenen Satz wird geprüft, ob dieser kleiner ist als der größte Satz im Puffer (steht beim sortierten Puffer an letzter Stelle). Ist dies nicht der Fall, wird der nächste Satz gelesen. Tritt aber nun der Fall ein, daß der Satz kleiner als der größte Satz des Puffers ist, passiert folgendes:

Wir nennen den eingelesenen Satz nun einmal A\$. A\$ muß nun irgendwie in den Puffer geschrieben werden. Anhand der binären Suchmethode (an anderer Stelle des Kurses beschrieben) ermitteln wir, wo im Puffer A\$ eingefügt werden muß. Der letzte Satz im Puffer wird dadurch aus dem Puffer hinausgeworfen (wir müssen schließlich Platz für A\$ schaffen). Damit er jedoch nicht verlorengeht, wird er wieder in die Datei zurückgeschrieben – und zwar an die Stelle, an der A\$, das sich nun im Puffer befindet, gestanden hat.

Diese Methode hat den Vorteil, daß pro Pufferfüllung die Datei nur ein einziges Mal durchlaufen werden muß. Auch das Problem, Platz für den Puffer-Zurückschreibevorgang zu

schaffen, entfällt. Bei Methode 2 wird zwar wesentlich mehr innerhalb des Rechners sortiert, es wird jedoch soviel Zeit bei Diskoperationen gespart, daß Methode 2 wesentlich schneller und Disk-schonender ist.

Wie gesagt, für diesen Kurs reicht Super-Sort I vollkommen aus. Wer viel sortieren will, sollte sich jedoch ein Sortierprogramm der oben beschriebenen Methode schreiben.

Trick am Rande: Diskettenseite abfragen

An dieser Stelle möchte ich einmal ein Problem abhandeln, das immer dann auftaucht, wenn man Programme schreiben will, die sich über zwei oder mehrere Disk-Seiten erstrecken. Bei Adventure-Spielen und KI-Programmen, die sich selbst verwalten, ist dies besonders häufig der Fall. Im Programm kommt dann früher oder später die Anweisung **BITTE DISK-SEITE X EINLEGEN**. Wie kann man nun innerhalb eines Programmes feststellen, ob auch die richtige Disk-Seite eingelegt worden ist? Dem Anfänger bereitet dies oft Schwierigkeiten, da Probleme wie diese nicht im Bedienungshandbuch des Floppy-Laufwerks behandelt werden. Die Lösung liegt einfach darin, den ID der eingelegten Disk einzulesen. Dies setzt natürlich voraus, daß der ID der ersten Diskette 01, der der zweiten 02 etc. beträgt. Im folgenden finden Sie deshalb zunächst ein kleines Hilfsprogramm, mit dem es möglich ist,

Diskname und ID einer Disk nachträglich zu verändern ohne, daß die Disk formatiert werden muß beziehungsweise ein sich auf ihr befindliches Programm verloren geht (Listing 25).

Beachten Sie bitte, daß der Diskname nicht mehr als 17 und der Disk ID nicht mehr als 5 Zeichen Länge haben darf. Nun brauchen wir nur noch ein kleines Basic-Unterprogramm, mit dem der Disk-ID gelesen werden kann (Listing 26).

Bevor das Programm mittels **GOSUB 51200** aufgerufen werden kann, muß der Floppykanal 15 geöffnet sein. Anschließend muß noch durch die Variable **SIDE** (beziehungsweise **SI**) bestimmt werden, welche Seite eingelegt werden soll. Das Unterprogramm führt erst dann ein **RETURN** aus, wenn die gewünschte Seite eingelegt ist. Innerhalb eines Basic-Programmes sieht dies dann so aus:

```
10 OPEN15,8,15
20 SIDE=2
30 GOSUB51200
40 END
```

Es bietet sich an, den ID der ersten Seite als **1SIDE**, den der zweiten Seite als **2SIDE** etc. zu benennen, da eine einzige Ziffer beim ID nicht ausreicht!

Vergessen Sie auch nicht aus Sicherheitsgründen, nach einem Disk-Wechsel immer neu zu initialisieren.

Sie können dieses Unterprogramm leicht in eigenen Basic-Programme einbauen. Das Disk- und ID-Änder-Programm speichern Sie am besten wieder auf Ihrer Utility-Disk, auf der sich auch die Editoren befinden. (M. Nickles/bs)

