

Daten verwalten mit der Datasette

Mit der Datasette kann man nicht nur Programme laden und speichern, sondern auch Daten verwalten. Wir zeigen Ihnen, wie das gemacht wird und was dabei beachtet werden muß.

Im folgenden Artikel soll dargestellt werden, welche Möglichkeiten zur Datenspeicherung die Datasette außer dem Speichern und Laden von Programmen bietet.

Die Arbeit mit Programmen ist unproblematisch und muß daher wohl nicht näher beschrieben werden. Sie alle wissen, daß zum Speichern eines Programms der Befehl »SAVE« und zum Laden der Befehl »LOAD« zur Verfügung steht, daß der Benutzer die Möglichkeit hat, ein Programm unter einem bestimmten Namen zu speichern (SAVE»TESTPROG«), und mit dem Befehl »VERIFY« überprüfen kann, ob das Speichern fehlerfrei gelang.

Probleme können vor allem bei der Verwendung von Kassetten großer Länge auftreten. Ich persönlich verwende seit langem sogenannte »Datenkassetten« mit einer Länge von 10 Minuten pro Seite, die mir noch nie Ärger bereitet haben.

Sollten Sie des öfteren einen »LOAD ERROR« oder einen »VERIFY ERROR« auf dem Bildschirm erhalten, ist wahrscheinlich entweder der Tonkopf Ihrer Datasette nicht einwandfrei justiert oder aber verschmutzt.

Ebenso wie bei jedem Kassettenrecorder sollte zuerst eine Tonkopfreinigung mit einer Reinigungskassette oder einem Reinigungsspray versucht werden. Wenn sich auch nach der Reinigung Probleme mit dem Laden von Programmen ergeben, ist der Tonkopf meist dejustiert und ein Gang zur Fachwerkstatt fällig, die über die zur einwandfreien Justierung nötigen Meßgeräte verfügt.

Die Datasette wird meistens zum Laden und Speichern von Programmen verwendet. Ihre Möglichkeiten sind damit jedoch bei weitem nicht ausgeschöpft. Mit diesem Gerät lassen sich nämlich Daten beliebiger Art verwalten, das heißt außer Programmen auch Termine, Benzinkosten, Adreß- oder Schallplattendateien und vieles mehr.

Die Verwaltung solcher Daten ist leider weitaus komplizierter als das Laden und Speichern eines normalen Programms. Im Handbuch des C 16/C 116 finden Sie dies nur sehr unzureichend beschrieben.

Grundsätzlich lassen sich mit der Datasette nur sogenannte »sequentielle« Dateien verwalten, im Gegensatz zur Floppy-Disk, die auch »Direktzugriffsdateien« verwalten kann. Wie der Name bereits sagt, kann auf alle Daten einer Direktzugriffsdatei unmittelbar zugegriffen werden. Wenn zum Beispiel die Telefonnummer des Herrn »Müller« gesucht wird, greift man auf jene Stelle in der gesamten Datei zu, an der sich der Datensatz »Müller« befindet. Der komplette Datensatz, der zum Beispiel aus den Teilen »Name: Müller«, »Vorname: Gerd«, »Adresse: München« und »Telefonnr.: 1234/56789« bestehen könnte, wird eingelesen und auf dem Bildschirm ausgegeben.

Ein solcher gezielter Zugriff auf bestimmte Teile einer Datei ist mit der Datasette leider nicht möglich. Um auf gesuchte Daten zuzugreifen, muß die gesamte Datei ab dem Dateian-

fang gelesen werden, bis die Bandposition erreicht wird, an der die gesuchten Daten abgespeichert wurden.

Zur Verdeutlichung des Unterschieds zwischen direktem und sequentielltem Zugriff bietet sich der Vergleich von Plattenspieler und Kassettenrecorder an: Um ein bestimmtes Lied zu hören, kann der Tonarm des Plattenspielers direkt zu jener Stelle der Schallplatte bewegt werden, an der das Stück beginnt. Beim Kassettenrecorder ist dieser direkte Zugriff nicht möglich. Das Band muß vor- oder zurückgespult werden, bis das jeweilige Lied gefunden wurde, die einzelnen Stücke werden »sequentiell« (der Reihe nach) abgesucht.

Halten wir somit fest: Mit einer Datasette als externem Speicher können nur sequentielle Dateien verwaltet werden, das heißt, Daten können nur der Reihe nach auf das Band geschrieben oder gelesen werden.

Nehmen wir nun an, Sie haben ein Programm zur Verwaltung der Adressen Ihrer Freunde, Bekannten, Verwandten etc. geschrieben, alle Adressen bereits eingetippt und gespeichert. Einen Monat später wollen Sie nun drei dieser Bekannten anschreiben, um sie zu Ihrer Geburtstagsfeier einzuladen.

Da Sie sich nicht mehr exakt an die Adressen dieser Bekannten erinnern können, wollen Sie in der Adreßdatei nachschauen. Sie legen die Kassette ein, auf der sich die Datei befindet, spulen sie zum Dateianfang zurück und suchen zum Beispiel nach »Andreas«. Der erste Bekannte wird nun gesucht. Wenn er gefunden wird, gibt Ihr Programm die komplette Adresse auf dem Bildschirm aus, und Sie können den ersten Brief adressieren.

Nun spulen Sie die Kassette wieder zum Dateianfang zurück, suchen nach dem zweiten Bekannten und notieren sich auch dessen Adresse, sobald sie auf dem Bildschirm erscheint. Die Kassette wird nun ein letztes Mal zurückgespult, und die Suche nach der dritten Adresse kann beginnen.

Wie Sie sehen, ist diese Art des Umgangs mit Ihrer Adreßdatei außerordentlich unkomfortabel. Jedesmal wenn bestimmte Daten gesucht werden sollen, müssen Sie die Kassette bis zum Anfang der Datei zurückspulen. Da die Datasette nicht gerade der schnellste Massenspeicher ist, kann die Suche geraume Zeit in Anspruch nehmen, je nachdem, wie umfangreich Ihre Datei ist.

In der Praxis wählt man daher eine andere Methode des Umgangs mit sequentiellen Dateien: Zu Beginn der Arbeit lädt man die komplette Datei vom Band in den Computerspeicher, das heißt, man liest jeden einzelnen Datensatz ein. Wird eine bestimmte Adresse gesucht, erfolgt die Suche nicht mehr auf der Kassette, sondern im Computerspeicher, der ja nun alle Adressen enthält. Diese Suche im Speicher des C 16 verläuft erheblich schneller als die Suche auf der Kassette. Der Nachteil dieser Methode besteht in der Beschränkung der maximalen Dateigröße. Da alle Daten in den Computer geladen werden, darf die Datei höchstens ebenso groß sein wie der verfügbare freie Speicherplatz. Beim C 16 stehen 12 KByte an freiem Speicher zur Verfügung, von dem jedoch der Platz abgezogen werden muß, den das Dateiverwaltungsprogramm selbst belegt. Da jedes Programm Variablen enthält, die ebenfalls Platz im Speicher benötigen, wird der für die Datei verfügbare freie Computerspeicher noch weiter eingeschränkt.

Ein Beispiel: Nehmen wir an, das Programm und die von ihm benötigten Variablen belegen 6 KByte. Für die Datei stehen somit noch $12 - 6 = 6$ KByte zur Verfügung. Da ein Kilo-byte 1024 Zeichen entspricht, darf die Datei bis zu $6 \cdot 1024 = 6144$ Zeichen enthalten. Wenn eine Adresse, die aus den Teilen »Name«, »Vorname«, »Straße«, »Wohnort« und »Telefonnr.« besteht, im Durchschnitt zirka 80 Zeichen enthält, kann die Datei etwa $6144 / 80 = 76,8$ Datensätze enthalten. Wir sehen an diesem Beispiel, daß uns der C 16 in Verbin-

derung mit der Datasette die Verwaltung kleinerer Adreßdateien erlaubt, jedoch ungeeignet ist für den kaufmännischen Einsatz, zum Beispiel die Verwaltung eines Lagers mit mehreren tausend Artikeln. Im folgenden werden wir uns daher auf den Einsatz im privaten Bereich beschränken.

Welche Befehle der C 16 zum Speichern und Laden von Dateien besitzt, will ich nun anhand eines kleinen Programms besprechen, das den Durchschnittsverbrauch eines Kraftfahrzeuges ermitteln soll. Um den Durchschnittsverbrauch zu berechnen, benötigt unser Programm Angaben über die gefahrenen Kilometer und den gesamten Benzinverbrauch. Der Gesamtverbrauch wird durch die Kilometerzahl geteilt und mit 100 multipliziert.

Beispiel:

Gefahrene Kilometer: 3435

Gesamtverbrauch: 273 Liter

Durchschnittsverbrauch pro 100 Kilometer:

$273/3435 * 100 = 7,9$ Liter

Selbstverständlich können wir den Durchschnittsverbrauch jederzeit auch ohne unseren C 16 ermitteln. Mit dem C 16 verringert sich der Aufwand hierzu jedoch erheblich: jedesmal wenn wir tanken, geben wir anschließend die Literzahl und den momentanen Kilometerstand ein. Alle nötigen Berechnungen soll unser Programm vornehmen.

```
200 rem durchschnittsverbrauch
210 ak=10000:rem anfangskilometerstand
220 :
230 input" anzahl getankter liter" ;li
240 sl=sl+li:rem summe liter
250 :
260 input" momentaner kilometerstand" ;km
270 sk=km-ak:rem summe kilometer
280 :
290 dv=sl/sk*100:rem verbrauch
300 print" durchschnittsverbrauch:" dv" l/100 km"
310 :
320 goto 230:rem naechste eingabe
```

Programmablauf:

Wenn Sie dieses Programm benutzen, müssen Sie zuerst eine Programmzeile ändern: In Zeile 210 ist der Kilometerstand zu Beginn der Programm Benutzung festgelegt. Korrigieren Sie bitte den als Beispiel verwendeten Wert 10 000 entsprechend Ihrem Tachostand.

Wenn Sie das Programm starten, werden Sie nach der jeweiligen Literzahl »li« gefragt. Alle von Ihnen nach Tankvorgängen eingegebenen Literzahlen werden addiert (Zeile 240). Die Variable »sl« stellt daher den bisherigen Gesamtverbrauch dar.

Anschließend werden Sie nach dem momentanen Kilometerstand gefragt. Die Summe der gefahrenen Kilometer ergibt sich aus der Differenz zwischen dem momentanen Kilometerstand »km« und dem Stand zu Beginn der Ermittlung des Verbrauchs »ak« (Zeile 270).

Wie besprochen wird nun der Durchschnittsverbrauch ermittelt, indem der Gesamtverbrauch »sl« durch die gefahrenen Kilometer »sk« geteilt und das Ergebnis mit 100 multipliziert wird (Zeile 290). Der in der Variablen »dv« enthaltene Durchschnittsverbrauch wird errechnet und auf dem Bildschirm ausgegeben.

Den Abschluß des Programms bildet ein Sprung zum Beginn der Abfragen (Zeile 320). Wenn Sie Ihren C 16 Tag und Nacht eingeschaltet lassen, wird das Programm geduldig warten, bis Sie wieder tanken und erneut Literzahl und Kilometerstand eingeben.

Dieses Programm funktioniert zwar einwandfrei (probieren Sie's aus), hat jedoch den Nachteil, daß Ihr C 16 für keinen anderen Zweck mehr verwendbar ist. Sobald Sie das Pro-

gramm unterbrechen und ein anderes Programm eingeben oder den Computer ausschalten, geht der Inhalt der Variablen »sl«, das heißt die Summe getankter Liter, verloren, und alle folgenden Berechnungen, die auf diesem Wert basieren, sind falsch!

Dieser Wert muß daher unbedingt nach jeder Programm Benutzung in einer Datei abgespeichert werden, und vor der eigentlichen Dateneingabe und Verbrauchsermittlung muß diese Datei in den Computerspeicher geladen werden.

Datenkanal öffnen:

Um Daten auf Band zu schreiben oder zu lesen, muß immer ein sogenannter »Datenkanal« geöffnet werden. Diesen Kanal kann man sich als Verbindung zwischen C 16 und Kassettenrecorder vorstellen, über den die Daten ausgetauscht werden. Der zugehörige Befehl besitzt folgende Syntax:

```
OPEN (LF),(GN),(SA)," (FN)"
```

LF: logische Filenummer, eine beliebige Zahl zwischen 1 und 255. Wenn im weiteren Programmablauf Daten geschrieben oder gelesen werden sollen, muß immer diese Filenummer angegeben werden.

GN: Gerätenummer. Sie gibt an, mit welchem Gerät (Drucker, Datasette, Floppy) Daten ausgetauscht werden sollen. Die Datasette besitzt die Gerätenummer 1, die Floppy die Gerätenummer 8.

SA: Sekundäradresse, im Fall der Datasette eine Null, Eins oder Zwei. Benötigt werden üblicherweise nur die Zahlen Null und Eins, wobei Null bedeutet, daß Daten vom Band gelesen werden sollen, und Eins, daß Daten auf das Band geschrieben werden.

FN: Filename (=Dateiname), der Name der betreffenden Datei. Die Angabe eines Filenamens ist sehr vorteilhaft, da mehrere Dateien auf einer Kassette enthalten sein können. Wenn wir eine bestimmte Datei lesen wollen, wird unser C 16 alle vorhergehenden Dateien überlesen, bis er zu jener Bandstelle kommt, an der die Datei beginnt, deren Filamen wir im OPEN-Befehl angeben.

Beispiele:

```
OPEN 1,1,0," TESTDATEI"
```

Öffnet einen Datenkanal zwischen C 16 und Datasette (Gerätenr. 1). Die Richtung des Datenaustauschs verläuft von der Datasette zum C 16 (Sekundäradresse 0 = Daten lesen), die gesuchte Datei hat den Namen »TESTDATEI«.

```
OPEN 1,1,1," VERBRAUCH"
```

Öffnet einen Datenkanal zwischen C 16 und Datasette. Da als Sekundäradresse eine Eins angegeben wurde, sollen Daten auf das Band geschrieben werden. Die Datei erhält den Namen »VERBRAUCH«.

Daten auf Band schreiben:

Um ein Programm zu laden oder zu speichern genügt die Eingabe eines einzigen Befehls (»LOAD«/»SAVE«), weitere Angaben sind nicht erforderlich.

Die Arbeit mit Dateien ist erheblich aufwendiger, da wir unserem Computer angeben müssen, welche Daten abgespeichert oder eingelesen werden sollen. Die Befehlssyntax zum Schreiben von Daten auf Kassette:

```
PRINT # (LF), (DATEN)
```

LF: Logische Filenummer, die wir beim Öffnen des Datenkanals angeben.

DATEN: Variable, die die zu schreibenden Daten enthält oder unmittelbare Angabe der Daten.

Beispiele:

```
PRINT #1,A$
```

Schreibt den Inhalt der Variablen A\$ auf Band (wenn zuvor ein Kanal zum Schreiben unter der log. Filenummer Eins geöffnet wurde).

```
PRINT #1,X
```

Schreibt den Inhalt der numerischen Variablen X auf Band.

```
PRINT #1,"DIES IST EIN TEST"
```

Schreibt die Zeichenkette »DIES IST EIN TEST« auf Band.

Die Ähnlichkeit mit dem PRINT-Befehl, der der Datenausgabe auf dem Bildschirm dient, ist unverkennbar. Der einzige Unterschied besteht in der Angabe der logischen Filenummer, damit unser Computer die Daten nicht auf dem Bildschirm ausgibt, sondern über den bereits geöffneten Datenkanal an die Datasette schickt.

Daten vom Band lesen:

Viele werden nun bereits ahnen, wie der entsprechende Befehl zum Einlesen von Daten lautet: INPUT #. Die genaue Syntax:

```
INPUT # (LF), (VARIABLE)
```

LF: Logische Filenummer, die wir beim Öffnen des Datenkanals angaben.

DATEN: Variable, in der die zu lesenden Daten übergeben werden sollen.

Beachten Sie auch beim INPUT #-Befehl die Analogie zum INPUT-Befehl, der eine über die Tastatur vorgenommene Eingabe einer Variablen zuweist. INPUT # erfüllt die gleiche Aufgabe für Eingaben von einem Peripheriegerät (Datasette, Floppy).

Beispiele:

```
INPUT #1,A$
```

Liest eine Zeichenkette und übergibt sie in der Variablen A\$.

```
INPUT #1,X
```

Liest eine Zahl und übergibt sie in der Variablen X.

Datenkanal schließen:

Wenn eine Datei komplett in den Computerspeicher gelesen oder auf Band geschrieben wurde, muß der Datenkanal wieder geschlossen werden. Vor allem nach dem Schreiben einer Datei kommt es zu Problemen, wenn dieser Vorgang nicht ordnungsgemäß durchgeführt wird: das spätere Einlesen der Datei ist dann nicht mehr möglich, unsere Daten sind verloren! Die Syntax des sogenannten »CLOSE«-Befehls:

```
CLOSE (LF)
```

LF: logische Filenummer, unter der der Kanal geöffnet wurde.

Beispiele:

```
CLOSE 1
```

Schließt einen unter der log. Filenummer Eins geöffneten Datenkanal.

```
CLOSE 7
```

Schließt einen unter der log. Filenummer Sieben geöffneten Datenkanal.

Halten wir fest: Um Daten vom Band zu lesen oder auf Band zu schreiben, muß zuerst ein Datenkanal mit dem OPEN-Befehl geöffnet werden. Mit Hilfe der Sekundäradresse können wir angeben, ob Daten gelesen oder geschrieben werden sollen. Lesen und schreiben zugleich ist nicht möglich!

Anschließend können beliebige Daten mit dem PRINT #-Befehl in eine Datei geschrieben oder mit dem INPUT #-Befehl gelesen werden. Beim Schreiben und Lesen von Daten können sowohl numerische als auch Stringvariablen verwendet werden.

Den Abschluß bildet das ordnungsgemäße Schließen der Datei mit dem CLOSE-Befehl, das vor allem nach Schreibvorgängen keinesfalls vergessen werden darf!

Wir besitzen nun das nötige Wissen, um unser Programmbeispiel vervollständigen zu können.

Datei speichern:

```
400 open 1,1,1,"verbrauch":rem datenkanal oeffnen
410 print#1,sl:rem gesamte literzahl speichern
420 close 1:rem kanal schliessen
```

In Zeile 400 wird ein Datenkanal zur Datasette zum Schreiben unter der logischen Filenummer Eins geöffnet (log. Filenummer 1, Gerätenr.1, Sekundäradresse 1). Die Datei erhält den Namen »VERBRAUCH«.

In Zeile 410 wird der Inhalt der Variablen »sl«, das heißt die gesamte Literzahl, in die Datei geschrieben.

In Zeile 420 wird der unter der log. Filenummer Eins geöffnete Datenkanal unter Angabe der gleichen Filenummer wieder geschlossen.

Datei einlesen:

```
100 open 1,1,0,"verbrauch":rem datenkanal oeffnen
110 input#1,sl:rem gesamte literzahl einlesen
120 close 1:rem kanal schliessen
```

Zeile 100: Im OPEN-Befehl zum Einlesen der Datei muß unbedingt der gleiche Dateiname angegeben werden, unter dem die Datei gespeichert wurde. Wichtig ist außerdem die Angabe einer Null als Sekundäradresse (= Lesen von Daten).

In Zeile 110 wird die Literzahl in die Variable »sl« eingelesen und in Zeile 120 der Datenkanal wieder geschlossen.

Vollständiges Programmlisting:

```
90 rem datei einlesen
100 open 1,1,0,"verbrauch":rem datenkanal
    oeffnen
110 input#1,sl:rem gesamte literzahl einlesen
120 close1:rem kanal schliessen
130 :
200 rem durchschnittsverbrauch
210 ak=10000:rem anfangskilometerstand, bitte
    individuell aendern
220 :
230 input" anzahl getankter liter" ;li
240 sl=sl+li:rem summe liter
250 :
260 input" momentaner kilometerstand" ;km
270 sk=km-ak:rem summe kilometer
280 :
290 dv=sl/sk*100:rem verbrauch
300 print" durchschnittsverbrauch:" dv" l/100 km"
310 print" bitte zurueckspulen und taste druecken"
320 getkey a$
380 :
390 rem datei speichern
400 open 1,1,1,"verbrauch":rem datenkanal
    oeffnen
410 print#1,sl:rem gesamte literzahl speichern
420 close1:rem kanal schliessen
```

Tippen Sie nun das vollständige Programm ab, ändern Sie Zeile 210 (Anfangskilometerstand) entsprechend Ihrem Tachostand ab, legen Sie eine leere Kassette ein und speichern Sie das Programm. Lassen Sie die Kassette bitte ein-

gelegt und stellen Sie das Zählwerk der Datasette auf »000«. Sie dürfen das Programm bei der ersten Benutzung keinesfalls mit »RUN« starten, da der C 16 sonst versuchen würde, die noch nicht vorhandene Datei »VERBRAUCH« zu laden (Zeilen 100–120)!

Starten Sie das Programm mit »RUN 200«, und geben Sie zum Testen beliebige – aber einigermaßen sinnvolle – Werte für die Anzahl getankter Liter und den momentanen Kilometerstand ein. Der bisherige Durchschnittsverbrauch wird nun ermittelt und auf dem Bildschirm ausgegeben.

Anschließend wird die Datei gespeichert. Sie erhalten die Aufforderung die Kassette zum Dateianfang zurückzuspulen und anschließend eine beliebige Taste zu drücken. Nach diesem Vorgang erscheint auf dem Bildschirm die bekannte Meldung »PRESS PLAY AND RECORD ON TAPE«. Drücken Sie beide Tasten gleichzeitig. Die Datei, die in unserem Programmbeispiel nur aus der Variablen »sl« besteht, wird gespeichert.

Spulen Sie nun zurück bis zum Zählerstand »000«, das heißt bis zu jener Bandposition, ab der die Datei gespeichert wurde, und starten Sie das Programm mit »RUN«. Sie erhalten die Aufforderung »PRESS PLAY ON TAPE«. Drücken Sie die entsprechende Taste der Datasette. Nachdem die Datei geladen wurde, ist in der Variablen »sl« somit die gesamte Literzahl enthalten. Nun können Sie neue Werte für die Anzahl der getankten Liter und den Kilometerstand eingeben. Das Programm ermittelt wiederum den Durchschnittsverbrauch und gibt ihn aus.

Spulen Sie nun wiederum bis zum Dateianfang zurück, bevor Sie die Tasten »PLAY« und »RECORD« betätigen. Die Datei wird erneut abgespeichert, wobei die alte Datei überschrieben wird.

Diese Vorgehensweise ist typisch für den Umgang mit Programmen, die sequentielle Dateien verwalten. Zusammenfassung der einzelnen Schritte:

1. Speichern Sie die Datei unmittelbar hinter dem Programm auf die Kassette. Wenn die Datei nach dem Programmstart eingelesen werden soll, ist kein Spulen nötig, da sich das Band automatisch am Dateianfang befindet.

2. Stellen Sie das Kassettenzählwerk nach dem Einlesen des Programms auf »000« zurück, um den Dateianfang zu kennzeichnen.

3. Wenn die Datei wieder gespeichert werden soll, spulen Sie zur Position »000« zurück, das heißt zum Dateianfang. Die alte Datei wird beim Speichern durch die neuen Daten überschrieben werden.

Damit Sie mit den beschriebenen Befehlen vertrauter werden, stelle ich ein weiteres Programmbeispiel zum Lesen und Schreiben von Daten vor:

```
100 a$=" open = oeffnen eines datenkanals"
110 b$=" print# = schreiben von daten"
120 c$=" input# = lesen von daten"
130 d$=" close = datenkanal schliessen"
140 :
150 rem datei schreiben
160 open 1,1,1," test"
170 print#1,a$
180 print#1,b$
190 print#1,c$
200 print#1,d$
210 close1
220 :
230 print" bitte zum dateianfang zurueckspulen"
240 print" und danach eine beliebige taste
druecken"
250 getkey a$
260 :
270 rem datei einlesen
```

```
280 open 1,1,0," test"
290 input#1,a$
300 input#1,b$
310 input#1,c$
320 input#1,d$
330 close1
340 :
350 printa$
360 printb$
370 printc$
380 printd$
```

Dieses Programm erzeugt mehrere Stringvariablen, schreibt sie in eine Datei, fordert Sie anschließend auf, zum Dateianfang zurückzuspulen und liest die Datei wieder ein.

Häufige Fehler beim Umgang mit sequentiellen Dateien:

1. Es kann vorkommen, daß Ihnen der C 16 einen »SYNTAX ERROR IN ...« ausgibt, Sie jedoch in der betreffenden Zeile keinen Fehler entdecken können. Meist handelt es sich um einen falsch eingegebenen PRINT #-Befehl. Wie Ihnen vielleicht bekannt ist, können Sie PRINT-Befehle mit dem Fragezeichen abkürzen, zum Beispiel: 10?»HALLO«. Beim LISTEN der Zeile erscheint der vollständige Befehl: 10PRINT»HALLO«. Der PRINT #-Befehl darf nicht (!) auf diese Weise abgekürzt werden. Dieser Fehler ist schwer zu entdecken, da beim LISTEN der betreffenden Zeile auch der auf diese Weise eingegebene PRINT #-Befehl vollständig und scheinbar korrekt erscheint, zum Beispiel: 10PRINT #1,A\$. Die einzige Möglichkeit, diesen Fehler zu beheben, ist die korrekte Neueingabe des Befehls, obwohl Sie im Listing des Programms keinen Unterschied entdecken werden. Ich gebe zu, daß dies schwer verständlich ist, und rate Ihnen daher, es selbst auszuprobieren.

2. »FILE DATA ERROR«: Dieser Fehler tritt meist auf, wenn beim Lesen einer Datei nicht die gleiche Reihenfolge wie beim Schreiben der Daten eingehalten wird. Wurden zum Beispiel folgende Variablen auf Kassette gespeichert:

```
10 print#1,a$
20 print#1,b
30 print#1,c$
```

und sollen sie wie folgt eingelesen werden:

```
10 input#1,b
20 input#1,a$
30 input#1,c$
```

dann erhalten Sie die erwähnte Fehlermeldung, da mit dem ersten INPUT #-Befehl eine numerische Variable eingelesen werden soll (»b«), sich an dieser Stelle der Datei jedoch alphanumerische Daten befinden, nämlich der Inhalt der Variablen »a\$«. Denken Sie daran, daß die Daten einer sequentiellen Datei der Reihe nach gelesen werden müssen und ein Überspringen von Daten nicht möglich ist!

3. Das Handbuch enthält eine Bemerkung, die viel Ärger bereiten kann. Angeblich soll es möglich sein, in einem PRINT #-Befehl eine Liste mehrerer Daten anzugeben und diese durch Kommata oder Semikolons zu trennen, zum Beispiel:

```
10 print#1,a$,b$,c$
```

Es soll nun möglich sein, diese Daten mit einem INPUT #-Befehl, in dem die gleichen Trennzeichen verwendet werden, wieder einzulesen:

```
10 input#1,a$,b$,c$
```

Diese Möglichkeit wäre zwar sehr komfortabel, hat jedoch ihre Tücken: Beim Einlesen kann der INPUT #-Befehl die einzelnen Daten nicht unterscheiden. Alle drei Daten (a\$,b\$,c\$)

werden in die erste im INPUT #-Befehl aufgeführte Variable »a\$« eingelesen, die danach den Inhalt »a\$+b\$+c\$« besitzt. Der C 16 ist nicht mehr in der Lage, das Ende einer der drei Zeichenketten zu erkennen, und behandelt alle drei wie eine einzige Zeichenkette. Verwenden Sie daher bitte zum Schreiben von Daten auf Kassette je einen PRINT #-Befehl zum Speichern einer Variablen.

Der GET #-Befehl:

Besondere Erwähnung verdient der Befehl GET # (LF), (VARIABLE). Mit ihm lassen sich mehrere Unzulänglichkeiten des INPUT #-Befehls ausmerzen. Mit dem PRINT #-Befehl können Zeichenketten mit einer Länge von maximal 255 Zeichen gespeichert werden. Der INPUT #-Befehl kann jedoch nur Zeichenketten einlesen, deren Länge höchstens 80 Zeichen beträgt (probieren Sie's aus!).

Mit dem Befehl GET # können Zeichenketten beliebiger Länge eingelesen werden. Dieser Befehl liest ein einziges Zeichen aus einer Datei ein und übergibt es in der angegebenen Variablen. Eine Zeichenkette kann auf folgende Weise komplett eingelesen werden:

```
10 open 1,1,0," test"
20 get #1,a$
30 if a$=chr$(13) then print zk$:end
40 zk$=zk$+a$
50 goto 20
```

Um dieses Programm zu verstehen, muß man wissen, daß nach dem PRINT #-Befehl ein Sonderzeichen auf das Band geschrieben wird. Dieses Sonderzeichen hat den ASCII-Code 13 (zum ASCII-Code siehe C 16-Handbuch) und dient als Trennzeichen zwischen den einzelnen Daten, die in einer Datei vorhanden sind. An diesem Trennzeichen erkennt unser C 16, ob eine Zahl oder Zeichenkette bereits komplett eingelesen wurde.

Das Programmbeispiel geht ebenso vor: Es liest Zeichen für Zeichen aus der Datei »TEST« und verknüpft die einzelnen Zeichen zum String »zk\$«. Wenn der ASCII-Code des gerade gelesenen Zeichens den Wert 13 hat (»asc(a\$)«), handelt es sich um das Trennzeichen, und die Zeichenkette wurde komplett eingelesen (Zeile 30).

Verwenden Sie bitte folgendes Programm, um eine Zeichenkette auf Band zu schreiben:

```
10 open 1,1,1," test"
20 a$=" dies ist ein test"
30 print #1,a$
40 close1
```

Wenn Sie diese Minidatei gespeichert haben, können Sie die Zeichenkette »dies ist ein test« mit dem obigen Programm wieder einlesen. Der Nachteil des GET #-Befehls besteht darin, daß das Einlesen von Daten erheblich langwieriger ist als mit dem schneller arbeitenden INPUT #-Befehl. Sollten Sie jedoch jemals das Problem sehr langer einzulesender Zeichenketten haben, denken Sie an diesen GET #.

Sequentielle Dateien mit der Floppy:

Wenn Sie zu den glücklichen Besitzern einer Floppy gehören, so können Sie selbstverständlich ebenfalls sequentielle Dateien verwalten. Außer den sequentiellen stehen Ihnen zudem noch die erwähnten Direktzugriffsdateien zur Verfügung. Um den doch etwas aufwendigeren Umgang mit Direktzugriffsdateien zu lernen, empfehle ich Ihnen die Lektüre einschlägiger Zeitschriften und Bücher, da eine ausführliche Beschreibung dieser Dateiformen den Rahmen dieses Artikels sprengen würde.

Zur Verwaltung sequentieller Dateien mit der Floppy kön-

nen Sie von dem bereits Beschriebenen ausgehen, wenn Sie folgende Unterschiede zur Datasette beachten:

1. Im OPEN-Befehl müssen Sie selbstverständlich die Gerätenummer der Floppy (acht) angeben.
2. Verwenden Sie als Sekundäradresse bitte nicht die Zahl 15, da diese Sekundäradresse einem besonderen Zweck vorbehalten ist.
3. Die Angabe, ob Daten gelesen oder in eine Datei geschrieben werden sollen, erfolgt nicht mehr in der Sekundäradresse, sondern im Filenamen, dem die Zeichenkette »s,w« für Schreiben oder »s,r« für Lesen folgen muß.
4. Um eine Datei zu überschreiben, müssen sich vor dem Dateinamen die beiden Zeichen »@:« befinden. Das Überschreiben geschieht nicht automatisch wie bei der Datasette.

Beispiele:

```
OPEN 2,8,2," TEST,S,R"
```

Öffnet die Datei »TEST« zum Lesen von Daten.

```
OPEN 2,8,2," TEST,S,W"
```

Öffnet die neue Datei »TEST« zum Schreiben von Daten.

```
OPEN 2,8,2,"@:TEST,S,W"
```

Öffnet die Datei »TEST« zum Schreiben von Daten. Eine unter diesem Namen bereits existierende Datei wird überschrieben.

Die Syntax aller übrigen beschriebenen Befehle bleibt unverändert. Zum Abschluß wünsche ich Ihnen viel Erfolg bei der Erstellung Ihrer eigenen Adress-, Haushaltskosten- und sonstigen Dateien, ob sich diese nun auf Kassette oder Diskette befinden mögen.

(Said Baloui/kn)

