



Mein Computer versteht mich nicht!

Jeder Programmierer, ob Anfänger oder Profi, macht beim Programmieren Fehler. Der Anfänger jedoch kennt meist den genauen Grund, und wie er ihn beheben kann, nicht. Dem wollen wir mit einer ausführlichen Fehlertabelle und vielen Tips zur Suche der Fehlerursache abhelfen.

Obwohl dieser Artikel eigentlich mehr als Nachschlagewerk für den Notfall gedacht ist, schadet es nicht, wenn Sie ihn einmal komplett durchlesen. Sie haben so ein »Grundwissen«, auf das Sie bei Bedarf zurückgreifen können. Auch können Sie dann auf Zusammenhänge zwischen den einzelnen Fehlern schließen. Doch kommen wir nun zum Praxisteil:

Sie haben sich einen neuen C64 besorgt, packen ihn voller Erwartung zu Hause aus, schließen ihn an und los geht's mit dem Programmieren. Aber o weh!, der Computer macht nicht das, was man von ihm verlangt. Es erscheinen Fehlermeldungen auf dem Bildschirm (erkennbar am vorangestellten »?« und daran, daß sich der Computer unerwartet mit »READY.« zurückmeldet). Was tun? Spätestens jetzt wird die Bedienungsanleitung zum C64 zur Hand genommen, um dem Fehler auf die Schliche zu kommen. Aufmerksam liest man das entsprechende Kapitel, doch was versteht man? Nichts!! Lauter Fachausdrücke, von denen man noch nie etwas gehört hat, helfen auch nicht weiter. Genau an diesem Punkt soll dieser Artikel ansetzen, um Ihnen bei der Fehlersuche und deren Beseitigung unterstützend unter die Arme zu greifen. Jeder mögliche Fehler wird ausführlich behandelt. Dabei ist jedem Fehler ein Absatz gewidmet, der wie folgt aufgebaut ist:

- Fehlermeldung, also das, was auf dem Bildschirm erscheint und kurze Erklärung des Fehlers
- Klärung der Fachausdrücke
- Wie entsteht der Fehler, mit Beispielen?
- Wie geht man vor, um die Fehlerursache zu suchen?
- Wie wird der Fehler beseitigt?

Hinweis: Bevor Sie ein neues Beispiel eintippen, sollten Sie erst das alte mit »NEW« löschen.

Fehler im Betriebssystem:

Wenn Sie mit einer RS232-Schnittstelle arbeiten, so ist folgendes zu beachten: Nach einem OPEN 1,2, ... also »Datenkanal zur RS232-Schnittstelle öffnen«, werden vom Computer sämtliche Variablen eines Basic-Programms gelöscht! Daher sollte dieser OPEN-Befehl unbedingt als erster Befehl in einem solchen Programm stehen.

?BAD SUBSCRIPT:

Der Fehler tritt auf, wenn versucht wird, eine Feldvariable anzusprechen, die nicht dimensioniert wurde.

Eine Feldvariable ist eine normale Variable wie »A« oder »TEST« mit zusätzlichem Index wie »A(5)« oder »TEST(X)«. Der Index in Klammern ist als Zeiger zu verstehen, der auf ein Element des Feldes mit dem Namen »A« oder »Test« zeigt. Felder, die mehr als elf Elemente enthalten, müssen mit dem Basic-Befehl DIM definiert werden. Felder, die weniger als elf Elemente enthalten, müssen nicht definiert werden. Zum Beispiel legt der Befehl DIM A(20) ein Feld mit insgesamt 21 Elementen fest (21 Elemente, weil die Feldvariable A(0) auch in der Dimensionierung enthalten ist). Wenn nach dieser Felddefinition versucht wird, der Feldvariablen »A(21)« einen Wert zuzuweisen »A(21)=123«, erscheint die Fehlermeldung »BAD SUBSCRIPT« auf dem Bildschirm. Die gleiche Fehlermeldung erhält man, wenn der Feldvariablen »A(11)« ein Wert

zugewiesen wird, ohne zuvor ein entsprechend großes Feld zu dimensionieren.

Tritt ein solcher Fehler in einem komplexeren Programm auf, ist zuerst zu überprüfen, wie groß der Index der Feldvariablen ist. Sollte er größer als 10 sein, ist die dazugehörige DIM-Anweisung zu suchen. Wenn eine solche nicht existiert, erweitern Sie eine der ersten Zeilen eben um diese DIM-Anweisung.

BREAK:

Bei dieser Meldung handelt es sich nicht um eine Fehlermeldung im herkömmlichen Sinn. Vielmehr macht der Computer darauf aufmerksam, daß das Programm oder ein Speicher- oder Ladevorgang absichtlich unterbrochen wurde. Diese Meldung existiert in zwei verschiedenen Versionen.

?BREAK ERROR zeigt an, daß beim Laden oder Speichern eines Programms die RUN/STOP-Taste gedrückt wurde.

?BREAK IN XXXX weist darauf hin, daß während eines Programmlaufes die RUN/STOP-Taste gedrückt wurde, oder daß sich in der angezeigten Zeile der Basic-Befehl »STOP« oder »BREAK« befindet. In einem solchen Fall kann das Programm mit dem Basic-Befehl »CONT« für »continue« fortgesetzt werden, vorausgesetzt es erscheint nicht die Fehlermeldung

?CAN'T CONTINUE: In einem solchen Fall ist einer der folgenden Punkte eingetreten:

1) Das Programm wurde nicht, wie oben beschrieben, absichtlich unterbrochen, sondern der Computer selbst hat es angehalten, weil ein »SYNTAX ERROR« auftrat (der Basic-Übersetzer konnte einen Befehl im Programm nicht übersetzen/verstehen).

2) Im Direktmodus wurden mit dem Basic-Befehl »CLR« die Variablen gelöscht; das heißt Sie haben, nachdem das Programm unterbrochen wurde, folgendes eingegeben:

```
CLR <RETURN>
```

3) Am Programm wurde nach einer Unterbrechung etwas geändert.

4) Im Direktmodus trat nach einer Unterbrechung ein Fehler auf.

Beispiel: Sie starten ein beliebiges Basic-Programm, halten es mit der RUN/STOP-Taste an und geben ein

```
A <RETURN>
```

Der Computer meldet einen »SYNTAX ERROR«. Klar, den Befehl »A« gibt es nicht. Nun versuchen Sie das Programm mit »CONT« fortzusetzen. Der Computer gibt die Fehlermeldung »CAN'T CONTINUE« aus.

5) Wenn Sie den Befehl »CONT« eingeben, bevor ein Basic-Programm gestartet wurde, reagiert der Computer ebenfalls mit »CAN'T CONTINUE«.

?DEVICE NOT PRESENT:

Es wurde versucht, ein peripheres Gerät wie Drucker, Diskettenlaufwerk oder ähnliches anzusprechen, das entweder nicht eingeschaltet oder überhaupt nicht vorhanden ist oder nicht reagiert. Dieser Fehler tritt gewöhnlich nicht beim eigentlichen »OPEN«-Befehl auf, wie das folgende Beispiel zeigt, sondern dann, wenn versucht wird, das Gerät mit den Befehlen »GET #, INPUT #, PRINT #« anzusprechen. Beispiel:

```
OPEN 7,7 <RETURN>
```

```
PRINT #7,"ABCDEF" <RETURN>
```

Meldet der Computer einen »DEVICE NOT PRESENT«-Error, sollten Sie zunächst überprüfen, ob alle erforderlichen Geräte (Drucker, Diskettenlaufwerk und so weiter) eingeschaltet und mit dem Computer verbunden sind. Ist das der Fall, haben Sie mit Sicherheit eine falsche Geräteadresse (zweite Zahl hinter dem OPEN-Befehl) gewählt. Um den für die Fehlermeldung verantwortlichen OPEN-Befehl zu finden, lassen Sie sich die in der Fehlermeldung angegebenen, Zeile LISTen. Suchen Sie in dieser Zeile einen Basic-Befehl, dem unmittelbar das Nummernzeichen (#) und eine Zahl (File-

nummer) folgt. Sind mehrere solcher Befehle vorhanden, ist die Zeile aufzuteilen und zwar so, daß jede Zeile nur einen Befehl mit Nummernzeichen enthält. Beispiel:

```
10 PRINT #4,CHR$(32):GET #8,A$
```

Diese Zeile ist in die folgenden zwei Zeilen aufzuteilen:

```
10 PRINT #4,CHR$(32)
```

```
11 GET #8,A$
```

Starten Sie nun das Programm erneut mit RUN und lassen sich wieder die in der Fehlermeldung angegebene Zeile LISTen. Im nächsten Schritt ist der OPEN-Befehl im Programm zu suchen, der die gleiche Filenummer (erste Zahl hinter dem OPEN-Befehl) hat, wie der Befehl (Zahl hinter dem Nummernzeichen), bei dem der Fehler aufgetreten ist. Die zweite Zahl hinter dem fehlerträchtigen OPEN-Befehl ist die falsche Geräteadresse. Damit Sie dort die richtige Geräteadresse einsetzen können, folgt eine Tabelle aller möglichen Geräteadressen:

1 Datasette

2 RS232

3 Bildschirm

4 Drucker

5 Drucker, falls im Drucker die Geräteadresse auf 5 umgestellt wurde

6 Plotter 1520

7 frei für externe Geräte

8 bis 11 Diskettenlaufwerke

12 bis 15 frei für externe Geräte

?DIVISION BY ZERO:

Dieser Fehlermeldung erscheint nur dann, wenn durch Null dividiert wurde. Aber Vorsicht! Es ist nicht immer auf Anhieb ersichtlich, daß der Nenner tatsächlich Null ist. Ist zum Beispiel bei der Funktion »A=B/(C*D*E*F)« eine der Variablen »C, D, E, F« Null, so ist natürlich der gesamte Nenner Null. Dieses Beispiel ist vielleicht trivial, aber was machen Sie, wenn bei der Funktion »A=TAN (PI/2)« ein »DIVISION BY ZERO«-Error erscheint? Um das zu verstehen, muß man wissen, wie der Computer die TAN-Funktion behandelt. Er berechnet nämlich nicht direkt den Tangens sondern ermittelt ihn mit Hilfe der SIN- und COS-Funktion ($\tan(x) = \sin(x)/\cos(x)$). Wenn für $x = \frac{\pi}{2}$ eingesetzt wird ist der Nenner »COS ($\frac{\pi}{2}$)« Null, und es erscheint die Fehlermeldung.

Um den »DIVISION BY ZERO«-Error zu beheben, ist jede Variable im Nenner, falls vorhanden, mit dem Basic-Befehl »PRINT«, gefolgt von der Variablen, zu überprüfen. Ist die Variable gefunden, die den Fehler verursacht hat, müssen Sie dafür sorgen, daß sie niemals den Wert Null annehmen kann. Dies läßt sich durch eine Abfrage vor der Funktion realisieren. Beispiel:

```
IF C*D*E*F <> 0 THEN A=B/(C*D*E*F)
```

?EXTRA IGNORED:

Zeigt an, daß hinter dem Basic-Befehl INPUT weniger Variablen stehen, als eingegeben wurden. Beispiel:

```
10 INPUT A$,B$ <RETURN>
```

Nach der Eingabe wird dieser Einzeiler mit RUN <RETURN> gestartet. Es erscheint ein Fragezeichen (?) auf dem Bildschirm. Geben Sie nun ein:

```
TEST1,TEST2,TEST3 <RETURN>
```

Der Computer meldet einen »EXTRA IGNORED«-Error, denn »TEST1« wird in die Variable »A\$« und »TEST2« in die Variable »B\$« eingelesen. Für »TEST3« existiert keine Variable und genau dies meldet der Computer. Häufig entsteht eine solche Fehlermeldung unbeabsichtigt bei Zeichenketteneingaben, die Kommata enthalten. Man sollte daher darauf achten, wenn mit dem INPUT-Befehl gearbeitet wird, daß eine Zeichenkette, die in eine Variable eingelesen werden soll, kein Komma enthält. Durch das Komma wird nämlich dem Computer mitgeteilt, daß alle folgenden Zeichen bis zum nächsten Komma in die nächste Variable eingelesen werden sollen. Eine Zeichenkette darf neben dem Komma auch keine

Doppelpunkte enthalten; denn alle Zeichen, die hinter dem ersten Doppelpunkt stehen, werden vom Computer verschluckt. Der gleiche Fehler tritt auch bei dem Befehl »INPUT #« auf, jedoch wird in diesem Fall keine Fehlermeldung ausgegeben. Übrigens noch ein Tip: Bei älteren C64 darf der Cursor nicht nach oben oder unten verschoben werden, wenn der INPUT-Befehl eine Eingabe verlangt.

?FILE DATA ERROR:

Diese Fehlermeldung erscheint, wenn im Programm versucht wird, Zeichenketten vom Diskettenlaufwerk, von der Datasette oder der Tastatur in eine numerische Variable einzulesen. Beispiel:

```
10 OPEN1,0:REM TASTATUR ALS EINGABEGERÄT ÖFFNEN
20 INPUT#1,A:REM AUF EINGABE WARTEN
30 CLOSE1:REM EINGABEGERÄT »TASTATUR« SCHLIESSEN
```

Zeile 10 öffnet einen Kanal mit der Filenummer 1 und setzt die Tastatur (Geräteadresse 0) als Eingabegerät.

In Zeile 20 wird auf die Eingabe gewartet. Es können beliebige Zahlen (keine Buchstaben) eingegeben werden. Nach dem Drücken der RETURN-Taste wird in Zeile 30 der Kanal mit der Filenummer 1 geschlossen.

Sind die drei oben stehenden Zeilen abgetippt, läßt sich das Programm mit RUN starten. Im Gegensatz zum normalen INPUT-Befehl ohne dem Nummernzeichen wird kein Fragezeichen ausgegeben. Geben Sie nun ein:

```
123 <RETURN>
456789 <RETURN>
und so weiter
```

Es erscheint, wie erwartet, keine Fehlermeldung. Nun versuchen Sie es mal mit »1A3«. Der Computer meldet einen »FILE DATA«-Error. Die erste Zahl »1« kann der numerischen Variablen »A« zugewiesen werden. Das zweite Zeichen jedoch ist keine Zahl, sondern ein alphanumerisches Zeichen beziehungsweise ein Buchstabe und läßt sich nicht einer numerischen Variablen zuordnen.

Der angezeigte Fehler kann sehr schnell gefunden werden, da die Zeile, in der der Fehler auftrat, mit der Fehlermeldung zusammen ausgegeben wird.

?FILE NOT FOUND:

Es wurde versucht, ein Programm- oder Daten-File von der Diskette zu laden, dessen Name nicht auf der momentan eingelegten Diskette existiert. Um diese Fehlermeldung kennenzulernen, tippen Sie einfach ein »LOAD"ALR",8«. Natürlich muß dazu ein Diskettenlaufwerk vorhanden, eingeschaltet und mit dem Computer verbunden sein. Erscheint eine solche Fehlermeldung innerhalb eines Programms, wird zusammen mit der Fehlermeldung die Zeile ausgegeben, in der der Fehler auftrat. Sollte in der fehlerhaften Zeile statt eines Namens hinter dem LOAD-Befehl nur eine alphanumerische Variable stehen, zum Beispiel »...:LOAD A\$,8:...«, dann läßt sich dieser Name mit »PRINT A\$« auf den Bildschirm bringen. Wenn sich aus irgendeinem Grund der Name nicht mehr feststellen läßt, kann er mit dem Befehl »SYS63123« noch einmal angezeigt werden. Übrigens: Die Meldung »FILE NOT FOUND« erscheint nur, wenn mit dem Diskettenlaufwerk gearbeitet wird. Bei der Datasette wird entweder nichts oder »DEVICE NOT PRESENT« ausgegeben.

?FILE NOT OPEN:

Es wurde versucht, mit den Befehlen »PRINT #, GET #, INPUT #« Zeichen an ein externes Gerät zu übermitteln oder von einem solchen zu lesen, ohne zuvor das File mit dem OPEN-Befehl geöffnet zu haben. Dabei ist es durchaus denkbar, daß eine falsche Filenummer (erste Zahl hinter dem OPEN-Befehl) hinter einem der oben stehenden Befehle oder hinter dem OPEN-Befehl eingesetzt wurde. Beispiel:

```
10 OPEN 5,4: REM DRUCKERKANAL ÖFFNEN
20 PRINT#4,"ABC":REM UND DIE ZEICHEN »ABC« DRUCKEN
30 CLOSE 4: REM DRUCKERKANAL SCHLIESSEN
```

Wird dieses Programm eingegeben und gestartet, so erhält man einen »FILE NOT OPEN«-Error, denn die Zeile öffnet einen Kanal mit der logischen Filenummer »5«. Die Zeilen 20 und 30 beziehen sich aber auf die logische Filenummer »4«.

Bei längeren ineinander verschachtelten Programmschleifen ist es häufig sehr schwierig, einen solchen Fehler zu beseitigen. Daher geht man einen anderen Weg. Man öffnet direkt vor dem PRINT #-Befehl in der gleichen Zeile einen Kanal mit der entsprechenden Filenummer (im Beispiel wird die Zeile 20 so ergänzt: 20 OPEN4,4:PR...). Jetzt muß der Kanal natürlich innerhalb des Programms wieder geschlossen werden. Wenn Sie unsicher sind, daß das tatsächlich gemacht wird, ist unmittelbar hinter dem PRINT #-Befehl der Kanal zu schließen (im Beispiel muß wieder die Zeile 20 ergänzt werden: 20 ...: "ABC":CLOSE 4). Tritt der gleiche Fehler nach dem Start des Programms in einer anderen Zeile auf, in der ebenfalls die gleiche Filenummer benutzt wurde, so muß der zuletzt eingefügte CLOSE-Befehl gelöscht und hinter dem Befehl eingesetzt werden, der den neuen Fehler verursacht hat. So können Sie sich durch das gesamte Programm »hangeln«, bis es fehlerfrei arbeitet.

?FILE OPEN:

Diese Fehlermeldung ist genau das Gegenstück zu der Fehlermeldung »FILE NOT OPEN«. Erscheint der »FILE OPEN«-Error auf dem Bildschirm, wurde ein Kanal mit der gleichen Filenummer zweimal geöffnet. Das heißt es fehlt ein CLOSE-Befehl, der den zuerst geöffneten Kanal wieder schließt. Beispiel:

```
10 OPEN 4,4
20 OPEN 4,4
```

Der einfachste Weg, diesen Fehler zu beseitigen, ist es, vor dem OPEN-Befehl, in der Zeile, die hinter der Fehlermeldung angegeben ist, den entsprechenden Kanal mit dem CLOSE-Befehl zu schließen (im Beispiel: 20 CLOSE 4:OPEN 4,4). Sie werden sich jetzt zu Recht fragen, warum wird nicht einfach die Zeile 20 gelöscht? Die Antwort darauf ist ganz einfach. Hinter dem ersten OPEN-Befehl mit der Filenummer 4 könnte ja auch eine andere Geräteadresse stehen, durch die ein anderes Gerät angesprochen wird (zum Beispiel OPEN 4, 8 für Diskettenlaufwerk). Würde jetzt der zweite OPEN-Befehl (im Beispiel Zeile 20) gelöscht werden, so würde alles, was zum Drucker (Geräteadresse 4) gesendet werden soll, zum Diskettenlaufwerk geschickt. Daher seien Sie äußerst vorsichtig mit dem Löschen von Befehlen innerhalb eines Programms.

?FORMULA TOO COMPLEX:

Es ist fast unmöglich, einen solchen Fehler in ein Basic-Programm einzubauen. Trotzdem existiert die Fehlermeldung. Durch sie meldet der Computer, daß in einer Zeichenkettenaddition zu viele Klammerausdrücke stehen. Beispiel: PRINT "A"+"("B"+"B1"+"("C"+"D")))

Die Anzahl der Zeichen innerhalb der einzelnen Zeichenketten ist völlig uninteressant. Wichtig ist, daß zweimal die Zeichenkombination »+(« vorkommt. Denn der Computer versucht sich alles vor den jeweiligen Klammerausdrücken zu merken. Im Beispiel merkt sich der C64 das Zeichen »A«, berechnet »"B"+"B1"="BB1"« und versucht sich auch diese Zeichen zu merken, um im nächsten Schritt »"C"+"D"« auszurechnen. Soweit kommt er aber nicht, weil er sich maximal eine Zeichenkette merken kann (im Beispiel »A«) und gibt daher die Fehlermeldung »FORMULA TOO COMPLEX« aus.

Anmerkung: Verschachtelungen innerhalb einer Zeichenkettenaddition sind völlig überflüssig. Sie erhalten nur dann einen Sinn, wenn nicht nur addiert, sondern auch multipliziert oder dividiert wird und genau dies ist mit Zeichenketten nicht möglich.

Die oben stehende Zeile könnte daher auch so aussehen: PRINT "A"+"B"+"B1"+"C"+"D"

oder einfach

```
PRINT "A"; "B"; "B1"; "C"; "D"
```

oder noch einfacher

```
PRINT "A" "B" "B1" "C" "D"
```

Das Ergebnis ist immer das Gleiche. Deshalb ist bei Zeichenkettenoperationen das Pluszeichen und vor allen Dingen das Arbeiten mit Klammern zu vermeiden.

Sollte trotzdem mal ein »FORMULA TOO COMPLEX«-Error erscheinen, ist ein fataler Fehler gemacht worden, der sich nur beseitigen läßt, indem man den Computer kurzzeitig ausschaltet.

?ILLEGAL DEVICE NUMBER:

Diese Fehlermeldung bedeutet, daß ein Befehl zu einem unzulässigen Gerät geschickt wurde. Mit anderen Worten: Die im Befehl angegebene Geräteadresse ist falsch. Zum Beispiel erzeugt »LOAD "TEST",0« einen »ILLEGAL DEVICE NUMBER«-Error, denn es ist unmöglich, ein Programm von dem Gerät mit der Geräteadresse »0« (gleich Tastatur) zu laden.

Erscheint diese Fehlermeldung auf dem Bildschirm, ist die Beseitigung des Fehlers recht einfach. Lassen Sie sich die Zeile, die hinter der Fehlermeldung angegeben ist, LISTen und schauen sich alle LOAD- und SAVE-Befehle an. Ist eine der Geräteadressen hinter diesem Befehl »0, 2 oder 3«, dann haben Sie den Fehler gefunden. Sollte nicht bekannt sein, welche Gerätenummer welchem Gerät zugeordnet ist, dann schauen Sie in der Gerätenummern-Tabelle nach, die bei der Beschreibung der Fehlermeldung »DEVICE NOT PRESENT« aufgeführt ist.

ILLEGAL DIREKT:

Wenn versucht wird, den Basic-Befehl »GET« oder »INPUT« im Direktmodus einzugeben, dann meldet sich der Computer mit dieser Fehlermeldung. Der Direktmodus unterscheidet sich vom Programmodus dadurch, daß im Direktmodus jeder eingegebene Befehl ausgeführt wird, sobald man die RETURN-Taste drückt. Den Programmodus erkennt der C64 an einer Zahl (Zeilennummer) am Anfang einer Zeile. Wird hinter einer Zeile, die mit einer Zeilennummer beginnt, die RETURN-Taste gedrückt, dann wird die Zeile nicht ausgeführt, sondern in den Speicher des C64 geschrieben. Beispiel:

```
INPUT A$ <RETURN>
```

führt zur Fehlermeldung »ILLEGAL DIREKT«

Wird vor den INPUT-Befehl eine Zahl geschrieben »10 INPUT A\$ <RETURN>«, dann akzeptiert der Computer diese Zeile und speichert sie. Wenn der INPUT-Befehl jetzt ausgeführt werden soll, ist RUN <RETURN> im Direktmodus einzugeben.

Da der »ILLEGAL DIREKT«-Error niemals in einem Programm auftauchen kann, erübrigt sich die Behebung dieses Fehlers.

ILLEGAL QUANTITY:

Diese Fehlermeldung zeigt an, daß die Berechnung einer Funktion eine Zahl ergab, die der Computer in dieser Form nicht verarbeiten kann. Arbeitet man mit Integer-Variablen (zum Beispiel A%), ist darauf zu achten, daß den Variablen niemals Zahlen zugewiesen werden, deren Wert 32000 überschreitet. Wird versucht, mit dem Basic-Befehl POKE eine Zahl zu speichern, deren Wert größer ist als 255 oder negativ, dann erscheint auch ein »ILLEGAL QUANTITY«-Error. Die gleiche Fehlermeldung gibt der C64 auch dann aus, wenn beim OPEN-Befehl eine logische Filenummer (erste Zahl hinter dem OPEN-Befehl) größer als 255 gewählt wurde, oder wenn aus einer negativen Zahl die Quadratwurzel gezogen werden soll. Alle folgenden Beispiele erzeugen einen »ILLEGAL QUANTITY«-Error:

```
A%=33000:REM DIE INTEGERVARIABLE A% IST GRÖßER 32000
```

```
POKE 2,300:REM ES SOLL EINE ZAHL GESPEICHERT
```

WERDEN, DIE GRÖßER IST ALS 255

```
POKE 2,-3:REM NEGATIVE WERTE LASSEN SICH NICHT SPEICHERN
```

```
OPEN 300,4:REM FILENUMMER GRÖßER 255
```

```
PRINT SQR(-1):REM AUS NEGATIVEN ZAHLEN KANN NICHT DIE QUADRATWURZEL GEZOGEN WERDEN
```

Einen solchen Fehler im Programm aufzuspüren ist nicht einfach, denn in den meisten Fällen enthalten Variable die fehlerträchtigen Zahlen. Um einen solchen Fehler zu beseitigen, lassen Sie sich zunächst die Zeile LISTen, die den Fehler enthält. Im Anschluß daran ist jede in dieser Zeile vorkommende Variable mit dem Befehl PRINT auf den Bildschirm zu schreiben. Wenn bei diesem Versuch die Fehlermeldung angezeigt wird, ist zumindest die fehlerhafte Variable gefunden. Jetzt muß gegebenenfalls Schritt für Schritt der Programmablauf verfolgt werden, um die Stelle zu finden, an der der Variablen die falsche Zahl zugewiesen wurde.

I/O ERROR 1 bis 29:

Diese Fehlermeldung existiert nach dem Einschalten des Computers nicht. Nur dann, wenn in die Speicherzelle 157 die Zahl 64 oder 192 geschrieben wird (zum Beispiel POKE 157,64), gibt der Computer im Falle eines Fehlers die Fehlermeldung »I/O ERROR« aus. Dabei entspricht eine Zahl zwischen 1 und 29 hinter der Fehlermeldung den normal angezeigten Fehlern wie zum Beispiel »ILLEGAL QUANTITY«. Beispiel:

```
POKE157,192:LOAD "ABC",9
```

Existiert kein Diskettenlaufwerk mit der Gerätenummer 9, so führt das Beispiel zu der Fehlermeldung:

```
SEARCHING FOR ABC
```

```
I/O ERROR #5
```

```
?DEVICE NOT PRESENT
```

?LOAD ERROR:

Ein Programm oder Daten-File konnte nicht fehlerfrei von der Datensette oder von dem Diskettenlaufwerk geladen werden. Meldet der Computer diesen Fehler, wenn ein Programm abgearbeitet wird, so liegt der Fehler nicht am Programm, sondern an dem Gerät, von dem es geladen wurde. Versuchen Sie daher das gleiche Programm noch einmal zu laden. Scheitert auch dieser Versuch, so ist das Programm verloren. Meistens kommt ein »LOAD ERROR« dann vor, wenn entweder die Diskette oder das Band der Kassette mechanisch beschädigt ist.

?MISSING FILE NAME:

Weist darauf hin, daß beim LOAD- oder SAVE-Befehl kein Name angegeben worden ist, denn die Angabe eines Namens ist bei Operationen mit dem Diskettenlaufwerk unbedingt erforderlich. »LOAD "",8 <RETURN>« führt folglich zur Fehlermeldung »MISSING FILE NAME«.

Tritt der Fehler innerhalb eines Programms auf, lassen Sie sich die Zeile, die hinter der Fehlermeldung angegeben ist, LISTen. Folgt dem entsprechenden LOAD- oder SAVE-Befehl ein Leerzeichen " " ", dann ist zwischen den Gänsefüßen ein beliebiger Name einzusetzen. Schwieriger wird es, wenn der Befehl so aussieht:

```
...:load a$,8:...
```

Schauen Sie sich in einem solchen Fall den Inhalt der Stringvariablen (Zeichenkettenvariable) mit dem Befehl

```
PRINT A$ <RETURN>
```

an. Sie wird kein einziges Zeichen enthalten. Folglich muß unmittelbar vor dem entsprechenden Befehl der Stringvariablen A\$ ein Name zugewiesen werden. Die Zeile (Beispiel) müßte dann so aussehen:

```
...:a$="zeichen":load a$,8:...
```

Häufig ist aber erwünscht, daß sich mit der gleichen Zeile unterschiedliche Programme laden lassen. In einem solchen Fall muß, bevor der LOAD- oder SAVE-Befehl ausgeführt wird, abgefragt werden, ob die Stringvariable A\$ eine Zeichenkette enthält. Dazu muß das Programm um eine Zeile ergänzt



werden, die die erforderliche »IF«-Abfrage enthält. Enthält A\$ nichts, also nur einen Leerstring, so ist der Stringvariablen eine beliebige Zeichenkette zuzuweisen.

```
19 IF A$ = "" THEN A$ = "ZEICHEN"
20 ...:LOAD A$,8
```

?NEXT WITHOUT FOR:

Erscheint, wenn zur Anweisung NEXT die entsprechende FOR-Anweisung fehlt. Dies kann verschiedene Gründe haben:

1) Im Programm befinden sich mehr NEXT- als FOR-Anweisungen.

2) Es wird versehentlich mit dem Basic-Befehl GOTO in eine FOR..NEXT-Schleife gesprungen.

3) Der Name der Schleifenvariablen (Variable hinter dem FOR-Befehl) in der FOR-Anweisung stimmt nicht mit derjenigen hinter der NEXT-Anweisung überein.

4) Es wurde innerhalb einer Schleifenkonstruktion mit dem Basic-Befehl GOSUB ein Unterprogramm aufgerufen, das eine NEXT- ohne entsprechende FOR-Anweisung enthält.

Fehler dieser Art lassen sich nur sehr schwer beheben. Vor allen Dingen dann, wenn das Programm lang und unübersichtlich geschrieben wurde. Der gesamte Programmablauf muß zu »Fuß« im Listing nachvollzogen werden. Zu jeder FOR-Anweisung ist die entsprechende NEXT-Anweisung zu suchen. Es muß überprüft werden, ob von irgendeiner Stelle in die Schleife gesprungen wird, in der der Fehler ausgegeben wurde.

Daher ist es sinnvoll, sich zu jedem Programm einen Ablaufplan (Flußdiagramm) anzufertigen, damit der Überblick nicht verloren geht und Fehler, die nur schwer zu beheben sind, vermieden werden.

?NOT INPUT FILE:

Erscheint, wenn versucht wird, mit den Befehlen INPUT # und GET # aus einer Datei etwas zu lesen, die zum Schreiben geöffnet wurde oder wenn als Filenummer (erste Zahl hinter dem OPEN-Befehl) »0« eingesetzt wird. Versuchen Sie mal im Direktmodus

```
OPEN0,3 <RETURN>
```

einzugeben. Der Computer meldet sofort einen »NOT INPUT FILE«-Error. Wenn eine Datasette zur Verfügung steht, erzeugt das folgende Beispiel die gleiche Fehlermeldung:

```
10 OPEN1,1,1,"TEST":REM BANDDATEI ZUM
SCHREIBEN ÖFFNEN
20 PRINT#1,"WORT":REM DAS WORT »WORT« IN DIE
DATEI SCHREIBEN
30 INPUT#1,A$:REM ERZEUGT DIE FEHLERMELDUNG
40 CLOSE1:REM KANAL SCHLIESSEN
```

Erscheint ein »NOT INPUT FILE«-Error, ist die hinter der Fehlermeldung angegebene Zeile zu LISTen. Ist dort die logische Filenummer hinter dem OPEN-Befehl »0«, so ist sie durch eine andere Zahl zu ersetzen. Vorsicht! Alle Ein-/Ausgabe-Befehle, die sich auf diesen OPEN-Befehl beziehen, müssen ebenfalls entsprechend geändert werden.

Steht in der geLISTeten Zeile kein OPEN-, sondern ein INPUT #- oder GET #-Befehl, dann ist versucht worden, aus einer Datei etwas zu lesen, die nur zum Schreiben geöffnet wurde (siehe Beispiel). Sollte dieser Fall eingetreten sein, ist zuerst zu überprüfen, ob der INPUT #- oder GET #-Befehl an der entsprechenden Stelle richtig ist. Wenn der Befehl stimmt, ist eine falsche Filenummer gewählt worden oder die fehlerhafte Zeile wurde mit dem GOTO- beziehungsweise GOSUB-Befehl angesprungen, ohne zuvor die geöffnete Datei zum Schreiben zu schließen.

?NOT OUTPUT FILE:

Es wurde versucht, mit dem Basic-Befehl PRINT # etwas in eine Datei zu schreiben, die nur zum Lesen geöffnet wurde. Auch dieser Fehler bezieht sich nur auf das Arbeiten mit Datasette. Beispiel:

```
10 OPEN 1,1,0,"TEST":REM KASSETTENDATEI ZUM LESEN
ÖFFNEN. ES MUSS EINE DATEI MIT DEM NAMEN TEST
EXISTIEREN.
20 A$="ABC":PRINT#1,A$:REM DIESE ZEILE ERZEUGT
DEN FEHLER
30 CLOSE1:REM KANAL SCHLIESSEN
```

Die gleiche Fehlermeldung erscheint auch dann, wenn man die Tastatur (Geräteadresse 0) mit dem Befehl »OPEN 1,0« zum Lesen öffnet und versucht, mit dem Befehl »PRINT # 1, "ABC"« etwas zum Kanal mit der Filenummer »1« zu schicken. Das heißt, die Tastatur läßt sich nicht als Ausgabegerät verwenden.

Tritt der Fehler in einem Programm auf, ist er genauso zu beseitigen wie der »NOT INPUT FILE«-Error.

?OUT OF DATA:

Die Fehlermeldung erscheint, wenn das Programm auf den Basic-Befehl READ stößt und entweder keine DATAs vorhanden sind oder schon alle gelesen wurden. Beispiel:

```
READ A$ <RETURN>
```

Wird der Befehl im Direktmodus eingegeben, erscheint der »OUT OF DATA«-Error, weil keine DATAs vorhanden sind.

Meldet der Computer diesen Fehler, sind entweder DATAs vergessen worden einzugeben oder die Schleifenvariable innerhalb einer FOR..NEXT-Schleife zählt zu weit. Beispiel:

```
10 FOR I=0 TO 10
20 READ A
30 NEXT I
40 END
50 DATA 1,2,3,4,5
```

Die Schleifenvariable »I« zählt von 0 bis 10, erwartet also 11 DATAs. Es sind aber nur fünf DATAs vorhanden. Folglich meldet der Computer einen »OUT OF DATA«-Error.

Wie der Fehler beseitigt wird, dürfte damit wohl auch geklärt sein. Hinweis: Der Fehler tritt zwar in der Zeile auf, deren Zeilennummer hinter der Fehlermeldung ausgegeben wird, befindet sich aber niemals in dieser Zeile, sondern immer in der Zeile, die die entsprechenden DATAs enthält.

?OUT OF MEMORY:

Dieser Fehler kann verschiedene Ursachen haben:

1) Der Speicherplatz im Computer reicht nicht für das Programm und die Variablen. Häufig tritt der Fehler auf, wenn DIM-Anweisungen zu groß gewählt wurden (zum Beispiel »DIM A(10000)«).

2) Wenn Maschinenprogramme geladen wurden, deren Startadresse über dem freien Basic-Bereich liegen (zum Beispiel SMON). In einem solchen Fall schafft der Befehl NEW Abhilfe, nachdem das Programm geladen wurde.

3) Verschachtelungstiefen bei FOR...NEXT-Schleifen

(maximal 10) oder Unterprogrammaufrufen, mit dem Befehl GOSUB (maximal 24) wurden überschritten. Die häufigste Fehlerursache entsteht dadurch, daß in einem Unterprogramm der Befehl RETURN fehlt.

Wurde die Verschachtelungstiefe überschritten, muß das Programm anders aufgebaut werden. FOR...NEXT-Schleifen lassen sich zum Beispiel durch IF-Abfragen ersetzen.

Beispiel

```
10 FOR I=0 TO 10
20 A=A+1
30 NEXT I
ist identisch mit
10 A=A+1
20 I=I+1
30 IF I <> 10 THEN 10
```

?OVERFLOW:

Entweder liegt das Ergebnis oder ein Zwischenergebnis eines Ausdrucks außerhalb des zulässigen Zahlenbereichs. Jedes Ergebnis muß zwischen »+/-1.37E38« liegen. Beispiel:

```
PRINT 9*2E37/100 <RETURN>
```

Wird dieses Beispiel eingegeben, erscheint der »OVERFLOW«-Error. Formt man das Beispiel um, so läßt sich dieser Ausdruck durchaus berechnen:

```
PRINT 9/100*2E37
```

Der Fehler ist dadurch entstanden, daß das Zwischenergebnis »9*2E37« den zulässigen Zahlenbereich überschritten hat. Im Falle eines »OVERFLOW«-Errors ist also ein Ausdruck so abzuändern, daß kein Zwischenergebnis den zulässigen Wert überschreitet.

?REDIM'D ARRAY:

Das Programm trifft zweimal auf die selbe DIM-Anweisung. Eine Feldvariable (zum Beispiel A(15)) darf in einem Programm nur ein einziges Mal dimensioniert werden.

Beispiel:

```
10 DIM A(20)
20 DIM A(30)
```

Es tritt ein »?REDIM'D ARRAY ERROR IN 20« auf. Dieser Fehler läßt sich meistens dadurch vermeiden, daß alle benötigten DIM-Anweisungen in die erste Programmzeile verlegt werden. Weiterhin sollte darauf geachtet werden, daß auf alle möglichen DIM-Zeilen kein GOTO-Sprung erfolgt. Wenn sich dies nicht vermeiden läßt, kann man ja ein CLR: vor den DIM-Befehl setzen. Dann werden allerdings auch alle anderen Variablen gelöscht.

Beispiel:

```
10 CLR : DIM A(20)
20 GOTO 10
```

Diesmal tritt kein Fehler auf.

?REDO FROM START:

Eine INPUT-Anweisung erwartet die Eingabe einer Zahl; es wurden jedoch Buchstaben oder Zeichen eingegeben. Die ganze INPUT-Anweisung wird noch einmal bearbeitet.

Beispiel:

```
10 INPUT A
20 PRINT A
```

Starten Sie das Programm mit »RUN« und geben Sie zum Beispiel ein »X« ein (plus »RETURN«-Taste). Der Computer wird dies mit der beschriebenen Fehlermeldung quittieren und die INPUT-Anweisung noch einmal ausführen. Es dürfen nur Zahlen eingegeben werden! Ausnahme ist der Buchstabe »E« zur Kennzeichnung von Exponential-Zahlen (zum Beispiel 1.5E+3, also 1,5 mal 10 hoch 3)

Tip: Überprüfen Sie Ihre Eingabe noch einmal sorgfältig auf unerlaubte Zeichen.

?RETURN WITHOUT GOSUB:

Der Computer trifft auf ein »RETURN«, ohne daß zuvor ein »GOSUB« aufgetreten ist.

Beispiel:

```
10 RETURN
```

Wenn Sie das Programm starten, tritt sofort der genannte Fehler auf. Aber: Es kann auch sein, daß zwar *eine* GOSUB-Anweisung auftritt, aber der Computer *einen zweiten* RETURN-Befehl findet. Häufigste Ursache: Ein Programm mit einem oder mehreren Unterprogrammen hat nach Beendigung des Hauptteils keine END-Anweisung und »läuft« deshalb unbeabsichtigt in die erste Unterroutine. An deren Ende trifft der C64 dann auf ein RETURN, ohne daß jedoch ein GOSUB gegeben wurde.

Beispiel:

```
10 REM HAUPTPROGRAMM
20 GOSUB 100
30 REM HAUPTPROGRAMMENDE
100 REM UNTERPROGRAMM
110 RETURN
```

Haben Sie den Fehler entdeckt? Zuerst wird in Zeile 20 die Unterroutine ab Zeile 100 korrekt mit GOSUB angesprungen und mit RETURN beendet. Dann jedoch bearbeitet der C64 Zeile 30, Zeile 100 und dann Zeile 110! Abhilfe: Am Ende des Hauptteils eine END-Anweisung. In obigem Beispiel:

```
40 END
```

?STRING TOO LONG ERROR:

a) Eine Stringvariable enthält mehr als die erlaubten 255 Zeichen, oder

b) Es wurde mittels INPUT # eine Zeichenkette eingelesen, die mehr als 89 (!) Zeichen enthält.

Beispiele:

```
a)
10 A$="X"
20 FOR I=1 TO 500
30 A$=A$+"X"
40 PRINT LEN(A$)
50 NEXT I
```

In Zeile 30 wird der String A\$ immer um ein Zeichen erweitert. Zeile 40 informiert über die aktuelle Länge des Strings. Sobald er mehr als 255 Zeichen enthält, wird ein »?STRING TOO LONG ERROR IN 30« ausgegeben.

```
b)
10 OPEN 1,8,1,"TEST,S,W"
20 FOR I=1 TO 100
30 PRINT #1,"X";
40 NEXT I
50 CLOSE 1
60 OPEN 1,8,0,"TEST,S,R"
70 INPUT #1,A$
80 PRINT A$
90 CLOSE 1
```

Der erste Programmteil (Zeilen 10 bis 50) eröffnet eine Datei auf Diskette und schreibt 100 Zeichen direkt hintereinander in diese Datei. Direkt hintereinander deshalb, weil in Zeile 30 nach dem PRINT #-Befehl ein »;« steht.

Der zweite Programmteil (Zeilen 60 bis 80) versucht nun. Diese 100 Zeichen auf einmal in die Variable A\$ einzulesen (was natürlich nicht funktioniert).

Um diesen Fehler besser zu verstehen, muß man sich die Funktionsweise des INPUT #-Befehls einmal vor Augen führen: INPUT # liest solange Zeichen in die angegebene Variable, bis in der Datei das erste »carriage return« (CHR\$(13)) folgt. Dieses Zeichen, auch kurz »cr« genannt, wird beim Schreiben von Daten in eine Datei immer genau dann gesendet, wenn der PRINT #-Befehl nicht mit einem »;« oder »;« endet. In unserem Beispielprogramm wird in Zeile 30 durch das »;« nach dem PRINT #-Befehl das Senden des »cr« verhindert. Deshalb meldet der Computer beim INPUT # in Zeile 70 einen STRING TOO LONG ERROR, sobald er merkt, daß nach dem 89. Zeichen immer noch kein »cr« in der Datei aufgetaucht ist. Die Daten werden nämlich nicht sofort in die

Variable A\$ eingelesen, sondern erst in einen Zwischenpuffer; und dieser kann nicht mehr als 89 Zeichen aufnehmen. Um unser kleines Programm doch noch lauffähig zu machen, müßte man den INPUT #-Befehl durch eine FOR-NEXT-Schleife mit einer GET #-Anweisung ersetzen (diese liest immer nur ein einziges Zeichen). Also:

```
70 FOR I=1 TO 100 : GET #1,B$ : A$=A$+B$ : NEXT I
```

Wenn in einem Programm ein STRING TOO LONG Error in einer Zeile mit einer INPUT #-Anweisung auftritt, ist der Fehler also nicht in dieser Zeile, sondern in der Routine zu suchen, die die Daten in die Datei geschrieben hat. Diese muß vor dem Senden eines Strings über die LEN-Funktion erst einmal testen, ob er die vorgeschriebene Länge von 89 Zeichen nicht überschreitet.

?SYNTAX ERROR:

Kann mehrere Gründe haben:

a) Generell ist der Syntax eines Befehls, also seine Schreibweise, von Ihnen falsch eingegeben worden.

b) In einer DATA-Zeile steht ein Zeichen oder eine Zeichenfolge, obwohl der READ-Befehl eine Zahl erwartet.

c) Das erste Zeichen des Basic-Speichers ist nicht Null.

Beispiele:

a) Allgemein tritt der Syntax-Error auf, wenn der Basic-Interpreter einen Befehl nicht versteht, also wenn Sie sich beim Schreiben eines Programms vertippt haben. Überprüfen Sie daher die Zeile, in der der Fehler auftrat, noch einmal sorgfältig, ob Sie die Befehle alle richtig geschrieben oder irgendein Zeichen vergessen haben! Häufigster Tippfehler aller Programmierer: »RUM« statt »RUN«.

Achtung! Dieser Fehler tritt auch auf, wenn in einer Variablenzuweisung die Anzahl der geöffneten Klammern ungleich Null ist. Beispiel: A=(2+(3*4). Es fehlt eine »Klammer zu«. Oder: A=2+(3*4)). Eine Klammer wurde zu viel geschlossen.

b)

```
10 READ A,B,C
20 DATA 67, Z, 20
```

Es tritt ein Syntax-Error in Zeile 20 (!) auf, da die READ-Anweisung in Zeile 10 drei Zahlen als DATA-Werte erwartet. Die »zweite Zahl« ist jedoch ein Buchstabe.

Tip: Wenn Sie in DATA-Zeilen Zahlen und Buchstaben zusammen verwenden möchten und es tritt dieser Fehler auf, dann überprüfen Sie, ob Sie die Werte auch in der richtigen Reihenfolge auslesen.

c) Besonders tückisch: Wenn aus irgendeinem Grund der Inhalt der ersten Basic-Adresse nicht Null ist, so tritt bei der Eingabe von »RUN« oder »NEW« ein Syntax-Error auf. Dies ist meistens der Fall, wenn Sie den Basic-Start (Adressen 43 und 44) verändert haben, zum Beispiel, um eine Basic-Erweiterung zu verwenden.

Abhilfe: POKE(PEEK(43)+PEEK(44)*256-1),0

?TOO MANY FILES:

Es wurden mehr als die maximal erlaubten zehn Dateien (über einen OPEN-Befehl) eröffnet.

Beispiel:

```
10 OPEN 1,3
20 OPEN 2,3
30 OPEN 3,3
40 OPEN 4,3
50 OPEN 5,3
60 OPEN 6,3
70 OPEN 7,3
80 OPEN 8,3
90 OPEN 9,3
100 OPEN 10,3
110 OPEN 11,3
```

Sicherlich, dies ist ein extremes Beispiel, das im »Alltagsbetrieb« nie auftreten wird, aber ebenso ist es mit dieser Fehlermeldung. Sollte also bei Ihnen dieser Fehler einmal auftreten,

so sollten Sie sich ernsthaft Gedanken darüber machen, ob Sie wirklich so viele offene Datenkanäle benötigen.

?TYPE MISMATCH:

Tritt auf, wenn versucht wird, eine Zeichenkette in einer numerischen Variablen abzulegen.

Beispiele:

```
A="TEST" oder A=A$
```

Untersuchen Sie die Zeile, in der der Fehler aufgetreten ist. Meistens wurde nur das \$-Zeichen vergessen.

?UNDEF'D FUNCTION:

Eine mathematische Funktion wurde angesprochen, ohne vorher definiert worden zu sein.

Beispiel:

```
10 A=FN X(2)
```

Wenn Sie trotzdem der Meinung sind, die entsprechende Funktion definiert zu haben, so überprüfen Sie den verwendeten Funktionsnamen noch einmal. Zur Veranschaulichung hier noch einmal der Syntax der FN-Anweisung:

Definieren: DEF FN name(variable)=funktion

Aufruf: variable=FN name(wert oder variable)

Ein korrektes Beispiel:

```
10 DEF FN TEST(X)=X*2
20 A=FN TEST(2)
30 PRINT A
```

?UNDEF'D STATEMENT:

a) Über RUN, GOTO oder GOSUB wurde eine Zeile angesprochen, die nicht existiert.

b) Es wurde versucht, eine Variable als Sprungziel zu verwenden.

Beispiel:

```
a)
10 GOTO 100
20 END
```

Dieser Fehler ist äußerst einfach zu beheben:

Zeile, in der der Fehler auftrat, listen, und überprüfen, ob die angesprochene Zeile wirklich vorhanden ist. Dann die GOTO- oder GOSUB-Anweisung entsprechend ändern.

b)

```
10 A=100 : GOTO A
100 END
```

Obwohl die Zeile 100 existiert, tritt der Fehler auf. Es darf nämlich nicht über eine Variable zu einer bestimmten Zeile gesprungen werden. Es gibt aber die Möglichkeit, über einen ON-GOTO-Befehl einen berechneten Sprung auszuführen:

```
10 INPUT "SPRUNG NACH ZEILE?";A
20 A=A/100 : ON A GOTO 100,200,300
100 PRINT "ZEILE 100" : END
200 PRINT "ZEILE 200" : END
300 PRINT "ZEILE 300" : END
```

Beachten Sie aber, daß der ON-GOTO-Befehl wie folgt arbeitet:

Wenn A=1, dann GOTO 100, wenn A=2, dann GOTO 200, ... Deshalb muß die eingegebene Zahl auch erst in dieses Format umgerechnet werden (A=A/100).

?VERIFY ERROR:

Durch ein VERIFY-Kommando wurde festgestellt, daß das Programm im Speicher mit dem auf Datasette beziehungsweise Diskette nicht übereinstimmt. Deutet bei Datasettenbetrieb meistens auf ein fehlerhaftes Band hin. Verwenden Sie dann eine neue Kassette.

Ist bei Diskettenbetrieb äußerst unwahrscheinlich, da die Floppy beim Speichern eines Programms automatisch einen Verify durchführt. Eventuell schadhafte Disketten werden daher schon frühzeitig erkannt. Generell kann man sagen, daß beim Arbeiten mit einer Floppy das »Verifizieren« entfallen kann. Nützlich dagegen ist es bei der Programmentwicklung, wenn man nicht sicher ist, ob man die aktuelle Version schon gespeichert hat. Hier bringt ein Verify die Antwort.

(ah/tr)