Verbotene Variablen

Hatten Sie schon Probleme mit dem SYNTAX ERROR in anscheinend korrekten Basic-Zeilen? Wahrscheinlich lag es an verbotenen Variablen, die hier näher erklärt werden. Außerdem zeigen wir Ihnen, wie Sie Basic-Programme strukturieren können.

Wer jemals versucht hat, fremde Basic-Programme zu verstehen und dem Gedankengang des Autors zu folgen, der hat sich bestimmt des öfteren darüber geärgert, daß Basic leider allzuoft in einem Spaghetticode ausartet. Sogar der Autor eines Programms findet sich, wenn er nicht 'zig REM-Zeilen eingearbeitet hat, nach einigen Monaten nicht mehr mit dem eigenen Programm zurecht. Das sind dann die Momente, wo man in Richtung Pascal oder all den anderen strukturierten Programmiersprachen schielt.

Der vorliegende Artikel wendet sich an alle, die nach einer Möglichkeit suchen, mit dem vorhandenen (zugegebenermaßen mageren) Basic des C 64 trotzdem ihre Listings etwas übersichtlicher zu gestalten und ihnen eine Struktur zu verleihen.

Übersichtliche Listings

Einige Basic-Programmierer benutzen zur Strukturierung ihrer Programme entweder REMs oder einfach nur den Doppelpunkt. Damit können einzelne Programmteile beim Listing optisch voneinander getrennt werden. Eine bessere Lesbarkeit wird erzielt. Eine andere Möglichkeit zur Strukturierung bieten die Grafikzeichen des C 64.

Nun, werden Sie sich fragen, was denn so besonderes daran sei, Grafikzeichen in Programmen zu verwenden. Man sieht sie doch alle Tage in Listings. Der Witz dieser Lösung zur optischen Aufbereitung von Listings ist, daß die verwendeten Grafikzeichen unsichtbar bleiben, aber die Lesbarkeit der Programme sichtbar unterstützen.

Um diesen scheinbaren Widerspruch aufzulösen, ist es notwendig, sich ein wenig mit der Art und Weise zu beschäftigen, wie der Basic-Interpreter Programmzeilen im Basic-Speicher ab $0801 (dezimal 2049) ablegt.

Schauen Sie sich dazu einmal das kleine Beispielprogramm an.

```
10  FOR I=0 TO 20
20  PRINT"TEST"
30  NEXT
```

Mit einem Monitor erhalten Sie den folgenden Speicherauszug:

```
0800  00  10  08  0A  00  81  20  49
0808  B2  31  20  A4  20  32  30  00
0810  1C  08  14  00  99  22  54  45
0818  53  54  22  00  22  08  1E  00
0820  82  00  00  00
```

Da das Beispielprogramm klein ist, wird noch keine Strukturierung sinnvoll. Stellen Sie sich aber vor, es wäre ein Programm mit 50 und mehr Zeilen, mit einer Menge von GOTOs, GOSUBs und FOR-NEXT-Schleifen. In diesem Fall wird ein herkömmliches Listing von Zeile zu Zeile unübersichtlicher.

Eine einfache, aber recht wirkungsvolle Möglichkeit, ein Listing »durchsichtig« zu machen, ist das Einrücken der Programmzeilen innerhalb einer Schleife. Das Einrücken geschieht meist durch die Voreinstellung eines oder mehrerer Doppelpunkte am Anfang einer Zeile in Verbindung mit einer Reihe von Leerzeichen.

Es geht aber auch anders. Geben Sie dazu die Zeile 20 neu ein:

20 {SHIFT-J} {4*Leertaste} PRINT"TEST"

Wenn Sie nun das kleine Programm listen, ist etwas Erstaunliches passiert:

```
10  FOR I=1 TO 20
20      PRINT"TEST"
30  NEXT
```

Das bei der Eingabe sichtbare Grafikzeichen ist unsichtbar geworden, der PRINT-Befehl aber ist wie bei der Eingabe um vier Leerzeichen (plus ein erzwungenes Leerzeichen nach der Zeilennummer) nach rechts gerückt worden. Wenn Sie das gleiche durch Eingabe von vier Leerschritten unmittelbar nach der Zeilennummer versucht hätten, so wäre der Versuch fehlgeschlagen.

Der Interpreter ignoriert nämlich alle Leerzeichen, die der Programmierer nach der Zeilennummer eingibt; vielmehr wirft er diese alle weg, bis er auf die erste Basic-Aweisung oder einem Doppelpunkt in der betreffenden Zeile trifft. Andererseits besteht er auf genau einem Leerzeichen unmittelbar nach der Zeilennummer, das er eventuell selbst einfügt.

Aber was ist durch das Grafikzeichen im Speicher passiert? Einen Speicherauszug liefert das folgende Bild:

```
0800  00  10  08  0A  00  81  20  49
0808  B2  31  20  A4  20  32  30  00
0810  20  08  14  00  20  20  20  20
0818  99  22  54  45  53  54  22  00
0820  26  08  1E  00  82  00  00  00
```

Die interessantesten Speicherstellen beginnen ab $0810, wo sich gegenüber unserem vorangehenden Beispiel etwas geändert hat. Wir sehen in $0810 und $0811 den Zeiger (hier 20 08 = $0820) auf den Anfang der nächsten Basic-Zeile. Die folgenden Hex-Zeichen 14 00 ergeben in Low-/High-Byte-Darstellung die Zeilennummer 20. Dann folgen die vier eingefügten Leerzeichen ($20) und erst danach das Token 99 (interne Abkürzung des Interpreters) für den PRINT-Befehl. Die folgenden Zeichen sind dann, jeweils um vier Speicherstellen verschoben, mit den Zeichen aus dem vorangehenden Beispiel identisch.

Wenn Sie nun spaßeshalber den Cursor nochmals auf die Zeile 20 des Programmes fahren und die RETURN-Taste drücken, schaut die Zeile 20 nach erneutem Listen wieder so aus, wie ohne Benutzung des »Grafikzeichen-Tricks«. Das geschieht, weil der C 64 einen bildschirmorientierten Editor besitzt, der in letzterem Fall nicht unterscheidet, ob die Zeile 20 neu eingetippt oder »vom Bildschirm übernommen« wird.

Mit dem Eingeben eines Grafikzeichens unmittelbar nach der Zeilennummer übertölpeln Sie den Interpreter, der diese Art von Zeichen an dieser Stelle nicht erwartet hätte. Vor lauter Staunen »vergißt« er dann die eingegebenen Leerzeichen zu ignorieren, wie es sonst seine Art ist.

Verwendung von »verbotenen« Variablen

Hand aufs Herz! Ist es Ihnen nicht auch schon einmal passiert, daß Sie ein Programm geschrieben haben und der Lesbarkeit halber leicht merkbare und selbsterklärende Variablennamen verwendeten. Beim ersten Testlauf stieg der C 64 dann mit der Fehlermeldung SYNTAX ERROR aus und Sie in die Fehlersuche ein.

Meist dauert es dann ein wenig, bis man den Fehler erkennt und beispielsweise einen Variablennamen wie SCHIFF als den Übeltäter entlarvt. SCHIFF ist ein verbotener Name für eine Variable, da der Name eine Basic-Anweisung (IF) enthält. Stößt der Interpreter auf einen solchen Namen, so ist er irritiert, meldet einen Fehler und beendet einfach seine Arbeit.

Schauen Sie sich anhand eines kleinen Beispiels einmal die Arbeitsweise des Interpreters an.

```
10  SCHIFF$="TITANIC"
20  PRINT SCHIFF$
```

Wenn Sie das Programm starten, so geht das Programm ebenso unter wie das besagte Schiff. Ein Monitorlisting schafft Klarheit:

```
0800  00 16 08 0A 00 53 43 48
0808  8B 46 24 B2 22 54 49 54
0810  41 4E 49 43 22 00 23 08
0818  14 00 99 20 53 43 48 8B
0820  46 24 00 00 00
```

Genauer die Speicherzelle $0808 (beziehungsweise $081F). Der C 64 übernimmt eine Programmzeile erst in seinen Programmspeicher, nachdem man die RETURN-Taste gedrückt hat. Dann wird vom Betriebssystem die eingegebene Zeile Zeichen für Zeichen vom Bildschirm übernommen. Der Interpreter hat also beim Eingeben des oben angeführten Programms die Zeichenfolge SCHIFF$ ASCII-Zeichen für Zeichen in die Zellen ab $0805 eingeschrieben, bis er auf die für ihn bekannte Zeichenfolge IF traf. Folgerichtig schrieb er für IF das Token $8B ein und übernahm den Rest der Zeile wie gewohnt. Beim Abarbeiten des Programms versucht der Interpreter, die im Programmspeicher vorgefundenen Zeichen wieder seriell zu lesen und die einzelnen Befehle abzuarbeiten.

Dies mißlingt aber bereits in der ersten Zeile, weil er die Folge (ASCII)SCH(BASIC)IF(ASCII)F$ nicht zu interpretieren vermag. Weil der Interpeter in diesen Fällen so unnachgiebig ist, sind also alle Variablennamen verboten, die Basic-Wörter enthalten (zum Beispiel OR oder TAN).

Beispiele für verbotene Namen sind also: **TAN**NE, L**AND**$, **OR**T$, **ST**ART, um nur einige zu nennen. Wie man an der kleinen Aufzählung erkennt, muß man ziemlich aufpassen, um nicht wieder vom »SYNTAX-TERROR« erwischt zu werden.

Die Lesbarkeit von Programmen leidet schon ein wenig unter dieser Einschränkung, da man sehr oft auf unverständlichere und weniger selbsterklärende Variablennamen ausweichen muß. Mit Hilfe eines kleinen Tricks läßt sich der Interpreter aber so »über's Ohr hauen«, daß er verbotene Variablennamen akzeptiert und dem Programmierer quasi freie Auswahl für seine Variablennamen gibt.

Wer aufmerksam den ersten Teil dieses Artikels gelesen hat, kann sich vielleicht schon denken, wie dieser Trick funktioniert. Richtig, es werden wieder die Grafikzeichen an den entsprechenden Stellen eingesetzt, um den Interpreter von unseren Absichten zu überzeugen.

Geben Sie also ein:

```
10  SCHI{SHIFT-J}F$="TITANIC"
20  PRINT SCHI{SHIFT-J}FF$
```

Ein RUN bestätigt, daß unser Schiff soeben auf den Namen TITANIC getauft wurde und die SYNTAX-ERROR-Meldung des C 64 ausbleibt. Der Monitor zeigt, was mit dem Programm, das sich beim Listen äußerlich in nichts vom vorstehenden Beispiel unterscheidet, geschehen ist.

```
0800  00 17 08 0A 00 53 43 48
0808  49 46 46 24 B2 22 54 49
0810  54 41 4E 49 43 22 00 25
0818  08 14 00 99 20 53 43 48
0820  49 46 46 24 00 00 00
```

Ein Blick auf die Speicherstellen ab $0808 und $0820 zeigt, daß der Interpreter sich überlisten ließ und SCHIFF$ als sieben ASCII-Zeichen gespeichert hat. So kann er das Programm auch ausführen, weil er nicht mehr durch das Basic-Token IF verwirrt wird.

Das Drücken der RETURN-Taste bewirkt, daß der Interpreter jedes einzelne Zeichen der aktuellen Programmzeile übernimmt und auf Zeichenfolgen achtet, die ihm als Basic-Anweisungen oder -Funktionen bekannt sind. Letztere würde er als Token abspeichern. Nachdem er die Zeichenfolge SCH gelesen hat, hat er bereits eine Menge Arbeit hinter sich gebracht.

Die Folge SC kennt er nicht, wohl aber die Folge CH. Als nächste Zeichen könnten R und $ dann CHR$ ergeben, was als Token $C7 abgespeichert würde. Aber da er ein I als nächstes Zeichen liest, sagt er sich, vergessen wir die vorangehende Zeichenfolge und legen diese als ASCII-Code im Speicher ab. Bei I allerdings hat der Interpreter wieder seine Ohren gespitzt, denn dies könnte der Anfang von IF (Token $8B) oder auch INPUT (Token $85), INPUT # (Token $84) oder INT (Token $B5) sein.

Der Trick besteht darin, an geeigneter Stelle innerhalb des verbotenen Variablennamens ein Grafikzeichen einzugeben, um die Zeichenfolge so aufzutrennen, daß der Interpreter kein Basic-Wort (Token) erkennen kann.

Bei der Benutzung dieses Tricks ist streng zu beachten, daß jeder Variablenname genau gleich eingegeben wird, damit die Abspeicherung überall gleich ist, wenn ein- und dieselbe Variable mehrmals in einem Programm benutzt wird. Natürlich gilt trotz Benutzung des Tricks mit dem Grafikzeichen weiterhin, daß der C 64 nur die ersten beiden Zeichen eines Variablennamens auswertet. Deshalb ist die Benutzung von TONNE und TORTE im gleichen Programm trotz Eingabe mit unserem Trick nicht eindeutig und kann zu Problemen führen. Weiterhin ist bei zeitkritischen Programmen die Verwendung einstelliger Variablennamen günstiger. Wie weit allerdings die Verwendung von verbotenen Variablen sinnvoll ist, bleibt Ihnen überlassen. War nun das »GOTO« ein Befehl oder eine Variable? Eine Frage, die sich häufiger stellen wird, beim sorglosen Umgang mit verbotenen Variablen. Denkbar ist auch, mit den »verbotenen Variablen« ein Listing nahezu unlesbar zu gestalten (IF GOTO=1 THEN GOTO THEN, etc.), obwohl es optisch wunderbar strukturiert aussieht. Zum Schluß noch eine Bitte der Redaktion. Schicken Sie uns keine derart manipulierten Listings. Es kennt schließlich nicht jeder den Trick mit den Grafikzeichen.

(Dipl.-Ing. Raimund Trierscheid/hm)