Basic 7.0-Programme mit Struktur

Die große Basic-Überraschung findet vor allen Dingen bei den Aufsteigern vom Commodore 64 zum C128 statt. Endlich Befehle, mit denen man die Fähigkeiten des Computers auch ausnützen kann.

Diese überragenden Basic-Befehle sieht man nur, wenn man Programme vom Commodore 64 mit demselben Programm im Basic 7.0 des C128 vergleicht. Das sieht dann bei einer Grafik aus wie in Listing 1.

Für die Lissajous-Figur werden im Basic 2.0 des Commodore 64 14 Programmzeilen gebraucht. Dagegen benötigt der Commodore 128 nur sechs Programmzeilen und das Programm sieht wesentlich aufgeräumter aus (Listing 2).

Noch krasser ist der Unterschied, wenn ein kleines Liedchen programmiert wird. Dazu braucht der C64 etliche Datazeilen, einen Wulst von POKEs und Schleifen (Listing 3).

Der Commodore 128 braucht für das gleiche Programm ganze vier Programmzeilen (Listing 4).

Neben den neuen Sound-, Grafik- und Datei-Befehlen sind auch neue Befehle für das komfortablere Programmieren hinzugekommen. In erster Linie sind das Programmierhilfen wie TRON oder HELP, die dem Programmierer das Testen von Basic-Programmen wesentlich erleichtern. Daneben gibt es aber auch eine kleine Gruppe von Befehlen, die die logische Struktur eines Basic 7.0-Programmes enorm vereinfachen.

Kleine Befehlserweiterungen – große Wirkung

Das Commodore Basic 2.0 kennt bereits zwei Befehle, mit denen ein Programm übersichtlich wird: FOR...TO...NEXT sowie IF...THEN. Allerdings waren hiermit keine weitreichenden Aufgaben zu erfüllen, weil die Anschlußbedingungen immer in eine Zeile passen mußten. Die Folge war ein mit GOTO und GOSUB gespicktes Listing. Wer schon ein solches Programm mit vielen Abfragen und

Das neue Basic 7.0 liefert nicht nur tolle Befehle, sondern auch einen Vorteil, wie man ihn sonst nur bei höheren Programmiersprachen findet: Es unterstützt strukturiertes Programmieren.

Programmverzweigungen programmiert hat, weiß, warum man diese Art von Programmen mit Spaghettis vergleicht. Das Programm »wurstelt« sich von einer Zeile zur nächsten, verzweigt in andere Programmzeilen und springt mehrmals in die gleichen Abfragen. Eine Analyse des Programmes ist je nach Länge sehr schwierig. Ein Beispiel für diese Art von Programmierung ist das Listing »Hamburger« (Listing 5).

Für die Abfrage einer Bedingung wird in Basic meist die Anweisung IF..THEN benutzt. Dieser Befehl ist auf dem Commodore 128 wesentlich verbessert worden. Er erhielt ein Element, wie man es normalerweise nur von höheren Programmiersprachen kennt: die ELSE-Anweisung.

Wenn das Wörtchen ELSE nicht wär

Bei der Verwendung von IF... THEN im C64-Basic muß man fast immer auch mit dem Befehl GOTO oder GOSUB arbeiten, da nur wenige Befehle hinter der Anweisung THEN Platz finden. Hat man mehrere Befehle, muß man in ein eigenes Programmteil springen und nach der Ausführung wieder zurückkehren. Dadurch wird umständlich im Basic-Programm verzweigt beziehungsweise herumgesprungen.

Die erste wesentliche Hilfe für bessere Programmstrukturen ist der Befehl ELSE. Der IF-Befehl sieht dann folgendermaßen aus: IF...THEN...ELSE.

Wenn die IF-Bedingung erfüllt wird, dann wird die hinter THEN stehende Anweisung ausgeführt. Ansonsten wird – sofern eine ELSE-Anweisung vorhanden ist – die ELSE-Bedingung ausgeführt.

```
10 INPUT"WER HAT BASIC 7.0";A$
20 IF A$="COMMODORE 128" THEN
PRINT"RICHTIG":END:ELSE PRINT
"FALSCH":GOTO 10
```

Bei unserem Programmbeispiel wird in Zeile 10 auf eine Eingabe gewartet. In Zeile 20 wird bei richtiger Beantwortung das Programm beendet, im anderen Fall wird weiter gefragt. Der Commodore 64 bräuchte eine Zeile mehr, das Listing sieht bei ihm folgendermaßen aus:

```
10 INPUT"WER HAT BASIC 2.0";A$
20 IF A$="COMMODORE 64" THEN
PRINT"RICHTIG":END
30 PRINT"FALSCH":GOTO 10
```

Ist beim Basic 7.0 nach einer IF...THEN-Abfrage kein ELSE vorhanden, dann springt das Programm in die nächste Zeile. Bei der ELSE-Anweisung muß man darauf achten, daß vor dem ELSE ein Doppelpunkt steht. Befindet sich vor dem ELSE ein Leerzeichen, dann wird die ELSE-Anweisung nicht erkannt und auch nicht ausgeführt. Das Programm fährt einfach im Ablauf in der nächsten Zeile fort.

Wie springt man in Basic?

Die ELSE-Anweisung muß sich in der gleichen Programmzeile befinden, in der THEN steht. Da bleibt natürlich nicht viel Platz für Programmbefehle. Also doch wieder GOTO und GOSUB? Glücklicherweise nicht, den die IF...THEN-Anweisung ist noch weiter verbessert worden.

Um umfangreichere Schleifenbedingungen abzufangen, hat das Basic 7.0 des Commodore 128 die Kommandos BEGIN und BEND bekommen. Diese beiden Befehle dürfen aber nur im Zusammenhang mit der IF...THEN-Anweisung benutzt werden und müssen sich hinter dem THEN befinden. Damit kann man einem beliebig langen,

mehrere Programmzeilen umfassenden Befehlscode mit der Abfrage verknüpfen.

```
10 PRINT"HAST DU HUNGER? (J/N)"
20 GETKEY A$
30 IF A$="J" THEN BEGIN
40 PRINT"WARUM MACHST DU DIR
DANN NICHTS ZU ESSEN? DU"
50 PRINT"HAST WOHL VERGESSEN
EINZUKAUFEN?"
60 PRINT"DA KANN ICH DIR LEIDER
NICHT HELFEN."
70 BEND:ELSE:PRINT"ICH AUCH
NICHT"
80 END
```

Dieses kleine Programm fragt, ob Sie Hunger haben. Wenn ja, dann werden die Zeilen 40 bis 60 ausgeführt. Wenn Sie N oder ein anderes Zeichen als J eingeben, dann verzweigt das Programm nach Zeile 70. Dort wird mit BEND der Programmteil nach der THEN-Anweisung beendet.

Starke Hilfe durch BEGIN und BEND

Wie man sieht, darf auch bei IF...THEN mit anschließendem BEGIN...BEND noch ein nachfolgendes ELSE stehen. Hier finden wir also eine Ausnahme zu der oben beschriebenen Regel, daß sich das ELSE in der gleichen Programmzeile wie THEN befinden muß. Das ELSE muß sich aber direkt nach der BEND-Anweisung befinden und auch hier wieder den obligatorischen Doppelpunkt setzen. Nach dem ELSE kann nochmals ein BEGIN...BEND folgen.

Durch diesen erweiterten IF-Befehl kann in der Regel auf GOTO und GOSUB verzichtet werden. Völlig neu ist ein weiterer, sehr mächtiger Basic-Befehl: DO...LOOP.

Mit DO...LOOP lassen sich im Basic 7.0 Programmschleifen ohne GOTO erzeugen. Diese Anweisung ist mit FOR...NEXT zu vergleichen. Bei der FOR...NEXT-Anweisung muß jedoch von Anfang an die Anzahl der Schleifen-Durchläufe festgelegt sein.

```
10 FOR A=0 TO 10
20 PRINT"2 HOCH "A" ="2↑A
30 NEXT
```

Unser Beispiellisting gibt die ersten 11 Zweierpotenzen auf dem Bildschirm aus. Jetzt gibt es jedoch Fälle, in denen von vornherein nicht bekannt ist, wie oft eine Schleife auszuführen ist. Beispielsweise wenn man auf die Erfüllung einer Bedingung wartet, die durch den Schleifenablauf beeinflußt wird. In einem solchen Fall nimmt man DO...LOOP.

Programmteile zwischen DO und LOOP werden endlos wiederholt. Um jetzt keine »toten« Endlosschleifen zu erzeugen, braucht man eine Anweisung, um aus der Schleife wieder heraus zu kommen. Eine Form, die DO...LOOP-Schleife abzubrechen besteht im Basic-Befehl EXIT. Dieser Befehl wird häufig mit einer IF-Abfrage verknüpft sein. Findet das Programm in einer Schleife ein EXIT, so springt es direkt hinter den LOOP-Befehl und setzt dort das Programm fort.

Exit – der Basic-Aussteiger aus Endlos-Schleifen

```
10 INPUT"WIE GROSS DARF DIE
ZWEIERPOTENZ MAXIMAL SEIN";ZAHL
20 DO
30 C=A:A=2↑B:B=B+1
40 IF A>ZAHL THEN EXIT
50 LOOP
60 PRINT"DIE UNTER"ZAHL"LIEGENDE
ZWEIERPOTENZ"
70 PRINT"HAT DEN WERT VON
2 HOCH"B-1"="C
80 PRINT"DIE UEBER"ZAHL"LIEGENDE
ZWEIERPOTENZ"
70 PRINT"HAT DEN WERT VON
2 HOCH"B"="A
```

Mit diesem Programm wird eine Zweierpotenz gesucht, die nicht größer als die von uns eingegebene Zahl ist. Dazu wird die Schleife DO...LOOP so lange durchlaufen, bis die Abfrage in Zeile 40 feststellt, daß die Zweierpotenz den vorgegebenen Wert überschritten hat. Dann wird hinter die Schleife verzweigt und in den Zeilen 60 bis 80 werden die Ergebnisse ausgegeben.

Eine andere Form, die DO...LOOP-Schleife abzubrechen, findet man in der Befehlserweiterung UNTIL oder WHILE. Beide Befehle können nur als Zusatz zu DO....LOOP benutzt werden.

Bei UNTIL wird geprüft, ob eine vorgegebene Bedingung zutrifft. Die Schleife wird also so lange durchlaufen, bis (englisch: until) der nach UNTIL stehende Ausdruck logisch wahr wird, das heißt bis dieser Ausdruck stimmt.

```
10 PRINT"SPIELEN WIR ZAHLENRATEN
? (J/N)"
20 GETKEY A$
30 IF A$="J"THEN BEGIN
40 X=RND(1):X=INT(100*X)
50 BEND:ELSE PRINT"SCHADE,
TSCHUESS!":END
60 DO UNTIL A=X
70 INPUT"WIE HEISST MEINE ZAHL";A
80 IF A>X THEN PRINT"ZU GROSS"
90 IF A<X THEN PRINT"ZU KLEIN"
100 LOOP
110 PRINT"HURRA, RICHTIG."
120 END
```

Bei diesem kleinen Spiel muß eine zufällige Zahl zwischen 0 und 100 geraten werden. Gibt man auf die Frage des Computers ein J ein, dann wird diese zufällige Zahl in der Zeile 40 mit der RND-Funktion vom Computer errechnet. Wenn man nicht spielen will, verzweigt das Programm hinter das BEND in Zeile 50. Die Schleife beginnt mit dem DO in Zeile 60. In dieser Zeile wird abgefragt, ob schon die richtige Antwort eingegeben wurde. Ansonsten gibt der Computer in den Zeilen 80 und 90 Tips, ob die Zahl größer oder kleiner als der eingegebene Wert ist. Hat man das Ergebnis erraten, wird nach der Zeile 110 hinter der Schleife verzweigt.

Schleifen nach Maß – DO...LOOP

Dieses Spiel muß nach dem Erraten einer Zahl wieder mit RUN gestartet werden. Das kann man sich natürlich einfacher machen, indem man noch ein DO...LOOP um das ganze Programm setzt. Fügen Sie folgende Zeilen hinzu:

```
5 DO
120 LOOP
```

Jetzt ist das Programm in einer Endlosschleife gefangen, die nur durch das END in Zeile 50 sowie durch die RUN/STOP-Taste abgebrochen wird.

In unserem Programm sitzt die UNTIL-Abfrage direkt hinter dem DO. Sie kann aber auch hinter dem LOOP sitzen. In unserem Fall wäre das sogar besser, weil noch ein kleiner Fehler im Programm steckt. Wenn der Computer in der Zeile 40 zufällig die Zahl 0 nimmt, dann ist die Abfrage in Zeile 60 wahr, obwohl wir noch keine Zahl eingegeben haben. Der Unterschied in der Plazierung hat also eine große Wirkung. Im Fall des DO UNTIL wird die Schleife erst nach der Abfrage ausgeführt, bei DO...LOOP UNTIL erfolgt die Abfrage der Bedingung erst nach dem ersten Durchlauf der Schleife.

Deshalb muß unser Listing so aussehen:

```
5 DO
10 PRINT"SPIELEN WIR ZAHLENRATEN?
(J/N)"
20 GETKEY A$
30 IF A$="J"THEN BEGIN
40 X=RND(1):X=INT(100*X)
50 BEND:ELSE PRINT"SCHADE,
TSCHUESS!":END
60 DO
70 INPUT"WIE HEISST MEINE ZAHL";A
80 IF A>X THEN PRINT"ZU GROSS"
90 IF A<X THEN PRINT"ZU KLEIN"
100 LOOP UNTIL A=X
110 PRINT"HURRA, RICHTIG."
120 LOOP
```

Bei WHILE wird die Schleife so lange ausgeführt, wie die Bedingung hinter WHILE logisch wahr ist, das heißt solange es stimmt, was hinter WHILE steht. Im Gegensatz zu UNTIL wird bei WHILE also darauf gewartet, das eine Bedingung nicht mehr logisch wahr ist. Auch WHILE kann sowohl hinter DO als auch hinter LOOP stehen.

```
10 DO WHILE A<100
20 B=A*A:PRINT A
30 LOOP
```

Diese Form der Basic-Schleifengestaltung macht sowohl die Programmierung einfacher als auch ein Programm übersichtlicher. Für die Übersichtlichkeit gibt es aber auch noch einen anderen Weg.

Programmzusammenhänge sichtbar machen

Unter strukturiertem Programmieren kann man auch die Form des Listing-Aufbaues sehen. Es gilt allgemein als guter Programmierstil, wenn bestimmte Programmteile durch Einrücken des Textes auf dem Bildschirm sichtbar gemacht werden. Dadurch sieht man sofort, wie das Programm gegliedert ist, wo ein Programmteil anfängt und wo er endet. In Basic ist diese sichtbare Strukturierung nur bei der Arbeit am 80-Zeichen-Bildschirm sinnvoll, da bei 40-Zeichen-Darstellung jeweils zwei Bildschirmzeilen für eine Programmzeile zur Verfügung stehen. Da nützt auch die schönste Einrückung von Programmzeilen nichts. Dasselbe Programm auf dem 80-Zeichen-Monitor wirkt dagegen aufgeräumt und übersichtlich. Bei dem Einrücken der Zeilen hilft die TAB-Taste. Bei einem Druck auf die TAB-Taste springt der Cursor jeweils in 8-Zeichen-Schritten weiter. Man muß nicht immer wieder eine Reihe von Leerzeichen eingeben. Allerdings ist zu Beginn jeder Zeile ein Doppelpunkt notwendig, weil der Basic-Editor des C 128 überflüssige Leerzeichen nach der Zeilennummer ignoriert. Er setzt die Befehle an den Zeilenanfang zurück. Mit dem Doppelpunkt wird jedoch der Zeilenanfang definiert und nachfolgende Leerzeichen als Anweisung ohne Wirkung interpretiert.

Lust auf Basic

Sicherlich ist es nicht leicht, in Basic übersichtlich zu programmieren. Sowohl die sichtbare als auch die logische Struktur erfordert Programmiererfahrung. Wer sich daran gewöhnt hat, wird die Vorteile schnell erkennen. Man spart Zeit und Nerven.

Das Basic 7.0 des Commodore 128 ist ein sehr wirksames Werkzeug, sowohl in der Ausnutzung der Fähigkeiten dieses tollen Computers als auch in der Art, wie sich das Basic programmieren läßt. Programme schreiben macht auf jeden Fall auf dem Commodore 128 mit seinem Basic 7.0 sehr viel Spaß.

(zu)

Quelle: Basic 7.0 auf dem Commodore 128, Markt&Technik Verlag AG, ISBN 3-89090-170-0

```
10 GRAPHIC 1,0:SCNCLR
20 COLOR 0,1: COLOR 1,2
30 FOR K=0 TO 2*↑ + 0.05
   STEP 0.01
40 X=150+10-150*COS(3*K)
45 Y=100-150*0.5*SIN(5*K)
50 DRAW,X,Y TO X,Y
60 NEXT
```

Listing 1. Grafik auf dem C128

```
10 VOL 15
20 ENVELOPE 0,8,8,6,0,1
30 TEMPO 30
40 PLAY "V1 T0 04 QCQDQEQFHGHG QAQA
   AQAQAHGR QAQAQAQAHGR QFQFQFQFHEHE
   QDQDQDQDHC"
```

Listing 4. Dasselbe Lied im 7.0 Basic

```
10 POKE 53272,PEEK(53272)OR 8:POKE
   646,0
20 POKE 53265,PEEK(53265)OR 32
30 FOR K=0 TO 999:POKE 1024+K,14:N
   EXT
40 FOR I=0 TO 7999:POKE 8192+I,0:N
   EXT
50 FOR K=0 TO 2*↑+0.05 STEP 0.01
60 X=150+10-150*COS(3*K)
70 Y=100-150*0.5*SIN(5*K)
80 FOR N=0 TO 24
90 IF Y>N*8-1 AND Y<(N+1)*8 THEN B
   Y=8192+N*320+8*INT(X/8)+Y-8*N:N
   =24:GOTO 110
100 NEXT N
110 BI=8*(1+INT(X/8))-X-1
120 BY=INT(BY+0.5)
130 POKE BY,PEEK(BY)OR 2↑INT(BI+0.
   5)
140 NEXT:NEXT
150 GET A$:IF A$="" THEN 140
```

Listing 2. Dieselbe Grafik auf dem C64

```
10 A$="HAMBURGER":B$="SENF":C$="KE
   TCHUP":D$="SALAT":E$="FLEISCH":
   F$="BROETCHEN"
11 INPUT"WAS MOECHTEN SIE BITTE";Z
   $
12 IF Z$=A$THEN GOTO 16
13 IF Z$<>A$THEN PRINT"HABEN WIR L
   EIDER NICHT. WIE WAERS MIT EINE
   M HAMBURGER?"
14 GOTO 11
15 PRINT"KOMMEN SIE MORGEN WIEDER,
    MIR WIRD UEBEL":GOTO 24
16 INPUT"NA PRIMA, WAS GEHOERT DAZ
   U";Z$
17 IF Z$=B$THEN PRINT"WIE DAS SCHM
   ECKT!":GOTO 23
18 IF Z$=C$THEN PRINT"DARF NICHT F
   EHLEN!":GOTO 23
19 IF Z$=D$THEN PRINT"SIE SIND EIN
    KENNER!":GOTO 23
20 IF Z$=E$THEN GOTO 15
21 IF Z$=F$THEN PRINT"DAMIT ER NIC
   HT SO IN DER HAND RUMFLUTSCHT!"
   :GOTO 23
22 PRINT"ABER WIRKLICH NICHT!"
23 INPUT"WAS NOCH";Z$:GOTO 17
24 FOR A=1 TO 3000:NEXT:GOTO 10
```

Listing 5. »Hamburger«

```
10 SI=54272
20 FOR L=SI TO SI+24:POKE L,0:NEXT
30 POKE SI+5,9:POKE SI+6,0
40 POKE SI+24,10
50 READ HB,LB,DR
60 IF HB<0 THEN POKE SI+24,0
70 POKE SI+1,HB:POKE SI,LB
80 POKE SI+4,33
90 FOR Z=1 TO DR:NEXT
100 POKE SI+4,32:FOR Z=1 TO 50:NEX
    T
110 GOTO 50
200 DATA 17,103,250
210 DATA 19,137,250
220 DATA 21,237,250
230 DATA 23,059,250
240 DATA 26,020,500
250 DATA 26,020,500
260 DATA 29,069,250
270 DATA 29,069,250
280 DATA 29,069,250
290 DATA 29,069,250
300 DATA 26,020,500
310 DATA 29,069,250
320 DATA 29,069,250
330 DATA 29,069,250
340 DATA 29,069,250
350 DATA 26,020,500
360 DATA 23,059,250
370 DATA 23,059,250
380 DATA 23,059,250
390 DATA 23,059,250
400 DATA 21,237,500
410 DATA 21,237,500
420 DATA 19,137,250
430 DATA 19,137,250
440 DATA 19,137,250
450 DATA 19,137,250
460 DATA 17,103,500
470 DATA -1, -1, -1
```

Listing 3. »Alle meine Entchen« auf dem C64