Basic 7.0 – Das Super-Basic des C 128

C 64-Besitzer können ein Klagelied davon singen: Nichts geht bei einer anspruchsvolleren Programmierung ohne die Befehle PEEK und POKE, ohne SYS und endlose DATA-Wüsten. Das wird nun mit dem C 128 ganz anders. Wir haben für Sie dieses völlig neue Programmier-Gefühl ausprobiert.

Schwachstellen im System sind die Vorliebe vieler Programmierer. Sie überlegen tagtäglich, wie man die Floppy schneller, das Druckbild schöner und das Basic komfortabler machen kann. Gerade letzteres eröffnet dem C 64-Benutzer ein großes Betätigungsfeld. Mit dem C 128 und dessen Basic 7.0 ändert sich das nun grundlegend. Viele Funktionen, die auf dem C 64 mühselig in Maschinensprache realisiert werden mußten, stehen beim C 128 als komfortable Basic-Befehle zur Verfügung. Harte Zeiten also für Maschinensprache-Freaks? Auf jeden Fall aber rosige Zeiten für Basic-Programmierer. Auf den nächsten Seiten wollen wir Ihnen zeigen, wie einfach nun das Programmieren geworden ist.

Strukturierte Programmierung

Einige Befehle des 7.0-Basic erlauben strukturierte Programmierung. So mancher Anwender wird sich nun fragen, welchen Nutzen man eigentlich davon hat. Nun, durch die Struktur wird ein Programm wesentlich übersichtlicher, das heißt man kann die Arbeitsweise wesentlich leichter durchschauen und verstehen, als die des normalerweise verwendeten »Spaghetticodes«, bei dem alle Verzweigungen durch GOTO-Befehle realisiert werden.

Nun aber zu den neuen Befehlen im einzelnen. Die DO..LOOP-Schleife, ist in etwa mit den FOR..NEXT-Befehlen vergleichbar. Damit ist es unter anderem möglich, GOTO-freie Schleifen zu erzeugen. Der Programmteil zwischen DO und LOOP wird endlos wiederholt. Damit sich der Computer jedoch nicht in dieser Schleife verfängt, können auch Bedingungen angegeben werden, unter denen die Schleife ausgeführt wird.

WHILE und UNTIL sind in ihrer Arbeitsweise sehr ähnlich. Sie können entweder dem DO-Kommando (also zum Beispiel DO WHILE) folgen, oder hinter dem Schleifenrumpf stehen (LOOP UNTIL). WHILE übersetzt man sinnvollerweise mit »arbeite, solange die Bedingung erfüllt ist«. Die Schleife wird also dann beendet, wenn die Bedingung **nicht** mehr erfüllt ist:

```
10 DO WHILE A<=5
20 : INPUT A
30 LOOP
40 PRINT "ENDE"
```

In diesem kleinen Beispiel wurde eine Schleife programmiert, die so lange durchlaufen wird, bis man eine Zahl eingibt, die größer als 5 ist. Dann ist die Schleifenbedingung nicht mehr erfüllt und die Programmausführung wird in Zeile 40 fortgesetzt.

UNTIL ist genau das Gegenstück zum WHILE-Kommando. Hier gilt: Die Abarbeitung wird beendet, sobald die Bedingung erfüllt ist (das Basic bleibt also in der Schleife, solange die Bedingung nicht erfüllt ist). Aber auch hierzu wieder ein kleines Beispiel:

```
10 DO
20 : INPUT A
30 LOOP UNTIL A<5
40 PRINT "ENDE"
```

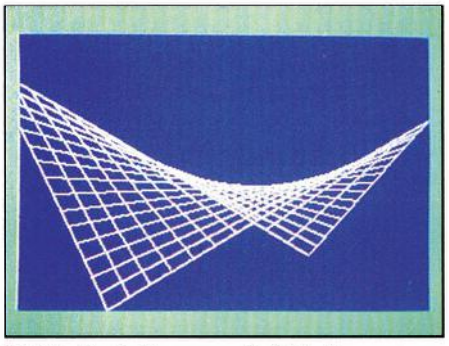

Bild 1: Dank der neuen Befehle kann man solche Figuren ohne viel Aufwand zeichnen

Diesmal arbeitet das Programm genau anders herum, das heißt wenn der eingegebene Wert von A kleiner als 5 ist, dann verzweigt das Programm nach Zeile 40. Welchen der beiden Befehle man einsetzt ist Geschmackssache, denn man kann die Bedingung immer so »verdrehen«, daß beide verwendet werden können.

Die Aufmerksamen unter Ihnen werden bestimmt schon bemerkt haben, daß die Bedingung im ersten Beispiel beim DO, im zweiten aber bei LOOP steht. Steht UNTIL oder WHILE am Schleifenkopf, dann fragt das Basic bereits beim Eintritt in die Schleife, ob die Aussprungbedingung zutrifft. Dies würde wiederum beim eben gezeigten Beispiel dazu führen, daß der Computer den Programmablauf gleich nach der Eingabe von »RUN« wieder beenden würde, da die Variable A dann ja die Null enthält. Damit wäre die Bedingung nicht erfüllt, obwohl man noch gar keine Gelegenheit gehabt hatte, in Zeile 20 einen Wert einzugeben.

Eine weitere Möglichkeit, eine Schleifenkonstruktion zu verlassen, bietet der Befehl EXIT. Wie der Name schon sagt, beendet man damit die laufende Abarbeitung und verzweigt in die nächste Zeile nach

```
100 REM ************************************
110 REM ***    HYPERBOLISCHES PARABOLID    ***

130 REM ************************************
140 REM
145 GRAPHIC1,1
150 SCALE 1,1023,1023
155 COLOR 1,2
160 A= 800:B= 300:C= 200:D= 700:X= 15
170 FOR T= 0 TO X: DRAW 1,B/X*T,C/X*T+D/X*(X-T)
    TO A+B/X*T,(C+D)/X*T: NEXT
180 FOR T= 0 TO X: DRAW 1,A/X*T,D/X*(X-T)
    TO B+A/X*T,C+D/X*T: NEXT
```

Listing 1. Das Basic-Programm zu Bild 1

LOOP. In der Regel wird man den EXIT-Befehl daher von einer Bedingung abhängig machen.

Eine weitere Quelle von Ärgernissen beim Programmieren waren bisher die IF..THEN-Konstruktionen. Hinter dem THEN fanden oft nur drei bis vier Befehle Platz, so daß man oftmals umständlich im Programm hin- und herspringen mußte, um die Ausführung fortzusetzen. Auch dies wurde jetzt mit den Kommandos BEGIN und BEND anders.

Man setzt in den THEN-Zweig einfach ein BEGIN-Kommando ein. Trifft die Bedingung bei der IF..THEN-Abfrage zu, dann behandelt das Basic alle folgenden Zeilen bis BEND so, als würden sie in der gleichen Zeile hinter dem THEN-Befehl stehen. Erst ein BEND-Befehl

beendet die Abarbeitung der THEN-Folge.

Ebenfalls neu ist beim 7.0-Basic das Befehlswort ELSE. Findet der Computer im Basic-Programm eine IF-Abfrage vor, dann hat er prinzipiell zwei Möglichkeiten. Entweder die Bedingung trifft zu, dann wird der Programmteil nach THEN ausgeführt, der sich jetzt dank BEGIN und BEND über beliebig viele Zeilen erstrecken kann. Andernfalls ignoriert das Basic den Rest der Zeile und fährt in der nächsten mit der Programmabarbeitung fort. Genau für diesen Fall wurde der Befehl ELSE vorgesehen. Wie er eingesetzt wird, soll wiederum ein Beispiel verdeutlichen:

```
10 INPUT A
20 IF A=2 THEN B=9 : ELSE B=2
30 PRINT B
```

Der ELSE-Zweig wird in der Abfrage genau dann ausgeführt, wenn die Bedingung (hier A = 2) nicht zutrifft. Die Benutzung dieser Konstruktion hat wiederum den Vorteil, daß man nicht mehr um die nächste Zeile »herumspringen« muß, was wiederum der Übersichtlichkeit des Programms zugute kommt.

Auch im ELSE-Zweig können die neuen BEGIN- und BEND-Kommandos eingesetzt werden, wodurch sich dieser wiederum auf mehrere Zeilen ausdehnen läßt.

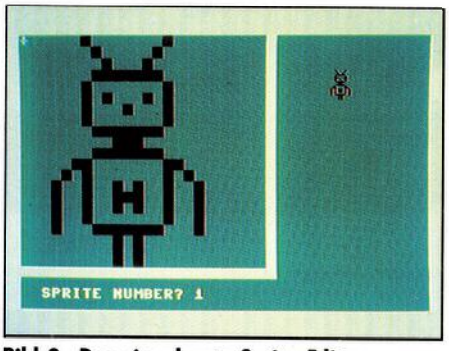

Bild 2: Der eingebaute Sprite-Editor des C 128

Auch auf einem ganz anderem Gebiet hat sich beim 7.0 Basic einiges verändert. Gegenüber dem Basic 2.0 des C 64 sind sehr komfortable Kommandos zum Zeichnen von Punkten, Linien, Rechtecken und Kreisen hinzugekommen. Auch das Ausfüllen von geschlossenen Flächen ist mit Basic 7.0 kein Problem mehr. Leider beziehen sich alle Grafik-Befehle nur auf die bereits vom C 64 bekannte Grafikauflösung von 320 x 200 Punkten. Die ebenfalls — wenn auch auf Umwegen — mögliche doppelte Auflösung von 640 x 200 Punkten wird vom Basic leider nicht unterstützt. Die Grafik-Befehle des 7.0-Basic entsprechen den bereits vom 3.5-Basic des C 16 und Plus 4 bekannten Kommandos. Auch die zahlreichen Befehlserweiterungen für den C 64 verwenden ähnliche Grafik-Kommandos. In Listing 1 ist ein kleines Beispielprogramm zu finden, um den Lesern, die keine C 64-Basic-Erweiterung besitzen, zu demonstrieren, wie einfach selbst kompliziertere Figuren (Bild 1) erstellt werden können.

Wirklich neu sind hingegen die Kommandos für die Verwaltung von Sprites. Beginnen wir mit der Definition eines solchen beweglichen Objekts. Sehr hilfreich ist da der eingebaute Sprite-Editor, der entweder vom Programm aus oder direkt mit »SPRDEF« aufgerufen werden kann (Bild 2). Hierin wird man zunächst einmal nach der Nummer des Sprites gefragt. Hat man diese Zahl zwischen 1 und 8 eingegeben, so kann es mit dem eigentlichen Sprite-Entwurf losgehen. Dazu bewegt man den Zeichenzeiger mit den normalen Cursortasten auf dem 24 x 21 großen Feld hin und her. Das Löschen beziehungsweise Setzen von einzelnen Pixels erfolgt dann mittels der Tasten 1 und 2 (bei Multicolorsprites auch noch die 3 und 4). Ist der künstlerische Entwurf geglückt, dann kann man mit SHIFT-RETURN die Definition beenden.

```
10 PRINT"HABEN SIE IHRE SPRITES
   SCHON DEFINIERT ?": GETKEY A$
20 IF A$= "J" THEN 50
30 PRINT"SPRITE #1": SLEEP 1: SPRDEF
40 PRINT"SPRITE #2": SLEEP 1: SPRDEF
50 COLLISION 1,1000
60 SCNCLR
70 SPRITE 1,1,1,0,1,1
80 SPRITE 2,1,2,1,1,1
90 MOVSPR 1,300,100
100 MOVSPR 2,0,100
110 MOVSPR 2,90#4
115 DO
120 IF BUMB (1)=0 THEN SCNCLR
130 LOOP
1000 SCNCLR: CHAR 1,10,20,"BOOOOOMMM"
1010 RETURN
```

Listing 2. So einfach ist der Umgang mit Sprites in Basic 7.0

Mit dem Editieren allein ist es natürlich nicht getan. Darum werden vom 7.0-Basic auch Befehle für das Setzen und Bewegen von Sprites zur Verfügung gestellt. Das Beispiel in Listing 2 soll hier wiederum die neuen Kommandos verstehen helfen. Hier hat der Benutzer zunächst einmal die Möglichkeit, sich zwei Sprites zu erstellen. Dazu wird der eingebaute Editor zweimal aufgerufen.

Der nächste neue Befehl COLLISION legt eine Programmzeile fest, in die das Basic verzweigen soll, wenn sich zwei Objekte treffen. Besonders positiv ist, daß sich der Benutzer nicht mehr explizit um die Abfrage zu kümmern braucht. Hat man die Ansprungzeile einmal festgelegt (in unserem Beispiel Zeile 1000), so unterbricht das Basic bei einer Kollision der entsprechenden Sprites automatisch den Programmablauf und verzweigt in die vorgegebene Zeile. Dort kann der Programmierer dann angemessen auf das Ereignis reagieren. Ist diese Abarbeitung beendet, so kehrt man mittels »RETURN« wieder in das zuvor unterbrochene Programm zurück.

Nun aber wieder zurück zu unserem Beispielprogramm in Listing 2. Als nächtes müssen die Attribute für jedes Sprite wie zum Beispiel Sichtbarkeit, Vordergrundfarbe, Priorität und so weiter, angegeben werden. Diese Festlegung erledigt man über den SPRITE-Befehl. Die Positionierung und das Bewegen der Objekte, wird mit einem anderen Kommando bewerkstelligt: mit »MOVSPR 1,200, 300« positioniert man Sprite 1 auf der Bildschirmposition X = 200 und Y = 300. Mit »MOVSPR 1,90#5« bringt man es dann auch noch zum Laufen. Die 90 gibt dabei die Himmelsrichtung an (90 = Bewegung nach rechts), die 5 ist für die Geschwindigkeit zuständig. Diese Bewegungen werden per Interrupt gesteuert, daß heißt auch darum muß sich der Benutzer nicht kümmern. Ist der Bewegungsablauf einmal programmiert, geht alles Weitere automatisch. Schaltet man die Sprites nicht wieder ab, so bewegen sie sich sogar nach Beendigung des Programms weiter über den Bildschirm!

Speichern und Laden von Sprites ist im neuen Basic natürlich auch möglich. Der Befehl SPRSAV überträgt die Grafikdaten — je nach Stellung der Parameter — in Strings oder zurück. Dies ist sehr einfach: Gibt man »SPRSAV 1, A$« ein, so wird das Sprite in diesem String abgespeichert. Vertauscht man die Parameter, also »SPRSAV A$,1«, so erreicht man den umgekehrten Prozeß: Der Inhalt des Strings A$ wird als Bitmuster dem Sprite Nummer 1 zugewiesen.

Dies war nun eine erste kleine Einführung in die Befehlsfülle des neuen 7.0-Basic. Mit einer näheren Beschreibung aller Basic-Befehle wäre leicht ein ganzes Buch zu füllen. Aber auch die Beschreibung der interessantesten neuen Sprachelemente dürfte schon so manchem C 64-Besitzer einen Vorgeschmack darauf geben, welche neuen Programmier-Dimensionen sich mit dem Basic 7.0 auftun.

(Christoph Sauer/ev)