Tips für sauberes Programmieren

Programmieren ist bekanntermaßen eine ausgesprochen kreative Tätigkeit, die viele mit Begeisterung ausüben. Erhalten Sie sich diese Freude durch die Anwendung einiger nützlicher Regeln.

Mal im Ernst, haben sie nicht auch schon Programme gesehen oder auch selbst erstellt, die im höchsten Maße unleserlich sind? Vor allem, wenn diese Programme in Basic geschrieben sind, vermißt man des öfteren eine gewisse Übersicht: Da wird ohne ein Konzept wild drauflos getippt, am Anfang weiß man gar nicht so recht, was aus der Programmidee eigentlich mal werden soll. Man fängt ganz locker an, und zuerst klappt alles auch sehr gut. Wenn dann die ersten Erfolge vorhanden sind, denkt man bei sich, daß das Programm ja eigentlich etwas mehr können müßte, oder man bemerkt einige Fehler, die sich während des Programmlaufs einschleichen. Nun beginnt man, kleine Routinen zu entwickeln, die dann an das Ende des Programms angehängt werden, oder, was noch schlimmer ist, irgendwo innerhalb des Programms, wo sie an sich nichts zu suchen haben. Schließlich hat man ja den GOTO-Befehl, der eventuelle Probleme wirksam »umgeht«. Und so entsteht dann mit der Zeit und mit wachsendem Programm eine kaum noch zu übersehende Aneinandereihung von Programmzeilen, gespickt mit GOTO-Befehlen. Wenn solche Programme dann veröffentlicht werden, hat der interessierte Leser zwar die Möglichkeit, dieses Produkt abzutippen, aber wenn er die Programmlogik erkennen und nachvollziehen will, stößt er auf allergrößte Schwierigkeiten. Da hilft manchmal auch eine einigermaßen ausführliche Programmbeschreibung nicht viel. Es soll allerdings Programmierer geben — und das sowohl bei den Amateuren als auch bei den sogenannten Profis — die allerhöchsten Wert darauf legen, von keinem durchschaut zu werden. Außerdem trauen sie sowieso keinem anderen eine Beurteilung ihrer Programme zu. Daß man denen einen schlechten Programmierstil vorwerfen kann, stört sie dann natürlich auch nicht. (Es gibt Fälle, wo Programmierer sich unkündbar gemacht haben, weil kein Mensch außer ihnen selbst das Programm begreift.)

Versuchen Sie dann mal, solch ein Programm zu erweitern, sinnvoll eine zusätzliche Funktion zu implementieren! Auch wenn Sie das Programm selbst »entworfen« haben, und dann nicht peinlich genau Buchführung über jeden Schritt geführt haben, sind Sie nach einem Jahr mit Sicherheit nicht mehr in der Lage, Ihr eigenes Produkt zu verstehen, geschweige es sinnvoll zu ändern beziehungsweise zu erweitern.

Es gibt aber Möglichkeiten, diese Schwierigkeiten zu verringern. Eine Möglichkeit davon ist die strukturierte Programmierung.

Ein Programm ist in der Regel eine Folge von Anweisungen, die der entsprechende Rechner ausführt. Ganz am Anfang der »Computerei« war man beschränkt auf eine sequentielle Methode der Ausführung, Das heißt, jeder Befehl wurde in der Reihenfolge ausgeführt, wie er auch im Programm vorkam. Eine Verzweigung zu einer anderen Stelle oder eine Wiederholfunktion gab es da noch nicht. Das führte dazu, daß diese Programme sehr starr waren. Man konnte keinen direkten Einfluß auf ihren Ablauf nehmen. Programmteile, die mehrmals vorkamen, mußten genauso oft eingegeben werden, wie sie benötigt wurden. Heute kennt jeder die Befehle, die Alternativen zum statischen Programmablauf zulassen. In Basic sind das die Befehle »GOTO« und »GOSUB«.

Programmanweisungen, die den Kontrollfluß bestimmen, zum Beispiel GOTOs, sind also die Ursache dafür, daß die Anweisungen eines Programms in einer anderen als der aufgeschriebenen Reihenfolge ausgeführt werden können (statisch-dynamisch). Ziel der strukturierten Programmierung ist es, durch eine disziplinierte Vorgehensweise die Fehleranfälligkeit zu reduzieren. In anderen höheren Programmiersprachen bedeutet dies zum Beispiel den Verzicht auf GOTOs. An dessen Stelle treten dann einige wenige andere logische Grundstrukturen. In erster Linie handelt es sich um die **Sequenz** von Operationen, **die Auswahl** IF...THEN...ELSE (Verzweigung mit einer oder zwei Bedingungen) und **die Wiederholung** DO..WHILE (einer Gruppe von Operationen, solange eine bestimmte Bedingung erfüllt ist). Neben diesen Grundstrukturen dürfen noch einige weitere Strukturen verwendet werden, im Regelfall jedoch nicht die unbedingte Verzweigung (GOTO).

Der Vorteil: Der Code ist sehr übersichtlich gruppiert und daher auch für andere Programmierer leicht lesbar. Das Testen von strukturiertem Code ist einfach. Strukturierter Code ist leichter wartbar als unstrukturierter.

Der Nachteil: Strukturierte Programme können durch den Verzicht auf GOTO-Anweisungen und durch Codewiederholungen umfangreicher werden als äquivalente, nichtstrukturierte Programme.

Nun besitzt das normale Standard-Basic diese Strukturen nicht. Wer aber als C64-Besitzer in der glücklichen Lage ist, die von Commodore angebotene Basic-Erweiterung Simons Basic sein eigen zu nennen, findet dort einige dieser Befehle (siehe Bericht in dieser Ausgabe). Auch die dort kritisierte Einschränkung des RENUMBER-Befehls wird somit gegenstandslos: Simons Basic erlaubt weitgehend eine vollstrukturierte Programmierung mit dem Verzicht auf GOTOs und GOSUBs. Das bedeutet, daß kein Sprung auf eine Programmzeile xyz mehr nötig ist. Sprungadressen erhalten einen Namen und auch Prozeduren (Unterprogramme) werden mit einem Namen aufgerufen.

Aber unabhängig davon, ob Sie mit oder ohne Simons Basic arbeiten, einige Regeln sollte jeder befolgen.

Grundregel: Der Code soll einfach, klar und übersichtlich (nachvollziehbar) sein.
Dazu gehören:

☐ Die Verwendung einfacher sprachlicher Mittel. Stehen zur Formulierung einer Aktion verschiedene sprachliche Mittel zur Verfügung, sollte das einfache gewählt werden. Das bedeutet: Verzicht auf undurchsichtige Programmierung!

☐ Das Einrücken von Befehlsfolgen zur Verdeutlichung von Programmzusammenhängen bei geschachtelten Ablaufstrukturen soweit es möglich ist. Anweisungen gleicher Schachtelungstiefe sollten direkt untereinander geschrieben werden (Bild 1.)

☐ Einfügen von Trennlinien zur optischen Trennung von in sich abgeschlossenen Komponenten. Die sollte man vor allem bei langen Programmen vorsehen. Sie

```
100 REM ********************************
110 REM * PROGRAMM
120 REM * EINGABE,SORTIEREN + AUSGABE *
130 REM ********************************
140 :
145 K=50          :REM ANZAHL DATEN
150 DIM FF$(K+1)
155 :
160 GOSUB 1000    :REM BESETZEN FF$()
165 :
170 OPEN1,4:CMD1  :REM DRUCKER
180 GOSUB 3000    :REM AUSGABE DRUCKER
190 PRINT#1:CLOSE1
195 :
200 GOSUB 2000    :REM SORTIEREN
205 :
210 GOSUB3000:REM AUSGABE BILDSCH.
220 END
230 :
240 :
250 :
260 :
270 :
1000 REM ------------------
1010 REM -          SUBROUTINE          -
1020 REM - BESETZEN FF$() MIT BUCHST.  -
1030 REM ------------------------------
1040 :
1050 :
1060 FOR I=1 TO K
1070 :  FF$(I)=CHR$(INT(RND(0)*26)+65)
1080 NEXT I
1090 RETURN
1100 :
1110 :
2000 REM ------------------------------
2010 REM -          SUBROUTINE          -
2020 REM - SORTIEREN VON FF$(1 BIS K) -
2030 REM ------------------------------
2040 :
2050 :
2060 FOR J=1 TO K-1
2070 :  FOR L=J+1 TO K
2080 :    IF FF$(J)<=FF$(L) THEN 2120
2090 :    A$    =FF$(J)
2100 :    FF$(J)=FF$(L)
2110 :    FF$(L)=A$
2120 :  NEXT L
2130 NEXT J
2140 RETURN
2150 :
3000 REM ------------------------------
3010 REM -          SUBROUTINE          -
3020 REM - AUSGABE FF$(1 BIS K)       -
3030 REM ------------------------------
3040 :
3050 :
3060 FOR I=1 TO K
3070 :  PRINT FF$(I);
3080 NEXT I
3090 RETURN
3100 :
```

OK

erhöhen die Lesbarkeit beträchtlich (Bild 1).

☐ Verwendung von Kommentar als Ergänzung des Codes im Hinblick auf die Problemstellung, nicht als Beschreibung des Codes. Das ist nicht nur bei Assemblerlistings sinnvoll. Der Kommentar zu:

POKE 53281,0 :REM Speicheradresse 53281 mit 0 besetzen

ist sicherlich nicht so sinnvoll wie

POKE 53281,0 :REM Bildschirmfarbe = Schwarz.

☐ Pro Programmzeile nur eine Anweisung. Sie sollten nicht versuchen, möglichst viele Befehle in eine Programmzeile hineinzupressen, wenn Sie sich keine Sorgen um den verfügbaren Speicherplatz machen brauchen.

☐ Die Größe eines Unterprogramms sollte eine Listingseite nicht überschreiten, ausschließlich der Kommentare.

☐ Unterprogramme sollen nur einen Eingang und nur einen Ausgang haben. Und das möglichst am physikalischen Anfang beziehungsweise Ende des Unterprogramms.

☐ Aufgerufene Unterprogramme müssen zum Aufrufpunkt zurückkehren. Verlassen Sie kein Unterprogramm mit GOTO! (Es sei denn zum Abbruch des Programms.)

☐ Benutzen Sie nie eine Variable für mehr als einen Zweck! Wenn in einem Teil des Programms zum Beispiel die Variable AL die Bedeutung Alpha für einen Winkel besitzt, aber im anderen Teil die Bedeutung: Alter hat, verwirrt es doch sehr, und Änderungen sind sehr fehleranfällig!

☐ Ändern Sie nie eine Laufvariable innerhalb der Schleife! (In der Anweisung: FOR I=1 TO 100:PRINT I: NEXT I ist »I« die Laufvariable.)

☐ Sprunganweisungen, die mehr als eine Listingseite auseinanderliegen, tragen sehr zur Unübersichtlichkeit bei.

☐ Vermeiden Sie es, mehr als zwei logische Vergleiche in eine Zeile zu setzen. Es ist sehr schwer, solche Zeilen mit einer Anzahl von logischen Verknüpfungen nachzuvollziehen.

If A AND B OR C AND D AND B<C AND D OR A THEN END

Es bereitet nicht nur sehr viel Mühe, solch eine Programmzeile zu verstehen, Sie werden auch Probleme haben, sie in einem Flußdiagramm sinnvoll darzustellen!

Wenn Sie sich an diese Regeln halten, werden Sie auch nach längerer Zeit noch in der Lage sein, Ihre Programme zu bearbeiten oder sie anderen zu erklären. Und auf diese Art erstellte und veröffentlichte Programme geben auch unseren Lesern eine wertvolle Hilfestellung. (gk)

(Fortsetzung folgt)

```
100 K=50:DIMFF$(51)
110 FORI=1TOK:FF$(I)=CHR$(INT(RND(0)*26)+65):NEXT
120 OPEN1,4:CMD1:GOSUB170
130 PRINT#1:CLOSE1
140 FORJ=1TOK-1:FORL=J+1TOK:IFFF$(J)<=FF$(L)THEN160
150 A$=FF$(J):FF$(J)=FF$(L):FF$(L)=A$
160 NEXTL:NEXTJ:GOTO180
170 FORI=1TOK:PRINTFF$(I);:NEXT:RETURN
180 GOSUB170
```

Bild 2. So sollte es nicht gemacht werden. Dieses Programm ist zwar vom Speicherbedarf her um einiges kürzer als das von Bild 1, die Anzahl der Befehle ist jedoch fast identisch! Aber wissen Sie hier sofort, um was es geht?

Bild 1. Der einzige Unterschied zum Bild 2 sind die eingefügten Kommentar-/Leerzeilen und eine andere Aufteilung des Programms. Die Reihenfolge und auch der Algorithmus der ausgeführten Tätigkeiten Besetzen, Sortieren und Ausgabe sind identisch. Aber ist diese Form der Darstellung nicht wesentlich verständlicher und übersichtlicher? Hier Erweiterungen anzufügen oder Teile zu ändern, dürfte keine Schwierigkeiten bereiten. Über diese Art der Aufteilung (in Unterprogramme) berichten wir in einer der nächsten Ausgaben.