Reise durch den C 128 — (Teil 3)

Wieder hat uns — gerade noch rechtzeitig für diese Ausgabe — ein Bericht unseres Korrespondenten erreicht, der derzeit durch den unbekannten Kontinent C 128 reist. Diesmal hat er den Tastaturpuffer entdeckt und untersucht.

Es gibt manches Mal Situationen, in denen Sie Eingaben per Tastatur nicht sofort bearbeiten können. Während eines Programmablaufes beispielsweise oder bei einem Ladevorgang. Lediglich einige wenige Tasten sind immer aktiv: Die STOP-Taste zum Beispiel. Trotzdem lohnt sich der Griff in die Tasten, denn es wird nichts vergessen: Die fraglichen Zeichen sind nur beiseite gelegt, bis sie bearbeitet werden können. Diese Ablage, in der sie warten, nennt man den Tastaturpuffer. Ist das beendet, was unseren C 128 davon abhielt, uns zuzuhören, dann wendet er sich diesem Pufferinhalt zu und arbeitet ihn durch.

Der Puffer befindet sich im Speicherbereich von 842 bis 851 (also $034A bis 0353). Außerdem braucht unser Computer eine Speicherstelle, wo ihm gezeigt wird, wieviele Zeichen er im Puffer findet. Diesen Job hat die Zelle 208 (das ist $D0). Was können wir nun mit dieser Erkenntnis anfangen? Sie werden überrascht sein, welche Möglichkeiten sich uns da auftun!

Lassen Sie uns zunächst einmal sehen, wie wir diesen Puffer nutzen können. Zunächst erschaffen wir den Zustand, daß der C 128 unsere Tastatureingaben nicht beachtet: Wir beschäftigen ihn mit einem Programm. Listigerweise wird das Programm etwas in seinen Tastaturpuffer schreiben:

```
 10 SCNCLR:PRINT:PRINT
 20 POKE842,ASC("?")
 30 POKE843,34
 40 POKE844,ASC("H")
 50 POKE845,ASC("I")
 60 POKE846,ASC("!")
 70 POKE847,34
 80 POKE848,13
 90 POKE208,7
100 END
```

Geben Sie doch mal RUN ein und sehen Sie, was geschieht: Nach einem READY taucht die Zeile

```
?"HI!"
```

auf dem Bildschirm auf und darunter das Wort »HI!«. Weshalb? Weil wir das in den Tastaturpuffer gePOKEt haben (34 ist der ASCII-Code für das Anführungszeichen und 13 der Code für die RETURN-Taste). Der Computer hat unsere Eingaben per Programm so verstanden, als hätten wir sie ihm im Direktmodus vorgelegt: Wir haben somit die Möglichkeit, einen scheinbaren Widerspruch zwanglos zu umgehen, nämlich den programmierten Direktmodus!

Nun müssen wir uns nur noch Gedanken darüber machen, was es im Direktmodus Interessantes gibt, das wir immer schon gerne per Programm erledigt hätten. Ihnen fällt bestimmt viel ein: Programmzeilen in ein bestehendes Programm einzufügen, den Monitor zu benutzen und so weiter.

Übrigens sind wir nicht unbedingt auf die zehn Speicherstellen des Tastaturpuffers festgelegt. Weitere zehn, die sich daran anschließen, können im allgemeinen ebensogut mitbenutzt werden. Sie gehören zur Tabelle der Tabulator-Stopps, die aber — soweit ich das bisher wahrnehmen konnte — lediglich für die Tabulatorsprungfunktion mittels CTRL-I verwendet wird. Wenn man aber noch mehr Speicherplätze benutzt, kann es kritisch werden, denn da liegen einige wichtige Notizen des Computers und ab 896 könnte sogar der Lebensnerv, nämlich die CHRGET-Routine, zerstört werden. Also begnügen wir uns lieber mit 20 Plätzen im Tastaturpuffer.

Im folgenden werden wir uns einige Anwendungen des programmierten Direktmodus ansehen (manchmal spricht man im englischen Sprachraum auch von »dynamic keyboard«, also von der »dynamischen Tastatur«). Die Programmbeispiele sollen als Module aufgebaut sein, die wir dann mittels der im Teil 2 vorgestellten MERGE-Funktion jederzeit in bestehende Programme einbauen können. Zuvor aber soll der Begriff des Programm-Moduls noch etwas erklärt werden.

Programm-Module

Es gibt — besonders als Ergebnis von Bemühungen der Basic-Programmierer — Programme, die wie ein lebender Organismus gewachsen sind: Da ist es — bei längeren Schöpfungen — mitunter schwierig zu ergründen, wie und warum sie überhaupt funktionieren und ebensowenig wie ein Bein oder ein Arm alleine sinnvoll sind, sind es dann Programmteile. Andere Programmsprachen fördern diese — doch auch irgendwie sympathische — Art der Programmlabyrinthe nicht: Da geht's oft streng nach der Reihe. Immer wieder gibt es auch mehr oder weniger erfolgreiche Versuche, Basic-Programme zu strukturieren und auch unser Computer beinhaltet einige Befehle, die dazu beitragen sollen. Ein weiterer Schritt in diese Richtung ist der Aufbau von Programmen aus Bausteinen, den Programm-Modulen.

Die Zielvorstellung wäre ein Programm, das aus lauter solchen Modulen zusammengesetzt wäre (die hätte der erfahrene Programmierer dann alle schon fertig in einer Modulbibliothek auf Diskette vorliegen), die nur noch durch einen Rahmen zusammenzubinden wären. Wie müßte solch ein Modul aussehen und vor allem: Welche Angaben müßte die begleitende Dokumentation enthalten?

Das Modul: Es sollte möglichst allgemein gehalten sein und daher vielseitig verwendbar. Nötige Anpassungen sollten leicht durchführbar sein, weshalb auch der Aufbau des Moduls überschaubar zu halten ist.

Die Dokumentation: Nach mehr oder weniger langer Zeit hat jeder Programmierer vergessen, was er da geschrieben hat. In der Dokumentation muß daher enthalten sein:

1) Was leistet das Modul?
2) Welche Variablen werden wie verwendet?
   a) Variable, die ins Modul hineingegeben werden.
   b) Variable, die aus dem Modul herausgegeben werden.
   c) Variable, die im Modul erzeugt werden und entweder globale oder nur lokale Verwendung finden.
3) Verwendung des Moduls:
   a) Wie wird es in das Bindeprogramm eingefügt und worauf ist dabei eventuell zu achten?
   b) Wie kann das Modul vom Hauptprogramm her in Betrieb genommen werden?

Es gibt sicher noch weitere Punkte, die manchmal Bedeutung haben: Eine spezielle Angabe von Fehlern, die im Modul auftreten können oder einen Hinweis auf andere Module.

Nach all diesen Vorbemerkungen sehen wir uns nun einige Module an, die den programmierten Direktmodus verwenden.

MODUL: Zeilen einfügen

1) Was leistet das Modul?

In ein bestehendes Programm werden durch programmierten Direktmodus zwei neue Zeilen eingefügt, die eine zu-

vor eingegebene Funktion definieren.
2) Verwendung von Variablen:

Die Variable Z1 muß in das Modul eingeführt werden. Z1 ist die Zeilennummer, in der wir den Funktionsstring abzulegen wünschen. Z1+10 enthält die Funktionsdefinition und Z1-10 ist die Zeilennummer, mit der der Neustart (siehe bei Einbindung des Moduls) des Programmes erfolgt.

Zwei weitere Variable und eine Funktion werden im Modul definiert:
F$ = String, welcher die Funktion enthält.
FN F(X) = Funktion, in welcher F$ verwendet wird.
X = Variable der Funktion.

Alle Variablen (und die Funktion) haben globale Bedeutung.
3) Einbindung des Moduls und Anwendung:

Die Einbindung ist an jeder beliebigen Stelle des Programmes möglich, die nicht in einem Unterprogramm oder einer Schleife steht. Zuvor sollte noch ein Z1 definiert sein, das größer als 10 ist und sicherstellt, daß Z1 und Z1+10 freie Zeilennummern sind.

Das Modul kann sowohl im direkten Programmablauf als auch durch GOTO angesteuert werden. Dies ist die Funktionsweise:
— Der Bildschirm wird gelöscht und eine Funktion Y=F(X) abgefragt.
— Nach erneutem Löschen des Bildschirms und Angleichen der Zeichenfarbe an den Hintergrund (hier als Schwarz angenommen) wird in der dritten Bildschirmzeile gedruckt:
(Zeilennummer Z1) F$="eingeg. Funktion"
In der 4.Zeile:
(Zeilennummer Z1+10) DEF FN F(X)=-"-
In Zeile 5 schließlich:
RUN (Zeilennummer Z1 — 10)

Der Cursor wandert in die Home-Position, schreibt drei RETURNs (das ist CHR$(13)) in den Tastaturpuffer und in die Speicherstelle 208 diese Anzahl von drei.

Das Programm endet nun und auf dem Bildschirm erscheint (ebenfalls unsichtbar) READY. Der Cursor steht nun auf Zeile 3. Der Tastaturpuffer wird abgearbeitet, was bedeutet, daß die Inhalte der Bildschirmzeilen 3 und 4 durch die zwei RETURNs übernommen und das RUN-Kommando ausgeführt wird.

Dieser Neustart löscht den Bildschirm und setzt die Zeichenfarbe auf einen sichtbaren Wert (hier auf Weiß).

Nach dieser ausführlichen Funktionsbeschreibung sollen Sie nun auch das Modul eintippen können. Als »ZEILEN EINF MOD« (Listing 1) finden Sie es hier noch zum Ausprobieren mit einer Zeile 1, die der Variablen Z1 den Wert 110 zuordnet:

Mit Hilfe dieses und ähnlicher Module ist es möglich, selbstmodifizierende Programme zu realisieren. Bedenken Sie, daß wir damit bei jedem Durchlauf bis zu 20 neue Programmzeilen übernehmen könnten, daß wir in laufenden Programmen ganze Abschnitte umzuschreiben imstande sind,...

MODUL: Monitoraufruf

Normalerweise ist ein Basic-Programmablauf in dem Moment beendet, in dem der MONITOR-Befehl bearbeitet ist. Dann meldet sich der Monitor mit einer Registeranzeige und wir befinden uns im Direktmodus. Weil wir diesen aber nun per Programm beherrschen, können wir jetzt auch Funktionen des Monitors, wie in diesem Beispiel den Hexdump von Speicherteilen, in Programme einbinden.
1) Was leistet das Modul?

Es erlaubt die Verwendung des Monitor-Kommandos M zur Anzeige von Speicherinhalten.
2) Variable:

Lediglich zwei Stringvariable spielen eine Rolle. Sie werden vor dem Modulaufruf definiert und bezeichnen die erste anzuzeigende Speicherstelle in Hexadezimalform. Dabei ist Z0$ die im Monitor verwendete Bank-Kennziffer, also die vordere Stelle der Hex-Adresse. Beispielsweise ist bei $FD800 für Z0$ »F« zu setzen.

```
1 Z1=110
10 REM ***** MODUL ZEILEN EINFUEGEN *****
20 PRINT CHR$(147) CHR$(17) CHR$(17)
30 PRINT "WELCHE FUNKTION ?"
40 INPUT "Y=F(X)=";F$
50 PRINT CHR$(147) CHR$(17) CHR$(144)
60 PRINT Z1"F$=" CHR$(34)F$ CHR$(34)
70 PRINT Z1+10"DEF FN F(X)="F$
80 PRINT "RUN"Z1-10 CHR$(19);
90 BANK 0: POKE 842,13: POKE 843,13: POKE 84
   4,13: POKE 208,3: END
100 PRINT CHR$(147) CHR$(5): LIST
110 :
120 :
```

Listing 1. »ZEILEN EINF MOD« — Ein Programm-Modul für selbstmodifizierende Programme

```
1 Z1$="1C00": Z0$="0"
10 REM ***** PROGR.DIREKTMODUS : MONITORAUFR
   UF *****
20 PRINT CHR$(147) CHR$(17)
30 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(17
   ) CHR$(17)
40 PRINT "M ";Z0$+Z1$;" ";Z0$+HEX$(DEC(Z1$)+
   DEC("30")) CHR$(17) CHR$(17) CHR$(17) CHR
   $(17)
50 BANK 0: IF PEEK(238)=39 THEN PRINT CHR$(1
   7) CHR$(17)
60 PRINT "X" CHR$(17)
70 PRINT "RUN100"
80 PRINT CHR$(19);
90 BANK 0: POKE 842,13: POKE 843,13: POKE 84
   4,13: POKE 845,13: POKE 208,4: END
95 REM ************************************
   ********
100 LIST 10
110 PRINT CHR$(17)"DAS WARS!"
120 END
```

Listing 2. »MONITOR MOD« — Programm-Modul zur Verwendung der Monitorfunktion M in Basic-Programmen

Z1$ ist die vierstellige Hexzahl, die sich an Z0$ anschließt. Im obigen Beispiel also Z1$="D800".
3) Einbindung und Verwendung des Moduls:

Ebenso wie das vorhin vorgestellte Modul ist auch dieses an jede beliebige Stelle des Programmes zu plazieren, außer in Unterprogramme oder Schleifen.

Es kann direkt im Programmlauf oder durch GOTO aktiviert werden.

Hier nun die Erklärung des Ablaufes, die aber nicht ganz so ausführlich wie beim ersten Beispiel sein wird:

Nach dem Löschen des Bildschirms und dem Überspringen der READY-Zeile werden nacheinander das Monitorkommando, der Monitorbefehl M, der Befehl zum Verlassen des Monitors X und ein RUN 100 auf den Bildschirm gedruckt. Eine bestimmte Anzahl von Leerzeilen ist hier nötig, um die Registeranzeige und die zu druckenden Zeilen der Speicherinhalte zu überspringen. Je nach verwendetem Bildschirm sind das letztere dann 4 Zeilen (80-Zeichen, hier werden 16 Kolonnen ausgegeben) oder 7 Zeilen (40-Zeichen, wobei wir nur 8 Kolonnen erhalten). Welcher Bildschirm aktiv ist, kann der Computer selbst herausfinden, indem er sich den Inhalt der Speicherstelle 238 ($EE) ansieht. Dort findet er die höchste Spaltenziffer: 79 beim 80-Zeichen-Betrieb und 39 im 40-Zeichen-Betrieb. In Programmzeile 50 fügt er im Bedarfsfall noch die nötige Anzahl von Cursor-Down-Kommandos hinzu.

In Zeile 40 wird das M-Kommando gedruckt. Hier berechnet der Computer noch die zweite M-Adresse, die um $30 hö-

```
1 REM ***** PROG. DIREKTMODUS : TRANSFERBEFE
  HL *****
2 COLOR 0,1: COLOR 1,3: COLOR 4,1: GRAPHIC 1
  ,1
3 FOR I=1 TO 30
4 DRAW 1,I,0 TO 10*I,100
5 NEXT I: COLOR 1,6: WIDTH 2: CIRCLE 1,230,5
  0,20,20
6 Z0$="0": Z1$="2000": Z2$="3030": Z3$="0":
  Z4$="2F00": Z=100
10 REM ******** TRANSFER MODUL ********
20 PRINT CHR$(147) CHR$(17)
30 PRINT "MONITOR" CHR$(17) CHR$(17) CHR$(17
   ) CHR$(17)
40 PRINT "T ";Z0$+Z1$;" ";Z0$+Z2$;" ";Z3$+Z4
   $;CHR$(17)
50 PRINT "X" CHR$(17)
60 PRINT "RUN "Z
70 PRINT CHR$(19);
80 BANK 0: POKE 842,13: POKE 843,13: POKE 84
   4,13: POKE 845,13: POKE 208,4: END
90 REM ***** WEITER MIT BASIC *****
100 LIST 10
110 PRINT CHR$(17)"DAS WARS!"
120 END
```

Listing 3. »TRANSFER MOD« — Ein Modul zum Verschieben von Speicherbereichen

her als die Startadresse gewählt wurde, um den gesamten Ausdruck auf den 40-Zeichen-Bildschirm bringen zu können. Falls Sie für eigene Anwendungen einmal einen anderen Wert als $30 benötigen, müssen Sie unter Umständen noch die Anzahl der CHR$(17)-Kommandos verändern.

In Zeile 70 kann anstelle der Zahl 100 auch eine Variable eingefügt werden, die vor dem Modulaufruf zu definieren ist. Dadurch kann die weitere Verarbeitung nach dem Modulablauf noch flexibler gestaltet sein. Im hier abgedruckten Programm »MONITOR MOD« (Listing 2) sind — um es als Beispiel lauffähig zu machen — noch vier Zeilen hinzugefügt worden: Zeile 1, welche die Stringvariablen definiert und die Zeilen 100 bis 120. Die Voreinstellungen zeigen dann im Programmablauf den Anfang des Programmes als Hex-Listing.

MODUL: Transferbefehl

Im Monitor existiert ein besonders starker Befehl, das T-Kommando. Damit können beliebig große Speicherbereiche verschoben werden. Zwar gibt es auch in Basic 7.0 drei Befehle, die das können, nämlich STASH, FETCH und SWAP. Leider aber sind diese Befehle nicht geeignet, Verschiebungen innerhalb der BANKs 0 und 1 oder der beiden untereinander vorzunehmen. Sie beziehen sich auf höhere BANKs, die erst mit den Speichererweiterungen erreichbar sind.

Damit hat es nun ein Ende. Falls Sie einmal beispielsweise eine Bitmap aus BANK 0 nach BANK 1 verschieben möchten, können Sie das mit diesem Modul erledigen.

1) Was leistet das Modul?

Beliebige Speicherbereiche werden an beliebige Zieladressen kopiert.

2) Variable:

Insgesamt spielen sechs Variable eine Rolle, die vor dem Modul-Aufruf definiert sein müssen:

Z0$ bis Z4$ sind Stringvariable, die die Adressen für den Transferbefehl enthalten, in Hexadezimalzahlen. Folgende Zuordnung ergibt sich aus dem Transferbeispiel:

```
T 02000 03030 12F00
```

Das bedeutet, daß der Speicherbereich zwischen $02000 und $03030 nach oben verschoben (in Wirklichkeit:kopiert) wird — und zwar aus der BANK 0 in die BANK 1 — ab $12F00 und folgende

Z0$ = "0" BANK, aus der verschoben wird.
Z1$ = "2000" Quelle Startadresse.
Z2$ = "3030" Quelle Endadresse.
Z3$ = "1" BANK, in die hineinverschoben wird.
Z4$ = "2F00" Ziel Startadresse.

Eine weitere Variable ist Z:
Z ist die Zeilennummer, von der an das Programm sinnvollerweise neu gestartet wird.

3) Einbindung und Verwendung:

Das Modul kann an beliebiger Stelle eines Programmes eingesetzt werden, nur nicht in Unterprogrammen oder Schleifen.

Im abgedruckten Listing »TRANSFER MOD« (Listing 3) sind vor das Modul zur Demonstration noch einige Grafikbefehle und die Definition der Variablen gehängt (Zeilen 1 bis 6). Ab Zeile 90 beginnt wieder das Hauptprogramm.

Die Demonstration zeichnet einige Dinge auf die obere Hälfte des Grafik-Bildschirmes und kopiert sie dann durch das Transfer-Modul in die untere Hälfte. Auf diese Weise ist auch zu erkennen, daß es sich hier um ein KOPIEREN, nicht um ein wirkliches VERSCHIEBEN handelt, denn das Bild auf der oberen Hälfte bleibt ja erhalten.

Das Demonstrationsprogramm ist für den Betrieb mit zwei Bildschirmen geschrieben. Sollten Sie lediglich mit dem 40-Zeichen-Monitor arbeiten, dann sollten Sie in Zeile 95 noch den Befehl GRAPHIC 0 einfügen.

Ein Problem gibt es noch: Nach jedem RUN sind bekanntlich immer alle Variablen eines Programmes gelöscht. Häufig stört das nicht weiter, weil ohnehin die Modifikation zu Beginn eines Programmablaufes eingebaut wird oder man durch ein GOSUB in die Zeile mit den Variablendefinitionen schnell wieder voreingestellte Variable zurückholen kann. Manchmal — besonders, wenn man ohne an die besondere Eigenart der Module zu denken, diese irgendwo in einem Programm verwendet — kann es aber schon zur bösen Überraschung werden, plötzlich ohne alle Variablen dazustehen. Nichts hindert uns dann aber, statt durch RUN, das Programm durch GOTO neu anlaufen zu lassen. Ersetzen Sie in solchen Fällen also einfach die Zeile:

```
70 PRINT"RUN"Z
```

(oder ähnliche)
durch:

```
70 PRINT"GOTO"Z
```

Die Variablen sind dann alle noch präsent, ja man kann nun auch die Module innerhalb von Schleifen aufrufen!

2D-Funktionen: Ein Programm mit Modulen

Als Beispiel für ein Programm mit solchen Modulen ist nachstehend das Listing »2D-FUNKTIONEN« (Listing 4) abgedruckt:

Es erlaubt Ihnen — innerhalb gewisser Grenzen, damit das Listing nicht zu umfangreich wird — die grafische Darstellung beliebiger zweidimensionaler Funktionen. Ein weiteres Modul wird hier angewendet, das Modul »TRANSFORM MOD« (Listing 5):

Hier ist seine Beschreibung:

1) Was leistet das Modul?

Es erfragt vom Benutzer die Grenzwerte eines Koordinatensystems und erzeugt zwei Funktionen, die die Transformation beliebiger Punkte des angegebenen Systems in Bildschirmkoordinaten vornehmen können.

2) Variable:

Alle benötigten Variablen werden im Modul erzeugt:
XU,XO — kleinste und größte X-Koordinate
YU,YO — dasselbe für die Y-Koordinaten des gewünschten Systems.
SX,SY — interne Variable. Das sind die Skalierungsfaktoren in X- und in Y-Richtung.
TX,TY — ebenfalls interne Variable. Hier dreht es sich um die Translation in X- und in Y-Richtung.

Um diese vier internen Variablen braucht man sich normalerweise nicht zu kümmern: Sie werden automatisch erzeugt und verwendet durch die beiden Funktionen:
FN TX(X) — Transformiert eingegebene Koordinaten des gewählten Systems (X) in Bildschirm-Koordinaten um,
FN TY(Y) — leistet dasselbe für die Y-Richtung.

```
10 CLR : TRAP 650
20 REM ************************************
   ************************
30 REM *
                              *
40 REM *       GRAFISCHE DARSTELLUNG BELIEBI
   GER 2D-FUNKTIONEN          *
50 REM *
                              *
60 REM *              HEIMO PONNATH  HAMBURG
    1985                      *
70 REM *
                              *
80 REM ************************************
   ************************
90 COLOR 0,1: COLOR 1,8: COLOR 4,1: COLOR 5,
   2: COLOR 6,1: Z1=250
95 GRAPHIC 1,1: GRAPHIC 5,1
100 BANK 15: SYS 65520,,10,15: PRINT "GRAFIS
    CHE DARSTELLUNG BELIEBIGER 2D-FUNKTIONEN
    "
110 BANK 15: SYS 65520,,15,10: PRINT "DIESE
    FUNKTION IST PROGRAMMIERT:": K=1: GOSUB
    250
120 PRINT : PRINT "Y = F(X) ="F$: PRINT : PR
    INT ,"SOLLS EINE ANDERE SEIN (J/N) ?";
130 GET A$: IF A$<>"J" AND A$<>"N" THEN 130
140 IF A$="N" THEN 240
150 FOR I=1 TO 18: PRINT CHR$(27)+"Y": NEXT
    I: BANK 15: SYS 65520,,10,10
160 REM ***** MODUL ZEILEN EINFUEGEN *****
170 PRINT "WELCHE FUNKTION ?": PRINT
180 INPUT "Y = F(X) =";F$: FAST
190 PRINT CHR$(147) CHR$(17) CHR$(144)
200 PRINT Z1"F$=" CHR$(34)F$ CHR$(34)
210 PRINT Z1+10"DEF FN F(X)="F$
220 PRINT "RUN"Z1-10 CHR$(19);
230 BANK 0: POKE 842,13: POKE 843,13: POKE 8
    44,13: POKE 208,3: END
240 PRINT CHR$(147) CHR$(5): SLOW : K=0
250 F$="EXP(COS(1/X))"
260 DEF FN F(X)=EXP(COS(1/X))
270 IF K=1 THEN RETURN
280 REM ***** MODUL TRANSFORMATION *****
290 PRINT CHR$(17) CHR$(17)"SYSTEMGRENZWERTE
    :" CHR$(17)
300 INPUT "XU,XO,YU,YO=";XU,XO,YU,YO
310 SX=319/(XO-XU): SY=-199/(YO-YU): TX=-XU*
    SX: TY=-YO*SY
320 DEF FN TX(X)=SX*X+TX
330 DEF FN TY(Y)=SY*Y+TY
340 REM ***** ZEICHNEN DES KOORDINATENSYSTEM
    S *****
350 GRAPHIC 1: COLOR 1,12
360 IF (XO-XU)>30 THEN 420: ELSE BEGIN
370 : FOR X=INT(XU) TO INT(XO)
380 : : IF X=0 THEN 400
390 : : DRAW 1,FN TX(X),FN TY(YU) TO FN TX(X
    ),FN TY(YO)
400 : NEXT X
410 BEND
420 IF (YO-YU)>30 THEN 480: ELSE BEGIN
430 : FOR Y=INT(YU) TO INT(YO)
440 : : IF Y=0 THEN 460
450 : : DRAW 1,FN TX(XU),FN TY(Y) TO FN TX(X
    O),FN TY(Y)
460 : NEXT Y
470 BEND
480 COLOR 1,3: WIDTH 2
490 DRAW 1,FN TX(XU),FN TY(0) TO FN TX(XO),F
    N TY(0)
500 DRAW 1,FN TX(0),FN TY(YU) TO FN TX(0),FN
     TY(YO)
510 WIDTH 1
520 REM ***** ZEICHNEN DER FUNKTION *****
530 COLOR 1,6: LOCATE FN TX(XU),FN TY(FN F(X
    U))
540 FOR X=XU TO XO STEP 1/SX
550 : Y=FN F(X)
560 : IF FN TY(Y)<0 OR FN TY(Y)>199 THEN 610
570 : DRAW  TO FN TX(X),FN TY(Y)
580 NEXT X
590 CHAR 1,0,0,"Y="+F$,1
600 END
610 IF FN TY(Y)>199 THEN LOCATE FN TX(X+1/SX
    ),FN TY(YU): ELSE BEGIN
620 : LOCATE FN TX(X+1/SX),FN TY(YO)
630 BEND
640 GOTO 580
650 REM ***** FEHLERBEHANDLUNG *****
660 IF ER=14 THEN RESUME 580
```

Listing 4. »2D-FUNKTIONEN« — Programm zur grafischen Darstellung von zweidimensionalen Funktionen

```
10 REM ***** MODUL TRANSFORMATION *****
20 PRINT CHR$(147) CHR$(17)"SYSTEMGRENZWERTE
   :"
30 INPUT "XU,XO,YU,YO=";XU,XO,YU,YO
40 SX=319/(XO-XU): SY=-199/(YO-YU): TX=-XU*S
   X: TY=-YO*SY
50 DEF FN TX(X)=SX*X+TX
60 DEF FN TY(Y)=SY*Y+TY
```

Listing 5. »TRANSFORM MOD« — Programm-Modul zur Transformation beliebiger Koordinatensysteme in das Bildschirmsystem

3) Einbau und Verwendung des Moduls:

Das Modul ist an beliebiger Stelle in Programme einzusetzen, muß aber vor der Verwendung der Funktionen im Programmlauf angesteuert werden, weil sonst ein UNDEFN'D FUNCTION ERROR auftritt.

Es kann auf beliebige Weise verwendet werden.

Eine Erklärung der mathematischen Grundlagen an dieser Stelle wäre etwas umfangreich. Falls Sie daran interessiert sind, lesen Sie bitte in der Serie »Grafik-Streifzüge« in der Zeitschrift 64'er nach, wo alles Wissenswerte über Transformationen auf einfache Weise erklärt wird. (Sie lernen dort auch, wie sich mit einfachen Mitteln Rotationen ins Modul einbauen lassen.)

Sehen wir uns nun nochmal das Programm »2D-FUNKTIONEN« an. Es ist gewissermaßen die Sparausführung eines solchen Grafikprogrammes. So kann man beispielsweise nicht bestimmte Bereiche beim Zeichnen ausklammern. Es wird immer von XU bis XO gezeichnet. Auch ist es nicht möglich, sowohl XU als auch XO als positive Werte einzugeben, ebensowenig, wie es möglich ist, YU und YO beide negativ anzugeben. Jedesmal müssen der untere und der obere Wert verschiedene Vorzeichen haben (aber auch noch die Null als höchster oder niedrigster Wert wird akzeptiert). Das Programm wurde für den Betrieb mit zwei Bildschirmen geschrieben. Sollten Sie lediglich den 40-Zeichen-Schirm verwenden, müssen Sie nur in Zeile 95 statt GRAPHIC5,1 nun GRAPHIC0,1 schreiben und in Zeile 600 eine Zurückschaltung in den Grafik-Modus 0 veranlassen. Beim Zeichnen werden Sie bemerken, daß sich ein Teil des Koordinaten-Rasters verfärbt. Das liegt daran, daß unser Programm mehrere Farben verwendet und nicht im Multicolormodus läuft, um keine Einbußen bei der Auflösung hinnehmen zu müssen. Weil aber die Farbgebung immer in 8 x 8 Bit-Feldern geschieht, wird eine alte Zeichnung immer dann neu gefärbt, wenn eine neue Linie in anderer Farbe durch dieses Feld läuft. Viel Spaß wünsche ich Ihnen beim Ausbauen dieser Sparversion zum professionellen 2D-Funktionen-Programm.

Damit endet dieser Bericht unseres Korrespondenten. Es ist schon interessant, wozu der Tastaturpuffer so alles taugt, nicht wahr? Wir warten mit Spannung auf das nächste Lebenszeichen aus dem unbekannten Kontinent C 128.

(Heimo Ponnath/dm)