Strubs — ein Precompiler für Basic

In der letzten Ausgabe haben Sie den Unterschied zwischen einem Compiler und einem Interpreter erfahren und sich kurz über die Vorteile von Strubs informieren können. Hier nun werden die in Strubs implementierten Befehlsstrukturen erläutert und das Objektprogramm von Strubs selbst vorgestellt.

An anderer Stelle in dieser Zeitschrift (oder auch in den unten aufgeführten Büchern) können Sie sich ausführlich über die Grundlagen der strukturierten Programmierung informieren. Aus diesem Grund werden wir uns hier darauf beschränken, einige Aspekte kurz anzusprechen und im übrigen vorzustellen, was Strubs in dieser Hinsicht zu bieten hat.

Gehören Sie auch zu denjenigen, die sich manchmal ein Programm aus einer Zeitschrift vornehmen, um zu analysieren, wie es arbeitet oder um eventuell Teile des Programms für eigene Programmprojekte zu verwenden? Dann erinnern Sie sich bestimmt an Programme, bei denen Sie sich verzweifelt von Sprung zu Sprung bewegen und nach nicht allzu langer Zeit vollkommen den Überblick verlieren. Oder vielleicht kennen Sie folgende Situation: Sie schreiben ein Programm und erinnern sich angesichts eines bestimmten Problems, daß Sie ein ganz ähnliches Problem schon einmal in einem anderen Programm gelöst haben. Aber sobald Sie sich den alten Programmtext vornehmen, um den entsprechenden Programmteil in ihr neues Programm zu übernehmen, müssen Sie enttäuscht feststellen, daß diese spezielle Problemlösung so sehr in das Programmgeflecht verwoben ist, daß es Ihnen weitaus einfacher scheint, den entsprechenden Programmteil vollkommen neu zu entwickeln.

Die Ursache für solche Erscheinungen liegt zum Teil darin, daß viele Basic-Programme mehr oder weniger aus der Sicht des Computer der »Basic-Maschine« — direkt am Computer nach dem Verfahren von Versuch und Irrtum entwickelt werden. Das kann in Einzelfällen sogar soweit führen, daß man zum Schluß zwar sieht, daß das Programm läuft, aber selbst nicht so recht weiß, warum eigentlich und wie es funktioniert. Der Hauptgrund für solche Unübersichtlichkeit aber liegt in der Verwendung zahlreicher wilder Sprünge und ausgefallener Programmier-Tricks. (Daß die Verwendung von GOTO-Anweisungen den mathematischen Beweis für die Korrektheit von Programmen praktisch unmöglich macht, ist für den Informatiker interessant, braucht uns hier aber nicht zu interessieren).

Den entgegengesetzten Weg geht die strukturierte Programmierung. Sie bedeutet vor allem sorgfältige Planung und den Verzicht auf GOTOs und unübersichtliche Programmiertricks. Hier steht die systematische Analyse des Problems im Vordergrund. Die eigentliche Codierung, das heißt die Formulierung des Programmtextes in einer bestimmten Programmiersprache, spielt nur eine untergeordnete Rolle.

In der Problemanalyse geht es darum, ein gegebenes Problem in relativ selbständige Teilprobleme zu zerlegen und deren Beziehungen zueinander festzulegen. Den Aufbau des Programms Strubs mit den jeweiligen Zeilennummern können Sie Bild 1 entnehmen. Das komplette Objektprogramm ist ebenfalls abgedruckt (siehe Listing).

Entsprechend setzt sich das strukturierte Programm aus einer Reihe möglichst selbständiger Programmeinheiten zusammen. Dieses Vorgehen spiegelt sich im Konzept der Blöcke und Module.

Ein Block ist eine Anweisung oder eine Folge von Anweisungen mit genau einem Eingang und genau einem Ausgang. Das heißt man darf weder in einen solchen Block hineinspringen, noch aus diesem Block herausspringen. Solche Blöcke können entweder aneinander gereiht oder beliebig tief ineinander geschachtelt werden; sie dürfen sich

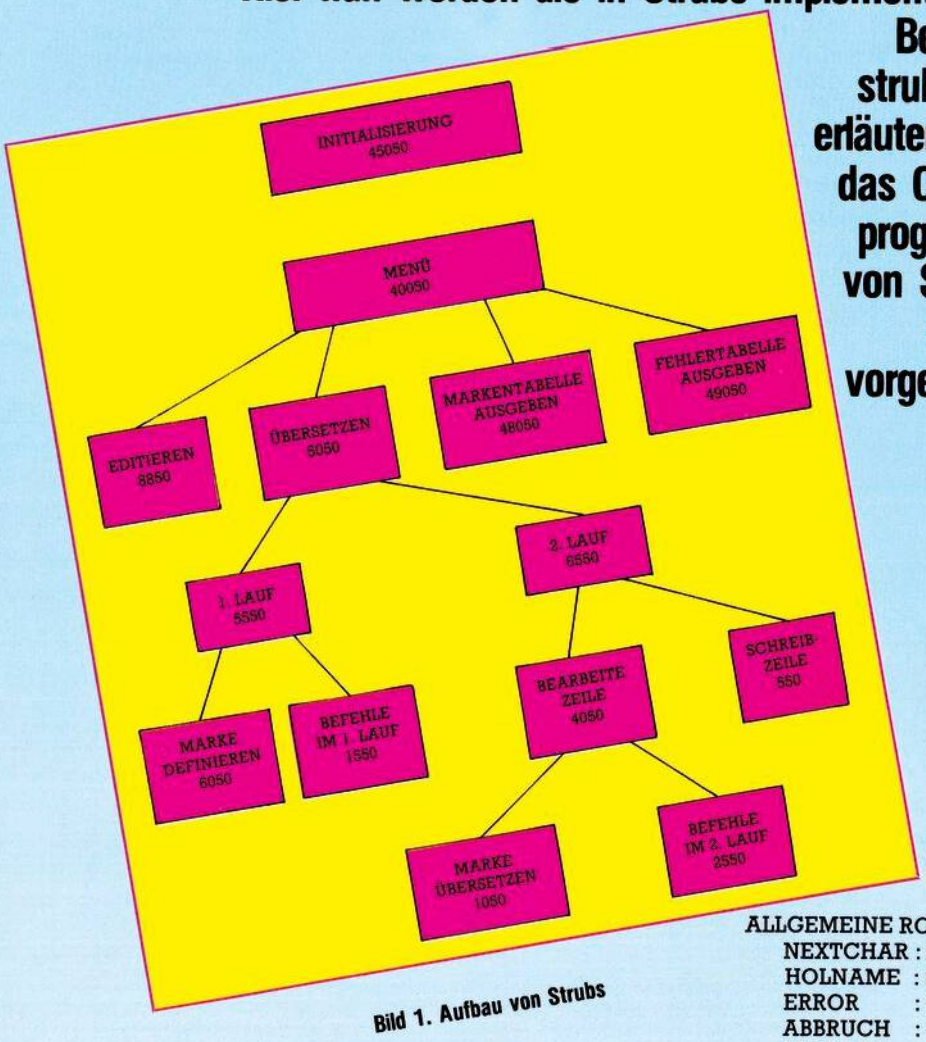

Bild 1. Aufbau von Strubs

ALLGEMEINE ROUTINEN:

NEXTCHAR	: 00250
HOLNAME	: 00750
ERROR	: 08050
ABBRUCH	: 50000
WARTEN	: 49550

-Programme (Teil 2)

aber nicht überschneiden. In letzterer Hinsicht verhält es sich mit diesen Blöcken also genauso, wie bei den bekannten FOR-Schleifen in Basic.

Ein strukturiertes Programm besteht nun ausschließlich aus einer geordneten Hierarchie solcher Blöcke. Der kleinste mögliche Block besteht aus einer einzelnen Anweisung, wie zum Beispiel PRINT "Text". Der größte, umfassendste Block besteht aus dem Programm selbst.

Da ist zunächst einmal die einfache IF-Anweisung, die schon von Basic her bekannt ist. Dieses normale Basic-IF kann natürlich wie alle Basic-Befehle weiterhin benutzt werden. Zusätzlich bietet Strubs aber eine erweiterte Form, bei welcher der THEN-Teil nicht auf den Rest einer Programmzeile begrenzt ist, sondern beliebig viele Zeilen umfassen kann, die durch den Befehl '!FI' — einfach ein umgedrehtes IF — abgeschlossen werden. Ein Beispiel:

```
10 ! IF X=Y THEN
20 :    PRINT "X und Y"
30 :    PRINT " SIND GLEICH"
...
99 !FI
```

Ist die Bedingung hinter IF erfüllt, so werden die Zeilen zwischen der IF- und der FI-Anweisung ausgeführt, ansonsten wird das Programm sofort hinter der FI-Zeile fortgesetzt.

Daneben existiert selbstverständlich auch die vollständige Form

```
10 !IF X=Y THEN
20:    PRINT "GLEICH'
...
50 !ELSE
60 :    PRINT "UNGLEICH"
...
99 !FI
```

Ist die Bedingung erfüllt, dann wird der Block zwischen IF und ELSE ausgeführt, sonst der Block zwischen ELSE und FI.

Für den Fall, daß mehr als nur zwei Fälle zu unterscheiden sind, bietet Strubs die CASE-Anweisung:

Bild 3. Struktogramm der Loop-Schleife

```
510 '*********************************
520 '* GESCHACHTELTE LOOP-BLOECKE    *
530 '*********************************
540 '
620 !LOOP 'L1
630 :    PRINT"AEUSSERE LOOP1"
640 :    IF X=1 THEN !EXIT 'LOOP1
650 :    !LOOP 'L2
660 :       PRINT "INNERE LOOP2"
670 :       IF X=0 THEN !EXIT 'LOOP2
680 :    !ELOOP ' L2
690 '    HIER WIRD PROGR. NACH EXIT LOOP2 FORTGESETZT
700 :    X=X+1
710 :    !IF X=2 THEN
720 :       PRINT "LOOP 1 VERLASSEN:":!EXIT 'LOOP1
730 :    !FI
740 :    X=X+1
750 !ELOOP 'L1
760 PRINT"HIER WIRD PROGRAMM NACH EXIT LOOP 1 FORTGESETZT"
770 '
READY.
```

Bild 2. Geschachtelte Loop-Schleife

```
10 !CASEOF X<0 THEN
15 :    PRINT "KLEINER ALS 0"
...
40 ! OF X=0 THEN
45 :    PRINT "GLEICH 0"
...
60 ! OF X>0 AND Y <XTHEN
65 : PRINT "X>0 UND Y<X"
...
80 ! ELSE
85 :    PRINT "KEINER DER FÄLLE TRIFFT ZU"
99 ! ECASE
```

Mit dieser Struktur können beliebig viele Fälle unterschieden werden, wobei jedes OF mit einer beliebigen Bedingung verbunden werden kann. Es sollte aber darauf geachtet werden, daß sich die Bedingungen gegenseitig ausschließen (sonst wird das erste Auftreten einer erfüllten Bedingung gewählt). Nach der Bearbeitung des entsprechenden Falles wird das Programm immer hinter ECASE fortgesetzt. Die Möglichkeit, daß keiner der Fälle zutrifft, kann mit Hilfe der ELSE-Anweisung behandelt werden. Ist dies nicht erforderlich, kann der ELSE-Teil auch entfallen.

Damit kommen wir nun zu den Schleifen. Die FOR-Schleife kann wie bisher benutzt werden. Die WHILE-Schleife wird durchlaufen, solange die Bedingung erfüllt ist. Anschließend wird das Programm hinter EWHILE fortgesetzt. Da die Bedingung am Anfang der Schleife abgefragt wird, kann es vorkommen, daß die Schleife auch überhaupt nicht durchlaufen wird. Ein Beispiel:

```
10 ! WHILE X<5 !DO
20 :  PRINT "IMMER NOCH KLEINER ALS 5"
30 : X = X + 1
...
99 !EWHILE
```

```
51 PRINT"■";"  ****************"
52 PRINT "  * --STRUBS.4 -- *"
55 PRINT "  *   M.TOERK     *"
57 PRINT "  * 4352 HERTEN   *"
58 PRINT "  ****************"
70 IFNOT(PEEK(46)<46ORPEEK(46)=46ANDPEEK(45)<3))THEN75
73 POKE46,46:POKE45,3:POKE46*256,0:CLR
75 :
80 EA=46*256+1

READY.

8860 PRINT"■■■■■■"
8870 PRINT "********************"
8880 PRINT "** ZURUECK MIT:    **"
8882 PRINT "** ' ! ' [RETURN]  **"
8940 PRINT "********************"

READY.
```

Dies Änderung sind für die Anpassung von Strubs an den VC 20 (mit mindestens 16 KByte Erweiterung) erforderlich.

```
STRUBS.4 OBJEKTPROGRAMM:

5 REMSTRUBS4/4.9.83
51 PRINT"J";TAB(10);"*****************"
52 PRINTTAB(10);"* --STRUBS.4 -- *"
55 PRINTTAB(10);"*   M.TOERK     *"
57 PRINTTAB(10);"* 4352 HERTEN   *"
58 PRINTTAB(10);"*****************"
70 IFNOT(PEEK(46)<400R(PEEK(46)=40ANDPEEK(45)<3))THEN75
73 POKE46,40:POKE45,3:POKE40*256,0:CLR
75 :
80 EA=40*256+1
100 GOSUB45060
140 GOTO40050
250 IFPEEK(NC)=BLTHENNC=NC+1:GOTO250
260 C=PEEK(NC)
265 IFC<>KOTHEN320
280 NC=NC+1:C=PEEK(NC):IFCANDC<>KOTHEN280
290 IFCTHENNC=NC+1:C=PEEK(NC)
295 IFC=BLTHEN250
320 IFC<>TETHENNC=NC+1:RETURN
350 Z$=Z$+CHR$(C):NC=NC+1:C=PEEK(NC):IFCANDC<>TETHEN350
370 NC=NC+1
390 RETURN
550 IFLEN(Z$)<4THENRETURN
555 PRINTFNAD(ZA+2)
560 AA=AA+LEN(Z$)+2
565 H%=AA/256
570 PRINT#1,CHR$(AA-256*H%);CHR$(H%);Z$;
580 RETURN
750 T$=""
790 :
795 C=PEEK(NC):IFC=DPORC=KMORC=BLORC=0THEN811
800 NC=NC+1:T$=T$+CHR$(C)
810 GOTO790
811 :
820 NC=NC+1:IFC=BLTHENGOSUB250
830 RETURN
1050 GOSUB750
1120 IFNOT(T$="THIS")THEN1131
1125 H=FNAD(ZA+2)
1130 GOTO1175
1131 :
1140 FORI=0TOMP:IFMA$(I)<>T$THENNEXT
1160 IFI>MPTHENER=2:GOTO8050:
1170 H=MA%(I)+DI
1175 :
1180 Z$=Z$+MID$(STR$(H),2)
1190 RETURN
1550 GOSUB750
1560 FORI=0TOBM:IFT$<>BE$(I)THENNEXT
1565 IFI>BMTHENER=0:GOTO8050
1567 B$=BE$(I):IFI=3THENB$="IF"
1569 I=I+1
1570 ONIGOSUB1600,1680,1640,2010,2040,2100,2160,2210,2260,
2400,1710,1740,1810,1860
1574 PRINTFNAD(ZA+2);
1575 IFIN=0THENPRINTTAB(TA);B$:RETURN
1577 IFIN=1THENPRINTTAB(TA);B$:TA=TA+1:RETURN
1579 IFIN=2THENPRINTTAB(TA-1);B$:RETURN
1581 IFIN=3THENTA=TA-1:PRINTTAB(TA);B$:RETURN
1586 RETURN
1600 IFSP>SMTHENER=3:GOTO50000
1605 IFLP>LMTHENER=5:GOTO50000
1610 S%(SP)=LP:SP=SP+1:LO%(LP,0)=FNAD(ZA+2)-DI:LP=LP+1
1615 IN=1:RETURN
1640 SP=SP-1:IFSP<0THENER=1:GOTO50000
1650 LO%(S%(SP),1)=FNAD(ZA+2)-DI
1660 IN=3:RETURN
1680 IN=0:RETURN
```

Listing. Das Objektprogramm Strubs

Von der WHILE-Schleife unterscheidet sich die REPEAT-Schleife in zwei Punkten: Erstens wird die Schleife durchlaufen, bis die Bedingung erfüllt ist, also solange sie nicht erfüllt ist. Zweitens wird die Bedingung erst am Ende der Schleife abgefragt, so daß die Schleife immer mindestens einmal durchlaufen wird. In diesem wie im nächsten Beispiel bezieht sich die Zeile 30 auf den Fall, daß X beim Eintritt in die Schleife größer als 5 ist:

```
10 ! REPEAT
20 :  PRINT "X KLEINER ALS 5"
30 : PRINT "VIELLEICHT ABER
AUCH NICHT"
40 : X = X+1
...
99 ! UNTIL X >= 5
```

Eine weniger weit verbreitete, aber sehr mächtige Schleifenstruktur stellt die LOOP-Schleife dar (sie befindet sich zum Beispiel in der Programmiersprache ADA):

```
10 ! LOOP
30 :  PRINT "EVENTUELL GROESSER ALS 5"
40 :  IF X>=5 THEN !EXIT
50 :  PRINT "KLEINER ALS 5"
60 :  X = X+1
...
99 !ELOOP
```

Verlassen einer Endlosschleife

Es handelt sich dabei um eine Endlosschleife, welche mit Hilfe des Befehls EXIT verlassen werden kann. Diese Schleife bietet im wesentlichen zwei Vorteile: Zum einen muß die Bedingung nicht entweder am Anfang oder am Ende der Schleife stehen, sondern kann an jeder beliebigen Stelle innerhalb des Blockes abgefragt werden. Darüber hinaus ist das Beenden der Schleife nicht nur von einer Bedingung abhängig, sondern die LOOP-Schleife kann beliebig viele EXIT-Anweisungen enthalten (dadurch wird nicht die oben erwähnte Forderung nach nur einem Ausgang verletzt, da das Programm in allen Fällen hinter dem ELOOP fortgesetzt wird). Damit eignet sich diese Konstruktion insbesondere gut für die Behandlung von Ausnahmen wie zum Beispiel von Eingabebefehlen etc. (eine Angelegenheit, die zum Beispiel in Pascal recht umständlich sein kann, falls man auf GOTOs verzichten will oder muß).

In Bild 2 (das Zeichen ' kennzeichnet Kommentare) sehen Sie ein Beispiel für geschachtelte LOOP-

Schleifen. Die Ausführung einer EXIT-Anweisung bewirkt die Fortsetzung des Programms bei der ersten Zeile hinter derjenigen Schleife, welche diese EXIT-Anweisung am nächsten umschließt. Im Beispiel enthält die äußere Schleife zwei EXIT-Anweisungen — eine davon vor, die andere hinter der inneren Schleife. Die innere Schleife enthält eine EXIT-Anweisung. Grafisch lassen sich blockstrukturierte Programme am besten durch Struktogramme — anstelle der verbreiteten Flußdiagramme — darstellen. Das Struktogramm für die LOOP-Schleifen finden Sie in Bild 3. Über die Diagramme der anderen Strukturen und den Umgang mit Struktogrammen können Sie sich an anderer Stelle in dieser Zeitschrift oder in den unten aufgeführten Büchern informieren. Kommen wir nun zu den Modulen. Dabei handelt es sich um besondere Blöcke, die ein bestimmtes Teilproblem — beispielsweise das Zeichnen einer Linie in einem Grafikprogramm — unter möglichst weitgehender Unabhängigkeit vom restlichen Programmtext bearbeiten. Stellen Sie sich vor, Sie finden in einer Zeitschrift ein Pascal-Programm zur Einstellung von Grafiken. Dieses Programm benutzt zum Beispiel die Anweisung PLOT (X,Y) zum Zeichnen eines Punktes mit den Koordinaten X und Y. Ihr Freund möge eine Sprache Super-Pascal besitzen, die diese Anweisung standardmäßig enthält. Er tippt das Programm ein, es läuft — fertig. Sie selbst besitzen aber nur ein mageres Mini-Pascal, das diesen Befehl nicht kennt. Nun, mit Pascal ist das kein Problem: Sie schreiben sich eine Procedur PLOT (X,Y) fügen diese in das Programm ein — fertig. An dem Programmtext selbst brauchen Sie nicht die geringste Änderung vorzunehmen. Ja, brauchen ihn nicht einmal näher anzusehen. Woran liegt das?

Vom Problem her — dem Erstellen einer Grafik — ist das Zeichnen eines Punktes das Zeichnen eines Punktes. Das einzige, was interessiert, ist, daß dazu zwei Koordinaten erforderlich sind. Dieser Tatsache trägt die Sprache Pascal dadurch Rechnung, daß sie keinen Unterschied macht zwischen dem Aufruf von vorgegebenen Standardanweisungen und selbst definierten Prozeduren.

Wenn Sie in einem Basic-Programm irgendwo eine Zeile PRINT "TEXT" stehen haben, erwarten Sie selbstverständlich, daß

```
1710 GOSUB1600:RETURN
1740 GOSUB1640:RETURN
1810 GOSUB1600:RETURN
1860 GOSUB1640:RETURN
2010 IFSP>SMTHENER=3:GOTO50000
2011 IFIP>IMTHENER=4:GOTO50000
2020 S%(SP)=IP:IP=IP+1:SP=SP+1
2025 IN=1:RETURN
2040 IFSP<1THENER=1:GOTO50000
2041 IFIP>IMTHENER=4:GOTO50000
2044 I%(S%(SP-1))=FNAD(ZA+2)+1-DI
2050 S%(SP-1)=IP:IP=IP+1
2052 IN=2:RETURN
2100 IFSP<1THENER=1:GOTO50000
2105 SP=SP-1:I%(S%(SP))=FNAD(ZA+2)-DI
2107 IN=3:RETURN
2160 IFSP>SMTHENER=3:GOTO50000
2165 S%(SP)=-1:SP=SP+1
2170 GOSUB2010
2180 IN=1:RETURN
2210 GOSUB2040
2230 GOSUB2010
2240 IN=2:RETURN
2260 H=FNAD(ZA+2)-DI
2270 :
2275 IFSP<1THENER=1:GOTO50000
2280 SP=SP-1:I=S%(SP)
2290 IFI<0THEN2311
2300 I%(I)=H
2310 GOTO2270
2311 :
2320 IN=3:RETURN
2400 :
2410 IFMP>MMTHENER=6:GOTO50000
2415 IFCANDC<>LATHENGOSUB250:GOTO2415
2420 IFCTHENGOSUB750
2423 IFCTHENGOSUB250
2425 IFC<48ORC>57THENER=9:GOTO8050
2430 MA$(MP)=T$:H=C
2440 GOSUB750
2450 MA%(MP)=VAL(CHR$(H)+T$)-DI
2460 MP=MP+1
2470 IFC=0THEN2481
2480 GOTO2400
2481 :
2485 IN=0:RETURN
2550 GOSUB750
2560 FORI=0TOBM:IFT$<>BE$(I)THENNEXT
2565 IFI>BMTHENER=0:GOTO8050
2568 I=I+1
2570 ONIGOSUB2590,2685,2630,3010,3090,3190,3260,3310,3360,
3400,3450,3550,3580,3600
2575 RETURN
2590 IFC=0THENZ$=Z$+":"
2595 S%(SP)=LP:SP=SP+1:LP=LP+1
2597 RETURN
2630 SP=SP-1
2640 Z$=Z$+GT$+MID$(STR$(LO%(S%(SP),0)+DI),2)+NU$
2642 GOSUB550
2647 L=PEEK(ZA+2)+1:H=PEEK(ZA+3):IFL>255THENL=0:H=H+1
2648 Z$=CHR$(L)+CHR$(H)+":"
2650 RETURN
2685 B$="":IFRIGHT$(Z$,1)<>CHR$(167)THENB$=GT$
2693 Z$=Z$+B$+MID$(STR$(LO%(S%(SP-1),1)+DI+1),2)
2695 RETURN
3010 Z$=Z$+IC$+NO$+"("+CHR$(C)
3020 GOSUB250:IFC<>THANDCTHENZ$=Z$+CHR$(C):GOTO3020
3030 Z$=Z$+")"+CHR$(TH)+MID$(STR$(I%(IP)+DI),2)
3036 IP=IP+1:C=0:RETURN
3090 Z$=Z$+GT$+MID$(STR$(I%(IP)+DI),2)+NU$
```

Listing. Das Objektprogramm Strubs (Fortsetzung)

```
3100 GOSUB550
3120 L=PEEK(ZA+2)+1:H=PEEK(ZA+3):IFL>255THENL=0:H=H+1
3130 Z$=CHR$(L)+CHR$(H)+":"
3140 IP=IP+1:RETURN
3190 L=PEEK(ZA+2):H=PEEK(ZA+3)
3200 Z$=CHR$(L)+CHR$(H)+":"
3210 RETURN
3260 GOSUB3010:RETURN
3310 GOSUB3090
3320 Z$=LEFT$(Z$,LEN(Z$)-1)
3330 GOSUB3010
3340 RETURN
3360 GOSUB3190
3370 RETURN
3400 Z$="":C=0:RETURN
3450 GOSUB2590
3460 Z$=Z$+IC$+NO$+"("
3470 IFC<>BEANDCTHENZ$=Z$+CHR$(C):GOSUB250:GOTO3470
3480 Z$=Z$+")"+CHR$(TH)
3490 Z$=Z$+MID$(STR$(LO%(S%(SP-1),1)+DI+1),2)
3495 C=0:RETURN
3550 GOSUB2630:RETURN
3580 GOSUB2590:RETURN
3600 Z$=Z$+IC$+NO$+"("
3610 IFCTHENZ$=Z$+CHR$(C):GOSUB250:GOTO3610
3620 SP=SP-1:IN=3
3630 Z$=Z$+")"+CHR$(TH)+MID$(STR$(LO%(S%(SP),0)+DI),2)
3640 RETURN
4060 Z$=CHR$(PEEK(ZA+2))+CHR$(PEEK(ZA+3))
4080 NC=ZA+4:GOSUB250
4090 IFC=DPTHENGOSUB250
4100 IFNOT(C=LA)THEN4110
4105 GOSUB750:IFC=DPTHENGOSUB250
4108 IFC=0THENZ$=Z$+":"
4110 :
4115 NC=NC-1:IFC=0THENZ$=Z$+NU$
4130 :IFC=0THEN4397
4132 GOSUB250
4150 IFNOT(C=BE)THEN4359
4155 GOSUB2550
4358 GOTO4378
4359 :
4360 IFC=LATHENGOSUB1050
4378 :
4380 Z$=Z$+CHR$(C)
4396 GOTO4130
4397 :
4398 RETURN
5050 PRINT"□   ***** UEBERSETZEN     ****▒▒▒"
5052 IFNOT(FNAD(EA)<EA+50RFNAD(EA)>EA+83)THEN5054
5053 PRINT"KEIN PROGRAMM VORHANDEN":GOSUB49550:RETURN
5054 :
5058 PRINT"BITTE DISK EINLEGEN ▒▒ "
5060 PRINT"NAME FUER OBJEKT-PROGRAMM"
5065 POKE198,1:POKE631,34
5070 INPUTF$
5080 OPEN1,8,1,F$+",P,W":OPEN15,8,15
5090 INPUT#15,E,E$:IFE=0THEN5101
5095 PRINT"DISK ERR:";E;E$
5096 INPUT"NEUER VERSUCH";Z$
5098 CLOSE1:CLOSE15
5099 IFZ$<>"J"THENRETURN
5100 GOTO5060
5101 :
5120 AA=EA
5130 PRINT#1,CHR$(AAAND256);CHR$(AA/256);
5135 PRINT"1.LAUF"
5136 TA=7
5140 GOSUB5555
5143 IFSP>0THENPRINTSP;:ER=8:GOTO50000
```

Listing.
Das Objektprogramm Strubs (Fortsetzung)

dadurch nicht 50 Zeilen weiter der Wert der Variablen A verändert wird. Entsprechend sorgt nun Pascal dafür, daß eine selbst definierte Prozedur genausowenig Auswirkungen auf andere Programmteile hat wie der Aufruf einer Standard-Anweisung. Die interne Arbeitsweise einer solchen Prozedur wird vor der Programmumgebung genauso versteckt, wie dies bei der internen Arbeitsweise von im Sprachumfang enthaltenen Anweisungen der Fall ist. Entsprechend nennt man dieses Konzept auch »Information Hiding«. Programmiersprachen wie ADA, MODULA oder SIMULA bieten in dieser Hinsicht noch sehr viel weitergehende Möglichkeiten als Pascal.

Schnittstellen:

Der Datenaustausch mit der Umgebung eines Moduls erfolgt über genau definierte Schnittstellen. Bei einer solchen Schnittstelle handelt es sich um eine Menge derjenigen Annahmen, die die Programmumgebung über ein Modul macht — das heißt welche Daten es als Eingabe erwartet, welche Daten es daraufhin wieder ausgibt und welche anderen Module es seinerseits benötigt.

Modulbibliothek:

Die relative Eigenständigkeit solcher Module sorgt nun nicht nur für einfache Änderbarkeit und Erweiterbarkeit, sondern ermöglicht auch das Anlegen einer sogenannten Modulbibliothek. Eine solche Bibliothek enthält eine Reihe von Programmbausteinen, die je nach Bedarf in zu entwickelnde Programme eingefügt werden können. Dabei kann es sich um Sortierroutinen, Grafik-Routinen, mathematische und statistische Routinen und so weiter handeln. Aber auch die Entwicklung von Spielen läßt sich auf diese Weise vereinfachen: Man kann Bibliotheken fertiger Sprites, von eigenen Zeichensätzen oder von diversen Soundroutinen anlegen.

Das wichtigste Hilfsmittel zur Unterstützung modularer Programmentwicklung stellen sicherlich die lokalen Variablen dar. Leider gibt es solche nicht in Basic und auch Strubs kann keine lokalen Variablen bieten. So ist es auch weiterhin erforderlich, beim Einsetzen oder Ändern eines Moduls darauf zu achten, ob und an welchen Stellen Variablen des Moduls in anderen Programmteilen benutzt werden, und gegebenenfalls Umbenennungen vorzunehmen. Der zweite große

Nachteil von Basic — die leidigen Zeilennummern — braucht uns dagegen nur noch wenig zu beschäftigen. Strubs bietet alle Möglichkeiten, die erforderlich sind, um ein Programm vollkommen unabhängig von Zeilennummern zu schreiben Als erstes sind da natürlich die oben besprochenen Kontrollstrukturen zu nennen. Darüber hinaus können bei allen Sprüngen Zeilennummern durch Labels (Marken) ersetzt werden. Solche Labels werden durch das Zeichen »£« gekennzeichnet und abgeschlossen durch ein Leerzeichen, Doppelpunkt, Komma oder Zeilenende. Die dürfen zwar reservierte Basic-Worte enthalten, dann können sich aber wegen der in der letzten Folge erwähnten Tokens bei der Ausgabe der Markentabelle seltsame Effekte ergeben. Die Labels werden definiert, indem sie an den Anfang einer Zeile gesetzt werden und können beliebig lang sein:

```
10  AUSGEBEN:
20 : PRINT "X:";X
30  RETURN
...
200 X=1:GOSUB £X-AUSGEBEN
210 X=2:GOSUB £X-AUSGEBEN
```

Schließlich bietet Strubs noch die Möglichkeit relativer Sprünge. Diese dienen vor allem dazu, kurze Schleifen innerhalb einer einzigen Zeile zu konstruieren, ohne dafür extra ein Label zu definieren:

```
90 NC=NC+1:C=PEEK(NC):IF C>0 THEN Z$+CHR$(C): GOTO £THIS
```

Der Befehl GOTO £THIS bewirkt einen Sprung an den Anfang derjenigen Zeile, in der dieser Befehl steht.

Da bei der Arbeit mit Strubs Quellprogramme in der Regel weit umfangreicher als die Objektprogramme sind, bietet Strubs die EXTERN-DEKLARATION, die es ermöglicht, Module und Programmteile getrennt zu übersetzen und erst auf der Objektprogrammebene zusammenzufügen. Hierbei müssen die einzelnen Programmteile allerdings verschiedene Zeilennummern belegen. In der Extern-Deklaration wird ein Name vereinbart, unter dem ein Programm ein externes Modul ansprechen kann. Diesen Namen wird die Einsprungadresse (bei Maschinenprogrammen) beziehungsweise die Zeilennummer bei Basic-Routinen zugewiesen:

```
20  REM VEREINBARUNG:
30  ! EXT:  £MAPRO:740,£PLOT: 50000
```

Fortsetzung auf Seite 126

```
5145 PRINT"2.LAUF"
5150 GOSUB6550
5160 PRINT#1,CHR$(0);CHR$(0);
5180 CLOSE1:PRINT"■■■";EP;" ERRORS **":GOSUB49550
5190 RETURN
5555 ZA=EA
5570 IFNOT(ZA<>0)THEN5931
5580 NC=ZA+4:C=PEEK(NC):NC=NC+1
5585 IFC=DPTHENGOSUB250
5590 IFC=LATHENGOSUB6050:IFC=DPTHENGOSUB250
5620 IFC=BETHENGOSUB1550
5920 ZA=FNAD(ZA)
5930 GOTO5570
5931 :
5935 RETURN
6050 IFMP>MMTHENER=6:GOTO50000
6070 GOSUB750
6100 MA$(MP)=T$:MA%(MP)=FNAD(ZA+2)-DI:MP=MP+1
6120 RETURN
6550 ZA=EA:Z1=FNAD(ZA)
6560 LP=0:SP=0:IP=0
6580 :
6585 IFNOT(PEEK(ZA+4)<>KO)THEN6650
6590 GOSUB4060
6600 GOSUB550
6650 :
6655 ZA=Z1:Z1=FNAD(Z1)
6660 IFNOT(Z1=0)THEN6580
6680 RETURN
8050 PRINT"ERROR IN";FNAD(ZA+2),ER$(ER)
8060 IFEP<EMTHENER%(EP,0)=FNAD(ZA+2)-DI:ER%(EP,1)=ER:EP=EP+1
8080 Z$=LEFT$(Z$,2)+"***** ERR:"+ER$(ER)+"********"
8090 C$=NU$:C=0
8099 RETURN
8860 PRINT"■■■■■■■"
8870 PRINTTAB(9);"**********************"
8880 PRINTTAB(9);"** ZURUECK MIT:    **"
8882 PRINTTAB(9);"** ' ! ' [RETURN]  **"
8940 PRINTTAB(9);"**********************"
8950 POKE44,EA/256:POKEEA-1,0:CLR:END
8990 END
40050 PRINT"■";TAB(10);"******************"
40052 PRINTTAB(10);"* -- STRUBS  -- *"
40053 PRINTTAB(10);"*  PRECOMPILER  *"
40055 PRINTTAB(10);"* BITTE WAEHLEN *"
40058 PRINTTAB(10);"******************"
40060 PRINT"■■■■DIT"
40070 PRINT"■■■BERSETZEN"
40080 PRINT"■■MARKEN-TABELLE AUSGEBEN"
40090 PRINT"■■FEHLER-TABELLE AUSGEBEN"
40100 PRINT"■■SCHLUSS"
40160 GETZ$:IFZ$=""THEN40160
40170 IFZ$="E"THEN8860
40180 IFZ$="U"THENGOSUB5050:GOTO40050
40190 IFZ$="S"THENEND
40195 IFZ$="M"THENGOSUB48050:GOTO40050
40200 IFZ$="F"THENGOSUB49050:GOTO40050
40495 GOTO40050
45060 MM=99:DIMMA$(MM),MA%(MM):MP=0
45135 LM=140:DIMLO%(LM,1):LP=0
45145 IM=270:DIMI%(IM):IP=0
45190 SM=60:DIMS%(SM):SP=0
45220 DI=32766
45250 DP=ASC(":"):KO=ASC("'"):LA=ASC("£"):NU$=CHR$(0):BL=ASC(" ")
45253 BE=ASC("!"):TE=34:GT$=CHR$(137)
45254 IC$=CHR$(139):TH=167:NO$=CHR$(168):KM=44
45265 BM=13:DIMBE$(BM)
45270 FORI=0TOBM:READBE$(I):NEXT
45271 BE$(3)=IC$
45272 DATALOOP,EXIT,ELOOP,IF,ELSE,FI
```

Listing. Das Objektprogramm Strubs (Fortsetzung)

```
45273 DATACASEOF,OF,ECASE,EXT
45274 DATAWHILE,EWHILE,REPEAT,UNTIL
45410 DEFFNAD(X)=PEEK(X)+256*PEEK(X+1)
45480 EM=40:DIMER%(EM,1):EP=0:DIMER$(40)
45500 FORI=0TO9:READER$(I):NEXT
45510 DATA"FALSCHER BEFEHL","BLOCKSCHACHTELUNG: ANFANG FEHLT"
45511 DATA"UNDEFINIERTE MARKE","STACK VOLL"
45512 DATA"ZU VIELE IF/ELSE/CASE/OF","ZU VIELE LOOP/WHILE/REPEAT"
45513 DATA"ZU VIELE MARKEN",,"BLOCK NICHT GESCHLOSSEN"
45514 DATA"EXTERN DECLARATION"
45600 I=0:READW
45610 POKE704+I,W:I=I+1:READW:IFW<256THEN45610
45620 DATA32,115,0,8,201,33,240,4,40,76,231,167
45630 DATA169,8,133,44,169,138,76,231,167,999
45650 FORI=0TO10:READW:POKE750+I,W
45660 NEXT
45670 SYS750
45680 DATA169,192,141,8,3,169,2,141,9,3,96
45999 RETURN
48050 IFMP=0THENRETURN
48055 H=0
48057 PRINT"■0      ** MARKENTABELLE AUSGEBEN **"
48060 INPUT"00 AUF DRUCKER (J/N)";B$
48070 IFNOT(B$="J")THEN48091
48075 PRINT" DRUCKER AN?":GOSUB49550
48080 OPEN1,4
48090 GOTO48104
48091 :
48100 OPEN1,3
48102 H=-1
48104 :
48120 FORI=0TOMP-1
48140 PRINT#1,MA%(I)+DI,MA$(I)
48150 IFI-INT(I/10)*10=0THENIFIANDHTHENGOSUB49550
48180 NEXT
48185 CLOSE1:GOSUB49550
48190 RETURN
49050 IFEP=0THENRETURN
49055 H=0
49057 PRINT"■0      ** FEHLERTABELLE AUSGEBEN **"
49060 INPUT"00 AUF DRUCKER (J/N)";B$
49070 IFNOT(B$="J")THEN49091
49075 PRINT"0 DRUCKER AN?00":GOSUB49550
49080 OPEN1,4
49090 GOTO49104
49091 :
49100 OPEN1,3
49102 H=-1
49104 :
49110 PRINT#1,EP;" ERRORS"
49120 FORI=0TOEP-1
49140 PRINT#1,ER%(I,0)+DI;ER$(ER%(I,1))
49150 IFI-INT(I/10)*10=0THENIFIANDHTHENGOSUB49550
49180 NEXT
49185 CLOSE1
49191 GOSUB49550
49190 RETURN
49550 PRINT"->■■";
49560 GETB$:IFB$=""THEN49560
49570 RETURN
50000 PRINT"** FEHLER BEHEBEN, DANN NEU VERSUCHEN *"
50008 PRINT:PRINTER$(ER);" IN ";FNAD(ZA+2)
50010 PRINT#1,CHR$(0);CHR$(0);
50020 CLOSE1
50030 GOSUB49550
50040 GOSUB49050
50050 RUN

READY.
```

Listing. Das Objektprogramm Strubs (Schluß)

Fortsetzung von Seite 121

...

```
90 REM AUFRUF:
99 SYS £MAPRO: X=13:Y=90:GOSUB £PLOT
```

Kommen wir abschließend zur Dokumentation: Vom Hobby-Programmierer kann kein Mensch erwarten, daß er Berge von Dokumentationsmaterial anlegt, die den Umfang des Programmtextes um ein Vielfaches übersteigen. Deshalb ist es gerade hier wichtig, Programme weitgehend selbstdokumentierend zu schreiben. Im Gegensatz zu höheren Programmiersprachen mit ihren zahlreichen Deklarationspflichten ist der Basic-Programmierer nahezu ausschließlich auf Kommentare angewiesen. Da Strubs Kommentare bei der Übersetzung eleminiert, stehlen diese weder Speicherplatz noch Laufzeit. Der Programmierer kann also ohne Bedenken einen exzessiven Gebrauch von Kommentaren machen.

Kommentare werden gekennzeichnet durch das Zeichen » «. Steht dieses Zeichen direkt am Zeilenanfang, so wird die ganze Zeile gelöscht. Sonst wird der Programmtext bis zum zweiten » « oder bis zum Zeilenende überlesen. Außer innerhalb von Befehls- und Markennamen können Kommentare an jeder beliebigen Programmstelle eingefügt werden. Kommentare, die in das Objektprogramm übernommen werden sollen, können wie bisher mit REM in den Programmtext eingefügt werden. Beispiel:

```
10 ´DIESE ZEILE WIRD VOLLSTÄNDIG GELÖSCHT
20 A´US´G´ABE´$="ENTSPRICHT AG$" ´KOMMENTAR
```

Die Lesbarkeit von strukturierten Programmen wird verbessert durch das Einrücken von Zeilen entsprechend der Blockstruktur. Hierzu dient der Tabulator (Bild 2): Ein Doppelpunkt am Zeilenanfang gefolgt von Leerzeichen. Für die Ungeduldigen ist das Opjektprogramm von Strubs bereits abgedruckt (Listing). In der nächsten Ausgabe werden wir auf die praktische Programmentwicklung mit Hilfe von Strubs eingehen.

(Matthias Törk)

Literatur

* N. Wirth: Systematisches Programmieren, Teubner, Stuttgart 1978
* Kimm, R./Koch,W./Simonsmeier,W./Tontsch,F.: Einführung in Software Engineering, De Gruyter, Berlin, New York 1979
* Schnupp, P./FLoyd, C.: Software: Programmentwicklung und Projektorganisation, De Gruyter, Berlin, New York 1976
* Nagl, M.: Einführung in die Programmiersprache ADA, Vieweg, Braunschweig, Wiesbaden 1982