# Programmieren Sie strukturiert!

## (Teil 2)

**Im ersten Teil haben wir uns mit Sequenzen, Schleifen und Verzweigungen beschäftigt, das heißt mit Steuerbausteinen, die festlegen, wie das Programm jeweils fließt. Heute geht es um einen anderen Bausteintyp, um die Unterprogramm-Bausteine.**

Jede Programmiersprache, und sei sie noch so reichhaltig, kann immer nur eine begrenzte Anzahl von Befehlen zur Verfügung stellen. Und so kommen wir beim Programmieren immer wieder an den Punkt, wo ein Befehl, den man eigentlich brauchte, nicht vorhanden ist.

Kein Grund zur Resignation. Was man nicht hat, verschafft man sich. Es gibt zwei Möglichkeiten, sich neue Befehle zu verschaffen: Entweder man besorgt sie sich, oder man macht sie sich selber.

Wie man sich Befehle besorgen kann? Nun ja, man sammelt sie zum Beispiel aus Zeitschriften, man studiert Programme anderer Leute, oder man kauft sich eine Befehlesammlung, wie zum Beispiel Macro Basic, und stellt sich daraus jeweils die Befehlsmenge zusammen, die man gerade braucht.

Was man sich nicht versorgen kann, muß man sich, wie gesagt, selber machen. Wie, darum geht es im folgenden.

Neue Befehle erstellt man mit Hilfe von Unterprogramm-Bausteinen. Basic stellt zwar solche Bausteine zur Verfügung (die Subroutinen und die Funktionen), aber diese sind, vom Standpunkt des strukturierten Programmierens aus, doch sehr verbesserungsbedürftig. Wir wollen sehen, wo Verbesserungen möglich sind und wie sie aussehen können.

Wie im ersten Teil wollen wir auch dabei die Programmiersprache Comal als Wegweiser benutzen. Diese Sprache ist, wie schon dort angemerkt wurde, besonders gut durchdacht, besonders menschenfreundlich ausgestaltet. Und sie stellt uns Unterprogramm-Bausteine zur Verfügung, die es dem Programmierer ausgesprochen leicht machen, neue Befehle zu erfinden und einzusetzen. Diese Unterprogrammstruktur wollen wir in Basic so weit wie möglich imitieren. Im übrigen werden wir uns, wo notwendig, auch von anderen Programmiersprachen, wie zum Beispiel Ada, anregen lassen.

### Befehlstypen

Es gibt in Programmiersprachen viele verschiedene Arten von Befehlen. Wenn wir zum Beispiel sagen »PREIS = 25.99«, dann haben wir einen *Zuweisungs*befehl benutzt. Schleifen und Verzweigungen gehören zu den *Steuer*befehlen, wir haben sie in der letzten Folge behandelt. Im heutigen Zusammenhang interessieren uns die beiden folgenden Typen: die *Handlungs*befehle und die *Funktionen*.

### Handlungsbefehle

»PRINT NAME$« ist ein Handlungsbefehl. Er bringt den Computer dazu, eine Druck*handlung* auszuführen. »POKE 1024,1« bewirkt, daß der Computer in die erste Bildschirmspeicherzelle des C 64 die Zahl 1 steckt, so daß ein »A« in der linken oberen Ecke des Bildschirms erscheint. »SAVE "programm",8« veranlaßt den Computer dazu, das Programm, das im Speicher ist, auf Diskette zu schreiben.

Handlungsbefehle benutzt man, wenn man will, daß der Computer eine bestimmte Handlung ausführt.

### Funktionen

Funktionen sind Befehle, die man als spezialisierte Handlungsbefehle ansehen kann. Ihre spezielle Aufgabe ist es, solche Handlungen durchzuführen, die Daten zum Ergebnis haben. Praktisch gesprochen: Funktionen sind Befehle, die Daten erzeugen. Die Funktion »INT(25.99)« erzeugt die Zahl »25«, die Funktion »MID$ ("Zeitschrift",5,3)« den Text »sch«.

Zu den speziellen Eigenschaften von Funktionen gehört, daß man ihnen das erzeugte Datum abnehmen muß. Während es bei Handlungsbefehlen genügt, einfach den Befehl auszusprechen, zum Beispiel »RESTORE«, reicht dies bei Funktionen nicht aus. »INT(25.99)« mag zwar möglicherweise dazu führen, daß der Computer die notwendigen Handlungen durchführt (Dezimalpunkt finden, Bruchteil abschneiden), aber er streikt spätestens dann, wenn er nicht weiß, was er mit dem erzeugten Zahlenwert anfangen soll. Wenn man eine Funktion verwendet, muß man also dem Computer gleichzeitig auch sagen, was mit dem erzeugten Datum zu geschehen hat. Beispiele: PRINT INT(25.99) oder GANZZAHL = INT(25.99) oder IF INT(25.99) < 100 THEN...

Das ist natürlich kein Nachteil; es hat vielmehr den Vorteil, daß man Funktionen wie Zahlen (beziehungsweise Texte) einsetzen kann. Funktionsbefehle, kann man auch sagen, werden immer in Ausdrücken aufgerufen. Lassen Sie uns nun sehen, wie man Handlungsbefehle und Funktionen herstellt, die nicht in der Programmiersprache vorhanden sind.

### Comal: Prozeduren und Funktionen

Comal stellt zwei Bausteintypen für selbstgemachte Befehle zur Verfügung, die einander sehr ähnlich sind: *Prozeduren* und *Funktionen*. Beide haben prinzipiell dieselbe Struktur. Für Handlungsbefehle benutzen wir die Prozedurstruktur, für selbstgestrickte Befehle vom Typ Funktion die Funktionsstruktur.

Beispiel 1: Bildschirm löschen

In einigen Programmiersprachen gibt es den Handlungsbefehl PAGE, der den Bildschirm löscht (und also eine neue Bildschirm*seite* anfängt). Für die Definition dieses Befehls benutzen wir folgenden Prozedurbaustein:

```
9000  PROC page
9010  PRINT CHR$(147),
9020  ENDPROC page
```

Wenn diese Prozedur in einem Comal-Programm enthalten ist, dann steht damit in diesem Programm der Befehl PAGE zur Verfügung. Das heißt, immer wenn der Bildschirm gelöscht werden soll, geben wir einfach den Namen der Prozedür, also PAGE, ein:

```
0010  page
0020  ...
```

Eine neue Funktion zu definieren, ist ebenso einfach. Sie unterscheidet sich in ihrem Aufbau von einer Prozedur nur durch den zusätzlichen Befehl RETURN (der übrigens nichts mit dem gleichnamigen Befehl in Basic zu tun hat!).

Beispiel 2: Wo befindet sich der Cursor?

Die Funktion CURSORZEILE soll die Nummer der Zeile ausgeben, in der sich der Cursor gerade befindet:

```
9000  FUNC cursorzeile
9010    zeile:=PEEK(214)+1
9020    RETURN zeile
9030  ENDFUNC cursorzeile
```

Die Speicherzelle 214 enthält beim Commodore 64 die Nummer der Zeile, in der sich der Cursor gerade befindet. Der Wert 1 wird addiert, damit die erste Zeile auch tatsächlich die erste ist und nicht etwa die nullte.

Der Befehl »RETURN zeile« (9020) weist die Funktion an, den Wert der Variablen ZEILE »zurückzugeben« (to return the value) und die Funktion zu verlassen. Dies ist der Wert, den die gesamte Funktion zur Verfügung stellt; den Sie also erhalten, wenn Sie sagen:

```
PRINT cursorzeile
```

Wenn also die Funktion CURSORZEILE in einem Comal-Programm steht, dann steht damit in diesem Programm der neue Befehl CURSORZEILE zur freien Verfügung. Beispiel:

Nehmen wir an, Sie drucken einen längeren Text auf dem Bildschirm aus. Sie wollen, daß immer nur 20 Zeilen gedruckt werden; danach soll der Bildschirm gelöscht werden und der Text wieder in der ersten Zeile beginnen. Sie könnten schreiben:

```
0100  IF cursorzeile > 20 THEN page
```

In dieser Zeile sind also zwei selbstdefinierte Befehle verwendet, der Handlungsbefehl PAGE und der Funktionsbefehl CURSORZEILE.

Lassen Sie uns nun anschauen, wie dasselbe in Basic aussieht.

### Basic: Subroutinen und Funktionen

Basic stellt ebenfalls zwei Bausteintypen für selbstdefinierte Befehle zur Verfügung, einmal die *Subroutinen*, zum andern die *Funktionen*. Im Gegensatz zu Comal sind die beiden Bausteintypen allerdings völlig verschieden aufgebaut. Die Subroutinenstruktur kann für selbstgemachte Handlungsbefehle, die Funktionsstruktur für Funktionsbefehle eingesetzt werden.

Eine Subroutine, die den Bildschirm löscht, könnte so aussehen:

```
1000  printchr$(147);
1020  return
```

Eine Funktion, die die Zeile abfragt, in der sich der Cursor gerade befindet, wird so definiert:

```
100   deffncz(x)=peek(214)+1
```

Die Programmzeile, die bewirkt, daß nur 20 Zeilen gedruckt und danach der Bildschirm freigemacht wird, sähe so aus:

```
500   iffncz(0)>=20thengosub1000
```

Zu beachten ist dabei, daß in Basic (anders als in Comal) die Funktion definiert sein muß, bevor sie aufgerufen werden kann. Funktionsdefinitionen findet man deshalb häufig zu Beginn von Basic-Programmen.

Ein Vergleich zwischen der Comal- und der Basic-Definition unserer selbstdefinierten Befehle macht sehr deutlich, um wieviel menschenfreundlicher Comal ist, einmal beim Codieren, vor allem aber beim Lesen eines Programms. Bei Comal benutzen wir einfach den Namen einer Prozedur oder einer Funktion, wenn diese abgearbeitet werden soll, und wenn wir die Namen geschickt gewählt haben, verstehen wir auf Anhieb, was das Programm jeweils tut. Eine Basic-Zeile hingegen erschließt sich nur nach langem Studium des Programms, falls überhaupt: Um FNCZ(0) zu verstehen, muß die Definition dieser Funktion gesucht werden; wenn man wissen will, was GOSUB1000 bewirkt, muß man zur Zeile 1000 gehen und das dortige Unterprogramm analysieren.

Basic hat es also bitter nötig, menschenfreundlicher gemacht zu werden. Lassen Sie uns dies in Angriff nehmen. Zwar können wir natürlich die Basic-Version, die auf dem Commodore 64 installiert ist, nicht ändern, aber wir können dasselbe tun, was wir in Teil I getan haben: Wir können Bausteinstrukturen entwickeln, die denen, die wir in Comal finden, nachempfunden sind, und uns auf diese Weise sowohl das Codieren wie das Lesen unserer Basic-Programme erleichtern.

## Prozeduren in Basic: Grundprinzipien

Neben den Grundprinzipien, die für alle Bausteine gelten (insbesondere, daß jeder Bausteinblock nur einen Eingang und einen Ausgang hat — vergleiche Teil 1), wollen wir zusätzlich folgendes beachten:

1. Eine Prozedur muß, wenn sie lesbar und verstehbar sein soll, überschaubar bleiben. Das bedeutet, sie darf eine gewisse Länge nicht überschreiten. Wenn irgend möglich, soll sie auf einer Seite Platz finden.

Das ist ein relatives Maß. Wer Listings nur gedruckt studiert, könnte als Maßstab die Druckseite festlegen; wer Programme auf dem Bildschirm verstehen will (und das ist zum Beispiel, wenn man Fehler verbessert, der Normalfall), wird diesen zum Maßstab machen. Wer einen 80-Zeichen-Bildschirm besitzt, kann mehr unterbringen als wer nur 40 Zeichen zur Verfügung hat. Da wir als C 64-Benutzer uns mit 40 Zeichen begnügen müssen, soll dies unser Maß sein: Prozeduren sollen möglichst auf einen C 64-Bildschirm passen.

2. Eine Prozedur ist so zu konzipieren, daß sie eine Welt für sich bildet. Was in der Außenwelt passiert, darf sie nicht berühren. Was in der Welt der Prozedur geschieht, darf nicht nach außen wirken.

Das hat einen sehr praktischen Grund. Veränderungen, die im Programm vorgenommen werden, wirken dann immer nur auf einen überschaubaren Bereich und bleiben auf diese Weise kontrollierbar. Wenn hingegen dieses Prinzip nicht beachtet wird, kann die Wirkung einer unbedeutenden Änderung an einer Stelle des Programms ein ganzes Programm unbrauchbar machen. Es gibt wohl keinen Programmierer, der dazu nicht ein garstig Liedchen beisteuern könnte.

Im übrigen haben Prozeduren dieser Art noch den Vorteil, daß man sie bei Bedarf auch in anderen Programmen verwenden kann, ohne daß man sie an die Situation des neuen Programms anpassen müßte.

3. Eine Prozedur soll zwar eine abgeschlossene Welt sein, das bedeutet aber nicht, daß nicht Kommunikation zwischen Prozedur und Außenwelt stattfinden könnte.

Solche Kommunikation kann zwei Richtungen haben: von der Außenwelt in die Prozedur und aus der Prozedur in die Außenwelt. Im einen Fall verarbeitet die Prozedur Daten, die sie von außen erhält, im andern Fall gibt sie Ergebnisse ihres Wirkens der Außenwelt bekannt.

Je nach dem Typ der Kommunikation zwischen Prozedur und Außenwelt können wir die folgenden Typen unterscheiden:

1. Prozeduren ohne Kommunikation mit der Außenwelt
2. Prozeduren, die Information hereinlassen
3. Prozeduren, die Information hinauslassen
4. Prozeduren, die Information sowohl herein- und hinauslassen

4. Damit keine unbeabsichtigte Kommunikation zwischen Prozedur und Außenwelt stattfinden kann, müssen wir dafür sorgen, daß Variablen, die in der Welt der Prozedur benutzt werden, nur in dieser Welt und sonst nirgendwo bekannt sind. Man nennt solche Variablen »lokal« (im Gegensatz zu »globalen« Variablen, die sowohl in der Außenwelt wie in der Prozedur gelten).

## Prozeduren ohne Kommunikation mit der Außenwelt

Beispiel 3: Linie

Die Prozedur soll eine Linie über den Bildschirm ziehen.

```
Prozedur ohne Kommunikation
LINIE
Anfang Block
  Linie zeichnen
Ende Block
```

Die Handlung »Linie zeichnen« kann hier sehr einfach mit Hilfe einer einzeiligen Zählschleife bewerkstelligt werden. In Comal wird das so codiert:

```
9000  PROC linie CLOSED
9010    FOR i#=1 TO 40 DO PRINT CHR$(192),
9020  ENDPROC linie
```

Die Zählvariable I# ist eine Integervariable, was in Comal im Gegensatz zu Basic möglich ist und die Geschwindigkeit des Schleifendurchlaufs um ein Mehrfaches erhöht.

Diese Variable gilt nur innerhalb der Prozedur, sie ist nur lokal gültig. Dies wird dadurch bewirkt, daß die Prozedur ausdrücklich mit dem Befehl CLOSED gegenüber der Außenwelt abgeschottet wird. Wenn also woanders im Programm die Variable I# noch einmal auftritt, macht das keine Probleme. Das heißt ein Comal-Programm könnte folgenden Schleifenblock enthalten, also 20mal Linie aufrufen, ohne daß Schwierigkeiten entstünden:

```
0100  FOR i#= 1 to 20
0110    linie
0120  ENDFOR i#
```

In Basic ist dies in so einfacher Weise nicht zu lösen. Es gibt keinen Befehl, der eine Subroutine abschließen und deren Variablen von der Außenwelt abschotten könnte. Wir müssen vielmehr selber dafür sorgen, daß Variablen lokal sind. Dies können wir dadurch erreichen, daß wir bestimmte Variablennamen für Prozeduren reservieren und außerhalb von Prozeduren grundsätzlich nicht verwenden. Ich schlage vor, daß wir Variablennamen, die wir in Prozeduren verwenden, mit U beginnen lassen und außerhalb von Prozeduren keine Namen, die mit U beginnen, benutzen (»U« steht für »Unterprogramm«).

In Basic gibt es weiterhin keinen Prozedurrahmen, der die Prozedur deutlich von ihrer Umgebung abgrenzen könnte. Wir brauchen jedoch einen, denn wir wollen ja, daß unsere Programme gut lesbar sind. Also müssen wir selber einen schaffen.

Den Prozedurkopf wollen wir mit einer REM-Zeile so markieren:

```
REM PROC:  Prozedurname
```

(Später werden wir noch eine Klammer für Variablen anfügen.)

Als Endemarkierung benutzen wir RETURN. Wir können uns deshalb damit begnügen, weil wir ja grundsätzlich jeden Baustein, also auch Unterprogramme, so bauen, daß Sie nur einen Ausgang haben, und diesen immer am Ende des Bausteins. (Vergleiche Teil I).

Wir können nun eine Basic-Prozedur LINIE analog zum Comal-Vorbild codieren.

```
42000  rem proc: linie
42010    for ui=1 to 40:print chr$(192);: next
42020  return
```

Aufgerufen wird eine solche Prozedur in Basic leider nicht einfach mit dem Prozedurnamen wie in Comal, sondern viel umständlicher mit »GOSUB Zeilennummer«. Ein Programmblock, der 20 Linien druckt, würde in Basic also so aussehen:

```
100  for i=1 to 20
110    gosub 42000: rem linie
120  next
```

Da die Schleifenvariable des Hauptprogramms I und die Schleifenvariable der Prozedur UI unterschiedlich sind, kann auch hier kein Konflikt entstehen. Aber dafür ist in Basic, wie gesagt, der Programmierer verantwortlich.

Der Programmierer muß auch für mehr Lesbarkeit sorgen. In Comal informiert der Prozedurname sowohl bei der Definition wie beim Aufruf der Prozedur darüber, was die Prozedur tut. In Basic müssen wir diese Information selber beisteuern — durch REM-Bemerkungen, sowohl im Prozedurkopf (Zeile 42000) als auch da, wo die Prozedur mit GOSUB aufgerufen wird (Zeile 110).

Und noch eins: Comal sorgt automatisch für bessere Lesbarkeit, indem es selbständig einrückt und Leerzeichen verlangt. Auch hier muß der Basic-Programmierer selber handeln.

Die Prozedur LINIE führt die gewünschten Handlungen durch, ohne daß sie Information von der Außenwelt benötigte. Das ist jedoch

nur selten der Fall — die meisten Befehle kommen ohne Kommunikation mit der Außenwelt nicht aus.

### Prozeduren mit Einwegkommunikation 1: Information kommt herein

Beispiel 4: Pause

Wenn die Prozedur PAUSE aufgerufen wird, soll das Programm die angegebene Anzahl Sekunden pausieren.

```
Prozedur mit Einwegkommunikation: Info kommt herein
                    PAUSE
Anfang Block
Anzahl Schleifen berechnen
  Schleifenanfang
  nichts tun
  Schleifenende
Ende Block
```

Die Comal-Prozedur:

```
9000  PROC pause(sekunden) CLOSED
9010    anzahl'schleifen:=sekunden*1050
9020    FOR i=1 TO anzahl'schleifen DO NULL
9030  ENDPROC pause
```

Bei Comal (Version 2.01) läuft der Computer ungefähr 1050mal in einer Sekunde durch eine leere Zählschleife. Wenn das Programm 4 Sekunden lang pausieren soll, dann muß man ihn anweisen, 4 x 1050mal eine solche Schleife zu durchlaufen.

Wieviel Sekunden die Pause dauern soll, muß der Prozedur natürlich mitgeteilt werden. Dies geschieht durch die Variable SEKUNDEN, die dem Prozedurnamen in Klammern folgt. Eine solche Variable in Klammern schlägt gleichsam ein Loch in die Mauer, welche die Prozedur umgibt, und schafft einen Eingang, durch den eine Information in das Innere der Prozedur gelangen kann.

Der Variablename SEKUNDEN gilt übrigens nur innerhalb der Prozedur, ist also lokal.

Wenn man im Programm eine Pause von 4 Sekunden Länge benötigt, gibt man folgenden Befehl ein:

```
pause (4)
```

Man kann natürlich statt der Zahl auch einen Variablennamen benutzen, zum Beispiel

```
pause (anzahl'sekunden)
```

Ja, man kann sogar denselben Variablennamen benutzen wie in der Prozedurdefinition:

```
pause (sekunden)
```

Für Comal handelt es sich trotzdem um zwei verschiedene Variablennamen, der eine ist in der Außenwelt zuhause, der andere gilt nur lokal, das heißt in der Welt der Prozedur.

In Basic müssen wir da wieder vorsichtig und selber um die »Lokalität« der Prozedurvariablen besorgt sein, indem wir, wie verabredet, U vor dem Variablennamen schreiben. Den Prozedurkopf erweitern wir jetzt, wie angekündigt, um eine Klammer für Variablen.

```
43000  rem proc: pause (usek: in)
43010    uanzahl=usek*950
43020    for ui=1 to uanzahl: next
43030  return
```

In Basic läuft der Commodore 64 nur 950mal in der Sekunde durch eine leere Zählschleife — deshalb die Zahl 950 im Basic-Programm.

Das (englische) Wort »IN« vor dem Namen der Variablen USEK soll andeuten, daß sie einen Wert von außen erhält und diesen in die Prozedur hineinnimmt. Nachher werden wir für die Gegenrichtung das Wort »OUT« benutzen. Die Anregung, IN und OUT in dieser Weise zu benutzen, kommt übrigens aus der Programmiersprache Ada, die sich an dieser Stelle noch menschenfreundlicher als Comal gibt.

Wie geben wir nun unseren Pausenbefehl in Basic ein? In Comal konnten wir den Befehlsnamen schreiben und in Klammern die Anzahl der Sekunden nennen: PAUSE(4). Der Wert »4« wird von Comal automatisch der Prozedur übermittelt. In Basic müssen wir wieder selber tätig werden und den Wert »4« der Subroutine eigenhändig mitteilen. Wir tun dies so:

```
usek=4: gosub 43000: rem pause
```

An dieser Stelle wird nun vielleicht auch klar, warum es sinnvoll ist, im Kopf der Basic-Prozedur »USEK: UN« anzugeben: Dies erinnert uns daran, daß wir beim Aufruf der Prozedur nicht vergessen dürfen, der Variablen USEK: einen Wert zuzuweisen.

Anmerkung: Zur guten Lesbarkeit von Comal-Programmen trägt auch die Möglichkeit bei, lange Variablennamen zu benutzen (sie können bis zu 78 Zeichen lang sein!). Basic kann Variablennamen nur bis zur Länge von 2 Zeichen verstehen. ZEIT und ZETTEL zum Beispiel kann es nicht unterscheiden, denn beide beginnen mit ZE. Trotzdem sind wir nicht auf zwei Zeichen beschränkt, längere Namen werden akzeptiert, nur müssen wir, wie immer bei Basic, selber denken und deshalb aufpassen, daß keine unserer Variablen in den ersten beiden Zeichen übereinstimmen. Und auf noch etwas müssen wir achten: Ein Variablenname darf kein Basic-Befehlswort enthalten; der Name KORREKT zum Beispiel erzeugt einen Syntaxfehler, weil er QR enthält. Man kann Basic überlisten, indem man den Namen so schreibt: KO RREKT; er trägt dann immer noch zur besseren Verständlichkeit des Programms bei, und der Computer macht keine Zicken. Es gibt noch einen Trick, um Basic zu überlisten. Schreiben Sie »KO«, tippen Sie dann irgendein Grafikzeichen ein, zum Beispiel Shift-O, und anschließend »RREKT«. Wenn Sie die Zeile wieder listen, ist das Grafikzeichen unsichtbar, obwohl es noch immer vorhanden ist und das »O« vom »R« trennt. Aber Vorsicht: Wenn Sie (zum Beispiel nach einer Veränderung der Zeile) noch einmal RETURN drücken, verschwindet das Grafikzeichen wieder, und Basic hat Sie überlistet!

### Prozeduren mit Einwegkommunikation 2: Information geht hinaus

Beispiel 5: Zufallswort

Wenn der Befehl ZUFALLSWORT'ERZEUGEN eingegeben wird, soll ein Wort hergestellt werden, das aus zufällig ausgewählten Buchstaben besteht. Die Wortlänge soll (ebenfalls zufällig) zwischen 1 und 10 Zeichen betragen. Das Wort soll in der Variablen TEXT$ gespeichert und an die Außenwelt gegeben werden. Wörter, die entstehen könnten: LM, LVBFPSNL, KEKN, YN,RGUY, etc. (Man benutzt derartige Befehle manchmal, um Sortierprogramme zu testen.)

```
Prozedur mit Einwegkommunikation: Info geht hinaus
                ZUFALLSWORT'ERZEUGEN
Anfang Block
  Variable initialisieren
  Wortlänge bestimmen
  Wort erzeugen
Ende Block
```

Die Comal-Prozedur:

```
9000  PROC zufallswort'erzeugen(REF text$) CLOSED
9010    text$:=" "
9020    wortlaenge:=RND(1,10)
9030    FOR i#=1 TO wortlaenge
9040      ascii=RND(65,90)
9050      text$=text$+CHR$(ascii)
9060    ENDFOR i#
9070  ENDPROC zufallswort'erzeugen
```

(Übrigens läßt Comal uns hier eigentlich im Stich. Was wir bräuchten, wäre ein Prozedurtyp, der nur Information hinausläßt. Den aber gibt's nicht. Aber es gibt einen, der Information herein- und wieder hinausläßt. Den haben wir hier ersatzweise benutzt. Wir ignorieren halt die hereinkommende Information. Das Fenster für den Informationsgegenverkehr wird durch den Zusatz REF geöffnet, der also bewirkt, daß über die Variable TEXT$ Information nicht nur hereinkommt, sondern auch wieder hinausgeht.)

Aufgerufen wird die Prozedur, wie bekannt (die Klammer enthält die Variable WORT$, die außerhalb der Prozedur gilt und nach dem Aufruf das erzeugte Zufallswort aufnimmt):

```
zufallswort'erzeugen(wort$)
```

Nach dem Aufruf enthält die Variable WORT$ also das erzeugte Zufallswort, zum Beispiel KEKN, so daß Sie nun also sagen könnten: PRINT WORT$.

Die Basic-Prozedur sieht so aus:

```
30000  rem proc: zufallswort erzeugen (utext$: out)
30010    utext$=""
30020    ulaenge=int(rnd(1)*10+1)
30030    for iu=1 to ulaenge
30040      ua=int(rnd(1)*(90-65)+65)
30050      utext$=utext$+chr$(ua)
30060    next
30070  return
```

Den Befehl ZUFALLSWORT'ERZEUGEN ruft man so im Basic-Programm auf:

```
gosub 30000:wo rt$=utext$:
```

Das heißt nach der Rückkehr aus der Subroutine müssen Sie der Variablen WO RT$ den in der Prozedur hergestellten Inhalt, das erzeugte Zufallswort, mit eigener Hand zuweisen. Erst dann können Sie sagen: PRINT WO RT$.

Bisher wurden Prozeduren ohne Kommunikation sowie mit Einwegkommunikation (nur Eingabe oder nur Ausgabe) beschrieben. Natürlich gibt es auch Prozeduren mit Zweiwegkommunikation, also mit Ein- und Ausgabe. Auch die Zahl der Variablen ist prinzipiell nicht beschränkt. Doch darüber mehr in der übernächsten Ausgabe.

(Burkhard Leuschner/gk)