Funktionen für Anfänger

Auch in Basic kann man Befehle selber entwickeln, zumindest einen bestimmten Typ von Befehl. Und dazu braucht man keine Maschinensprache und keinen Assembler, sondern nur den gesunden Menschenverstand, wie man ihn auch sonst beim Programmieren einsetzt. Gemeint sind die »benutzerdefinierten Funktionen«.

Anfänger haben mit der Definition von neuen Befehlen oft Schwierigkeiten. Das liegt aber nicht an den Anfängern, sondern eher an den meist recht verwirrenden Erklärungen der Handbücher. Handbücher sind zumeist von Computerexperten geschrieben, die oft vergessen, daß ihre Leser erst noch Experten werden wollen und deshalb zunächst mit Begriffen wie »Dummy-Variable« oder »Übergabeparameter« und was es sonst noch an stolzen Termini gibt, nichts anfangen können. Ich jedenfalls konnte es nicht und habe deshalb lange gebraucht, bis ich selbstgestrickte Funktionen so selbstverständlich in meinen Programmen benutzte wie zum Beispiel PRINT.

Vergessen Sie also alles, was Sie bisher verwirrt haben mag, und fangen Sie, zusammen mit mir, noch einmal von vorne zu denken an.

Ich will in drei Schritten vorgehen. Wir wollen zunächst klären, was Funktionen überhaupt sind, was für Eigenschaften sie haben, was sie tun, wofür man sie braucht. Auf dem Hintergrund dieser allgemeineren Informationen wollen wir uns in einem zweiten Schritt der Herstellung eigener Funktionen widmen. Ein drittes Kapitel soll dann ein paar speziellere Hinweise geben. Ein kleiner Anhang schließlich wird ein paar einfache Funktionen zusammenstellen, die nicht die Welt bewegen, sondern nur Sie anregen sollen, sich eine eigene Funktionen-Bibliothek aufzubauen.

1. Funktionen in Basic

1.1 Was ist eine Funktion?

Eine Funktion ist ein Befehl, der den Computer anweist, eine Zahl oder einen Text (Zeichenkette, »String«) zu erzeugen. Die RND-Funktion zum Beispiel erzeugt eine Zufallszahl; die Funktion LEFT$ erzeugt eine Zeichenkette.

Es gibt zwei Grundtypen von Funktionen: solche, die lediglich Daten (Zahlen oder Zeichenketten) ausgeben, und solche, denen man Daten (Zahlen oder Zeichenketten) eingibt, die sie dann in anderer Form wieder ausgeben. Wir wollen die einen Ausgabe-Funktionen nennen und die anderen Eingabe-Ausgabe-Funktionen (Bild 1).

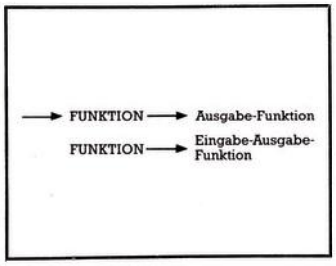

Bild 1. Die zwei Grundtypen von Funktionen

Die oben genannte RND-Funktion ist in diesem Sinne eine reine Ausgabe-Funktion. Ausgegeben wird eine Zufallszahl zwischen 0 und 1. Die INT-Funktion hingegen ist eine Eingabe-Ausgabe-Funktion. Eingegeben wird eine Zahl, zum Beispiel 12.78, ausgegeben werden die Ziffern vor dem Komma, also 12 (Bild 2).

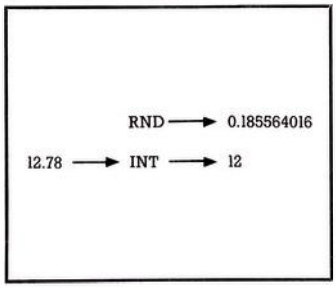

Bild 2. Zwei Grundtypen

Die Funktion INT erzeugt eine Zahl aus einer anderen Zahl; die Funktion LEFT$ erzeugt einen Text aus einem Text. Es geht aber auch »überkreuz«. Die Funktion LEN erhält als Eingabedatum einen Text und gibt eine Zahl aus, während umgekehrt die Funktion STR$ aus einer Zahl einen Text macht (Bild 3).

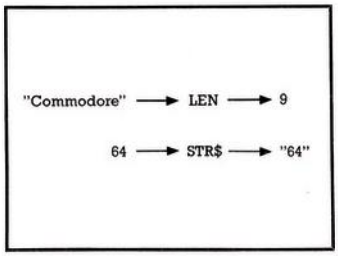

Bild 3. Eingabe: Text, Ausgabe: Zahl — und umgekehrt

Fassen wir zusammen: In (Commodore-) Basic finden wir die folgenden sechs Typen von Funktionen (Bild 4).

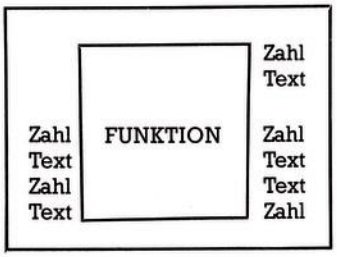

Bild 4. Basic-V.2.0 besitzt sechs Typen von Funktionen

Eine kleine Anmerkung noch: Die Befehle SPC und TAB, im Commodore-Handbuch unter der Überschrift »Funktionen« aufgeführt, sind in unserem Sinne keine Funktionen, da sie keine Daten erzeugen, sondern etwas bewirken, so wie zum Beispiel PRINT etwas bewirkt.

1.2 Mitteilungen an die Funktion

Wenn eine Funktion eine Zahl in eine andere umwandeln soll, dann muß ihr die Zahl in irgendeiner Weise mitgeteilt werden. Für Mitteilungen an Funktionen ist ein bestimmter Platz vorgesehen, nämlich die Klammern, die jedem Funktionsnamen (in Basic) folgen (Bild 5).

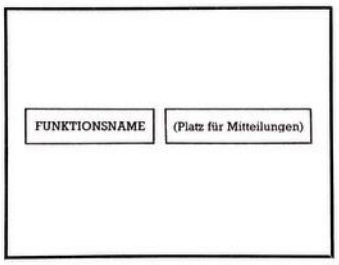

Bild 5. Einige Funktionen benötigen zusätzliche Angaben

Die Information, die eine Funktion braucht (sofern sie überhaupt eine braucht), kann von zweierlei Art sein: Es kann sich einmal um ein Eingabedatum handeln, das von der Funktion bearbeitet werden soll (zum Beispiel INT (12.78)), oder um Informationen darüber, wie die Funktion arbeiten soll. Daraus ergeben sich die folgenden Möglichkeiten (Bild 6).

ARTEN VON MITTEILUNG AN FUNKTIONEN
—
Eingabedatum
Arbeitshinweis
Eingabedatum und Arbeitshinweis

Bild 6. Arten von Mitteilung an Funktionen

Ein paar Beispiele zur Illustration:

Die Funktion POS, die ausgibt, in welcher Bildschirmspalte sich der Cursor gerade befindet, weiß alles, was sie wissen muß, um ihre Aufgabe erfüllen zu können; der Programmierer muß ihr also keinerlei Informationen mitgeben. Da POS nun aber eine Funktion ist, ist ein Platz für Mitteilungen vorgesehen, das heißt der Programmierer muß die dem Funktionsnamen folgenden Klammern mit irgend etwas füllen. Der Einfachheit halber nimmt man dafür »0«: POS (0). Wenn Sie unbedingt wollen, können Sie auch irgend etwas anderes in die Klammern stecken, zum Beispiel »X« oder Ihren Namen — die Mitteilung landet in jedem Fall im Papierkorb, der Computer ignoriert sie.

Nicht im Papierkorb landet die Eingabezahl, die Sie zum Beispiel der Funktion SQR mitgeben. So berechnet SQR (25) die Wurzel aus 25, erzeugt also die Zahl 5.

Ebensowenig ignoriert wird eine Mitteilung, die Sie der Ausgabe-Funktion PEEK mitgeben. In diesem Fall wird die Information als Arbeitshinweis aufgefaßt: PEEK (2048) schaut in der Speicherzelle 2048 nach und sagt Ihnen dann, welchen Inhalt es gefunden hat.

Manchmal benötigt eine Funktion beide Arten von Information, so etwa die Funktion LEFT$. Sie muß zuerst wissen, was für ein String bearbeitet werden soll, und dann, wie lang der auszugebende String zu sein hat: LEFT$ ("Commodore",4) ergibt den Ausgabe-String "Comm". Mehrere Mitteilungen werden durch Kommata voneinander getrennt.

Natürlich können auch mehr als zwei Mitteilungen mitgegeben werden. Die Funktion MID$ benötigt, wie Sie wissen, im Normalfall drei: das Eingabedatum und zwei Arbeitshinweise (an welcher Stelle der Schnitt im Eingabe-String gemacht wer-

den und wie lang der Ausgabe-String sein soll). MID$("Commodore",4,5) ergibt also "modor".

Vielleicht sollte zum Schluß noch angemerkt werden, daß Art und Anzahl der Informationen, die einer Funktion mitgeteilt werden können, natürlich nicht dem Belieben des Programmierers anheimgestellt, sondern für jede Funktion vorgegeben sind. Dasselbe gilt für die Reihenfolge, in der die verschiedenen Informationen angegeben werden.

Nun weiß eine Funktion also alles, was sie wissen muß, um ihre Arbeit zur Zufriedenheit des Programmierers zu tun — aber wohin mit dem Ergebnis, das sie erzeugt?

1.3 Wohin mit dem Ergebnis?

Das Datum, das eine Funktion erzeugt, eine Zahl oder ein Text, muß ihr irgendwie abgenommen werden.

Man kann das Ergebnis auf den Bildschirm bringen, zum Beispiel PRINT INT (12.78); man kann es einer Variablen zuordnen, zum Beispiel B$ = MID$ ("Commodore",4,5); man kann es für einen Vergleich benutzen, zum Beispiel IF PEEK (214) = 24 THEN PRINT CHR$ (147).

Mit einem Wort: Die durch Funktionen erzeugten Zahlen und Zeichenketten werden genauso verwendet wie Zahlen und Zeichenketten. Genauso wenig wie der Computer die isolierte Zahl 3.5 oder den isolierten String »Commodore« verstehen würde, genauso wenig versteht er ein alleinstehendes INT (12.78). Man muß ihm immer sagen, was er mit einer Zahl, einem Text oder einer Funktion (das heißt mit dem Ergebnis, das sie erzeugt) tun soll.

Und wozu überhaupt Funktionen?

1.4 Der Daseinszweck von Funktionen

Je mehr Funktionen eine Programmiersprache zur Verfügung stellt, um so leichter ist das Programmieren. Funktionen nehmen Programmierarbeit ab. Lassen Sie mich dies an zwei Beispielen illustrieren.

Beispiel 1: Die Funktion ABS

Die Funktion ABS erzeugt aus einer Zahl deren absoluten Wert, zum Beispiel aus 5 oder —5 den Wert 5. Ein Commodore-Programmierer schreibt also einfach:

```
110 WERT = ABS (ZAHL)
```

Es gibt aber Computer, die diese Funktion nicht kennen; in diesem Fall muß der Programmierer eine spezielle Programmroutine schreiben, und das sieht dann zum Beispiel so aus (Bild 7).

```
100 Y = ZAHL
110 GOSUB 40000 : REM ABSOLUT-
    WERT
120 WERT = Y
130 PRINT WERT
140 END
-----------------------------------
40000 REM SUB: ABSOLUTWERT
40010      IF Y > = 0 THEN 40030
40020           Y = –Y
40030 REM ENDIF
40090 RETURN
```

Bild 7. Manche Funktionen müssen erst programmiert werden

Die armen Programmierer, die keinen Commodore haben! Andererseits sind wir Commodore-Programmierer arm dran, wenn wir zum Beispiel wissen wollen, ob die Zeichenkette B$ (»UTE«) in der Zeichenkette A$ (»COMPUTER«) enthalten ist, und wenn ja, ab welcher Stelle. Das geht ungefähr so (Bild 8).

```
100  TEXT$=A$
110  SUCH$=B$
120  LAENGE=LEN (TEXT$)–LEN
     (SUCH$) + 1
130  PO=0
140  GOSUB 40000:REM STRING
     SUCHEN
150...
-----------------------------------
40000  REM SUB: STRING SUCHEN
40010    FOR I=1 TO LAENGE
40020      X$=MID$ (TEXT$,I,
           LEN (SUCH$))
40030      IF X$=SUCH$ THEN
           PO=I : I=LAENGE
40040    NEXT I
40090  RETURN
```

Bild 8. Die INSTR-Funktion

Beispiel 2: Die Funktion INSTR

PO enthält den Wert 5, was bedeutet, daß »UTE« gefunden wurde und in Position 5 beginnt. Wenn PO = 0 bleibt, dann wurde der Suchstring gefunden.

Ach, gäbe es doch eine Funktion, die uns diese ganze Programmierarbeit abnimmt! Tatsächlich gibt es sie bei anderen Computern, und sie heißt meist INSTR. Wenn der Programmierer sie hat, dann schreibt er zum Beispiel einfach

```
140 PO = INSTR (A$,B$)
```

Daß Funktionen das Programmieren erleichtern, daß sie dazu Programme kürzer, übersichtlicher und lesbarer machen, daß sie schließlich den Programmablauf beschleunigen, dürfte nun leicht einleuchten.

Es ist deshalb kein Wunder, daß es viele Versuche gibt, das eingebaute Basic durch zusätzliche Funktionen zu erweitern; zum Beispiel durch einzelne Routinen in Maschinensprache, die eine einzelne erwünschte Funktion zur Verfügung stellen; oder durch spezielle Basic-Erweiterungen wie Simons Basic (für den C 64) oder Exbasic Level II oder Macro Basic.

Wer Maschinensprache beherrscht und seinen Computer kennt, kann sich seine Erweiterungen jeweils nach Bedarf selber anfertigen. Wer beides nicht beherrscht, braucht aber auch nicht zu verzweifeln; denn, wie zu Beginn angedeutet: Es gibt auch in Basic die Möglichkeit, Funktionen selber zu basteln. Und wenn auch die Möglichkeiten von Commodore-Basic nicht das sind, was sie vielleicht sein könnten, sie sind noch immer größer als der Anfänger im allgemeinen weiß.

2. Selbstdefinierte Funktionen

2.1 Die möglichen Typen

Im Commodore-Basic können Funktionen, die Zahlen ausgeben, selber gemacht werden, und zwar sowohl vom Typ Ausgabe-Funktion als auch vom Typ Eingabe-Ausgabe-Funktion. Die letzteren sind auf die Eingabe von Zahlen beschränkt.

Die Mitteilungsmöglichkeiten sind ebenfalls beschränkt: Es kann maximal eine Information mitgegeben werden, und sie muß vom Typ »Eingabedatum« sein. Daß nur Zahlen Eingabedaten sein können, wurde schon erwähnt.

2.2 Die Syntax

Selbstgestrickte Funktionen werden gekennzeichnet durch FN, dem ein individueller Name folgt, zum Beispiel

```
FN KREISUMFANG
```

Für den Namen gelten die üblichen Regeln für Variablennamen, das heißt, nur die beiden ersten Zeichen eines Namens werden berücksichtigt. Die obige Funktion kann also ebensogut folgendermaßen geschrieben werden:

```
FN KR
```

Dem Funktionnamen FN KR folgen dann, wie bei Funktionen üblich, die Klammern, die für die Mitteilung eines eventuellen Eingabedatums zur Verfügung stehen.

2.3 Die Benutzung

Selbstgestrickte Funktionen werden genauso benutzt wie vorgefertigte. So könnte eine Programmroutine, die einen Würfel simuliert, so aussehen:

```
10  PRINT FN WUERFEL (0)
20  GOTO 10
```

Die Funktion FN WUERFEL ist eine reine Ausgabe-Funktion, weshalb ich als Mitteilung die nichtssagende 0 gewählt habe.

Die im folgenden besprochene Funktion FN KREISUMFANG ist eine Eingabe-Ausgabe-Funktion, der jeweils der Radius mitgeteilt werden muß. Ein Programm könnte so aussehen:

```
10  INPUT "WELCHES IST DER
RADIUS"; RD
20  UM = FN KREISUMFANG (RD)
30  PRINT "UMFANG BETRAEGT" UM
```

Aber woher weiß der Computer eigentlich, daß in der Funktion FN WUERFEL die Mitteilung in der Klammer ignoriert werden soll und daß sie andererseits bei der Funktion FN KREISUMFANG den Radius meint? Und woher weiß der Computer überhaupt, wie er die Ausgabe-Zahl erzeugen soll?

2.4 Die Definition

Bevor Sie eine selbstgestrickte Funktion einsetzen können, müssen Sie sie erst einmal definieren. Das muß logischerweise **vor** der Benutzung geschehen, am besten gleich zu Angang des Programms.

Dazu steht der Befehl DEF zur Verfügung. Lassen Sie uns zuerst die Funktion FN KREISUMFANG definieren. Das geht so:

```
DEF FN KREISUMFANG (RD) = 2 * PI *
RD
```

Die Variable RD in der Klammer auf der linken Seite der »Gleichung« bezieht sich auf das Eingabedatum, also den Radius, der der Funktion mitgeteilt wird, wenn sie im Programm erscheint. Das Interessante dabei ist, daß der Radius nachher bei der Benutzung der Funktion keineswegs RD heißen muß. Man kann ihm jeden Namen geben, der einem in den Sinn kommt, und bei jedem Einsatz der Funktion kann man sich einen neuen einfallen lassen. Was allein wesentlich ist, das ist die Beziehung zwischen der Variablen RD in der Klammer auf der linken Seite und der Variablen RD auf der rechten Seite der Definitionsgleichung. Das bedeutet, daß man bei der Definition einer Funktion jede beliebige Variable benutzen kann. Es muß nur darauf geachtet werden, daß links und rechts dieselbe Variable benutzt wird. Die meisten Leute nehmen einfach X, was aber nicht in jedem Fall zu empfehlen ist. Ich komme darauf noch zurück.

Lassen Sie uns nun als nächstes die Ausgabe-Funktion FN WUERFEL definieren. Hier haben wir ein Problem: Wir geben ja dieser Funktion keine Information mit. Was also schreiben wir in die Klammer auf der linken Seite, die ja auf jeden Fall gefüllt werden muß? Nun, dieses Mal können wir ohne Bedenken X benutzen:

```
DEF FN WUERFEL (X) = INT (RND (1) *
6) + 1
```

Wie Sie sehen, erscheint auf der rechten Seite kein X. Aus dieser Tatsache schließt der Computer elektronenscharf, daß er bei der Benutzung dieser Funktion das, was in Klammern mitgeliefert wird, zu ignorieren hat.

In Commodore-Basic, so sahen wir, können wir einer selbstdefinierten Funktion also entweder gar keine Information mitgeben (Ausgabe-Funktion) oder ei-

nen Zahlenwert (Eingabe-Ausgabe-Funktion). Was aber, wenn wir zwei oder mehr Eingabedaten mitgeben möchten, sagen wir etwa bei einer Funktion FN RECHTECKINHALT? Nun, dies ist eben nicht möglich, aber wir können uns wie folgt aus der Affäre ziehen. Wir definieren zum Beispiel:

```
DEF FN RECHTECKINHALT (BREITE) =
BREITE * LAENGE
```

Wenn wir die Funktion später aufrufen, müssen wir einfach dafür sorgen, daß die Länge dem Programm an dieser Stelle schon bekannt ist:

```
100 LAENGE = 5 : PRINT FN RECHT-
ECKINHALT (BREITE)
```

Bei Funktionen dieser Art wird es vielleicht besonders deutlich: Wenn man eine Funktion benutzt, muß man sich darüber im klaren sein, welchen Eingabewert man ihr mitteilen muß. Deshalb ist es immer besser, bei der Definition einer Funktion »sprechende« Variablen zu benutzen statt des nichtssagenden X. Die folgende Definition verstehen Sie nach einem Jahr mit großer Wahrscheinlichkeit nicht mehr:

```
DEF FN A (X) = INT ((INT (X) + (X-INT
(X)) * .6) + 100 + .5) / 100
```

Das bedeutungsleere X hat seine Berechtigung allein in reinen Ausgabe-Funktionen wie FN WUERFEL, und das ist auch die Konvention, an die ich mich selber halte.

Übrigens — wenn Sie einen Fehler bei der Definition einer Funktion machen, kann es sein, daß dieser erst beim Einsatz der Funktion angezeigt wird. Manch einen Anfänger hat dies schon zur Verzweiflung gebracht. Angenommen, Sie erhalten einen SYNTAX ERROR IN 220; Sie schauen sich die Zeile an:

```
220 PRINT FN A (5)
```

Sie können absolut keinen Fehler erkennen. Klar, der Fehler liegt ja auch ganz woanders, in Zeile 10 nämlich, wo Sie folgendermaßen definiert hatten:

```
10 DEF FN A (T) = T * WAND
```

Ihre Variable WAND enthält AND, was in Variablen nicht vorkommen darf, weil es ein Basic-Wort ist. Wenn also ein Syntax Error angezeigt wird für eine Zeile, die einen Funktionsaufruf enthält, schauen Sie sich zuerst einmal die dazugehörige Definition an, bevor Sie den Computer an die Wand werfen.

3. Der Wert ist der springende Punkt

Eigenbaufunktionen erzeugen Zahlen aus Zahlen, oder richtiger: sie erzeugen Zahlenwerte aus Zahlenwerten. Es ist wichtig, daß man sich folgendes ganz klar macht: Worauf es ankommt, ist der **Wert**. In welcher Form der Wert ausgedrückt wird, ist hingegen unerheblich. Das kann eine Zahl sein, aber ebenso eine Variable, ein mathematischer Ausdruck oder sogar eine Funktion. Die Länge des Radius eines Kreises könnte also zum Beispiel in einem Basic-Programm (und also auch im Zusammenhang mit Funktionen) folgendermaßen erscheinen:

```
10.5
RD
DURCHMESSER / 2
LEN (LINIE$)
FN HM (Y)
```

Bei der Definition einer Funktion ist also alles erlaubt — solange das Ergebnis ein Zahlenwert ist. Zwei Beispiele sollen dies deutlich machen.

Beispiel 1: Text zentrieren

Wir wollen eine Funktion definieren, die berechnet, ab welcher Bildschirmspalte ein Text gedruckt werden soll, um in der Mitte des Bildschirms zu erscheinen.

Beim C 64 hat die Bildschirmzeile 40 Spalten, die Mitte liegt bei Spalte 20. Die halbe Zeichenkette muß also vor der Mitte, die andere Hälfte nach der Mitte gedruckt werden. Die Funktion kann folgendermaßen definiert und benutzt werden:

```
10 DEF FN MITTE (X) = 20 — LEN
(TEXT$) / 2
...
...
300 PRINT TAB(FN MITTE (0)); TEXT$
```

Unsere Funktion ist leider noch nicht vollkommen definiert, was deutlich wird, wenn die zu druckende Zeichenkette länger als die Bildschirmzeile ist, zum Beispiel 42 Zeichen lang. In diesem Fall erzeugt unsere Funktion ein negatives Ergebnis (—1), was TAB nicht verträgt, und was deshalb zu einer Fehlermeldung führt. Also müssen wir dafür sorgen, daß unsere Funktion nur dann rechnet, wenn die Zeichenkette gleich oder kleiner als 40 Zeichen lang ist. Wir könnten dieses Problem folgendermaßen lösen:

```
300 IF LEN (TEXT$) > 40 THEN PRINT
TEXT$: GOTO 320
310 PRINT TAB(FN MITTE (0)); TEXT$
320 ...
```

Es gibt jedoch eine sinnvollere Möglichkeit, die es erlaubt, die Entscheidung, ob die Funktion rechnet oder nicht, sozusagen von der Funktion selber treffen zu lassen.

Dies erreichen wir, indem wir einen logischen Ausdruck in die Definition einbauen. Der Ausdruck

```
(LEN (TEXT$) <= 40)
```

ergibt den Wert —1, wenn er wahr ist, das heißt wenn TEXT$ 40 Zeichen lang ist oder kürzer. Wenn TEXT$ länger ist, ergibt der Ausdruck den Wert 0.

Unsere Definition lautet also:

```
10 DEF FN MITTE (X) = (20 — LEN
(TEXT$) / 2) * ABS (LEN (TEXT$) <=
40)
```

FN MITTE ergibt den Wert 0, wenn TEXT$ länger als 40 Zeichen ist, und der Druck der Zeichenkette beginnt in der ersten Bildschirmspalte.

Das Beispiel zeigt, daß durchaus Strings in der Definition von Funktionen vorkommen können, wenn gewährleistet ist, daß das Endergebnis des Ausdrucks auf der rechten Seite der »Gleichung« ein Wert ist.

Beispiel 2: Kleinbuchstaben in Großbuchstaben verwandeln

Im Commodore-Basic kann die Definition einer Funktion höchstens eine Zeile lang sein. Diese Beschränkung läßt sich bis zu einem gewissen Grade umgehen, indem wir einfach mehrere Funktionen definieren und sie ineinander einbetten.

Es soll eine Funktion programmiert werden, mit deren Hilfe wir Kleinbuchstaben in Großbuchstaben verwandeln können. Wir benutzen dazu wieder logische Ausdrücke.

Wir gehen in zwei Schritten vor. Im ersten Schritt prüfen wir, ob das untersuchte Zeichen überhaupt ein Buchstabe ist. Das ist der Fall, wenn sein ASCII-Wert zwischen 65 und 93 oder 193 und 221 liegt:

```
10 DEF FN BU (Z) = (Z > 64 AND Z <
91) OR (Z > 192 AND Z < 219)
```

Das Ergebnis der Funktion FN BU ist —1, wenn der Ausdruck zutrifft. Das heißt, wenn die Funktion den Wert —1 erzeugt, dann ist das geprüfte Zeichen ein Buchstabe.

In Schritt 2 untersuchen wir, ob es sich um einen Kleinbuchstaben (dann muß er umgewandelt werden) oder einen Großbuchstaben handelt (dann darf er sich nicht verändern).

Kleinbuchstaben liegen zwischen 65 und 90. Das heißt, wenn der ASCII-Wert des geprüften Zeichens kleiner als 128 ist, haben wir es mit einem Kleinbuchstaben zu tun, und es muß 128 addiert werden. Das aber nur dann, wenn das Zeichen ein Buchstabe ist, das heißt, wenn FN BU den Wert —1 hat. Die Definition:

```
20 DEF FN KG (Z) = Z + (Z < 128) *
128 * FNBU (Z)
```

Die Funktion, die im Programm benutzt wird, ist natürlich lediglich die letztere; daß sie eine zweite Funktion enthält, braucht uns jetzt nicht mehr zu kümmern:

```
300 Z = ASC ("a")
310 Z = FN KG (Z)
320 PRINT CHR$ (Z)
```

Die Tatsache, daß es bei einer Funktion nur auf den Wert ankommt und nicht etwa darauf, daß dieser in Form einer Zahl ausgedrückt wird, betrifft nicht etwa nur die Definition von Funktionen, sondern auch ihre Anwendung. Was ich sagen will ist, daß es bei der einer Funktion mitgeteilten Information völlig gleichgültig ist, in welcher Form der mitgeteilte Wert ausgedrückt ist. Beispiele:

```
400 PRINT FN KG (65)
400 PRINT FN KG (A)
400 PRINT FN KG (A-128)
400 PRINT FN KG ("a")
400 PRINT FN KG (FN C (Y))
```

Ein Anwendungsbeispiel:

Um einen Namen, der in Kleinbuchstaben gespeichert ist, mit einem großen Anfangsbuchstaben zu versehen, könnte die Basic-Zeile 510 verwendet werden:

```
500 N$ = "commodore"
510 N$ = CHR$ (FN KG (ASC (N$))) +
MID$ (N$,2)
```

ASC ergibt den ASCII-Wert des ersten Buchstabens von »commodore«, also 67; FN KG addiert 128 und erzeugt den Wert 195; CHR$ ergibt das Zeichen »C«; dies wird mit Hilfe von MID$ mit »ommodore« verknüpft und der Variablen N$ zugeordnet, die also schließlich die Zeichenkette »Commodore« enthält. Die ganze Transaktion wird übrigens nur mit Hilfe von (vorgefertigten und selbstdedinierten) Funktionen durchgeführt.

4. Legen Sie sich eine Funktionen-Bibliothek an

Die Möglichkeiten, die das Commodore-Basic für die Herstellung selbstdefinierter Funktionen zur Verfügung stellt, sind beschränkt — andere Basic-Dialekte sind da oft großzügiger. Da lassen sich Funktionen definieren, die auf Zeichenketten wirken; da können Definitionen viele Zeilen lang sein; da kann mehr als eine Information mitgegeben werden — aber wir wollen uns den Mund nicht wässrig machen.

Auch unsere Funktionen sind ein »mächtiges« Werkzeug (oder richtig deutsch) ein leistungsfähiges Werkzeug und keineswegs auf die Verarbeitung mathematischer Formeln beschränkt, wie man häufig annimmt. Man muß die Möglichkeiten nur nutzen.

Es lohnt sich, eine individuelle Bibliothek von Funktionsdefinitionen anzulegen, die man je nach Bedarf in seine Programme einbaut. Funktionen ersparen, wenn sie einmal zur Verfügung stehen, viel Programmierarbeit.

Im Anhang habe ich ein paar Funktionsdefinitionen zusammengestellt, die ich immer wieder in Programmen benutze. Da ist nichts Weltbewegendes dabei, aber ich habe mir dadurch schon manch unnötige, weil sich wiederholende, Denkarbeit erspart. Und das ist es, worauf es ankommt.

Welches sind **Ihre** Lieblingsfunktionen?

(Prof.Dr. Leuschner/gk)

Anhang: Einige einfache Funktionen zur Anregung

1. Zufallszahl zwischen 1 und ENDZAHL

DEF FN RD (ENDZAHL) = INT (RND (1) * ENDZAHL) + 1
PRINT FN RD (6) würfelt eine Zahl zwischen 1 und 6.
Anmerkung: Damit der Zufallsgenerator mit einer zufälligen Zufallszahl anfängt, sollte man zu Beginn des Programms folgende Zeile einfügen:
X = RND (—RND (0))

2. Zufallszahl zwischen ANFZAHL und ENDZAHL

DEF FN ZUFALL (ENDZAHL)
= INT(RND (1) * (ENDZAHL — ANFZAHL)) + ANFZAHL +1
ANFZAHL = 65 : PRINT CHR$ (FN ZUFALL (90)) erzeugt einen zufälligen Kleinbuchstaben.

3. Kommazahl zu Ganzzahl aufrunden

DEF FN AUFRUNDEN (ZAHL) = — INT (— ZAHL)
PRINT FN AUFRUNDEN (23.05) ergibt 24.
Anmerkung: Zum Abrunden benutzt man die einfache INT-Funktion.

4. Zahl mit festgelegter Anzahl von Nachkommastellen mit Rundung

DEF FN KOMMA (ZAHL)
= INT (ZAHL * 10 ↑ NACHKOMMA + .5) / 10 ↑ NACHKOMMA
NACHKOMMA = 2: PRINT FN KOMMA (25/6) ergibt 4.17.

5. Zahl mit festgelegter Anzahl signifikanter Ziffern (2 Funktionen)

DEF FN SG (ZAHL)
= 10 ↑ (1 — ZIFFERN + INT (LOG (ABS (ZAHL)) / LOG (10)))
DEF FN SIGNI (ZAHL)
= INT (ZAHL / FN SG (ZAHL) + .5) * FN SG (ZAHL)
ZIFFERN = 4 : PRINT FN SIGNI (ZAHL) ergibt bei ZAHL = 1234567 die Zahl 1235000, bei ZAHL = 12.345 die Zahl 12.35, etc.

6. Zahlen in einer Spalte drucken (PRINT US ING)

DEF FN US ING (ZAHL)
= SPALTE—ABS((ZAHL > 10)+(ZAHL > 100)+(ZAHL > 1000) + (ZAHL > 10↑4)+(ZAHL > 10↑5)+(ZAHL > 10↑6))
SPALTE = 20 : PRINT TAB(FN US ING (ZAHL)); ZAHL druckt Zahlen bis zu einer Million richtig als Kolonne.
Anmerkung: Wenn Sie die Variablen abkürzen, paßt die Definition in eine Zeile. Zwischen US und ING **muß** eine Leerstelle stehen!

7. Ungerade oder gerade Zahl?

DEF FN ODD (ZAHL) = ZAHL AND 1
PRINT FN ODD (25) ergibt den Wert 1, da 25 eine ungerade Zahl ist. Gerade Zahlen ergeben 0.

8. Modulus (Rest bei einer Division)

DEF FN MOD (ZAHL)
= INT (((ZAHL / TEILER — INT (ZAHL / TEILER)) * TEILER) + .5)
TEILER = 6: PRINT FN MOD (25) ergibt den Divisionsrest 1.

9. Uhrzeit dezimal darstellen

DEF FN DEZUHR (HR)
= INT ((INT (HR) + (HR — INT (HR)) / .6) * 100 + .5) / 100
H = FN DEZUHR (17.30) ergibt den Dezimalwert 17.5, mit dem man dann normal rechnen kann.

10. Dezimal ausgedrückte Uhrzeit als normale Uhrzeit darstellen

DEF FN UHR (DEZZT)
= INT ((INT (DEZZT) + (DEZZT — INT (DEZZT)) * .6) * 100 + .5) / 100
PRINT FN UHR (17.25) ergibt die normale Uhrzeit 17.15 Uhr.

11. ASCII-Code eines Zeichens in den Bildschirm-Code umwandeln

DEF FN SCREEN (AS) = (AS AND 128) / 2 OR (AS AND 63)
POKE 1024, FN SCREEN (ASC("A")) POKEt in die linke obere Ecke des C 64-Bildschirms den Buchstaben A.

12. Inhalt einer Bildschirmspeicherzelle lesen

DEF FN CRT (SPALTE)
= PEEK (1024 + (ZEILE — 1) * 40 + (SPALTE — 1))
ZEILE = 5 : PRINT FN CRT (20) ergibt den Inhalt der Speicherzelle der 20. Spalte in der 5. Zeile.

13. Exklusives Oder: Von zwei Bedingungen darf nur eine zutreffen.

Beispiel: Wenn Tante Amalie allein kommt, oder wenn Onkel Otto allein kommt, dann gehen wir auch zum Fest. Wenn aber beide kommen, dann gibt's Streit zwischen den beiden, also bleiben wir daheim. Wenn keiner von den beiden kommt, wird's langweilig, dann bleiben wir auch daheim. (Drei Funktionen werden dafür definiert.)
DEF FN B1 (X) = (A = AWERT) oder sonst eine Bedingung
DEF FN B2 (X) = (B = BWERT) oder sonst eine Bedingung
DEF FN EO R (X)
= ABS ((FN B1 (0) AND FN B2 (0)) < > (FN B1 (0) OR FN B2 (0)))
Als einander ausschließende Bedingungen seien A$ = "Amalie" und B$ = "Otto" für die beiden ersten Funktionen definiert worden. Dann ergibt die Funktion FN EO R den Wert 1 zum Beispiel bei folgender Situation:
A$ = "Amalie" : B$ = "Emilia" : PRINT FN EO R (0)
Anmerkung: Der Name der Funktion ist EO R, weil EOR das Basic-Wort OR enthielte, was zu einer Fehlermeldung führen würde.

14. ASCII-Wert eines Zeichens innerhalb eines Strings bestimmen

DEF FN AS C (PS) = ASC (MID$ (S$,PS,1))
S$ = "Commodore": PRINT FN AS C (6) ergibt den ASCII-Wert von »d«, also 68.

15. Einen Teilstring aus einem String »herausschneiden« und dessen Wert bestimmen

DEF FN WERT (PS) = VAL (MID$ (S$,PS,LAENGE))
S$ = "028255063": LAENGE = 3: PRINT FN WERT (4) ergibt den Wert 255.