Comal — eine Einführung (Teil 2)

Nachdem wir in der ersten Folge die Grundkenntnisse erworben haben, um mit Comal umgehen zu können, wollen wir jetzt die ersten kleinen Programme erstellen.

Die Verwandtschaft zwischen Basic und Comal wird uns dabei den Einstieg in die einfache Programmierung sehr erleichtern. Wir brauchen uns also nicht wie beim Erlernen anderer Programmiersprachen mit völlig neuen Befehlsstrukturen herumzuschlagen, sondern kommen weitgehend mit unserem Basic-Wissen aus. Man kann sagen, daß zwischen Basic und Comal eine Art Aufwärtskompatibilität besteht. Basic-Programme lassen sich mit minimalen Änderungen in Comal übersetzen; in der anderen Richtung können allerdings größere Schwierigkeiten auftreten. So sind zwar alle numerischen Funktionen und Operatoren von Basic auch in Comal vorhanden, es gibt jedoch zusätzlich einige Comal-Funktionen, die in Basic nicht vorkommen (Tabelle 1), beispielsweise MOD (Restbildung bei Division). Auch die Datenein- und Ausgabe ist in Comal um einiges komfortabler.

Betrachten wir einmal das folgende kleine Beispiel einer Mehrwertsteuer-Berechnung, zunächst in Basic:

```
10 rem Mehrwertsteuer
20 input "Betrag";b
30 s = b * 0.14
40 b = b + s
50 print "Mehrwertsteuer:";s
60 print "Gesamtbetrag:";b
```

Dies ist zugegebenermaßen ein sehr einfaches Beispiel, und man hätte es auch gut in einer Zeile unterbringen können. Aber sehen wir uns dieses Programm doch einmal in Comal an:

```
10 // Mehrwertsteuer
20 input "Betrag ?" : betrag
30 mwert := betrag * 0.14
40 betrag := betrag + mwert
50 print "Mehrwertsteuer:"
mwert
60 print "Gesamtbetrag:" , be-
trag
```

Man erkennt sofort die sehr große Ähnlichkeit beider Programme. Allerdings gibt es auch einige mehr oder weniger auffällige Unterschiede. Zunächst versteht Comal auch lange Variablennamen, wodurch die Programme generell übersichtlicher werden. Als nächstes fällt die Verwendung von »:=« für die Wertzuweisungen auf. Bei der Eingabe braucht man allerdings nur ein Gleichheitszeichen zu schreiben. Comal merkt dann, was gemeint ist und wandelt das Gleichheitszeichen in »:=« um.

Bei genauerem Hinsehen entdeckt man schließlich noch die Verwendung des Doppelpunktes statt eines Semikolons bei der INPUT-Anweisung und die Verwendung des Kommas statt eines Semikolons bei den PRINT-Befehlen.

Befassen wir uns zunächst mit dem INPUT-Befehl. Genau wie in Basic werden damit Daten vom Benutzer erfragt und an die im Befehl angegebenen Variablen zugewiesen. Mehrere Variablen können dabei durch Komma getrennt eingegeben werden.

Enthält die INPUT-Anweisung nur eine Variablenliste und keinen Text, dann erscheint beim Programmlauf ein Fragezeichen, um dem Benutzer mitzuteilen, daß jetzt eine Eingabe erwartet wird.

Im Unterschied zu Basic kann hinter INPUT nicht nur ein Text in Anführungszeichen stehen, sondern auch ein beliebiger Stringausdruck. Hinter diesem Stringausdruck muß ein Doppelpunkt folgen, und dahinter wiederum die Liste der einzulesenden Variablen. In unserem kleinen Beispiel könnten wir also die Zeile 20 ersetzen durch:

```
20 frage$ := "Betrag"
25 input frage$ : betrag
```

Wir müssen allerdings beachten, daß Strings in Comal dimensioniert werden müssen, da vor dem eigentlichen Programmlauf die Adressen aller Variablen festgelegt werden (siehe Teil 1). Um die Adressen von Stringvariablen aber festlegen zu können, muß Comal deren maximale Länge kennen. Bevor wir also die Stringvariable »frage$« das erste Mal benutzen können, muß eine Dimensionie-

Funktion	Bedeutung	Beispiel	Basic-Äquivalent
ABS	Absolutwert	A := ABS(-5)	ABS
AND	Logisches UND	IF A=3 AND B=4 THEN...	AND
ATN	Arcustangens	X := ATG(PHI)	ATN
CHR$	Zeichen	PRINT CHR$(13)	CHR$
COS	Cosinus	X := COS(ALPHA)	COS
DIV	Integerdivision	X := A DIV B	INT(A/B)
EXP	Exponentialfunktion	E := EXP(1)	EXP
IN	Teilstringsuche	IF A$ IN B$ THEN...	(nicht vorhanden)
INT	Ganzzahliger Anteil	N := INT(1.5)	INT
LEN	Stringlänge	L := LEN(X$)	LEN
LOG	Logarithmus	X := LOG(A)	LOG
MOD	Restfunktion	X := A MOD B	A-INT(A/B)*B
NOT	Logisch NICHT	IF NOT FLAG THEN...	NOT
OR	Logisch ODER	IF FLAG OR A>3 THEN...	OR
ORD	ASCII-Wert	PRINT ORD("A")	ASC
PEEK	Speicher lesen	PRINT PEEK(828)	PEEK
RND	Zufallszahl	N := RND(1)	RND
SGN	Vorzeichen-Funktion	S := SGN(X)	SGN
SIN	Sinus	X := SIN(ALPHA)	SIN
SQR	Quadratwurzel	PRINT SQR(2)	SQR
TAN	Tangens	T := TAN(ALPHA)	TAN

Tabelle 1. Comal-Funktionen und Operatoren

Comal – Teil 2

Anweisung	Bedeutung
CASE <Variable> OF	Beginn einer CASE-Anweisung
WHEN <Wert>	Vergleicht <Variable> mit <Wert>; bei Übereinstimmung wird der darauffolgende Programmabschnitt ausgeführt
OTHERWISE	Der folgende Abschnitt wird ausgeführt, wenn keine WHEN-Anweisung zutraf (optional)
ENDCASE	Abschluß einer CASE-Anweisung
IF <Bedingung> THEN	Beginn einer IF-Anweisung. Bei wahrer <Bedingung> wird der Teil nach THEN ausgeführt
ELIF <Bedingung> THEN	Falls die vorhergehende <Bedingung> nicht zutraf, wird die angegebene neue <Bedingung> getestet (optional)
ELSE	Falls keine vorhergehende <Bedingung> zutraf, wird der Programmteil nach ELSE ausgeführt (optional)
ENDIF	Kennzeichnet das Ende eines IF-Blocks (entfällt beim einzeiligen IF)
FOR <Laufvariable> := <Anfang> TO <Ende> DO	Beginn einer Zählschleife, bei der die <Laufvariable> von <Anfang> bis <Ende> hochgezählt wird.
STEP <Schrittweite>	Angabe einer Schrittweite zum Hochzählen der <Laufvariable> (optional)
ENDFOR	Ende einer FOR-Schleife. Statt ENDFOR kann auch NEXT eingegeben werden.
REPEAT	Beginn einer REPEAT..UNTIL-Schleife
UNTIL <Bedingung>	Bei erfüllter Bedingung wird die Schleife beendet, sonst nochmals durchlaufen.
WHILE <Bedingung>	Der nachfolgende Programmteil wird nur durchlaufen, wenn die <Bedingung> erfüllt ist, sonst wird das Programm hinter END WHILE fortgesetzt.
ENDWHILE	Ende einer WHILE...END WHILE-Schleife, springt stets zurück nach WHILE.
LOOP	Beginn einer Endlosschleife. Der Programmteil zwischen LOOP und ENDLOOP wird ständig wiederholt.
EXIT	Ermöglicht das Verlassen einer LOOP...ENDLOOP-Schleife.
ENDLOOP	Erzeugt stets einen Rücksprung nach LOOP.

Tabelle 2. Kontrollstrukturen in Comal. LOOP...ENDLOOP ist in der Version 0.14 noch nicht implementiert.

rung erfolgen. Dies geschieht, indem wir noch eine weitere Zeile einfügen:

```
15 dim frage$ of 20
```

Durch diese Anweisung wird Speicherplatz für eine Stringvariable »frage$« mit einer maximalen Länge von 20 Zeichen reserviert. Die Stringlänge ist in Comal übrigens grundsätzlich nur durch den Speicherplatz begrenzt. Nach »dim text$ of 3000« beispielsweise kann text$ bis zu 3000 Zeichen enthalten.

Doch kommen wir nun zur Print-Anweisung, die im wesentlichen analog zu Basic ist, darüber hinaus aber einige zusätzliche Feinheiten kennt.

Formatierte Ausgabe ohne Probleme

Die einzelnen zu druckenden Argumente (numerische oder Stringausdrücke) werden grundsätzlich durch Komma getrennt. Wünscht man die Ausgabe an einer bestimmten Tabulatorstelle, kann man wie in Basic die TAB(n)-Funktion verwenden. Die durch Komma getrennten Ausdrücke werden normalerweise unmittelbar nebeneinander gedruckt — so, als hätte man in Basic ein Semikolon verwendet. Zum Drucken von Tabellen ist das natürlich nicht besonders sinnvoll. Es ist jedoch mit dem Comal-Befehl »ZONE« möglich, die Spaltenbreite für die Print-Anweisung festzulegen. Mit ZONE 10 erhält man eine Spaltenbreite wie bei Basic.

Zur weiteren Formatierung von Zahlenausgaben kann man »PRINT USING« verwenden. Hinter »USING« muß dabei ein String stehen, der das Ausgabeformat bestimmt. Für jede Ziffernstelle der auszudruckenden Zahl steht in diesem String ein Nummernzeichen »#«. Außerdem kann die Position des Dezimalpunktes angegeben werden. In unserem kleinen Mehrwertsteuer-Programm wäre es zum Beispiel sinnvoll, die Geldbeträge mit zwei Nachkommastellen auszugeben. Dazu ersetzen wir die Zeilen 50 und 60 durch die folgenden vier Comal-Zeilen:

```
50 print »Mehrwertsteuer:«,
55 print using "#####.##":
mwert
60 print "Gesamtbetrag:",
65 print using "#####.##":
betrag
```

Jetzt werden die Beträge rechtsbündig mit fünf Stellen vor und zwei Stellen nach dem Komma (oder besser Dezimalpunkt) ausgegeben. Die beiden zusätzlichen PRINT-Befehle waren nötig, da hinter dem Doppelpunkt im Anschluß an das »USING«-

Statement nur noch numerische Parameter folgen dürfen. Die Konstruktion »PRINT USING"###": "Hallo",5« führt zu einer Fehlermeldung, weil »USING« sich an dem String "Hallo" — etwas salopp gesagt — die Zähne ausbeißt.

Der zur »USING«-Anweisung gehörende Formatierungsstring darf übrigens auch andere Zeichen enthalten. Probieren Sie doch einmal folgende Zeile (im Direktmodus) aus:

```
PRINT USING "DM ###.##": 12.6
```

Experimentieren Sie ruhig einmal mit diesen Formatierungsmöglichkeiten, auch unter Verwendung des ZONE-Befehls.

Strukturiert programmieren

Jede höhere Programmiersprache kennt sogenannte »Kontrollstrukturen«, um den Programmablauf in Abhängigkeit von bestimmten Bedingungen beeinflussen zu können. In Basic gibt es zwei derartige Strukturen, nämlich die Wiederholung mit FOR...NEXT und die Bedingungsabfrage mit IF...THEN. Die Realisierung der IF-Abfrage in Basic hat dabei zwei entscheidende Nachteile. Zum einen fehlt, zumindest im Commodore-Basic, die Angabe einer Alternative (ELSE-Teil einer IF-Anweisung), zum anderen ist die Beschränkung auf eine Zeile in vielen Fällen sehr störend. Man behilft sich in Basic dann mehr schlecht als recht mit GOTO-Sprüngen vor, nach und innerhalb von IF-Anweisungen, was die Übersichtlichkeit eines Programms nicht gerade fördert.

Comal unterstützt nun strukturiertes Programmieren durch eine Vielzahl von Strukturbefehlen (Tabelle 2). Zur Bildung von Programmschleifen stehen neben der von Basic bekannten FOR...NEXT-Struktur noch WHILE...ENDWHILE und REPEAT...UNTIL zur Verfügung. Am einfachsten davon ist die Schleife mit REPEAT...UNTIL (»Wiederhole ... bis«). Hinter UNTIL muß eine Bedingung stehen. Ist die Bedingung nicht erfüllt, wird die Schleife ab REPEAT wiederholt, und zwar so oft, bis entweder die Bedingung wahr wird, oder bis der entnervte Programmierer die STOP-Taste drückt. Der folgende Vierzeiler wartet zum Beispiel, bis die Taste »X« gedrückt wird.

```
10 DIM EINGABE$ OF 1
20 REPEAT
30 EINGABE$ := KEY$
40 UNTIL EINGABE$ = "X"
```

Die Systemvariable »KEY$« enthält stets die gerade gedrückte Taste. Ist keine Taste gedrückt, ist KEY$ = CHR$(0).

WHILE...ENDWHILE funktioniert ähnlich wie REPEAT...UNTIL, nur steht hier die Bedingung direkt hinter WHILE, wird also überprüft, bevor die Schleife zum ersten Mal durchlaufen wird. Dadurch wird eine WHILE-Schleife möglicherweise nie durchlaufen, nämlich dann, wenn die Bedingung von Anfang an schon nicht erfüllt war. In diesem Fall werden alle Befehle zwischen WHILE und ENDWHILE übersprungen und das Programm nach ENDWHILE normal fortgesetzt.

Das genaue Format der WHILE-Schleife ist »WHILE ⟨Bedingung⟩ DO ⟨Anweisungen⟩ ENDWHILE«.

Mit dem »DO« hat es dabei eine besondere Bewandtnis. Steht hinter dem »DO« in der gleichen Zeile eine Anweisung, dann faßt Comal dies als eine einzeilige WHILE-Schleife auf. In diesem Fall darf kein ENDWHILE mehr folgen, sonst gibt es einen »Fehler in der Programmstruktur«. Mit dieser Kurzform einer WHILE-Schleife und dem Comal-Befehl »NULL« läßt sich sehr elegant eine Warteschleife auf einen Tastendruck aufbauen:

```
10 WHILE KEY$ = CHR$(0) DO NULL
```

Die Anweisung NULL ist eine »Dummy«-Anweisung mit der speziellen Eigenschaft, nichts zu bewirken. Die obige Zeile könnte man also etwas frei übersetzen mit »solange keine Taste gedrückt, tue nichts«.

Neben diesen beiden Schleifenstrukturen gibt es natürlich noch die Zählschleife FOR...TO... ENDFOR, die völlig analog zur FOR...NEXT-Schleife in Basic arbeitet, so daß die Besprechung der Arbeitsweise entbehrlich erscheint.

Entscheidungen fällen

In praktisch jedem Programm müssen logische Entscheidungen, meist sogar in großer Anzahl, getroffen werden. Comal stellt dafür eine sehr mächtige IF...THEN...ELIF...ELSE...ENDIF-Konstruktion zur Verfügung die sich in der Regel über mehrere Zeilen erstreckt und ganze Programmblöcke umfassen kann. Daneben gibt es — wie bei »WHILE« — noch eine einzeilige Kurzform. Diese Kurzform besteht einfach darin, daß hinter dem »THEN« in der gleichen Zeile noch ein Befehl folgt. Das funktioniert dann völlig analog zu Basic, nur mit dem Unterschied, daß in Basic noch weitere Befehle, jeweils durch Doppelpunkt getrennt, in der gleichen Zeile folgen dürfen. Für derartige Fälle — und für Fälle, die man in Basic so gar nicht lösen kann — wird in Comal die mehrteilige Form der IF-Anweisung verwendet.

Bei dieser Form muß die Zeile nach dem »THEN« beendet werden. Dann werden, falls die Bedingung hinter dem IF erfüllt ist, alle folgenden Programmzeilen bis zum Ende der IF-Anweisung ausgeführt. Eine mehrzeilige IF-Anweisung muß immer mit dem Schlüsselwort »ENDIF« beendet werden. Außer »ENDIF« darf die entsprechende Zeile allenfalls noch einen Kommentar (//) enthalten. War die IF-Bedingung nicht erfüllt, dann wird das Programm in der auf das »ENDIF«-Statement folgenden Zeile fortgesetzt.

Doch damit sind wir noch längst nicht am Ende. Die IF-Anweisung kann auch um einen »ELSE«-Teil erweitert werden und hat dann das folgende Format:

IF ⟨Bedingung⟩ THEN ⟨Teil 1⟩ ELSE ⟨Teil 2⟩ ENDIF.

Der Programmteil ⟨Teil 1⟩ wird ausgeführt, falls die ⟨Bedingung⟩ erfüllt war, sonst wird ⟨Teil 2⟩ ausgeführt. Jeder dieser beiden Teile ist ein völlig unabhängiges Programmstück und kann seinerseits auch wieder IF-Abfragen enthalten.

Will man gleich mehrere verschiedene Bedingungen testen, dann kann man das Comal-Schlüsselwort »ELIF« verwenden. »ELIF« ist eine Abkürzung für »ELSE IF« und hat auch die gleiche Wirkung, nur mit dem Unterschied, daß keine zweite IF-Anweisung (zu der dann auch ein zweites ENDIF gehören müßte) eröffnet wird. Das folgende Beispielprogramm testet eine einzugebende Zahl auf bestimmte Werte:

```
10 INPUT "ZAHL ?": ZAHL
20 IF ZAHL = 1 THEN
30 PRINT "EINS"
40 ELIF ZAHL = 2 THEN
50 PRINT "Zwei"
60 ELSE
70 PRINT "WEDER EINS NOCH ZWEI"
80 ENDIF
```

Ich erspare mir — und Ihnen — an dieser Stelle, ein entsprechendes Basic-Programm vorzustellen (GOTO, GOTO, ...).

Für den Fall, daß die zu testenden Bedingungen durch einen Variablenwert dargestellt werden können, ist die CASE-Anweisung vorgesehen. Die Wirkungsweise wird wohl am besten klar, wenn wir unser Beispiel zur IF-Anweisung auf die CASE-Konstruktion umschreiben:

```
10 INPUT "ZAHL ?": ZAHL
20 CASE ZAHL OF
30 WHEN 1
40 PRINT "EINS"
50 WHEN 2
60 PRINT "ZWEI"
70 OTHERWISE
80 PRINT "WEDER EINS NOCH ZWEI"
90 ENDCASE
```

In der Kopfzeile einer CASE-Anweisung wird also eine Variable angegeben, gefolgt vom Schlüsselwort »OF«.

Dann folgen beliebig viele Zeilen mit »WHEN«-Konstruktionen. Hinter WHEN ist immer ein Wert angegeben, der bei der Programmausführung mit dem aktuellen Wert der CASE-Variablen verglichen wird. Wird eine Übereinstimmung festgestellt, dann wird der Programmteil hinter der entsprechenden WHEN-Anweisung bis zum folgenden WHEN ausgeführt. Trifft keine WHEN-Bedingung zu, dann wird der Programmteil hinter OTHERWISE ausgeführt. OTHERWISE ist optional und muß nicht vorhanden sein. Trifft keine WHEN-Bedingung zu und ist kein OTHERWISE vorhanden, dann wird das Programm hinter ENDCASE normal fortgesetzt.

Kommen wir nun noch, sowohl last als auch least, zu einem Befehl, den hartgesottene Spaghetticode-Programmierer schon vermißt haben mögen. Gemeint ist die GOTO-Anweisung, die, obschon weitgehend entbehrlich, auch in Comal noch für Spezialfälle zur Verfügung steht. In Comal wird allerdings nicht zu bestimmten Zeilennummern gesprungen, sondern ein GOTO bezieht sich immer auf ein LABEL. Ein Label ist einfach irgendein Name, wie er auch als Variablenname verwendet werden könnte, gefolgt von einem Doppelpunkt. Vor diesem Namen kann, muß aber nicht, das Schlüsselwort LABEL stehen. Die betreffende Zeile darf nur dieses Label und keine weiteren Befehle enthalten. Wer also von GOTO wirklich nicht loskommt, kann das Warten auf einen Tastendruck auch folgendermaßen programmieren (nicht zur Nachahmung empfohlen):

```
5 DIM A$ OF 1
10 LABEL WARTEN:
20 A$ := KEY$
30 IF A$ = CHR$(0) THEN GOTO WARTEN
```

Es sollte auch nicht unerwähnt bleiben, daß man nicht in FOR...ENDFOR-Schleifen, Funktionen und Prozeduren hinein oder aus ihnen hinaus springen sollte. Mit »Funktionen und Prozeduren« ist im übrigen bereits das Stichwort für die nächste Folge gegeben. Bis dahin können Sie sich ja vielleicht die Zeit damit vertreiben, Ihre Basic-Programme in Comal umzuschreiben, und zwar selbstverständlich ohne GOTO! (ev)