Pa leistungsfähiger

Für den Commodore 64 gibt es bereits einige Versionen der Programmiersprache Pascal. Dieser Bericht stellt wichtige Elemente der Sprache vor und vergleicht sie mit äquivalenten Basic-Lösungen.

```
PROGRAM WURZEL(INPUT,OUTPUT);
VAR z : INTEGER;

BEGIN
  WRITE('? '); READLN(z);
  WHILE z >= 0 DO
    BEGIN
      WRITELN (SQRT(z));
      WRITE('? '); READLN(z);
    END
END.
```

Listing 1. Programm mit WHILE-Schleife

```
PROGRAM WURZEL(INPUT,OUTPUT);
VAR z : INTEGER;

BEGIN
  REPEAT
    WRITE('? '); READLN(z);
    IF z >= 0 THEN
      WRITELN (SQRT(z))
  UNTIL z < 0
END.
```

Listing 2. Programm mit REPEAT-Schleife

```
10 REM          PROGRAMM WURZEL
20 REM          VARIABLE Z%
30 INPUTZ%
40 PRINT Z%
50 IF Z% >= 0 THEN PRINT SQR(Z):GOTO 30
60 END
```

Listing 3. Basic-Programm mit derselben Leistung wie die Pascal-Programme aus Listing 1 und 2.

Wie lange besitzen Sie schon den Commodore 64? Wenn Sie ihn wie ich schon längere Zeit und einige Gehversuche in Basic schon hinter sich haben, die Phase der Spielsucht und des Programme-Scheffelns überstanden haben, dann werden Sie sich wohl wieder dem Programmieren in Basic zuwenden, nur diesmal ungleich intensiver. Jetzt stellt sich jedoch heraus: das eingebaute Basic hält den gewachsenen Anforderungen nicht stand.

Eine neue Sprache, gut, aber welche? Da gibt's ja eine Unmenge davon, zum Beispiel Forth, Cobol, Algol, Fortran, Pascal, Modula und Assembler, die nur »ehrfürchtig« genannt wird, da Assembler angeblich nur von absoluten Spitzenkönnern beherrscht wird. Zu den Könnern zählt man sich im allgemeinen noch nicht und läßt eben die Maschinensprache beiseite (weshalb eigentlich?) und sucht sich eine sogenannte »höhere« Programmiersprache. Die Wahl dürfte wahrscheinlich auf Pascal fallen, weil Forth zu fremd, Fortran zu antiquiert, Cobol zu »geschäftig«, Algol zu wissenschaftlich und Modula zu neu ist. Nun, Pascal hat schon einiges zu bieten, was Basic nicht hat. Die Schlagwörter sind:

— Strukturierte Programmierung
— Blockorientierte Programmierung
— Operatoren auf Mengen
— Lokale Parameter
— Möglichkeit für Rekursionen
— eigene Typendefinitionen
— Records etc.

Über Schlagworte läßt sich bekanntlich streiten, aber ich möchte beweisen, daß sie für Pascal wirklich zutreffen.

Niklaus Wirth, der diese Sprache aus den beiden Hauptlinien Algol 68 und Fortran beziehungsweise PL/1 1971 an der ETH Zürich entwickelte, hat sie Pascal genannt, zu Ehren des französischen Mathematikers und Philosophen Blaise Pascal (zirka 1650). Man kann deshalb von einem Wirth-Standard im Gegensatz zum UCSD-Pascal sprechen. Das UCSD-Pascal entstand in Kalifornien und enthält nun auch den Typ »STRING« sowie Operatoren auf diesen Typ wie man es von Basic her gewöhnt ist. Ansonsten ist der Unterschied nicht annähernd so groß, wie derjenige zwischen Basicdialekten.

Pascal ist wegen seiner Klarheit im Aufbau und seiner enormen Leistungsfähigkeit zur wichtigsten, wissenschaftlichen Programmiersprache geworden.

Ein äußerlicher Unterschied gegenüber Basic besteht darin, daß Pascal vollkommen formatfrei ist, das heißt es gibt keine Zeilennummerierung. Man darf daher eine Befehlssequenz (statements) über eine Bildschirmzeile hinaus auf der nächsten fortsetzen, denn, als gültigen Statementstrenner gibt es nur das Semikolon (;). Pascal kennt daher den Befehl »GOTO/GOSUB Zeilennummer« nicht. Aber dafür ist Pascal sehr stark blockorientiert. Ein Block ist im einfachsten Fall entweder ein Befehl oder eine mit »BEGIN« und »END« umschlossene Befehlssequenz. Andere Blocks sind »PROCEDURE«, eine Art Unterprogramm, und »FUNCTION«, auch eine Art Unterprogramm, aber mit dem Unterschied, daß man einer »FUNCTION« einen Wert zuweisen kann. Darauf komme ich später zurück. Ein Beispiel dazu:
FOR i := 10 DOWNTO 3 DO statement

Diesem Statement entspricht in Basic die folgende Sequenz:
FOR i = 10 TO 3 STEP —1: befehle: NEXT

Diese ganze Schleife ist ein Statement. Obwohl sie mehrere Befehlswörter enthält, muß man sie nicht mit »BEGIN« und »END« umklammern. Ebenfalls als ein Statement gilt eine Wertzuweisung der Form:
i := i + 1;

Der Operator ':=' weist der links davon stehenden Variablen den Wert des rechten Ausdrucks (expression) zu. Er ersetzt nicht den Vergleichsoperator '='. Diesen braucht man in Pascal korrekterweise nur für Vergleiche wie »IF i = 5 THEN....«

Wenn ich schon bei Vergleichsstatements bin, kann ich die drei ebenfalls »mehrwortigen« Statements besprechen, die Pascal für bedingte Abarbeitung eines Blocks besitzt, wobei eine Bedingung (condition) von der Form ist »variable =,⟨ ⟩,⟨=,=⟩,⟨,⟩ expression«:

— IF condition THEN statements ELSE statements
— WHILE condition DO statement
— REPEAT statements UNTIL condition

Das erste Statement existiert fast in identischer Form in Basic, außer der Tatsache, daß »ELSE« im Basic des Commodore 64 natürlich fehlt. Die unteren beiden haben einen Schleifencharakter. Wofür nun aber zwei Schleifen, die sich doch fast nicht unterscheiden? Nun, bei näherer Betrachtung finden Sie sehr wohl einen interessanten Unterschied: Die »WHILE«-Schleife testet eine Bedingung bevor eine Befehlssequenz ausgeführt wird (Solange wie...wiederhole...), die »REPEAT«-Schleife hingegen testet erst nach der Ausführung (Wiederhole...bis...). Sie werden vielleicht sagen, das sei doch nun wirklich nur Haarspalterei. Mitnichten! In den folgenden zwei Beispielen sehen Sie, wie man damit elegant iterative Probleme mit Bedingungen programmieren kann. Kümmern Sie sich vorerst nicht um

Pascal-

und eleganter als Basic (Teil 1)

die anderen Statements, die in den beiden Prgrämmchen sonst noch vorkommen, sondern konzentrieren Sie sich bitte nur auf die Schleifen.

Beide Programme (Listing 1 und Listing 2) berechnen die Quadratwurzel einer Zahl:

Die unterstrichenen Worte sind reservierte »Steuerworte« in Pascal, die nicht anders gebraucht werden dürfen. Die übrigen Worte sind Standardprozeduren. Man erkennt sie auch daran, daß sie ein oder mehrere Argumente in Klammern annehmen können. Ein Statement kann aus Steuerworten und Prozeduren bestehen, was man leicht erkennen kann, weil nach 'WHILE'..'DO' bekanntlich nur ein Statement folgen darf, denn sonst müßte man ein Ende der »WHILE«-Schleife definieren. »PROGRAM«... sind die Programmköpfe, »VAR«... die Deklarationsteile, auf die ich bald eingehen werde und innerhalb »BEGIN« und »END.« befindet sich das eigentliche Programm. »WRITE« beziehungsweise »WRITELN« entsprechen im Commodore-Basic »PRINT;« beziehungsweise »PRINT«, »READLN« dem »INPUT«. »SORT« entspricht »SOR« und berechnet die Quadratwurzel.

Nun zur Routine selbst: Wie Sie wissen, kann man nur aus Zahlen >= 0 reelle Quadratwurzeln ziehen, deshalb muß man vor der Berechnung jeweils testen, ob die mit »READLN« gelesene Zahl >= 0 ist. Dies geht natürlich sehr elegant mit der »WHILE«-Schleife, die ja vor der Ausführung den Test durchführt. In der »REPEAT«-Schleife muß hingegen unbedingt ein Test mit »IF« gemacht werden. Weshalb wird aber die Eingabe zweimal geschrieben? Nun, erstens enthalten Variablen in Pascal, nachdem sie ins Leben gerufen wurden, keinen bestimmten Wert und zweitens muß der erstmalige Test in der »WHILE«-Schleife etwas »Wohldefiniertes« zum Testen haben. Man sieht auch gleich, daß der ganze Inhalt der »WHILE«-Schleife übersprungen wird, falls die erste Eingabe < 0 ist. Hätte man nun in der »WHILE«-Schleife keine Gelegenheit mehr, »z« zu verändern, käme man niemals mehr aus ihr heraus. In der »REPEAT«-Schleife verhält es sich diesbezüglich ähnlich. Nun noch schnell das äquivalente Basicprogramm:

In dem äquivalenten Basic-Programm (Listing 3) entsprechen der kleinen Standardprozedur »READLN« die Zeilen 30 und 40. Es ist sehr schön zu sehen, daß man in Basic nicht ohne das »GOTO« auskommt und daß das Programm jetzt schon schlechter ist, als dasjenige in Pascal. Stellen Sie sich schon jetzt mal vor, wie es sich erst mit längeren bis sehr langen Programmen verhält, bei denen man nicht so schnell sieht, was alles wiederholt wird, weil zwischen der angesprungenen Zeile und dem entsprechenden »GOTO« eventuell mehr als 100 Zeilen liegen.

Ein weiteres Pascal-Statement, von dem man sagen könnte, daß es irgend etwas teste, ist das »CASE«-Statement. Es lautet:

— CASE selektor OF
  marke 1 : STATEMENT;
  marke 2 : STATEMENT;
  marke 3 : STATEMENT;
  ..... : .....
  ..... : .....
  marke n : STATEMENT
END:

Mit ihm kann man eine aus mehreren Möglichkeiten auswählen, je nachdem welcher Marke der Selektor entspricht. Es entspricht den »ON x GOTO« und »ON x GOSUB«-Verteilern. Aber »CASE« ist dagegen ungleich vielseitiger. Bekanntlich darf »x« in Basic weder ein Ausdruck noch ein alphanumerisches Zeichen sein. In Pascal darf der Selektor ein beliebiges Zeichen sein. Listing 4 zeigt dies.

Je nachdem, welchen Wert »tagnr« hat, werden der Variablen »tag« die entsprechenden 3 Buchstaben zugewiesen. Stimmt der Selektor mit keiner der Marken überein, so ist im Standardpascal nicht definiert, was dann geschieht. In UCSD-Pascal hingegen wird einfach beim nächsten Statement fortgefahren. Hier folgt nun noch das kleine äquivalente Basicprogramm dazu (Listing 5). Da im Basic des Commodore 64 dem »IF...THEN«-Befehl das »ELSE« fehlt und weil eine Basiczeile nicht mehr als 80 Zeichen zuläßt, muß man auf die Konstruktion verschachtelter »IF...THEN...ELSE IF...THEN... ELSE IF...'s zur Emulation von »CASE« verzichten. Man sieht, daß man zur Emulation dieses Statements in Basic sehr redundant programmieren muß.

Es ist dies sicher nicht die einzige und schon gar nicht die eleganteste Lösung, aber wahrscheinlich dieje-

```
PROGRAM TAGRECHNUNG (INPUT,OUTPUT);
VAR     tag   : (MON,DIE,MIT,DON,FRE,SAM,SON);
        tagnr : integer;

BEGIN
  (*
  (* HIER SOLLTE EINE RECHNUNG STEHEN, DIE EINE  *)
  (* ZAHL ZWISCHEN 0 UND 6 IN tagnr HINTERLEGT.  *)
  (*                                             *)
  CASE tagnr OF
       0 : tag := MON;
       1 : tag := DIE;
       2 : tag := MIT;
       3 : tag := DON;
       4 : tag := FRE;
       5 : tag := SAM;
       6 : tag := SON
  END;   (* OF CASE *)
  WRITELN (tagnr,' entspricht ',tag)
END.
```

Listing 4. Der Selektor in der CASE-Anweisung kann eine Variable oder ein Ausdruck sein. Hier dient eine INTEGER-Variable als Selektor.

nige, die einem zuerst in den Sinn kommt. Eine weitere bevorzugte Verwendung von »CASE« ist die Menüsteuerung, wie Listing 6 zeigt.

Zu Listing 6 ist nicht viel zu sagen, außer daß man sieht, daß der Selektor (Variable »w«) auch ein alphanumerisches Zeichen sein kann und daß den Marken (»A«, »B«, »C«) mehrere Statements folgen können. Hier also Zuweisungen und Prozeduren. Hinzu kommt noch eine neue Standardprozedur genannt »PAGE«. Sie produziert einen Seitenvorschub auf einer Text-Ausgabedatei, weshalb dieser Prozedur ein Dateiname folgen muß. »PAGE (OUTPUT)« produziert also einen Seitenvorschub auf dem Bildschirm, der sich als Löschen des Schirms äußert.

Blockstruktur, lokale und globale Variablen

Nun komme ich endlich zu demjenigen Punkt, welchen ich bisher immer vor mir herschob, Ihnen aber schon vier mal in den Programmen vorgesetzt habe: Es ist dies der Programmkopf mit dem wichtigen Deklarationsteil. Die Blockstruktur ist ein Hauptmerkmal von Pascal. Man kann sich eine Ebene vorstellen, die bezeichnet wird mit »PROGRAM name«. In ihr sind überall und jederzeit alle Konstanten und Variablen verfügbar und änderbar, die in dem zu dieser Ebene gehörigen Deklarationsteil angegeben werden müssen. Ein Programm, das völlig ohne Prozeduren und Funktionen auskommt, liegt in dieser Ebene, und somit leben alle Variablen in ihm (das heißt sie sind gültig), eben weil sie im Deklarationsteil dieser Ebene stehen. Im Deklarationsteil, der zu dieser Ebene gehört, wird nun gesagt, ob und welche Konstanten, Typen und Variablen in dieser Ebene gebraucht werden. Man muß also alle in dieser Ebene verwendeten Variablen deklarieren. Wenn man nun ein Unterprogramm, genannt Prozedur, einführt, so bildet man eine neue Ebene, die jetzt auf derjenigen liegt, die »PROGRAM« genannt wurde (es entsteht mit der Zeit eine Art Relief, indem Ebene auf Ebene zu liegen kommt). Will man die Variablen aus der Hauptebene (»PROGRAM«) benützen, so kann man das bedenkenlos tun.

Man kann aber auch Konstanten, Typen und Variablen definieren, die nur in dieser und jeder darauffliegenden Ebene leben, das heißt man kann von der Hauptebene diese Variablen nicht ansprechen, weil sie für die Hauptebene nicht existieren, da sie nicht in deren Deklarationsteil verzeichnet sind. Daraus ergibt sich, daß man die gleichen Variablennamen verwenden darf, die eigentlich schon in der Hauptebene verteilt worden sind. Obwohl, gleich benannt, beeinflussen sie einander in keiner Weise. Beide Inhalte bleiben erhalten. Je nachdem, in welchem Block man sich befindet, sind gerade die Variablen von dessen Deklarationsteil aktiv. Man kann also, indem man in der Prozedur einen Deklarationsteil hat, eine Sperre aufbauen, die ein Verändern einer gleichlautenden Variable aus einer tieferliegenden Ebene verhindern

```
 10 REM       PROGRAMM TAGRECHNUNG
 20 REM       VARIABLEN tag$
 30 REM                tagnr%
 40 REM       BERECHNUNG VON tagnr
 50 IF tagnr% = 0 THEN tag$ = "MON"
 60 IF tagnr% = 1 THEN tag$ = "DIE"
 70 IF tagnr% = 2 THEN tag$ = "MIT"
 80 IF tagnr% = 3 THEN tag$ = "DON"
 90 IF tagnr% = 4 THEN tag$ = "FRE"
100 IF tagnr% = 5 THEN tag$ = "SAM"
110 IF tagnr% = 6 THEN tag$ = "SON"
120 PRINT tagnr%" entspricht "tag$
130 END
```

Listing 5. Dem Pascal-Programm in Listing 4 entsprechendes Basic-Programm.

```
PROGRAM RAHMEN (INPUT,OUTPUT);
VAR     i,k,l : real;
          c,w : char;

 PROCEDURE GETDATA;
 BEGIN
   (*                                  *)
   (*    HIER STEHT DIE DEFINITION     *)
   (*                                  *)
 END;        (* VON GETDATA *)

 PROCEDURE USEDATA;
 BEGIN
   (*                                  *)
   (*    HIER STEHT DIE DEFINITION     *)
   (*                                  *)
 END;        (* VON USEDATA *)

 PROCEDURE STOREDATA;
 BEGIN
   (*                                  *)
   (*    HIER STEHT DIE DEFINITION     *)
   (*                                  *)
 END;        (* VON STOREDATA *)


BEGIN            (* DES HAUPTPROGRAMMES *)
  REPEAT
    PAGE (OUTPUT);
    WRITELN; WRITELN; WRITELN;
    WRITELN (' BITTE WAEHLEN SIE: ');
    WRITELN;
    WRITELN (' A : DATEN HOLEN ');
    WRITELN (' B : DATEN BEARBEITEN ');
    WRITELN (' C : DATEN ABSPEICHERN ');
    WRITELN (' Q : BEENDEN ');
    READ (w);
    CASE w OF
      'A' :  BEGIN   i := i + 1.5; GETDATA;   END;
      'B' :  BEGIN   k := k - 2.9; USEDATA;   END;
      'C' :  BEGIN   l := l - 1; STOREDATA;   END
    END  (* OF CASE *)
    WRITELN;
    WRITELN ('IST IHRE ARBEIT BEENDET     J/N ?');
    READ (c);
  UNTIL c = 'J'
END.             (* DES HAUPTPROGRAMMES *)
```

Listing 6. Menüsteuerung über CASE-Anweisung in Pascal (Programmauszug)

```
PROGRAM INITMATRIX (INPUT,OUTPUT);
CONST       n = 25;      (* ZEILEN *)
            m = 15       (* SPALTEN *)
TYPE  matrix = ARRAY [0..n,0..m] OF INTEGER;
VAR         a : matrix;
          k,i : INTEGER;

 PROCEDURE ZEILE;
 VAR         i : INTEGER;
 BEGIN
   FOR i := 0 TO m DO
       a[k,i] := 0;
 END;

BEGIN    (* DES HAUPTROGRAMMES *)
  k := 0;
  FOR i := 0 TO n DO
    BEGIN
      ZEILE;
      k := k + 1;
    END;
    WRITELN ('DIE MATRIX IST JETZT INITIALISIERT WORDEN.');
END.     (* DES HAUPTROGRAMMES *)
```

Listing 7. Sowohl im Hauptprogramm als auch im Unterprogramm »Zeile« ist eine Variable i definiert worden. Beide Variablen, trotz demselben Namen, sind völlig unabhängig voneinander.

```
PROCEDURE OUT;
VAR  a,b : INTEGER;

 PROCEDURE IN (i :INTEGER; VAR j :INTEGER);
 BEGIN
   j := i * 2;
 END;

BEGIN
  .........
  .........
  IN (a,b);
  .........
  .........
END.
```

Listing 8. Parameterübergabe an Prozeduren. Es gibt formale und aktuelle Parameter.

kann. Aus dieser Tatsache entstand der Begriff der »Lokalen Variablen«. Ein typisches Anwendungsbeispiel dazu ist das »FOR i := x TO y DO«-Statement. Da man mit diesem Statement oft Indices verändert, heißt die Laufvariable meistens »i« für Index, und so hat es sich eingebürgert, daß man sie wenn möglich immer »i« nennt. Damit nun niemals irgendwelche Komplikationen auftreten können, deklariert man diese Variable »i« meistens in jeder Ebene und zwar vom Typ »INTEGER«. Ich hoffe, daß dies am nächsten Beispiel (Listing 7) ein bißchen klarer wird:

In Listing 7 gibt es zum ersten Mal einen fast ausgelasteten Deklarationsteil. Er ist strikt gegliedert und die Teile der Deklaration müssen immer in derselben Reihenfolge auftreten. Im Standardpascal lautet die Abfolge folgendermaßen:

— Labelteil	Zuweisung der Sprungmarken
— Konstantenteil	Deklaration und Zuweisung
— Typenteil	Deklaration und Definition
— Variablenteil	Deklaration und Typenzugehörigkeit
— Prozeduren und Funktionen	Deklaration und Definition

In UCSD-Pascal gibt es zwar den Labelteil, aber das »GOTO«-Statement nur bedingt. Man darf daher nicht ein Label anspringen, welches außerhalb des aktuellen Blockes liegt. Ein Aussteigen aus dem gerade bearbeiteten Block ist nur mit »EXIT procedure p« möglich. Der aktuelle Block wird verlassen und die Prozedur »p« wird ausgeführt. Eine Labeldeklaration geschähe dann wie folgt:
LABEL 0,27,56,876,9999
Damit hätte ich 5 Labels deklariert, die irgendwo im betreffenden Block stehen können. Die Zahlen haben nichts mit irgendwelchen Zeilennummern zu tun (man kann sie natürlich so verwenden).

Die Konstantendeklaration:
CONST name = wert;
Die Typendeklaration:
TYPE name = Definition des Typs;
Die Variablendeklaration:
VAR name : Typenzugehörigkeit;
Die Prozeduren- und Funktionendeklaration:
PROCEDURE name (Übergabeparameter);
Definition der Prozedur
FUNCTION name (Übergabeparameter);
Definition der Funktion

Nun ist zwar endlich klar, was ein Deklarationsteil ist, aber die Typenarten, von denen ich schon zwei in den Beispielprogrämmchen benutzt habe, sind noch immer nicht klar. Nun, es gibt einmal vordefinierte Typen, diese sind:

INTEGER	Bereich der ganzen Zahlen von —32768 bis +32767
REAL	Bereich der reellen Zahlen von zirka 1E-38 bis zirka 1E38
BOOLEAN	Nur 2 Werte, 0 = false, 1 = true
CHAR	1 Zeichen, das dem ASCII-Code von 32 bis 127 entspricht
TEXT	Abkürzung für »file of char«, siehe weiter unten
Im UCSD-Pascal existiert noch der Typ:	
STRING	Zeichenkette, array of char

Eine ebenfalls vordefinierte Abänderung der Standardtypen erhält man, indem man schreibt: »FILE OF standard typ«. So deklariert man eine sequentielle Datei, deren Elemente einem der Standardtypen angehören. Eine ungeheure Flexibilität erreicht Pascal nun, indem man eigene Typen definieren kann. Im letzten Beispiel habe ich auch einen eigenen Typen definiert, nämlich den Typ »matrix«. Er ist definiert als ein zweidimensionales Feld der Größe »m« * »n« Elemente des Standardtyps »INTEGER«. In der Typendeklaration hat man fast unbeschränkte Möglichkeiten der Typengenerierung. Da nun der Typ definiert ist, darf ich ihn im Variablendeklarationsteil verwenden. Auch das habe ich gemacht: Es sei »a« vom Typ »matrix«. Im Konstantenteil hat man nur die Standardtypen zur Auswahl, da die Typendeklaration erst später folgt. Einen großen Nachteil hat die Sache mit dem Deklarationsteil: Da man nur Variablen benutzen darf, die man deklariert hat, kann man die Reservation von Speicherplatz für die Variablen nicht optimieren. Man sieht das sehr deutlich an der Variablen »a«. Die Größe der Matrix ist bestimmt durch die Konstanten »m« und »n«, die ich aber schon vorher deklarieren mußte. Ob ich nun wirklich die ganze Größe voll ausnutze oder nicht, ich muß die Array-Grenzen angeben. Unter Umständen verschleudere ich sehr viel Speicherplatz, eventuell reicht er aber nicht einmal. In Basic dagegen kann man schreiben:
10 INPUT "WIEVIELE ZEILEN, SPALTEN";m,n
20 DIM matrix(m,n)

In Pascal geht es nicht so bequem, aber es ist möglich. Es werden noch einige Beispiele folgen, in denen an-

dere Typen gebraucht werden. Das Beispiel (Listing 7) zeigt, daß neben- und nacheinander zwei Variablen »i« leben können, die einander nicht zerstören. Die eine ist global, weil sie im Programmkopf deklariert worden ist, die andere ist lokal, da sie erst in der Prozedur deklariert wird. Man muß aber klar erkennen, daß man zu keinem Zeitpunkt die Werte beider Variablen zugleich

```
PROGRAM VERGLEICH (INPUT,OUTPUT);
CONST       max = ....;
TYPE   zahlenmenge = ARRAY [0..max] OF BOOLEAN;

FUNCTION M1INM2 (m1,m2 :zahlenmenge) :BOOLEAN;
VAR          i : INTEGER;
          flag : BOOLEAN;

BEGIN
  flag := TRUE;  i := 1;
  WHILE flag AND i <= max DO
    BEGIN
      IF m1[i] AND m2[i]
          THEN flag := TRUE
          ELSE flag := FALSE;
      i := i + 1;
    END;
END;        (* VOM M1INM2 *)

BEGIN
  (*                                              *)
  (* HIER FOLGT NUN EIN HAUPTPROGRAMM, DAS ZWEI   *)
  (* MENGEN ENTSPRECHEND DER DEKLARATION ERZEUGT, *)
  (* DIE NUN VERGLICHEN WERDEN MUESSEN MIT M1INM2 *)
  (*                                              *)
  (*
END.
```

Listing 9. Es wird geprüft, ob eine Menge in einer anderen Menge enthalten ist.

ausgeben kann. Zu den Prozeduren ist zu sagen: Jede Prozedur hat einen Namen. Man kann ihr Werte übergeben, indem man hinter dem Namen eine Klammer öffnet und die Variablennamen sowie deren Typenzugehörigkeit eingibt. Man kann nun aber auf zwei Arten Werte übergeben. Dies soll am Beispiel zweier Prozeduren (Listing 8) gezeigt werden:

Man erkennt sofort, daß an die Prozedur »IN« zwei Werte übergeben werden, fragt sich nur, wo der Unterschied der zwei Übergabearten liegt. Dies ist schnell geklärt:

Die Parameter in Klammern bei der Prozedur »IN (i: INTEGER; VAR j: INTEGER)« sind sogenannte »formale Parameter«, diejenigen aber im Prozedurenaufruf »IN(a,b)« nennt man »aktuelle Parameter«. Die formalen Parameter zeigen an, daß (hier zwei) Parameter übergeben werden, die in der betreffenden Prozedur »i« und »j« heißen werden. Die aktuellen Parameter dagegen beinhalten die Werte, die tatsächlich übergeben werden. In unserem Beispiel heißt das, daß die aktuellen Parameter »a« beziehungsweise »b« den formalen Parametern »i« beziehungsweise »j« entsprechen. Nun, einen kleinen Unterschied gibt es aber doch, wie die Notation der formalen Parameter vermuten läßt:

Dem formalen Parameter »j« wird der Inhalt des aktuellen Parameters »a« übergeben.

Dem formalen Parameter »j« jedoch wird die Speicherplatzadresse des aktuellen Parameters »b« übergeben. Ein kleiner aber wichtiger Unterschied.

Wenn nämlich die Prozedur »IN (i: INTEGER; VAR j: INTEGER)« ausgeführt wird, wird der Inhalt von »a« in die Variable »i« kopiert. Der Inhalt von »a« wird in dieser Prozedur nicht verändert. Dagegen wird beim Aufruf von »IN (i: INTEGER; VAR j: INTEGER)« nicht der Inhalt von »b« nach »j«, sondern deren Adresse zur Adresse von »j« kopiert. Das hat eine konsequenzenreiche Auswirkung: Alle Veränderungen von »j« in der Prozedur »IN« beeinflussen auch den Inhalt der globalen Variablen »b«, weil »j« nun ja einfach nur ein anderer Name für die Variable »b« ist. Die erste Methode der Parameterübergabe nennt man daher »call by value«, die zweite hingegen »call by reference«. Die Variablen sind nun in der betreffenden Prozedur sowie jeder weiteren darin verschachtelten Prozedur lokal bekannt. Man hat nur darauf zu achten, daß man bei der Übergabe keinen Typenkonflikt verursacht und daß beim Aufruf ebenso viele Parameter stehen wie beim Prozedurenkopf.

Diese Art der Parameterübergabe eignet sich besonders dazu, eigene Programmodule für eine Programmroutinenbibliothek bereitzustellen. Ein weiteres Beispiel dazu mit einer »FUNKTION«, für eine ganz bestimmte Anwendung, nämlich einer Vergleichsfunktion:

Gegeben sei folgende Deklaration:
CONST max = 10000;
TYPE zahlenmenge = ARRAY (1..max) OF BOOLEAN;

Eine Variable vom Typ »zahlenmenge« kann man auffassen als die Darstellung einer Menge von Zahlen zwischen »1« und »max«, wobei für jede solche Zahl der entsprechende Boolean-Wert im Array angibt, ob sie Element der Menge ist oder nicht (das heißt ob der korrespondierende Wert im Array »1« oder »0« ist).

Gesucht ist nun eine Pascal Funktion, die das Enthaltensein einer Menge »m1« in einer Menge »m2« (beide vom obigen Typ) prüft. Eine mögliche Lösung zeigt Listing 9.

Diese Funktion übernimmt nun aus dem Umfeld der Funktion zwei Variablen vom Typ »zahlenmenge« und weist sie den formalen Parametern »m1« und »m2« mittels der »call by value«-Methode zu. Neu bei der Funktion ist nun, daß hinter der Klammer noch deklariert werden muß, welchen Typs das Resultat der Funktion sein wird. Die Funktion ist ein wichtiger Bestandteil von Pascal. Ihr entspricht in Basic teilweise der Befehl »DEF FN name (variable) = expression«, außer daß er in Basic nur eine mathematische Formel definieren und nur eine numerische Variable übernehmen kann. Man kann mit ihm keinerlei Operationen definieren, die als Resultat einen String oder einen Charakter ausgeben.

Der zweite Teil erscheint in der nächsten Ausgabe und beschreibt die Programmierung mit Funktionen, dem Mengentyp, Aufzählungs- und Auszähltyp sowie die Definition von Datensätzen und den Einsatz von Zeigern (POINTER). Kurz erläutert werden Befehle zur Dateibearbeitung. Den Abschluß bildet eine kritische Auseinandersetzung mit vier Pascal-Versionen, die alle auf dem Commodore 64 lauffähig sind.

(Martin Baur)