Pascal-Kurs für Einsteiger (Teil 5)

Das Thema des vorerst letzten Teils unseres Kurses ist die Blockstruktur von Pascal. Dabei werden wir den Unterschied zwischen globalen und lokalen Variablen in der Theorie und in der praktischen Anwendung erläutern.

Pascal wird oft als eine Sprache bezeichnet, die in Blökken organisiert ist. Das Hauptprogramm wird dabei bereits als ein Block bezeichnet. Prozeduren und Funktionen bilden ebenfalls einen Block. Da jeder Block weitere Blöcke enthalten kann, läßt sich eine Hierarchie von Blöcken aufbauen. Teilaufgaben zur Lösung eines Problems werden solchen Blöcken übertragen. Damit ist eine Programmiermethode anwendbar, die man als »Programming by stepwise refinement« bezeichnet. Eine Aufgabe wird dabei in Teilaufgaben aufgegliedert und durch Unterprogramme — Prozeduren oder Funktionen — realisiert. Eine Teilaufgabe wird wieder in Teilaufgaben gegliedert, bis man schließlich bei Befehlen der Programmiersprachen angekommen ist. Pascal ermöglicht eine besonders einfache Handhabung dieser Methode.

Prozeduren

Eine Prozedur besteht aus einer Kopfzeile und einem Block. Die Kopfzeile enthält das reservierte Wort »PROCEDURE«, einen Prozedurnamen und eine Liste von Parametern. Diese darf auch entfallen:

```
PROCEDURE Prozedurnamen;
```

Der Block einer Prozedur ist genauso wie beim Hauptprogramm aufgebaut. Es gibt lediglich einen kleinen Unterschied: Am Ende des Hauptprogramms steht ein ».«, am Ende einer Prozedur oder Funktion ein »;«. Der Definitionsteil des Prozedurblocks kann somit selbst wieder Prozedur- und Funktionsvereinbarungen enthalten. Ist dies der Fall, spricht man auch von Schachtelung.

Eine Funktion besteht ebenfalls aus einer Kopfzeile und einem Block. Die Kopfzeile enthält das reservierte Wort »FUNCTION«, einen Funktionsnamen, eine Parameterliste, einen Doppelpunkt und den Funktionstyp. Die Parameterliste kann auch entfallen. Die allgemeine Form der Kopfzeile lautet:

```
FUNCTION Funktionsname:Datentyp;
```

Der Typ einer Funktion muß skalar sein (Integer, Real, Boolean, Char, Aufzählungs- oder Ausschnittstyp). Auch der Definitionsteil einer Funktion kann wieder andere Funktionen und Prozeduren enthalten. Im Unterschied zu Prozeduren muß einer Funktion im Programm ein Wert an den Funktionsnamen zugewiesen werden. Funktionsnamen dürfen überall im Programm stehen, wo Variablen desselben Typs stehen dürfen. Typischerweise werden Funktionen innerhalb eines Ausdrucks verwendet.

In jedem Block können Variablen vereinbart werden. Eine Variable heißt »lokal« für einen Block, wenn sie in diesem definiert ist. Lokale Variablen sind nur in diesem Block bekannt. Wertzuweisungen an lokale Variablen haben keinen direkten Einfluß auf das Programm außerhalb dieses Blocks.

Eine Variable heißt »global« für einen Block, wenn sie in einem übergeordneten Block definiert ist. Globale Variablen sind in allen Blöcken bekannt, die dem Block untergeordnet sind, in dem sie vereinbart wurden.

Globale und lokale Variablen können gleiche Namen haben. Sobald eine lokale Variable mit gleichem Namen wie eine globale Variable definiert wird, tritt folgende Regel in Kraft:

In dem Block, in dem eine lokale Variable mit gleichem Namen wie eine globale Variable vereinbart wird, ist lediglich die lokale Variable bekannt. Die globale Variable ist in diesem Block nicht mehr verfügbar. Von Wertzuweisungen an die lokale Variable bleibt die globale Variable unberührt.

Globale und lokale Variablen

Trotz dieser Regelung sollte zugunsten der Übersichtlichkeit eines Programms auf die Definition lokaler und globaler Variablen gleichen Namens verzichtet werden. Geeignete Variablennamen, die die Aufgabe der Variablen beschreiben, tragen zur Klarheit eines Programms erheblich bei.

Variablen, die nur in einem

```
program lotto;
const k0=10;
var a:packed array[1..49] of boolean;
    i,k: integer;
function rnd(zahl:integer):real;
const a=13;
      b=29;
      m=63;
begin
  zahl:=(a*zahl+b) mod m;
  rnd:=zahl/m;
end;
begin
  for i:=1 to 49 do a[i]:=false;
  k:=k0;
  writeln('die lottozahlen: ');
  i:=0;
  while i<6 do
  begin
    k:=trunc(rnd(k)*48+0.5)+1;
    if a[k]=false then
    begin
      write(k:4);
      i:=i+1;
      a[k]:=true
    end;
  end; (* while-schleife *)
end.
```

Listing 1. Ein selbstdefinierter Zufallszahlengenerator

```
program matmult;
(* multiplikation von zwei  *)
(* matizen vorgegebener groesse *)
const n=5;
      m=3;
      l=4;
type a1=array[1..n,1..m] of real;
     b1=array[1..m,1..l] of real;
     c1=array[1..n,1..l] of real;
var a:a1;
    b:b1;
    c:c1;
procedure eingeben(var x:a1; var y:b1);
var i,j:integer;
begin
  writeln('bitte ',n,' kreuz ',m,' matrix eingeben');
  for i:=1 to n do
      for j:=1 to m do read (x[i,j]);
  writeln('bitte ',m,' kreuz ',l,' matrix eingeben');
  for i:=1 to m do
      for j:=1 to l do read(y[i,j]);
end;
procedure multiplizieren (f1:a1; f2:b1; var produkt:c1);
var i,j: integer;
function mult (f1:a1; f2:b1; var i,j:integer): real;
var d:real;
    k:integer;
begin
   d:=0.0;
   for k:=1 to m do d:=d+f1[i,k]*f2[k,j];
   mult:=d;
end;
begin
   for i:=1 to n do
       for j:=1 to l do produkt[i,j]:=mult(f1,f2,i,j)
end;
procedure ausgeben (var c:c1);
var i,j: integer;
begin
   writeln('ergebnis der multiplikation');
   for i:=1 to n do
   begin
      for j:=1 to l do write (c[i,j]:8:2);
      writeln;
   end;
end;
begin
   eingeben(a,b);
   multiplizieren(a,b,c);
   ausgeben(c)
end.
```

Listing 2. Ein Programm zur Matrizenmultiplikation

Block benötigt werden, sollten stets als lokale Variablen vereinbart werden. Durch die Definition lokaler Variablen und einer geeigneten Parameterübergabe können Unterprogramme als unabhängige Programmeinheiten formuliert werden. Diese Programmtechnik gestaltet ein Programm übersichtlich und trägt dazu bei, Fehler zu vermeiden.

Die Kommunikation zwischen den Blöcken

Was in unseren Überlegungen noch fehlt, ist die Übergabe von einem Block an einen anderen. Diese Übergabe erfolgt durch Parameterlisten. Jede Funktion oder Prozedur kann mit verschiedenen Eingangsgrößen berechnet werden. Aus diesen Eingangsgrößen berechnet das Unterprogramm Ergebnisse, die an den aufrufenden Block zurückgegeben werden können. Bei Funktionen ist eine solche Größe immer vorhanden: der Funktionswert. Beispiel für eine Funktion mit Parameterliste:

```
FUNCTION
ITERATIV(N:INTEGER):INTEGER;
```

In dieser Parameterliste werden die zu übergebenden Parameter formal definiert. Beim Aufruf der Funktion ersetzt man die formalen Parameter durch aktuelle:

```
Y:=ITERATIV(5);
```

Ist mehr als ein Parameter vorhanden, müssen die aktuellen Parameter mit den formalen Parametern bezüglich Reihenfolge und Datentyp übereinstimmen. Es existieren drei Formen von Parametern:

— Wertparameter
— Variablenparameter
— Prozedur- und Funktionsparameter

Wertparameter dienen nur der Eingabe von Werten in ein Unterprogramm. In der Parameterliste werden sie mit ihrem Namen und Datentyp aufgeführt: (Bezeichner:Datentyp; ..... ; Bezeichner:Datentyp)

Ein Beispiel dazu:

```
FUNCTION
EUCLID(A:INTEGER):INTEGER;
```

Beim Aufruf ist die Angabe des Typs nicht erforderlich. Es dürfen auch Ausdrücke als aktuelle Parameter übergeben werden. Mit Wertparametern können keine Daten an aufrufende Blöcke zurückgegeben werden. Dazu bedient man sich der Variablenparameter. Diese müssen in der formalen Parameterliste mit ⟨VAR⟩ gekennzeichnet werden:

```
PROCEDURE POT (X:REAL;Y:IN
TEGER; VAR Z:REAL);
```

Wertparameter	Variablenparameter
Name und Typ werden in der Liste der formalen Parameter angegeben.	Gekennzeichnet durch VAR in der Liste der formalen Parameter.
Der aktuelle Parameter bleibt durch Wertzuweisung an den formalen Parameter unberührt.	Mit dem formalen Parameter wird zugleich der aktuelle Parameter verändert.
Der aktuelle Parameter kann ein Ausdruck sein.	Der aktuelle Parameter muß eine Variable sein.
Wertparameter sind Eingangsgrößen.	Variablenparameter können Eingangs- und Ausgangsgrößen sein.
Speicher für aktuelle und formale Parameter nötig.	Aktuelle und formale Variablen greifen auf denselben Speicher zu.

Tabelle 1. Unterschiede zwischen Wert- und Variablenparametern

```
program fakultaet;
var n:integer;
function rekursiv(n:integer):integer;
begin
   if n=0 then rekursiv:=1
      else rekursiv:=rekursiv(n-1)*n
end;
function iterativ(n:integer):integer;
var i,f:integer;
begin
   f:=1;
   for i:=2 to n do f:=f*i;
   iterativ:=f
end;
begin
   n:=3;
   writeln(rekursiv(n));
   writeln(iterativ(n))
end.
```

Listing 3. Rekursion und Iteration im Vergleich

Iteration	
Schleife	
kein Extraspeicher nötig	
schneller	
möglicherweise komplizierte Programmstruktur	
Programmerstellung schwieriger	
Rekursion	
wiederholter Aufruf	
Extraspeicher nötig	
langsamer	
einfache Datenstruktur	
Programm problemnah	

Tabelle 2. Vergleich Iteration — Rekursion

Wird jetzt der Wert des formalen Parameters im Unterprogramm verändert, ändert sich auch der aktuelle Parameter des aufrufenden Blocks. Der aktuelle Parameter muß daher eine Variable und darf kein Ausdruck sein.

Das Programmbeispiel »LOTTO« (Listing 1) zeigt die Verwendung von Funktionen. Zur Erzeugung einer Zufallszahl wird dabei ein selbstdefinierter Zufallszahlengenerator verwendet, der Zahlen zwischen 0 und 1 erzeugt.

Das Beispielprogramm »MATMULT« (Listing 2) verwendet sowohl lokale als auch globale Parameter sowie Wert- und Variablenparameter. Es werden zwei Matrizen von der Tastatur eingelesen, miteinander multipliziert und auf den Bildschirm ausgegeben. Für die Multiplikation von Zeilen und Spalten wurde — zur besseren Demonstration — noch eine Funktion entwickelt.

Hierarchie von Unterprogrammen

Eingangsgrößen werden als Wertparameter, Ausgangsgrößen als Variablenparameter definiert. In der Tabelle 1 werden nochmals die Eigenschaften von Wert- und Variablenparameter gegenübergestellt. Aus diesem Vergleich kann die Regel abgeleitet werden, daß Eingangsgrößen als Wertparameter übergeben werden und Ausgangsgrößen als Variablenparameter.

Bevor ein Block von einem anderen aufgerufen werden kann, muß er vereinbart worden sein. Es gibt jedoch die Möglichkeit, von dieser Regel abzuweichen. Man bedient sich dazu der »FORWARD«-Definition. Dabei wird die Kopfzeile der Prozedur einschließlich der Parameterliste hingeschrieben und mit »FORWARD« gekennzeichnet:

```
PROCEDURE A1(X,Y:REAL); FOR
WARD;
```

Später, bei der eigentlichen Definition des Unterprogramms, entfällt dann die Parameterliste:

```
PROCEDURE A1;
 BEGIN
  ......
 END;
```

Rekursion

Rekursion bedeutet, daß Prozeduren und Funktionen sich selber aufrufen. Bei der direkten Rekursion ruft ein Block sich selbst auf. Bei der indirekten Rekursion ruft ein Block einen nebengeordneten Block auf und über diesen wieder sich selbst.

Das klassische Beispiel für Rekursion ist die Fakultät. Die rekursive Definition lautet:

n! = n * (n—1)!
0! = 1

Die Fakultät kann man aber auch iterativ definieren:

n! = 1 x 2 x ..... x n

Die Iteration läßt sich mit Wiederholungen realisieren (FOR, REPEAT, WHILE). Zur Rekursion benötigt man Unterprogramme, die einen Namen tragen und sich daher selbst aufrufen können.

Jede Rekursion muß terminieren, das heißt die rekursiven Aufrufe müssen unter bestimmten Bedingungen ein Ende finden. Bei der Fakultät tritt das beispielsweise ein, wenn n gleich Null wird. Im Beispielprogramm (Listing 3) werden die Möglichkeiten zur Berechnung der Fakultät — rekursiv oder iterativ — gezeigt. In Tabelle 2 werden Iteration und Rekursion nochmals verglichen.

Theoretisch sind Iteration und Rekursion gleichwertig, da jede Rekursion als Iteration formuliert werden kann und umgekehrt. Eine rekursive Prozedur erfordert jedoch erheblich mehr Speicher, da bei jedem Aufruf die lokalen Variablen zur Fortsetzung gespeichert werden müssen. Trotzdem bietet die rekursive Lösung häufig Vorteile gegenüber der iterativen. Viele Probleme haben bereits eine rekursive Struktur, die sich leicht in ein Programm umsetzen läßt.

(A. Gruber/nj)