Procedury i funkcje

1. Funkcje

Na podstawie przykładów, które wykonaliśmy podczas poznawania ATD zdążyliśmy zapoznać się z jednostkami programowymi takimi jak funkcje i procedury. Ponieważ zrobiliśmy to trochę pobieżnie warto poświęcić nieco więcej czasu na dokładniejsze ich poznanie.

Funkcja jest w PASCALu niezależną jednostką programową innego rodzaju niż `while` lub `for`. Jest to podprogram, a więc program zawarty w głównym programie PASCALa. Wymaga ona własnych deklaracji stałych, zmiennych, typów oraz innych podprogramów.

1.1. Definicja funkcji

Oto przykład prostej funkcji, takiej jaka była nam potrzebna w rozwiązywaniu równania metodą *regula falsi*. Jest to deklaracja funkcji, która powinna być umieszczona tam gdzie jest miejsca na wszystkie deklaracje.

```
function f(punkt: real): real;
{
   przyklad funkcji jakiej dostarcza uzytkownik
   w problemie znajdowania pierwiastka np. metoda regula falsi
}
begin
   f:= sqrt(punkt)-10.0
end; { funkcji f }
```

Po słowie (zgłoszeniu) `function` następuje jej nazwa; tutaj `f`. Po nazwie, w nawiasach okrągłych, wymienia się tzw parametry formalne funkcji wraz z ich typami. Dalej, podobnie jak w programie głównym, jest miejsce na deklaracje. W końcu mamy blok główny funkcji zaczynający się słowem `begin`, a kończący się słowem `end`. Po słowie `end` umieszcza się zawsze średnik.

1.2. Wywołanie funkcji

Sama deklaracja funkcji nie powoduje jej wykonania się. Funkcja (procedura) zostaje wykonana dopiero po jej wywołaniu. Zauważmy, że nazwa funkcji jest określonege typu (w przykładzie jest to real). Prócz tego w ciele funkcji pod tę właśnie nazwę dokonuje się przypisanie wielkości tego typu (w przykładzie jest to wynik obliczenia `f:= sqrt(punkt)-10.0`. Dzięki temu można w programie, w którym funkcja została zadeklarowana, używać tej nazwy podobnie jak nazwy innych wielkości, a więc w instrukcji przypisania i innych konstrukcjach. Nazwa ta musi jednak być opatrzona dodatkowymi informacjami. Są to wartości lub podstawniki parametrów występujących w funkcji. W przykładzie, parametrem takim jest wielkość formalna `punkt` typu rzeczywistego. Funkcja z przykładu może być wywołana w następujący sposób.

```
...
```

```
var
 x, y, z: real;
 ...
begin
 ...
 z:=7.0;
 y:=f(z);
 ...
 x:=y;
 if (f(z) > f(x)) then
 ...
 if (f(x) > 1.3 then writeln('Funkcja jest wieksza niz 1.3', );
 ...
end.
```

W podanym przykładzie funkcja `f` została wywołana kilkakrotnie z różnymi parametrami. Dwa razy z parametrem `z` i dwa razy z `x`. W obu przypadkach wartości tych parametrów *aktualnych* różniły się. Należy odróżniać parametry aktualne (`x`, `z`) od parametrów formalnych funkcji (w naszym wypadku `punkt`). Liczba parametrów aktualnych musi być taka sama jak liczba parametrów formalnych.

1.3. Procedury

Funkcja pozwala obliczyć i przekazać za pośrednictwem nazwy jedną wielkość skalarną. Często potrzebujemy przekazać do programu więcej wielkości niż jedna lub wykonać złożone operacje na grupie danych. Taką możliwość daje procedura: `procedure`. Jej deklaracja jest podobna do deklaracji funkcji. Różnica polega na tym, że nazwie procedury nie można przypisać żadnej wartości. Oto przykład deklaracji procedury, która wypisuje informacje o odległości do ... .

```
procedura raport(odleglosc: real);
{
   procedura wypisuje srednia odleglosc
}
begin
   writeln;
   writeln('Srednia odleglosc do ... wynosi ', odleglosc)
end;  { procedury raport }
```

Można jej wielokrotnie użyć w programie, który oblicza średnie odległości do ....

```
sum:=0.0;
licz:=0;
while not eof do
begin
   readln(odl);
   sum:=sum+odl;
   licz:=licz+1
end;
srednia:=sum/łicz;
raport(odl);
...
```

W przykładzie tym pokazany jest sposób wywołania procedury. Piszemy jej nazwę podając w nawiasie parametry aktualne.

1.4. Parametry funkcji i procedur

Pamiętamy, że deklarując jakąś zmienną wielkość w programie pisaliśmy słowo `var`. W nagłówkach procedur i funkcji z prostych przykładach, które podaliśmy do tej pory, parametry formalne nie były opatrywane żadną informacją o ich zmienności. Co to oznacza? Czy można opatrzyć słowem `var` parametr formalny funkcji? Rzecz ta jest trochę skomplikowana i komplikacja ta jest zamierzona.

Popatrzmy na następujący przykład.

```
procedure przyklad(x: integer);
begin
   x:=x+1;
   writeln(x)
end;
```

W przykładzie, zmienne formalna `x` (nie opatrzona słowem `var`) nie jest zmienną. Jest to więc może stała? Też nie. Ta zmienna to zmienna w pewnym sensie *wewnętrzna* dla podprogramu. W momencie gdy wywołuje się procedurę `przyklad`, wartość aktualnego parametru kopiowana jest na parametr formalny znany tylko wewnątrz podprogramu. Parametr aktualny nie zmienia się przy operacji wywołania i pozostaje taki sam do końca działania podprogramu. Nie można go zmienić w podprogramie. Mówimy, że nastąpiło *wywołanie przez wartość*. Co się stanie jeśli wywołamy podprogram `przyklad` w następującym fragmencie programu.

```
...
a:=1;
writeln(a);
przyklad(a);
writeln(a);
...
```

Wynikiem działania tego fragmentu jest
1
2
1
Parametr a został przekazany przez wartość. Założymy teraz, że pewne parametry

formalne podprogramu zostały poprzedzone słowem `var`.

```
procedure przyklad(var x: integer); { zwroc uwage na slowo VAR }
begin
   x:=x+1;
   writeln(x)
end;
```

Jeśli znów wykonamy program

```
...
a:=1;
writeln(a);
przyklad(a);
writeln(a);
...
```

to wynikiem jego działania jest
1
2
2
Nie jest to dziwne, gdyż w tym przypadku nazwa, a właściwie nazwa miejsca gdzie

znajduje się wartość parametru a została podana podprogramowi i może on „operować w tym miejscu". Mówiąc inaczej, podprogram może zmieniać wartości parametrów aktualnych jeśli parametry formalne poprzedzone są słowem `var`. Należy dodać, że wywołania procedur muszą być zgodne z ich deklaracjami. Np. nie możemy wywołać procedury

```
procedure zamien(x: integer; var y: integer);
var
   z: real;
begin
   z:=y;
   y:=x;
   x:=z
end;
```

tak

```
zamien(y, 1);
zamien(2, 3);
```

gdyż wprowadza to niezgodność z deklaracją. Wielkość 1, 2, 3 (a być może i y) są stałymi i nie mogą zostać zmienione w podprogramie. Prawidłowym wywołaniem jest natomiast

```
zamien(1, y);
```

o ile tylko y jest zmienną (tzn. zostało zadeklarowane z dodatkowym słowem `var`).

Opisana własność deklarowania podprogramów i różnice ich wywołań mają swoje zastosowania.

1.5. Procedury i funkcje jako parametry formalne

Dopisać.

1.6. Procedury i funkcje rekurencyjne

Przykładami książkowymi procedur rekurencyjnych są wieże z Hanoi, liczby Fibonacciego i jeszcze parę innych. W fizyce często oblicza się wielokrotne całki, np. licząc energię kulombowską naładowanej bryły. Może to być np. elipsoida trójosiowa. Energia elektrostatyczna takiej naładowanej bryły wyraża się formułą

$$E = \frac{1}{2}\int_V \int_{V'} \frac{\rho(\vec{r})\rho(\vec{r}')}{|\vec{r}-\vec{r}'|}\,dV\,dV'\,.$$

Jest to całka sześciokrotna. W celu jej wyliczenia można użyć tzw. kwadratur Gaussa. Nazwijmy procedurę Gaussa jego nazwiskiem i niech jej argumentami będą zakresy zmiennych całkowania x_i gdzie $i = 1, \ldots 6$, np. x_{1d}, x_{1g}. Całkę kulombowską zapiszemy w postaci:

$$E = \frac{1}{2}\int_{x_{1d}}^{x_{1g}} \ldots \int_{x_{6d}}^{x_{6g}} \frac{\rho(x_1,x_2,x_3)\rho(x_4,x_5,x_6)}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2+(x_3-x_6)^2}}\,dx_1\,dx_2\ldots dx_6'\,.$$

Nie wygląda to ładnie. Pomimo to, dzięki takiemu zapisowi możemy zastosować kwadraturę, którą posiadamy. Zapiszmy to schematycznie tak

$$\begin{aligned} E \;:=\; \frac{1}{2}\mathrm{Gauss}(&x_{1d},x_{1g},\\ &\mathrm{Gauss}(x_{2d},x_{2g},\\ &\quad\mathrm{Gauss}(x_{3d},x_{3g},\\ &\qquad\mathrm{Gauss}(x_{4d},x_{4g},\\ &\qquad\quad\mathrm{Gauss}(x_{5d},x_{5g},\\ &\qquad\qquad\mathrm{Gauss}(x_{6d},x_{6g},f(x_1,x_2,\ldots,x_6)))))))\,. \end{aligned}$$

Wygląda to jeszcze gorzej niż poprzednio! W tym wyrażeniu $f(x_1, x_2, \ldots, x_6)$ jest funkcją podcałkową

$$f(x_1, x_2, \ldots, x_6) = \frac{\rho(x_1, x_2, x_3)\rho(x_4, x_5, x_6)}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2 + (x_3 - x_6)^2}}$$

Powstał następujący schemat. Dla zadanych z góry współrzędnych, przez pierwszą, następnie drugą itd. kwadratury Gaussa, wyliczana jest funkcja podcałkowa i pierwsza całka po x_6. Następnie wylicza się całka po x_5 itd. W ten sposób obliczona zostaje sześciokrotna całka kulombowska. Nie jest to nic nadzwyczajnego, szczególnie, że czas obliczeń jest napewno za długi, a po drugie wynik nie posiada dobrej dokładności. To co chcemy tu jednak pokazać polega na tym, że wciąż używamy tej samej procedury obliczania całki, procedury `Gauss`. Jest ona tutaj wielokrotnie zastosowana. Wewnątrz pierwszej jest druga, wewnątrz drugiej trzecie itd, aż do ostatniej, szóstej. Ten sposób używania procedur lub funkcji nazywa się **rekurencją**. Procedury są zagnieżdżone jedna w drugiej. Kiedy ostatnia z procedur wykona swoją robotę, sterowanie przechodzi do procedury *wyższej*.

Moglibyśmy wykonać te same obliczenia używając sześciu kopii procedury Gauss: Gauss1, …, Gauss6. Wyszłoby na to samo.

Wiele języków programowania nie pozwala na stosowanie rekurencji. Takim jest np. FORTRAN. PASCAL pozwala na stosowanie rekurencji.

Można powiedzieć, że program rekurencyjny składa się z instrukcji nie zawierających go i z niego samego. Zapisujemy to w postaci

$$P = \mathcal{P}[\mathcal{I}_i, P]\,.$$

1.7. Przykłady rekurencji

Podamy kilka typowych przykładów rekurencji.

PRZYKŁAD. Innym przykładem, może bardziej naturalnym, będzie procedura znajdowanie potęgi liczby. W TURBO PASCALu brakuje takiej procedury. Przy wyliczaniu potęgi x^n można skorzystać z własności

$$x^n = x \cdot x^{n-1}\,.$$

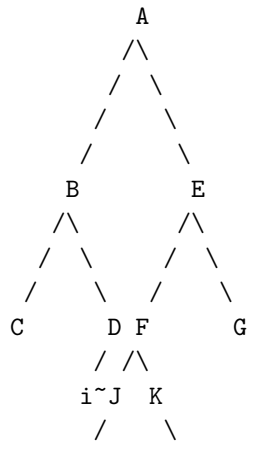
Funkcja wyliczająca potęgę może być następującej postaci.

```
function potega(x: real; n: integer): real;
{
   oblicza n-ta potege liczby x; dla x=0 zwraca 0
}
begin
   if x = 0 then
      potega:=0
   else
      if n = 0 then
         potega=1
      else
         if n < 0 then
            potega := potega(x, n+1) // x
         else
            potega := potega(x, n-1) * x
end; { potega }
```

Jeśli chcemy wyliczyć x^2 dla $x \neq 0$ to procedura `potega` po jej wywołaniu z parametrami $(x, 2)$ *uruchomi siebie* (drugą kopię) jeszcze raz z parametrami $(x, 1)$, i następnie z parametrami $(x, 0)$ trzecia kopia funkcji `potega`). Kiedy wyliczy się

ta ostatnia jej wartość (równa $x^0 = 1$) to przez pomnożenie jej przez x w 15 linii programu, w drugiej kopii funkcji, otrzymamy wartość x^1, która pomnożona przez x w pierwszej swojej kopii da x^2.

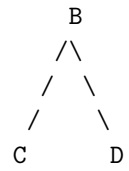
PRZYKŁAD. Załóżmy, że chcemy przejrzeć następującą strukturę (drzewo binarne)



Drzewo binarne

która złożona jest z tzw. węzłów i co najwyżej dwu gałęzi, które nazwiemy wskazami. Wskazy te nazywamy lewym i prawym. Wskazy podają miejsce następnych węzłów. Zadaniem jest wypisanie nazw wszystkich węzłów, a więc `A, B, C, ...`

Natychmiastowe podanie rekursywnego algorytmu przeglądania drzewa nie jest łatwe. Zauważymy, że drzewo ma następującą własność, która pomoże nam w dalszym postępowaniu. W każdym węźle drzewa binarnego zaczepione jest inne drzewo binarne. Np. w A zaczepione jest całe drzewo. Lewy wskaz wychodzący z A pokazuje drzewo zaczepione w B, przedstawione na kolejnym diagramie.



Jeśli dotrzemy do węzła C (lub D) to widzimy, że nie wychodzi z niego żadna gałąź. Możemy też powiedzieć, że mamy tu zaczepione drzewo *puste*.

Korzystając z opisanej własności możemy utworzyć algorytm przeglądania drzewa binarnego. Podany jest on poniżej. sprawdzaj drzewp (wskaz)=

```
Begin
  Jesli wskaz nie pokazuje drzewa pustego to
        sprawdzaj drzewo (lewy wskaz)
        wypisz nazwe pnia
        sprawdzaj drzewo (prawy wskaz)
End
```

Algorytm ten jest rekurencyjnym algorytmem przeglądania drzewa. Wynikiem jego działania dla przypadku drzewa pokazanego na początku będzie ciąg: A, B, C, D, I, E, F, J, K, G.

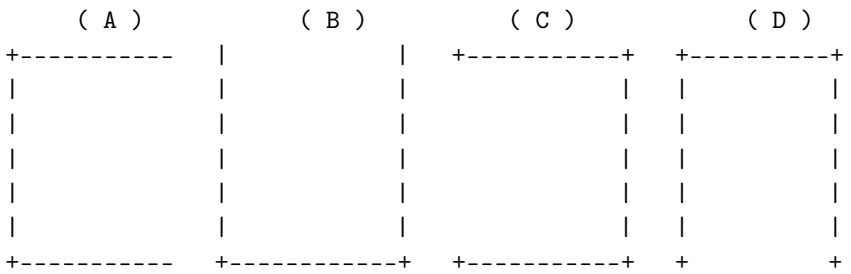
1.8. Deklaracje wyprzedzające (forward)

W PASCALU obowiązuje zasada, że każda używana zmienna, stała, funkcja itp. musi zostać wcześniej zadeklarowana. Dotyczy to również kolejnści deklarowania. Najpierw deklaruje się i definiuje jakieś wielkości, a następnie za ich pomocą inne.

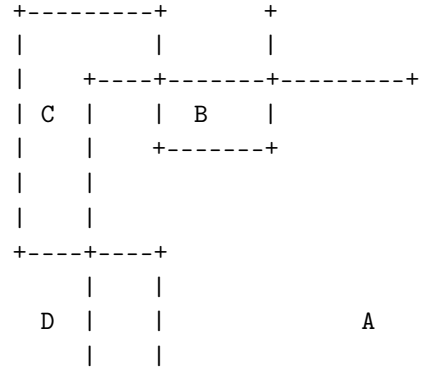
Często zachodzi potrzeba użycia procedur lub funkcji tak, że zarówno pierwsza korzysta z drugiej jak też odwrotnie, druga z pierwszej. Deklarowanie takich funkcji w zwykły sposób jest więc niemożliwe. Można to jednak zrobić używając słowa `forward` w następujące sposób.

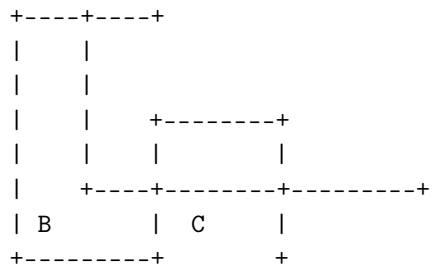
```
{ nastepna linia jest deklaracją wyprzedzajaca, forward }
function pierwsza(x, y: real): real; forward;

procedure druga(var r1, r2: real);
begin
  .
  .
  .
  z := pierwsza(a,b)    { ta instrukcja wymaga deklaracji forward }
  .
  .
end; { procedury pierwsza}

function druga;    { zwroc uwage na skrocona postac naglowka funkcji }
begin
  .
  .
  druga(u, v);     { niejawna rekursja }
  .
  .
end; { funkcji druga }
```

Kolejny przykład jest programem graficznym, który rysuje tzw. krzywe Hilberta. Pochodzi on z podręcznika Wirtha, Struktury danych itp. Jego zadaniem jest w pierwszym rzędzie narysowanie którejś krzywej A, B, C lub D.



W przypadku gdy rząd jest większy (2, 3 ...) to krzywe te muszą pojawić się pomniejszone i to tak by powstał odpowiedni wzór. Np w drugim rzędzie z A powstaje następująca krzywa:





która zbudowana jest z krzywych B, C i D, ale zmniejszonych i polaczonych liniami prostymi. Jeśli procedury, które będą rysowały krzywe A, B, C i D, (w różnej skali) oznaczymy przez A, B, C i D, to aby narysować taką krzywą jak pokazana na rysunku, nalezy najpierw wywołać procedurę A, a ta wywoła kolejno C, B, D, C, B, które narysuję małe krzywe C, B, itd. i połączy je liniami prostymi. Oczywiście, jeśli rząd jest większy od 2 to procedury C, B, itd. wywołają jeszcze raz te same procedury itd. Jest to więc proces rekurencyjny. Oto kompletny program w TURBO PASCALu, który rysuje krzywe Hilberta dowolnego stopnie.

```
program Hilbert;
{ Krzywe Hilberta rzedu n }

uses
  Graph;
var
  grDriver : integer;
  grMode   : integer;
  ErrCode  : integer;

const
  n=2;
var
  i,h0,h,x,y,x0,y0: integer;

procedure B(i: integer); forward;
procedure C(i: integer); forward;
procedure D(i: integer); forward;

procedure A(i: integer);
begin
  if i> 0 then
  begin
    D(i-1); x:=x-h; lineto(x,y);
    A(i-1); y:=y-h; lineto(x,y);
    A(i-1); x:=x+h; lineto(x,y);
    B(i-1);
  end
end;

procedure B(i: integer);
begin
  if i>0 then
  begin
    C(i-1); y:=y+h; lineto(x,y);
    B(i-1); x:=x+h; lineto(x,y);
```

```
    B(i-1); y:=y-h; lineto(x,y);
    A(i-1);
  end
end;


procedure C(i: integer);
begin
  if i>0 then
  begin
    B(i-1); x:=x+h; lineto(x,y);
    C(i-1); y:=y+h; lineto(x,y);
    C(i-1); x:=x-h; lineto(x,y);
    D(i-1);
  end
end;


procedure D(i: integer);
begin
  if i>0 then
  begin
    A(i-1); y:=y-h; lineto(x,y);
    D(i-1); x:=x-h; lineto(x,y);
    D(i-1); y:=y+h; lineto(x,y);
    C(i-1);
  end
end;


begin
  grDriver := Detect;
  InitGraph(grDriver,grMode,'');
  ErrCode := GraphResult;
  if ErrCode = grOk then
    begin
      h0:=GetMaxX; h:=GetMaxY; if h0>h then h0:=h;
      i:=0; h:=h0;  x0:=h div 2; y0:=x0;
      { Do graphics }
      repeat { rysuj krzywa Hilberta rzedu i~}
            i:=i+1; h:=h div 2;
            x0:=x0+(h div 2); y0:=y0+(h div 2);
            x:=x0; y:=y0; moveto(x,y);
            A(i);
      until i=n;
      Readln;
      CloseGraph;
    end
  else
    Writeln('Graphics error:', GraphErrorMsg(ErrCode));
end.
```

Zadanie 1. Nic... 1

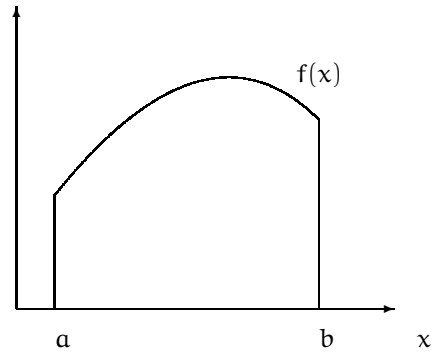
Napisz program w PASCALu, który rozwiązuje następujący problem. Wczytuje ... [Spis]

Zadanie 2. We-Wy 2

Napisz procedurę, która wywołana z parametrem, który jest łańcuchem znaków zwraca ten sam łańcuch z usuniętymi znakami pustymi i liczbą usuniętych znaków pustych. Jaki będzie typ procedury i jaki jest mechanizm przekazywania parametrów? [Spis]

Zadanie 3. Całka 2

Napisać funkcję w PASCALu, która oblicza całkę z innej funkcji $f(x)$ na przedziale $< a, b >$ stosując metodę prostokątów. Całka jest polem pod krzywą f(x)



Zakładając, że jest tych prostokątów n, całkę I możemy zapisać w postaci

$$I = h(f(a) + f(a+b) + f(a+2h) + \ldots + f(a+(n-1)h)$$

gdzie krok h jest szerokością prostokąta i wynosi

$$h = \frac{b-a}{n}, \quad n > 0.$$

Napisana procedura powinna całkować dowolną funkcję w dowolnych granicach a i b. [Spis]

Zadanie 4. Sinus 2

Napisz podprogram w PASCALu, który oblicza $\sin(x)$ korzystając z rozwinięcia:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Liczba członów jakiej należy użyć ma być parametrem podprogramu. Wywołanie $\sin(y, 3)$ powinno dać wartość sinusa w punkcie y obliczoną na podstawie 3-ch pierwszych składników. Przeprowadź analizę dokładności obliczeń. [Spis]