Procedury i funkcje - powtórzenie i uzupełnienia

Przykład funkcji – potęgowanie przy wykładniku naturalnym

```
program potegowanie;
{$APPTYPE CONSOLE}
uses SysUtils;

var     x: real;
        n: integer;

function Potega(podstawa: real; wykladnik: integer): real;
var     wynik: real;
i: integer;
begin
wynik:=1;
for i:=1 to wykladnik do
   wynik:=wynik*podstawa;
potega:=wynik; { przypisanie wyniku pod nazwę funkcji }
end;

begin
writeln('Podaj podstawe: ');
readln(x);
writeln('Podaj wykladnik: ');
readln(n);
writeln(x:8:2, ' do potegi ', n, '  =  ', Potega(x, n):8:2);
writeln('Czekam na Enter…'); readln;
end.
```

Wywołanie dla obliczenia $a^x + b^y$:

```
wyr:= potega(a, x) + potega(b, y)
```

Składnia funkcji

```
function <nazwa_funkcji>(<lista_parametrów_formalnych>) : <typ_wyniku>;
 <deklaracje_lokalne>
begin
<instrukcje>
<nazwa> := <wyrażenie>;        { lub return <wyrażenie>; }
end;
```

Zasady:

- funkcja wyznacza jedną wartość (typu prostego),
- w nagłówku funkcji musi być zdefiniowany typ wyniku funkcji (funkcja ma typ),
- w treści funkcji musi występować przypisanie pod nazwę funkcji (lub instrukcja return)
- w funkcji można deklarować wiele zmiennych lokalnych, które w treści funkcji są wykorzystywane przy zapisie wykonywanego algorytmu

- zmienne lokalne istnieją tylko w czasie wykonywania funkcji, są niedostępne poza funkcją
- funkcja może posiadać dowolną liczbę parametrów
- wywołanie funkcji ma postać:

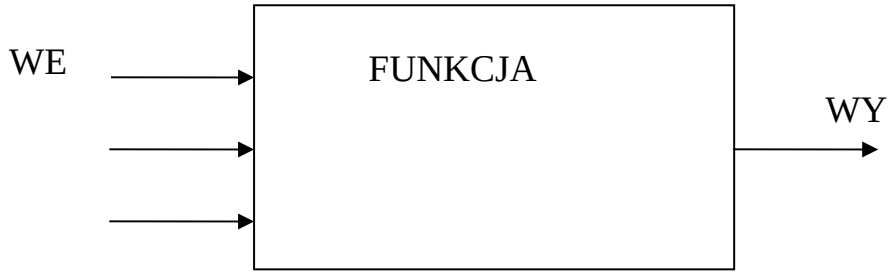
  *<nazwa_funkcji>***(***<lista_parametrów_aktualnych>***)**

  i może występować wszędzie tam gdzie może pojawić się wyrażenie

  np.

```
wyr:=potega(2, 4);
writeln(potega(4, 3));
if potega(47, 3)>1000 then writeln('47 do 3 jest większe od 1000');
```

Funkcję wygodnie jest traktować jako „czarną skrzynkę".



Do wnętrza funkcji można przekazywać wartości tylko przez parametry
Wynik można otrzymać tylko przez nazwę funkcji.
UWAGA.
Przestrzeganie schematu :"czarnej skrzynki" jest tylko umową. W funkcji są dostępne zmienne globalne zdefiniowane w programie głównym. Wartości tych zmiennych można zmieniać w funkcji i zmienione wykorzystywać poza nią - w programie głównym za instrukcją wywołującą. Takie działanie nazywamy „wykorzystywaniem efektów ubocznych".

Przykład procedury – wyznaczanie obwodu i pola trójkąta.

```
var a, b, c, obw, p: real;

procedure poleiobwod (a, b, c: real; var obw, pole: real);
var p: real;
begin
obw:=a + b +c;
p:=obw/2;
pole:=sqrt((p-a)*(p-b)*(p-c)*p);
end;

begin
writeln('Podaj boki');
readln(a, b, c);
poleiobwod(a, b, c, obw, p);
writeln('Obwod = ', obw:8:2);
```

```
writeln('Pole = ', p:8:2);
readln;
end.
```

Składnia procedury

```
procedure <nazwa_procedury>(<lista_parametrów_formalnych>);
 <deklaracje_lokalne>
begin
<instrukcje>
end;
```

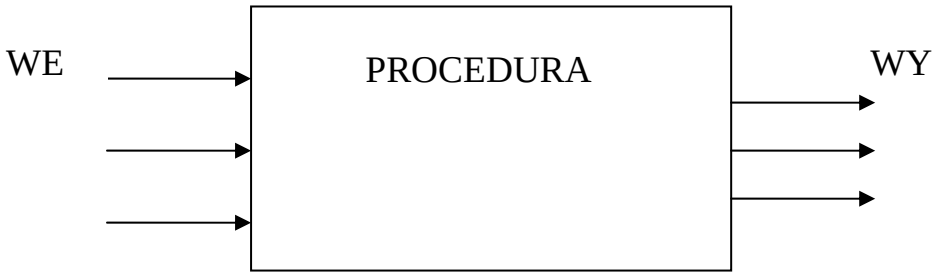
Zasady:

- nazwa procedury nie przenosi wartości – służy jedynie do identyfikacji procedury
- procedura może posiadać dowolną liczbę parametrów
- parametry procedury mogą służyć do przekazywania wyników obliczonych w procedurze
- parametry przekazujące wyniki muszą być poprzedzone na liście parametrów słowem **var**
- w procedurze (podobnie jak w funkcji) można deklarować wiele zmiennych lokalnych, które w treści funkcji są wykorzystywane przy zapisie wykonywanego algorytmu
- zmienne lokalne istnieją tylko w czasie wykonywania funkcji, są niedostępne poza funkcją
- wywołanie procedury ma postać:

  `<nazwa_procedury>(<lista_parametrów_aktualnych>)`

  i jest samodzielną instrukcją

Procedurę wygodnie jest traktować jako „czarną skrzynkę”.



Do wnętrza procedury można przekazywać wartości tylko przez parametry
Wyniki można otrzymać tylko poprzez parametry poprzedzone **var**.

UWAGA.
Podobnie jak w przypadku funkcji przestrzeganie schematu ”czarnej skrzynki” jest tylko umową. W procedurze można również wykorzystywać „efekty uboczne”.

Parametry poprzedzone słowem **var** są przekazywane „przez zmienną”. O pozostałych mówimy, że są „przekazywane przez wartość.”

Oba rodzaje parametrów można wykorzystywać także w zapisie funkcji.

Parametry formalne i aktualne

Parametry z definicji procedury lub funkcji mają postać listy elementów rozdzielanych średnikami. Każdy element listy ma składnię:

```
[var] <nazwa1>, <nazwa2>, ... : <typ>;
```

i może być poprzedzony słowem **var**.
Parametry te nazywamy formalnymi.
Jak widać z opisu składni można określać typ jednocześnie dla całej grupy parametrów.
Specyfikacja typu musi być wyrażona identyfikatorem typu, a nie opisem typu.

W wywołaniu funkcji lub procedury umieszczamy listę parametrów „aktualnych”. Na tej liście musi występować tyle samo parametrów co na liście parametrów formalnych. W czasie realizacji wywołania funkcji lub procedury parametry aktualne i formalne są kojarzone według kolejności występowania. Odpowiadające sobie parametry aktualne i formalne muszą być tych samych rodzajów i zgodnych typów.
Parametry aktualne są rozdzielane przecinkami i muszą być wyrażeniami właściwych typów.
Parametry aktualne kojarzone „przez zmienną” muszą być nazwami zmiennych.

Parametry tablicowe

Parametry procedur mogą być tablicowe jak w przykładzie poniżej:

```
program Tablica;
{$APPTYPE CONSOLE}
uses  SysUtils;

type Tab=array[1..10] of real;
var  n, i: integer;
       suma, srednia: real;
       a: Tab;

procedure czytaj(var n: integer; var x: Tab);
var i: integer;
begin
write('N='); readln(n);
writeln('Podaj ', n, ' liczb');
for i := 1 to n do
   begin
   write('Liczba ', i, ' wynosi: '); readln(x[i]);
   end;
end;

procedure drukuj(n: integer; x: Tab);
var i: integer;
begin
writeln('Tablica ', n, ' elementowa');
for i := 1 to n do
   writeln(x[i]:8:2);
end;
```

```
begin
czytaj(n, a);
suma:=0;
for i := 1 to n do suma := suma+a[i];
srednia:=suma/n;
writeln('Srednia = ', srednia:8:2);
for i := 1 to n do
   a[i]:=a[i]/srednia;
drukuj(n, a);
writeln('Czekam na Enter...'); readln;
end.
```

W przykładzie powyższym konieczne było zdefiniowanie identyfikatora typu tablicowego z uwagi na wymaganie składniowe, które nie dopuszcza specyfikacji parametrów formalnych z użyciem opisu typu.

Własne typu definiuje się w pascalu po słowie **type**. W przykładzie powyższym umieszczono deklarację:

```
type Tab=array[1..10] of real;
```

co pozwoliło na zadeklarowanie tablicy **a** przy pomocy zapisu:

```
var     a: Tab;
```

a ponadto pozwoliło spełnić wymóg składniowy dla opisów nagłówków procedur np.:

```
procedure czytaj(var n: integer; var x: Tab);
```

Programowanie strukturalne

Programowanie strukturalne to metodyka opracowywania, organizacji i pisania programów. Zakłada ona rozwiązywanie złożonych problemów poprzez ich dekompozycję na mniejsze zagadnienia. Każde elementarne zagadnienie zapisywane jest w postaci oddzielnej procedury. Program główny zawiera wtedy jedynie rozmieszczone we właściwej kolejności odwołania do procedur
Podział na zagadnienia elementarne powinien być tak szczegółowy, by rozwiązanie każdego dało się zapisać w postaci procedury mieszczącej się na jednej stronie (ekranie).
Zastosowanie zasad programowania strukturalnego prowadzi do powstawania programów zwięzłych, o przejrzystych i łatwych do analizy strukturach.

Pierwsza aplikacja w Delphi

- automatyczne generowanie szkieletu aplikacji
- aplikacja wyświetla pustą formatkę
- kod źródłowy rozmieszczony w dwu plikach
- plik główny:

```
program Project1;

uses                                    { deklaracja dołączanych bibliotek }
```

```
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);      { generowanie formatki }
  Application.Run;                            { uruchomienie pętli głównej }
end.
```

- plik modułu zawiera deklaracje opisujące formatkę

Rozmieszczanie komponentów na formatce

Generowanie szkieletu metody (procedury) obsługi zdarzenia.

UWAGA. Szczegółowy opis Delphi i przykłady w książce:

K. Strzałkowski „Podstawy Delphi” Wyb. Stachurski. Kielce. 2000

Projektowanie aplikacji zdarzeniowej

Przykład – 2 przyciski i etykieta