Funkcje, Pojęcie podprogramu

Podprogram to pewna wyodrębniona część programu, która sama oddzielnie też mogłaby służyć jako program. Jeśli mówimy, że w programie są podprogamy oznacza to, że program składa się jakby z "małych programików". Można wyróżnić następujące cele przyświecające podziałowi programu na podprogramy:

- wielokrotne korzystanie z ciągu tych samych instrukcji opisanych raz – wewnątrz podprogramu,
- zwiększenie czytelności długich programów,
- uproszczenie zapisu programu i organizacji pamięci dzięki zastosowaniu tzw. rekurencji, czyli wywoływaniu podprogramu przez samego siebie.

W języku C(++) jedyną formą poprogramu jest **funkcja**. (W innych językach programowania - np. w Delphi - wyróżniamy jeszcze procedury)

Rozważmy program, który wczytuje dwie liczby rzeczywiste i liczy ich średnią.

```
  1 #include <iostream.h>
  2
  3 main()
  4 {
  5   double x, y, srednia;
  6   cout << "podaj dwie liczby rzeczywiste: " ;
  7   cin >> x >> y ;
  8
  9   srednia = (x + y)/2 ;
 10
 11   cout << "srednia z liczb " << x <<", " << y << " wynosi " <<
srednia ;
 12 }
```

Oto ten sam program zapisany z użyciem funcji obliczającej średnią.

```
  1 #include <iostream.h>
  2 double srednia(double x, double y)
  3 {
  4   return (x+y)/2;
  5 }
  6
  7 main()
  8 {
  9   double x, y;
 10   cout << "podaj dwie liczby rzeczywiste: ";
 11   cin >> x >> y ;
 12
 13   cout << "srednia z liczb " << x <<", " << y << " wynosi " <<
srednia(x, y) ;
```

```
14 }
```

W poniższym przykładzie deklaracja funkcji została oddzielona od jej definicji:

```
 1 #include <iostream.h>
 2 double srednia(double x, double y); // Deklaracja funkcji
 3 main()
 4 {
 5   double x, y;
 6   cout << "podaj dwie liczby rzeczywiste: ";
 7   cin >> x, y ;
 8
 9   cout << "srednia z liczb" << x <<", " << y << " wynosi " <<
srednia(x, y) ;
10 }
11
12 double srednia(double x, double y)
13 /* Definicja funkcji */
14 {
15   return (x+y)/2;
16 }
```

Istotne jest, by deklaracja funkcji poprzedziła jej wywołanie - natomiat definicja może nastąpić po wywołaniu funkcji.

Zauważmy, że funkcja, która zwraca pewną wartość (tak jak funkcja *srednia* zwraca wynik będący średnią dwóch liczb), może być wywołana wewnątrz wyrażenia - w powyższym przykładzie było to wyrażenie wypisywania wartości.

Funkcja w C nie musi zwracać żadnej wartości. Poniższa funkcja **tab** dostaje na wejściu liczbę całkowitą *n* i wypisuje jej wielokrotności (od 1 do 10) (wypisuje na ekranie, ale po wypisaniu "zapomina" i nie zwraca tej wartości). Zwróć uwagę na słówko kluczowe **void**, które oznacza, że funkcja nie zwraca wartości. Taka funkcja nie może być wywoływana jako część wyrażenia. Wywołanie funkcji *tab* musi być w osobnej instrukcji:

```
 1 void tab(int n)
 2 {
 3   int i;
 4
 5   for ( i = 1; i <= 10; i++ )
 6     cout << n << " " << i << " " << n*i <<"\n";
 7 }
 8 main()
 9 {
10 tab(5) ;
11 }
```

Funkcja może nie mieć żadnych parametrów, np.

```
1 void gwiazdki(void)
2 {
3   cout <<"**********************" ;
4 }
```

Taką funkcję wywołujemy tak, że w nawiasach nie podajemy parametrów

```
1 main()
2 {
3 gwiazdki();
4 }
```

Ogólnie definicja funkcji ma postać:

typ wartości nazwa--funkcji (parametry) {

- deklaracje instrukcje

}

Parametrami funkcji mogą być także tablice. W deklaracji funkcji możemy nie specyfikować rozmiaru tablicy. Oto przykład zastosowania parametru - tablicy:

```
1 int sum_tab(int a[], int n) ;
2 int main()
3 {
4         int t[4] = {1,2,3,4} ;
5         cout << "Suma pierwszych 3 elementów tablicy wynosi: "
<< sum_tab(t, 3) ;
6 }
7 //-------------------------------------------------------------
-------------
8 int sum_tab(int a[], int n)
9 {
10   int i, sum = 0;
11
12   for (i = 0; i < n; i++)
13     sum += a[i];
14
15   return sum;
16 }
```

Gdy chcemy parametrem uczynić tablicę dwuwymiarową, musimy podać liczbę kolumn tablicy:

```
1 #define L_kol 10
2
3 int sum_tab(int a[][L_kol], int n)
4 {
```

```
  int i, j, sum = 0;

  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      sum += a[i][j];

  return sum;
}
```

Pamiętajmy, że każda funkcja nie-void musi zwracać wartość - niezbędne jest więc podanie instrukcji **return**, która służy do zwracania wartości. Instrukcja return ma postać:

```
return wyrażenie;
```

Funkcja main() również może posiadać argumenty wywołania, mogą nimi być np parametry z jakimi został uruchomiony program. Ma wtedy następującą postać

```
int main(int argc, char* argv[])
```

W zmiennej **argc** znajduje się liczba parametrów z jakimi został uruchomiony program, a zmienna **argv** zawiera tablicę wszystkich parametrów (zerowy element zawiera nazwę programu).

np. Jeśli uruchomimy nasz z linii poleceń: program p1 p2 p3 to argc będzie równe 3, argv[0]='Program', argv[1]='P1', argv[2]='P2', argv[3]='P3'.

```
#include <stdio.h>
/* Program wypisuje wszystkie parametry wykonania */

int main(int argc, char* argv[])
{
 int i;
 for (i=0; i < argc; i++)
     printf("%s\n", argv[i]);
 return 0;
}
```