Funkcje

Często w programach spotykamy się z sytuacją, kiedy chcemy wykonać określoną czynność kilka razy np. dodać dwie liczby w trzech miejscach w programie. Oczywiście moglibyśmy to zrobić pisząc trzy razy ten sam kod tylko dla różnych danych. Domyślamy się jednak, że taki sposób nie jest najlepszy. Aby rozwiązać tego rodzaju problem skorzystamy z tzw. Funkcji. Jest to fragment kodu, który jest wpisywany raz, ale może być wykonywany wielokrotnie. Realizuje on najczęściej jakąś pojedynczą czynność przy użyciu ustalonego przez programistę algorytmu. Budowa funkcji wygląda następująco:

```
Typ_zwracany_przez_funkcje nazwa_funkcji(lista_parametrow)
      {
       Blok funkcji;
       return wartosc_zwracana;
      }
```

Omówimy teraz krótko różne możliwości budowy funkcji. Zacznijmy od tego, że już od samego początku korzystaliśmy (może nieświadomie) z jednej funkcji, a mianowicie z funkcji głównej programu `main()`. Funkcja ta jest przykładem funkcji, która nie zawiera żadnych parametrów i zwraca liczbę całkowitą równą 0. Może się też zdarzyć, że funkcja będzie bezparametrowa i nie będzie nic zwracać (wtedy typ zwracany takiej funkcji to `void` i w ciele funkcji nie ma słowa kluczowego `return`). Tego rodzaju funkcje najczęściej wykorzystuje się do wypisania jakiejś informacji na ekranie monitora. Przykładem takiej funkcji jest funkcja użyta w poniższym programie.

Zadanie 1.
Stworzyć plik `funkcje1.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
void pokazTekst();
int main()
{
   pokazTekst();
   system("pause");
   return 0;
}
void pokazTekst()
{
   cout << "Moja pierwsza funkcja\n";
}
```

następnie skompilować plik oraz uruchomić program.

Na pierwszy rzut oka widać, że tego typu funkcje są mało użyteczne. Chodzi oczywiście o brak parametrów oraz zwracanej wartości. Dodając np. dwie liczby chcemy, aby obliczona przez funkcję suma była dalej wykorzystana w programie. Funkcja zatem musi zwrócić jej wartość, czyli musi być określony typ zwracany tej funkcji. Z kolei, żeby obliczyć tę sumę funkcja musi mieć dane potrzebne do jej obliczenia, czyli dwie liczby. Owe dwie liczby będą zatem przechowywane w zmiennych określonego typu zwanych parametrami funkcji. Parametry oddzielone są przecinkiem. Na podstawie tej prostej funkcji obliczającej sumę widać zatem, że parametry rozszerzają użyteczność i możliwości funkcji tak znacznie, że bez nich w zasadzie trudno wyobrazić sobie skuteczne i efektywne programowanie. Możemy powiedzieć więc, że **parametry funkcji** to dodatkowe dane, przekazywane do funkcji podczas jej wywołania.
Pełnią rolę dodatkowych zmiennych wewnątrz funkcji i można ich używać podobnie jak innych zmiennych,

zadeklarowanych w niej bezpośrednio. Różnią się one oczywiście tym, że wartości parametrów pochodzą z „zewnątrz” – są im przypisywane podczas wywołania funkcji. Poniższy program obrazuje to, co prze chwilą powiedziałem.

Zadanie 2.
Stworzyć plik `suma.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
int dodaj(int a, int b);

int main()
{
   int liczba1 = 10;
   int liczba2 = 15;
   cout<<"Suma danych liczb wynosi "<<dodaj(liczba1, liczba2)<<endl;

   int x;
   cout << "Podaj pierwsza liczbe: ";
   cin >> x;

   int y;
   cout << "Podaj druga liczbe: ";
   cin >> y;

   int Suma=dodaj(x,y);
   cout<<"Suma liczb x i y wynosi "<<Suma<<endl;
   system("pause");
   return 0;
}
int dodaj(int a, int b)
{
   int suma = a + b;
   return suma;
}
```

następnie skompilować plik oraz uruchomić program.

Jeśli nie chcemy dalej w programie wykorzystywać obliczonej sumy, a interesuje nas tylko informacja ile ona wynosi , możemy napisać taki program:

Zadanie 2'.
Stworzyć plik `suma1.cpp` i wpisać w nim następujący kod:

```
#include <iostream>
using namespace std;
```

```
void dodaj(int a, int b);


int main()
{
   int x;
   cout << "Podaj pierwsza liczbe: ";
   cin >> x;


   int y;
   cout << "Podaj druga liczbe: ";
   cin >> y;


   dodaj(x,y);
   system("pause");
   return 0;
}
void dodaj(int a, int b)
{
   int suma = a + b;
   cout<<"Suma wczytanych liczb wynosi "<<suma<<endl;
}
```

Spójrzmy teraz na napisane wcześniej trzy programy. W każdym z nich, przed funkcją główną `main()`, występuje deklaracja funkcji (inaczej prototyp funkcji). Dzięki temu kompilator jest informowany, że w kodzie występuje definicja funkcji (warto zauważyć, że deklaracja funkcji zakończona jest średnikiem). Widzimy także, że na samym końcu kodu (poza programem) umieszczona jest definicja funkcji (można ją umieścić również przed funkcją `main()`). Wreszcie w samym programie następuje wywołanie zdefiniowanej wcześniej funkcji. Wywołanie następuje poprzez nazwę funkcji. W przypadku funkcji, która zwraca jakąś wartość jej wywołanie możemy zrealizować na dwa sposoby (patrz zadanie 2): bezpośrednio przez nazwę lub przez przypisanie jej wartości do nowej zmiennej i dalej operowanie już na tej zmiennej.

Warto zwrócić uwagę jeszcze na dwa fakty. Po pierwsze parametry funkcji użyte w jej definicji są zmiennymi lokalnymi, czyli można z nich korzystać tylko w obrębie funkcji. W zadaniu 2 są to parametry `a` i `b`. W samym programie możemy używać natomiast innych zmiennych (w zadaniu 2 są to `liczba1`, `liczba2` lub `x,y`), pamiętając, że przy wywoływaniu funkcji odnosimy się właśnie do tych zmiennych. Po drugie funkcja nie może zwracać danych w postaci tablicy (ale tablica może być parametrem funkcji). Oto przykład deklaracji oraz wywołania funkcji, której parametrem jest tablica:

```
int jakas_funkcja(int a[], int wymiar_tablicy); - deklaracja
int zmienna = jakas_funkcja(x,n); - wywołanie w programie, gdzie x jest tablicą, a
                                    n jej wymiarem (x i n są dane)
```

Zadanie 3.
Napisać funkcję o nazwie **`trojmian`**, która zwraca liczbę pierwiastków równania kwadratowego postaci ax^2+bx+c=0 (założyć, ze a jest różne od zera). Funkcja pobiera współczynniki a, b, c. Następnie wykorzystać ją w przykładowym programie.

Zadanie 4.
Napisać funkcję o nazwie **potega**, która oblicza n-tą potęgę liczby 2. Funkcja pobiera wykładnik n i zwraca potęgę liczby 2 o wykładniku n. Przetestować tę funkcję w przykładowym programie.

Zadanie 5.
Napisać funkcję **pole**, która pobiera trzy liczby rzeczywiste a, b, c reprezentujące długości boków trójkąta i zwraca jego pole obliczone ze wzoru Herona: sqrt(p*(p-a)*(p-b)*(p-c)), gdzie p jest połową obwodu trójkąta. Za pomocą tej funkcji obliczyć pole przykładowego trójkąta.

Zadanie 6.
Napisać funkcję o nazwie **wybraneLiczby**, która pobiera dwie liczby całkowite dodatnie reprezentujące końce przedziału liczbowego i zwraca ilość liczb całkowitych z tego przedziału, które są podzielne przez 3 i 4.

Zadanie 7.
Napisać dwie funkcje o nazwach **pole, obwod,** które pobierają dwie liczby rzeczywiste reprezentujące długości boków prostokąta i zwracają odpowiednio jego pole oraz obwód. Przetestować te funkcje w przykładowym programie.

Zadanie 8.
Napisać funkcję **silnia**, która pobiera liczbę naturalną `n` i zwraca silnię tej liczby. Następnie wykorzystać ją w programie, który będzie obliczał wartości funkcji `sin x, cos x, e^x` (liczbę x podaje użytkownik) z zadaną dokładnością `epsilon` w oparciu o definicję tych funkcji za pomocą szeregów.

Zadanie 9.
Napisać funkcję **norma,** która pobiera 20-elementową tablicę liczb rzeczywistych reprezentującą wektor oraz jako drugi parametr liczbę całkowitą n reprezentującą wymiar tego wektora. Funkcja zwraca normę tego wektora. Następnie należy wykorzystać ową funkcję w programie, który po wczytaniu wymiaru i współrzędnych wektora znormalizuje go (wypisana zostanie za pomocą tablicy postać wektora znormalizowanego).

Rekurencja

Rekurencja to wywołanie funkcji wewnątrz jej definicji. Spójrzmy na przykład.

```
#include <iostream>
using namespace std;
double potega(int n);
int main()
{
   int wykladnik;
   cout<<"Podaj wykladnik naturalny ";
   cin>>wykladnik;
   double P = potega(wykladnik);
   cout<<wykladnik<<" - potega liczby 2 wynosi "<<P<<endl;
   system("pause");
   return 0;
}
double potega(int n)
{
```

```
   if(n==0)
     return 1;
   else
     return 2*potega(n-1);
}
```

Jeśli np. n = 4, to funkcja `potega` wywoła się pięć razy. Otrzymamy kolejno:

```
potega(4) → 2*potega(3) → 2*(2*potega(2)) → 2*(2*(2*potega(1))) →
2*(2*(2*(2*potega(0)))) = 2*2*2*2*1 = 16.
```

Zadanie 10.
Napisać za pomocą rekurencji funkcję obliczającą silnię danej liczby.

Zadanie 11.
Napisać za pomocą rekurencji funkcję znajdującą n – ty wyraz ciągu Fibonacciego postaci:

```
a(1) = a(2) = 1; a(n) = a(n-1) + a(n-2).
```

Przeciążanie funkcji

C++ umożliwia tworzenie wielu funkcji o tej samej nazwie i o tym samym przeznaczeniu. Mechanizm ten nazywa się przeciążaniem lub przeładowaniem funkcji. Listy parametrów funkcji przeciążonych muszą się różnić od siebie albo typami parametrów, albo ich ilością, albo jednocześnie typami i ilością. Na przykład:

```
float Suma(int,int);
float Suma(double,double);
float Suma(int,int,int);
```

Najczęściej funkcji przeciążonych używa się, gdy chcemy wykonać tę samą czynność na różnych typach danych lub różnej ilości danych.

```
float Suma(int a, int b)
{
   return a + b;
}
float Suma(double a, double b)
{
   return a + b;
}
float Suma(int a, int b, int c)
{
   return a + b + c;
}
```