Języki programowania

Funkcje

Funkcje (1)

Podprogram – pozwala na definiowanie własnych „instrukcji”. Podprogramy w języku C++ nazywane są funkcjami.

Funkcja – podprogram o swojej własnej nazwie, który może być wywołany z innej części programu dowolną liczbę razy.

Składnia funkcji:

nagłówek funkcji

```
zwracany_typ nazwa([parametry]) {

        instrukcje;
        ...
        [return [obiekt_zwracany]];
}
```

Przykłady nagłówków funkcji:

```
double objWalca double r, double h);
```
lub
```
double objKuli(double, double);
```

```
void echo(char znak, unsigned int ile);
```
lub
```
void echo(char, unsigned int);
```

```
void napis();
```
lub
```
void napis(void);
```

Uwaga: na końcu każdej deklaracji znajduje się **średnik**!!!

Funkcje (2)

- Przed **odwołaniem** się do nazwy funkcji wymagana jest jej **deklaracja** (niekoniecznie definicja).
- Sama funkcja może być zdefiniowana później. **Definicja funkcji** jest równoznaczna z napisaniem jej treści.

```
float pole(float);

void main() {
  ...
  cout << pole(7.3) << endl;
}

float pole(float r) {
  return 3.14159f * r * r;
}
```

deklaracja

wywołanie funkcji

definicja

- Funkcja wypisująca na ekranie komunikat:

```
void napis() {
  cout << "Komunikat\n";
}
```

- Funkcja wypisująca na ekranie znak zadaną ilość razy:

```
void echo(char znak, unsigned int ile) {
  while(ile--)
    cout << znak;
}
```

- Funkcja obliczająca objętość walca:

```
double objKuli(double r, double h) {
  double pole_podst = 3.1415 * r * r;
  return h * pole_podst;
}
```

Zwracanie wyniku przez funkcję

Funkcja zwraca wartość przez instrukcję **`return`** ***`wyrażenie`***.
`Wyrażenie` musi być typu takiego, jaki jest typ zwracany przez funkcję lub dającego się skonwertować do typu zwracanego

```
double objWalca(double r, double h) {
  double pole_podst = 3.1415 * r * r;
  return h * pole_podst;
}
```

wyrażenie

Jeśli funkcja zwraca typ **`void`**, czyli nie zwraca niczego, to wewnątrz funkcji **nie można** użyć instrukcji **`return wyr`**, a jedynie samą instrukcję **`return`**.

Funkcja zwraca wartość wyrażenia, i sama może być użyta w wyrażeniu:

```
a + objWalca(5,6) *0.1
```

Parametry formalne funkcji i aktualne wywołania funkcji

parametry (argumenty) formalne

Definicja funkcji:

```
double objWalca(double r , double h ) {
  double pole_podst = 3.1415 * r * r;
  return h * pole_podst;
}
```

Wywołanie funkcji w programie:

```
#include <iostream.h>
//.................
double promien, wysokosc;
//.................
promien=3.4;  wysokosc=5.8;
cout<<objWalca(promien,wysokosc)<<endl;
cout<<objWalca(2,10.5)<<endl;
```

parametry (argumenty) aktualne wywołania funkcji

Argumenty funkcji – przesyłanie przez wartość (1)

Do funkcji przesyłana jest **wartość liczbowa argumentu aktualnego** wywołania funkcji. Wartość ta służy do inicjalizacji parametru formalnego. Funkcja pracuje na kopii argumentu aktualnego. Po opuszczeniu funkcji kopia znika, wobec tego znika też to, co zostało zrobione w funkcji.

```
#include <iostream.h>
void dodaj5(int a);
void main(){
  int b=2;
  cout<<"Przed wywolaniem funkcji dodaj5 parametr aktualny = " <<b<<endl;
  dodaj5(b);
  cout<<"\nPo wywolaniu funkcji dodaj5 parametr aktualny = " <<b<<endl;
}
void dodaj5(int a){
  a+=5;
  cout<<"\nW funkcji dodano do parametru formalnego 5"<<endl;
  cout<<"arg formalny = "<<a<<endl;
}
```

Wynik działania programu:

Przed wywolaniem funkcji dodaj5 parametr aktualny = 2
W funkcji dodano do parametru formalnego 5
arg formalny = 7
Po wywolaniu funkcji dodaj5 parametr aktualny = 2

Argumenty funkcji – przesyłanie przez referencję

Przesyłanie argumentów funkcji przez referencję pozwala funkcji na modyfikowanie zmiennych znajdujących się pozą tą funkcją. Przy przekazywaniu argumentu przez referencję przesyłany jest **adres argumentu wywołania funkcji**.

```
#include <iostream.h>
void dodaj5(int a, int &aa);
void main(){
  int b=2, bb=10;
  cout<<"Przed wywolaniem f. dodaj5 p. akt. 1 = "<<b<<endl;
  cout<<"Przed wywolaniem f. dodaj5 p. akt. 2 = "<<bb<<endl;
  dodaj5(b,bb);
  cout<<"\nPo wywolaniu f. dodaj5 p. akt. 1 = "<<b<<endl;
  cout<<"Po wywolaniu f. dodaj5 p. akt. 2 = "
        <<bb<<endl;
}
void dodaj5(int a, int &aa){
   a+=5; aa+=5;
   cout<<
    "\nW funkcji dodano do p. form. liczbe 5\n"
   cout<<"arg formalny 1 = "<<a<<endl;
   cout<<"arg formalny 2 = "<<aa<<endl;
}
```

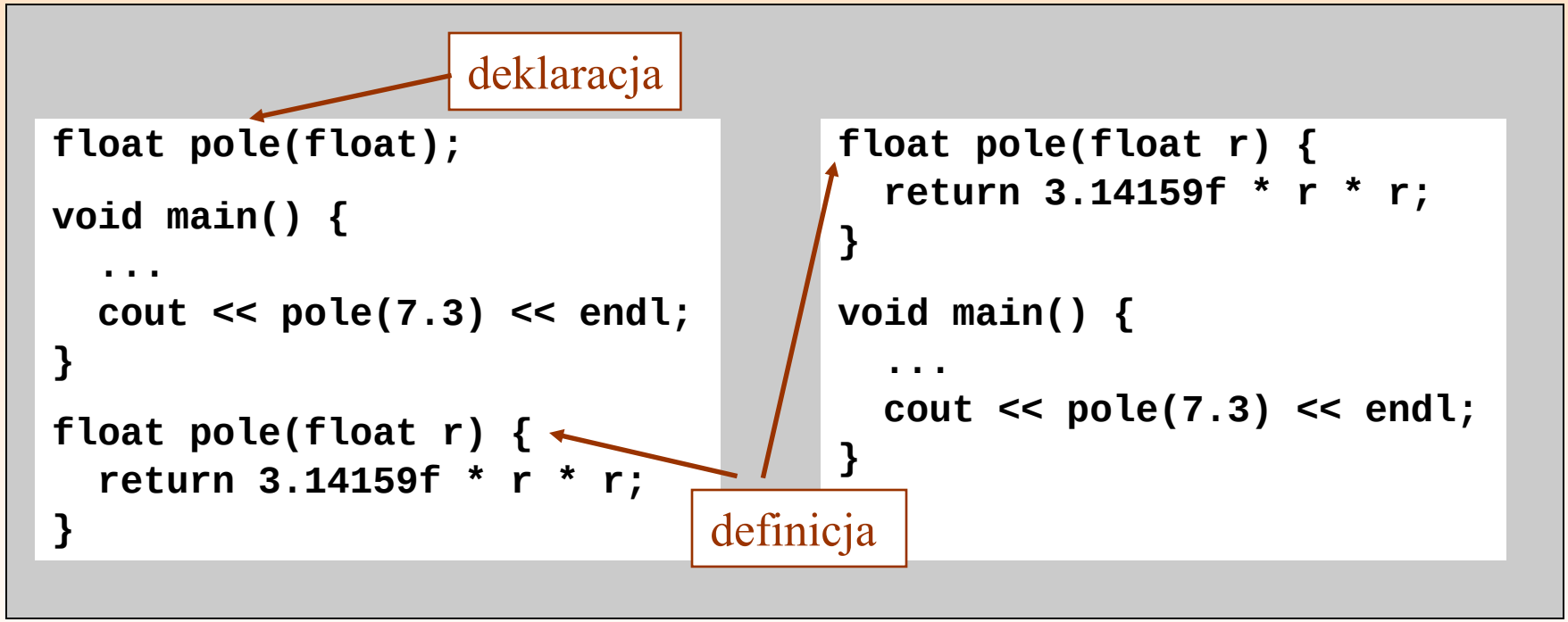
Wynik działania programu:

Przed wywolaniem f. dodaj5 p. akt. 1 = 2
Przed wywolaniem f. dodaj5 p. akt. 2 = 10

W funkcji dodano do p. form. liczbe 5
arg formalny 1 = 7
arg formalny 2 = 15

Po wywolaniu f. dodaj5 p. akt. 1 = 2
Po wywolaniu f. dodaj5 p. akt. 2 = 15

Czy deklaracja funkcji jest zawsze konieczna?



Z deklaracją

Definicja funkcji poniżej jej wywołania

Bez deklaracji

Definicja funkcji przed jej wywołaniem

Argumenty domniemane (1)

argument domniemany

```
#include <iostream.h>
double obwod(double a,double b, int p=0 );
void main(){
        double x=2, y=10;
        cout<< obwod(x,y)   <<" standardowo w milimetrach\n";
        cout<< obwod(x,y,1) <<" w centymetrach\n";
        cout<< obwod(x,y,2) <<" w metrach\n";
}

double obwod(double a,double b,int p){
        double result=2*(a+b);
        switch(p){
          case 1: result *= 0.1; break;
          case 2: result *= 0.001; break;
          default: break;
        }
        return result;
}
```

wywołanie funkcji
z domniemanym argumentem

Wynik działania programu:
24 standardowo w milimetrach
2.4 w centymetrach
0.024 w metrach

Uwaga: Kompilator jest informowany, że argument jest domniemany tylko raz, przy deklaracji funkcji. Jeśli definicja funkcji podana jest później, to informacja nie jest powtarzana.

Argumenty domniemane (2)

Kilka argumentów domniemanych

argumenty domniemane

```
int funkcja (int i, int j, int a=3, int b=4, int c=2);
```

Możliwe wywołania funkcji:

```
funkcja(0,1);              // a=3, b=4, c=2
funkcja(0,1,8);            // a=8, b=4, c=2
funkcja(0,1,9,0)           // a=9, b=0, c=2
funkcja(0,1,5,-3, 1)       // a=5, b=-3, c=1
```

Niemożliwe jest opuszczenie argumentu domniemanego **b** i użycie argumentu **c**!!!

```
funkcja(0,1,5,, 1)
```

Funkcje `inline`

Funkcje **`inline`** powinny być stosowane dla funkcji, których treść jest bardzo krótka, i które wywoływane są w programie wiele razy.

Kompilator napotykając funkcję zadeklarowaną w ten sposób, umieszcza treść tej funkcji tam, gdzie jest ona wywoływana.

Zaleta:
• nie traci się czasu, potrzebnego na wywołanie funkcji
Wada:
• rozmiar programu może ulec zwiększeniu

Przykład:

```
inline double przelicz(double x) //funkcja przelicza metry na milimetry
{
        return (x*1000.0);
}
......
y = przelicz(x)*k + przelicz(z)/p;
......
```

Uwaga! Jeśli funkcja jest typu **`inline`**, to jej **definicja** musi być umiejscowiona przed jej pierwszym użyciem. Sama deklaracja nie wystarczy!

Zmienne lokalne statyczne

Słowo kluczowe **`static`** przed zmienną wewnątrz funkcji pozwala na zdefiniowanie zmiennej, która *po zakończeniu działania funkcji nie jest „usuwana z pamięci" ale zachowuje swoje wartości do następnego wywołania funkcji*.

- **Zmienne globalne** – są wstępnie inicjalizowane zerami
- **Zmienne lokalne** – nie są wstępnie inicjalizowane, zawierają „śmieci"
- **Zmienne lokalne statyczne** – są wstępnie inicjalizowane zerami i zajmują w pamięci ten sam obszar, co zmienne globalne

Definiowanie zmiennych statycznych:

```
static typ nazwa_zmiennej;
```

```
void f1(){
        static int ile_razy=0; //lub static int ile_razy;
        ile_razy++;
        cout<<"Wywolanie funkcji f1: "<<ile_razy<<endl;
}
void main(){
        int i;
        for(i=0;i<3;i++)
                f1();
}
```

Wynik działania programu:

```
Wywolanie funkcji f1: 1
Wywolanie funkcji f1: 2
Wywolanie funkcji f1: 3
```

Funkcje i zmienne w programie składającym się z kilku plików

Funkcje i zmienne programu mogą być umieszczone w kilku plikach.

Aby funkcje z pliku B miały dostęp do funkcji i zmiennych globalnych z pliku A, trzeba w pliku B zamieścić **deklaracje** tych zmiennych i funkcji.

```
//plik A - definicje
int n;
double x;
void f1(){
        static int ile_razy=0;
        ile_razy++;
}
```

```
//plik B - deklaracje
extern int n;
extern double x;
void f1();
```

Najczęściej wszystkie deklaracje umieszczane są w pliku nagłówkowym, który bezpośrednio przed kompilacją jest włączany do pliku przez dyrektywę `#include`

```
//plik nagl.h
extern int n;
extern double x;
void f1();
```

```
//plik B
#include "nagl.h"
```

Nazwa ujęta w cudzysłów informuje kompilator, że plik nagłówkowy znajduje się w bieżącym katalogu

Globalne nazwy statyczne

Postawienie przed zmienną globalną słowa kluczowego **`static`** jest informacją dla kompilatora, że nazwa tej zmiennej ma być **niewidoczna** w innych plikach, z których składa się dany program.

Rekurencja

Rekurencja – wywoływanie funkcji przez samą siebie

Przykłady:

Obliczanie silni:
Def. silni
jeśli n=0 to silnia=1
jeśli n>0 silnia=n*silnia(n-1)

```
unsigned int silnia(unsigned int n){
   unsigned int wynik;
   if(n) wynik=n*silnia(n-1);
   else  wynik=1;
   return wynik;
}
```