Programowanie 1

Sławomir Pluciński (splucinski@pjwstk.edu.pl)

- **Funkcja** jest to fragment programu, któremu nadano nazwę i który możemy wykonać poprzez podanie jego nazwy oraz ewentualnych argumentów (o ile istnieją). Argumentami są natomiast dane przekazywane do funkcji, które są opcjonalne.

```
typ_zwracany nazwa(typ_a nazwa_a_1 /*,...*/, typ_a_n nazwa_a_n ){
    return zwracana_wartosc;
}
```

```
void nazwa(typ_a nazwa_a_1 /*,...*/, typ_a_n nazwa_a_n ){
    //kod aplikacji
}
```

- **Programowanie proceduralne**

paradygmat programowania zalecający dzielenie kodu na procedury, czyli fragmenty wykonujące ściśle określone operacje. Procedury nie powinny korzystać ze zmiennych globalnych (w miarę możliwości), lecz pobierać i przekazywać wszystkie dane (czy też wskaźniki do nich) jako parametry wywołania.

Zasięg zmiennych

Należy zwrócić uwagę, że deklarując zmienne w ramach jednej funkcji są one dostępne jedynie dla ciała tej funkcji. Tego rodzaju zmienne nazywamy zmiennymi lokalnymi. W ramach naszego przykładu, zmienna a i b z funkcji dodawanie nie będzie dostępna za po tą właśnie funkcją.

```
#include <iostream>

using namespace std;

void dodawanie();

int main() {
    dodawanie();
    return 0;
}

void dodawanie() {
    int a = 2;
    int b = 2;
    cout << "a + b = " << a+b;
}
```

Istnieje możliwość deklaracji zmiennych, jako zmienne globalne – dostępne dla całego programu. Jeżeli chcemy zdefiniować takiego rodzaju zmienne to należy je umieścić po za ciałem jakiejkolwiek z funkcji.

```cpp
#include <iostream>

using namespace std;

void dodawanie();
int a = 2;
int b = 2;

int main() {
    dodawanie();
    return 0;
}

void dodawanie() {
    cout << "a + b = " << a+b;
}
```

Zmienne do funkcji można również przekazać. Należy wtedy w ramach nagłówka funkcji opisać jakiego rodzaju parametry będą przekazywane, a przy wywołaniu funkcji wymagane będzie przekazanie parametrów w ilości i poprawnym typie, tak jak zostało to zadeklarowane w nagłówku.

```
#include <iostream>

using namespace std;

void dodawanie(int x, int y);


int main() {
    int a = 2;
    int b = 2;
    dodawanie(a,b);
    return 0;
}

void dodawanie(int x, int y) {
    cout << "a + b = " << x+y;
}
```

Domyślnym sposobem przekazywania do funkcji parametrów, jest przekazywanie parametru przez **wartość**. W ciele funkcji tworzona jest kopia zmiennej i funkcja zaczyna operować na swojej lokalnej kopii, nie zmieniając wartości oryginalnej.

```cpp
#include <iostream>

using namespace std;

void dodawanie(int x, int y);


int main() {
    int a = 2;
    int b = 2;
    dodawanie(a,b);
    return 0;
}

void dodawanie(int x, int y) {
    cout << "a + b = " << x+y;
}
```

Alternatywnie do funkcji możemy przekazać parametr przez **referencje** to znaczy, że przekazujemy adres „szufladki” w której zapisana jest zmienna, przez co metoda nie musi tworzyć sobie lokalnej kopii, tylko może pracować na wartości oryginalnej. Do przekazania zmiennej przez referencje służą wskaźniki.

```
#include <iostream>

using namespace std;

void dodawanie(int &x, int &y);


int main() {
    int a = 2;
    int b = 2;
    dodawanie(a,b);
    return 0;
}

void dodawanie(int &x, int &y) {
    cout << "a + b = " << x+y;
}
```

- Procedura

Taka funkcja, która nic nie zwraca (jest typu void) nazywamy procedurą. Zwykle będziemy korzystać z procedur w sytuacji kiedy potrzebujemy zmienić stan jakiejś zmiennej lub wyświetlić gotowy wynik na ekran.

```
#include <iostream>

using namespace std;

void dodawanie();
int a = 2;
int b = 2;

int main() {
    dodawanie();
    return 0;
}

void dodawanie() {
    cout << "a + b = " << a+b;
}
```

- Zwracanie wartości przez funkcję.

Funkcja prócz tego, że może być typu void (nie zwracać nic) to może też zwracać jakąś wartość. Aby to osiągnąć w deklaracji funkcji należy wskazać jaki typ będzie ta funkcja zwracała, a w ciele funkcji użyć return po którym wskazujemy co powinno zostać zwrócone.

```
#include <iostream>

using namespace std;

int dodawanie(int x, int y);


int main() {
    int a = 2;
    int b = 2;
    cout << "a + b = "<< dodawanie(a,b);
    return 0;
}

int dodawanie(int x, int y) {
     return x+y;
}
```

- Zagnieżdżenie funkcji

W jednej funkcji możemy wywoływać inne funkcje i takiego rodzaju zagnieżdżenia są możliwe bez ograniczenia stopnia zagnieżdżenia.

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

int dodawanie(int x, int y);
int odejmowanie(int x, int y);

int main() {
    int a = 2;
    int b = 2;
    cout << "a - b = "<< odejmowanie(a,b);
    return 0;
}

int dodawanie(int x, int y) {
     cout << "Tu nic nie ma";
     return 0;
}

int odejmowanie(int x, int y) {
    return dodawanie(x,y);
}
```

Przypisanie do zmiennej wartości zwracanej przez funkcję.

Wynik, który jest zwracany przez funkcję nie musi zostać od razu wykorzystane. Można go zapisać do innej zmiennej – ważne, żeby typ zmiennej i wartości zwracanej przez funkcję się zgadzał.

```
#include <iostream>

using namespace std;

int dodawanie(int x, int y);

int main() {
    int a = 2;
    int b = 2;
    int wynik = dodawanie(a,b);
    cout << "a + b = "<< wynik;
    return 0;
}

int dodawanie(int x, int y) {
     return x+y;
}
```

Deklaracja i definicja funkcji nie muszą być rozłączone. Zaraz po deklaracji, można napisać też definicję funkcji. Jednak z uwagi na czytelność, zwykle przed metodą main() piszemy tylko nagłówek funkcji, a jej ciało po zakończeniu metody main().

```
#include <iostream>

using namespace std;
int a = 2;
int b = 2;

void dodawanie() {
    cout << "a + b = " << a+b;
}

int main() {
    dodawanie();
    return 0;
}
```

Wykorzystanie funkcji z instrukcją warunkową

Wynik działania funkcji możemy również wykorzystać od razu w funkcji warunkowej, jedyne wymaganie jest takie, żeby funkcja zwracała wartość typu bool.

```
#include <iostream>

using namespace std;

bool isPrime(int x);

int main() {
    if(isPrime(4)){
        cout << "TAK";
    }
    else {
        cout << "NIE";
    }

    return 0;
}

bool isPrime(int x){
    return x%2==0;
}
```

Ciekawostka

Instrukcję If możemy zapisać w formie skróconej i będzie miała ona taki sam rezultat jak na poprzednim slajdzie.

```
#include <iostream>

using namespace std;

bool isPrime(int x);

int main() {
    isPrime(4)? cout <<"TAK": cout <<"NIE";
    return 0;
}

bool isPrime(int x){
    return x%2==0;
}
```

Rekurencja

Rekurencja zwana rekursją, polega na wywołaniu przez funkcję samej siebie.

silnia(5) = 5*silnia(4)

= 5*(4*silnia(3))

= 5*(4*(3*silnia(2)))

= 5*(4*(3*(2*silnia(1))))

= 5*(4*(3*(2*(1*silnia(0)))))

= 5*(4*(3*(2*(1*1))))

= 5*(4*(3*(2*1)))

= 5*(4*(3*2))

= 5*(4*6)

= 5*24

= 120

```cpp
#include <iostream>

using namespace std;

int silnia (int liczba){
    if (liczba < 2) {
        return liczba;
    }
    return liczba * silnia(liczba - 1);
}

int main(){
    cout << silnia(5) << endl;

    return(0);
}
```

Niebezpieczeństwo rekurencji

- nieskończona liczba wywołań rekurencyjnych
- wielokrotne wykonanie identycznych obliczen
- przepełnienie stosu
- bardzo abstrakcyjny zapis

Po co funkcje?

- unikanie powtarzania kodu
- podział aplikacji na funkcjonalności
- łatwiejsze lokalizowanie i poprawianie błędów
- ponowne wykorzystanie raz napisanego kodu
- ułatwienie w utrzymaniu