Programowanie I
Wykład 3

dr inż. Rafał Brociek

Wydział Matematyki Stosowanej
Politechnika Śląska

18.10.2021

Funkcje

Funkcja (podprogram, procedura) – fragment kodu posiadający nazwę, który możemy wielokrotnie wywoływać (podając jego nazwę) z różnych miejsc programu.
Dzięki umiejętnemu wykorzystaniu funkcji kod staje się czytelniejszy oraz łatwiejszy w utrzymaniu.
Definiując funkcję w języku C++ musimy podać:

- nazwę funkcji,
- typ zwracany przez funkcję (`void` w przypadku, gdy funkcja nie zwraca żadnej wartości),
- parametry oraz ich typy, o ile funkcja takie przyjmuje.

Funkcje

Deklaracja funkcji - schemat

```
typ_zwracany nazwaFunkcji(typ_par1 nazwa_par1, typ_par2 nazwa_par2);
void nazwaFunkcji(void);
void nazwaFunkcji();
```

Przykłady

```
double poleKola(double r);
bool czyKoniecGry(int wynik);
double oblicz(int nrProbki, double cisnienie, double temperatura);
void rysujRamke(void);
void rysujRamke();
```

Przed odwołaniem się do funkcji musi być znana jej deklaracja.

Funkcje

Definicja funkcji - schemat

```
typ_zwracany nazwaFunkcji(typ_par1 par1, typ_par2 par2)
{
    typ_zwracany zmienna;
    //instrukcje
    return zmienna;
}
```

```
double poleProstokata(double a, double b)
{
    double pole;
    pole = a*b;
    return pole;
    // krócej: return a*b;
}
```

Funkcje - deklaracja, definicja, wywołanie

```
1  #include <iostream>
2  using namespace std;
3
4  double poleProstokata(double a, double b); // deklaracja
5  // również tak: double poleProstokata(double, double);
6
7  int main()
8  {
9      double dl{}, szer{};
10     cout << "Podaj wymiary prostokata: " << endl;
11     cin >> dl >> szer;
12     cout << "Pole wynosi: ";
13     cout << poleProstokata(dl, szer) << endl;
14 }
15 // definicja funkcji
16 double poleProstokata(double a, double b)
17 {
18     return a * b;
19 }
```

Funkcje

- Nawiasy przy nazwie funkcji są konieczne zarówno przy deklaracji, definicji jak również wywołaniu funkcji.
- W przypadku, gdy funkcja nie posiada parametrów, w miejscu listy parametrów formalnych funkcji nic nie piszemy lub piszemy słowo kluczowe `void`.
- Na liście parametrów formalnych, dla każdego parametru określamy jego typ.
- Jeżeli funkcji zwraca wartość, wówczas w jej ciele powinna wystąpić instrukcja `return`, a po niej wyrażenie o typie zgodnym z typem rezultatu funkcji.
- W przypadku, gdy funkcja nie zwraca żadnego rezultatu, wówczas jest typu `void`.

Zmienne lokalne

Zmienne lokalne (automatyczne) przestają istnieć w momencie, gdy kończymy blok, w którym zostały zadeklarowane (np. zmienna jest tworzona wewnątrz funkcji). Zmienne lokalne komputer przechowuje na stosie. Stos ma ustalony i ograniczony rozmiar. Przed odczytaniem wartości ze zmiennej lokalnej należy najpierw wpisać do niej wartość. Jeśli zmienna lokalna nie została zainicjalizowana, to przechowuje przypadkową wartość.

Zmienne lokalne

```
1  #include<iostream>
2  using namespace std;
3
4  long long silnia(unsigned int);
5
6  int main() {
7     int i = 0; // zmienna lokalna funkcji main
8     for (; i < 20; i++)
9        cout << i << "! = " << silnia(i) << endl;
10    system("PAUSE");
11 }
12
13 long long silnia(unsigned int n)
14 {
15    n++; // n - zmienna lokalna funkcji silnia
16    // i = 10; zmienna i nie jest znana w funkcji silnia
17    long long wynik = 1; // zmienna lokalna
18    for (int i = 2; i < n; i++)
19       wynik *= i;
20    return wynik;
21 }
```

Zmienne globalne

Zmienne globalne są przechowywane w statycznym obszarze pamięci. Zmienna globalna, jeśli jej nie inicjalizowaliśmy, ma wartość 0 (stosowną do typu obiektu). Zmienne globalne są dostępne w całym programie i przez cały czas jego działania.

Zmienne globalne

```
1  #include<iostream>
2  using namespace std;
3
4  void drukuj(int);
5  int globalna;
6
7  int main() {
8      cout << globalna << endl;
9      globalna = 8;
10     cout << globalna << endl;
11     drukuj(4);
12 }
13
14 void drukuj(int wartosc)
15 {
16     globalna += wartosc;
17     cout << globalna << endl;
18 }
```

Zmienne globalne/lokalne - przesłanianie nazw

Zmienna lokalna przesłania zmienną globalną.

```
1  #include<iostream>
2  using namespace std;
3
4  int zmienna = 8;
5
6  int main() {
7      double zmienna = 2;
8      cout << zmienna << endl;
9      zmienna = 15;
10     cout << zmienna << endl;
11     cout << ::zmienna << endl;
12     system("pause");
13 }
```

Zmienne statyczne

Zmienne statyczne, podobnie jak zmienne globalne, są przechowywane w obszarze pamięci statycznej. Domyślnie są inicjalizowane zerem (chyba, że zażądamy inicjalizacji inną wartością). Jeżeli zmienna statyczna została utworzona wewnątrz bloku (np. wewnątrz funkcji), to po opuszczeniu bloku i ponownym wejściu do niego, wartość zmiennej zostanie zachowana. Zmienne te są opatrzone słowem kluczowym `static`.

Zmienne statyczne

```
1  #include<iostream>
2  using namespace std;
3  double funkcjaCelu(int wartosc)
4  {
5      static int licznik_wywolan_funkcji;
6      licznik_wywolan_funkcji++;
7      double wynik = 0;
8      //funkcja przeprowadza obliczenia
9      wynik += wartosc*wartosc;
10     cout << "licznik = " << licznik_wywolan_funkcji << endl;
11     return wynik;
12 }
13
14 int main() {
15     double w{};
16     for (int i = 0; i < 9; i++)
17     {
18         w = funkcjaCelu(i);
19         cout << "w = " << w << endl;
20     }
21 }
```

Przesyłanie argumentów przez wartość

Przy przesyłaniu argumentu do funkcji przez wartość, argument aktualny wywołania funkcji kopiowany jest do argumentu formalnego funkcji.
Z wnętrza funkcji nie jesteśmy w stanie zmienić wartości argumentu aktualnego wywołania (wartość ta jest tylko kopiowana).

```
1  #include<iostream>
2  using namespace std;
3
4  void zwieksz(int licznik)
5  {
6      licznik++;
7      cout << "z wnetrza funkcji: " << licznik << endl;
8  }
9
10 int main() {
11     int a = 5;
12     zwieksz(a);
13     cout << a << endl;
14     system("PAUSE");
15 }
```

Przesyłanie argumentów przez referencję

Przesłanie funkcji argumentu przez referencję pozwala funkcji na modyfikowanie zmiennej znajdującej się poza tą funkcją. Argument aktualny i formalny odnoszą się do tego samego adresu pamięci. Przekazując funkcji argument poprzez referencję, przekazujemy adres zmiennej.

Przesyłanie argumentów przez referencję

```
1  #include<iostream>
2  using namespace std;
3
4  void zwieksz(int&);
5
6  int main() {
7      int ilosc = 0;
8      for (int i = 0; i < 20; i++)
9      {
10         zwieksz(ilosc);
11         cout << ilosc << endl;
12     }
13     system("PAUSE");
14 }
15 void zwieksz(int &licznik) // &licznik jest adresem do zmiennej w↩
       argumencie
16 {
17     licznik++;
18 }
```

Argumenty domniemane

Argumenty domniemane zawsze stoją na końcu listy.

```
1  // deklaracja funkcji
2  int suma(int x, int y=0, int z=2, int t=5);
3
4  // wywołania funkcji
5  suma(1); // x = 1, y = 0, z = 2, t = 5
6  suma(3, 4); // x = 3, y = 4, z = 2, t = 5
7  suma(3, 4, 7); // x = 3, y = 4, z = 7, t = 5
8  suma(3, 4, 7, 8); // x = 3, y = 4, z = 7, t = 8
9
10 // definicja funkcji
11 int suma(int x, int y, int z, int t)
12 {
13     cout << "x = " << x << " y = " << y << "z = " << z << "t = " ↩
           << t << endl;
14     return x + y + z + t;
15 }
```

Argumenty domniemane

Określenie wartości domyślnej w deklaracji i definicji tego samego argumentu spowoduje błąd.

```
1 int suma(int x, int y=1);
2 ...
3 int suma(int x, int y=1) // błąd
4 { return x + y; }
```

Przeładowanie nazw funkcji

W danym zakresie ważności może być więcej niż jedna funkcja o tej samej nazwie. Wywołanie odpowiedniej funkcji następuje po dopasowaniu argumentów wywołania (liczba oraz typ argumentów). Przy przeładowaniu nazw funkcji typ zwracany przez funkcję nie jest brany pod uwagę.

```
1  void wypisz(int n){ cout << "liczba: " << n << endl; }
2
3  void wypisz(char znak){ cout << "Znak: " << znak << endl; }
4
5  void wypisz(double x, int n){
6     cout << "Liczba1: " << x << endl;
7     cout << "Liczba2: " << n << endl; }
8
9  void wypisz(double x1, double x2){
10    cout << "Obie zmiennoprzecinkowe " << endl;
11    cout << "Liczba1: " << x1 << endl;
12    cout << "Liczba2: " << x2 << endl; }
13
14 // to nie jest przeładowanie
15 int wypisz(int calkowita){ return calkowita + 100; }
```

Przeładowanie nazw funkcji

Przeładowanie nazw funkcji a argumenty domniemane.

```
1 double oblicz(double a);
2 double oblicz(int k, double r = 0);
3 // double oblicz(int k); // nie może być
4 double oblicz(char znak);
5
6 oblicz(1)        // oblicz(int, double=0)
7 oblicz('j')      // oblicz(char)
8 oblicz(4, 5.0)     // oblicz(int, double)
9 oblicz(4.0)      // oblicz(double)
```

Przeładowanie nazw funkcji

Jak kompilator zamienia nazwy funkcji?

```
1 void wypisz(int n)              // wypisz_Fi
2 void wypisz(char znak)          // wypisz_Fc
3 void wypisz(double x, int n)    // wypisz_Fdi
4 void wypisz(double x1, double x2)   // wypisz_Fdd
```

Sposób oznaczenia zależy od typu kompilatora.

Program składający się z kilku plików

Dla lepszej „organizacji” kodu, większej czytelności, pewne funkcjonalności możemy grupować w moduły (lub też biblioteki) i podzielić program na kilka plików.
Program może się składać z kilku odzielnie kompilowanych części, łączonych przez linker (kosolidator) w jeden plik wykonywalny.
Przyjęło się, że moduł składa się z dwóch części:

- część publiczna – nagłówek (najczęściej o rozszerzeniu `.h`),
- część implementacyjna (plik źródłowy o rozszerzeniu `.cpp`).

Program składający się z kilku plików

W pliku nagłówkowym `naglowek.h` umieszczone zostały deklaracje zmiennych globalnych oraz funkcji z plików `plikA.cpp` oraz `plikB.cpp`.

```
1  // naglowek.h
2  // część publiczna
3
4  #pragma once
5
6  // deklaracje zmiennych zewnętrznych
7  // ich definicje znajdują się w osobnych plikach
8  extern double globalnaA;
9  extern int globalnaB;
10
11 double funkcjaA(double, double);
12 int funkcjaB(int, int);
```

Dyrektywa preprocesora `#pragma once` zabezpiecza plik nagłówkowy przed wielokrotnym włączeniem jego treści w tym samym zakresie. Taki sam efekt można uzyskać dzięki odpowiednio użytym instrukcjom preprocesora `#ifndef`, `#define`, `#endif`.

Funkcje w programie składającym się z kilku plików

```
1  // plikA.cpp
2  // część implementacyjna
3
4  //#include "naglowek.h"
5  // definicja zmiennej zewnętrznej
6  double globalnaA = 10.5;
7
8  double funkcjaA(double a, double b)
9  {
10     return a / b;
11 }
```

Funkcje w programie składającym się z kilku plików

Jeśli w pliku `plikB.cpp`, chcemy mieć dostęp do funkcji, zmiennych z `plikA.cpp` musimy wówczas dołączyć odpowiedni plik nagłówkowy.

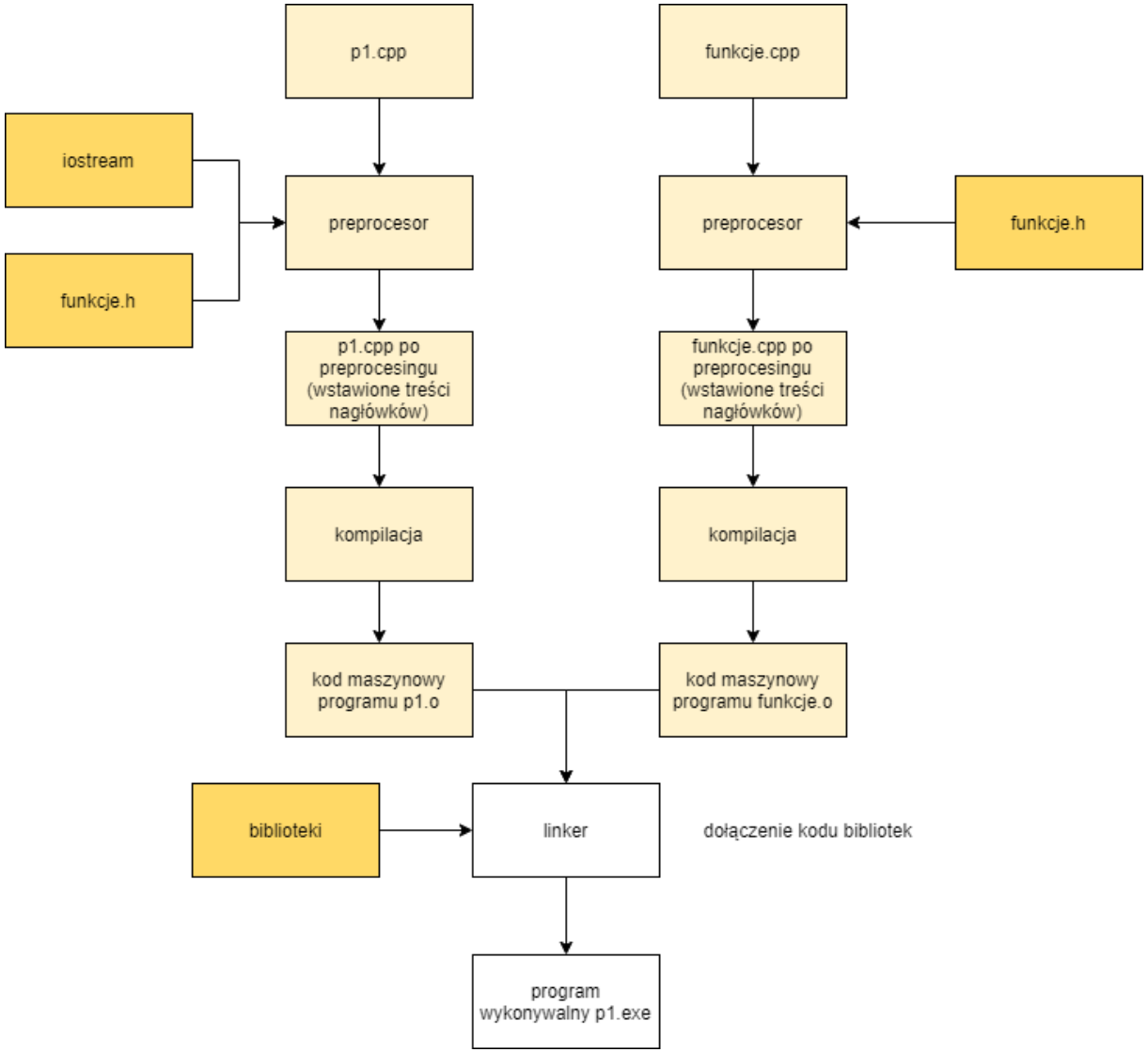
```
1  // plikB.cpp
2  // część implementacyjna
3
4  #include "naglowek.h"
5  // definicja zmiennej zewnętrznej
6  int globalnaB = 200;
7
8  int funkcjaB(int a, int b)
9  {
10     return (a * b) - globalnaA;
11 }
```

Funkcje w programie składającym się z kilku plików

Funkcja `main`.

```
1  #include<iostream>
2  #include "naglowek.h"
3  using namespace std;
4
5  int main()
6  {
7      cout << "globalnaA = " << globalnaA << endl;
8      cout << "globalnaB = " << globalnaB << endl;
9      cout << funkcjaA(10, 3) << endl;
10     cout << funkcjaB(5, 4) << endl;
11
12     system("pause");
13 }
```

Kompilacja rozłączna



Podsumowanie cz. I

- Deklaracja funkcji składa się z nazwy, zwracanego typu (w przypadku, gdy funkcja nie zwraca wartości piszemy `void`) oraz listy argumentów (opcjonalnie).
- Argumenty do funkcji możemy przekazywać poprzez wartość lub referencję.
- Rozróżniamy zmienne lokalne (automatyczne), globalne, statyczne.
- Zmienne lokalne umieszczane są na stosie. Nie są one domyślnie inicjalizowane.
- Zmienne statyczne oraz globalne nie są umieszczane na stosie. Domyślnie są inicjalizowane zerami.

Podsumowanie cz. II

- Argumenty domniemane funkcji stoją na końcu listy argumentów.
- Mechanizm przeładowania nazw funkcji pozwala na nazywanie kilku funkcji tą samą nazwą (wówczas funkcje różnią się typem lub liczbą argumentów).
- Program może dzielić na osobno kompilowane „części".
- Część publiczna znajduje się z reguły w nagłówku, zaś część implementacyjna w pliku źródłowym.
- Za dołączanie nagłówków odpowiada dyrektywa preprocesora `#include`.

Zadania

Zadanie 1. Napisać dwie funkcje (o przeładowanych nazwach) które rysują kwadrat o zadanej długości boku, zadanym znakiem:

```
void rysujKwadrat(int n = 5) // tylko gwiazdkami
void rysujKwadrat(char znak, int n = 5) // dowolny znak
```

Dziękuję za uwagę