Podstawy języka C++

Maciej Trzebiński

Instytut Fizyki Jądrowej
Polskiej Akademii Nauk

Praktyki studenckie na LHC

IV edycja, 2016 r.

IFJ PAN

Przygotowanie środowiska pracy

Niniejsza prezentacja stanowi minimalny wstęp do praktyk. Dyskutowane komendy będą często używane w praktyce. **Przed praktykami należy dokładnie zapoznać się z treścią kursu i samodzielnie wykonać zadania!** Proszę nie przesyłać nam odpowiedzi.

Ćwiczenia można wykonywać używając dowolnego kompilatora C/C++. Wiele darmowych programów jest dostępnych w sieci. Przykłady instalacji oraz uruchamiania można znaleźć w Internecie.

Ważne!

W przypadku jakichkolwiek problemów z instalacją należy szukać pomocy w Internecie lub u kolegów. To samo dotyczy problemów z kompilacją, działaniem programu, *itp.* W czasie trwania praktyk będziemy używali narzędzi zainstalowanych i przetestowanych w chmurze CC1. **Nie będzie trzeba posiadać kompilatora na własnym komputerze.**

Proszę nie zwracać się o pomoc do koordynatorów praktyk!

W Internecie istnieje bardzo dużo kursów C++, którymi można się wspomóc. Dla przykładu:

- http://cpp0x.pl/kursy/Kurs-C++/1
- http://miroslawzelent.pl/kurs-c++/
- http://main.edu.pl/en/user.phtml?op=show&page=cpp&c=70000

Tworzenie minimalnego, działającego programu

```
1  int main() {
2    return 0;
3  }
```

Program zawiera funkcję główną `main()`. W tym przypadku funkcja ta zwraca (`return`) liczbę całkowitą (`int`); tutaj jest to 0. Wszystkie polecenia wykonywane w ramach tej funkcji muszą się znaleźć pomiędzy nawiasami klamrowymi.

Skompilować i uruchomić program. **Jaki jest wynik?**

Dodatek: kompilacja i uruchomienie programu z terminala (Linuks)

Skompilować program (trzeba być w zawierającym go katalogu):
$ g++ -o program1 program1.cpp

Uruchomić program:
$ ./program1

Witaj świecie

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6    cout << "Witaj swiecie!" << endl;
7    return 0;
8  }
```

Program przekierowuje (<<) do konsoli (`cout`, *ang.* console output) napis *Witaj swiecie!* oraz znak końca linii (`endl`). Polecenia te nie są podstawowe w języku C++. Ich definicja znajduje się w osobnej bibliotece (`iostream`), którą należy dołączyć do programu (`#include`). Ponadto, polecenia znajdują się w przestrzeni nazw `std`. Kompilator jest o tym informowany przez dyrektywę `using namespace`. Każdy ciąg poleceń kończymy średnikiem.

Skompilować i uruchomić program. **Jaki jest wynik?**

Definiowanie zmiennych

```
1  int main() {
2    int a; //deklaracja zmiennej typu calkowitego (integer)
3  //zmienne mozna deklarowac po kilka, oddzielajac je przecinkami
4  //mozne tez przypisac im wartosc poczatkowa
5    int b = 3, c = 10;
6    float A = 10.25; //liczba zmiennoprzecinkowa (pojedyncza precyzja)
7    double B; //liczba zmiennoprzecinkowa (podwojna precyzja)
8    B = 0.015; //przypisanie zmiennej B wartosci 0.015
9    B = 1.25e5; //a teraz zmienna B ma wartosc 125000
10   char znak = 'A'; //pojedynczy znak
11   string ciag = "Ciag znakow bla, bla, bla ..."; //ciag znakow;
         biblioteka <string>; namespace std
12   bool p = true, f = false; //zmienna logiczna prawda/falsz
13
14   a = b; //przypisanie liczbie a wartosci b
15   a = a + 1; //zwiekszenie wartosci a o 1
16   a += 1; //jw.
17   a++; //jw.
18
19   a = b + c; //przypisanie liczbie a wartosci sumy b + c
20   a = b - c; //odejmowanie
21   a = b * c; //mnozenie
22   a = b / c; //dzielenie
23
24   return 0;
25 }
```

1. Jaki jest wynik działań 1/2 (obie liczby typu int) oraz 1./2. (obie liczby typu double)? Czy otrzymane wartości są takie same?

2. Zadeklarować zmienne całkowite i1 = 2, i2 = 3 oraz zmiennoprzecinkowe f1 = 0.5, f2 = 2.5. Stworzyć i wyświetlić na ekran zmienne a = i1 + i2, b = f1 + f2, c = i1 + f2, d = i1 / i2, e = i1 / f1, f = i1 * f1, g = f1 * i1.

3. Wyświetlić resztę z dzielenia 12331 przez 334.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    for (int a=0; a<100; a++)
6    {
7      cout << "Wyswietlam liczbe calkowita\t" << a << endl;
8    }
9
10   for (double b=-3.5; b<3.5; b+=0.5)
11   {
12     cout << "Wyswietlam liczbe zmiennoprzecinkowa\t" << b << endl;
13   }
14   return 0;
15 }
```

Pętle służą do wielokrotnego wykonywania poleceń. W powyższym kodzie wykorzystana jest pętla `for`. Wykonywana jest do czasu, gdy wartość zadeklarowanej zmiennej o ustalonej wartości początkowej (`int a=0`) nie przekroczy danej wartości (`a<100`). Po każdej iteracji zmienna `a` ma zostać zwiększona o 1.

Drugi przykład jest analogiczny, liczba typu `double` jest zwiększana o 0.5.

Warunki if, else, else if

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    for (int a=0; a<100; a++)
6    {
7      if (a<10) cout << a << endl; //wyswietl a jezeli jest ono
           mniejsze niz 10
8      else if (a<50) //wykonaj, jezeli powyzszy warunek nie jest
           spelniony i a jest mniejsze niz 50
9      {
10       //warunkiem moze byc objetych kilka linijek
11       cout << "ponad" << endl;
12       cout << "50" << endl;
13     }
14     else cout << "Ponad\t" << a << endl; //wykonaj gdy wszystkie
           poprzednie warunki nie sa spelnione
15   }
16   return 0;
17 }
```

Porównanie wartości:
$a > b, a < b$ // a większe niż b, a mniejsze niż b
$a >= b, a <= b$ // a większe lub równe b, a mniejsze lub równe b
$a == b$ //a równe b
$a! = b$ //a nie równe b

Operacje logiczne:
$A\&\&B$ // koniunkcja logiczna (A **i** B)
$A||B$ // alternatywa logiczna (A **lub** B)
$!A$ // negacja logiczna (nieprawda że A)

Instrukcje continue i break

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  int main() {
 5    for (int a=0; a<100; a++)
 6    {
 7      if (a<10) continue;
 8      if (a==50) break;
 9      cout << "Wyswietlam\t" << a << endl;
10    }
11    cout << "Koniec!" << endl;
12    return 0;
13  }
```

Powyższy program wyświetli liczby od 10 do 49. W pętli najpierw sprawdzany jest warunek $a < 10$ jeżeli jest spełniony to program wraca do początku pętli, zwiększając zmienną a (w tym przykładzie o 1). Instrukcje po linijce z *continue* nie są wykonywane. Jeżeli wartość a będzie wynosić 50, to pętla zostanie zakończona (`break`).

Funkcje

```
1  #include <iostream>
2  using namespace std;
3
4  void wyswietl(string napis){
5    cout << napis << endl;
6  }
7
8  double x_pow(double x, int y){
9    double wynik = x;
10   for (int a=1; a<y; a++) wynik *= x;
11   return wynik;
12 }
13
14 int main() {
15   wyswietl("Poczatek programu");
16   double a;
17   a = x_pow(10., 4);
18   cout << wyswietl("3.5^4 wynosi: ") << x_pow(3.5, 4) << endl;
19   return 0;
20 }
```

W programie znajdują się trzy funkcje. Każda z nich jest innego typu – program oczekuje, że funkcja będzie zwracać (`return`) określoną wartość: `main` jest typu całkowitego (`int`), `x_pow` zwraca liczbę z podwójną precyzją (`double`), `wyswietl` jest typu `void` – nie zwraca wartości. Funkcja `wyswietl` przyjmuje jeden parametr (typu `string`): `napis`, a funkcja `x_pow` dwa parametry (pierwszy typu `double`, drugi `int`).

Zadania

1. Napisać program, który sprawdzi czy 113 jest liczbą pierwszą.

2. Napisać funkcję, która sprawdzi czy liczba podana w argumencie jest pierwsza.

3. Przyśpieszyć funkcję z pkt. 2, aby nie szukała dalej po pierwszym stwierdzeniu podzielności.

4. Wypisać 1000 pierwszych liczb pierwszych.

5. Zabezpieczyć funkcję z pkt. 4 tak, aby działała tylko w przypadku podania dodatniej liczby całkowitej. W innych przypadkach ma informację o błędzie.

Klasy

```
1  #inlcude <iostream>
2  using namespace std;
3
4  class Wektor{
5    public: //prawo dostepu: publiczne
6      void ustaw(double, double); //deklaracja funkcji
7      double dlugosc(); //deklaracja funkcji
8    private: //prawo dostepu: prywatne (widoczne z poziomu klasy)
9      double x, y; //deklaracja zmiennych
10 }; //klase konczymy srednikiem
11
12 //definicja funkcji "ustaw" z klasy "Wektor"
13 void Wektor::ustaw(double a, double b){
14   x = a; y = b;
15 }
16
17 //definicja funkcji "dlugosc" z klasy "Wektor"
18 void Wektor::dlugosc(){
19   return sqrt(x*x + y*y);
20 }
21
22 int main() {
23   Wektor v;//utworzenie obiektu klasy "Wektor"
24   v.ustaw(3., 4.);
25   cout << v.dlugosc() << endl;
26   return 0;
27 }
```

W programie znajduje się klasa `Wektor`. Zawiera deklaracje dwóch zmiennych (`x` oraz `y`) oraz dwóch funkcji (`ustaw(double, double)`, `dlugosc()`). Funkcje są zdefiniowane w dalszej części programu. Kompilator wie, że należą one do klasy przez określenie `Wektor::` w definicji funkcji.

Plik **program.h** zawiera informacje o: bibliotekach, przestrzeniach nazw, klasach, funkcjach.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Wektor{
6    public:
7      void ustaw(double, double);
8      double dlugosc();
9    private:
10     double x, y;
11 };
12
13 void Wektor::ustaw(double a, double b){x = a; y = b;}
14
15 void Wektor::dlugosc(){return sqrt(x*x + y*y);}
```

Plik **program.cpp** zawiera program główny.

```
1  #include "program.h"
2
3  int main() {
4    Wektor v;
5    v.ustaw(3., 4.);
6    cout << v.dlugosc() << endl;
7    return 0;
8  }
```

Zadania

1. Dodać funkcję, która skaluje wektor o zadaną liczbę. Definicje mają się znajdować w pliku **.h**.

2. Zmienić współrzędne na `public` zdefiniować operator dodawania dwóch wektorów.