Podstawy języka C++

Maciej Trzebiński

Instytut Fizyki Jądrowej
Polskiej Akademii Nauk

Praktyki studenckie na LHC

IFJ PAN

6 lipca 2015

Uruchomienie maszyny w CC1

- Otworzyć stronę: **www.cloud.ifj.edu.pl**
- *Zaloguj się*
- Maszyny wirtualne → Nowa maszyna
- Obraz: praktyki_2015
- Sprzęt: 2 cpu
- Zasoby:
  - Adres IP: none
  - Dołącz dysk: **zaznaczyć opcję podpięcia swojego dysku**
  - Dołącz obraz ISO: none
  - VNC: **zaznaczyć**
- Nadać maszynie nazwę i utworzyć.
- Poczekać, aż status maszyny zmieni się z *init* na *running*
- Kliknąć na utworzoną maszynę → rozwinie się okno z jej statusem
- Kliknąć *Konsola graficzna (VNC)*
- Zalogować się na konto *student*

- Otworzyć LXTerminal (skrót jest dostępny np. na pasku zadań)
- Zamontować dysk:
  - Sprawdzić, czy dysk /dev/vda1 jest widoczny: **sudo fdisk -l**
  - Zamontować dysk do katalogu /media:
    sudo mount -t ext4 /dev/vda1 /media/
  - Zmienić właściciela katalogu: **sudo chown student /media/**
  - Sprawdzić, czy wszystko działa: **cd /media**
    echo "Witaj swiecie!" > witaj.txt
    cat witaj.txt

Proszę pamiętać, że **wszystko należy zapisywać w katalogu /media**.

Dane zapisane w innych miejscach (np. w katalogu /home) zostaną skasowane wraz ze zniszczeniem maszyny wirtualnej!

Tworzenie minimalnego, działającego programu

Otworzyć dowolny edytor tekstu:
@> leafpad program1.cpp &

```
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 program:
@> g++ -o program1 program1.cpp

Uruchomić program:
@> ./program1

@> leafpad program1.cpp &

```
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.

@> g++ -o program1 program1.cpp && .program1

Środowisko programistyczne i komentarze

@> geany program1.cpp &

Bo kolorowanie składni oraz inne ułatwienia są przydatne :)

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6
7  /* to jest komentarz
8  w
9  kilku
10 linijkach */
11
12 // A to komentarz w jednej linii
13    cout << "Witaj swiecie!" << endl;
14    return 0;
15 }
```

Skompilować program:
kliknąć symbol cegiełki

Uruchomić program:
kliknąć symbol koła zębatego

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 }
```

Zadania

1. Jaki jest wynik działań 1/2 oraz 1./2.?

2. Zadeklarować zmienne całkowite i1 = 2, i2 = 3 oraz zmiennoprzecinkowe f1 = 0.5, f2 = 2.5. Stworzyć i wyświetlić 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 (przydatny link: www.google.pl).

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

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

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   int main() {
2     for (int a=0; a<100; a++)
3     {
4       if (a<10) continue;
5       if (a==50) break;
6       cout << "Wyswietlam\t" << a << endl;
7     }
8     cout << "Koniec!" << endl;
9     return 0;
10  }
```

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

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. 3, aby nie szukała dalej po pierwszym stwierdzeniu podzielności.

4. Wypisać 1000 pierwszych liczb pierwszych.

5. Porównać czas działania programu z pkt. 5 w przypadku zastosowania przyśpieszenia (pkt. 4) i przeciwnym.

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

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

W programie znajdują 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.