Środowisko programistyczne GEANT4

Leszek Adamczyk

Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza

Wykłady w semestrze zimowym 2013/2014

Informacje wstępne

- Kurs oparty jest na:
  - dokumentacji Geant4
    `http://geant4.cern.ch/support/gettingstarted.shtml`
  - przykładach zawartych w dystrybucji GEANT4
    `$G4INSTALL/examples`
  - kursach dostępnych na stronach GEANT4
    `http://geant4.cern.ch/support/training.shtml`
- Strona kursu:
  `http://home.agh.edu.pl/~leszekad/dydaktyka/wfiis_geant4_2013`

Informacje wstępne

- wersja 4.9.4.p01 GEANT4 jest zainstalowana na:
  - komputerach w pracowniach komputerowych
  - na serwerze `fatcat`
- na zajęciach będziemy korzystać z wersji 4.9.6.p02 dostępnej na "dowolnej"maszynie przez CVMFS Cern Virtual Machine File system
- instrukcja instalacji na stronie przedmiotu
- zachęcamy do instalacji wersji4.9.6.p02 na prywatnych komputerach.
- zaliczenie kursu na podstawie projektów (aplikacja GEANT4 + analiza wyników)

Wymagania programistyczne

- Unix/Linux
  - podstawowe polecenia systemowe
  - edycja tekstów
  - korzystanie ze zmiennych środowiskowych
  - kompilacja i uruchamianie programów
- Znajomość podstawowych cech programowania obiektowego:
  abstrakcja, hermetyzacja, polimorfizm, dziedziczenie
- Podstawowa znajomość C++
  tworzenie aplikacji GEANT4 to pisanie kodu w C++
- Analiza i prezentacja wyników symulacji
  poznamy podstawowe możliwości pakietu ROOT

C++: Wbudowane typy danych

- `char` - typ znakowy
- `int` - typ całkowitoliczbowy
- `float` - typ zmiennoprzecinkowy
- `double` - typ zmiennoprzecinkowy podwójnej precyzji
- `bool` - typ boolowski o wartościach `true`, `false`
- specyfikatory `long`, `short`, `signed`, `unsigned`
- GEANT4
  `G4int`, `G4float`, `G4double`, `G4bool`, `G4long`

C++: Strumienie wejścia, wyjścia

```
#include <iostream>

using namespace std;

int main ()
{
 int liczba;
 cout << "Podaj liczbę parzystą: " ;
 cin >> liczba;

 if ( !(liczba%2) )
     cout << liczba << endl;
 else
     cerr << "Błąd: " << liczba
          << " nie jest liczbą parzystą" << endl;
 return 0;
}
```

GEANT4 : G4cout, G4cin, G4cerr, G4endl

C++: Przestrzeń nazw

```
#include <iostream>

int main ()
{
 int liczba;
 std::cout << "Podaj liczbę parzystą: " ;
 std::cin >> liczba;

 if ( !(liczba%2) )
     std::cout << liczba << std::endl;
 else
     std::cerr << "Błąd: " << liczba
               << " nie jest liczbą parzystą"
               << std::endl;
 return 0;
}
```

zakres::obiekt (szukaj obiektu poza lokalną przestrzenią)

C++: Rzutowanie typów

- rzutowanie niejawne (niezalecane)

```
Typ1   obiekt1;
Typ2   obiekt2 = wartość;
..............
obiekt1 = obiekt2;
```

- rzutowanie wymuszone (niezalecane)

```
obiekt1 = (Typ1) obiekt2;
```

- operator `static_cast` (zalecane)

```
obiekt1 = static_cast<Typ1> (obiekt2);
```

C++: Typ tekstowy String

```
#include <string>
............
string text1("Jeden ")
string text2;

cin >> text2;
text1 = text1 + " " + text2;
cout << text1 << endl;

text1.size();
text1.compare(text2)
text1.find(text2)
```

GEANT4 - G4string

C++: Przeciążanie funkcji

```
int    suma( int, int)
double suma( double, double)
double suma( double)
```

```
int  suma( int a, int b)
{ return a+b; }


double suma( double a, double b)
{ return a+b; }


double suma( double a)
{ return a+a; }
```

C++: Wskaźniki

```
int liczba = 5;
int* pLiczba;
pLiczba = &liczba;

cout << liczba   << endl // zwraca 5
cout << pLiczba << endl // zwraca adres w pamięci
cout <<*pLliczba << endl // zwraca 5
```

```
*pLiczba = 10;

cout << liczba  << endl // zwraca 10
```

```
int tablica[5] = {1,2,3,4,5};
int *pTablica = tablica // to  samo  pTablica=&tablica[0];

cout << tablica[1]    << endl // zwraca 2
cout << *(pTablica+1)  << endl // zwraca 2
```

```
f(tablica)  // do funkcji przekazany jest adres tablicy nigdy jej kopia
```

C++: Referencja

Referencja - inna nazwa obiektu

```
int liczba = 5;
int &rLiczba=liczba;

cout << liczba   << endl // zwraca 5
cout << rLiczba << endl // zwraca 5
```

```
rLiczba = 10;

cout << liczba  << endl // zwraca 10
```

Referencja nie może być ponownie przypisana

```
int liczba2 = 20;
rLiczba = liczba2

cout << liczba  << endl // zwraca 20
```

C++: Przekazywanie argumentów do funkcji

```
void fPrzezWartość ( Typ obiekt)
{
  obiekt = .... // tworzy i modyfikuje lokalną kopię
}
```

```
void fPrzezWskaźnik ( Typ * obiekt)
{
  *obiekt = ....
}
```

```
void fPrzezReferencje ( Typ & obiekt)
{
  obiekt = ....
}
```

C++: Argumenty lini poleceń i zmienne środowiskowe

```
program arg1 arg2
```

```
int main (int argc, char *argv[], char *env[])
{
......
}
```

`argc` - ilość argumentów + 1
`argv[0]` - nazwa programu
`argv[1]` - argument pierwszy
.....
`env[0]` - pierwsza zmienna srodowiskowa
.....

C++: Argumenty lini poleceń i zmienne środowiskowe

```
#include <iostream>
#include <string>
#include <cstdlib>

using namespace std;

 int main (int argc, char *argv[], char *env[])
{
  if ( argc >= 2 ) {
   double a = atof ( argv[1] );
   int    b = atoi ( argv[2] );
  }
  string n = "PWD=";
  int i =0;
  do {
    if ( n.compare(0, n.size(), env[i], n.size()) == 0)
    {
       cout << "Program uruchomiono z: ";
       cout << env[i] + n.size() << endl;
    }
    i++;
  } while ( env[i] != NULL ) ;
  return 0;
}
```

C++: Zasięg i czas ”życia” zmiennych

- Obiekty globalne i statyczne są tworzone w chwili uruchomienia programu i ”żyją” do jego końca;
- Pozostałe obiekty są tworzone w chwili napotkania ich definicji i niszczone gdy ich nazwa opuszcza zasięg.

```
int zmiennaGlobalna;

int main ()
{
  ..........
  {
  int zmiennaLokalna;
  }
}
void f( void)
{
  int zmiennaLokalna;

  static int licznik = 0;
  licznik++;
  cout << "Funkcja wywołana po raz: " << licznik << endl;
}
```

C++: Operatory new i delete

- `new` - przyporządkowuje dynamicznie pamięć;
- `delete` - zwalnia przydzieloną pamięć;
- wymagają stosowania wskaźników;
- pamięcią jawnie zarządza programista

```
double *p = new double;
.......................
delete p;
```

```
double *p = new double[20];
.......................
delete [] p;
```

```
cout << "Podaj rozmiar tablicy: ";
int rozmiar;
cin >> rozmiar
double *p = new double[rozmiar];

double r[rozmiar]; // błąd  ->  const int rozmiar = 5;
.......................
delete [] p;
```

Uwaga na wyciek pamięci !

C++: Wzorce(szablony) funkcji

Wzorzec umożliwia jednoczesną definicję funkcji dla wielu typów argumentów.

```
template <typename T>
void pisz (T *t, int n )
{
  for( int i=0; i<n; i++)
  {
     T m = t[i];
     cout << m << endl;
  }
}
```

```
int main()
{
 int    tabi[] = {-2, -1, 1, 2, 4};
 double tabd[] = {-2.5, -1.5, 1.5, 2.5,};

 pisz (tabi,5);
 pisz (tabd,4);
}
```

C++: Klasy

Klasy umożliwiają tworzenie własnych typów danych i funkcji operujących na nich. Składowe klasy:

- dane inaczej pola;
- funkcje inaczej metody;

Deklaracja klasy:

```
class Klasa
{
 double x;
 double y;
 double suma ( void );
};
```

C++: Klasy

Klase dzielimy na dwie główne części:

- publicznego zbioru pól i metod (interfejs publiczny)
- prywatnego zbioru pól i metod (implementacja)

```
class Klasa
{
 public:
    // interfejs publiczny
 private:
    // prywatna implementacja
};
```

- Użytkownik klasy programuje korzystając z interfejsu publicznego;
- Zmiana prywatnej implementacji przy zachowaniu interfejsu publicznego powoduje, że kod w którym go stosujemy nie wymaga zmian;
- GEANT4: Budowa aplikacji polega na modyfikacji implementacji przy zachowaniu struktury interfejsu publicznego, dzięki czemu jądro GEANT'a poprawnie obsługuje zmodyfikowane klasy.

C++: Tworzenie obiektu i dostęp do pól klasy

```
class Klasa
{
 public:
    double x;
    double y;
 private:
    double z
};
```

```
 Klasa k;
 Klasa *p = new Klasa();
.........................
 k.x  = 2.0;
 p->y = 3.0;

 k.z = 4.0; // błąd z jest polem prywatnym
```

C++: Podział kodu klasy na pliki

Klasa.h:

```
#include <iostream>

class Klasa
{
 public:
    double x;
    double y;
    void pisz( void);
};
```

Klasa.cpp:

```
#include "Klasa.h"

void Klasa::pisz(void)
{
 std:cout << "Pola klasy: " << a
          << "," << b << std:endl;
}
```

C++: Dziedziczenie

- klasę z której tworzymy(wyprowadzamy) nową klasę nazywamy klasą bazową;
- klasę wyprowadzoną z klasy bazowej nazywamy klasą pochodną
- GEANT4: wszystkie klasy bazowe maja nazwę: `G4KlasaBazowa`

```
class KlasaPochodna : public G4KlasaBazowa
{
    ..................................
}
```

- klasa pochodna może być klasą bazową innych klas;
- klasa pochodna nie ma dostępu do składowych prywatnych swojej klasy bazowej.

C++: Dziedziczenie: Przykład

```
class Baza
{
  public:
    Baza( void ) { cout << "Konstruktor klasy bazowej" << endl; }
   ~Baza( void ) { cout << "Destruktor klasy bazowej" << endl; }

    double x,y;
};
```

```
class Pochodna : public Baza
{
  public:
    Pochodna( void ) { cout << "Konstruktor klasy pochodnej" << endl; }
   ~Pochodna( void ) { cout << "Destruktor klasy pochodnej" << endl; }

    double z;
};
```

```
int main()
{
  Pochodna o;
  o.x=1.0;
  o.z=2.0;
  return 0;
}
```

```
Konstruktor klasy bazowej
Konstruktor klasy pochodnej
Destruktor klasy pochodnej
Destruktor klasy bazowej
```

C++: Dziedziczenie: modyfikacja interfejsu publicznego

```
class Baza
{
  public:
    void f( void ) { .... }
};

class Pochodna : public Baza
{
 public:
   void f( void ) { .... }
};
```

```
int main()
{
 Baza ob;
 Pochodna op;
 ob.f(); // f z Baza
 op.f(); // f z Pochodna
.........................
 Baza *pb = &ob;
 bp->f(); // f z Baza
 pb = &op;
 bp->f(); // f z Baza

 Pochodna *pp = &op;
 pp->f(); // f z Pochodna

}
```

C++: Dziedziczenie: funkcje wirtualne

```
class Baza
{
  public:
  virtual  void f( void ) {}
};

class Pochodna : public Baza
{
 public:
   void f( void ) { .... }
};
```

```
int main()
{
 Baza ob;
 Pochodna op;
 ob.f(); // f z Baza
 op.f(); // f z Pochodna
.........................
 Baza *pb = &ob;
 bp->f(); // f z Baza
 pb = &op;
 bp->f(); // f z Pochodna

 Pochodna *pp = &op;
 pp->f(); // f z Pochodna

}
```

C++: Klasa abstrakcyjna

- funkcja wirualna może być zadeklarowana jako czysto wirtualna (bez implementacji)

```
virtual void f (void) = 0;
```

- klasę która deklaruje przynajmniej jedna funkcję czysto wirtualną nazywamy klasą abstrakcyjną;
- w programie nie można utworzyć obiektów klasy abstarkcyjnej;
- implementacji funkcji czysto wirtualnych dostarczają dopiero klasy pochodne;
- klasa abstrakcyjna służy do zadania części interfejsu klas pochodnych;
- GEANT4: jądro GEANT'a korzysta z interfejsu zadanego przez klasy abstarkcyjne;
- GEANT4: wszystkie klasy abstarkcyjne maja nazwę: `G4VKlasaAbstrakcyjna`

C++: Kolekcje standardowe

- Kolekcjami nazywamy klasy przechowujące inne obiekty
- vector - kolekcja danych typu uporządkowanego

Zastosowanie wektorów:

- dostęp do elementów kolekcji poprzez ich indeks;
- iteracja po elementach w dowolnym kierunku;
- dodawanie i usuwanie elementów z końca kolekcji.

```
#include <vector>
```

Przykładowa deklaracja:

```
vector<TypDanych>  w(100);
```

Dostęp do elementów kolekcji:

```
w[5]    = 11;
z       = w.at(5);
```

C++: Metody klasy vector

```
w.size();              // ilość elementów w kolekcji
w.empty()              // true jeśli kolekcja pusta
w.clear();             // usuwa wszystkie elementy
w.push_back (10);      // dodaje do końca kolekcji 10
w.pop_back();          // usuwa ostatni element z kolekcji
w.front();             // pierwszy element
w.back();              // ostatni element
```

Iterator - wskaźnik, ułatwiający pracę z kolekcjami.

```
vector<TypDanych>::iterator  it = w.begin();
 while ( it != w.end() ) {
   *it = .... // zmiana wskazywanej wartości
   it++;      // przejście do następnego elementu
 }
```

C++: Kolekcja typu map

Mapa składa się z par `(klucz, wartość)`. Mapa umożliwia dostęp do wartości pary poprzez wartość klucza.

```
#include <map>
```

Przykładowa deklaracja:

```
map<TypKlucza, TypWartości>  w;
```

```
map<String, int>  indeks;

indeks["X"] = 12345;
indeks["Y"] = 54321;

cout << indeks["X"] << endl;
```

C++: Iterator kolekcji map

Iterator mapy wskazuje na pary. Para jest obiektem posiadającym pola `first` (klucz) oraz `second` (wartość)

```
map<String, int>  indeks;

indeks["X"] = 12345;
indeks["Y"] = 54321;


map<string, int >::iterator  it = indeks.begin();
 while ( it != indeks.end() ) {
   cout << (*it).first << " - " << (*it).second << endl;
   it++;      // przejście do następnego elementu mapy
 }

it = indeks.find("Y");
indeks.erase("X");
```