

Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 2

Aleksandra Fijałkowska

19 października 2018

```
class A {  
public:  
    int x() { return 1; }  
    int y() { return 2; }  
    virtual int z() { return 3; }  
};
```

```
class B : public A {  
public:  
    int y() { return 22; }  
    int z(){ return 33; }  
};
```

```
class C {  
public:  
    int cax(A* a) { return a->x(); }  
    int cay(A* a) { return a->y(); }  
    int caz(A* a) { return a->z(); }  
    int cby(B* b) { return b->y(); }  
    int cbz(B* b) { return b->z(); }  
};
```

```
A* a = new A();  
B* b = new B();  
C* c = new C();  
cout << c->cax(a) << endl;  
cout << c->cax(b) << endl;
```

```
class A {  
public:  
    int x() { return 1; }  
    int y() { return 2; }  
    virtual int z() { return 3; }  
};
```

```
class B : public A {  
public:  
    int y() { return 22; }  
    int z(){ return 33; }  
};
```

```
class C {  
public:  
    int cax(A* a) { return a->x(); }  
    int cay(A* a) { return a->y(); }  
    int caz(A* a) { return a->z(); }  
    int cby(B* b) { return b->y(); }  
    int cbz(B* b) { return b->z(); }  
};
```

```
A* a = new A();  
B* b = new B();  
C* c = new C();  
cout << c->cax(a) << endl; 1  
cout << c->cax(b) << endl; 1
```

```
class A {  
public:  
    int x() { return 1; }  
    int y() { return 2; }  
  
    virtual int z() { return 3; }  
};
```

```
class B : public A {  
public:  
    int y() { return 22; }  
    int z(){ return 33; }  
};
```

```
class C {  
public:  
    int cax(A* a) { return a->x(); }  
    int cay(A* a) { return a->y(); }  
    int caz(A* a) { return a->z(); }  
    int cby(B* b) { return b->y(); }  
    int cbz(B* b) { return b->z(); }  
};
```

```
B* b = new B();  
C* c = new C();  
cout << c->cay(b) << endl;  
cout << c->cby(b) << endl;
```

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }

    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};

B* b = new B();
C* c = new C();
cout << c->cay(b) << endl;    2
cout << c->cby(b) << endl;    22
```

```
class A {
public:
    int x() { return 1; }
    int y() { return 2; }
    virtual int z() { return 3; }
};

class B : public A {
public:
    int y() { return 22; }
    int z(){ return 33; }
};

class C {
public:
    int cax(A* a) { return a->x(); }
    int cay(A* a) { return a->y(); }
    int caz(A* a) { return a->z(); }
    int cby(B* b) { return b->y(); }
    int cbz(B* b) { return b->z(); }
};

A* a = new A();
B* b = new B();
C* c = new C();
cout << c->caz(a) << endl;
cout << c->caz(b) << endl;
cout << c->cbz(b) << endl;
```

```
class A {  
public:  
    int x() { return 1; }  
    int y() { return 2; }  
    virtual int z() { return 3; }  
};
```

```
class B : public A {  
public:  
    int y() { return 22; }  
    int z(){ return 33; }  
};
```

```
class C {  
public:  
    int cax(A* a) { return a->x(); }  
    int cay(A* a) { return a->y(); }  
    int caz(A* a) { return a->z(); }  
    int cby(B* b) { return b->y(); }  
    int cbz(B* b) { return b->z(); }  
};
```

```
A* a = new A();  
B* b = new B();  
C* c = new C();  
cout << c->caz(a) << endl;    3  
cout << c->caz(b) << endl;    33  
cout << c->cbz(b) << endl;    33
```

Omówienie testu

Zadanie 2

```
class KlasaCpp {  
public:  
    virtual int foo() = 0;  
};
```

Co jest wynikiem wywołania:

```
KlasaCpp* x = new KlasaCpp();  
std::cout << x->foo() << std::endl;
```

Zadanie 3

```
class Counter {  
private:  
    static int _x;  
public:  
    Counter(int x) { _x = x;}  
    void inc() { _x++; }  
    int getState() { return _x; }  
};  
int Counter::_x = 0;
```

Co jest wynikiem wywołania:

```
Counter* c1 = new Counter(10);  
Counter* c2 = new Counter(100);  
c1->inc();  
c2->inc();  
std::cout << c1->getState() << std::endl;  
std::cout << c2->getState() << std::endl;
```


Omówienie testu

Zadanie 2

```
class KlasaCpp {  
public:  
    virtual int foo() = 0;  
};
```

Co jest wynikiem wywołania:

```
KlasaCpp* x = new KlasaCpp();  
std::cout << x->foo() << std::endl;
```

Zadanie 3

```
class Counter {  
private:  
    static int _x;  
public:  
    Counter(int x) { _x = x;}  
    void inc() { _x++; }  
    int getState() { return _x; }  
};  
int Counter::_x = 0;
```

Co jest wynikiem wywołania:

```
Counter* c1 = new Counter(10);    // _x = 10  
Counter* c2 = new Counter(100);   // _x = 100  
c1->inc();                         // _x = 101  
c2->inc();                         // _x = 102  
std::cout << c1->getState() << std::endl; //102  
std::cout << c2->getState() << std::endl; //102
```

Zadanie 4

Napisz funkcję main która wypisze na standardowe wyjście wartości wszystkich parametrów przekazanych podczas uruchomienia programu app:

```
./app paramValue1 paramValue2
```

```
int main(int argc, char *argv[])
{
    for(int i = 1; i!= argc; ++i)
        cout << argv[i] << endl;
    return 0;
}
```

Co zwróci argv[0]?

Zadanie 5 (wersja odrobinę zmniejszona)

Napisz konstruktor kopiujący oraz operator przypisania klasy:

```
#include "G4VHit.hh"
#include "G4LogicalVolume.hh"
class PMTHit : public G4VHit {
public:
    PMTHit(double energyDep);
    virtual ~PMTHit();
    PMTHit(const PMTHit &right);
    const PMTHit& operator=(const PMTHit &right);
private:
    int hitsNr;
    std::vector<double> energyDep;
    bool drawIt;
    G4LogicalVolume* logVol;
};
```

Konstruktor kopiujący:

```
PMTHit::PMTHit(const PMTHit &right) : G4VHit()
{
    hitsNr = right.hitsNr;
    drawIt = right.drawIt;
    energyDep = right.energyDep;
    logVol = G4LogicalVolume(right.logVol); //deep copy
}
```

Zadanie 5 (wersja odrobinę zmniejszona)

Napisz konstruktor kopiujący oraz operator przypisania klasy:

```
#include "G4VHit.hh"
#include "G4LogicalVolume.hh"
class PMTHit : public G4VHit {
public:
    PMTHit(double energyDep);
    virtual ~PMTHit();
    PMTHit(const PMTHit &right);
    const PMTHit& operator=(const PMTHit &right);
private:
    int hitsNr;
    std::vector<double> energyDep;
    bool drawIt;
    G4LogicalVolume* logVol;
};
```

Operator przypisania:

```
const PMTHit& PMTHit::operator=(const PMTHit &right)
{
    hitsNr = right.hitsNr;
    drawIt = right.drawIt;
    energyDep = right.energyDep;
    logVol = right.logVol;
    return *this;
}
```

Omówienie testu

Praca domowa

Pierwszy projekt

Zadanie 6

Napisz szablon funkcji (np. `applyForEach`), która przyjmie jako argumenty funkcję f o jednym argumencie i wektor elementów tego samego typu T , i zwróci wektor elementów $f(x_i)$ Wskazówka: Deklaracja szablonu funkcji może wyglądać tak:

```
template<typename T>
std::vector<T> applyForEach(T (*fun)(T), std::vector<T> arrayIn)
```

Wektor

```
template<typename T>
std::vector<T> applyForEach(T (*fun)(T), std::vector<T> arrayIn)
{
    vector<T> arrayOut;
    for (int i = 0; i < arrayIn.size(); ++i)
        arrayOut.push_back(fun( arrayIn.at(i) ) );
    return arrayOut;
}
```

Tablica

```
template<typename T>
T* applyForEach(T (*fun)(T), T* arrayIn, int size)
{
    T *arrayOut = new T[size];
    for(int i = 0; i != size; ++i)
        arrayOut[i] = fun(arrayIn[i]);
    return arrayOut;
}
```

Zadanie 7

Popraw błędy w załączonym kodzie.

```
const int TAB[] = { 2, 3, 8, 5 };
const int TAB_SIZE = sizeof(TAB)/sizeof(TAB[0]);
std::vector<int> v(TAB, TAB + TAB_SIZE);
const std::vector<int>::iterator it = v.begin();
++it;
*it = 5;
std::vector<int>::const_iterator it2 = v.begin();
++it2;
*it2 = 8;
```

Rodzaje iteratorów w bibliotece standardowej:

```
T::const_iterator it; //stała wartość wskazywana przez iterator
const T::iterator it; //stały iterator
const T::const_iterator it; //stała wartość wskazywana przez stały iterator
```

```
const int TAB[] = { 2, 3, 8, 5 };
const int TAB_SIZE = sizeof(TAB)/sizeof(TAB[0]);
std::vector<int> v(TAB, TAB + TAB_SIZE);
const std::vector<int>::iterator it = v.begin();
++it; //błąd, iterator jest stały
*it = 5;
std::vector<int>::const_iterator it2 = v.begin();
++it2;
*it2 = 8; //stała wartość wskazywana, nie można jej zamienić
```

Pytanie testowe

1. Tablice są zbiorami elementów:

- a) dowolnego typu
- b) tego samego typu

2. Dostęp do tablicy odbywa się przez:

- a) iterator
- b) indeks
- c) obie odpowiedzi prawidłowe

Komentarz: W zasadzie sprawny programista byłby w stanie napisać swój iterator do tablic, pytanie mogłoby być bardziej precyzyjne i określać, że chodzi o iteratory wbudowane w bibliotekę standardową

3. Czym jest tablica dynamiczna?

Jest to tablica, której rozmiar jest określany w trakcie działania programu, a nie na etapie kompilacji.

4. Dostęp do pola struktury odbywa się przez wykorzystanie operatora:

- a) &
- b) .
- c) *

Komentarz: Do pól klasy lub struktury uzyskujemy dostęp przez operator ".", jeśli instancja klasy lub struktury jest stworzona przez wartość oraz "→", jeśli dysponujemy wskaźnikiem do obiektu.

Omówienie testu

Praca domowa

Pierwszy projekt

Pytanie testowe cd.

5. Jak poprawnie zwolnić pamięć tak stworzonego obiektu: `Foo* f = new Foo[N];`
`delete [] f;`
Komentarz: Wywołanie operatora **delete** zwalnia pamięć (wywołuje destruktory) po jednym obiekcie stworzonym jako wskaźnik, czyli z wykorzystaniem operatora **new**. W przypadku zwalniania pamięci po tablicy dynamicznej należy wywołać "tablicową" wersję tego operatora, czyli **delete []**.
6. Czy można stworzyć instancję klasy, której konstruktor jest prywatny?
Można, taką klasą są min. **singletony**, czyli klasy, które w zamierzeniu nie mogą mieć więcej niż jednej instancji w programie. Aby zapobiec możliwości tworzenia wielu instancji konstruktor czyni się prywatnym. Dostęp do klasy uzyskuje się poprzez publiczną statyczną metodę (np. **getInstance()**), która zwraca wskaźnik do obiektu. Ten sam wskaźnik jest zwracany przy każdym wywołaniu metody **getInstance()**.
Geant4 w swoich bibliotekach wykorzystuje koncept singletonów i my też będziemy takie klasy tworzyć.

Orkiestra składająca się ze 100 muzyków gra koncert. Każdy muzyk gra na innym instrumencie. Po zagranie koncertu schodzą ze sceny i chowają swoje instrumenty do pudełek. Na każdym pudełku jest etykieta jaki instrument powinien znaleźć się w pudełku. Pudełka są identyczne, na tyle duże, że każdy z instrumentów może się tam zmieścić. Muzycy są jednak zmęczeni i chowają instrumenty niedbale (nie zwracają uwagi na etykiety). Gdy wszystkie instrumenty są schowane do pudełek przybiega menadżer i krzyczy: "co Wy robicie! musicie zagrać jeszcze jeden utwór". Muzycy muszą znaleźć swoje instrumenty. Muszą to robić sekwencyjnie, jeden po drugim. Nie mogą się komunikować, oznaczać pudełek, zostawiać otwartych itp. Każdy muzyk podchodzi do wybranego przez siebie pudełka, otwiera je, sprawdza czy to jego instrument. Jeśli znalazł, zapamiętuje naklejoną etykietę i zamyka pudełko. Jeśli nie znalazł, może powtórzyć to dla innego pudełka. Każdy muzyk może sprawdzić maksymalnie 50 pudełek ($100/2$). W jaki sposób powinni sprawdzać pudełka aby zmaksymalizować szanse na powodzenie?

- ▶ Zapisz na dysku projekt PET zamieszczony na stronie przedmiotu
- ▶ Uruchom skrypt geant4, określający zmienne systemowe, niezbędne do wykorzystania pakietu Geant4 (. geant4)
- ▶ Skompiluj program wykorzystując zamieszczony CMakeList.txt
- ▶ Uruchom program
- ▶ Przenieś polecenia określające sposób wyświetlania z pliku PET.cpp (main) do zewnętrznego skryptu, uruchamianego domyślnie
- ▶ Dodaj możliwość wczytywania skryptu z argumentu podanego podczas uruchamiania programu ("argsów")
- ▶ Zapoznaj się z poleceniami dostępnymi w interfejsie
- ▶ Spróbuj zmodyfikować geometrię - dodaj kręgosłup do różowego walca