Wstęp do programowania

Wykład 9. Programowanie obiektowe

Plan wykładu

1. Wprowadzenie do C++
- instalacja,
- pierwszy program,
2. Klasy, atrybuty, metody.
- konstruktor i destruktor
3. Przeciążanie operatorów
- referencje,
- const, friend,
- strumienie, pliki.

C++: instalacja

sudo apt-get install g++

Pierwszy program:

```
#include <iostream>

int main(){
  std::cout << "Hello world!" << std::endl;
  return 1;
}
```

Kompilacja:

```
g++ hellp.cpp
```

Uruchomienie:

```
./a.out
```

C++: Klasy, pola, metody

Plik Vector3.h:

```
class Vector3 {

private:
    double coordinates[3];

public:
    Vector3();
    Vector3(double x, double y, double z);

    Vector3 add(Vector3 v);
    void print();
};
```

C++: Klasy, pola, metody

Plik Vector3.cpp:

```
#include "Vector3.h"
#include <iostream>

Vector3::Vector3() {
    for(int i=0; i<3; i++){
        this->coordinates[i] = 0.0;
    }
}

Vector3::Vector3(double x, double y, double z) {
    this->coordinates[0] = x;
    this->coordinates[1] = y;
    this->coordinates[2] = z;
}
```

C++: przeciążanie metod, modyfikatory dostępu

Klasa określa typ „zmiennej” zwanej obiektem. Obiekt składa się z pól i metod (funkcji). Metody są wywoływane na rzec wskazanego obiektu i mają dostęp do wszystkich pól i innych metod w danym obiekcie. Mogą one również korzystać, ze wskazanych pól i metod w obiektach opisywanych przez inne klasy, ale tylko do tych, z sekcji **`public`**.

Sekcja **`private`** to zmienne i metody prywatne, z których inni nie mogą korzystać.

Jest jeszcze sekcja **`protected`**, którą poznamy na kolejnym wykładzie.

C++: Konstruktor, destruktor

Metoda o nazwie klasy to konstruktor i jest ona wywoływana zawsze, gdy tworzony jest nowy obiekt (instancja klasy). Klasa może posiadać kilka konstruktorów. Jeśli klasa nie posiada konstruktora, to jego rolę pełni konstruktor domyślny, który inicjuje obiekt.

Gdy obiekt ma zostać usunięty z pamięci wywoływany jest destruktor:

```
Vector3::~Vector3();
```

Destruktor jest jeden (może być domyślny). Jego zadaniem jest posprzątanie po obiekcie – np.. gdy obiekt w konstruktorze alokuje pamięć na swoje atrybuty, to w destruktorze ta pamięć powinna być zwolniona.

C++: Klasy, pola, metody

Plik Vector3.cpp:

```
Vector3 Vector3::add(Vector3 v){
    Vector3 res;
    for(int i=0; i<3; i++){
        res.coordinates[i] = this->coordinates[i] + v.coordinates[i];
    }
    return res;
}

void Vector3::print(){
    std::cout << "[" << this->coordinates[0] << ", " <<
              this->coordinates[1] << ", " << this->coordinates[2] << "]";
}

int main(){
    Vector3 v1, v2(1,2,3), v3(2,3,4);
    v1.print();
    v2.print();
    (v2.add(v3)).print();
}
```

C++: Przeciążanie operatorów

Plik Vector3.h, w sekcji **`public`** dodajemy wpis:

```
Vector3 operator+(Vector3 v);
```

Plik Vector3.cpp, dodajemy wpis:

```
Vector3 Vector3::operator+(Vector3 v){
    Vector3 res;
    for(int i=0; i<3; i++){
        res.coordinates[i] = this->coordinates[i] + v.coordinates[i];
    }
    return res;
}
```

Teraz w procedurze **`main()`** możemy napisać:

```
(v2+v3).print();
```

C++: Przeciążanie operatorów

Plik Vector3.h, w sekcji **`public`** dodajemy wpis:

```
double operator[](int i);
```

Plik Vector3.cpp, dodajemy wpis:

```
double Vector3::operator[](int i){
    return this->coordinates[i];
}
```

Teraz w procedurze **`main()`** możemy napisać:

```
Vector3 v4 = v2+v3;
std::cout << v4[1] << std::endl;
```

Ale nie możemy napisać:

```
v4[1] = 22;  // błąd: v4[1] jest tymczasową kopią wartości zapisanej w v4[1]
```

C++: Rozwiązanie - wskaźniki

Problem można rozwiązać używając wskaźników:

```
double *operator[](int i);
```

Plik Vector3.cpp, dodajemy wpis:

```
double *Vector3::operator[](int i){
    return &(this->coordinates[i]);
    //  alternatywnie return (this->coordinates+i);
}
```

Teraz w procedurze `main()` możemy napisać:

```
std::cout << *(v4[1]) << std::endl;
*(v4[1]) = 22;  // pod adresem v4[1] zostanie wpisana wartość 22
std::cout << *(v4[1]) << std::endl;
```

C++: Rozwiązanie - referencje

Do oznaczenia referencji w C++ używa się znaku **&** (takiego jak adres zmiennej), jednak przy korzystaniu z referencji nie trzeba używać *****

```
double &operator[](int i);
```

Plik Vector3.cpp, dodajemy wpis:

```
double &Vector3::operator[](int i){
    return this->coordinates[i];
}
```

Teraz w procedurze **main()** możemy napisać:

```
std::cout << v4[1] << std::endl;
v4[1] = 22;  // pod adresem v4[1] zostanie wpisana wartość 22
std::cout << v4[1] << std::endl;
```

Referencje działają podobnie do wskaźników, z tym, że nie wymagają specjalnej semantyki - referencjami posługujemy się tak jak obiektami.

C++: const

Referencje są zalecanym sposobem przekazywania danych do metod:

```
Vector3 &operator+(Vector3 &v);
```

vjest referencją, a więc zmiany dokonane na **v** wewnątrz metody będą globalne. Jeśli chcemy się przed tym „zabezpieczyć” używamy słowa kluczowego **const**:

```
Vector3 &operator+(const Vector3 &v);
```

Jeśli wewnątrz metody spróbujemy zmienić v, kompilator pokaże błąd. Możemy też użyć słowa **const** przy deklaracji metody – oznacza to, że nie zmienia ona zawartości obiektu na rzecz którego jest ona wywoływana, np.:

```
void print() const;
```

Słowo **const** pojawia się zarówno w deklaracji jak i definicji metody.

C++: friend

Jeśli chcemy by metoda wywołana na rzecz innego obiektu miała dostęp do pól prywatnych naszego obiektu, przy jej definicji używamy słowa kluczowego **`friend`**.

```
friend std::ostream& operator<<(std::ostream& os, const Vector3 &v);
```

Implementacja:

```
std::ostream& operator<<(std::ostream& os, const Vector3 &v){
    os << "[" << v.coordinates[0] << ", " << v.coordinates[1] << ", "
       << v.coordinates[2] << "]";
    return os;
}
```

Przykład użycia (odpowiednik „starego” **`v4.print()`**):

```
std::cout << v4 << std::endl;
```

C++: strumienie

`std::cout` jest standardowym strumieniem wyjściowym (ekranem). Standardowym strumieniem wejściowym jest `std::in`. Przykład:

```
std::cin >> c;
```

Zapisze do zmiennej c wartość wpisaną z klawiatury. Strumienie można też związać z plikami:

```
void Vector3::toFile(const std::string &filename) {
    std::ofstream file(filename);
    file << "[" << v.coordinates[0] << ", " << v.coordinates[1] << ", "
         << v.coordinates[2] << "]";
    file.close();
}
```

Oczywiście należy też dopisać deklaravję tej metody w pliku nagłówkowym (z rozszerzeniem h):

Grafy

Dziękuję za uwagę