Języki i Paradygmaty Programowania

dr inż. Michał Ciesielczyk
michal.ciesielczyk@put.poznan.pl

Konsultacje:
środa, 08:30-09:30, pokój BM-319
środa, 11:15-11:45, sala M-216

Laboratorium 1/2

Powtórka

Wczesna inicjalizacja

```
class Binomial {
private:
        double a, b, c, x1, x2;
public:
        Binomial(int a, int b, int c);
};

Binomial::Binomial(int a, int b, int c) : a(a), b(b), c(c) {
        double delta = b*b - 4 * a*c;
        x1 = (-b - sqrt(delta)) / (2 * a);
        x2 = (-b + sqrt(delta)) / (2 * a);
}
```

Zwracamy uwagę na kolejność działań:

$2.0 / 4.0 * 2.0 \neq 2.0 / (4.0 * 2.0)$

Dziedziczenie

Laboratorium 03

Przeciążanie konstruktorów

```
struct A {
    struct B {
        B() { }
        B& operator=(const B& other) {
            return *this;
        }
    };
    B b;
    A() {
        this->b = B();
    }
};
A a;
```

Pusta struktura B z domyślnym konstruktorem i assignment operator

Jedno pole w klasie A (zmienna b typu B)

Przeciążony konstruktor

Jakie funkcje wywoła ta instrukcja?

Konstruktory: Member initialization

```
struct A {
    struct B {
        B() { }
        B& operator=(const B& other) {
            return *this;
        }
    };
    B b;
    A() : b(B()) {
    }
};

A a;
```

Pusta struktura B z domyślnym konstruktorem i assignment operator

Jedno pole w klasie A (zmienna b typu B)

Przeciążony konstruktor (member initialization)

Jakie funkcje wywoła ta instrukcja?

Dziedziczenie

```
class A {
     /* ... */
};

class B : A {// klasa B dziedzicząca po A
     /* ... */
};
```

Specyfikator protected

```
class A {
protected:
        int a;
};

class B : A {
        void b() {
                a = 1;   // OK
        }
};

void c() {
        A a;
        a.a = 1; // ERROR
}
```

Tryby dziedziczenia

```
class A {};
```

- private (dziedziczenie domyślne)

```
class B : private A {};
```

- protected

```
class C : protected A {};
```

- public

```
class D : public A {};
```

Funkcje zaprzyjaźnione

```
class C {                       // klasa C
private:
        int e;                  // zmienna prywatna
        friend int f();         // funkcja zaprzyjaźniona
};

int f() {
        e = 1;                  // dostęp do zmiennej prywatnej
}
```

- Funkcja, która ma dostęp do prywatnych składników klasy.
  - Funkcja zaprzyjaźniona nie jest składnikiem klasy, która deklaruje przyjaźń.
  - Funkcja może być przyjacielem wielu klas.

Przysłanianie

```
class A {
public:
	void sayHello() {
		cout << "Hello, my name is A." << endl;
	}
};

class B : public A {
public:
	void sayHello() {
		cout << "Hello, my name is B." << endl;
	}
};
```

Przeciążanie

```
class A {
public:
	virtual void sayHello() {
		cout << "Hello, my name is A." << endl;
	}
};

class B : public A {
public:
	virtual void sayHello() {
		cout << "Hello, my name is B." << endl;
	}
};
```

Demo

Przysłanianie vs. Przeciążanie

Konwersja dynamic_cast

```
B b;
b.sayHello();
A& a = b;
a.sayHello();
B& b2 = a;
B& b2 = dynamic_cast<B&> (a);
b2.sayHello();
```

Wirtualny destruktor

```
class A {
public:
        virtual ~A() {
                cout << "usuwamy A" << endl;
        }
};

class B : public A {
public:
        virtual ~B() {
                cout << "usuwamy B" << endl;
        }
};
```

Co zostanie wyświetlone na ekranie po wywołaniu destruktora B?

Demo

Wirtualny destruktor

Zadania