

Programovacie techniky

Pavol Marák

C++ Triedy



OBSAH

- Trieda a objekt
- Atribúty a metódy
- Špecifikátory prístupu
- Konštruktor
- Deštruktor
- Vzorová implementácia v C/C++

Trieda a objekt

- C++ je objektovo orientovaný jazyk a prináša do programovania koncept tried.

Trieda a objekt

- C++ je objektovo orientovaný jazyk a prináša do programovania koncept tried.
- **Trieda** je používateľom definovaný dátový typ skladajúci sa z:
 - **atribútov** (údaje triedy)
 - **metód** (operácie/chovanie triedy)

Trieda a objekt

- C++ je objektovo orientovaný jazyk a prináša do programovania koncept tried.
- **Trieda** je používateľom definovaný dátový typ skladajúci sa z:
 - **atribútov** (údaje triedy)
 - **metód** (operácie/chovanie triedy)
- Atribúty a metódy sa volajú **členy** triedy (z angl. class members).

Trieda a objekt

- **Trieda** je predpis (z angl. blueprint), t.j. určuje aké vlastnosti a operácie má obsahovať entita (študent, zviera, predmet, atď.).

Trieda a objekt

- **Trieda** je predpis (z angl. blueprint), t.j. určuje aké vlastnosti a operácie má obsahovať entita (študent, zviera, predmet, atď.).
- **Objekt** je konkrétny nositeľ vlastností a správania, vytvorený podľa predpisu triedy (t.j. konkrétna premenná).

Trieda a objekt

- **Trieda** je predpis (z angl. blueprint), t.j. určuje aké vlastnosti a operácie má obsahovať entita (študent, zviera, predmet, atď.).
- **Objekt** je konkrétny nositeľ vlastností a správania, vytvorený podľa predpisu triedy (t.j. konkrétna premenná).
- Keď vytvárame objekt, hovoríme, že **vytvárame inštanciu triedy** (z angl. object instantiation).

Príklad triedy

```
class Time {  
    private:  
        int hours;  
        int minutes;  
        int seconds;  
};
```

Príklad triedy

```
class Time {  
    private:  
        int hours;  
        int minutes;  
        int seconds;  
};
```



Atribúty

Príklad triedy

```
class Time {  
    private:  
        int hours;  
        int minutes;  
        int seconds;  
    public:  
        void setTime(int,int,int);  
        char* getTime();  
};
```



Atribúty

Príklad triedy

```
class Time {  
    private:  
        int hours;  
        int minutes;  
        int seconds;  
    public:  
        void setTime(int,int,int);  
        char* getTime();  
};
```

Atribúty

Metódy

Metódy triedy

- Metódy triedy môžeme definovať:
 - Vo vnútri triedy.
 - Mimo triedy.

Definícia metódy vo vnútri triedy

```
class Point {  
    private:  
        int x,y;  
    public:  
        int getX(){ return x; }  
        int getY(){ return y; }  
};
```

Definícia metódy mimo triedy

```
class Point {  
    private:  
        int x,y;  
    public:  
        int getX();  
        int getY();  
};  
int Point::getX(){return x;}  
int Point::getY(){return y;}
```

Špecifikátory prístupu

private:

K členom sa dá prístupit' len zvnútra triedy (+spriatelené funkcie a premenné).

Špecifikátory prístupu

private:

K členom sa dá prístupit' len zvnútra triedy (+spriatelené funkcie a premenné).

protected:

K členom sa dá prístupit' zvnútra triedy a v odvodených triedach (+spriatelené funkcie a premenné).

Špecifikátory prístupu

private:

K členom sa dá prístupit' len zvnútra triedy (+spriatelené funkcie a premenné).

protected:

K členom sa dá prístupit' zvnútra triedy a v odvodených triedach (+spriatelené funkcie a premenné).

public:

Členy sú prístupné zvnútra/zvonku.

Špecifikátory prístupu

```
class MyClass {  
    int a;  
};
```

Špecifikátory prístupu

```
class MyClass {  
    int a;  
};
```



Prístup je automaticky
private

Špecifikátory prístupu

```
class MyClass {  
    private:  
        int a;  
    public:  
        int b;  
};
```

```
int main(){  
    MyClass m;  
    cin >> m.a;  
    return 0;  
}
```

Špecifikátory prístupu

```
class MyClass {  
    private:  
        int a;  
    public:  
        int b;  
};
```

```
int main(){  
    MyClass m;  
    cin >> m.a;  
    return 0;  
}
```



Pôjde kompilovať?

Špecifikátory prístupu

Nepôjde kompilovať

```
class MyClass {  
    private:  
        int a;  
    public:  
        int b;  
};
```

```
int main(){  
    MyClass m;  
    cin >> m.a;  
    return 0;  
}
```

```
int main(){  
    MyClass m;  
    cin >> m.b;  
    return 0;  
}
```



Špecifikátory prístupu

Nepôjde kompilovať

```
class MyClass {  
    private:  
        int a;  
    public:  
        int b;  
};
```

```
int main(){  
    MyClass m;  
    cin >> m.a;  
    return 0;  
}
```



```
int main(){  
    MyClass m;  
    cin >> m.b;  
    return 0;  
}
```

Pôjde kompilovať?

Špecifikátory prístupu

Nepôjde kompilovať

```
class MyClass {  
    private:  
        int a;  
    public:  
        int b;  
};
```

```
int main(){  
    MyClass m;  
    cin >> m.a;  
    return 0;  
}
```



```
int main(){  
    MyClass m;  
    cin >> m.b;  
    return 0;  
}
```

Pôjde kompilovať

Štruktúra vs. trieda v C++

- Štruktúra je v jazyku C++ rovnaká ako trieda, až na rozdielnu predvolenú viditeľnosť atribútov.

Štruktúra vs. trieda v C++

- Štruktúra je v jazyku C++ rovnaká ako trieda, až na rozdielnu predvolenú viditeľnosť atribútov.

Štruktúra

```
struct Time {  
    int hours;  
    int minutes;  
    int seconds;  
};
```

Automaticky
public

Štruktúra vs. trieda v C++

- Štruktúra je v jazyku C++ rovnaká ako trieda, až na rozdielnu predvolenú viditeľnosť atribútov.

Štruktúra

```
struct Time {  
    int hours;  
    int minutes;  
    int seconds;  
};
```

Automaticky
public

Trieda

```
class Time {  
    int hours;  
    int minutes;  
    int seconds;  
};
```

Automaticky
private

Konštruktor triedy

- Špeciálna členská metóda triedy, ktorá sa zavolá pri vzniku objektu.
- Slúži na inicializáciu objektu.
- Má rovnaký názov ako trieda.
- Nemá návratový typ.



Konštruktor triedy

Typy konštruktorov

- default
- parametrický
- kopírovací (copy)
- presúvací (move)
- konverzný



Default konštruktor

Je to konštruktor bez parametrov alebo s default (implicitnými) hodnotami parametrov.

Default konštruktor

Verzia bez parametrov

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(){  
            a = 0;  
        }  
};
```

```
int main(){  
    MyClass a;  
    return 0;  
}
```


Default konštruktor

Verzia s default parametrom

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int new_a = 0){  
            a = new_a;  
        }  
};
```



Default
parameter

```
int main(){  
    MyClass a;  
    MyClass b(1);  
    return 0;  
}
```

Parametrický konstrukt

Je to konstrukt s parametry.

Parametrický konstruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int b, int c){  
            a = b+c;  
        }  
};
```

```
int main(){  
    MyClass a(1,2);  
    return 0;  
}
```

Konštruktor s inicializačným zoznamom členov

- Používa sa na inicializáciu atribútov triedy (namiesto priradenia hodnôt).
- Anglický názov je „member initializer list”.
- Nemýliť si s knižničnou C++ triedou **std::initializer_list**.

Konštruktor s inicializačným zoznamom členov

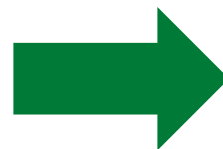
```
class MyClass {  
    private:  
        int a,b;  
    public:  
        MyClass(int n_a, int n_b){  
            a = n_a;  
            b = n_b;  
        }  
};
```



priradenie

Konštruktor s inicializačným zoznamom členov

```
class MyClass {  
    private:  
        int a,b;  
    public:  
        MyClass(int n_a, int n_b){  
            a = n_a;  
            b = n_b;  
        }  
};
```



```
class MyClass {  
    private:  
        int a,b;  
    public:  
        MyClass(int n_a, int n_b) :  
            a(n_a),  
            b(n_b){  
        }  
};
```

inicializácia

Kopírovací konštruktor

Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy.

Kopírovací konštruktor

Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy.

Prototyp kopírovacieho konštruktora:

ClassName (**ClassName**& other);

ClassName (**const** **ClassName**& other);

Kopírovací konštruktor

Zavolá sa v týchto prípadoch:

- Návrat objektu z funkcie (**copy elision**: kompilátor môže vykonať **return value optimization** a vtedy sa kopírovací konštruktor nezavolá)
- Objekt je odovzdaný ako parameter funkcie (by value)
- Objekt sa inicializuje iným objektom

```
ClassName obj(other_obj);
```

```
ClassName obj = other_obj;
```

Kopírovací konstruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass() : a(0) { }  
        MyClass(const MyClass& obj){  
            a = obj.a;  
        }  
};
```

```
int main(){  
    MyClass a;  
    MyClass b(a);  
    // MyClass b = a;  
    return 0;  
}
```

Kopírovací konštruktor

Plytká vs. hlboká kópia

Kopírovací konštruktor vygenerovaný kompilátorom vykonáva plytkú kópiu.

V prípade, že objekt vlastní dynamicky alokované zdroje a potrebujeme spraviť hlbokú kópiu, vtedy je nutné definovať svoj vlastný kopírovací konštruktor.

Kopírovací konstruktor

Plytká kópia

```
class MyClass {  
    private:  
        char* data;  
    public:  
        MyClass(int n = 10) {  
            data = new char[n];  
        }  
};
```

```
int main(){  
    MyClass a;  
    MyClass b(a);  
    return 0;  
}
```

Kopírovací konštruktor

Plytká kópia

```
class MyClass {  
    private:  
        char* data;  
    public:  
        MyClass(int n = 10) {  
            data = new char[n];  
        }  
};
```

```
int main(){  
    MyClass a;  
    MyClass b(a);  
    return 0;  
}
```

Zavola sa kompilátorom
vygenerovaný kopírovací
konštruktor

Kopírovací konštruktor

Hlboká kópia

V triede **MyClass** musíme vytvoriť vlastný kopírovací konštruktor, ktorý vytvorí hlbokú kópiu poľa '**data**'.

Kopírovací konštruktor

Hlboká kópia

V triede **MyClass** musíme vytvoriť vlastný kopírovací konštruktor, ktorý vytvorí hlbokú kópiu poľa '**data**'.

```
MyClass(const MyClass& obj) {  
    data = new char[strlen(obj.data)+1];  
    strcpy(data,obj.data);  
}
```

Move konštruktor

- Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy presunutím jeho zdrojov (vyhneme sa tak relatívne náročnému kopírovaniu).

Move konštruktor

- Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy presunutím jeho zdrojov (vyhneme sa tak relatívne náročnému kopírovaniu).
- Po presunutí, zostane argument move konštruktora v platnom, ale nešpecifikovanom stave.

Move konštruktor

- Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy presunutím jeho zdrojov (vyhneme sa tak relatívne náročnému kopírovaniu).
- Po presnutí, zostane argument move konštruktora v platnom, ale nešpecifikovanom stave.
- Volá sa, keď je objekt inicializovaný z **rvalue** (čo môže byť **prvalue** alebo **xvalue**).

Move konštruktor

- Inicializuje objekt pomocou iného existujúceho objektu rovnakej triedy presunutím jeho zdrojov (vyhneme sa tak relatívne náročnému kopírovaniu).
- Po presunutí, zostane argument move konštruktora v platnom, ale nešpecifikovanom stave.
- Volá sa, keď je objekt inicializovaný z **rvalue** (čo môže byť **prvalue** alebo **xvalue**).
- Kompilátor vytvorí move konštruktor, keď nemáme zadefinovaný vlastný copy konštruktor, copy a move assignment operátor a deštruktor

Move konštruktor

Copy vs. move konštruktor

Copy konštruktor

```
Class_name(Class_name& obj)
{
    data = new char[strlen(obj.data)+1];
    strcpy(data,obj.data);
}
```

Move konstruktor

Copy vs. move konstruktor

Copy konstruktor

```
Class_name(Class_name& obj)
{
    data = new char[strlen(obj.data)+1];
    strcpy(data,obj.data);
}
```

Move konstruktor

```
Class_name(Class_name&& obj)
{
    data = obj.data;
    obj.data = nullptr;
}
```

Move konstruktor

```
class MyClass {  
    int a;  
public:  
    MyClass(int n=10) {  
        a = n;  
        cout << "default" << endl;  
    }  
    MyClass(const MyClass& m){  
        cout << "copy" << endl;  
    }  
    MyClass(const MyClass&& m){  
        cout << "move" << endl;  
    }  
};
```

Move konstruktor

```
class MyClass {  
    int a;  
public:  
    MyClass(int n=10) {  
        a = n;  
        cout << "default" << endl;  
    }  
    MyClass(const MyClass& m){  
        cout << "copy" << endl;  
    }  
    MyClass(const MyClass&& m){  
        cout << "move" << endl;  
    }  
};
```

```
MyClass fn(MyClass a){  
    return a;  
}
```

```
int main() {  
    MyClass a;  
    MyClass b(a);  
    MyClass c = fn(a);  
    return 0;  
}
```

Move konstruktor

```
class MyClass {  
    int a;  
public:  
    MyClass(int n=10) {  
        a = n;  
        cout << "default" << endl;  
    }  
    MyClass(const MyClass& m){  
        cout << "copy" << endl;  
    }  
    MyClass(const MyClass&& m){  
        cout << "move" << endl;  
    }  
};
```

```
MyClass fn(MyClass a){  
    return a;  
}
```

```
int main() {  
    MyClass a;  
    MyClass b(a);  
    MyClass c = fn(a);  
    return 0;  
}
```

default
copy
copy
move

Konverzný konštruktor

- Pri inicializácii objektu priradením hodnoty kompilátor vyhladá a zavolá vhodný konštruktor triedy.

Konverzný konštruktor

- Pri inicializácii objektu priradením hodnoty kompilátor vyhladá a zavolá vhodný konštruktor triedy.
- Ak označíme konštruktor triedy kľúčovým slovom **explicit**, daný konštruktor nebude môcť slúžiť ako konverzný konštruktor.

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int data) {}  
        MyClass(double data) {}  
};
```

```
int main(){  
    MyClass a = 6;  
    return 0;  
}
```

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int data) {}  
        MyClass(double data) {}  
};
```

```
int main(){  
    MyClass a = 6;  
    return 0;  
}
```

Zavolá sa konštruktor
MyClass(int data);

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int data) {}  
        MyClass(double data) {}  
};
```

```
int main(){  
    MyClass a = 6;  
    MyClass b = 9.88;  
    return 0;  
}
```

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        MyClass(int data) {}  
        MyClass(double data) {}  
};
```

```
int main(){  
    MyClass a = 6;  
    MyClass b = 9.88;  
    return 0;  
}
```

Zavolá sa konštruktor
MyClass(double data);

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        explicit MyClass(int data) {}  
        explicit MyClass(double data) {}  
};
```

Označenie
konštruktorov
ako explicit

```
int main(){  
    MyClass a = 6;  
    MyClass b = 9.88;  
    return 0;  
}
```

Konverzný konštruktor

```
class MyClass {  
    private:  
        int a;  
    public:  
        explicit MyClass(int data) {}  
        explicit MyClass(double data) {}  
};
```

Označenie
konštruktorov
ako explicit

```
int main(){  
    MyClass a = 6;  
    MyClass b = 9.88;  
    return 0;  
}
```

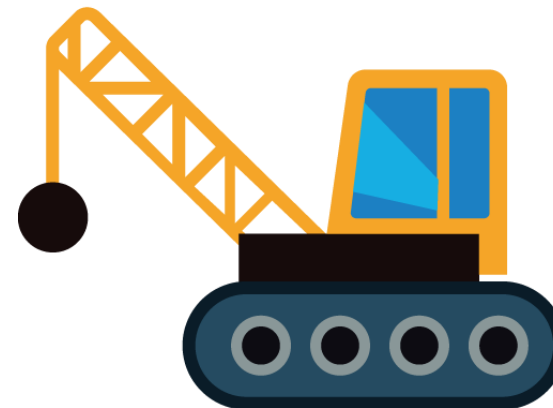
Program nepôjde
kompilovať.

Deštruktor triedy

Je členská metóda, ktorá sa zavolá v čase zániku objektu.

Objekt **zaniká** keď:

- končí jeho rozsah platnosti (z angl. scope)
- keď sa explicitne de-alokuje (ak bol predtým dynamicky alokovaný)




Deštruktor

```
class MyClass {  
    private:  
        char* data;  
    public:  
        ...  
    ~MyClass(){  
        delete[] data;  
    }  
};
```

```
void fn() {  
    MyClass obj;  
    // praca s obj  
}
```

```
int main(){  
    fn();  
    return 0;  
}
```



Kedy sa zavolá
deštruktor objektu
'obj'?

Deštruktor

```
class MyClass {  
    private:  
        char* data;  
    public:  
        ...  
        ~MyClass(){  
            delete[] data;  
        }  
};
```

```
MyClass* fn() {  
    MyClass* obj = new MyClass;  
    return obj;  
}
```

```
int main()  
{  
  
    MyClass* obj = fn();  
    delete obj;  
    return 0;  
}
```

Kedy sa zavolá
deštruktor objektu
'obj'?

Vzorová implementácia v C/C++