2. lekce

Objekty v jazyce C++

Objekt je speciální datový typ, který má atributy a metody.

Atributy jsou **členské proměnné**, které jsou součástí objektu.

Např: objekt je Student, jeho atributy jsou Jmeno, Prijmeni, Datum_Narozeni,....

Metody jsou **funkce,** které lze volat (spouštět) **pouze na daný objekt.**

Např. objekt je Student, metoda je Zmen_Jmeno (funkce, která v objektu změní hodnotu členské proměnné Jmeno).

Definice objektu

```
class Jmeno_objektu
{
private:
    atributy
public:
    metody
};
```

Tělo objektu můžeme rozdělit až na tři části:

- **public** Všechno, co je za *public* je viditelné (použitelné) ze všech částí programu. Obvykle zde dáváme metody, které budeme na objekt volat.
- **private** Všechno, co je za *private* je použitelné pouze v daném objektu. To znamená, že s touto částí můžeme pracovat pouze uvnitř objektu. Obvykle sem dáváme atributy a pomocné metody, které nebudeme využívat v jiných částech programu (mimo objekt).
- **protected** Podobné, jako *private*, rozdíl je v tom, že to, co je za *protected* je viditelné pro všechny potomky daného objektu. (Potomek je každý objekt, která od originálního objektu dědí atributy i metody).

Když nepoužijeme *public* ani *private* ani *protected*, je všechno uvnitř objektu *private*!

Konstruktor objektu

Konstruktor je metoda, která vytvoří daný objekt.
Metoda konstruktoru se jmenuje stejně jako objekt!!!

Příklad:

```
class Student
{
private:
    string m_Jmeno;
    string m_Prijmeni;
    date m_Dat_nar;
public:
    Student(string Jmeno, string Prijmeni, date Dat_nar) : m_Jmeno(Jmeno),
                                                           m_Prijmeni(Prijmeni),
                                                           m_Dat_nar(Dat_nar)
    {
        nějaké příkazy na ošetření vstupu – kontrola vstupních hodnot
    }
};
```

Poznámka:

(string Jmeno, string Prijmeni, date Dat_nar) = *vstupní parametry kontruktoru*

Před atribut objektu dáváme **m_**, které značí **member of.**

Pokud nevytvoříme žádný konstruktor, můžeme používat defaultní konstruktor, který vytvoří sám kompilátor. Takový konstruktor definuje všechny atributy, ale neinicializuje je – v takto vytvořených atributech je nedefinovaná hodnota (náhodná hodnota, která se nachází v místě paměti, kde byl vytvořen atribut).
To znamená, že se vytvoří místo v operační paměti, ale neznáme hodnotu v této oblasti (buňce) paměti. Hodnoty atributů objektu vytvořeného defaultním konstruktorem následně zadáme pomocí členských metod.

Nově vytvořené objekty jsou uloženy v operační paměti. Mohou být uložené :

- samostatně
- v poli
- datovém kontejneru

Kopírující konstruktor

Metoda, která zkopíruje atributy jednoho objektu do druhého objektu stejného typu
– ze Student do Student (vytvoří kopii objektu)

Pro objekt Student vytvoříme kopírující konstruktor:

```
Student(const Student & Zdroj) : m_Jmeno(Zdroj.m_Jmeno),
                                 m_Prijmeni(Zdroj.m_Prijmeni),
                                 m_Dat_nar(Zdroj.m_Dat_nar)
{
}
```

Při kopírování se použije reference, ale po vytvoření kopie jsou zdrojový a nový objekt na sobě nezávislé a změna atributů jednoho se neprovede ve druhém.

m_Jmeno(Zdroj.m_Jmeno) znamená:
do nového atributu m_Jmeno se zkopíruje hodnota atributu ze zdrojového objektu.

Příklad definice zdroje:

```
Student A("Miroslav","Jilek","1.1.2000");
Student B(A);
```

Pokud kopírující konstruktor nevytvoříme, pak kompilátor vytvoří vlastní, který bude, v tomto případě, vypadat stejně jako ten náš. Problém nastane, jestliže použijeme poinetry. V takovém případě většinou musíme vytvořit vlastní kopírující konstruktor.

Destruktor

Metoda, která provede uvolnění objektu z paměti.
V případě, že používáme pouze staticky definované atributy (ne pointery), není třeba destruktor vytvářet!

Příklad: ***objekt s ukazatelem***

```
class Cislo
{
public:
	int* m_hodnota1;
	int* m_hodnota2;
//konstruktor:
	Cislo (int x, int y) : m_hodnota1(new int(x)), m_hodnota2(new int(y)){}
//destruktor:
	~Cislo(void)
	{
		delete m_hodnota1;
		delete m_hodnota2;
	}
};
```

Přetížení operátorů u objektů

Protože C++ neví jak aplikovat standardní operátory (+,-,*,/,<,>,=,==,<<,>>, && (AND), || (OR), !,...) na uživatelsky definované objekty musíme tyto operátory přetížit = vytvořit funkce obsluhují objekt – například definice operandu pro násobení zlomků.

```
class Zlomek
{
private:
    int m_citatel;
    int m_jmenovatel;
public:
    Zlomek(int citatel, int jmenovatel) : m_citatel(citatel),
                                          m_jmenovatel(jmenovatel)
    {
        if (!jmenovatel) throw "Jmenovatel nesmí být nula!";
    }
    Zlomek operator * (const Zlomek & Cinitel) const
        // první const znamená, že parametr (objekt) Cinitel zůstane po dobu funkce konstantní
        // druhý const znamená, že objekt, na který tuto metodu voláme, zůstane také konstantní
        {return Zlomek(m_citatel * Cinitel.m_citatel, m_jmenovatel *
        Cinitel.m_jmenovatel);}
};
```

Přetížení operátoru pro výstup

U většiny operátorů se předpokládá, že první operand bude objekt, pro který operátor přetěžujeme a druhý operand bude jakýkoli datový typ.

Např.: **zlomek + int** můžeme přetížit standardním způsobem, ale
int + zlomek nemůžeme, protože první operand nebyl objekt, který přetěžujeme

U přetížení operátoru Výstup je ale vždy prvním operandem výstupní stream (klíčové slovo cout) a až druhým operandem je náš objekt. Proto tento operátor není možné přetížit standardním způsobem. Musíme ho přetížit pomocí spřátelené funkce **(friend)** s daným objektem. Funkci friend zapisujeme mimo objekt!

Spřátelená funkce může používat i ty metody a atributy, které jsou v sekci private!

Příklad:

class Zlomek *…. (stejná definice objektu jako v předešlé ukázce)*

```
friend ostream & operator << (ostream & O, const Zlomek & Z)
{
    return O << Z.m_citatel << "/" << Z.m_jmenovatel;
    //ve streamu, který se jmenuje O je uloženo  Z.m_citatel <<"/" << Z.m_jmenovatel
}
```

ostream je výstupní proud (datový typ), cout je jeden z výstupních proudů

<< znamená, že vše postupně vkládáme do streamu se jménem O

… a vše vypíšeme příkazem v programu (mimo funkci friend):

cout << proměnná_typu_Zlomek;

cout je první operand – výstupní proud (stream).
Zlomek je druhý operand.