Základy programování v C++ 21. cvičení Letmý úvod do objektově orientovaného programování (v C++) II. Konstruktory a destruktor

Zuzana Petříčková

6. prosince 2019

Přehled

1. Letmý úvod do objektového programování (v C++)
   - Konstruktor
   - Konvence při práci se třídami
   - Destruktor
   - Příklad

Terminologie

- **třída (class)** ... objektový datový typ
- **instance (objekt, object)** ... proměnná objektového typu

Složky třídy

- **atribut** ... datová složka
- **metoda** ... funkce
  - **konstruktor** ... postará se o vytvoření a inicializaci nové instance
  - **destruktor** ... postará se o zrušení instance

1. **instanční** ... metoda pro práci s jednotlivými instancemi
2. **třídní** ... metoda pro práci s třídou jako celek

stručně:

- na rozdíl od struktury definujeme u třídy zároveň **atributy** i **metody**
- **třída** ale přináší další nové možnosti

Konstruktor a destruktor

- speciální metody, které slouží (k vytvoření a) inicializaci instance třídy (**konstruktor**) a k zrušení instance třídy (**destruktor**)
- nevoláme je přímo, volají se automaticky při vzniku / zániku instance třídy

Co bývá v konstruktoru:

- inicializace datových složek (atributů)
- vytvoření (alokace) dynamických datových složek
- příp. otevření souboru ap.

Co bývá v destruktoru:

- zrušení (dealokace) dynamických datových složek
- příp. zavření souboru ap.

Konstruktor

- metoda, která slouží k (vytvoření a) inicializaci instance
- konstruktor nelze volat přímo, je volán automaticky:
  - při vytvoření instance
  - při předávání parametrů objektového typu hodnotou
  - při konverzích
- konstruktor se jmenuje stejně jako identifikátor třídy
- jeho deklarace neobsahuje typ návratové hodnoty
- konstruktor může mít parametry libovolného typu s výjimkou své třídy (ale může mít jako parametr referenci na svou třídu)
- třída může mít několik konstruktorů
- pokud nedeklarujeme žádný konstruktor, překladač vytvoří implicitní konstruktor bez parametrů s prázdným tělem a implicitní tzv. kopírovací konstruktor

Konstruktor ... příklad

```
class Zamestnanec
{
    string jmeno;
    unsigned int plat;
    /* konstruktor nelze kombinovat s inicializaci, napr.:
       unsigned int plat=0;
    */
public:
    ...
    Zamestnanec(string _jmeno, unsigned int _plat)
    {
        jmeno = _jmeno;
        plat = _plat;
    }
    Zamestnanec()
    {
        jmeno = "";
        plat = 0;
    }
};
int main()
{
```

Konstruktor ... defaultní hodnoty parametrů

```
class Zamestnanec
{
    string jmeno;
    unsigned int plat;
    /* konstruktor nelze kombinovat s inicializaci, napr.:
       unsigned int plat=0;
    */
public:
    ...
    Zamestnanec(string _jmeno="", unsigned int _plat=0)
    {
        jmeno = _jmeno;
        plat = _plat;
    }
};
int main()
{
    Zamestnanec x, y("Jan"), z("Petr",45000);
    Zamestnanec *px = new Zamestnanec; //new Zamestnanec();
                    // new Zamestnanec("Jan"); new Zamestnanec("Pet
    delete px;
}
```

Inicializační část konstruktoru

- alternativní způsob inicializace (nestatických) atributů:

```
Zamestnanec::Zamestnanec() : jmeno(""), plat(30000)
{
    ...
}

Zamestnanec::Zamestnanec(string _jmeno, int _plat) : jmeno(_jmeno)
{
    ...
}
```

inicializační část:

- je od hlavičky oddělená dvojtečkou
- proběhne před vstupem do těla konstruktoru
- konstantní atributy lze inicializovat jen v inicializační části

Konvence při práci se třídami

- každé třídě odpovídá jeden hlavičkový a jeden zdrojový soubor (oba pojmenované stejně jako třída)
  - hlavičkový soubor ... deklarace třídy (obsahuje pouze informativní deklarace metod)
  - zdrojový soubor ... definiční deklarace metod
- Většina moderních vývojových nástrojů umožňuje vytvářet nové třídy takto rozdělené do souborů ,,v jednom kroku”

<**Ukázka ve Visual Studiu** >

Destruktor

- speciální metoda, která slouží k zrušení instance:
  - uvolnění dynamicky alokované paměti
  - uzavření souborů, s nimiž třída pracuje apod.
  - má i své skryté úkoly
- destruktor je volán automaticky při zániku instance
- jmenuje se stejně jako identifikátor třídy, před identifikátorem je znak ~
- jeho deklarace neobsahuje typ návratové hodnoty a nesmí mít parametry
- každá třída může mít pouze jeden destruktor

```
Zamestnanec::~Zamestnanec()
{
    ...
}
```

Destruktor

- destruktor je volán automaticky při zániku instance:
  - **lokální instance třídy** ... destruktor je volán na konci bloku, v němž je instance deklarovaná
  - **globální a statické lokální instance třídy** ... destruktor je volán po ukončení funkce main()
  - **dynamická instance třídy** ... operátor delete nejprve zavolá destruktor a pak uvolní paměť, kterou instance zabírala
- pokud nedeklarujeme destruktor, překladač vytvoří implicitní destruktor s prázdným tělem

Příklad: třída s dynamickými datovými složkami

```
class Vektor
{
  float *a;
  const int n;
 public:
  Vektor(int _n): n(_n)
  {
        a = new float[n];
        for (int i = 0; i < n; i++)
              a[i] = 0;
  }
  ~Vektor()
  {
        delete[] a;
  }
};
```

Příklad na procvičení I : Zlomek (doplnění)

- Pozměňte třídu Zlomek následovně:
  - Přidejte Zlomku konstruktor bez parametrů, který inicializuje čitatel i jmenovatel na 1.
  - Přidejte Zlomku druhý konstruktor se dvěma parametry (hodnota čitatele a hodnota jmenovatele).
  - Oba konstruktory vyzkoušejte (pro lokální i dynamické proměnné)

Příklad na procvičení II : Chytré pole

- Přeměňte strukturu **ChytrePole** z 16. cvičení na třídu se soukromými datovými složkami.
  - Deklarace třídy bude v hlavičkovém souboru **ChytrePole.h** a definiční deklarace metod budou v souboru **ChytrePole.cpp**.
  - Přidejte třídě **ChytrePole** konstruktor a destruktor (na základě původních funkcí **vytvor(...)** a **zrus(...)**).
  - Implementujte konstantní metodu **vypis()** (na základě původní funkce)
  - Implementuje metody **pridejNaKonec()** a **smazPrvek(...)** (na základě původních funkcí **pridej(...)** a **smaz(...)**)