Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 18.01. 2001
Url: http://www.builder.cz/art/cpp/cpp_konstruktor.html

Vytváření instancí - konstruktory, destruktory

Konstruktor

Každá instance je vytvořena pomocí speciální metody - konstruktoru. Konstruktor se musí jmenovat stejně jako třída, jejíž instance bude vytvářet, nesmí mít návratovou hodnotu a měl by být veřejný. Každá třída může mít k dispozici více konstruktorů s různými parametry. Jedná se o přetěžování konstruktorů - o přetěžování jsem se zmiňoval v 1. článku "Od C k C++". Nemá-li třída žádný konstruktor (Například třída `MojePrvniTrida` v mém předchozím článku "Vytváření tříd, instance třídy, zasílání zpráv".) vytvoří překladač tak zvaný implicitní konstruktor bez parametrů. V konstruktoru by se měla provést inicializace dané instance. Nastavit všechny její atributy, vytvořit instance tříd, které obsahuje atd... Uvedu příklad jednoduché třídy `Zlomek` i s konstruktory.

```
class Zlomek
{
   private:
      int Jmenovatel, Citatel;
   public:
      Zlomek();  /* Bezparametrický konstruktor */
      Zlomek(int cislo);
      Zlomek(int citatel, int jmenovatel);
      /* Ostatní metody třídy zlomek: */
      int dejCitatel() const;
      int dejJmenovatel() const;
      void nastavCitatel(int citatel);
      void nastavJmenovatel(int jmenovatel);
      float hodnota() const;
};

Zlomek::Zlomek()
{
   Jmenovatel = 1;
   Citatel = 0;
}

Zlomek::Zlomek(int cislo)
{
```

```
    Citatel = cislo;
    Jmenovatel = 1;
}

Zlomek::Zlomek(int citatel, int jmenovatel)
{
    Citatel = citatel;
    if (jmenovatel != 0)
       Jmenovatel = jmenovatel;
    else
       Jmenovatel = 1;
}

int Zlomek::dejCitatel() const
{ return Citatel; }

int Zlomek::dejJmenovatel() const
{ return Jmenovatel; }

void Zlomek::nastavCitatel(int citatel)
{ Citatel = citatel; }

void Zlomek::nastavJmenovatel(int jmenovatel)
{
    if (jmenovatel != 0)
       Jmenovatel = jmenovatel;
}

float Zlomek::hodnota() const
{ return ((float)Citatel)/Jmenovatel; }
```

Nejprve vysvětlím některé možné nejasnosti ke zdrojovému textu, i když se přímo netýkají zaměření článku:

1. Klíčové slovo `const` za některými metodami. - Takto označená metoda nemění nijak vnitřní stav instance. V C++ existují "konstantní instance", stejně jako v C konstantní proměnné. V C++ může být konstantní jak proměnná "primitivního" datového typu, tak i instance. "Konstantní instanci" nelze nijak měnit vnitřní stav. Lze pro takovou instanci vyvolat pouze metody deklarované klíčovým slovem `const` .
2. Možná někoho napadne otázka, jestli není lepší napsat atributy jako veřejné a ušetřit psaní metod nastav... a dej... - Veřejné atributy porušují princip zapouzdření. Kdybych se nyní rozhodl nějak změnit reprezentaci čitatele a jmenovatele, musel bych změnit jen metody třídy Zlomek. Objekt se bere jako černá skříňka, která by se změnila jen uvnitř, vůči okolí by se jako celek nezměnila. S veřejnými atributy by to tak lehké nebylo. Musel by se změnit celý program všude tam, kde by byly používány nějaké instance třídy `Zlomek` . I když uznávám, že v takto jednoduché třídě jako je `Zlomek` bych velmi těžce odolával pokušení dát atributy jako soukromé. Podle principů OOP je ale určitě správné přistupovat k atributům přes metody. Jestliže máte obavu ze zpomalení programu voláním, můžete metody deklarovat jako `inline`.

Konstruktory v této třídě jsem napsal tak, aby bylo jasné, že slouží k inicializaci instance. Jsou napsány sice jasně a přehledně, ale ne nejlépe. Všechny konstruktory, které jsem zde napsal, lze napsat efektivněji. Je třeba vzít na vědomí, že před spuštěním samotného těla konstruktoru dojde k inicializaci (volání konstruktorů) všech členských dat třídy. Například v konstruktoru `Zlomek(int citatel, int jmenovatel)` dojde nejprve k inicializaci položek `Citatel, Jmenovatel` a poté k provedení těla konstruktoru, kde tyto dvě položky znovu změním. V tomto konkrétním případě to snad ani nevadí, protože `int` žádné konstruktory nemá. Kdyby se ale jednalo o instance nějaké třídy, zjistili by jste, že je pro položky nejprve zavolán konstruktor bez parametrů a poté přiřazena nějaká hodnota operátorem =. Obojí lze efektivněji provést v jednom kroku. Za název konstruktoru, před jeho tělo umístíme za dvojtečkou seznam položek s inicializačními hodnotami v závorce. Zmiňovaný konstruktor lze přepsat takto:

```
Zlomek::Zlomek(int citatel, int jmenovatel)
:Jmenovatel(jmenovatel),Citatel(citatel)
{
  /* Zde provedu nějakou činnost, nebo nechám tělo prázdné. */
}
```

Při vytváření instance třídy již mohu použít její metody, i když vlastně instance je vytvořená až po dokončení konstruktoru.
Někdy by se mohlo zdát, že je zbytečné vytvářet konstruktor bez parametrů. Bezparametrický konstruktor je ale nutný například při vytváření polí objektů a podobně. Vše ukážu dále.
Ke konstruktorům se ještě vrátím.

Destruktor

Destruktor, jak již asi sám jeho název napovídá, slouží k likvidaci objektů. Destruktor (stejně jako konstruktor) je metoda, která je zavolána na instanci v momentě, kdy je instance likvidována. Destruktor se jmenuje stejně jako třída, jen před názvem třídy je znak `~` . Destruktor nesmí mít návratovou hodnotu a žádné parametry. V destruktoru by se měly uvolnit všechny zdroje, se kterými instance pracovala. Například uzavřít datové soubory, uvolnit případnou alokovanou paměť a podobně. Nemá-li třída destruktor, překladač vytvoří implicitní destruktor. Ve třídě `Zlomek` naprosto stačí implicitní destruktor. Já ale jako příklad uvedu. Připište prosím do třídy `Zlomek` mezi veřejné prvky deklaraci: `~Zlomek();` a dále připište metodu:

```
Zlomek::~Zlomek()
{
    cout << "Loučí se s vámi instance třídy zlomek. " << endl << dejCitatel() << '/' << dejJmenovatel() << endl;
    /* Jinak tu není co dělat */
}
```

Protože požívám `cout` nezapomeňte na začátek zdrojového textu napsat: `#include<iostream.h>`

Jak vytvářet instance

Na instance lze v C++ někdy pohlížet jako na proměnné. Instance stejně jako proměnné mohou být "statické", nebo "dynamické". Statická instance má stejně jako proměnná platnost (viditelnost) v aktuálním bloku, kde byla vytvořena. V momentě, kdy její platnost končí, je automaticky zlikvidována zavoláním svého destruktoru. Příklad: (Připište jej do souboru, kde je definována třída `Zlomek`

```
Zlomek Globalni(2,1);      /* Globalni je vytvořeno konstruktorem Zlomek(int jmenovatel, int citatel); Tento konstruktor se provede dříve, než funkce main.*/
void main(void)
{
    Zlomek a,b(1),c(1,2);
   /* a je vytvořeno bezparametrickým konstruktorem
     b je vytvořeno konstruktorem Zlomek(int cislo);
     b je vytvořeno konstruktorem Zlomek(int jmenovatel, int citatel);*/
    Zlomek pole[3]; /* Pole objektů. Každý prvek v poli je vytvořen bezparametrickým konstruktorem. */
    pole[2].nastavCitatel(10);
    pole[2].nastavJmenovatel(2);
    cout << a.hodnota() << " " << b.hodnota() << " " << c.hodnota() << " " << Globalni.hodnota() << endl;
    for (int p = 0; p<3; p++)
    {
       cout << "Prvek " << p << " je " << pole[p].hodnota() << endl;
    }
  /* Nyní končí viditelnost instancí a,b,c a všech prvků pole . Automaticky budou vyvolány jejich destruktory. */
}  /* Nyní bude zavolán destruktor instance Globalni. */
```

V programu jsou nejprve zavolány konstruktory globálních instancí, poté je až spuštěna funkce `main` , ve které jsou volány konstruktory a destruktory na lokální instance. Po skončení `main` jsou likvidovány globální instance.

Další možností je dynamické alokace paměti na haldě, a vytvoření instance. K takové instanci přistupujeme přes ukazatel, který na ní "ukazuje". Ukazatel deklarujeme jako v jazyce C. Pro alokaci paměti existuje nový operátor `new` . V takovém případě instance "přežije" konec bloku, a je potřeba ji zrušit operátorem `delete` . Stejně jako je tomu v C s dynamickými prom. Obdoba `new` a `delete` je v C `malloc` a `free` . Není správné ale používat funkce `malloc` a `free` , protože jen alokují paměť. narozdíl od toho `new` navíc zavolá konstruktor a `delete` zavolá destruktor. Jako příklad uvedu novou funkci `main` :

```
void main(void)
{
    Zlomek *a, *b, *c, *pole;/* Ukazatelé na instance třídy Zlomek.*/
   /* Žádné konstruktory se nevolají.*/
```

```
    a = new Zlomek; /* Instance je vytvořena bezparametrickým konstruktorem.*/
    b = new Zlomek(1,2); /* Je zavolán konstruktor Zlomek(int jmenovatel, int citatel); */
    c = b; /* c ukazuje na stejnou instanci jako b! */
    pole = new Zlomek[3]; /* Takto se dynamicky vytváří pole instancí.*/
    cout << a->hodnota() << " " << b->hodnota() << c->hodnota() << endl;
    for (int p = 0; p<3; p++)
    {
      cout << "Prvek " << p << " je " << pole[p].hodnota() << endl;
    }
   /* Nyní instance likviduji. Neprovede se to automaticky.*/
   delete a;
   delete b;
   delete[] pole; /* Takto se uvolňuje pole. */
   /* NENÍ správné napsat delete c; Tato instance již byla uvolněná příkazem delete b; */
}
```

V těchto příkladech jsem vždy přepokládal, že paměť pro instance je vždy k dispozici. To je samozřejmě velmi naivní. Operátor `new` v případě, že neuspěje s alokací paměti, vrací `NULL` , nebo vypustí tak zvanou vyjímku. Vyjímkám věnuji jeden z mých dalších článků.
Článek je již dost dlouhý a já jsem se bohužel ještě nezmínil o jednom specielním a velmi důležitém konstruktoru, který se nazývá kopírovací konstruktor. Ve třídě `Zlomek` nebyl potřeba. Absence kopírovacího konstruktoru je velmi častou chybou a velmi častou příčinou nevysvětlitelných pádů programů. Někdy trvá dost dlouho, než si programátor uvědomí, že příčina nepochopitelných pádů programu je nepřítomnost kopírovacího konstruktoru. Proto vám doporučuji si přečíst můj další článek, který o kopírovacích konstruktorech bude.