Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 07.03. 2002
Url: http://www.builder.cz/art/cpp/automatpointer.html

Automatické ukazatele v C++

Dnes si povíme něco o tak zvaném automatickém ukazateli. Automatický ukazatel je šablona třídy, která "zapouzdřuje" ukazatel na libovolný objekt. Objekt jednou vytvoříme, zapouzdříme jej v automatickém ukazateli, a již se nemusíme starat o jeho dealokaci. Automatický ukazatel jej odstraní (pomocí `delete`) automaticky sám ve svém destruktoru.

Automatický ukazatel je šablona z STL. Je deklarována v hlavičkovém souboru `memory` v prostoru jmen `std`. Šablona se jmenuje `auto_ptr`. Má přetížené operátory * a -> tak, aby nám práce s šablonou `auto_ptr` co nejvíce připomínala práci s obyčejným ukazatelem. Parametrem šablony je typ, na který se bude odkazovat zapouzdřený ukazatel. Pozor parametrem tedy je samotný typ objektu, nikoliv typ ukazatel na objekt. Metody a přetížené operátory (Typ je zde parametr šablony):

- `auto_ptr(Typ *pointer = NULL);` - konstruktor. Jeho parametr je ukazatel, který chceme zapouzdřit. Nezadáme-li ukazatel, jeho implicitní hodnota je NULL.
- `auto_ptr(auto_ptr<Typ> &original);` - kopírovací konstruktor. Vytvoří kopii automatického ukazatele. Chování tohoto konstruktoru je asi jiné než by jsme mohli na první pohled čekat. Originální objekt totiž "předá" zapouzdřený ukazatel a sám si "svůj" zapouzdřený ukazatel nastaví na NULL. Obdobně se chová také operátor =. V jednom okamžiku tedy existuje pouze jeden zapouzdřený ukazatel, který si automatické ukazatele nekopírují, ale "předávají".
- `~auto_ptr();` - destruktor. Zlikviduje automatický ukazatel, pokud je v automatickém ukazateli zapouzdřen ukazatel na jiný objekt, dojde k likvidaci tohoto objektu pomocí `delete`.
- `operator=(auto_ptr<Typ> &original);` - obdobné chování jako kopírovací konstruktor. Originál předá, nikoliv zkopíruje zapouzdřený ukazatel.
- `Typ &operator* () const;` - vrátí referenci na objekt, na který se odkazuje zapouzdřený ukazatel.
- `Typ *get() const;` - vrátí zapouzdřený ukazatel.
- `X *operator-> () const;` - umožní přístup k prvkům objektu, na který se odkazuje zapouzdřený ukazatel. V podstatě také vrací zapouzdřený ukazatel jako `get()`, ale v případě, že budeme přistupovat k prvkům objektu, nabízí přehlednější zápis.
- `void reset(Typ *pointer = NULL);` - nastaví zapouzdření nového ukazatele. Objekt, na který se odkazoval starý ukazatel bude zlikvidován operátorem `delete`. Místo něj bude nyní nastaven nový zadaný ukazatel. Metoda `reset` vlastně přenastaví zapouzdřený ukazatel. Implicitní hodnota nového ukazatele je `NULL`.
- `X *release();` - vrátí zapouzdřený ukazatel. Zapouzdřený ukazatel bude nyní `NULL`. Objekt, na který se odkazoval "starý" ukazatel, který je vrácen, nezlikviduje. Metoda v podstatě slouží k "vysvobození" zapouzdřeného ukazatele. Použijeme ji tehdy, jestliže chceme, aby se automatický ukazatel přestal starat o náš normální ukazatel.

Vše si uvedeme na jednoduchém příkladě. Vytvoříme instanci naší pokusné třídy. Ukazatel na ni předáme automatickému ukazateli jménem `a`. Ukážeme si jak zavolat metodu pomocí operátoru ->. Potom vytvoříme automatický ukazatel `b`, který bude "kopie" `a`. Až skončí aktuální blok, automatický ukazatel `b` bude zlikvidován. S ním bude také zlikvidován objekt, na který se odkazoval zapouzdřený ukazatel. Když jsme vytvářeli `b`, přepsali jsme zapouzdřený ukazatel v `a` na `NULL`. O všem se přesvědčíte, jestliže program spustíte.

```
#include<memory>
#include<iostream>
using namespace std;

class Trida
{
    public:
       Trida() { cout << "Konstruktor" << endl; }
       ~Trida() { cout << "Destruktor" << endl; }
       void metoda() { cout << "Metoda" << endl; }
};

int main()
{
    auto_ptr<Trida> a(new Trida);
    a->metoda();
    {
        auto_ptr<Trida> b(a); // Od teď je a.get() == NULL  !
        (*b).metoda();
        b.get()->metoda();
        cout << "Objekt b konci svou platnost " << b.get() << endl;
    }
    cout << "Objekt b uz neexistuje" << endl;
    /* Nyní uvidíte, že a skutečně zapouzdřuje NULL. */
    cout << "Objekt a konci svou platnost " << a.get() << endl;
    return 0;
}
```

Já osobně automatickými ukazateli nejsem příliš nadšen, a nepoužívám je. Původně jsem se o nich nechtěl ani zmiňovat. Je ale dobré vědět, že existují. Hlavně jsem tento článek napsal jako úvod k mému příštímu článku.

Na automatických ukazatelích mi nejvíce vadí, že při jejich kopírování si zapouzdřený ukazatel pouze předávají. Daleko lepší by bylo, kdyby ukazatele kopírovali. Objekt by byl pouze jeden, na který by se odkazovalo více ukazatelů. Každý z těchto ukazatelů by byl zapouzdřen v automatickém ukazateli. Bylo by to krásné do té doby, než by jeden z automatických ukazatelů byl zlikvidován. Ve svém destruktoru by totiž dealokoval onen objekt. Ostatní ukazatele by se odkazovali do nealokované paměti. Kdyby ale někde existovala informace o tom, kolik takových ukazatelů se na objekt odkazuje, mohl by být objekt zničen až v době, kdy na něj neexistuje ukazatel. Tím by jsme udělali něco skvělého. Vytvořili by jsme objekt, se kterým by jsme pracovali pomocí našich "inteligentních" ukazatelů. O likvidaci tohoto objektu by jsme se již nestarali. Objekt by byl sám zlikvidován v momentě, kdy

by na něj již neexistoval žádný odkaz. Myslíte, že něco takového v C++ nejde? V příštím článku něco takového uděláme.

------ http://www.builder.cz ------