Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 17.09. 2001
Url: http://www.builder.cz/art/cpp/sablcpp.html

Šablony datových typů v C++

V minulém článku jsme si pověděli něco o šablonách funkcí v C++. Dnes se zaměříme na šablony datových typů. V tomto článku se budu výhradně věnovat šablonám tříd, ale vše co zde uvedu je použitelné také na struktury a unie. Uveďme si příklad jednoduché šablony třídy.

```
template<class Typ> class Obal
{
        private:
                Typ Promenna;
        public:
                Obal();
                Obal(Typ p);
                Obal(Obal &druhy);
                inline Typ dejPromennou();
                inline void nastavPromennou(Typ p);
};
```

Šablona třídy `Obal` je jakýsi "objektový obal" pro jakoukoliv proměnnou. Možná by měl být ještě přetížen operátor `=`, případně i operátory `==` a `!=`, o to ale teď nejde. Vytvoříme instanci této šablony pro nějaký konkrétní typ, instance šablony třídy je třída. Šablona třídy by se dala považovat za něco jako metatřídu. Tedy třídu třídy. Není to ale přesné přirovnání, jazyk C++ metatřídy nemá. Problém je v tom, že šablony jsou v C++ instanciovány již v době překladu, tedy všechny parametry pro jakoukoliv šablonu (i šablonu funkce, i šablonu datového typu) musejí být známy již v době překladu. Nejprve si ukažme jak implementovat metody této šablony.

```
template<class Typ> Obal<Typ>::Obal() : Promenna()
{}

template<class Typ> Obal<Typ>::Obal(Typ p) : Promenna(p)
{}

template<class Typ> Obal<Typ>::Obal(Obal &druhy) : Promenna(druhy.dejPromennou())
{}

template<class Typ> Typ Obal<Typ>::dejPromennou()
{
```

```
        return Promenna;
}

template<class Typ> void Obal<Typ>::nastavPromennou(Typ p)
{
        Promenna = p;
}
```

Metody šablony třídy se implementuji obdobně jako metody třídy. Jen je třeba si uvědomit, že i metoda šablony třídy je vlastně šablona. Proto musíme uvést klíčové slovo `template` i s parametry šablony. Tímto máme vytvořenou šablonu třídy obal. Je dobré si také všimnout, že o parametru šablony `Typ` v době kdy šablonu píšu vlastně nic nevím. Přesto předpokládám mnoho vlastností, které tento typ bude mít. Například musí mít bezparametrický konstruktor, kopírovací konstruktor a operátor `=`. Pokud si dobře prohlédne těla metod, zjistíte, že je používám. Samozřejmě je nemusím přetěžovat, jestli k tomu není důvod, ale je nutné, aby každý kdo bude šablonu používat tento fakt věděl. Nyní vytvoříme jednoduchý program, kde naši šablonu použijeme.

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

int main(void)
{
        Obal<int> a(10);
        cout << a.dejPromennou() << endl;
        a.nastavPromennou(3);
        cout << a.dejPromennou() << endl;
        Obal<char> b;
        b.nastavPromennou('a');
        cout << b.dejPromennou() << endl;
        Obal<char> c(b);
        cout << c.dejPromennou() << endl;
        return 0;
}
```

Třída `Obal<int>` je instance šablony třídy `Obal`. Objekt `a` je instance třídy `Obal<int>`. Někdy je dobré pro přehlednost použít klíčové slovo `typedef`. Například `typedef Obal<int> INT`. Zde jsme vytvořili třídu INT, která je instancí šablony třídy `Obal<int>`. Dále lze INT použít normálním způsobem: `INT a, *b = new INT(2);`. Zde jsou objekty a, *b (objekt, na který ukazuje ukazatel b) instance třídy INT.

Co kdyby byl parametrem šablony Obal datový typ, který je instanci nějaké šablony? To není v praxi nic neobvyklého. Vytvoříme "obal obalu int": `Obal<Obal<int> > DvojObal;` . Důležité je, že mezi znaky > je mezera. Kdybychom napsali `>>`, překladač by tento token považoval za operátor binárního posunu, a nahlásil by chybu v syntaxi.

Specializace šablon tříd

Šablony tříd nelze přetěžovat tak, jako šablony funkcí. Lze ale vytvářet specializace šablon tříd. Naopak šablony funkcí nelze specializovat. Specializace šablony třídy může být částečná, nebo úplná. Nevím přesně proč, ale úplné specializaci se někdy také říká explicitní specializace. Specializace spočívá v tom, že vytvořím speciální případ šablony pro nějakou podmnožinu parametrů - částečná specializace, nebo specielní případ šablony pro konkrétní hodnoty parametrů - úplná specializace.

Částečná specializace

Pokusíme se pro ukázku částečně specializovat naší šablonu `Obal`. V případě, že parametrem bude ukazatel, vytvoříme kopii objektu, na který se ukazatel odkazuje. Nejprve je nutné uvést obecnou šablonu, která je napsána výše. Poté se uvede částečná specializace šablony:

```
template<class Typ> class Obal<Typ*>
{
        private:
                Typ *Promenna;
        public:
                Obal() : Promenna(NULL) {}
                Obal(Typ *p) : Promenna(new Typ(*p)) {}
                Obal(Obal &druhy) : Promenna (new Typ(*drhy.dejPromennou())) {}
                ~Obal() { delete Promenna; }
                Typ *dejPromennou() { return Promenna; }
                void nastavPromennou(Typ *p) { Promenna = new Typ(*p); }
};
```

Specializace šablon nemá nic společného s dědičností tříd. Specializovaná šablona je úplně nový a nezávislý typ. Všechny metody i atributy je nutné znovu opsat, nic se nedědí po obecné šabloně. Všimněte si, že v prvním řádku specializované šablony uvádím za slovem "Obal" onu podmnožinu parametrů, pro které se má použít tato specializace šablony. Použití částečné specializace šablony si ukážeme v dalším odstavci společně s použitím úplné specializace.

Úplná specializace

Nyní vytvoříme úplnou specializaci naší šablony `Obal`. Pro případ, že by parametrem šablony byl ukazatel na `char`, jednalo by se o ukazatel na první znak klasického řetězce z C, který je ukončen nulou. V tomto případě si budeme chtít mít vytvořenou kopii tohoto řetězce v našem obalu. Nezapomeňte, že nejprve musí být deklarována obecná šablona. Potom částečná specializace, budeme-li chtít používat částečně specializovanou šablonu. A na konec nejkonkrétnější případ:

```
#include<string.h>
// Úplná specializace

template<> class Obal<char*>
{
        private:
                char *Promenna;
        public:
                Obal();
                Obal(char *r);
                Obal(Obal &druhy);
                ~Obal();
                inline char *dejPromennou();
                inline void nastavPromennou(char *p);
};

Obal<char*>::Obal() : Promenna(NULL)
{}

Obal<char*>::Obal(char *r) : Promenna(strdup(r))
{}

Obal<char*>::Obal(Obal &druhy) : Promenna(strdup(druhy.dejPromennou()))
{}

Obal<char*>::~Obal()
{
        delete[] Promenna;
}

char *Obal<char*>::dejPromennou()
{
        return Promenna;
}

void Obal<char*>::nastavPromennou(char *p)
{
        delete[] Promenna;
        Promenna = strdup(p);
}
```

Opět jsme z obecné šablony nic nezdědili, museli jsme všechny metody i atributy vytvořit znova. Uvedeme si příklad použití všech tří šablon:

```
int main(void)
```

```
{
        int cislo = 5;
        Obal<int> a(10);  // Použije se obecná šablona
        cout << a.dejPromennou() << endl;
        a.nastavPromennou(3);
        cout << a.dejPromennou() << endl;
        Obal<int*> b(&cislo); // Použije se částečná specializace pro ukazatele.
        cout << b.dejPromennou() << endl;
        Obal<char*> cc;   // Použije se konkrétní specializace pro typ char*
        cc.nastavPromennou("Ahoj");
        cout << cc.dejPromennou() << endl;
        return 0;
}
```

Na závěr bych chtěl jen upozornit na fakt, který nemusí být zřejmý. Jak jsem již vysvětloval v minulém článku pojem šablony, upozornil jsem, že šablona existuje pouze v době překladu. Za běhu programu (dokonce i v době "linkování" (spojování) linkerem) existují jen instance šablony. Proto musí být celá šablona přímo deklarovaná v kompilovaném zdrojovém textu, nebo v hlavičkovém souboru, který bude vložen pomocí `#include`. Šablona je celá považována za deklaraci, i když třeba někomu šablona funkce či metody připadá jako definice (implementace). U šablony třídy budou při instanciaci vytvořeny jen ty metody, které budou použity. Ostatní nechá překladač bez povšimnutí, i kdyby obsahovaly chyby. Jinou možností je vytvořit šablonu explicitně tak, jak jsem to popsal v minulém článku.

Článek je již dost dlouhý, a bohužel jsme se nedostali ke klíčovému slovu `typename` (V minulém článku jsem to sliboval.), které souvisí s vnořenými typy. Podíváme se na něj v příštím článku. Stejně tak se v příštím článku podíváme na vnořené šablony.

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