Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 07.09. 2001
Url: http://www.builder.cz/art/cpp/sablfci.html

Šablony funkcí v C++

Co jsou to šablony funkcí a proč je používat? Šablona funkce je konstrukce, která umožňuje vytvořit funkci na základě parametrů. Proto se také vytváření a používání šablon někdy nazývá parametrické programování, a v souvislosti s šablonami se hovoří o parametrismu. Šablony nemusí být jen šablony funkcí, ale také šablony datových typů (struktur - `struct`, tříd - `class`, unií - `union`). O těch si povíme něco příště, dnešní článek bude věnován jen šablonám funkcí. Význam šablon se často a nejlépe demonstruje na příkladu funkce swap. Swap je funkce, která vymění ("přehodí") obsah dvou proměnných. Představme si situaci, kdy chceme mít možnost vyměnit hodnoty proměnných mnoha typů, nebo i instancí mnoha tříd. Bez použití šablon bychom museli napsat mnoho funkcí swap, které by se od sebe lišili jen typem parametrů. Lepší řešení je vytvořit jednu šablonu funkce swap, podle které se dle potřeby vytvoří požadovaná funkce na základě parametru, kterým bude typ. Syntaxe šablony funkce je:

```
template< parametry šablony > deklarace
```

Šablona funkce swap:

```
template<class T> void swap(T &a, T &b)
{
        T c(a);
        a = b;
        b = c;
}
```

Parametrem šablony je typ T. Klíčové slovo `class` značí, že parametrem bude datový typ. Nemusí se jednat pouze o třídu, ale o jakýkoliv datový typ, i primitivní datový typ. Dále následuje funkce, swap. Tato funkce bude mít dva parametry, jenž budou referencí na typ T. V těle funkce vytvoříme dočasnou proměnnou c, která je typu T. Dále provedeme záměnu hodnot proměnných pomocí třetí proměnné. Konkrétní funkci pro konkrétní typ (instanci šablony) vytvoříme dosazením typu za parametr T. Překladač ještě v době překladu vytvoří funkcí swap, která bude mít tělo jako šablona, ale za T bude dosazen typ int. Příklad instanciace a použití šablony swap:

```
#include<iostream>

using namespace std;
```

```
int main()
{
  int a(0),b(100);
  swap<int>(a,b);
  cout << a << " " << b << endl;
  return 0;
}
```

Podívejme se na řádek `swap<int>(a,b);`. V závorkách < > je parametr šablony. v závorkách ( ) jsou parametry funkce. Je-li parametr šablony zřejmý z volání funkce, nemusí se parametry šablony vůbec uvádět. V našem případě je z typu proměnných `a` a `b` jasné, že parametrem šablony je `int`, proto stačí dokonce napsat jen `swap(a,b)`. Naše šablona může být použitá pro jakýkoliv typ. Jen je dobrá si všimnout, že ve funkci swap bude vytvořena kopie proměnné (objektu) `a`, a že bude použít operátor `=` pro přiřazení hodnot. Proto by typ, který bude parametrem naší šablony měl mít kopírovací konstruktor, pokud to bude nutné (viz můj článek Kopírovací konstruktor v C++), a také, pokud to bude nutné, přetížený operátor `=` (viz můj článek Přetěžování operátorů). Ukažme si nyní využití šablony funkce swap. Spolu s deklarací šablony swap vytvořme program:

```
#include<iostream>
#include<string>

using namespace std;

class Trida
{
        private:
                int atribut;
        public:
                int dejAtribut() const {return atribut;}
                void nastavAtribut(int atribut) { this->atribut = atribut;}
                /* Pro tuto třídu není potřeba
               ani kopírovací konstruktor, ani operátor = */
};

int main()
{
        int a(0),b(100);
        cout << a << " " << b << endl;
        swap(a,b);
        cout << a << " " << b << endl;
        char x('x'), y('Y');
        cout << x << " " << y << endl;
        swap(x,y);
        cout << x << " " << y << endl;
        string s1("Ahoj "), s2("svete");
        cout << s1 << s2 << endl;
```

```
        swap(s1,s2);
        cout << s1 << s2 << endl;
        Trida instance1, instance2;
        instance1.nastavAtribut(100);
        instance2.nastavAtribut(-100);
        cout << instance1.dejAtribut() << " " << instance2.dejAtribut() << endl;
        swap(instance1,instance2);
        cout << instance1.dejAtribut() << " " << instance2.dejAtribut() << endl;
        return 0;
}
```

Jak je vidět naše šablona funkce swap je použitelná pro mnoho typů. Je nutné upozornit, že v přeloženém programu šablona vlastně neexistuje. Existují v něm pouze instance šablony, které se vytvoří v době překladu. Nemusíme se tedy obávat nějakého zpomalení programu používáním šablon. Naopak šablony se mnohdy používají pro optimalizaci výsledného programu, protože mnoho činností přebírá překladač v době překladu, čímž může být výsledný program rychlejší. Šablony mohou někomu připomínat makra s parametry. Dalo by se skutečně říci, že šablony jsou vlastně lepší a "inteligentnější" makra.

Přetěžování šablon funkcí

Šablony funkce lze přetěžovat stejně tak jako samotné funkce. Pokusíme se přetížit naší šablonu funkce swap. Další šablona bude šablona funkce zaměňující hodnoty prvků dvou stejně dlouhých polí:

```
template<class T, int N> void swap(T *a, T *b)
{
        T temp;
        for (register int p = 0; p < N; p++, a++, b++)
        {
                temp = *a;
                *a = *b;
                *b = temp;
        }
}
```

Dalším parametrem šablony swap je celé číslo. Za tento parametr lze dosadit jen konstantu, nebo výraz, který lze vyhodnotit v době překladu. Společně s předchozí šablonou swap lze obě šablony použít například takto:

```
int main()
{
        int pole1[2] = {1, 2}, pole2[2] = {3,4};
        swap<int,2>(pole1,pole2);
```

```
        for(a = 0; a < 2; a++)
        {
                cout << pole1[a] << " " << pole2[a] << endl;
        }
        swap(pole1[0],pole2[0]);
        for(a = 0; a < 2; a++)
        {
                cout << pole1[a] << " " << pole2[a] << endl;
        }
        return 0;
}
```

Priority volání

Vytvořme si do třetice obyčejnou funkci (Ne šablonu funkce!), která jako své parametry dostane ukazatele na první znak řetězce zakončeného znakem '\0'. Funkce přehodí samotné řetězce, ne pouze ukazatele na ně. Takto napsaná funkce je použitelné pouze pro řetězce stejné délky.

```
void swap(char *a, char *b)
{
        register char temp;
        while ((*a != '\0') && (*b != '\0'))
        {
                temp = *a;
                *a++ = *b;
                *b++ = temp;
        }
        cout << "Není šablona" << endl;
       /* Výpis, abychom věděli, že funkce byla zavolána. */
}
```

Nyní společně se šablonami swap vytvořme jednoduchou funkci `main`:

```
#include <string.h>
#include <iostream>

using namespace std;

int main()
{
        char *r1 = strdup("svete"), *r2 = strdup("Ahoj ");
        int i1(1), i2(2);
        swap(r1,r2); /* Bude volána void swap(char *a, char *b) */
```

```
        cout << r1 << r2 << endl;
        swap(i1,i2); /* Bude volána template<int> void swap(int &a, int &b) */
        cout << i1 << i2 << endl;
        return 0;
}
```

Po spuštění programu se můžeme podle výpisu přesvědčit, že funkce swap budou skutečně volány tak, jak jsem uvedl v poznámkách. Pro dva řetězce nebude vytvořena instance šablony, a přeloženo její volání, ale bude přeloženo volání funkce swap(char *a, char *b). Priorita volání funkcí by šla shrnout do následujících pravidel:

1) Existuje-li pro daný typ parametrů funkce, přeloží se volání této funkce.
2) Neexistuje-li pro daný typ parametrů funkce, překladač zjistí, jestli nemůže vytvořit instanci šablony.
3) Není-li k dispozici ani funkce, ani šablona, ze které lze vytvořit potřebnou instanci, jedná se o chybu.

Explicitní a implicitní vytváření instancí šablon.

Existují dva způsoby jak vytvářet instance šablon. První je implicitní vytvoření instance. Znamená to, že překladač vytvoří instanci a přeloží ji jen tehdy, jestliže ji potřebuje. Všechny instance šablony swap v předchozích příkladech jsou vytvořeny implicitně. Další možnost jak vytvořit instanci šablony explicitně. Budeme-li chtít, aby překladač vytvořil instanci šablony swap pro parametr `float`, i když nebude v daném modulu požadována, napíšeme:

```
template void swap<float>(float &a, float &b);
```

Tolik tedy k šablonám funkcí. Příště se podíváme na šablony datových typů. Na jednoduché příkladu si vytvoříme šablonu třídy. Instance šablony třídy je třída. Také se podíváme na další, pro nás nové, klíčové slovo `typename`.