Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 10.12. 2001
Url: http://www.builder.cz/art/cpp/stdfobj.html

Standardní funkční objekty v C++

V minulém článku "Funkční objekty v C++" jsme se seznámili s pojmem funkční objekt. Dnes si ukážeme jaké funkční objekty (třídy funkčních objektů) obsahuje standardní knihovna STL. Jestliže chceme pracovat se standardními algoritmy, je dobré znát standardní funkční objekty, nebo si umět vytvořit vlastní. Standardním algoritmům v STL se budeme věnovat v příštím článku.

Šablony unary_function a binary_function

V STL existují třídy unárních a binárních funkčních objektů. Unární funkční objekt je funkční objekt, který má jen jeden parametr operátoru (). Binární funkční objekt je funkční objekt, který má dva parametry operátoru (). Šablony `unary_function` a `binary_function` jsou šablony struktur. Obě šablony slouží jako nadtřídy (Jedná se sice o struktury, ale budu používat raději pojem třída. Slovo "nadstruktura" se mi zdá nějaké divné.) pro třídy funkčních objektů. Nemají žádnou metodu, ani atribut. Obsahují pouze definici vnitřních typů. Neobsahují dokonce ani operátor (), proto se vlastně nejedná ani o třídy funkčních objektů. Každá standardní třída funkčních objektů je šablonou třídy (struktury), a dědí z jedné z těchto dvou šablon tříd (struktur). Každá standardní třída funkčních objektů rozšíří tyto šablony minimálně o operátor (). Šablona `unary_function` má dva parametry, první udává typ parametru, druhý udává typ návratové hodnoty operátoru(). Šablona `binary_function` má 3 parametry. První dva udávají typy prvního a druhého parametru operátoru(). Poslední parametr udává typ návratové hodnoty parametru (). Programátor, který používá funkční objekty s šablonami `binary_function` a `unary_function` nepříjde nijak do styku. Bude pracovat s šablonami odvozenými. Přesto si myslím, že je dobré o `binary_function` a `unary_function` vědět. Rovněž je dobré použít je jako nadtřídu při psaní vlastních tříd funkčních objektů. (Což jsem ve svém minulém článku neudělal, protože jsme takové šablony ještě neznali.)

Třídy unárních funkčních objektů

Šablony tříd unárních funkčních objektů z STL jsou potomky šablony třídy (struktury) `unary_function`. Seznam některých tříd unárních funkčních objektů:

Název šablony	Činnost
negate	Šablona má jeden parametr. Udává typ návratové hodnoty, i typ parametru operátoru (). Funkční objekt neguje svůj parametr. Je-li

	parametrem operátoru () proměnná (objekt) x, výsledek bude -x.
logical_not	Šablona má jeden parametr. Udává typ parametru operátoru (). Návratová hodnota operátoru () je vždy typu bool. Funkční objekt neguje svůj parametr. Je-li parametrem operátoru () proměnná (objekt) x, výsledek bude !x.

Třídy binárních funkčních objektů

Šablony tříd binárních funkčních objektů z STL jsou potomky šablony třídy (struktury) `binary_function`. Seznam některých tříd binárních funkčních objektů:

Název šablony	Činnost
divides	Šablona má 1 parametr. Parametr udává typ obou parametrů a návratové hodnoty operátoru (). Operátor () vydělí své dva parametry.
equalto	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě, že první parametr == druhý parametr.
greater	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě, že první parametr > druhý parametr.
greater_equal	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě, že první parametr >= druhý parametr.
less	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě, že první parametr < druhý parametr.
less_equal	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě, že první parametr <= druhý parametr.
logical_and	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Operátor () vrací první parametr && druhý parametr.
logical_or	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Operátor () vrací první parametr || druhý parametr.
minus	Šablona má 1 parametr. Parametr udává typ obou parametrů a návratové hodnoty operátoru (). Operátor () vrátí rozdíl svých dvou parametrů.
modulus	Šablona má 1 parametr. Parametr udává typ obou parametrů a návratové hodnoty operátoru (). Operátor () vrátí zbytek po celočíselném dělení.
	Šablona má 1 parametr. Parametr udává typ obou parametrů operátoru (). Operátor () vrací proměnnou typu bool. Vrací true v případě,

not_equal	že první parametr != druhý parametr.
plus	Šablona má 1 parametr. Parametr udává typ obou parametrů a návratové hodnoty operátoru (). Operátor () sečte své dva parametry.
times	Šablona má 1 parametr. Parametr udává typ obou parametrů a návratové hodnoty operátoru (). Operátor () vynásobí své dva parametry.

Všechny tyto třídy jsou deklarovány v hlavičkovém souboru `function` v prostoru jmen `std`. Omezení na parametr šablony vyplývá z její činnosti. Například šablona `plus` může mít jako svůj parametr typ, který má definován operátor + (implicitně, nebo definován `operator+`).

Nyní si ukážeme jednoduchý příklad. Ve svých článcích Množina a multimnožina v C++ a Asociativní pole v C++ jsem uvedl, že tyto kontejnery mají poslední, nepovinný parametr, kterým je funkční objekt. Implicitní hodnota posledního parametru je `less<T>`, kde `T` je první parametr kontejneru.

```
#include <map>
#include <string>
#include <iostream>
#include <functional>

using namespace std;
int main()
{
    map<string,string,greater<string> > pole;
    pole.insert(make_pair(string("Klic1"),string("Hodnota1")));
    pole.insert(make_pair(string("Klic2"),string("Hodnota2")));
    cout << pole[string("Klic1")] << endl;
    /* Klíče jsou seřazeny pomocí relačního operátoru >,
            nikoliv < jako v předchozích článcích.*/
    return 0;
}
```

Tento příklad zatím asi nikoho nepřesvědčil o důležitosti funkčních objektů. Jejich význam se ukáže až v mých dalších článcích.

Šablony binder1st a binder2nd

V souvislosti se standardními třídami funkčních objektů je dobré se také seznámit s šablonami `binder1st` a `binder2nd`. Jedná se o šablony tříd s operátorem (), který má 1 parametr. Obě třídy zapouzdřují binární funkční objekt (operátor () má 2 parametry) a jednu hodnotu. Tato hodnota bude daná jako první parametr u šablony `binder1st` nebo jako druhý parametr u šablony `binder2nd`. Zbývající parametr bude předán operátoru () jako parametr. Obě třídy tedy vlastně slouží k převedení binárního funkčního objektu na unární s pevně danou hodnotou jednoho ze svých parametrů. Zní to možná složitě, ale vše si ukážeme na jednoduchém příkladu. K vytvoření instancí tříd `binder1st` a `binder2nd` můžeme použít jednak konstruktory, nebo pomocné funkce jménem `bind1st` a `bind2nd`.

Ve svém minulém článku jsem ve svém posledním příkladu vytvořil šablonu funkce `pocetPlatnych`, která vrací počet prvků v kontejneru vyhovujících nějaké podmínce, která je také předána jako parametr. Podmínka je buď ukazatel na funkci s jedním parametrem, nebo funkční objekt, jehož operátor () má jeden parametr. Vytvořil jsem třídu funkčních objektů, která má jeden atribut. Takto jsem se snažil řešit problém, kdy operátor () má jen jeden parametr, já bych potřeboval aby měl dva. Nyní se podíváme jak napsat stejný příklad jen pomocí standardních šablon z STL. V STL existuje algoritmus `count_if` (Algoritmy podrobně probereme příště.), který dělá stejnou činnost jako moje funkce `pocetPlatnych`. S tím rozdílem, že vrací `void`, protože výsledek je uložen v posledním parametru, který je předáván jako reference. Zatímco moje funkce umožňovala vrátit výsledek pouze jako `int`, funkce `count_if` z STL má jako parametr i typ výsledku, což umožňuje ještě větší abstrakce. Můj poslední příklad z minulého článku napsaný jen pomocí standardních šablon vypadá takto:

```
#include <vector>
#include <iostream>
#include <functional>
#include <algorithm>

using namespace std;
int main(int argc, char **argv)
{
    vector<int> v;
    int vysledek = 0;
    int pole[5] = { 1, 2, 3, 4, 5} ;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
    v.push_back(5);
    v.push_back(-20);
    binder2nd<less<int> > funkcniObjekt = bind2nd(less<int>(),3);
    /* Nyní volání funkcniObjekt(x) je stejne jako volání less<int>(x,3) */
    count_if(v.begin(),v.end(),funkcniObjekt,vysledek);
    cout << vysledek << endl;
    vysledek = 0;
    count_if(pole,&pole[5],funkcniObjekt,vysledek);
    cout << vysledek << endl;
    vysledek = 0;
    count_if(pole,&pole[5],bind2nd(less<int>(), 5) ,vysledek);
    cout << vysledek << endl;
    vysledek = 0;
    count_if(v.begin(),v.end(),bind2nd(less<int>(), 5) ,vysledek);
    cout << vysledek << endl;
    return 0;
}
```

Ve svých příkladech používám vždy kontejnery s primitivními datovými typy, nebo s řetězci. Musím ale zdůraznit, že vše lze použít i na kontejnery

obsahující instance libovolných tříd, které mají definovány příslušné operátory.

Příště se podíváme na standardní algoritmy v STL.