Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 09.10. 2001
Url: http://www.builder.cz/art/cpp/cppstl.html

Datové kontejnery v C++ - Úvod do STL

STL je zkratka slov Standard Template Library. Jedná se o standardní knihovnu šablon jazyka C++. Knihovna by měla být dodávána s každým překladačem C++. V STL můžeme najít mnoho různých a užitečných šablon. Mimo jiné zde jsou šablony datových kontejnerů, iterátory a šablony algoritmů, které pomocí iterátorů pracují s datovými kontejnery. Kompletní knihovnou STL se zabývat nebudeme. V několika článcích si uděláme něco jako přehled podstatných částí STL.

Datové kontejnery v C++

Pod pojmem datový kontejner si představme něco, co má schopnost "zapamatovat si" nějaké data. Data jsou v kontejneru nějakým způsobem uloženy. kontejnery v STL se dělí na posloupnosti a asociativní kontejnery. Posloupnosti jsou logické posloupnosti dat, které ale nemusejí být v paměti fyzicky za sebou. Asociativní kontejnery jsou kontejnery, kde k datům přistupujeme pomocí nějakého klíče. Způsob uložení dat, přístupu k nim je dán typem kontejneru. V knihovně STL jsem jich napočítal celkem 11. Snad jsem na žádný nezapomněl. Přehled datových kontejnerů je v následující tabulce.

Název kontejneru	Typ kontejneru	Hlavičkový soubor	Popis kontejneru
bitset	posloupnost	bitset	Posloupnost bitů pevné délky.
deque	posloupnost	deque	Oboustranná fronta. Prvky lze vkládat, nebo odebírat z obou konců. Sice lze rovněž odebírat, nebo vkládat prvky na libovolné místo ve frontě (kontejner deque to umožňuje), ale tato operace není příliš efektivní.
list	posloupnost	list	Oboustranně zřetězený seznam.
map	asociativní kontejner	map	Asociativní pole. pole, které nemusí být indexováno celočíselným typem, ale čímkoliv. Třeba řetězcem. Pro daný klíč může existovat pouze 1 asociovaná hodnota. Tomuto kontejneru se budeme v budoucnu zabývat v samostatném článku.
multimap	asociativní kontejner	map	Asociativní pole. Pro daný klíč (index) může existovat více asociovaných hodnot. Tomuto kontejneru se budeme v budoucnu zabývat v samostatném článku.
multiset	asociativní	set	Multimnožina. množina, ve které se mohou prvky opakovat. Tomuto kontejneru se budeme věnovat

	kontejner		později v samostatném článku.
priority_queue	posloupnost	queue	Prioritní fronta. Fronta, ve které neplatí pravidlo "první dovnitř, první ven". Prvky, které se do fronty uloží jsou uspořádány podle nějaké relace. Dalo by se říci, že předbíhají ve frontě podle nějaké předem dané priority.
queue	posloupnost	queue	Klasické fronta. platí pravidlo, že prvek, který byl jako první vložen do fronty, z ní bude také první vybrán.
set	asociativní kontejner	set	Množina. Daná hodnota může být v množině obsažena jen jednou. Tomuto kontejneru se budeme věnovat později v samostatném článku.
stack	posloupnost	stack	Klasický zásobník. Platí pravidlo, že prvek, který byl vložen do zásobníku jako poslední bude vybrán jako první.
vector	posloupnost	vector	Obdoba jednorozměrného pole. Tomuto kontejneru se budeme věnovat později v samostatném článku.

Každý kontejner je šablona třídy. Parametrem této šablony je typ prvků, které mají být v kontejneru uloženy. Pole s libovolným intervalem indexování, které jsem předvedl v minulém článku byl vlastně takový pokus o napsání vlastního datového kontejneru. Každý kontejner v STL obsahuje následující veřejné typy. "`Typ`" je zde parametr šablony, tedy typ prvků, které jsou v kontejneru uloženy.

- `value_type`, což je přejmenování (pomocí `typedef`) typu `Typ`.
- `reference`, což je přejmenování (pomocí `typedef`) typu `Typ&`.
- `const_reference`, což je přejmenování typu `const Typ&`.
- `iterator`, což je typ iterátor na prvky daného kontejneru. Iterátorům se budeme věnovat později.
- `const_iterator`
- `size_type`, což je přejmenování typu `size_t`.
- `difference_type`, což je přejmenování typu `ptrdiff_t`.

Dále každý kontejner má přetížené operátory `= == !- < >`, má k dispozici bezparametrický a kopírovací konstruktor. Nemusíme se tedy ničeho bát, a klidně jakkoliv kontejner kopírovat, nebo porovnávat. Dále každý kontejner má následující veřejné metody:

- `begin` - touto metodou se budeme podrobně zabývat až ve článku o iterátorech.
- `empty` - vrací `true`, jestliže je kontejner prázdný.
- `end` - touto metodou se budeme podrobně zabývat až ve článku o iterátorech.
- `max_size` - vrací maximální možnou velikost kontejneru.
- `size` - vrací aktuální velikost kontejneru.
- `swap` - zajistí výměnu prvků s jiným kontejnerem.

Další metody se v různých kontejnerech liší.

U typu, který má být parametrem kontejneru, musíme počítat s tím, že jeho instance budou kopírovány pomocí kopírovacího konstruktoru, případně pomocí operátoru `=`. U některých kontejnerů, mohou být prvky také porovnávány pomocí relačních operátorů. Proto nemohou-li být použity implicitní operátory je nutné je přetížit.

Dále je nutno upozornit, že všechny kontejnery (Nejen kontejnery, v podstatě vše z STL.) jsou deklarovány v prostoru jmen `std`.

Chce-li někdo proniknout do "tajů" kontejnerů, může si podrobně prostudovat hlavičkové soubory, ve kterých jsou deklarovány. Je v nich k dispozici úplný zdrojový text šablon. Rovněž se můžete podívat na můj velice jednoduchý ukázkový kontejner array z minulého článku.

Tolik tedy k přehledu o datových kontejnerech z STL jazyka C++. Některými kontejnery se budeme věnovat v samostatných článcích, dnes si letmo jen pro názornou ukázku ukážeme práci s dalšími kontejnery.

Musím ještě pro úplnost upozornit na tak zvané adaptéry. Některé kontejnery, které jsem zde uvedl, nejsou vlastně "skladiště dat". Jsou to třídy, které zapouzdří nějaký jiný kontejner a přizpůsobí (adoptují - adaptéry) ho jiným požadavkům. Například adaptér je `stack`, který jako svůj druhý parametr šablony vyžaduje nějaký kontejner. Implicitní hodnota tohoto parametru je `deque`. Adaptér kontejneru poznáme tak, že je u něj možnost jako další parametr šablony zadat jiný kontejner.

A nyní již jednoduchý příklad. Vytvoříme zásobník čísel `int`. Nějaké čísla do něj vložíme, a potom zase vybereme.

```
#include <stack>
#include <iostream>
```

```
int main()
{
        std::stack<int> zasobnik;
        for(int i = 0; i < 10; i++)
        {
                zasobnik.push(i); // Metoda push uloží prvek na vrchol zásobníku.
        }
        std::cout << "Počet prvků je " << zasobnik.size() << endl;
        while (!zasobnik.empty())
        {
                std::cout << zasobnik.top() << endl;
                // Metoda top vrátí prvek na vrcholu zásobníku.
                zasobnik.pop();
                // Metoda pop odstraní prvek na vrcholu zásobníku.
        }
        return 0;
}
```

Kdybychom z nějakých důvodů nechtěli mít jako zásobník (`stack`) adoptovánu `deque`, ale například `vector`, potom by řádek definice objektu zásobník vypadal takto:`std::stack<int,std::vector<int> >`.

Dalším příkladem bude `bitset`, tedy posloupnost bitů. Tento kontejner má oproti všem ostatním jednu zvláštnost. Parametrem šablony není typ, ale počet bitů, které budou v kontejneru uloženy. Podívejme se na jednoduchý příklad.

```
#include <bitset>
#include <iostream>

int main()
{
        std::bitset<32> bity;
        for(int i = 0; i < 32; i++)
        {
                bity.set(i, (i % 2));
                /* Metoda set nastaví bit (jeho pořadí je dáno 1.parametrem)
                na hodnotu druhého parametru. My zde máme nastaven každý lichý
                bit na 1, sudý na 0. */
        }
        std::cout << bity.to_string() << endl;
        std::cout << (~bity).to_string() << endl; /* Operátor ~ provádí negaci */
        std::cout << bity.to_ulong() << endl;
        /*Metoda to_ulong vrátí unsigned long, které reprezentují bity v poli. */
        for(int i = 0; i < 32; i++)
        {
                if (bity[i])
```

```
                {   /* Vypíši pořadí bitů s hodnotou 1 */
                        std::cout << i << endl;
                }
        }
        try
        {
                std::cout << bity[1000] << endl;
                /*Pokud indexuji mimo zadanou horni mez indexovaní, bude vyvržena
                vyjimka typu out_of_range. */
        }
        catch (std::out_of_range &vyjimka)
        {
                std::cerr << vyjimka.why() << endl;
        }
        return 0;
}
```

Tolik tedy pro úvod ke kontejnerům. V následujících článcích se budu některými kontejnery zabývat podrobněji. Nebudu zde ale uvádět výčet všech metod kontejnerů. Bylo by to naprosto zbytečné. Pro tyto účely slouží dokumentace ke knihovně. Jedna podle mého názoru pěkná dokumentace je na adrese http://www.sgi.com/tech/stl/stl_index_cat.html. Příště se podrobně budeme zabývat kontejnerem `vector`, což je jednorozměrné pole. V dalších článcích potom asociativním polem, a množinou.