Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 09.11. 2001
Url: http://www.builder.cz/art/cpp/asocspp.html

Asociativní pole v C++

Dnes se podíváme na šablony `map` a `multimap`. Jedná se o asociativní pole. V asociativním poli jsou uloženy hodnoty ve tvaru (klíč,hodnota), kde klíč je vlastně "index" prvku. Klíčem může být libovolný objekt, například `string`.

Šablony `map` i `multimap` jsou kontejnery z knihovny STL (Viz můj článek Datové kontejnery v C++ - Úvod do STL). Jedná se o asociativní kontejnery. Jak ukládané prvky, tak i klíče musejí mít k dispozici:

- Bezparametrický konstruktor
- Musejí být schopny se kopírovat pomocí kopírovacího konstruktoru a operátoru =. Nemohou-li být použity implicitní, musí se operátor = přetížit, a vytvořit kopírovací konstruktor.
- Destruktor

Oba kontejnery mají vnitřní typ `value_type`. Jedná se o šablonu struktury jménem `pair`. Tato šablona je přejmenovaná pomocí `typedef`. Šablona struktury `pair` je struktura parametrizovaná dvěma typy - klíčem a hodnotou. K vytvoření "páru" můžeme použít konstruktor této šablony, nebo pomocí šablony funkce `make_pair`. Identifikátory `pair` a `make_pair` jsou deklarovány v prostoru jmen `std`. Jak v šabloně `map`, tak i v `multimap` jsou vlastně uloženy tyto "páry" (klíč,hodnota). Ke každé hodnotě se přistupuje pomocí klíče. Rozdíl mezi `map` a `multimap` je v tom, že `map` umožňuje mít pro jeden klíč pouze jednu hodnotu, kdežto `multimap` může mít pro jeden klíč více asociovaných hodnot. Oba kontejnery jsou deklarovány v hlavičkovém souboru `map.h` v prostoru jmen `std`.

Šablona map

Šablona `map` má tři parametry. Jedná se o typ klíče, typ prvku, a "něco", co porovnává klíče. Třetí parametr má svou implicitní hodnotu, a my jej zatím nebudeme používat. Kdyby někdo chtěl zajistit, aby klíče nebyly porovnávány pomocí relačního operátoru <, zadal by třetí parametr. Třetí parametr může být funkce (ukazatel na funkci), nebo tak zvaný funkční objekt. Funkčními objekty se chci zabývat v budoucnu v samostatném článku. K pochopení šablony `map` nejsou důležité, proto se jimi dnes nebudeme zabývat. V dnešním článku budeme používat pouze první dva parametry šablon `map` a `multimap`. Šablona `map` má k dispozici konstruktory a destruktor jako všechny ostatní kontejnery (Viz moje předchozí články). K jednotlivým prvkům se přistupuje pomocí klíčů. Operátor indexování [] má jako svůj parametr také hodnotu klíče. Všechny páry v kontejneru jsou vždy seřazeny podle hodnoty svého klíče. Podívejme se nyní na některé metody:

- `begin, empty, size, max_size, swap` - metody společné pro všechny kontejnery z STL. Viz můj článek Datové kontejnery v C++ - Úvod do STL
- `find` - parametrem je klíč, metoda vrátí pár (klíč,hodnota), ve kterém klíč odpovídá klíči, který byl zadán jako parametr. Není-li takový pár v kontejneru, metoda vrátí iterátor za poslední prvek (iterátor, který vrací metoda `end()`).
- `count` - parametrem je klíč. Metoda vrátí 1, jestliže v kontejneru existuje hodnota asociována k danému klíči, jinak vrátí 0. Vlastně vrací počet párů s daným klíčem.
- `insert` - existují opět 3 varianty této metody (tak jako u šablony `vector`. Jedna možnost je předat metodě `insert` jako parametr objekt typu `value_type`. Můžeme jej například vytvořit pomocí šablony funkce `std::make_pair`. V případě, že v kontejneru neexistuje pár s klíčem daným jako parametr, bude zadaný pár do kontejneru vložen. Jestliže daný klíč již v kontejneru existuje, nestane se nic. Poloha pro nový prvek bude bude vybrána tak, aby kontejner zůstal uspořádán pomocí hodnot klíčů. Další dvě varianty metody `insert` jsou obdoné jako u šablony vector (Viz můj článek Šablona vector v C++ a iterátory). Lze pomocí iterátor upřesnit polohu, nebo vložit prvky s jiného kontejneru. Upřesníme-li polohu iterátorem, kontejner si zkontroluje, zda je poloha správná. Kontejner musí být uspořádán podle klíčů. Je-li správná, potom dojde ihned k vložení páru. Ušetří se čas, který by kontejner potřeboval k nalezení správného místa. V opačném případě kontejner nalezne vhodné místo sám. Zachová se tedy, jako by jsme polohu iterátorem vůbec nezadali.
- `erase` - existují dvě varianty této metody. Mohu zadat klíč nějaké asociované hodnoty, nebo iterátor udávající polohu páru. Metoda erase odstraní prvek (i s klíčem, tedy celý pár) z kontejneru.
- Operátor `[]` - slouží k přístupu k hodnotám pomocí klíčů. Není-li v kontejneru pár s daným klíčem, bude vložen pár s daným klíčem, a hodnotou vytvořenou bezparametrickým konstruktorem. Takové chování nemusí být vždy žádoucí (viz příklad).

To je výčet asi nejdůležitějších metod. Nyní si vše osvětlíme na příkladu

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

int main(int argc, char **argv)
{
    std::map<std::string,int> TelefoniSeznam;
    /* Klíč je string, hodnota je int */
    TelefoniSeznam.insert(std::make_pair(std::string("Dostal"),4578963));
    TelefoniSeznam.insert(std::make_pair(std::string("Novak"),12458632));
    TelefoniSeznam.insert(std::make_pair(std::string("Hasici"),150));
    TelefoniSeznam.insert(std::make_pair(std::string("Policie"),158));
    std::cout << "Hasici " << TelefoniSeznam[std::string("Hasici")] << std::endl;
    std::cout << "Dostal Radim " << TelefoniSeznam[std::string("Dostal")]
            << std::endl;
    std::string jmeno("Bond");
    std::cout << "Bond James ";
    if (!TelefoniSeznam.count(jmeno))
       std::cout <<  "neni v seznamu." << std::endl;
    else
```

```
      std::cout << TelefoniSeznam[jmeno] << std::endl;
    std::cout << "Muzete se o tom presvedcit: " << TelefoniSeznam[jmeno]
            << std::endl;
    /* Právě jsem zapsal Bonda do seznamu!
    Takové chování operátoru [] nemusí být vždy žádoucí.
         Může být lepší raději použít metodu find. */
    std::map<std::string,int>::iterator i;
    for(i = TelefoniSeznam.begin(); i != TelefoniSeznam.end(); i++)
    {
          /* Bond je tady taky! */
          std::cout << i->first << " " << i->second << std::endl;
          /* Šablona struktury pair má dva atributy: first a second */
    }
    TelefoniSeznam.erase(TelefoniSeznam.begin());
    TelefoniSeznam.erase(std::string("Novak"));
    std::cout << std::endl;
    for(i = TelefoniSeznam.begin(); i != TelefoniSeznam.end(); i++)
    {
          std::cout << i->first << " " << i->second << std::endl;
    }
    return 0;
}
```

Doufám, že tentokrát jsem nikde nezapoměl na prostor jmen `std`. Dobře si všimněte, jak se v kontejneru objevil Bond. Použil jsem ho jako parametr pro operátor [].

Šablona multimap

Šablona `multimap` je asociativní pole, které umožní mít k jednomu klíči asociováno několik hodnot. Práce se šablonou `multimap` je podobná jako práce se šablonou `map`. Zaměřím se pouze na rozdíly. První důležitá věc je, že šablona třídy `multimap` již nemá operátor []. Prvky s daným klíčem lze nalézt pomocí metody `find`. Tím ale nalezneme pouze jeden (ten první) prvek s daným klíčem. Chceme-li nalézt všechny prvky s daným klíčem, musíme použít metody `lower_bound`, `upper_bound`, nebo `equal_range`. Tyto metody byly i v šabloně `map`, ale tam neměly snad žádné využití.

- `lower_bound` - parametrem je klíč. Vrací iterátor na prvek s nejmenším klíčem, který je větší, nebo roven zadanému klíči. Neexistuje-li takový prvek, vrací iterátor za konec kontejneru (`end()`).
- `upper_bound` - parametrem je klíč. Vrací iterátor na prvek s největším klíčem, který je menší, nebo roven zadanému klíči. Neexistuje-li takový prvek, vrací iterátor za konec kontejneru (`end()`).
- `equal_range` - parametrem je klíč. Vrací dvojci iterátorů (`pair` - viz příklad níže). První ve dvojci je výsledek volání metody `lower_bound`. Druhý ve dvojci je výsledek volání metody `upper_bound`. Metoda `equal_range` za nás vlastně zavolá tyto dvě metody.

Nyní si vytvoříme ukázkový příklad. Bude se jednat o jednoduchou ukázkovou hash tabulku s velice jednoduchou hash funkcí - mod 13. V tabulce budou uloženy čísla `int`. Jejich klíče budou také typu `int`. Klíče jsou `int` pouze pro přehlednost tohoto ukázkového zdrojového textu. Pro úsporu místa by bylo asi lepší mít klíče `short int`, nebo i `char`. Hodnota klíče totiž nebude nikdy větší než 12.

```
#include <iostream>
#include <map>

using namespace std;

typedef multimap<int,int> THashTabulka;

int main(int argc, char **argv)
{
    THashTabulka Tabulka;
    int p;
    THashTabulka::iterator i;

    for(p = 39; p >= 0; p--)
    {
             Tabulka.insert(make_pair(p % 13, p));
    }
    cout << "Pocet prvku v tabulce " << Tabulka.size() << endl;
    for(i = Tabulka.begin(); i != Tabulka.end(); i++)
    {
        cout << "(" << i->first << "," << i->second << ")\t";
    }
    cout << endl;
    cout << "PocetPrvku s klicem 10 je " << Tabulka.count(10)
         << ". Prvni je: " << (*Tabulka.find(10)).second << endl;
    i = Tabulka.find(5);
    cout << "PocetPrvku s klicem 5 je " << Tabulka.count(5) << endl;
    for(p = 0; p < Tabulka.count(5); p++,i++)
    {
        cout << "(" << i->first << "," << i->second << ")\t";
    }
    cout << endl;
    /* Lepší způsob je: */
    THashTabulka::iterator j = Tabulka.upper_bound(0);
    cout << "Prvky s indexem 0:" << endl;
    for(i = Tabulka.lower_bound(0); i != j; i++)
    {
        cout << "(" << i->first << "," << i->second << ")\t";
    }
    cout << endl;
    /* Asi nejlepší způsob je: */
```

```
    pair<THashTabulka::iterator,THashTabulka::iterator>
                      par=Tabulka.equal_range(7);
    cout << "Klici s indexem 7 odpovida:" << endl;
    for(i = par.first; i != par.second; i++)
    {
        cout << i->second << "\t";
    }
    cout << endl;
    return 0;
}
```

Tolik tedy k asociativním polím. Až se budeme zabývat tématem funkčních objektů v C++, připomeneme si, že i v šablonách `map` a `multimap` se dají funkční objekty použít. Je to v případě, kdy nechcete, aby kontejner třídil klíče podle relačního operátoru <, ale podle jiné, Vámi zadané relace. Příště se podíváme na množinu a multimnožinu v C++.