Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 17.10. 2001
Url: http://www.builder.cz/art/cpp/cpp_vector.html

Vector - datový kontejner v C++

Dnes se podíváme na šablonu třídy `vector`. Jedná se o datový kontejner z knihovny STL jazyka C++. Vektor je šablona jednorozměrného pole. Na rozdíl od "klasického" pole má `vector`, mnoho užitečných vlastností a služeb. Lze do něj například pomocí různých metod vložit prvek, a tím zvětšit jeho velikost.

Šablona `vector` je datový kontejner - posloupnost. Má tedy všechny prvky, které jsem ve svém minulém článku vyjmenoval pro datové kontejnery posloupnosti. Tento kontejner je deklarován v hlavičkovém souboru `vector` v prostoru jmen `std`. Podívejme se na velice jednoduchý příklad.

```
#include <vector>
#include <iostream>

int main()
{
        std::vector<int> a(7); // Pole 7 čísel
        std::vector<int> b; // Pole 0 čísel
        std::cout << "Počet prvků v a je " << a.size() << "  ";
        std::cout << "Počet prvků v b je " << b.size() << std::endl;
        for(int i = 0; i < a.size(); i++)
        { // "Normálně lze používat operátor []
                a[i] = i + 1;
                std::cout << "a[" << i << "] = " << a[i] << std::endl;
        }
        b = a; //Bez problémů lze použít operátor =, lze použít
              //také kopírovací konstruktor
        a.resize(15); // a má nyní 15 prvků
        std::cout << "Počet prvků v a je " << a.size() << "   ";
        std::cout << a[13] << std::endl; // Náhodný
        a.resize(3); // v a zůstalo jen prvních 3 prvků
        for(int i = 0; i < a.size(); i++)
        {
                std::cout << "a[" << i << "] = " << a[i] << std::endl;
        }
        for(int i = 0; i < b.size(); i++)
        {
                std::cout << "b[" << i << "] = " << b[i] << std::endl;
```

```
        }
        if (a != b)
        {
                std::cout << "a != b " << std::endl;
        }
        return 0;
}
```

V příkladu je vidět základní práci s vektorem. Další zajímavé metody jsou `back`, `push_back`, `pop_back`. Metoda `back` vrátí referenci na poslední prvek. Metoda `push_back` vloží nový prvek na konec. Ne tak, že by jím "přemazala" poslední prvek, ale zvýší velikost vektoru o 1 prvek (na konci). Metoda `pop_back` naopak odstraní poslední prvek, čímž sníží velikost vektoru o 1. Například:

```
#include <vector>
#include <iostream>

int main()
{
        std::vector<int> a;
        for(int i = 0; i < 5; i++)
        {
                a.push_back(i+1);
                std::cout << "Poslední prvek je: " << a.back() << std::endl;
        }
        for(int i = 0; i < a.size(); i++)
        {
                std::cout << a[i] << '\t';
        }
        std::cout << std::endl;
        while (!a.empty()) // Opakuj, dokud není prázdný (Dokud nemá 0 prvků )
        {
                a.pop_back(); // Odeberu poslední prvek
        }
        std::cout << "Velikost " << a.size() << std::endl;
        return 0;
}
```

Další velmi užitečné metody pro vkládání a likvidování prvků jsou `insert` a `erase`. K jejich používání musíme ale vědět něco o iterátorech, kterým se budeme věnovat příště. Zatím se tedy těmto metodám nebudeme věnovat.

Podívejme se ještě jak udělat pomocí vektoru vícerozměrné pole. Vytvoříme vlastně vektor vektorů.

```
#include <vector>
```

```
#include <iostream>

using namespace std;

int main()
{
        vector<vector<int> > matice(3); //Matice 3 x 0
        /*Mezera mezi > a > je důležitá!
         Viz moje předchozí články o šablonách. */
        for(int a = 0; a < 3; a++)
        {
                matice[a].push_back(a+1);
                matice[a].push_back(a+2);
                matice[a].push_back(a+3);
        }
        /* Nyní máme matici 3 x 3 */
        for(int y = 0; y < 3; y++)
        {
                for(int x = 0; x < 3; x++)
                {
                        cout << matice[x][y] << '\t';
                }
                cout << endl;
        }
        return 0;
}
```

Chtěl bych je upozornit, že podobným způsobem lze vytvořit vícerozměrné pole i z mé šablony array, kterou jsem vytvořil v článku "Pole s libovolným intervalem indexování v C++ ".

Závěrem bych chtěl ještě připomenout, že každý typ, jehož proměnné chceme vkládat do vektoru musí být schopen vytvořit svou kopii pomocí kopírovacího konstruktoru a operátoru=. Nebudou-li vyhovovat implicitní kopírovací konstruktor a operátor =, musíme je vytvořit, resp. přetížit. Vyhneme se tím zbytečným a na první pohled nepochopitelným pádům programu. Dále typ, jehož prvky budeme vkládat do vektoru musí mít k dispozici bezparametrický konstruktor. Já jsem používal ve svých příkladech typ `int`, nebo v posledním `vector`. U typu `int` je vše potřebné již implicitně dáno. Instance šablony `vector<>` je třída, která má všechny potřebné konstruktory k dispozici a operátor = je přetížen.

V příštím článku se podrobněji podíváme na iterátory. Ukážeme si také další metody šablony `vector`. V dalších článcích se podíváme na podle mne také důležité kontejnery. Budou to mapa a multimapa (asociativní pole).