Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 26.11. 2001
Url: http://www.builder.cz/art/cpp/mnozinacpp.html

Množina v C++

Dnes si povíme něco o množině a multimnožině v C++. Množina je datová struktura, ve které jsou uloženy nějaké prvky. V množině nesmí být dva stejné prvky. Naopak multimnožina může obsahovat i stejné prvky. Nad množinou lze provádět množinové operace jako jsou například sjednocení a průnik. Pro množinu a multimnožinu existují v C++ šablony `set` a `multiset`.

Šablony `set` a `multiset` jsou setříděné asociativní kontejnery. Na rozdíl od šablon `map` a `multimap` v nich nejsou uloženy dvojce (klíč, hodnota), ale jsou zde uloženy pouze klíče. O kontejnerech `map` a `multimap` jsem se zmiňoval ve svém předchozím článku Asociativní pole v C++.

Množina

Množina je datová struktura ve které jsou uloženy jednotlivé prvky. Prvky v množině nesmí být stejné. Množina je v C++ reprezentována šablonou `set`. Šablona `set` má jeden povinný parametr, který udává typ prvků v množině, a jeden nepovinný parametr, který udává způsob, jakým se budou prvky v množině porovnávat. Nezadáme-li druhý parametr, bude se používat operátor <. Druhým parametrem může být ukazatel na funkci, nebo tak zvaný funkční objekt. Prozatím nebudeme druhý parametr používat. Příští článek bude na téma funkční objekty v C++, potom si povíme něco více o těchto možnostech. Jak je zatím vidět, šablona `set` se přece jenom trochu liší od množiny, jakou známe z matematiky. Prvním rozdílem je, že v šabloně `set` mohou být uloženy pouze prvky stejného typu. Mohou zde být i instance různých tříd, ale tyto třídy musejí mít stejného předka (musejí být odvozeny ze stejné třídy). Druhý zásadní rozdíl je v tom, že pro prvky v množině jsou vždy seřazené. Musí být tedy definováno porovnání pomocí operátoru <, nebo jiné porovnání (které se zadá právě tím druhým, nepovinným parametrem). Důležitý je fakt, že je nutné, aby prvky v kontejneru byly seřazeny podle nějaké relace. Kontejner totiž setřídění prvků využívá při vyhledávání a ukládání, kdy použije rychlejší algoritmus.

Šablona `set` má metody velmi podobné jako šablona `map` z minulého článku. Pro vkládaní prvků zde existují metody `insert`. Pro odstranění prvků zase `erase`. Existují zde metody `begin` a `end` pro práci s iterátory atd... Myslím, že nemá smysl zde význam těchto metod znovu vypisovat. Všechny jsou popsány v mých předchozích článcích. Prvky, které se budou do množiny ukládat musejí být schopny vytvářet kopie pomocí kopírovacího konstruktoru, a operátoru =. Budeme-li chtít zjistit, jestli se nějaký prvek nachází v množině použijeme metodu `count`, která vrátí 0, jestliže prvek v množině není, jinak vrátí 1. Tato metoda vlastně vrací počet prvků v množině, což je 0, nebo 1. U multimnožiny může vrátit 0, nebo celé kladné číslo. V souvislosti s množinou bych ale chtěl upozornit na některé množinové operace, které jsou v STL implementovány. V STL jsou tak zvané standardní algoritmy, což jsou šablony funkcí pracujících s kontejnery. Standardním algoritmům se chci věnovat v, samostatném článku. Dnes si v souvislosti s množinou ukážeme pouze 4 algoritmy - množinové operace.

Operace nad množinou

Průnik množin A a B je množina, jejíž prvky jsou v A a zároveň v B. Sjednocení množin A a B je množina, jejíž prvky jsou v A, nebo v B. Množinový rozdíl množin A a B je množina, jejíž prvky jsou v A, a zároveň nejsou v B. Symetrická diference dvou množin je množina, která je Sjednocení(A,B) - Průnik(A,B), kde symbol - značí množinový rozdíl. Snad není tento zápis moc matoucí, ale já si vůbec nevím rady s tím, jak napsat v HTML symboly pro množinové operace.

operace	**algoritmus**
průnik	set_intersection
sjednocení	set_union
množinový rozdíl	set_difference
symetrická diference	set_symmetric_difference

Všechny algoritmy jsou šablony funkcí. Všechny šablony mají 3 parametry. První dva jsou typy vstupních iterátorů obou množin. Třetí parametr je typ výstupního iterátoru výsledku. Každá funkce má 5 parametrů. Nejlépe bude napsat deklaraci jedné z nich:

```
template<class InputIterator_1, class InputIterator_2, class OutputIterator > OutputIterator set_union(InputIterator_1 zacatekPrvniho,
InputIterator_1 konecPrvniho, InputIterator_2 zacatekPrvniho, InputIterator_2 konecPrvniho, OutputIterator zacatekVysledku);
```

První dva parametry funkce udávají začátek a konec první množiny. Třetí a čtvrtý parametr udává začátek a konec druhé množiny. První a druhá dvojce parametrů nejsou stejného typu. Znamená to, že nemusíme provádět sjednocení pouze množin (`set`), ale i jiných kontejnerů. Ale pokud použijeme jiné kontejnery, musejí být setříděné. Posledním parametrem je výstupní iterátor výsledku. Dalším důležitým algoritmem je algoritmus `includes`, který určí, zda jedna množina je podmnožinou druhé. Šablona `includes` má 2 parametry. Jsou to typy vstupních iterátorů první a druhé množiny. Parametry funkce jsou 4. Začátek a konec obou množin. Všechny tyto algoritmy jsou deklarovány v hlavičkovém souboru `algorithm` v prostoru jmen `std`. Nejlépe bude, ukážeme-li si příklad.

```
#include <string>
#include <set>
#include <algorithm>
#include <iterator>

using namespace std;

template<class InputIterator> void vypis(ostream &os, InputIterator zacatek,
         InputIterator konec)
{
    /* Funkci jsem si vypůjčil ze svého článku o iterátorech */
    for(InputIterator i = zacatek; i != konec; i++)
    {
```

```
        os << *i << " ";
    }
    cout << endl;
}

int main(int argc, char **argv)
{
    set<string> ctyruhelniky, stejneDlouhe, vysledek;
    ctyruhelniky.insert(string("ctverec"));
    ctyruhelniky.insert(string("obdelnik"));
    ctyruhelniky.insert(string("kosoctverec"));
    ctyruhelniky.insert(string("lichobeznik"));
    ctyruhelniky.insert(string("rovnobeznik"));
    stejneDlouhe.insert(string("ctverec"));
    stejneDlouhe.insert(string("kosoctverec"));
    stejneDlouhe.insert(string("rovnostranny trojuhelnik"));
    cout << "Vypis ctyruhelniku:" << endl;
    vypis(cout,ctyruhelniky.begin(),ctyruhelniky.end());
    cout << "Vypis rovinnych obrazcu, ktere maji stejne velikosti svych stran:"
        << endl;
    vypis(cout,stejneDlouhe.begin(),stejneDlouhe.end());
    insert_iterator<set<string> > iter(vysledek,vysledek.begin());
    /* Sjednocení: */
    set_union(ctyruhelniky.begin(),ctyruhelniky.end(),stejneDlouhe.begin()
        ,stejneDlouhe.end(),iter);
    cout << "Vypis \"vsech\" rovinnych obrazcu:" << endl;
    vypis(cout,vysledek.begin(),vysledek.end());
    vysledek.erase(vysledek.begin(),vysledek.end());
    /* Průnik: */
    set_intersection(ctyruhelniky.begin(),ctyruhelniky.end(),stejneDlouhe.begin()
        ,stejneDlouhe.end(),iter);
    cout << "Vypis rovnostrannych ctyruhelniku:" << endl;
    vypis(cout,vysledek.begin(),vysledek.end());
    vysledek.erase(vysledek.begin(),vysledek.end());
    /* Rozdíl: */
    set_difference(ctyruhelniky.begin(),ctyruhelniky.end(),stejneDlouhe.begin()
        ,stejneDlouhe.end(),iter);
    cout << "Vypis nerovnostranných ctyruhelniku:" << endl;
    vypis(cout,vysledek.begin(),vysledek.end());
    vysledek.erase(vysledek.begin(),vysledek.end());
    /* Symetrická diference: */
    set_symmetric_difference(ctyruhelniky.begin(),ctyruhelniky.end(),
                    stejneDlouhe.begin(),stejneDlouhe.end(),iter);
    cout << "Vypis \"vsech\" rovinnych obrazcu,
        krome rovnostranych ctyruhelniku:" << endl;
    vypis(cout,vysledek.begin(),vysledek.end());
    vysledek.erase(vysledek.begin(),vysledek.end());
```

```
    /* Je čtverec čtyřúhelník? (Náleží čtverec množině čtyřúhelníků?) */
    if (ctyruhelniky.count(string("ctverec")) == 1)
    {
         cout << "Ano" << endl;
    }
    /* Je autor článku čtyřúhelník ? */
    if (ctyruhelniky.count(string("Radim Dostál")) == 0)
    {
         cout << "Neni" << endl;
    }
    if (!includes(ctyruhelniky.begin(),ctyruhelniky.end(), stejneDlouhe.begin()
         , stejneDlouhe.end()))
    {
        cout << "Rovnostranne plosne obrazce nejsou podmnozinou ctyruhelniku"
             << endl;
    }
    return 0;
}
```

V článku jsem použil zmiňované množinové operace. Také jsem ukázal jak zjistit, zda nějaký prvek náleží do množiny, nebo ne. Šablona `set` má metody `begin` a `end` vracející konstantní iterátory. Konstantní iterátor nelze použít jako výstupní, proto nemůže být pátým parametrem funkcí set_.... parametr `vysledek.begin()`. Kdybych pracoval s kontejnerem, který nemá metodu `begin` pouze konstantní (viz další příklad) musel bych řešit další problém. Musel bych mít ve výsledném kontejneru vytvořeno místo pro výsledné prvky. Abych se vyhnul těmto problémům použil jsem adaptér jménem `insert_iterator`. Jedná se o šablonu třídy. Parametrem šablony je typ kontejneru, do kterého se bude iterátor "odkazovat". Parametry konstruktoru jsou konkrétní instance kontejneru, a pozice v kontejneru, na kterou se bude zapisovat. Takto vytvořený objekt se pro své okolí jeví jako výstupní iterátor. Může se kdykoliv jako výstupní iterátor použít. Zavolá-li se pro něj operátor = ,tedy zapisuje-li se na danou pozici, objekt třídy `insert_iterator` vloží do "svého" kontejneru nový prvek pomocí metody kontejneru `insert`. Tím vytvoří v kontejneru místo pro nový prvek, a na toto místo nový prvek zapíše. Všechna tato činnost je zapouzdřena v metodě `insert_iterator::operator=(const typename Container::value_type& novaHodnota)`, a uživatel třídy se o ní nemusí starat. `Container` je parametr šablony, `value_type` je typ prvků v kontejneru.
V mém druhém příkladě tento adaptér schválně nepoužiji, abych ukázal jeho výhody. Ještě jen musím dodat, že existují i další, podobné adaptéry, které jsem zde mohl použít. Adaptér `insert_iterator` vkládá do kontejneru vždy na pozici udanou parametrem v konstruktoru. Dále existují `back_insert_iterator`, který vkládá vždy na konec kontejneru, nebo `front_insert_iterator`, který vkládá vždy na začátek kontejneru. Ale u setříděných kontejnerů nehraje žádnou roli poloha, na kterou se ukládá, protože prvek bude vložen vždy na své místo dané uspořádáním. Proto je jedno jaký adaptér jsem zvolil pro náš příklad. Všechny zmiňované adaptéry jsou deklarovány v hlavičkovém souboru `iterator` v prostoru jmen `std`.

Multimnožina

Práce s multimnožinou je obdobná jako práce s množinou. Multimnožina může obsahovat více stejných prvků. Pomocí metody `count` zjistíme nejen to, zda prvek v množině je, ale také kolikrát v dané množině je. Přejděme rovnou na příklad. Jeden ze způsobu jak najít nejvyššího společného dělitele

dvou čísel je rozložit obě čísla na součin prvočísel. Každý rozklad je vlastně multimnožina, a součin prvků jejich průniků je výsledek. Všimněte si, jaké problémy přináší, když se pokusím nepoužít adaptér vkládací iterátor.

```
#include <set>
#include <algorithm>
#include <vector>

using namespace std;

void rozloz(int cislo, multiset<int> &vysledek)
{
    register int i = 2;
    vysledek.insert(1);
    while(cislo != 1)
    {
        if (cislo % i == 0)
        {
            cislo /= i;
            vysledek.insert(i);
        }
        else
        {
            i++;
        }
    }
}

int main()
{
    multiset<int> prvni,druha;
    /* Množinové operace nemusíme provádět pouze se set, nebo multiset*/
    vector<int> vysledny Rozklad(100,0);
    /*
      MUSÍM zadat velikost vektoru!! Jinak by v těle funkce set_intersection
      bylo zapsáno do nealokované paměti. Protože výraz vysledny Rozklad.begin()
      by se odkazoval ZA poslední prvek kontejneru. Daleko vhodnější by bylo
      použití nějakého "vkládacího" iterátoru. Tento příklad je pouze
      demonstrační. Krom toho, že musím zadávat velikost kontejneru, jsem také
      touto velikostí limitován. Kdyby měl výsledný rozklad více než 100 cifer,
      program by havaroval. Proto bylo lepší použít nějaký adaptér jako v
      minulém příkladě. Protože vector není seřazený kontejner, bylo by již
      podstatné, jaký adaptér zvolíme.
    */
    int vysledek = 1;
    rozloz(120,prvni);
    rozloz(740,druha);
```

```
    set_intersection(prvni.begin(),prvni.end(),druha.begin(),druha.end()
         ,vysledny Rozklad.begin());
    for(vector<int>::iterator i = vysledny Rozklad.begin(); *i != 0; i++)
    {
         vysledek *= *i;
    }
    cout << vysledek << endl;
    return 0;
}
```

Příště se podíváme na funkční objekty v C++.

------ http://www.builder.cz ------