Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 27.02. 2002
Url: http://www.builder.cz/art/cpp/stdalg_end.html

Standardní algoritmy v C++ - dokončení

Dnes dokončíme téma standardních algoritmů z STL. Ve svých poslední několika článcích jsem se zabýval některými šablonami funkcí z STL. Dnes se pokusím vše shrnout do přehledné tabulky, ze které by mělo být zřejmé, kdy jaký algoritmus použít. Ještě než začneme přehledem musím jen upozornit, že algoritmy o kterých jsem psal nejsou zdaleka všechny.

Použití	Popis	Jména šablon	V článku
Algoritmy pro vyplňování kontejnerů.	Algoritmy vyplní kontejner nebo jeho část určenou hodnotou. Hodnota může být konstantní, nebo proměnná.	fill, fill_n, generate, generate_n	Úvod do standardních algoritmů v C++
Algoritmy pro kopírování dat	Algoritmy kopírují data mezi datovými kontejnery. Data lze kopírovat "ze předu", nebo "ze zadu".	copy, copy_backward	Kopírovací a přesouvací algoritmy v C++
"Přesouvací" algoritmy	Algoritmy ve skutečnosti data nepřesunují. Některé algoritmy data z kontejneru smažou, jiné vytvoří kopie určených prvků. Slovo přesouvací je nepřesný překlad slova remove.	remove, remove_copy, remove_if, remove_copy_if	Kopírovací a přesouvací algoritmy v C++
Algoritmy pro vyhledávání dat	Algoritmy vyhledávají data v kontejnerech podle různých kritérií. Existují jak sekvenční vyhledávací algoritmy, tak i binární vyhledávací algoritmy.	find, find_if, search, binary_search	Vyhledávací algoritmy v C++
Prohledávací ("skenovací") algoritmy	Algoritmy prohlédnou kontejner, ve kterém provedou pro každý prvek nějakou operaci. Může se jednat přímo o zavolání funkce, kde je daný prvek parametrem (algoritmus for_each), nebo může třeba jenom zvýšit čítač, tedy počítat počet nějakých prvků.	for_each, count, count_if, accumulate	Skenovací (prohlížecí) algoritmy v C++
Algoritmy pro transformaci	Algoritmy změní (přetransformují) data v kontejneru. Transformace se provádějí podle různých kritérií.	transform, reverse, reverse_copy, replace, replace_if, replace_copy, replace_copy_if	Transformační algoritmy v C++
			Řadící algoritmy v

Řadící algoritmy	Algoritmy seřadí (nepřesně řečeno třídí) data v kontejneru.	sort, stable_sort	C++
Algoritmy pro práci s haldou	Algoritmy pracují se standardní haldou. Jsou zde algoritmy pro vytvoření hady, přidání a odebrání prvku z haldy, převedení haldy na seřazenou posloupnost.	make_heap, pop_heap, push_heap, sort_heap	Halda v C++
Množinové operace	Algoritmy pro práci s množinou a multimnožinou. Nalezneme zde algoritmy pro průnik, sjednocení, podmnožinu, množinový rozdíl, symetrickou diferenci.	set_intersection, set_union, set_difference, set_symmetric_difference, includes	Množina v C++
Algoritmy, které nepracují s kontejnery	Algoritmy, které nepracují s kontejnery. Provádí většinou nějakou jednoduchou činnost	min, max, swap	Úvod do standardních algoritmů v C++

Znovu připomínám, že se nejedná o všechny algoritmy z STL.

Až na pár vyjímek jsou všechny algoritmy deklarovány v hlavičkovém souboru `algorithm`. Všechny jsou deklarované v prostoru jmen std. Všechny jsou nezávislé na typu kontejneru. Ke kontejneru přistupují pomocí iterátorů. Velmi často se při používání standardních algoritmů používají funkční objekty.

V mnoha algoritmech se jako parametr udávají iterátory začátek a konec. Chtěl bych upozornit, že algoritmus proběhne vždy od prvku daného iterátorem začátek a konče prvkem před prvkem daným iterátorem konec. Prvek, na který se odkazuje iterátor konec již nebude použit. Pracujeme-li s celým kontejnerem, zadáváme jako konec iterátor za poslední prvek. Tedy iterátor, který vrací metoda `end()`. Pracujeme-li s polem, zadáme jako konec ukazatel na prvek za posledním prvkem. Můžete si být jisti, že ukazatel, nebo iterátor konec nebudou nikdy dereferencováni.

Algoritmus adjacent_find

Na závěr bych se chtěl zmínit o algoritmu `adjanced_find`. Tento algoritmus by se dal zařadit mezi vyhledávací algoritmy, ale já jsem na něj ve článku o vyhledávacích algoritmech nějak zapomněl. Jak jsem již napsal, neuváděl jsem všechny algoritmy, jen ty, které se mi zdály užitečné. Myslím, že `adjanced_find` mezi užitečné patří. Deklarace:

- `template <class ForwardIterator> ForwardIterator adjacent_find(ForwardIterator zacatek, ForwardIterator konec);` Algoritmus v oblasti dané iterátory nalezne dva po sobě jdoucí prvky, které jsou si rovny. K porovnání použije operátor ==. Existuje-li dvojce, funkce vrátí iterátor na první prvek dvojce. Neexistuje-li funkce vrátí iterátor `konec`. Parametry šablony jsou typy iterátorů.
- `template <class ForwardIterator, class Podminka> ForwardIterator adjacent_find(ForwardIterator zacatek, ForwardIterator konec, Podminka pod);` Obdobná činnost jako předchozí. Jen s tím rozdílem, že nebude použit operátor ==, ale binární funkční objekt. Tedy nalezne první dvojici, pro které je `pod(*i,*(i+1)) == true`. Identifikátor `i` je iterátor.

Příklad:

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

using namespace std;

int main()
{
    int pole[5] = { 90, 30, 10, 10, 7 };
    /* Naleznu první dva prvky v poli, které jsou stejné */
    int *i = adjacent_find(pole,&pole[5]);
    if (i != &pole[5])
    {
        cout << "Stejná dvojce: " << *i << " " << *(i+1) << endl;
    }
    /* Naleznu první dva prvky v poli, které nejsou dělitelné */
    int *ii = adjacent_find(pole,&pole[5],modulus<int>());
    /*
      Modulus je zbytek po celočíselném dělení.
      Modulus vrací 0 (false) pro čísla dělitelná.
    */
    if (ii != &pole[5])
    {
        cout << "První dvě čísla, která nejsou beze zbytku dělitelná "
             << *ii << " " << *(ii+1) << endl;
    }
    /* Nyní to samé pro vektor */
    vector<int> v(pole,&pole[5]);
    vector<int>::iterator a = adjacent_find(v.begin(),v.end());
    vector<int>::iterator b = adjacent_find(v.begin(),v.end(),modulus<int>());
    if (a != v.end())
    {
        cout << "Stejná dvojce: " << *a << " " << *(a+1) << endl;
    }
    if (b != v.end())
    {
        cout << "První dvě čísla, která nejsou beze zbytku dělitelná "
             << *b << " " << *(b+1) << endl;
    }
    return 0;
}
```

Tímto jsme téma standardních algoritmů z STL ukončili. U STL ale ještě zůstaneme. Příště se podíváme na tak zvané automatické ukazatele.