Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 05.02. 2002
Url: http://www.builder.cz/art/cpp/cpptranform.html

Transformační algoritmy v C++

Transformační algoritmy jsou algoritmy, které nějakým způsobem mění (přetransformují) kontejner, nebo jeho část. Mohou změnit hodnoty prvků (algoritmy `replace`), mohou obrátit pořadí prvků (algoritmy `reverse`), nebo mohou kontejner přetransformovat tak, že na každý prvek zavolají funkci, nebo operátor () funkčního objektu. Některé varianty algoritmů mění přímo jim daný kontejner, jiné vytvoří nový kontejner, do kterého uloží výsledek. Originální kontejner pak zůstane nezměněn.

Nejprve se podívejme na algoritmus transform. O něm jsem se zmínil již minule, když jsme se zabývali algoritmem `for_each`. Algoritmus `transform` je tak jako každý jiný algoritmus z STL šablona funkce. Existují dvě varianty `transform`. Deklarace jsou:

- `template <class InputIterator,class OutputIterator,class TUnarniOperace> OutputIterator transform (InputIterator zacatek, InputIterator konec, OutputIterator zacatekVysledku, TUnarniOperace operace);` - Parametry šablony jsou typy vstupních a výstupních iterátorů. Dále je parametrem typ unární operace. Může se jednat buď o typ ukazatele na funkci, nebo o třídu funkčních objektů. Parametry funkce jsou `začátek` a `konec` kontejneru, nebo jeho části, která má být transformována. Dalším parametrem je začátek oblasti výsledku. Posledním parametrem je unární operace. Činnost algoritmu se dá popsat takto. Pro všechny prvky počínaje prvkem který je dán iterátorem `začátek` a konče prvkem před prvkem daným iterátorem `konec` bude postupně provedena daná operace, která má prvek jako svůj parametr. Návratová hodnota této operace bude vložena na pozici danou iterátorem `zacatekVysledku`. Chceme-li navíc transformovat kontejner, jehož iterátory mají vlastnosti i vstupních i výstupních iterátorů, lze výsledek zapisovat rovnou do originálního kontejneru. Vše je uvedeno v příkladu.
- `template <class InputIterator1, class InputIterator2, class OutputIterator, class TBinarniOperace> OutputIterator transform (InputIterator1 zacatek1, InputIterator1 konec1, InputIterator2 zacatek2, OutputIterator zacatekVysledku, TBinarniOperace operace);` - Parametry šablony jsou dva typy vstupních iterátorů. Bude se pracovat s dvěma vstupními kontejnery, proto pro každý jeden. Dále typ výstupního iterátoru a typ binární operace. Parametry funkce jsou začátek a konec oblasti v prvním kontejneru. Začátek oblasti v druhém kontejneru, následuje začátek oblasti pro výsledek a binární operace. Algoritmus postupně provede binární operaci nad prvky prvního a druhého kontejneru, a výsledek uloží na pozici pro výsledek. Tedy:

```
*začátekVýsledku = operace(*začátek1,*začátek2);
*(začátekVýsledku + 1) = operace(*(začátek1 + 1),*(začátek2 + 1));
atd...
```

Předpokládá se, že je k dispozici potřebný počet prvků za prvkem, který je dán iterátorem začátek2. Stejně tak musí být zajištěno, že je dostatek místa v kontejneru od pozice dané iterátorem zacatekVysledku.

Další transformující algoritmy jsou `reverse` a `reverse_copy`. Deklarace jsou:

- `template <class BidirectionalIterator> void reverse (BidirectionalIterator zacatek, BidirectionalIterator konec);` - V oblasti dané iterátory `začátek` a `konec` obrátí pořadí prvků. Tedy první prvek bude poslední a naopak, druhý předposlední a naopak, atd...
- `template <class BidirectionalIterator, class OutputIterator> OutputIterator reverse_copy (BidirectionalIterator zacatek, BidirectionalIterator konec, OutputIterator zacatekVysledku);` - V podstatě stejná činnost jako výše uvedený algoritmus. Rozdíl je jen v tom, že originální kontejner nebude nijak pozměněn. Prvky v obráceném pořadí budou dány do jiného kontejneru na pozici `začátekVýsledku`. Bude tedy vlastně vytvořena kopie dat.

Příklad:

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<functional>
#include<stdlib.h>

using namespace std;

int main(int, char**)
{
    int pole[10] = {-1, 30, 43, -20, 80, -76, 100, 193, -456, 354};
    vector<int> vektor(pole,&pole[10]);
    copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
    cout << endl;
    /*
       Všechny prvky v poli vynásobím deseti.
    */
    transform(pole,&pole[10],pole,bind1st(times<int>(),10));
    copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
    cout << endl;
    /*
       Všechny prvky v poli nahradím jejich absolutní hodnotou.
    */
    transform(pole,&pole[10],vektor.begin(),abs);
    copy(vektor.begin(),vektor.end(),ostream_iterator<int>(cout,"\t"));
    cout << endl;
    /*
       Sečtu prvky v poli s prvky ve vektoru.
```

```
      Výsledek uložím opět do vektoru.
   */
   transform(pole,&pole[10],vektor.begin(),vektor.begin(),plus<int>());
   copy(vektor.begin(),vektor.end(),ostream_iterator<int>(cout,"\t"));
   cout << endl;
   /*
      Obrátím pořadí prvků v poli.
   */
   reverse(pole,&pole[10]);
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   /*
      Do vektoru uložím prvky z pole v obráceném pořadí.
   */
   reverse_copy(pole,&pole[10],vektor.begin());
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   copy(vektor.begin(),vektor.end(),ostream_iterator<int>(cout,"\t"));
   cout << endl;
   return 0;
}
```

Další užitečné šablony funkcí jsou šablony `replace`. Jejich deklarace:

- `template <class ForwardIterator, class Typ> void replace (ForwardIterator zacatek, ForwardIterator konec, const Typ& puvodniHodnota, const Typ& novaHodnota);` - Parametry šablony jsou typ iterátoru a typ prvků v kontejneru. Parametrem funkce jsou iterátory `začátek` a `konec`. Dále původní hodnota a nová hodnota. Algoritmus v úseku daným iterátory začátek a konec nahradí všechny původní hodnoty za hodnoty nové. K porovnání bude použit operátor ==.
- `template <class ForwardIterator, class TPodminka, class Typ> void replace_if (ForwardIterator zacatek, ForwardIterator konec, TPodminka podminka, const Typ& novaHodnota);` - Obdobně jako předchozí algoritmus. Pouze hodnoty, které mají být nahrazeny nejsou dány konstantní hodnotou, ale podmínkou. K nahrazení dojde, je-li podmínka pravdivá.
- `template <class InputIterator, class OutputIterator, class Typ> OutputIterator replace_copy (InputIterator zacatek, InputIterator konec, OutputIterator zacatekVysledku, const Typ& puvodniHodnota, const Typ& novaHodnota);` - V podstatě stejná činnost jako u algoritmu `replace` s tím rozdílem, že původní kontejner zůstane nezměněn. Jeho "změněná kopie" bude v jiném kontejneru na pozici dané iterátorem začátek výsledku.
- `template <class InputIterator, class OutputIterator, class TPodminka, class Typ> OutputIterator replace_copy_if (InputIterator zacatek, InputIterator konec, OutputIterator zacatekVysledku, TPodminka podminka, const Typ& novaHodnota);` - Kombinace algoritmů `replace_if` a `replace_copy`.

Srozumitelnější bude příklad.

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

using namespace std;

int main(int, char**)
{
   int pole[10] = {1, 2, 3, -10, -20, -30, 2, 54, -78, 10};
   vector<int> vektor;
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   /*
       Nahradím všechny čísla 2 čísly -2
   */
   replace(pole,&pole[10],2,-2);
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   /*
       Nahradím všechny záporné čísla číslem 0
   */
   replace_if(pole,&pole[10],bind2nd(less<int>(),0),0);
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   insert_iterator<vector<int> >  ins(vektor,vektor.begin());
   /*
       Do vektoru uložím všechny čísla z pole, a při tom nahradím všechny
       čísla 10 čísly 100.
   */
   replace_copy(pole,&pole[10],ins,10,100);
   copy(pole,&pole[10],ostream_iterator<int>(cout,"\t"));
   cout << endl;
   copy(vektor.begin(),vektor.end(),ostream_iterator<int>(cout,"\t"));
   cout << endl;
   return 0;
}
```

Doporučuji porovnat dnešní algoritmy, které mají ve svém názvu copy s podobnými algoritmy, se kterými jsme se již setkali. Bylo to v článku Kopírovací a přesouvací algoritmy v C++.

Příště se podíváme na řadící algoritmy. I řadící algoritmy jsou v C++ již implementovány. Nemusíte žádný quick sort sami psát. Vše je již naprogramováno, stačí jen dané algoritmy používat.

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