Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 14.02. 2002
Url: http://www.builder.cz/art/cpp/sortalg.html

Řadící algoritmy v C++

Řadící algoritmy jsou algoritmy, které seřadí prvky v kontejneru, nebo jeho části. Řadícím algoritmům se často říká třídicí algoritmy. Není to ale přesné označení, protože pod pojmem třídicí algoritmus by jsme si měli představit spíše něco, co rozděluje objekty podle toho, jaké jsou třídy. Prvky lze seřadit podle nějaké relace. Algoritmy z STL k řazení používají buď operátor <, který může být implicitní, nebo přetížený. Dále také umožňují programátorovi zadat funkci vracející `bool`, která definuje relaci, podle níž mají být prvky seřazeny.

Řazení v C

Ještě než se začneme věnovat řadícím šablonám z STL chtěl bych připomenout, že existuje možnost standardního řazení i v ANSI C (nejenom v C++). Jazyk C nezná šablony. K řazení je v C k dispozici funkce `qsort` deklarovaná v hlavičkovém souboru `stdlib.h`. Její deklarace je: `void qsort(void *pole, size_t pocet, size_t velikost, int (*porovnani)(const void *prvni, const void *druhy));`. Prvním parametrem je ukazatel na pole, které chceme seřadit. Jedná se vlastně o ukazatel na první prvek pole (prvek s indexem 0). Druhým parametrem je počet prvků, které chceme seřadit. Následuje velikost jednoho prvku a ukazatel na porovnávací funkci. Mluvíme o jazyku C, nikoliv C++, proto žádné přetěžování operátoru < není možné. Ukazatel na funkci ukazuje na funkci vracející `int` mající dva parametry, což jsou ukazatele na void. Tato funkce by měla vracet libovolné záporné číslo, je-li *a < *b, měla by vracet 0, jestliže *a == *b, a vracet libovolné kladné číslo, jestliže *a > *b. Tím, že zadáváme porovnávací funkci, i velikost prvku, je funkce `qsort` univerzální. Je použitelná pro pole jakýchkoliv prvků. Uvedeme si příklad. Nejprve seřadíme pole `int`, potom pole bodů v 2D podle jejich vzdálenosti k bodu 0,0. Úmyslně program napíši v ANSI C, nepoužiji nic z ANSI C++.

```
#include<stdio.h>
#include<stdlib.h>

/*
   Pozor! Tento příklad je C, ne C++! Žádné jednořádkové komentáře.
   Žádné class, count, atd... :-)
*/

struct TBod
{
   int X,Y;
};
```

```
int porovnejInt(const void *a, const void *b)
{
   /*
      Budu vracet kladné, záporné číslo, nebo 0, podle pravidel,
      která jsem uvedl.
   */
   if ( *((int*)a) < *((int*)b) )
   {
      return -1;
   }
   if ( *((int*)a) > *((int*)b) )
   {
      return 1;
   }
   return 0;
}

int porovnejBody(const void *a, const void *b)
{
   /*
      Nejprve vytvořím ukazatele typu TBod, abych
      nemusel pořád přetypovávavat.
   */
   struct TBod *bodA = (struct TBod*)a, *bodB = (struct TBod*)b;
   /* Nyní spočítám čtverce vzdáleností. */
   int mocninaVzdalenostiA = bodA->X * bodA->X + bodA->Y * bodA->Y;
   int mocninaVzdalenostiB = bodB->X * bodB->X + bodB->Y * bodB->Y;
   if (mocninaVzdalenostiA < mocninaVzdalenostiB)
   {
       return -1;
   }
   if (mocninaVzdalenostiA > mocninaVzdalenostiB)
   {
       return 1;
   }
   return 0;
}

int main(int argc, char** argv)
{
   int p,pole[10] = {12, 35, 100, -14, 45, -89, 8, 87, 100, -100};
   struct TBod body[5] = { {0,100} , {20 , 50}, {-98,45}, {0,0}, {100,1000} };
   qsort((void*)pole,10,sizeof(int),porovnejInt);
   qsort((void*)body,5,sizeof(struct TBod),porovnejBody);
   for(p = 0; p < 10; p++)
   {
      printf("%d ",pole[p]);
```

```
   }
   for(p = 0; p < 5; p++)
   {
      printf("\n(%d,%d)",body[p].X,body[p].Y);
   }
   return 0;
}
```

Tolik k funkci `qsort` z jazyka C. Uvedl jsem ji, protože se na ni často zapomíná.

Řazení v C++

Jazyk C++ po C "zdědil" funkci `qsort`. Můžeme ji tedy používat i v C++. Já to ale nedoporučuji. Funkce `qsort` umí uspořádat pouze pole. Neuspořádáme s ní žádný kontejner z STL. V C++ máme k dispozici šablony funkcí, které umí uspořádat i pole, i datové kontejnery. Než se jim budeme věnovat, podívejme se na trochu teorie o "stabilním" řazení.

Stabilní řazení je řazení, které garantuje, že prvky, které mají stejný klíč (podle kterého se prvky řadí) vůči sobě nezmění pořadí. U "nestabilního" řazení může být tato vlastnost také splněna, ale nemáme jistotu, že tomu tak bude vždy.

K řazení v C++ existují algoritmy `sort` a `stable_sort`. Deklarace:

- `template <class RandomAccessIterator> void sort (RandomAccessIterator zacatek, RandomAccessIterator konec);` - Seřadí oblast danou iterátory `začátek` a `konec` podle operátoru <. Operátor < může být implicitní, nebo přetížený. Parametrem šablony je typ iterátoru, parametry funkce jsou iterátory udávající oblast pro seřazení.
- `template <class RandomAccessIterator, class TPorovnani> void sort (RandomAccessIterator zacatek, RandomAccessIterator konec, TPorovnani porovnani);` - Stejná činnost jako u předchozího algoritmu. Rozdíl je jen v tom, že k porovnání dvou prvků nebude použit operátor <, ale funkce, nebo funkční objekt zadaný programátorem. TPorovnani je buď třída funkčních objektů, nebo ukazatel na funkci. Musí mít dva parametry stejného typu (prvky), a vracet typ `bool`, případně celé číslo.
- `template <class RandomAccessIterator> void stable_sort (RandomAccessIterator zacatek, RandomAccessIterator konec);` - Význam parametrů je stejný jako u algoritmu `sort`. Rozdíl je jen v tom, že `stable_sort` řadí stabilně.
- `template <class RandomAccessIterator, class TPorovnani> void stable_sort (RandomAccessIterator zacatek, RandomAccessIterator konec, TPorovnani porovnani);` - Význam parametrů je stejný jako u algoritmu `sort`. Rozdíl je jen v tom, že `stable_sort` řadí stabilně.

Ukážeme si vše v příkladu.

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

```
#include<algorithm>
#include<functional>
#include<string>

using namespace std;

int main(int, char**)
{
    int pole[5] = {1, 80, -87, 25, 0 };
    vector<int> vektor(pole,&pole[5]);
    /* Seřadím pole vzestupně. */
    sort(pole,&pole[5]);
    copy(pole,&pole[5],ostream_iterator<int>(cout,"\t"));
    cout << endl;
    /*
       Teď prvky vektoru seřadím sestupně. Použiji stabilní řazení,
       i když zde se to nemá jak projevit!
    */
    sort(vektor.begin(),vektor.end(),greater<int>());
    copy(vektor.begin(),vektor.end(),ostream_iterator<int>(cout,"\t"));
    cout << endl;
    return 0;
}
```

Příště se podíváme haldu, povíme si co to je, a ukážeme si standadní operace nad haldou, které nám C++ poskytuje.