Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 01.02. 2002
Url: http://www.builder.cz/art/cpp/scanalg.html

Skenovací (prohlížecí) algoritmy v C++

Skenovací algoritmy jsou algoritmy, které prohlédnou kontejner prvek po prvku, a po každé provedou nějakou operaci. Mezi tyto algoritmy patří: `accumulate` a `for_each`. Také by se do této skupiny mohl zařadit algoritmus `count`, resp. `count_if`. Deklarace jsou:

- `template <class InputIterator, class Typ, class TypVysledek> void count(InputIterator zacatek, InputIterator konec, const Typ &hodnota, TypVysledek &n);` - Parametry šablony jsou 3. První udává typ iterátoru, druhý udává typ prvků v kontejneru a poslední je typ výsledku. Jako typ výsledku bude asi v 99% použit nějaký celočíselný typ. Není to ale podmínkou. Důležité pouze je, aby typ výsledku měl k dispozici operátor ++. Parametry funkce jsou iterátory udávající `začátek` a `konec` oblasti. Dále následuje hledaná hodnota a proměnná pro výsledek. Po každé, co je mezi prvky danými iterátory `začátek` a `konec` nalezena hledaná `hodnota` (Použije se operátor == implicitní, nebo přetížený.), bude zvýšena hodnota výsledku (V našem případě n) pomocí operátoru++.
- `template <class InputIterator, class TPodminka, class TypVysledek> void count(InputIterator zacatek, InputIterator konec, const TPodminka podminka, TypVysledek &n);` - Obdobně jako minule. S rozdílem, že nyní nebude hledána konkrétní hodnota, ale hodnoty, které vyhovují zadané podmínce.
- `template <class InputIterator, class Typ> Typ accumulate (InputIterator zacatek, InputIterator konec, Typ pocatecniHodnota);` - Parametry šablony jsou typ iterátoru a typ prvků v kontejneru. Parametry funkce jsou iterátory udávající `začátek` a `konec` kontejneru. Dále počáteční hodnota výsledku. Funkce vrací součet všech prvků. Použije operátor +, který je buď implicitní, nebo přetížený. Algoritmus vlastně sečte počáteční hodnotu a všechny prvky v kontejneru. Zadáte-li jako počáteční hodnotu 0 (Což uděláte asi v 99%), vrátí funkce sumu prvků v oblasti dané iterátory `začátek` a `konec`.
- `template <class InputIterator, class Typ, class BinaryOperation operace> Typ accumulate (InputIterator zacatek, InputIterator konec, Typ pocatecniHodnota, BinaryOperation operace);` - Obdobně jako předchozí. Rozdíl je v tom, že tato varianta `accumulate` nebude sčítat, ale provede binární operaci zadanou programátorem.

U algoritmu `accumulate` musím upozornit, že jeho deklarace se nenachází v hlavičkovém souboru `algorithm`, ale v `numeric`. Vše si ukážeme na jednoduchém příkladu. Úmyslně zde střídám pole s vektorem (kontejnerem), aby bylo zřejmé, že algoritmy lze použít i na obyčejné pole.

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

```
using namespace std;

bool podminka(int a)
{
    return (20 < a) && (a < 80);
}

int main()
{
    int pole[10] = { 12, 80, 34, 5, 23, 60, 82, 80, 9, 10 };
    vector<int> vektor(pole,&pole[10]);
    int pocet = 0;
    cout << "Počet výskytů čísla 80: ";
    count(vektor.begin(), vektor.end(), 80, pocet);
    cout << pocet << endl << "Počet čísel 20 < x < 80: ";
    pocet = 0; // Nezapomeňte na to!
    count_if(pole,&pole[10],podminka,pocet);
    cout << pocet << endl << "Součet všech prvků je: ";
    int soucet = accumulate(vektor.begin(), vektor.end(), 0);
    cout << soucet << endl << "Součin prvních 3 prvků :";
    int soucin = accumulate(pole, &pole[3], 1, times<int>());
    cout << soucin << endl;
    return 0;
}
```

Algoritmus for_each v C++

Často jsem se setkal s mylným tvrzením, že v C++ neexistuje `for_each`, nebo nějaká jeho obdoba. Hlavně velmi často v článcích (tištěných i na internetu) o jiných programovacích jazycích autoři srovnávají daný jazyk s C++. Vždy při tom prohlásí, že jednou z nevýhod C++ oproti jazyku, o kterém píšou, je absence `for_each`. Snad se mi podaří tuto nepravdu vyvrátit. Algoritmus `for_each` provede pro všechny prvky v zadaném rozmezí nějakou zadanou operaci. Deklarace:
`template void for_each(InputIterator zacatek, InputIterator konec, TFunkce f);` - Parametry šablony jsou typ iterátoru a typ funktoru (typ funkce, nebo třída funkčních objektů). Parametry funkce jsou začátek a konec oblasti a ukazatel na funkci, nebo funkční objekt. Vrací-li funkce, či operátor() funkčního objektu nějakou hodnotu, bude ignorována. Funkce, nebo operátor() funkčního objektu mohou při svém volání udělat nějaký "vedlejší efekt" jako třeba změna globální proměnné, atributu objektu, nebo poslání dat do datového proudu. Ukážeme si na příkladu použití `for_each`. Vytvoříme si nějaké body v 2D, které dáme do kontejneru. Tyto body bude přesouvat.

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

using namespace std;
```

```
void funkce(int a)
{
         cout << "Je volána funkce s parametrem " << a << endl;
}

class Bod
{
    private:
       int X,Y;
    public:
       Bod() : X(0), Y(0) {}
       Bod(int x, int y) : X(x), Y(y) {}
       void nastavX(int x) { X = x; }
       void nastavY(int y) { Y = y; }
       int dejX() const { return X; }
       int dejY() const { return Y; }
};

class Posun
{
    private:
         int X,Y;
    public:
        Posun() : X(0), Y(0) {}
        Posun(int x, int y) : X(x), Y(y) {}
        void nastavX(int x) { X = x; }
        void nastavY(int y) { Y = y; }
        int dejX() const { return X; }
        int dejY() const { return Y; }
        void operator()(Bod &b) const
        {
            b.nastavX(b.dejX() + this->dejX());
            b.nastavY(b.dejY() + this->dejY());
        }
};

ostream &operator<<(ostream &o, const Bod &b)
{
    o << "Bod: << b.dejX() << " Y = " << b.dejY();
    return o;
}

int main()
{
    int pole[7] = {1 , 2, 100, 23, 43, 56, 75 };
    for_each(pole,&pole[7],funkce);
```

```
    vector<Bod> body;
    Bod b1(0,0), b2(10,10), b3(-100, 1000), b4(10,7);
    body.push_back(b1);
    body.push_back(b2);
    body.push_back(b3);
    body.push_back(b4);
    copy(body.begin(),body.end(),ostream_iterator<Bod>(cout,"\n"));
    cout << endl;
    /* Teď posuneme body o 10 jednotek na ose x i y */
    Posun posun(10,10);
    for_each(body.begin(),body.end(),posun);
    copy(body.begin(),body.end(),ostream_iterator<Bod>(cout,"\n"));
    cout << endl;
    /* Teď posuneme body o -10 jednotek na ose x i y. Vrátíme je zpět*/
    posun.nastavX(-10);
    posun.nastavY(-10);
    for_each(body.begin(),body.end(),posun);
    copy(body.begin(),body.end(),ostream_iterator<Bod>(cout,"\n"));
    cout << endl;
    return 0;
}
```

Určitě je dobré si pro procvičení tento příklad rozšířit. Pomocí znalostí, které již máme s tohoto a předchozích článků můžeme například zjistit kolik bodů je ve kterém kvadrantu. Vyhledat body v daném kvadrantu, nebo v dané oblasti, atd... Všem, kteří nemají příliš zkušenosti s používáním standardních algoritmů toto doporučuji jako cvičení. Konečně nemáme v kontejneru `int`, `char`, nebo jiné primitivní datové typy.
Funkce, nebo operátor () funkčního objektu mohou ve svém těle volat i nekonstantní metody. V mém příkladě volám metody `nastavX` a `nastavY`. Pomocí `for_each` tedy lze měnit prvky v kontejneru. Někdy by bylo ale lepší, kdyby algoritmus na základě svého argumentu vytvořil nový objekt (přetransformoval parametr), který by byl vložen do kontejneru místo původního. Takový algoritmus v C++ existuje. Jmenuje se `transform`. Transformačních algoritmů je v C++ více. Budeme se jim věnovat v příštím článku.