ლაბორატორიული მეცადინება 3: **ForwardIterator - ის სპეციფიკა**

განსახილველი საკითხები:

- მარტივი მაგალითი single pass იტერატორების სპეციფიკაზე
- დიაპაზონში მასიმალური ელემენტის განსაზღვრის ალგორითმი >>>
- ფუნქტორის (ფუნქციისმაგვარი ობიექტის) შექმნა სხვა ფუნქციის შიგნით >>>
- სავარჯიშოები >>>

მარტივი მაგალითი single pass იტერატორების სპეციფიკაზე

`single pass` იტერატორები ეწოდება Input და Output იტერატორებს, რადგან შეგვიძლია იტერატორის დამახსოვრება, მაგრამ დამახსოვრებული იტერატორის ცვლილება არაპროგნოზირებადია და მისი მნიშვნელობები ნახტომისებურად იცვლება. მაგალითად, ქვემოთ ჩვენ განვიხილავთ იტერატორების დიაპაზონში მაქსიმალური ელემენტის იტერატორის განსაზღვრის ალგორითმს. თუ ამ ალგორითმს გამოვიყენებთ შემავალ ნაკადზე, მაშინ მის მიერ დაბრუნებული იტერატორი ამ ნაკადში მაქსიმალურის მომდევნო ობიექტებზე ვერ გადავა (გარდა იმ ტრივიალური შემთხვევისა, როდესაც იგი დიაპაზონის ბოლოში აღმოჩნდება შემთხვევით).

მარტივ მაგალითზე ვნახოთ, თუ რა შეიძლება იყოს მიზეზი. ვთქვათ, ფაილში წერია რიცხვები

```
1  2  3  4  5  45  32  4  47  12  55  32
```

პროგრამას აქვს სახე:

```
#include <iostream>
#include <iterator>
using namespace std;

int main()
{
	ifstream ifs("data.txt");
	std::istream_iterator<int> i(ifs),j;     // stdin iterator
	j = i;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;
	++i;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;
	++i;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;

	++j ;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;
	++j;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;
	++j;
	cout << "*i=" << *i << '\t' << "*j=" << *j << endl;

}
```

შედეგად მივიღებთ მარჯვნივ მოთავსებული ტექსტური არის შიგთავსს. როგორც ვხედავთ, ვიდრე საწყისი იტერატორი იცვლება, დამახსოვრებული ინარჩუნებს თავის მნიშვნელობას. როგორც კი დამახსოვრებული იწყებს ზრდას, იგი მაშინვე ხტება საწყისი იტერატორის ბოლო მნიშვნელობაზე. შეგვიძლია რიგრიგობით ვცვალოთ ისინი და დავაკვირდეთ კანონზომიერებას.

```
*i=1  *j=1
*i=2  *j=1
*i=3  *j=1
*i=3  *j=4
*i=3  *j=5
*i=3  *j=45
Press any key to continue. . .
```

<<< დიაპაზონში მაქსიმალური ელემენტის განსაზღვრის ალგორითმი.

საქაღალდეში "..\muLibrary, რომელიც წინა მეცადინეობაზე შევქმენით, არის ფაილი რამდენიმე ალგორითმით. იგივე ფაილში ჩავამატოთ ალგორითმი

```
template <class ForwardIterator>
ForwardIterator my_max_element(ForwardIterator first, ForwardIterator last)
{
	if (first == last) return last;
	ForwardIterator largest = first;

	while (++first != last)
	if (*largest<*first)    // or: if (comp(*largest,*first)) for version (2)
		largest = first;
	return largest;
}
```

აგრეთვე მისი გადატვირთული ვარიანტი, რომელიც მესამე არგუმენტად მიიღებს ბინარულ პრედიკატს. ამჯერადაც, პროექტში ამ ფუნქციების გამოყენებითვის მივუთითოთ დამატებითი ბიბლიოთეკა და თავსართი ფაილი როგორც წინა მეცადინეობაზე.

თუ ასეთი საქაღალდე და შიგ ალგორითმების ფაილი არაა კომპიუტერზე, მაშინ ახლა შევქმნათ (ფაილს ვუწოდოთ \„myAlgorithms.h“ ან რაიმე მსგავსი).

იგივე ფაილში შეინახეთ ალგორითმი

```
template<typename InputIterator>
void printRange(InputIterator first, InputIterator last)
{
	while (first != last)
	{
		std::cout << *first << endl;
		++first;
	}
}
```

და მისი გადატვირთული ვარიანტი, რომელიც მოცემული დიაპაზონიდან დაბეჭდავს მხოლოდ გარკვეული თვისების მქონე ელემენტებს.

<<< **ფუნქტორის (ფუნქციისმაგვარი ობიექტის) შექმნა სხვა ფუნქციის შიგნით.**

ამოცანა 1. /დიაპაზონიდან გარკვეული თვისების მქონე ელემენტების შეცვლა/. წარმოვიდგინოთ, რომ ნამდვილი რიცხვების დიაპაზონიდან გვინდა გარკვეული თვისების მქონე ელემენტების შეცვლა. თვისება და შესაცვლელი მნიშვნელობა წარმოადგენენ ალგორითმის პარამეტრებს. პრედიკატის გაკეთება შეიძლება რამდენიმენაირად: ლამბდა ფუნქციით, დამმაგრებელით და ფუნქტორით. ალგორითმს და პროგრამა დრაივერს შესაძლოა ჰქონდეს შემდეგი სახე

```
#include <iostream>
#include <vector>
#include <list>
#include <forward_list>
#include <functional>                  //bind2nd
#include <string>
using namespace std;

template<typename ForwardIterator, typename T, typename Predicate>
void myReplace_if
(
	ForwardIterator first,
	ForwardIterator last,
	Predicate g,
	const T& new_value
)
{
	while (first != last) {
		if (g(*first)) // L1
```

```
                *first = new_value; // L2
            ++first;
        }
}
int main() {
        vector<int> v = { 11, 24, 541, 43, -23 };
        myReplace_if(v.begin(), v.end(), bind2nd(greater<int>(), 100), 100);
        for (auto e : v) cout << e << "  ";
        cout << endl << endl;

        list<double> lst = { 11.88, 21.4, 541.9 };
        myReplace_if(lst.begin(), lst.end(), [](double x) {return x > 100; }, 100);
        for (auto e : lst) cout << e << "  ";
        cout << endl << endl;

        cout << "Function object:" << endl;
        struct isLowerCaseChar {
                bool operator() (const string& a)
                {
                        return(a[0] > 'a' && a[a.size() - 1] <= 'z');
                }
        };

        forward_list<string> s{ "Nino", "Khatia", "marth", "Joy" };
        myReplace_if(s.begin(), s.end(), isLowerCaseChar(), "Replaced");
        for (auto e : s) cout << e << "  ";
        cout << endl << endl;

}
```

<<< სავარჯიშოები.

1. შექმნენით ალგორითმი, რომელიც მიწოდებულ დიაპაზონში იპოვის გარკვეული თვისებების მქონე ელემენტებს შორის მაქსიმალურს. გასინჯეთ იგი სხვადასხვა დიაპაზონზე სხვადასხვა მეთოდით შექმნილი პრედიკატებისთვის.
2. შექმნენით ალგორითმი, რომელიც მოცემული დიაპაზონიდან დაბეჭდავს მხოლოდ გარკვეული თვისების მქონე ელემენტებს. გასინჯეთ იგი სხვადასხვა დიაპაზონზე. გასინჯეთ იგი სხვადასხვა დიაპაზონზე სხვადასხვა მეთოდით შექმნილი პრედიკატებისთვის.
3. ამოცანა 1-ის ალგორითმი აგრეთვე შეინახეთ ალგორითმების ფაილში. შემდეგ შექმენით პროგრამა, რომელიც ისარგებლებს ამ „გარე ბიბლიოთეკით“.
4. განხილულ ამოცანებში გამოიყენეთ სტანდარტული ბიბლიოთეკის ალგორითმები.
5. ვთქვათ, ფაილში ”lakes.txt” ფაილში წერია რამდენიმე ტბის მონაცემი:

```
Paravani      37.5   2073   3.3
Paliastomi    18.2   -0.3   3.2
Tabackuri     14.2   1997   40.2
Jandari       10.6   291    7.2
Ritsa         1.49   884    101
Bazaleti      1.22   879    7
```

პირველი ველი სახელია, მეორე ფართობი კვადრატულ კილომეტრებში, მესამე - სიმაღლე ზღვის დონიდან, ხოლო მეოთხე არის სიღრმე (ბოლო ორი მეტრებსი იზომება).

ვთქვათ, ფაილში “lake.h“ მოთავსებულია განაცხადი კლასზე:

```
#pragma once
class lake
{
public:
 string name;
 double area;        // km^2
 double height;      // meter
 double depth;       // meter
 lake(void);
 ~lake(void);
 lake(istream& is);
 bool operator!=(const lake& other);
 bool operator<(const lake& other);
 friend istream& operator>>(istream& is, lake& x);
 friend ostream& operator<<(ostream& os, const lake& x);
};
```

ხოლო “lake.cpp“ ფაილში არის იმპლემენტაცია:

```
#include<iostream>
#include<string>
using namespace std;
#include "lake.h"

lake::lake(void)
: area(0)
, height(0)
, depth(0)
{
}
lake::~lake(void)
{
}
lake::lake(istream& is)
{
 is >> name >> area >> height >> depth;
}
bool lake::operator!=(const lake& other)
{
 return (name != other.name
        || area != other.area
        || height != other.height
        || depth != other.depth);
}
bool lake::operator<(const lake& other)
{
 return (height<other.height);
}
istream& operator>>(istream& is, lake& x)
{
 is >> x.name >> x.area >> x.height >> x.depth;
 return is;
}
ostream& operator<<(ostream& os, const lake& x)
{

 os << x.name << '\t' <<   x.area << '\t' <<
       x.height << '\t' << x.depth << '\t' << endl;
 return os;
}
```

პროგრამა დრაივერი:

```
#include<iterator>
#include<fstream>
#include<algorithm>
#include<vector>
using namespace std;
#include "Lake.h"
#include "maximal.h"

int main()
{
   vector<lake> v;
   lake tmp;
   ifstream ifs("lakes.txt");
   while (ifs >> tmp)
         v.push_back(tmp);

   auto it = my_max_element(v.begin(), v.end());
   cout << "max lake: " << *it << endl;

   ostream_iterator<lake> ou(cout);
   copy(v.begin(), v.end(), ou);
   cout << endl;

   cout << "100 m-ze magla mdebare tbebis raodenoba:" << endl;
   int k = count_if(v.begin(), v.end(),
         [](const lake& a){return (a.height > 100); });
   cout << k << endl;
}
```

იყენებს ამ ფაილებს და ფაილში ჩაწერილ ალგორითმებს, რომლებიც არაა პროექტის ნაწილი.

გაარკვიეთ რას აკეთებს პროგრამა. შემდეგ, ჩვენს მიერ შექმნილი ალგორითმები ჩაანაცვლეთ მათი ეკვივალენტებით სტანდარტული ბიბლიოთეკიდან.

6. ტბების შესაძლებელია კონტეინერისთვის <algorithm> ბიბლიოთეკის სხვა ალგორითმების გამოყენება. გასინჯეთ რამდენიმე ჩვენთვის ცნობილი ალგორითმი და შედეგები გამობეჭდეთ ეკრანზე.

7. იგივე შედეგი შეგვიძლია მივიღოთ, თუ გამოვიყენებთ მასივს და არა ვექტორს. განსაკუთრებით იმ შემთხვევაში, თუ მონაცემების ფაილში ტბების აღმწერი სტრიქონების წინ დაბეჭდილი იქნება ტბების რაოდენობა.

8. შემდეგი მონაცემებისთვის შექმენით კლასი. შემდეგ სტრიქონები ჩაწერეთ ობიექტების რაიმე კონტეინერში და მოძებნეთ მაქსიმალური ელემენტები გარკვეული ველის მიხედვით.

ცნობები მსოფლიოში მომხდარი რამდენიმე დამანგრეველი მიწისძვირის სიმძლავრისა და დროის შესახებ //სტუდენტების შემოთავაზება, ავტორი ნ. შავერდაშვილი/:

```
8
Indonesia        8.5    2007
Tibet            8.6    1950
KurilIslands     8.5    1963
Chile            9.5    1960
Alaska           9.2    1964
Japan            9.0    2011
Ecuador          8.8    1906
Sumatra          9.1    2004
```