ლაბ 8: set, map კონტეინერები

განსახილველი საკითხები:

- სიმრავლის და მულტისიმრავლის ზოგიერთი ალგორითმი STL -იდან
- მაგალითი 1 / ახალი ელემენტის ჩამატება
- მაგალითი 2 /set კონტეინერის შესახებ/ >>>
- მაგალითი 3 /map კონტეინერის შესახებ/ >>>
- მაგალითი 4 /set კონტეინერი თარგის ორი პარამეტრით/ >>>
- სავარჯიშოები >>>

სიმრავლის (და მულტი სიმრავლის) ზოგიერთი ალგორითმის აღწერა:

- ასლის კონსტრუქტორი: თუ სიმრავლეზე განაცხადის გაკეთების მომენტში ახალ ობიექტს ფრჩხილებში მივუწერთ რაიმე სხვა კონტეინერის `[first, last)` ფრაგმენტს, ამ ფრაგმენტიდან ობიექტების ასლები ჩაემატება ახალ სიმრავლეში (და განთავსდებიან გასაღებების სიდიდის მიხედვით).
- სიმრავლეს აქვს ახალი ელემენტის ჩამატების მეთოდი `insert();`
- აქვს ელემენტის წაშლის მეთოდი.
- ძებნისთვის შეგვიძლია გამოვიყენოთ ალგორითმი `find`, რომელიც წრფივ დროში ეძებს, და შეგვიძლია გამოვიყენოთ იგივე სახელის მქონე სიმრავლის კლასის მეთოდი, რომელიც ლოგარითმულ დროში ეძებს. ეს უკანასკნელი მოქმედებს თითქმის ზუსტად ისე, როგორც პრაქტიკულ მეცადინეობებზე განხილული search ალგორითმი.

მაგალითი 1 / ახალი ელემენტის ჩამატება. განვიხილოთ ახალი ელემენტის ჩამატების ფუნქციები http://www.cplusplus.com/reference/set/set/insert/-ის მიხედვით.

```
                        pair<iterator,bool> insert (const value_type& val);
    single element (1)  pair<iterator,bool> insert (value_type&& val);

                        iterator insert (const_iterator position, const value_type& val);
         with hint (2)  iterator insert (const_iterator position, value_type&& val);

                        template <class InputIterator>
             range (3)    void insert (InputIterator first, InputIterator last);

  initializer list (4)  void insert (initializer_list<value_type> il);
```

(1)-ში, მეორე არის გადაადგილებადი ტიპებისგან შედგენილი სიმრავლისთვის. (1) აბრუნებს წყვილს, რომლის პირველი კომპონენტი ან ახალ ჩამატებულ ელემენტს (ამ დროს მეორე კომპონენტი ჭეშმარიტია), ან მის ეკვივალენტურს, რომელიც უკვე იყო სიმრავლეში (რადგან ელემენტების დუბლირება აკრძალულია) და ამ დროს მეორე კომპონენტი მცდარია. (2)-ის მეორე ვარიანტიც გადაადგილებადი ტიპებისთვისაა. ეს აბრუნებს იტერატორს ან ახალ ელემენტზე, ან სიმრავლის იმ ელემენტზე, რომელსაც აქვს იგივე მნიშვნელობა.

```
// set::insert (C++98)
#include <iostream>
#include <set>

int main()
{
	std::set<int> myset;
	std::set<int>::iterator it;
	std::pair<std::set<int>::iterator, bool> ret;

	// set some initial values:
	for (int i = 1; i <= 5; ++i) myset.insert(i * 10);    // set: 10 20 30 40 50

	ret = myset.insert(20);               // no new element inserted
```

```
	if (ret.second == false) it = ret.first;  // "it" now points to element 20

	myset.insert(it, 25);                 // max efficiency inserting
	myset.insert(it, 24);                 // max efficiency inserting
	myset.insert(it, 26);                 // no max efficiency inserting

	int myints[] = { 5,10,15 };              // 10 already in set, not inserted
	myset.insert(myints, myints + 3);

	std::cout << "myset contains:";
	for (it = myset.begin(); it != myset.end(); ++it)
		std::cout << ' ' << *it;
	std::cout << '\n';
}
```

<<< **მაგალითი 2** /set კონტეინერის შესახებ/. განვიხილოთ შემდეგი პროგრამა, რომელიც ანხორციელებს შემდეგ მოქმედებებს:

1. ქმნის მთელი რიცხვებისთვის სიას რამდენიმე რიცხვით;
2. იქმნება მთელი რიცხვების ახალი სიმრავლე და იგი ინიციალიზდება სიის ელემენტებით ასლის კონსტრუქტორის საშუალებით;
3. ამავე სიმრავლეში ემატება ორი სხვა რიცხვი;
4. ამ სიმრავლიდან წაიშლება ახლად ჩამატებული რიცხვები;
5. ორივე მეთოდით იძებნება სიმრავლის ელემენტი და, თუ არსებობს, წაიშლება.

ყოველი მოქმედების შემდეგ, იბეჭდება სიმრავლის შიგთავსი. შეგვიძლია დავრწმუნდეთ, რომ სიიდან სიმრავლეში ასლის გადაღების შემდეგ რიცხვები გამოიბეჭდება დახარისხებული სახით, რადგან სიმრავლე იმპლემენტირებულია ძებნის ორობითი ხის საფუძველზე.

```
#include<iostream>
#include<list>
#include<set>
#include<iterator>

using namespace std;

int main()
{
	list<int> l { 11, 23, 1, 5, 143, -12 };

	ostream_iterator<int> out(cout, "  ");
	cout << "List:" << endl;
	copy(l.begin(), l.end(), out);
	cout << endl;

	cout << "After using set's copy constructor:" << endl;
	set<int> s(l.begin(), l.end());
	copy(s.begin(), s.end(), out);
	cout << endl;

	s.insert(22);
	s.insert(222);
	cout << "After inserting:" << endl;
	copy(s.begin(), s.end(), out);
	cout << endl;

	s.erase(22);
	s.erase(222);
```

```
	cout << "After erasing:" << endl;
	copy(s.begin(), s.end(), out);
	cout << endl;

	auto it = find(s.begin(), s.end(), 5);
	if (it != s.end())
	{
		cout << *it << "  found and erased!" << endl;
		s.erase(*it);
	}
	else
		cout << 5 << "  not found!" << endl;
	copy(s.begin(), s.end(), out);
	cout << endl;

	it = s.find(143);
	if (it != s.end())
	{
		cout << *it << "  found and erased!" << endl;
		s.erase(*it);
	}
	else
		cout << 143 << "  not found!" << endl;
	copy(s.begin(), s.end(), out);
	cout << endl;
}
```

შედეგი:

```
List:
11  23  1  5  143  -12
After using set's copy
constructor:
-12  1  5  11  23  143
After inserting:
-12  1  5  11  22  23  143  222
After erasing:
-12  1  5  11  23  143
5  found and erased!
-12  1  11  23  143
143  found and erased!
-12  1  11  23
Press any key to continue . . .
```

გასათვალისწინებელია, რომ თუ გადავანაცვლებთ შემდეგ ორ სტრიქონს

```
		cout << *it << "  found and erased!" << endl;
		s.erase(*it);
```

მივიღებთ შეცდომის შესახებ გზავნილს, რადგან ვცდილობთ მნიშვნელობის ამოღებას გაუქმებული იტერატორიდან.

<<< **მაგალითი 3** /map კონტეინერის შესახებ/. ეს კონტეინერი საკმაოდ სპეციფიკურია. სიმრავლისგან განსხვავებით, ასახვაში ინახება წყვილები, მათგან პირველი არის გასაღები, რომლის მიხედვითაც ხდება წყვილის ჩასმა ასახვაში. საკმარისია ვთქვათ, რომ ასახვაში ახალი (a,b) ელემენტის ჩასაგდებად საკმარისია m[a] = b; განაცხადის გაკეთება. თუმცა, ასახვის ასაგებად მარტივი გზაა არჩეული. შემდეგი მაგალითი გვიჩვენებს, რომ ეს კონტეინერი წარმოადგენს სიმრავლისა და წყვილის კომპოზიციას.

```
#include<iostream>
#include<fstream>
#include<set>
```

```
#include<map>
#include<string>

using namespace std;

int main()
{
	map<string, int> m;
	{
		ifstream ifs("data.txt");
		string a; int b;
		while (ifs >> a >> b)
			m[a] = b;
	}
	for (auto i = m.begin(); i != m.end(); ++i)
		cout << i->first << "  " << i->second << endl;

	cout << endl << "Do you see difference?" << endl;

	set<pair<string, int>> s;
	{
		ifstream ifs("data.txt");
		string a; int b;
		while (ifs >> a >> b)
			s.insert(pair<string, int>(a, b));
	}
	for (auto i = s.begin(); i != s.end(); ++i)
		cout << i->first << "  " << i->second << endl;
}
```

<<< **მაგალითი 4** /set კონტეინერი თარგის ორი პარამეტრით/. set კონტეინერი საკმაოდ უცნაურად არის მოწყობილი. თუ მასში ჩავყრით ობიექტებს, მაშინ უნდა ვიზრუნოთ რომ ობიექტების შედარებაც ვასწავლოთ. ამას აკეთებს თარგის მეორე პარამეტრი. ამ შემთხვევაში კიდევ უფრო უცნაურობა (ერთი შეხედვით) არის სიმრავლეში ობიექტის ძებნა. მაგალითად, განვიხილოთ კოდი:

```
#include<iostream>
#include<fstream>
#include<set>
#include<string>
using namespace std;

struct cmpBySecond {
	bool operator()(const pair<string,int> & a, const pair<string, int>& b) const {
		return a.second < b.second;
	}
};

int main()
{
	set<pair<string, int>, cmpBySecond> t;
	{
		ifstream ifs("data.txt");
		string a; int b;
		while (ifs >> a >> b)
			t.insert(make_pair(a, b));
	}
	for (auto m : t)
		cout << m.first << "   " << m.second << endl;
	auto i = t.find(make_pair("", 88));
	if(i != t.end())
		cout << endl << i->first << "   " << i->second << endl;
```

}

როგორც ვხედავთ, სიმრავლის ობიექტის განაცხადში არის გადაწოდებული კლასის სახელი, რომელიც სინამდვილეში შედარების ფუნქტორს წარმოადგენს. მისი შინაარსი მარტივია- გასაღების როლს წყვილის მეორე წევრი ასრულებს. ახლა, თუ მოვინდომებთ რომ მოვძებნოთ ისეთი წყვილი, რომლის მეორე კომპონენტი 88-ის ტოლია, საჭიროა შევქმნათ რაიმე წყვილი მეორე კომპონენტით 88. ნებისმიერი წყვილი 88-ის ტოლი მეორე კომპონენტით ამისი ტოლია ჩვენს მიერ შექმნილი შედარების ფუნქციის აზრით (რადგან არც ნაკლებია მასზე და არც ტოლი).

<<< სავარჯიშოები:

1. მეორე ამოცანაში, სიმრავლის ნაცვლად გააკეთეთ განაცხადი მულტისიმრავლეზე

   ```
   multiset<int> s(1.begin(), 1.end());
   ```

   შემდეგ, `it = s.find(143);` შეტყობინების წინ რამდენჯერმე გაიმეორეთ 143-ის ჩამატების `s.insert(143);` შეტყობინება. დასასრულ, ერთმანეთს შეადარეთ `s.erase(*it);` და `s.erase(it);` გამოყენების შედეგები. `s.erase(*it);` წაშლის ყველა `*it` -ს. `s.erase(it);` წაშლის მხოლოდ იმ მნიშვნელობას, რომელიც ამ იტერატორზეა დამაგრებული.

2. http://www.cplusplus.com/reference/set/set/erase/-ში მოცემულია სიმრავლიდან ელემენტის წაშლის ფუნქციების განმარტება და მოყვანილია თვალსაჩინო მაგალითი. გაარჩიეთ მასალა.

3. შემდეგი პროგრამა გვიჩვენებს ასახვის კონტეინერის (Map) საშუალებების ნაწილს. ჩვენ სამნაირად ვითვლით რეკურსიული

   $$f(i) = 1 \text{ თუ } i < 0 \quad \text{და} \quad f(i) = f(i-11) + f(i-22) \text{ თუ } i >= 0$$

   ფუნქციის მნიშვნელობას როცა `i= 390.` რეკურსიული ფუნქცია ისეა შერჩეული, რომ მისი შტოები იმეორებს ერთმანეთს და შორსაა ე.წ. ”დაყავი და დაიმორჩილე” შემთხვევისგან. უშუალოდ განმარტების საფუძველზე ითვლება პროგრამის მესამე ნაწილში. პირველ ნაწილში ამ ფუნქციის მნიშვნელობები ითვლება დაწყებული უმარტივესი შემთხვევებიდან ვიდრე არ მიიღწევა სასურველი მნიშვნელობა.

   მეორე ნაწილში ვიყენებთ ასახვას და ირიბ რეკურსიას. ასახვაში დამახსოვრებული მნიშვნელობებიდან ვცდილობთ ამოვიღოთ საჭირო მნიშვნელობა. თუ ასეთი ჯერ არ შეგვხვედრია, მაშინ გამოითვლება და ჩაეწერება ასახვაში.

```
#include<iostream>
#include<vector>
#include<map>
#include<time.h>
using namespace std;

int f(int i);
int f(int i, map<int,int> & a);
int read(int i, map<int,int> & a);
const int N = 390;

int main()
{
    clock_t st, fin;
      st = clock();
    vector<int> v(N+1);
    for(int i=0; i<v.size(); i++)
    {
         int a,b;
         if(i-11 >= 0)
             a=v[i-11];
```

```
            else
                a = 1;
            if(i-22 >=0)
                b=v[i-22];
            else
                b = 1;
            v[i] = a+b;
        }
        fin = clock();
        cout<<"\n veqtoris DRO: "<<((double)(fin-st))/CLOCKS_PER_SEC<<endl<<endl;

        for(int i=0; i<v.size(); i++)
        {
            cout << "i=" <<i << "\t" << "v[" << i << "]=" <<v[i] <<endl;
        }
        map<int,int> m;
        cout << endl << endl;
        system("PAUSE");
        st = clock();
        m[N]=f(N,m);
        fin = clock();
        cout<<"\n asaxvis DRO: "<<((double)(fin-st))/CLOCKS_PER_SEC<<endl<<endl;
        map<int,int>::iterator it;
        for(it = m.begin(); it != m.end(); it++)
        {
            cout<<"i="<<it->first << "\t" << "m[" << it->first << "]=" <<it->second <<endl;
        }
        cout << endl << endl;
        system("PAUSE");
        st = clock();
        cout << "esaa martivi rekursiit: "<< f(N)<<endl;
        fin = clock();
        cout<<"martivi rekursiis dro "<<((double)(fin-st))/CLOCKS_PER_SEC <<endl;
}
int f(int i){
    if(i<0) return 1;
    return f(i-11)+f(i-22);
}
int f(int i, map<int,int> & a){
    if(i<0) return 1;
    return read(i-11,a)+read(i-22,a);
}
int read(int i, map<int,int> & a){
    if(i<0) return 1;
    if(a.find(i)!=a.end())
        return a[i];
    return a[i]=f(i,a);
}
```

იგივე მეთოდით, გამოთვალეთ მსგავსი რეკურსიული ფუნქციების მნიშვნელობები, რომლებისთვისაც აუცილებელია მნიშვნელობების დამახსოვრება, მაგალითად ფიბონაჩის რიცხვები.