ლაბ 12: ორგანზომილებიანი კონტეინერები. ჰეშირების ელემენტები

განსახილველი საკითხები:

- **ამოცანა 1** /ფაილიდან დაკბილული ცხრილების წაკითხვა სტრინგების ნაკადების გამოყენებით/
- **ამოცანა 2** /ორგანზომილებიანი ვექტორის შექმნაში move ალგორითმის გამოყენების შემთხვევა/ >>>
- **ამოცანა 3** / `unordered_set` ტიპის ობიექტის შექმნა და რამდენიმე მარტივი ოპერაცია/ >>>
- **სავარჯიშოები** >>>

მრავალგანზომილებიანი კონტეინერების დანიშნულებას წარმოადგენს ცხრილების (ორგანზომილებიანი მონაცემების), ან უფრო მაღალი განზომილების მონაცემების შენახვა და ეფექტური დამუშავება. გასათვალისწინებელია, რომ ზოგ შემთხვევებში მონაცემები არაა მართკუთხა ფორმის. ორგანზომილებიან შემთხვევებში, ზოგადი ფორმის ცხრილებისთვის შეგვიძლია გამოვიყენოთ კონტეინერების „წმინდა და შერეული“ კომბინაციები: ვექტორების ვექტორი, სიების სია, ვექტორების სია, სიების ვექტორი, სიების დეკი და ა.შ.. ამ ტიპის **ცარიელ ორგანზომილებიან კონტეინერები** იქნება შემდეგი განაცხადებით:

```
vector<vector<string>> A;
list<list<double>> L;
vector<list<int>> H;
```

მართკუთხა ფორმის მონაცემების შენახვისთვის წარმატებით შეგვიძლია გამოვიყენოთ აგრეთვე ორგანზომილებიანი მასივი. STL-ში, ჰეშირების საშუალებით შექმნილი ასოცირებული (დაუხარისხებელი) კონტეინერები იყენებს მასივების სიას.

ამოცანა 1 /ფაილიდან დაკბილული ცხრილების წაკითხვა სტრინგების ნაკადებით/.

ვთქვათ, ფაილში “data.txt” წერია არამართკუთხა ფორმის ცხრილი:

```
//data.txt:
 32    123    312    12     3    31   12
 32    1      4      43     123
 -312  -231   -23    1      31
 32    321
 231   321    312    231
```

შევქმნათ პროგრამა, რომელიც ამ მონაცემებს შეინახავს ვექტორების ვექტორში.

ამოხსნა. მონაცემები შეგვიძლია წავიკითხოთ სტრიქონ-სტრიქონ, მაგრამ სტრინგებში. შემდეგ გამოვიყენოთ სტრინგიდან შეტანის ნაკადი (istringstream). მათი დახმარებითაც მოცემული სტრინგიდან მონაცემები ჩვწეროთ საჭირო ტიპის ვექტორში. ეს ნაწილი ფაქტიურად არ განსხვავდება ფაილიდან ვექტორში რიცხვების ჩაწერის ჩვენთვის ცნობილი ტექნიკისგან. შესაბამის პროგრამას აქვს სახე:

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

#include<sstream>
using namespace std;

template<typename InputIterator>
void showVect(InputIterator first, InputIterator last);

template<typename InputIterator>
void showMatr(InputIterator first, InputIterator last);
int main()
{
	vector<vector<int> > m;
	string s;
	ifstream ifs("data.txt");
	while (getline(ifs, s))
```

```
	{
		vector<int> tmp;
		int n;
		istringstream iss(s);
		while (iss >> n)
			tmp.push_back(n);
		m.push_back(tmp);
	}
	showMatr(m.begin(),m.end());
}
template<typename InputIterator>
void showVect(InputIterator first, InputIterator last)
{
	while (first != last)
	{
		cout << *first << '\t';
		first++;
	}
	cout << endl;
}
template<typename InputIterator>
void showMatr(InputIterator first, InputIterator last)
{
	while (first != last)
	{
		showVect(first->begin(), first->end());
		first++;
	}
	cout << endl;
}
```

#include<sstream> დირექტივის საშუალებით ირთვება სტრინგებთან სამუშო ნაკადები. ყურადღება მივაქციოთ, რომ ორაზროვნების თავიდან ასაცილებლად, შემოკლებები ნაკლებად გამოიყენება. მაგ. istringstream. ყურადღება მივაქციოთ, თუ როგორ ხდება მატრიცის ბეჭდვა.

<<< **ამოცანა 2** /ორგანზომილებიანი ვექტორის შექმნაში move ოპერაციის გამოყენების შემთხვევა/. 2011 წლის სტანდარტი საშუალებას იძლევა გამოვიყენოთ ე.წ. move სემანტიკა. განვიხილოთ მარტივი პროგრამა:

```
#include<iostream>
#include<vector>
#include <ctime>
#include <ratio>
#include <chrono>

using namespace std;
using namespace std::chrono;

int main()
{
   steady_clock::time_point t1 = steady_clock::now();

   vector<vector<int>> matrix;
   for (int i = 0; i < 1000; ++i)
   {
      vector<int> tmp;
      for (int j = 0; j < 1600; ++j)
            tmp.push_back(rand());
      matrix.push_back(move(tmp));
      //matrix.push_back(tmp);
   }
```

```
    steady_clock::time_point t2 = steady_clock::now();
    duration<double> time_span = duration_cast<duration<double>>(t2 - t1);

    std::cout << "It took me " << time_span.count() << " seconds.";
    std::cout << std::endl;
}
```

თუ გავშვებთ პროგრამას, შემდეგ გადავაადგილებთ კომენტარის ნიშანს შემდეგ ორ სტრიქონს შორის

```
        matrix.push_back(move(tmp));
        //matrix.push_back(tmp);
```

და შევადარებთ პროგრამის მუშაობის დროს, დიდი ზომის ცხრილებზე შევნიშნავთ გარკვეულ განსხვავებას.

სხვასახვა ამოცანის ამოხსნისას ხშირია შემთხვევა, როდესაც ისეთი ობიექტების ასლის გაკეთება ხდება საჭირო, რომლებიც შემდეგში აღარ გამოიყენება. ასეთი შემთხვევა შეიქმნა ჩვენს ამოცანაშიც. ამ დროს სასარგებლოა move. იგი გადაიტანს ასლს და გააუქმებს ძველ ობიექტს. შევნიშნოთ, რომ move -ის გამოყენების სფერო გაცილებით ვრცელია, ვიდრე ეს განხილეული კერძო შემთხვევა.

<<< **ამოცანა 3** / unordered_set ტიპის ობიექტის შექმნა და რამდენიმე მარტივი ოპერაცია/. განვიხილოთ შემდეგი მარტივი პროგრამა, რომელშიც იქმნება ჰეშ-კონტეინერი და ხდება მასზე მარტივი მანიპულაციები. ეს კონტეინერი სინამდვილეში წარმოადგენს მასივების სიას. შემდეგ თემებში ვნახავთ თუ როგორ გავიგებთ სიების ელემენტების (კალათების) როდენობას და ზოგიერთ სხვა საკითხს.

```
#include <iostream>
#include <unordered_set>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
        // create and initialize unordered set
        unordered_set<int> s = { 1, 2, 3, 5, 7, 11, 22, 33, 55 };
        ostream_iterator<int> out(cout, "  ");
        cout << "Unordered set, containing { 1, 2, 3, 5, 7, 11, 22, 33, 55 }:" << endl;
        copy(s.begin(), s.end(), out);
        s.insert({ -5, -55, -33, -11, 111, 777 });
        cout << endl << "Values  { -5, -55, -33, -11, 111, 777 } inserteded:" << endl;
        copy(s.begin(), s.end(), out);

        s.erase(33);
        cout << endl << "Value 111 is removed:" << endl;
        copy(s.begin(), s.end(), out);

        if (s.find(777) != s.end()) {
                puts("19 is available");
        }

        for (auto i = s.begin(); i != s.end();) {
                if (*i < 0) {
                        i = s.erase(i);
                }
                else {
                        ++i;
                }
        }
        cout << endl << "All Negative Valuea are removed:" << endl;
```

```
	copy(s.begin(), s.end(), out);
	cout << endl;
}
```

<<< სავარჯიშოები

1. შექმენით მსგავი პროგრამა

```
	unordered_multiset<int> s = { 1, 2, 2, 3, 5, 7, 11, 22, 33, 55 };
```

ობიექტისთვის. შეამოწმეთ და ახსენით, თუ რა განსხვავებას იწვევს

```
	auto i = s.find(2);
	if (i != s.end())
		s.erase(*i);
		// s.erase(i);
```

კოდში კომენტარის ნიშანის ზედა სტრიქონში გადანაცვლება.

1. გაარჩიეთ პროგრამა, თუ “numbers.txt” ფაილში წერია ნამდვილი რიცხვები

```
 32 123 312 3123
-312 -231 -23 1 31
32 321
231 321 312 231
```

რამდენიმე სტრიქონად. სტრიქონში რიცხვებს შორის მხოლოდ ერთი ჰარია დატოვებული.

```
#include<iostream>
#include<fstream>
#include<iterator>
#include<algorithm>
#include<vector>
#include<string>
using namespace std;
int main()
{
	vector <vector<double> > A;
	ifstream ifs("data.txt");

	string line;

	while (getline(ifs, line))
	{
		double f;
		vector<double> fLoats;

		size_t pos;

		while ((pos = line.find(' ')) != string::npos)
		{
			string part = line.substr(0, pos);
			f = (double)atof(part.c_str());
			fLoats.push_back(f);
			line = line.substr(pos + 1);
		}
		// now, do the last double in the line
		f = (double)atof(line.c_str());
		fLoats.push_back(f);
		A.push_back(fLoats);
	}
	ostream_iterator<double> output(cout, "\t");
	for (int i = 0; i < A.size(); i++)
	{
		copy(A[i].begin(), A[i].end(), output);
		cout << endl;
	}
```