ლაბ 6++: ბინარული გროვა ობიექტებისთვის

განსახილველი საკითხები:

- გროვასთან დაკავშირებული STL -ის ზოგიერთი ალგორითმი
- ამოცანა >>>
  - კლასის ინტერფეისი >>>
  - კლასის იმპლემენტაცია >>>
  - პროგრამა-დრაივერი >>>
- სავარჯიშოები >>>

გროვასთან დაკავშირებული STL -ის ზოგიერთი ალგორითმი

გავიხსენოთ, რომ ბინარული გროვის შესაქმნელად და მასზე მანიპულირებისათვის, STL გვამარაგებს შემდეგი ალგორითმებით:

```
template<typename RandomAccessIterator>
void push_heap(RandomAccessIterator first, RandomAccessIterator last);

template<typename RandomAccessIterator, typename Compare>
void push_heap(RandomAccessIterator first, RandomAccessIterator last, Compare
comp);

template<typename RandomAccessIterator>
void pop_heap(RandomAccessIterator first, RandomAccessIterator last);

template<typename RandomAccessIterator, typename Compare>
void pop_heap(RandomAccessIterator first, RandomAccessIterator last, Compare
comp);

template<typename RandomAccessIterator>
void make_heap(RandomAccessIterator first, RandomAccessIterator last);

template<typename RandomAccessIterator, typename Compare>
void make_heap (RandomAccessIterator first, RandomAccessIterator last, Compare
comp);

template<typename RandomAccessIterator>
void sort_heap(RandomAccessIterator first, RandomAccessIterator last);

template<typename RandomAccessIterator, typename Compare>
void sort_heap (RandomAccessIterator first, RandomAccessIterator last, Compare
comp);
```

ალგორითმების აღწერა:

- თუ ფრაგმენტი `[first, last-1)` არის გროვა, `push_heap` გადაადგილებს `[first, last)`-ის ელემენტებს და ამ ფრაგმენტს გარდაქმნის გროვად. ამ დროს ჩვენ ვამბობთ, რომ `(last-1)`-ე ელემეტი შეყვანილ იქნა გროვაში;
- თუ `[first, last)` ფრაგმენტი არის გროვა, მაშინ `pop_heap` ადგილებს გაუცვლის `first` და `(last-1)` ადგილებზე მოთავსებულ მნიშვნელობებს და შემდეგ ელემენტებს ისე გადაადგილებს `[first, last-1)`-ში, რომ ეს ფრაგმენტი ისევ გროვად იქნება. ამ დროს ვამბობთ, რომ `(last-1)`-ე ელემეტი გამოყვანილ იქნა გროვიდან;
- `make_heap` გადაადგილებს `[first, last)` ფრაგმენტის ელემენტებს და ამ ფრაგმენტს აქცევს გროვად;
- `sort_heap` დაახარისხებს `[first, last)` ფრაგმენტში შექმნილი გროვის ელემენტებს.

<<< **ამოცანა.** ვთქვათ, ფაილში "data.txt" მოთვსებულია მონაცემები რამდენიმე ტბის შესახებ:

```
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
```

შევქმნათ კლასი, რომლის ობიექტები შეესაბამება ფაილის სტრიქონებს და პროგრამა დრაივერი, რომელიც ფაილიდან მონაცემებს ჩაწერს ვექტორში, ამ ვექტორის ელემენტებისგან შექმნის გროვას და დაახარისხებს მას. სიმარტივისთვის. საჭირო ბინარული დამოკიდებულება შექმნილია ლამბდა ფუნქციით.

<<< კლასის სინტერფეისი.

```
#pragma once
#include<iostream>

using namespace std;
class lake
{
private:
	string name;
	double area;		// km^2
	double height;			// meter
	double depth;			// meter
public:
	lake(void);
	~lake(void);
	lake(string itsName, double itsArea, double itsHeight, double itsDepth);
	string getName(void) const;
	double getArea(void) const;
	double getHeight(void) const;
	double getDepth(void) const;
	void setName(const string& itsName);
	void setArea(const double& itsArea);
	void setHeight(const double& itsHeight);
	void setDepth(const double& itsDepth);
	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);
};
```

<<< კლასის იმპლემენტაცია

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

lake::lake(void)
	: area(0)
	, height(0)
	, depth(0)
{
}
lake::~lake(void)
{
}
lake::lake(string itsName, double itsArea, double itsHeight, double itsDepth)
{
	name = itsName;
	area = itsArea;
	height = itsHeight;
	depth = itsDepth;
}
```

```
string lake::getName(void) const
{
	return name;
}
double lake::getArea(void) const
{
	return area;
}
double lake::getHeight(void) const
{
	return height;
}
double lake::getDepth(void) const
{
	return depth;
}
void lake::setName(const string& itsName)
{
	name = itsName;
}
void lake::setArea(const double& itsArea)
{
	area = itsArea;
}
void lake::setHeight(const double& itsHeight)
{
	height = itsHeight;
}
void lake::setDepth(const double& itsDepth)
{
	depth = itsDepth;
}
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 !(*this == other);
}
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.area << '\t' <<
		x.height << '\t' <<
		x.depth << '\t' <<
		x.name << '\t' << endl;
	return os;
}
```

<<< პროგრამა დრაივერი

```
#include<iterator>
#include<fstream>
#include<algorithm>
```

```
#include<vector>
#include "lake.h"

int main()
{
	vector<lake> v;
	lake tmp;
	{
		ifstream ifs("lakes.txt");
		while (ifs >> tmp)
			v.push_back(tmp);
	}
	for (int i = 0; i<v.size(); i++)
		cout << v[i];
	cout << endl;
	auto lm =
		[](const lake& first, const lake& other) {return (first.getName() <
other.getName()); };
	make_heap(v.begin(), v.end(),lm);
	cout << endl << endl;
	cout << "This is heap of lakes:\n";
	cout <<  endl;
	for (int i = 0; i<v.size(); i++)
		cout << v[i];
	cout << endl;

	sort_heap(v.begin(), v.end(),lm);
	cout << endl << endl;
	cout << "This is sorted sequence of  lakes:\n";
	cout << endl << endl;

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

<<< სავარჯიშოები

1. განხილულ პროგრამაში ვექტორების ნაცვლად გამოიყენეთ კონტეინერი დეკი. ან დინამიკური მასივი.
2. შექმენით ანალოგიური პროგრამა ვულკანების და პლანეტების კლასებისთვის.
3. გააკეთეთ ბინარული პრდეკატები (ობიექტების შედარებისთვის) ფუნქტორების საშუალებით და გამოიყენეთ ისინი ლამბდა ფუნქციების ნაცვლად.