Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 5
Вариант 27
Итераторы и умные указатели

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

Main.cpp

```
// //  Created by kosha on 9/01/2020.
// //  Copyright © 2020 kosha. All rights reserved.
// //
#include <iostream>
#include <algorithm>
#include <iostream>

#include "DynamicArray.h"

void menu() {
	std::cout << "0 - Terminate\n";
	std::cout << "1 - Array initialization\n";
	std::cout << "2 - Show coordinates of the center by index (beginning with 0)\n";
	std::cout << "3 - Show number of items with square less than ...\n";
	std::cout << "4 - Show every step of iterator \n";
	std::cout << "5 - Insert a new item in vector before this one\n";
	std::cout << "6 - Erase this item in iterator\n";
	std::cout << "7 - Add item by index\n";
	std::cout << "8 - Delete item by index\n";
	std::cout << "> ";
}

int main() {
	int cmd;
	std::cout << "Enter size of your vector : ";
	int size;
	std::cin >> size;

	containers::DynamicArray< rectangle<int> > Vector(size);

	while (true) {
		menu();
		std::cin >> cmd;
		if (cmd == 0) return 0;
		else if (cmd == 1) {
			for (int i = 0; i < Vector.getSize(); i++) {
				std::cout << "Enter vertices : \n";
				rectangle<int> rect(std::cin);
				Vector.push_back(rect);
			}
		} else if (cmd == 2) {
			std::cout << "Enter index : ";
			int index;
			std::cin >> index;
			std::cout << Vector[index].center();
		} else if (cmd == 3) {
			int res = 0;
			std::cout << "Enter your square : ";
			double square;
			std::cin >> square;

			int cmdcmd;
			std::cout << "Do you want to use std::count_if? \n Type '1' for yes or type '2' for no : ";
			std::cin >> cmdcmd;

			if (cmdcmd == 1) res = std::count_if(Vector.begin(), Vector.end(), [square](rectangle<int> i) {return i.area() < square;});
			else {
				auto it = Vector.begin();
				auto end = Vector.end();

				while (it != end) {
					if ((*it).area() < square) res++;
					++it;
				}
			}
```

```
                std::cout << "Number of items [with square less than "<< square<< "] is " << res << '\n';
        } else if (cmd == 4) {
                int cmdcmd;
                std::cout << "Do you want to use std::for_each? \n Type '1' for yes or type '2' for no : ";
                std::cin >> cmdcmd;

                if (cmdcmd == 1) std::for_each(Vector.begin(), Vector.end(), [](rectangle<int> i) -> void{i.print();});
                else {
                        auto it = Vector.begin();
                        auto end = Vector.end();

                        int n = 0;

                        while (it != end) {
  std::cout << "-- item number " << n << "--\n";
                                std::cout << *it;
                                ++it;
                                n++;
                        }

                }

        } else if (cmd == 5) {
                std::cout << "Enter vertices of item you want to delete : ";
                rectangle<int> toDelete(std::cin);

                std::cout << "Enter vertices of item you want to add : ";
                rectangle<int> toInsert(std::cin);

                auto it = Vector.begin();
                auto end = Vector.end();

                while (it != end) {
                        if (*it == toDelete) {
                                Vector.insert(it, toInsert);
                                break;
                        }
                        ++it;
                }

                it = Vector.begin();
                end = Vector.end();

                std::cout << "Now vector is like : \n";
                int n = 0;

                while (it != end) {
std::cout << "-- item number " << n << "--\n";
                        std::cout << *it;
                        ++it;
                        n++;
                }
        } else if (cmd == 6) {
                std::cout << "Enter vertices of item you want to erase : ";
                rectangle<int> toDelete(std::cin);

                auto it = Vector.begin();
                auto end = Vector.end();

                while (it != end) {
                        if (*it == toDelete) {
                                Vector.erase(it);
                        }
                        ++it;
                }

                it = Vector.begin();

                std::cout << "Now vector is like : \n";
```

```
                int n = 0;

                while (it != end) {
        std::cout << "-- item number " << n << "--\n";
                        std::cout << *it;
                        ++it;
                        n++;
                }
        } else if (cmd == 7) {
                std::cout << "Enter vertices of item you want to insert : ";
                rectangle<int> toInsert(std::cin);
                std::cout << "Enter index : ";
                int id;
                std::cin >> id;
                auto it = Vector.begin();
                if (id >= Vector.getSize()) Vector.addOnOutOfRange(id, toInsert);
                else {
                        for (int i = 0; i < id; i++) ++it;

                        Vector.insert(it, toInsert);

                }
                std::cout << "Now vector is like : \n";
                int n = 0;

                it = Vector.begin();
                auto end = Vector.end();

                while (it != end) {
        std::cout << "-- item number " << n << "--\n";
                        std::cout << *it;
                        ++it;
                        n++;
                }
        } else if (cmd == 8) {
                std::cout << "Enter index : ";
                int id;
                std::cin >> id;

                auto it = Vector.begin();
                for (int i = 0; i < id; i++) ++it;

                Vector.erase(it);

                std::cout << "Now vector is like : \n";
                int n = 0;

                it = Vector.begin();
                auto end = Vector.end();

                while (it != end) {
        std::cout << "-- item number " << n << "--\n";
                        std::cout << *it;
                        ++it;
                        n++;
                }
        }
    }
}
```

DynamicArray.h

```
#ifndef D_CONTAINERS_DynamicArray_H_
#define D_CONTAINERS_DynamicArray_H_ 1

#include <iterator>
#include <memory>
#include <cstddef>
#include "vertex.h"
#include "rectangle.h"

namespace containers {
```

```
template<class T>
class DynamicArray {

public:
	DynamicArray();
	DynamicArray(int sz);

	struct forward_iterator {
		using value_type = T;//тип значения, на которое указывает итератор типа T
	using reference = T&; //тип ссылки, возвращаемой при разыменовании итератора
	using pointer = T*; //тип указателя, возвращаемого при обращении к объекту итератора
	using difference_type = ptrdiff_t; //целочисленный тип, представляющий значения смещений итераторов относительно друг друга
	using iterator_category = std::forward_iterator_tag; //тип, указывающий на набор операций, поддерживаемых итератором

	forward_iterator(T *ptr);
	forward_iterator() = default;
	T& operator*();

	forward_iterator& operator++();

	bool operator== (const forward_iterator& o) const;
	bool operator!= (const forward_iterator& o) const;

  private:
	T *p;
	friend DynamicArray;

	};

	forward_iterator begin();
	forward_iterator end();

	T& operator[](int index);
	void reSize(int newSize);
	void push_back(T object);
	void addOnOutOfRange(int position, T object);
	int getSize();
	forward_iterator insert(forward_iterator it, T object);
	void erase(forward_iterator it);

private:

	std::unique_ptr<T[]> data;
	int size;
	int used;

};

template<class T>
DynamicArray<T>::DynamicArray() {

	data = nullptr;
	size = 0;
	used = 0;

}

template<class T>
void DynamicArray<T>::reSize(int newSize) {
	if (size == newSize) return;
	std::unique_ptr<T[]> resizing = std::unique_ptr<T[]>(new T[newSize]);

	for (int i = 0; i < std::min(size, newSize); i++) {
		resizing[i] = data[i];
	}
	size = newSize;
	data = std::move(resizing);
}
```

```
template<class T>
DynamicArray<T>::DynamicArray(int sz) {

	data = std::unique_ptr<T[]>(new T[sz]);
	size = sz;
	used = 0;

}

template<class T>
void DynamicArray<T>::push_back(T object) {

	if (used >= size) reSize(size++);
	data[used] = object;
	used++;

}

template<class T>
void DynamicArray<T>::addOnOutOfRange(int position, T object) {

	reSize(position + 1);

	data[position] = object;
	used++;

}

template<class T>
DynamicArray<T>::forward_iterator::forward_iterator(T *ptr) {
	p = ptr;
}

template<class T>
T& DynamicArray<T>::forward_iterator::operator*() {
	return *p;
}

template<class T>
typename DynamicArray<T>::forward_iterator& DynamicArray<T>::forward_iterator::operator++() {
	++p;
	return *this;
}

template<class T>
bool DynamicArray<T>::forward_iterator::operator== (const forward_iterator& o) const {
	return p == o.p;
}


template<class T>
bool DynamicArray<T>::forward_iterator::operator!= (const forward_iterator& o) const {
	return p != o.p;
}

template<class T>
typename DynamicArray<T>::forward_iterator DynamicArray<T>::begin() {
	return &data[0];
}

template<class T>
typename DynamicArray<T>::forward_iterator DynamicArray<T>::end() {
	return &data[size];
}

template<class T>
T& DynamicArray<T>::operator[](int index) {
	if (index > size - 1) throw std::logic_error("index is out of range!\n");
	T& result = data[index];
	return result;
```

```
}

template<class T>
int DynamicArray<T>::getSize() {
	return size;
}

template<class T>
typename DynamicArray<T>::forward_iterator DynamicArray<T>::insert(forward_iterator it, T object) {
	for (int i = 0; i < size; i++) {
		if (it == &data[i]) {
			if (used == size) reSize(size + 1);
			for (int j = size - 1; j > i; j--) {
				data[j] = data[j - 1];
			}
			data[i] = object;
			used++;
			return &data[i];
		}
	}
	throw std::logic_error("Place doesn't exist!\n");
}

template<class T>
void DynamicArray<T>::erase(forward_iterator it) {
	for (int i = 0; i < size; i++) {
		if (it == &data[i]) {
			for (int j = i; j < size - 1; j++) {
				data[j] = data[j + 1];
			}
			reSize(size - 1);
			used--;
			return;
		}
	}
	throw std::logic_error("Place doesn't exist!\n");
   }
}

#endif
```

Rectangle.h

```
#ifndef D_RECTANGLE_H_
#define D_RECTANGLE_H_ 1

#include <algorithm>
#include <iostream>

#include "vertex.h"
#include "vector.h"

template<class T>
struct rectangle {
	vertex<T> vertices[4];
	bool existance;

	rectangle(std::istream& is);
	rectangle() = default;

	vertex<double> center() const;

	bool operator==(const rectangle<T>& comp) const;

	double area() const;
	void print() const;
};



template<class T>
rectangle<T>::rectangle(std::istream& is) {
	for(int i = 0; i < 4; ++i){
		is >> vertices[i];
```

```
		}

		if (isPerpendicular(Vector< vertex<T> >(vertices[0], vertices[1]), Vector< vertex<T> >(vertices[0], vertices[3])) && isPerpendicular(Vector< vertex<T> >(vertices[0], vertices[1]), Vector< vertex<T> >(vertices[1], vertices[2])) &&
			isPerpendicular(Vector< vertex<T> >(vertices[1], vertices[2]), Vector< vertex<T> >(vertices[2], vertices[3])) && isPerpendicular(Vector< vertex<T> >(vertices[2], vertices[3]), Vector< vertex<T> >(vertices[0], vertices[3]))) {



		} else if (isPerpendicular(Vector< vertex<T> >(vertices[0], vertices[3]), Vector< vertex<T> >(vertices[3], vertices[1])) && isPerpendicular(Vector< vertex<T> >(vertices[3], vertices[1]), Vector< vertex<T> >(vertices[1], vertices[2])) &&
			isPerpendicular(Vector< vertex<T> >(vertices[1], vertices[2]), Vector< vertex<T> >(vertices[2], vertices[0])) && isPerpendicular(Vector< vertex<T> >(vertices[0], vertices[2]), Vector< vertex<T> >(vertices[0], vertices[3]))) {

				vertex<T> tmp;
				tmp = vertices[0];
				vertices[0] = vertices[3];
				vertices[3] = tmp;

		} else if (isPerpendicular(Vector< vertex<T> >(vertices[0], vertices[1]), Vector< vertex<T> >(vertices[1], vertices[3])) && isPerpendicular(Vector< vertex<T> >(vertices[1], vertices[3]), Vector< vertex<T> >(vertices[3], vertices[2])) &&
			isPerpendicular(Vector< vertex<T> >(vertices[3], vertices[2]), Vector< vertex<T> >(vertices[2], vertices[0])) && isPerpendicular(Vector< vertex<T> >(vertices[2], vertices[0]), Vector< vertex<T> >(vertices[0], vertices[1]))) {

				vertex<T> tmp;
				tmp = vertices[2];
				vertices[2] = vertices[3];
				vertices[3] = tmp;

		} else if (vertices[0] == vertices[1] || vertices[0] == vertices[2] || vertices[0] == vertices[3] || vertices[1] == vertices[2] || vertices[1] == vertices[3] || vertices[2] == vertices[3]) {
			throw std::logic_error("No points are able to be equal");
		} else {
			throw std::logic_error("That's not a Rectangle, sides are not Perpendicular");
		}

		if (!(Vector< vertex<T> >(vertices[0], vertices[1]).length() == Vector< vertex<T> >(vertices[2], vertices[3]).length() && Vector< vertex<T> >(vertices[1], vertices[2]).length() == Vector< vertex<T> >(vertices[0], vertices[3]).length())) {
			throw std::logic_error("That's not a Rectangle, sides are not equal");
		}

		existance = true;
}

template<class T>
double rectangle<T>::area() const {
		if (existance == false) std::logic_error("Item doesn't exist");
		return Vector< vertex<T> >(vertices[0], vertices[1]).length() * Vector< vertex<T> >(vertices[1], vertices[2]).length();
}

template<class T>
void rectangle<T>::print() const {
		if (existance == true) std::cout << vertices[0] << vertices[1] << vertices[2] << vertices[3] << '\n';

}

template<class T>
vertex<double> rectangle<T>::center() const {
		if (existance == false) std::logic_error("Item doesn't exist");
		vertex<double> p;
		p.x = (vertices[0].x + vertices[1].x + vertices[2].x + vertices[3].x) / 4;
		p.y = (vertices[0].y + vertices[1].y + vertices[2].y + vertices[3].y) / 4;
		return p;
}

template<class T>
bool rectangle<T>::operator==(const rectangle<T>& comp) const {
		for (int i = 0; i < 4; i++) {
			if (vertices[i] != comp.vertices[i]) return false;
		}
		return true;
}

template<class T>
std::ostream& operator<< (std::ostream& os, const rectangle<T>& rect) {
```

```
	if (rect.existance) os << rect.vertices[0] << rect.vertices[1] << rect.vertices[2] << rect.vertices[3];
	return os;
}

#endif // D_TRIANGLE_H_
```

Vector.h

```
#ifndef VECTOR_H_
#define VECTOR_H_

#include "vertex.h"
#include <cmath>
#include <numeric>
#include <limits>

template<class T>
struct Vector {
	explicit Vector(T a, T b);
	double length() const;
	double x;
	double y;
	double operator* (Vector b) ;
	bool operator== (Vector b);
};

template<class T>
Vector<T>::Vector(T a, T b) {
	x = b.x - a.x;
	y = b.y - a.y;
}

template<class T>
double Vector<T>::length() const{
	return sqrt(x * x + y * y);
}

template<class T>
double Vector<T>::operator* (Vector<T> b) {
	return x * b.x + y * b.y;
}

template<class T>
bool Vector<T>::operator== (Vector<T> b) {
	return std::abs(x - b.x) < std::numeric_limits<double>::epsilon() * 100
	&& std::abs(y - b.y) < std::numeric_limits<double>::epsilon() * 100;
}

template<class T>
bool isPerpendicular(const Vector<T> a, const Vector<T> b) {
	return (a.x * b.x + a.y * b.y) == 0;
}


#endif
```

Vertex.h

```
#ifndef D_VERTEX_H
#define D_VERTEX_H 1

#include <iostream>

template<class T>
struct vertex {
	T x;
	T y;

};

template<class T>
```

```
std::istream& operator>> (std::istream& is, vertex<T>& p) {
        is >> p.x >> p.y;
        return is;
}

template<class T>
std::ostream& operator<< (std::ostream& os, const vertex<T>& p) {
        os << p.x << ' ' << p.y << '\n';
        return os;
}

template<class T>
vertex<T> operator+(vertex<T> a,vertex<T> b){
   vertex<T> res;
   res.x = a.x + b.x;
   res.y = a.y + b.y;
   return res;
}

template<class T>
bool operator == (vertex<T> a, vertex<T> b) {
        return (a.x == b.x && a.y == b.y);
}

template<class T>
bool operator != (vertex<T> a, vertex<T> b) {
        return (a.x != b.x || a.y != b.y);
}


template<class T>
vertex<T>& operator/= (vertex<T>& vertex, int number) {
   vertex.x = vertex.x / number;
   vertex.y = vertex.y / number;
   return vertex;
}

#endif // D_VERTEX_H
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/oop_exercise_05

3.Набор testcases

1. **Входные данные:**

3
1

4 2
4 7
14 7
14 2

-5 -1

```
5 -1
5 -7
-5 -7

-5 1
-7 3
-3 7
-1 5

0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 2
Enter index : 0
9 4.5
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 2
Enter index : 1
0 -4
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 2
Enter index : 2
-4 4
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
```

```
8 - Delete item by index
> 3
Enter your square : 29
Do you want to use std::count_if?
 Type '1' for yes or type '2' for no : 2
Number of items [with square less than 29] is 1
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 4
Do you want to use std::for_each?
 Type '1' for yes or type '2' for no : 2
-- item number 0--
4 2
4 7
14 7
14 2
-- item number 1--
-5 -1
5 -1
5 -7
-5 -7
-- item number 2--
-5 1
-7 3
-3 7
-1 5
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 5
Enter vertices of item you want to delete :
4 2
4 7
14 7
14 2
Enter vertices of item you want to add : 3 2
3 7
14 7
14 2
Now vector is like :
```

-- item number 0--
3 2
3 7
14 7
14 2
-- item number 1--
4 2
4 7
14 7
14 2
-- item number 2--
-5 -1
5 -1
5 -7
-5 -7
-- item number 3--
-5 1
-7 3
-3 7
-1 5
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 6
Enter vertices of item you want to erase : -5 1
-7 3
-3 7
-1 5
Now vector is like :
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 4
Do you want to use std::for_each?
Type '1' for yes or type '2' for no : 0
-- item number 0--
3 2
3 7
14 7
14 2
-- item number 1--

4 2
4 7
14 7
14 2
-- item number 2--
-5 -1
5 -1
5 -7
-5 -7
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index
8 - Delete item by index
> 7
Enter vertices of item you want to insert : -5 1
-7 3
-3 7
-1 5
Enter index : 2
Now vector is like :
-- item number 0--
3 2
3 7
14 7
14 2
-- item number 1--
4 2
4 7
14 7
14 2
-- item number 2--
-5 1
-7 3
-3 7
-1 5
-- item number 3--
-5 -1
5 -1
5 -7
-5 -7
0 - Terminate
1 - Array initialization
2 - Show coordinates of the center by index (beginning with 0)
3 - Show number of items with square less than ...
4 - Show every step of iterator
5 - Insert a new item in vector before this one
6 - Erase this item in iterator
7 - Add item by index

```
8 - Delete item by index
> 8
Enter index : 2
Now vector is like :
-- item number 0--
3 2
3 7
14 7
14 2
-- item number 1--
4 2
4 7
14 7
14 2
-- item number 2--
-5 -1
5 -1
5 -7
-5 -7
```

5. Объяснение результатов программы

В файле main.cpp реализован интерфейс пользователя. В зависимости от команды выполняется :
0) Завершение программы
1) Инициализация массива
2) Вывод координат центра по индексу
3) Вывод количества объектов, площадь которых меньше заданной
4) Прохождение массива с помощью итератора
5) Добавление элемента в массив по индексу
6) Удаление элемента из массива по индексу

В файле DynamicArray.h реализован forward_iterator и динамический массив.
В rectangle.h описаны методы рассчета координат центра и площади прямоугольника.
В vertex.h реализована перегрузка операторов для координат точек.
В vector.h реализована работа с координатами точек.

Вывод: Проделав данную работу я изучила итераторы и умные указатели. Я поняла, что использование умных указателей и итераторов расширяет возможности программы.