Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 4: «Итераторы и умные указатели»

Группа:	М8О-206Б-18, №27
Студент:	Шорохов Алексей Павлович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	

Москва, 2019

Задание

Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;
3. Вычисление площади фигуры;

27.	Прямоугольник	Динамический массив

Адрес репозитория на GitHub

Код программы на C++

```
cmake_minimum_required(VERSION 3.2)

project(lab5)

add_executable(run
        Source.cpp
        )

set_property(TARGET lab5 PROPERTY CXX_STANDARD 17)
```

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> lhs,vertex<T> rhs){
    vertex<T> res;
    res.x = lhs.x + rhs.x;
    res.y = lhs.y + rhs.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
```

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("Object 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("Object 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 isParallel(const Vector<T> a, const Vector<T> b) {
	return (a.x * b.y - a.y * b.y) == 0;
}

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


#endif
```

DobriyArray.h

```
#ifndef D_CONTAINERS_DOBRIYARRAY_H_
#define D_CONTAINERS_DOBRIYARRAY_H_ 1

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

namespace containers {
```

```
template<class T>
class DobriyArray {

public:
	DobriyArray();
	DobriyArray(int sz);

	struct forward_iterator {
		using value_type = T;
	using reference = T&;
	using pointer = T*;
	using difference_type = ptrdiff_t;
	using iterator_category = std::forward_iterator_tag;

	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 DobriyArray;

	};

	forward_iterator begin();
	forward_iterator end();

	T& operator[](int index);
	void reSize(int newSize);
	void push_back(T object);
	int getSize();
	int getUsed();
	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>
DobriyArray<T>::DobriyArray() {

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

}

template<class T>
void DobriyArray<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(used, newSize); i++) {
                resizing[i] = data[i];
        }
        data = std::move(resizing);
}

template<class T>
DobriyArray<T>::DobriyArray(int sz) {

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

}

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

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

}


template<class T>
DobriyArray<T>::forward_iterator::forward_iterator(T *ptr) {
```

```
	p = ptr;
}

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

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

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


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

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

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

template<class T>
T& DobriyArray<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 DobriyArray<T>::getSize() {
	return size;
}

template<class T>
int DobriyArray<T>::getUsed() {
	return used;
}

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

template<class T>
void DobriyArray<T>::erase(forward_iterator it) {
	for (int i = 0; i < size; i++) {
		if (it == &data[i]) {
			for (int j = i; j < size; j++) {
				data[j] = data[j + 1];
			}
			used--;
			return;
		}
	}

	throw std::logic_error("Place doesn't exist!\n");
}
```

```
}


#endif
```

File01.test

```
3
1
0 0
0 1
1 0
1 1
0 0
0 2
2 0
2 2
0 0
0 3
3 0
3 3
2 2
4 0
6
0 0
0 2
2 2
2 0
5
0 0
0 3
3 3
3 0
0 0
0 4
4 4
4 0
0
```

File02.test

```
2
0 0
0 1
1 0
1 1
0 0
0 10
```

10 0
10 10
4 1
3 100
0

Source.cpp

```
#include <algorithm>
#include <iostream>

#include "containers/DobriyArray.h"

void menu() {
	std::cout << "0 : EXIT\n";
	std::cout << "1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA\n";
	std::cout << "2 : GET ITEM CENTER BY INDEX\n";
	std::cout << "3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...\n";
	std::cout << "4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP\n";
	std::cout << "5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE\n";
	std::cout << "6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT\n";
	std::cout << "7 : INSERT OBJECT BY INDEX\n";
	std::cout << "8 : ERASE OBJECT BY INDEX\n";
	std::cout << "> ";
}

int main() {


	int cmd;
	bool doWeHaveAVector = false;

	std::cout << "Enter size of your vector : ";
	int size;
	std::cin >> size;

	containers::DobriyArray< 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? : 1 - yes; 0 - 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 << "Amount is " << res << '\n';
		} else if (cmd == 4) {
			int cmdcmd;
			std::cout << "Do you want to use std::for_each? : 1 - yes; 0 - 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 << "___OBJECT_" << n << "__\n";
                std::cout << *it;
                ++it;
                n++;
            }

        }

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

        std::cout << "Enter vertices of object you want to insert : ";
        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 << "___OBJECT_" << n << "__\n";
            std::cout << *it;
            ++it;
            n++;
        }
    } else if (cmd == 6) {
        std::cout << "Enter vertices of object 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 << "___OBJECT_" << n << "__\n";
			std::cout << *it;
			++it;
			n++;
		}
	} else if (cmd == 7) {
		std::cout << "Enter vertices of object you want to insert : ";
		rectangle<int> toInsert(std::cin);
		std::cout << "Enter index : ";
		int id;
		std::cin >> id;
		auto it = Vector.begin();
		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 << "___OBJECT_" << n << "__\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 << "___OBJECT_" << n << "__\n";
                std::cout << *it;
                ++it;
                n++;
```

Результаты тестов

1:

```
Enter size of your vector : 3
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT
BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 1
Enter vertices :
0 0
0 1
1 0
```

```
1 1
Enter vertices :
0 0
0 2
2 0
2 2
Enter vertices :
0 0
0 3
3 0
3 3
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT
BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 2
Enter index : 2
1 1
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT
BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 4
Do you want to use std::for_each? : 1 - yes; 0 - no; : 0
___OBJECT_0__
0 0
0 1
1 1
1 0
___OBJECT_1__
0 0
0 2
2 2
2 0
___OBJECT_2__
0 0
0 3
```

```
3 3
3 0
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT
BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 6
Enter vertices of object you want to erase : 0 0
0 2
2 2
2 0
Now vector is like :
___OBJECT_0__
0 0
0 1
1 1
1 0
___OBJECT_1__
0 0
0 3
3 3
3 0
___OBJECT_2__
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT
BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 5
Enter vertices of object you want to delete : 0 0
0 3
3 3
3 0
Enter vertices of object you want to insert : 0 0
0 4
4 4
4 0
Now vector is like :
```

___OBJECT_0__
0 0
0 1
1 1
1 0
___OBJECT_1__
0 0
0 4
4 4
4 0
___OBJECT_2__
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 0

2:

Enter size of your vector : 2
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 1
Enter vertices :
0 0
0 1
1 0
1 1
Enter vertices :
0 10
0 0
10 0
10 10
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA

2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 4
Do you want to use std::for_each? : 1 - yes; 0 - no; : 1
0 0
0 1
1 1
1 0

0 10
0 0
10 0
10 10

0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 3
Enter your square : 100
Do you want to use std::count_if? : 1 - yes; 0 - no; : 1
Amount is 1
0 : EXIT
1 : GO THROUGH VECTOR WITH ITERATOR AND INPUT DATA
2 : GET ITEM CENTER BY INDEX
3 : GET AMOUNT OF OBJECTS WITH SQUARE LESS THAN...
4 : GO THROUGH VECTOR WITH ITERATOR AND SHOW EVERY STEP
5 : GO THROUGH VECTOR WITH ITERATOR AND INSERT A NEW OBJECT BEFORE THIS ONE
6 : GO THROUGH VECTOR WITH ITERATOR AND ERASE THIS OBJECT
> 0

Объяснение результатов

Программа получает на вход команды из меню. В зависимости от команды совершается одно из действий: заполнение массива, получений координат центра по индексу, получение количества объектов, площадь

которых меньше заданной, прохождение массива с помощью итератора, вставка в массив и удаление из массива.

Вывод

Были изучены основы темы итераторов и умных указателей, применены в лабораторной работе. Применение указателей и итераторов значительно расширяет возможности программы. Порой они сложны в изучении, однако будут очень полезны в практической деятельности и иногда незаменимы при написании программного кода.