

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

**Лабораторная работа
по курсу «ООП»**

Тема:
Основы работы с коллекциями: итераторы.

Студент:	Николаев В.А.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	14
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

point.h:

```
#pragma once
#include <iostream>
```

```
template <class T>
struct point {
    T x, y;
    point (T a,T b) { x = a, y = b;};
    point() = default;
};
```

```
template <class T>
std::istream& operator >> (std::istream& npt,point <T>& p ) {
    return npt >> p.x >> p.y;
}
```

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

pentagon.h:

```
#pragma once
template <class T>
struct pentagon
{
    point <T> a1,a2,a3,a4,a5;
    pentagon (point <T> x1, point <T> x2, point <T> x3, point <T> x4, point <T> x5)
    {
        a1 = x1; a2 = x2; a3 = x3; a4 = x4; a5 = x5;
    }
    pentagon() = default;
    point <T> center() const {
        T x,y;
        x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;
        y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;
        point <T> p(x,y);
        return p;
    }

    void print(std::ostream& out) {
        out << "Coordinates are:\n" << "{\n" << a1 << a2 << a3 << a4 << a5 << "}" << "\n";
    }
    T area() const {
```

```

        return (0.5) * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y +
a5.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a1.x ));
    }
    pentagon(std::istream& is) {
        is >> a1 >> a2 >> a3 >> a4 >> a5;
    }
};

```

hexagon.h:

```
#pragma once
```

```
template <class T>
```

```
struct hexagon
```

```
{
```

```
    point <T> a1, a2, a3, a4, a5, a6;
```

```
    hexagon() = default;
```

```
    point <T> center() const {
```

```
        T x,y;
```

```
        x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;
```

```
        y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;
```

```
        point <T> p(x,y);
```

```
        return p;
```

```
    }
```

```
    void print(std::ostream& out) {
```

```
        out << "Coordinates are:\n{\n" << a1 << a2 << a3 << a4 << a5 << a6 << "}\n";
```

```
    }
```

```
    T area() const {
```

```
        return 0.5 * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y
+ a6.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a6.x +
a6.y*a1.x ));
```

```
    }
```

```
    hexagon(std::istream& is) {
```

```
        is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
```

```
    }
```

```
};
```

octagon.h:

```
#pragma once
```

```
template<class T>
```

```
struct octagon
```

```
{
```

```
    point <T> a1, a2, a3, a4, a5, a6, a7, a8;
```

```
    point <T> center() const {
```

```
        T x,y;
```

```
        x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x + a7.x + a8.x) / 8;
```

```

        y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y + a7.y + a8.y) / 8;
        point <T> p(x,y);
        return p;
    }
    void print(std::ostream& out) {
        out << "Coordinates are:\n{\n" << a1 << a2 << a3 << a4 << a5 << a6 << a7 <<
a8 << "}\n";
    }

    T area() const {
        return 0.5 * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y
+ a6.x*a7.y + a7.x*a8.y + a8.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x +
a4.y*a5.x + a5.y*a6.x + a6.y*a7.x + a7.y*a8.x + a8.y*a1.x ));
    }

    octagon(std::istream& is) {
        is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
    }
};

```

main.cpp:

```

#include <algorithm>
#include <iostream>
#include "point.h"
#include "pentagon.h"
#include "list.h"

int main()
{
    std::string i;
    size_t j;
    containers::list<pentagon<double>> l;
    while (true) {
        std::cout << "1- insert by index\n2 - delete by index\n3 - count if(Количество
фигур с площадью больше вводимого значения)\n4 - print by index\n5 - print all\
n6 - quit\n";
        std::cin >> i;
        if (i == "1") {
            std::cout << "Enter coordinates:\n";
            pentagon <double> p(std::cin);
            std::cout << "Enter index:\n";
            std::cin >> j;
            try {
                l.insert_by_number(j, p);
            } catch (std::logic_error &err) {

```

```

        l.delete_by_number(j);
        std::cout << err.what() << "\n";
        break;
    }

}

if (i == "2") {
    std::cout << "Enter index:\n";
    std::cin >> j;
    if (j == (l.length()-1)){
        l.pop_back();
    }else {
        try {
            l.delete_by_number(j);
        } catch (std::logic_error &err) {
            std::cout << err.what() << std::endl;
            break;
        }
    }
}

if (i == "3") {
    std::cout << "Enter the value:\n";
    double val;
    std::cin >> val;
    std::cout << std::count_if(l.begin(), l.end(), [val](pentagon<double> el)
{return el.area() > val; }) << " pentagons\n";
}

if (i == "4") {
    std::cout << "Enter index:\n";
    std::cin >> j;
    l[j].print(std::cout);
}

if (i == "5") {
    for (auto elem : l) {
        elem.print(std::cout);
    }
}

if (i == "6") {
    break;
}

}
}

```

list.h:

```
#pragma once
#include <iterator>
#include <memory>
#include "pentagon.h"
```

```
namespace containers {
```

```
template<class T>
class list {
private:
    struct element;
    unsigned int size = 0;
public:
    list() = default;

    class forward_iterator {
public:
        using value_type = T;
        using reference = T&;
        using pointer = T*;
        using difference_type = std::ptrdiff_t;
        using iterator_category = std::forward_iterator_tag;
        explicit forward_iterator(element* ptr);
        T& operator*();
        forward_iterator& operator++();
        forward_iterator operator++(int);
        bool operator==(const forward_iterator& other) const;
        bool operator!=(const forward_iterator& other) const;

private:
        element* it_ptr;
        friend list;

    };
    forward_iterator begin();
    forward_iterator end();
    void pop_back();
    void pop_front();
    size_t length();
    void delete_by_it(forward_iterator d_it);
    void delete_by_number(size_t N);
    void insert_by_it(forward_iterator ins_it, T& value);
    void insert_by_number(size_t N, T& value);
    T& operator[](size_t index) ;
```

```
list& operator=(list&& other);
```

```
private:
```

```
    struct element {  
        T value;  
        std::shared_ptr<element> next_element = nullptr;  
        std::shared_ptr<element> prev_element = nullptr;  
        forward_iterator next();  
    };  
    static std::shared_ptr<element> push_impl(std::shared_ptr<element> cur);  
    static std::shared_ptr<element> pop_impl(std::shared_ptr<element> cur);  
    std::shared_ptr<element> first = nullptr;  
};
```

```
template<class T>  
typename list<T>::forward_iterator list<T>::begin() {  
    return forward_iterator(first.get());  
}
```

```
template<class T>  
typename list<T>::forward_iterator list<T>::end() {  
    return forward_iterator(nullptr);  
}
```

```
template<class T>  
size_t list<T>::length() {  
    return size;  
}
```

```
template<class T>
```

```
                                std::shared_ptr<typename list<T>::element>  
list<T>::push_impl(std::shared_ptr<element> cur) {  
    if (cur -> next_element != nullptr) {  
        return push_impl(cur->next_element);  
    }  
    return cur;  
}
```

```
template<class T>  
void list<T>::pop_front() {  
    if (size == 0) {  
        throw std::logic_error ("stack is empty");  
    }
```

```
    first = first->next_element;  
    first->prev_element = nullptr;  
    size--;  
}
```

```

template<class T>
void list<T>::pop_back() {
    if (size == 0) {
        throw std::logic_error("can't pop from empty list");
    }
    first = pop_impl(first);
    size--;
}
template<class T>
                                std::shared_ptr<typename          list<T>::element>
list<T>::pop_impl(std::shared_ptr<element> cur) {
    if (cur->next_element != nullptr) {
        cur->next_element = pop_impl(cur->next_element);
        return cur;
    }
    return nullptr;
}
template<class T>
void list<T>::delete_by_it(containers::list<T>::forward_iterator d_it) {
    if (d_it.it_ptr == nullptr) {
        throw std::logic_error("попытка доступа к несуществующему элементу");
    }
    if (d_it == this->begin()) {
        this->pop_front();
        size--;
        return;
    }
    if (d_it == this->end()) {
        this->pop_back();
        size--;
        return;
    }
    d_it.it_ptr->prev_element->next_element = d_it.it_ptr->next_element;
    d_it.it_ptr->next_element->prev_element = d_it.it_ptr->prev_element;

    size--;
}
template<class T>
void list<T>::delete_by_number(size_t N) {
    forward_iterator it = this->begin();
    for (size_t i = 1; i <= N; ++i) {
        ++it;
    }
    this->delete_by_it(it);
}

```



```

template<class T>
void list<T>::insert_by_it(containers::list<T>::forward_iterator ins_it, T& value) {
    if (first != nullptr) {
        if (ins_it == this->begin()) {
            std::shared_ptr<element> tmp = std::shared_ptr<element>(new
element{ value });
            tmp->next_element = first;
            first->prev_element = tmp;
            first = tmp;
            if (tmp->value.area() > tmp->next_element->value.area()) {
                throw std::logic_error("Area is too big");
            }
            size++;
            return;
        } else {
            if (ins_it.it_ptr == nullptr) {
                std::shared_ptr<element> tmp = std::shared_ptr<element>(new
element{value});

                tmp->prev_element = push_impl(first);
                push_impl(first)->next_element = std::shared_ptr<element>(tmp);
                if (tmp->value.area() < tmp->prev_element->value.area()) {
                    throw std::logic_error("Area is too low");
                }
                size++;
                return;
            } else {
                std::shared_ptr<element> tmp = std::shared_ptr<element>(new
element{value});
                tmp->prev_element = ins_it.it_ptr->prev_element;
                tmp->next_element = ins_it.it_ptr->prev_element->next_element;
                ins_it.it_ptr->prev_element = tmp;
                tmp->prev_element->next_element = tmp;

                if (tmp->value.area() > tmp->next_element->value.area()) {
                    throw std::logic_error("Area is too big");
                }
                if (tmp->value.area() < tmp->prev_element->value.area()) {
                    throw std::logic_error("Area is too low");
                }
            }
        }
    }
    } else first=std::shared_ptr<element>(new element{value});
}

```

```

        size++;
    }

template<class T>
void list<T>::insert_by_number(size_t N, T& value) {
    forward_iterator it = this->begin();
    for (size_t i = 0; i < N; ++i) {
        ++it;
    }
    this->insert_by_it(it, value);
}

template<class T>
typename list<T>::forward_iterator list<T>::element::next() {
    return forward_iterator(this->next_element.get());
}

template<class T>
list<T>::forward_iterator::forward_iterator(containers::list<T>::element *ptr) {
    it_ptr = ptr;
}

template<class T>
T& list<T>::forward_iterator::operator*() {
    return this->it_ptr->value;
}

template<class T>
typename list<T>::forward_iterator& list<T>::forward_iterator::operator++() {
    if (it_ptr == nullptr) throw std::logic_error ("Выход за границу списка");
    *this = it_ptr->next();
    return *this;
}

template<class T>
typename list<T>::forward_iterator list<T>::forward_iterator::operator++(int) {
    forward_iterator old = *this;
    ++*this;
    return old;
}

template<class T>
bool list<T>::forward_iterator::operator==(const forward_iterator& other) const {
    return it_ptr == other.it_ptr;
}

template<class T>

```

```

list<T>& list<T>::operator=(list<T>&& other){
    size = other.size;
    first = std::move(other.first);
}

template<class T>
bool list<T>::forward_iterator::operator!=(const forward_iterator& other) const {
    return it_ptr != other.it_ptr;
}

template<class T>
T& list<T>::operator[](size_t index) {
    if (index < 0 || index >= size) {
        throw std::out_of_range("Выход за границу списка");
    }
    forward_iterator it = this->begin();
    for (size_t i = 0; i < index; i++) {
        it++;
    }
    return *it;
}
}

```

2. Ссылка на репозиторий на GitHub.

https://github.com/a1dv/oop_exercise_05.git

3. Набор тестов.

test_01.txt:

```

1
0 0 2 0 2 2 1 3 0 2
0
1
-4 0 4 0 4 4 0 6 -4 4
1
3
0
3
5
5
2
1
5
6

```

test_02.txt:

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

4. Результаты выполнения тестов.

```
test_01.result:
1 pentagons
Coordinates are:
{
0 0
2 0
2 2
1 3
0 2
}
Coordinates are:
{
-4 0
4 0
4 4
0 6
-4 4
}
```

```
Coordinates are:
{
0 0
2 0
2 2
1 3
0 2
}
```

```
test_02.result:
2 pentagons
```

Coordinates are:

```
{  
0 0  
-2 0  
-2 -2  
-1 -3  
0 -2  
}
```

Coordinates are:

```
{  
4 0  
-4 0  
-4 -4  
0 -6  
4 -4  
}
```

Coordinates are:

```
{  
4 0  
-4 0  
-4 -4  
0 -6  
4 -4  
}
```

5. Объяснение результатов работы программы.

- 1) Ввод осуществляется через поток стандартного ввода
- 2) Вывод осуществляется через поток стандартного вывода.
- 3) С помощью класса point реализуется запись в память координат в двумерном пространстве.
- 4) В классе pentagon реализованы функции для работы с пятиугольниками
- 5) В классе hexagon реализованы функции для работы с шестиугольниками
- 6) В классе octagon реализованы функции для работы с восьмиугольниками
- 7) В классе list реализованы функции для работы со списками

6. Вывод.

Изучил основы работы с итераторами.