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.

Студент:	Суворова С. А.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	22
Оценка:	
Дата:	

Москва
2019

1.**Код на C++:**

point.h:

```
#ifndef D_POINT_H_
#define D_POINT_H_

#include <iostream>

template<class T>
struct point {
    double x,y;
    point<T> point_1(double x, double y);
};

template<class T>
point<T> point<T>::point_1(double x, double y) {
    point<T> p;
    p.x=x;
    p.y=y;
    return p;
}

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

template<class T>
std::ostream& operator<< (std::ostream& os, const point<T>& p){
    os << p.x << " " << p.y << " ";
    return os;
}

template<class T>
point<T> operator+(point<T> x1,point<T> x2){
    point<T> x3;
    x3.x=x1.x+x2.x;
    x3.y=x1.y+x2.y;
    return x3;
}

template<class T>
point<T>& operator/= (point<T>& x1, int number){
    x1.x=x1.x/number;
    x1.y=x1.y/number;
    return x1;
}

/*
template<class T>
std::istream& operator>>(std::istream& is, point<T>& p);

template<class T>
std::ostream& operator<<(std::ostream& os,const point<T>& p);

template<class T>
point<T> operator+(point<T> x1,point<T> x2);

template<class T>
point<T>& operator/= (point<T>& x1, int number);
*/
#endif
```

five_angles.h:

```
#ifndef D_FIVE_ANGLES_H_
#define D_FIVE_ANGLES_H_

#include <iostream>
#include "point.h"

template<class T>
struct five_angles {

    five_angles(std::istream &is);

    point<T> center() const ;
    void print() const ;
    double square() const ;


point<T> one,two,three,four,five;

};

template<class T>
five_angles<T>::five_angles(std::istream &is){
    is >> one >> two >> three >> four >> five;
}

template<class T>
point<T> five_angles<T>::center() const {
    point<T> p;
    p=one+two+three+four+five;
    p/=5;
    return p;
}

template<class T>
void five_angles<T>::print() const {
    std::cout << one << " " << two << " " << three << " " << four << " " <<
five <<"\n";
}

template<class T>
double five_angles<T>::square() const {
    double s=0;
    s=(one.x*two.y+two.x*three.y+three.x*four.y+four.x*five.y+five.x*one.y-
two.x*one.y-
       three.x*two.y-four.x*three.y-five.x*four.y-one.x*five.y)/2;
    if(s<0){
        return -s;
    }else {
        return s;
    }
}
#endif
```

queue.h:

```
#ifndef D_QUEUE_H_
#define D_QUEUE_H_
```

```
#include <iostream>
#include "five_angles.h"
#include <memory>
#include <functional>
#include <cassert>
#include <iterator>


namespace containers {

    template<class T>
    struct queue {
    private:
        struct node;

    public:
        queue() = default;

        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(node *ptr);

            T &operator*();

            forward_iterator &operator++();

            forward_iterator operator++(int);

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

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

        private:
            node *ptr_;

            friend queue;

        };

        forward_iterator begin();

        forward_iterator end();

        void insert(const forward_iterator &it, const T &value);

        void erase(const forward_iterator &it);

        T pop();

        void push(const T &value);

        T top();

        node *end_node = nullptr;

        node *end_help(node *ptr);

    private:
```

```
        struct node {
            T value;
            std::unique_ptr<node> next = nullptr;
            node *parent = nullptr;

            forward_iterator nextf();
        };

        std::unique_ptr<node> root = nullptr;
    };


//

    template<class T>
    typename queue<T>::node *queue<T>::end_help(containers::queue<T>::node
*ptr) {
        if ((ptr == nullptr) || (ptr->next == nullptr)) {
            return ptr;
        }
        return queue<T>::end_help(ptr->next.get());
    }


    template<class T>
    typename queue<T>::forward_iterator queue<T>::begin() {
        if (root == nullptr) {
            return nullptr;
        }
        forward_iterator it(root.get());
        it.ptr_->parent= nullptr;
        it.ptr_->next=std::move(root->next);
        return it;
    }

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

    template<class T>
    void queue<T>::insert(const queue<T>::forward_iterator &it, const T
&value) {
        std::unique_ptr<node> new_node(new node{value});
        new_node->parent=it.ptr_;
        if(it.ptr_) {
            new_node->next = std::move(it.ptr_->next);
            if (it.ptr_->next) {
                it.ptr_->next->parent = new_node.get();
            }
            it.ptr_->next=std::move(new_node);
        }else{
            new_node->next= nullptr;
            queue<T>::root=std::move(new_node);
        }
        end_node=end_help(root.get());
    }

    template<class T>
    void queue<T>::erase(const queue<T>::forward_iterator &it) {
        if (it.ptr_ == nullptr) {
            throw std::logic_error("erasing invalid iterator");
        }
        node *parent = it.ptr_->parent;
```

```
        std::unique_ptr<node> &pointer_from_parent = [&]() ->
std::unique_ptr<node> & {
            if(it.ptr_ == root.get()){
                return root;
            }
            return it.ptr_->parent->next;
        }();
        pointer_from_parent = std::move(it.ptr_->next);
        if (pointer_from_parent) {
            pointer_from_parent->parent = parent;
        }
        end_node=end_help(root.get());
    }

//
    template<class T>
    typename queue<T>::forward_iterator queue<T>::node::nextf() {
        return this->next.get();
    }

    template<class T>
    queue<T>::forward_iterator::forward_iterator(node *ptr): ptr_{ptr} {}

    template<class T>
    T &queue<T>::forward_iterator::operator*() {
        return ptr_->value;
    }

    template<class T>
    typename queue<T>::forward_iterator
&queue<T>::forward_iterator::operator++() {
        *this = ptr_->nextf();
        return *this;
    }

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

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

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

    template<class T>
    T queue<T>::top() {
        if (queue<T>::root == nullptr) {
            throw std::logic_error("no elements");
        }
        return queue<T>::root->value;
    }

    template<class T>
```

```
    T queue<T>::pop() {
        if (queue<T>::root == nullptr) {
            throw std::logic_error("no elements");
        }
        T result = queue<T>::root->value;
        erase(queue<T>::begin());
        return result;
    }

    template<class T>
    void queue<T>::push(const T &value) {
        forward_iterator it(end_node);
        insert(it,value);
    }

}
#endif
```

main.cpp:

```
#include <iostream>
#include "five_angles.h"
#include "point.h"
#include "queue.h"
#include <string.h>
#include <algorithm>

int main() {
    char str[10];
    containers::queue<five_angles<double> > q;
    while(std::cin >> str){
        if(strcmp(str,"push")==0){
            five_angles<double> five_angle(std::cin);
            q.push(five_angle);
        }else if(strcmp(str,"pop")==0){
            try {
                five_angles<double> f = q.pop();
                f.print();
                std::cout << "\n";
            }catch (std::exception& ex){
                std::cout <<ex.what() << "\n";
            }
        }else if(strcmp(str,"top")==0){
            try {
                q.top().print();
                std::cout << "\n";
            }catch (std::exception& ex){
                std::cout <<ex.what() << "\n";
            }
        }else if(strcmp(str,"square")==0){
            int g;
            std::cin >> g;
            long res=std::count_if(q.begin(),q.end(),[g](five_angles<double>
f){ return f.square() < g;});
            std::cout << res << "\n";
        }else if(strcmp(str,"all")==0){
            std::for_each(q.begin(),q.end(),[](five_angles<double> f){
f.print(); });
            std::cout<< "\n";
        }

    }
    return 0;
```

```
}
```

2. **Ссылка на репозиторий в GitHub:**
https://github.com/Suvorova-Sofya/oop_exercise_05

3.**Набор testcases:**

test1:
```
pop
no elements
push 1 1 2 2 3 3 4 4 5 5
pop
1 1 2 2 3 3 4 4 5 5
pop
no elements
```

test2:
```
push 1 1 2 2 3 3 4 4 5 5
push 2 2 3 3 4 4 5 5 6 6
all
1 1 2 2 3 3 4 4 5 5
2 2 3 3 4 4 5 5 6 6
```

test3:
```
push 1 1 2 2 3 3 4 4 5 5
push 2 2 3 3 4 4 5 5 6 6
square 10
2
square 0
0
```

4.**Результаты выполнения программы:**

test1:
```
pop
no elements
push 1 1 2 2 3 3 4 4 5 5
pop
1 1 2 2 3 3 4 4 5 5

pop
no elements
```

test2:
```
push 1 1 2 2 3 3 4 4 5 5
push 2 2 3 3 4 4 5 5 6 6
all
1 1 2 2 3 3 4 4 5 5
2 2 3 3 4 4 5 5 6 6
```

test3:
```
push 1 1 2 2 3 3 4 4 5 5
push 2 2 3 3 4 4 5 5 6 6
square 10
2
square 0
0
```

5**. Объяснение результатов работы программы:**
Пользователь вводит команду , и если команда была push -координаты фигуры. Далее программа выполняет определенное действие с очередью взависимости от команды и либо возвращает определенное значение ,либо нет.

6.**Вывод:**
В данной программе показывается , как создавать контейнеры, благодаря которым упрощается дальнейшая работа с различными типами данных, потому что контейнер будет работать одинаково с любыми типами данных.