

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 метапрограммирования.**

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;
}
```

templates.h:

```
#ifndef D_TEMPLATES_H_
```

```

#define D_TEMPLATES_H_ 1

#include <tuple>
#include <type_traits>

#include "five_angles.h"
#include "six_angles.h"
#include "eight_angles.h"
#include "point.h"

template<class T>
struct is_point : std::false_type {};

template<class T>
struct is_point<point<T>> : std::true_type {};

template<class T>
struct is_figurelike_tuple : std::false_type {};

template<class Head, class... Tail>
struct is_figurelike_tuple<std::tuple<Head, Tail...>> :
    std::conjunction<is_point<Head>,
        std::is_same<Head, Tail>...> {};

template<class Type, size_t SIZE>
struct is_figurelike_tuple<std::array<Type, SIZE>> :
    is_point<Type> {};

template<class T>
inline constexpr bool is_figurelike_tuple_v =
    is_figurelike_tuple<T>::value;

template<class T, class = void>
struct has_print_method : std::false_type {};

template<class T>
struct has_print_method<T,
    std::void_t<decltype(std::declval<const T>().print())>> :
    std::true_type {};

template<class T>
inline constexpr bool has_print_method_v =
    has_print_method<T>::value;

template<class T>
std::enable_if_t<has_print_method_v<T>, void>
print(const T& figure) {
    figure.print();
}

template<size_t ID, class T>
void single_print(const T& t) {

```

```

    std::cout << std::get<ID>(t);
    return ;
}

template<size_t ID, class T>
void recursive_print(const T& t) {
    if constexpr (ID < std::tuple_size_v<T>){
        single_print<ID>(t);
        recursive_print<ID+1>(t);
        return ;
    }else{
        return ;
    }
}

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, void>
print(const T& fake) {
    return recursive_print<0>(fake);
}

//center

template<class T, class = void>
struct has_center_method : std::false_type {};

template<class T>
struct has_center_method<T,
    std::void_t<decltype(std::declval<const T>().center())>> :
    std::true_type {};

template<class T>
inline constexpr bool has_center_method_v =
    has_center_method<T>::value;

template<class T>
std::enable_if_t<has_center_method_v<T>, point<double>>
center(const T& figure) {
    return figure.center();
}

template<class T>
inline constexpr const int tuple_size_v =
    std::tuple_size<T>::value;

template<size_t ID, class T>
point<double> single_center(const T& t) {
    point<double> p;
    p=std::get<ID>(t);
    p/=tuple_size_v<T>;
    return p;
}

```

```

template<size_t ID, class T>
point<double> recursive_center(const T& t) {
    if constexpr (ID < std::tuple_size_v<T>){
        return single_center<ID>(t) + recursive_center<ID+1>(t);
    }else{
        point<double> p;
        p.point_1(0,0);
        return p;
    }
}

```

```

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, point<double>>
center(const T& fake) {
    return recursive_center<0>(fake);
}

```

//square

```

template<class T, class = void>
struct has_square_method : std::false_type {};

```

```

template<class T>
struct has_square_method<T,
    std::void_t<decltype(std::declval<const T>().square())>> :
    std::true_type {};

```

```

template<class T>
inline constexpr bool has_square_method_v =
    has_square_method<T>::value;

```

```

template<class T>
std::enable_if_t<has_square_method_v<T>, double>
square(const T& figure) {
    return figure.square();
}

```

```

template<size_t ID, class T>
double single_square(const T& t) {
    const auto& a = std::get<0>(t);
    const auto& b = std::get<ID - 1>(t);
    const auto& c = std::get<ID>(t);
    const double dx1 = b.x - a.x;
    const double dy1 = b.y - a.y;
    const double dx2 = c.x - a.x;
    const double dy2 = c.y - a.y;
    return std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
}

```

```

template<size_t ID, class T>
double recursive_square(const T& t) {

```

```

    if constexpr (ID < std::tuple_size_v<T>){
        return single_square<ID>(t) + recursive_square<ID + 1>(t);
    }else{
        return 0;
    }
}

```

```

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, double>
square(const T& fake) {
    return recursive_square<2>(fake);
}

```

```

#endif // D_TEMPLATES_H_

```

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

```

six_angles.h:

```

#ifndef D_SIX_ANGLES_H_
#define D_SIX_ANGLES_H_

```

```

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

```

```

template<class T>
struct six_angles {

    six_angles(std::istream &is);

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

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

```

```

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

```

```

template<class T>
point<T> six_angles<T>::center() const {
    point<T> p;
    p=one+two+three+four+five+six;
    p/=6;
    return p;
}

```

```

template<class T>

```

```

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

```

```

template<class T>
double six_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*six.y+six.x*one.y-
two.x*one.y-
    three.x*two.y-four.x*three.y-five.x*four.y-six.x*five.y-one.x*six.y)/2;
    if(s<0){
        return -s;
    }else {
        return s;
    }
}

```

```

#endif

```

eight_angles.h:

```

#ifndef D_EIGHT_ANGLES_H_
#define D_EIGHT_ANGLES_H_

```

```

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

```

```

template<class T>
struct eight_angles {

    eight_angles(std::istream &is);

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

    point<T> one,two,three,four,five,six,seven,eight;
};

```

```

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

```

```

template<class T>
point<T> eight_angles<T>::center() const {
    point<T> p;
    p=one+two+three+four+five+six+seven+eight;
    p/=8;
    return p;
}

```



```

}

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

template<class T>
double eight_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*six.y+six.x*seven.y+seven.
x*eight.y+
    eight.x*one.y-two.x*one.y-three.x*two.y-four.x*three.y-five.x*four.y-six.x*five.y-
seven.x*six.y
    -eight.x*seven.y-one.x*eight.y)/2;
    if(s<0){
        return -s;
    }else {
        return s;
    }
}

#endif

```

main.cpp:

```

#include <iostream>
#include <stdio.h>
#include <vector>
#include <string.h>

#include "five_angles.h"
#include "six_angles.h"
#include "eight_angles.h"
#include "templates.h"

int main() {
    char str[12];
    std::cin >> str;
    if(strcmp("five_angles",str)==0) {
        five_angles<double> real_five_angles(std::cin);
        print(real_five_angles);
        std::tuple<point<double>, point<double>, point<double>, point<double>, point<double>>
            fake_five_angles{real_five_angles.one,
                real_five_angles.two,
                real_five_angles.three,
                real_five_angles.four,
                real_five_angles.five};
    }
}

```

```

    print(fake_five_angles);
    std::cout << std::endl;
    std::cout << center(real_five_angles) << "\n";
    std::cout << center(fake_five_angles) << "\n";
    std::cout << square(real_five_angles) << "\n";
    std::cout << square(fake_five_angles);
    std::cout << std::endl;
} else {
    if(strcmp("six_angles",str)==0) {
        six_angles<double> real_six_angles(std::cin);
        print(real_six_angles);
        std::tuple<point<double>, point<double>, point<double>, point<double>,
point<double>, point<double>>
            fake_six_angles{real_six_angles.one,
                real_six_angles.two,
                real_six_angles.three,
                real_six_angles.four,
                real_six_angles.five,
                real_six_angles.six };
        print(fake_six_angles);
        std::cout << std::endl;
        std::cout << center(real_six_angles) << "\n";
        std::cout << center(fake_six_angles) << "\n";
        std::cout << square(real_six_angles) << "\n";
        std::cout << square(fake_six_angles);
        std::cout << std::endl;
    } else {
        if(strcmp("eight_angles",str)==0) {
            eight_angles<double> real_eight_angles(std::cin);
            print(real_eight_angles);
            std::tuple<point<double>, point<double>, point<double>, point<double>,
point<double>, point<double>, point<double>,
            point<double>> fake_eight_angles{real_eight_angles.one,
                real_eight_angles.two,
                real_eight_angles.three,
                real_eight_angles.four,
                real_eight_angles.five,
                real_eight_angles.six,
                real_eight_angles.seven,
                real_eight_angles.eight };
            print(fake_eight_angles);
            std::cout << std::endl;
            std::cout << center(real_eight_angles) << "\n";
            std::cout << center(fake_eight_angles) << "\n";
            std::cout << square(real_eight_angles) << "\n";
            std::cout << square(fake_eight_angles);
            std::cout << std::endl;
        }
    }
}

return 0;

```

}

2. Ссылка на репозиторий в GitHub:

https://github.com/Suvorova-Sofya/oop_exercise_04

3.Набор testcases:

test1:

five_angle

0 0 1 1 2 2 3 3 4 4

0 0 1 1 2 2 3 3 4 4

1 2 2 -1 -3 -3 -4 0 -3 2

2 2

-1.4 0

0

21

test2:

six_angle

0 0 1 1 2 2 3 3 4 4 5 5

0 0 1 1 2 2 3 3 4 4 5 5

1 2 2 -1 1 -3 -3 -3 -4 0 -3 2

2.5 2.5

-1 -0.5

0

25

test3:

eight_angle

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7

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

3.5 3.5

-1 -1.625

0

31

4. Результаты выполнения программы:

test1:

five_angle

0 0 1 1 2 2 3 3 4 4

0 0 1 1 2 2 3 3 4 4

1 2 2 -1 -3 -3 -4 0 -3 2

2 2

-1.4 5.55112e-17

0

21

test2:

six_angles

0 0 1 1 2 2 3 3 4 4 5 5

0 0 1 1 2 2 3 3 4 4 5 5

1 2 2 -1 1 -3 -3 -3 -4 0 -3 2

2.5 2.5

-1 -0.5

0

25

test3:

eight_angle

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7

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

3.5 3.5

-1 -1.625

0

31

5. Объяснение результатов работы программы:

Пользователь вводит название фигуры, координаты фигуры. Программа выполняет все три функции с указанной фигурой и с tuple ,соответствующий данной фигуре, и после выводит все результаты.

6. Вывод:

В данной программе показывается каким образом можно использовать такие возможности языка C++, как шаблоны, которые помогают выполнять указанные в них действия для всех объектов, подходящих по описанию в шаблоне, некоторого класса, которые по свойствам (тип данных, количество элементов и т. д.) могут различаться.