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 ;

private:

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 ;

private:

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

};

#include <iostream>

#include "six\_angles.h"

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 ;

private:

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

};

#include <iostream>

#include "eight\_angles.h"

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() {

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{{1, 2}, {2, -1}, {-3, -3}, {-4,0}, {-3,2}};

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;

std::cout << std::endl;

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{{1,2}, {2, -1}, {1, -3}, {-3,-3}, {-4,0},{-3,2}};

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;

std::cout <<std::endl;

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{{1, 2}, {2, -1}, {1, -3}, {0,-5}, {-2,-5},{-3,-3},{-4,0},{-3,2}};

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 5.55112е-17

0

21

test2:

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 С++, 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.