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: | 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;

}

};

**tempaltes.h:**

#pragma once

#include <tuple>

#include <type\_traits>

#include "point.h"

template<class T>

struct is\_vertex : std::false\_type {};

template<class T>

struct is\_vertex<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\_vertex<Head>,

std::is\_same<Head, Tail>...> {};

template<class Type, size\_t SIZE>

struct is\_figurelike\_tuple<std::array<Type, SIZE>> :

is\_vertex<Type> {};

template<class T>

inline constexpr bool is\_figurelike\_tuple\_v =

is\_figurelike\_tuple<T>::value;

template<class T,class = void>

struct has\_area\_method : std::false\_type {};

template<class T>

struct has\_area\_method<T,

std::void\_t<decltype(std::declval<const T>().area())>> :

std::true\_type {};

template<class T>

inline constexpr bool has\_area\_method\_v =

has\_area\_method<T>::value;

template<class T>

std::enable\_if\_t<has\_area\_method\_v<T>, double>

area(const T& figure) {

return figure.area();

}

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::cout))>> :

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,std::ostream& os) {

return figure.print(os);

}

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< decltype(std::declval<const T>().center().x)>>

center (const T& figure) {

return figure.center();

}

template<size\_t ID, class T>

double single\_area(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\_area(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>){

return single\_area<ID>(t) + recursive\_area<ID + 1>(t);

}else{

return 0;

}

}

template<class T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, double>

area(const T& fake) {

return recursive\_area<2>(fake);

}

template<size\_t ID, class T>

double single\_center\_x(const T& t) {

return std::get<ID>(t).x / std::tuple\_size\_v<T>;

}

template<size\_t ID, class T>

double single\_center\_y(const T& t) {

return std::get<ID>(t).y / std::tuple\_size\_v<T>;

}

template<size\_t ID, class T>

double recursive\_center\_x(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>) {

return single\_center\_x<ID>(t) + recursive\_center\_x<ID + 1>(t);

} else {

return 0;

}

}

template<size\_t ID, class T>

double recursive\_center\_y(const T& t) {

if constexpr (ID < std::tuple\_size\_v<T>) {

return single\_center\_y<ID>(t) + recursive\_center\_y<ID + 1>(t);

} else {

return 0;

}

}

template<class T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, point<double>>

center(const T& tup) {

return {recursive\_center\_x<0>(tup), recursive\_center\_y<0>(tup)};

}

template<size\_t ID, class T>

void single\_print(const T& t, std::ostream& os) {

os << std::get<ID>(t) << ' ';

}

template<size\_t ID, class T>

void recursive\_print(const T& t, std::ostream& os) {

if constexpr (ID < std::tuple\_size\_v<T>) {

single\_print<ID>(t, os);

os << '\n';

recursive\_print<ID + 1>(t, os);

} else {

return;

}

}

template<class T>

std::enable\_if\_t<is\_figurelike\_tuple\_v<T>, void>

print(const T& tup, std::ostream& os) {

recursive\_print<0>(tup, os);

os << std::endl;

}

**main.cpp:**

#include <iostream>

#include "point.h"

#include "pentagon.h"

#include "hexagon.h"

#include "octagon.h"

#include "templates.h"

int main()

{

std::cout << "1 - pentagon \n 2 - hexagon \n 3 - octagon \n 4 - exit";

int i;

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

while(true) {

std::cin >> i;

if (i == 1) {

pentagon <double> p(std::cin);

std::cout << "Enter a tuple for pentagon:\n";

std::cin >> a1 >> a2 >> a3 >> a4 >> a5;

p.print(std::cout);

std::tuple <point<double>, point<double>, point<double>, point<double>, point <double>> p1{a1, a2, a3, a4, a5};

print(p1, std::cout);

std::cout << "Area:\n" << area(p1) << "\nCenter:\n" << center(p1);

}

if (i == 2) {

hexagon <double> h(std::cin);

std::cout << "Enter a tuple hexagon:\n";

std::cin >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;

h.print(std::cout);

std::tuple <point<double>, point<double>, point<double>, point<double>, point <double>, point <double>> h1{a1, a2, a3, a4, a5, a6};

print(h1, std::cout);

std::cout << "Area:\n" << area(h1) << "\nCenter:\n" << center(h1);

}

if (i == 3) {

octagon <double> o(std::cin);

std::cout << "Enter a tuple octagon:\n";

std::cin >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;

o.print(std::cout);

std::tuple <point<double>, point<double>, point<double>, point<double>, point <double>, point <double>, point <double>, point <double>> o1{a1, a2, a3, a4, a5, a6, a7, a8};

print(o1, std::cout);

std::cout << "Area:\n" << area(o1) << "\nCenter:\n" << center(o1);

}

if (i == 4) {

break;

}

std::cout << "1 - pentagon \n 2 - hexagon \n 3 - octagon \n 4 - exit";

}

}

**CmakeLists.txt:**

project(lab4)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(lab4

pain.cpp

point.h

pentagon.h

hexagon.h

octagon.h

templates.h

)

**2. Ссылка на репозиторий на GitHub.**

**https://github.com/a1dv/oop\_exercise\_04.git**

**3. Набор тестов.**

test\_01.txt:

1

0 0 2 0 2 2 1 3 0 2

0 0 2 0 2 2 1 3 0 2

2

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

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

3

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

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

4

test\_02.txt:

1

0 0.5 0.5 0 1 0 1 0.5 0.5 0.5

0 0.5 0.5 0 1 0 1 0.5 0.5 0.5

2

0 0.5 0.5 0 1 0 1.5 0.25 1 0.5 0.5 0.5

0 0.5 0.5 0 1 0 1.5 0.25 1 0.5 0.5 0.5

3

0 0.5 0.5 0 0.75 -0.5 1 0 1.5 0.25 1 0.5 0.75 0.75 0.5 0.5

0 0.5 0.5 0 0.75 -0.5 1 0 1.5 0.25 1 0.5 0.75 0.75 0.5 0.5

4

test\_03.txt:

1

0 100 0 0 100 0 150 50 100 100

0 100 0 0 100 0 150 50 100 100

2

0 100 -50 50 0 0 100 0 150 50 100 100

0 100 -50 50 0 0 100 0 150 50 100 100

3

0 100 -50 50 0 0 50 -50 100 0 150 50 50 100 100 50 150

0 100 -50 50 0 0 50 -50 100 0 150 50 50 100 100 50 150

4

**4. Результаты выполнения тестов.**

test\_01.result:

Coordinates are:

{

0 0

2 0

2 2

1 3

0 2

}

0 0

2 0

2 2

1 3

0 2

Area:

5

Center:

1 1.4

Coordinates are:

{

0 0

1 -1

2 0

2 2

1 3

0 2

}

0 0

1 -1

2 0

2 2

1 3

0 2

Area:

6

Center:

1 1

Coordinates are:

{

0 0

1 -1

2 0

3 1

2 2

1 3

0 2

-1 1

}

0 0

1 -1

2 0

3 1

2 2

1 3

0 2

-1 1

Area:

8

Center:

1 1

test\_02.result:

Coordinates are:

{

0 0.5

0.5 0

0.75 -0.5

1 0

1.5 0.25

1 0.5

0.75 0.75

0.5 0.5

}

0 0.5

0.5 0

0.75 -0.5

1 0

1.5 0.25

1 0.5

0.75 0.75

0.5 0.5

Area:

0.9375

Center:

0.75 0.25

test\_03.result:

Coordinates are:

{

0 100

0 0

100 0

150 50

100 100

}

12500

70 50

Coordinates are:

{

0 100

-50 50

0 0

100 0

150 50

100 100

}

15000

50 50

Coordinates are:

{

0 100

-50 50

0 0

50 -50

100 0

150 50

50 100

100 50

}

15000

50 37.5

**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 двухмерном 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

**6. Вывод.**

Изучил основы метапрограммирования.