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.