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:	Касимов М.М.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	6
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Pentagon.h:

```
#pragma once

#include "point.h"

template<class T>
struct pentagon {
private:
   point<T> a1,a2,a3,a4,a5;
public:
   point<T> center() const;
   void print(std::ostream& os) const ;
   double area() const;
   pentagon(std::istream& is);
};

template<class T>
double pentagon<T>::area() const {
   return  (0.5) * 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 )));
}

template<class T>
pentagon<T>::pentagon(std::istream& is) {
   is >> a1 >> a2 >> a3 >> a4 >> a5;
}

template<class T>
void pentagon<T>::print(std::ostream& os) const {
   os << "coordinate:\n"<<"{\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n'
<< a5 << '\n' << "}\n";
}

template<class T>
point<T> pentagon<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;
   return {x,y};
}
```

Point.h:

```
#pragma once
```

```
#include <iostream>
#include <algorithm>

template<class T>
struct point {
    T x;
    T y;
};

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;
}
```

Hexagon.h:

```
#pragma once

#include "point.h"

template<class T>
struct hexagon {
private:
    point<T> a1,a2,a3,a4,a5,a6;
public:
    point<T> center() const;
    void print(std::ostream& os) const;
    double area() const;
    hexagon(std::istream& is);
};

template<class T>
point<T> hexagon<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;
    return {x,y};
}

template<class T>
```

```
hexagon<T>::hexagon(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}

template<class T>
void hexagon<T>::print(std::ostream& os) const {
    os << "coordinate:\n"<<"{\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n'
<< a5 << '\n' << a6 << "}\n";
}

template<class T>
double hexagon<T>::area() const {
    return (0.5) * 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
)));
}
```

Octagon.h:

```
#pragma once

#include "point.h"

template<class T>
struct octagon {
private:
    point<T> a1,a2,a3,a4,a5,a6,a7,a8;
public:
    point<T> center() const;
    void print(std::ostream& os) const;
    double area() const;
    octagon(std::istream& is);
};

template<class T>
point<T> octagon<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;
    return {x,y};
}

template<class T>
octagon<T>::octagon(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
}
```

```
template<class T>
void octagon<T>::print(std::ostream& os) const {
   os << "coordinate:\n"<<"{\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n'
<< a5 << '\n' << a6 << '\n' << a7 << '\n' << a8 << "}\n";
}

template<class T>
double octagon<T>::area() const {
   return (-0.5) * ((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));
}
```

Templates.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 "pentagon.h"
#include "hexagon.h"
#include "octagon.h"
#include "templates.h"

void help () {
    std::cout << "1 - pentagon\n"
                 "2 - hexagon\n"
                 "3 - octagon\n"
                 "4 - exit\n";
}

int main() {
    int choice;
    point<double> v1,v2,v3,v4,v5,v6,v7,v8;
    help();
    std::cin >> choice;
    while (choice != 4) {
        if (choice == 1) {
            pentagon<double> p(std::cin);
            std::cout << "area: " << area(p) << '\n' << "center: " << center(p) << '\n';
            print(p, std::cout);
            std::cout << "Enter a pentagonal tuple\n";
            std::cin >> v1 >> v2 >> v3 >> v4 >> v5;
            std::tuple<point<double>, point<double>, point<double>, point<double>,
point<double>> p1{v1, v2, v3, v4,v5};
            std::cout << "area: " << area(p1) << '\n' << "center: " << center(p1) << '\n';
            print(p1, std::cout);
        } else if (choice == 2) {
            hexagon<double> h(std::cin);
            std::cout << "area: " << area(h) << '\n' << "center: " << center(h) << '\n';
            print(h, std::cout);
            std::cout << "Enter a pentagonal tuple\n";
            std::cin >> v1 >> v2 >> v3 >> v4 >> v5 >> v6;
            std::tuple<point<double>, point<double>, point<double>, point<double>,
point<double>,point<double>> h1{v1, v2, v3, v4, v5, v6};
            std::cout << "area: " << area(h1) << '\n' << "center: " << center(h1) << '\n';
            print(h1, std::cout);
        } else if (choice == 3) {
            octagon<double> o(std::cin);
            std::cout << "area: " << area(o) << '\n' << "center: " << center(o) << '\n';
            print(o, std::cout);
```

```
            std::cout << "Enter a pentagonal tuple\n";
            std::cin >> v1 >> v2 >> v3 >> v4 >> v5 >> v6 >> v7 >> v8;
            std::tuple<point<double>, point<double>, point<double>, point<double>, point<double>, point<double>, point<double>, point<double>> o1{v1, v2, v3, v4, v5, v6, v7, v8};
            std::cout << "area: " << area(o1) << '\n' << "center: " << center(o1) << '\n';
            print(o1, std::cout);
        } else {
            std::cout << "The command is uncertain\n";
        }
        std::cin >> choice;
    }
    return 0;
}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/magomed2000kasimov/oop_exercise_04

3. Набор тестов.

test_01.test:

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

1 1
3
4 3
4 5
5 6
7 6
8 5
8 3
7 2
5 2
4 3
4 5
5 6
7 6
8 5
8 3
7 2
5 2
4

test_02.test:

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

4. Результаты выполнения тестов.

test_01.result:

1 - pentagon
2 - hexagon
3 - octagon
4 - exit
area: 4
center: 1.8 3.2
coordinate:
{
1 2
1 4
2 5

3 3
2 2
}
Enter a pentagonal tuple
area: 4
center: 1.8 3.2
1 2
1 4
2 5
3 3
2 2

area: 5
center: 1.5 2
coordinate:
{
0 2
0 3
3 3
3 2
2 1
1 1}
Enter a pentagonal tuple
area: 5
center: 1.5 2
0 2
0 3
3 3
3 2
2 1
1 1

area: 14
center: 6 4
coordinate:
{
4 3
4 5
5 6
7 6
8 5
8 3
7 2
5 2}
Enter a pentagonal tuple

area: 14
center: 6 4
4 3
4 5
5 6
7 6
8 5
8 3
7 2
5 2
test_02.result:
1 - pentagon
2 - hexagon
3 - octagon
4 - exit
area: 0
center: 0 0
coordinate:
{
0 0
0 0
0 0
0 0
0 0
}
Enter a pentagonal tuple
area: 3
center: 1.8 2
1 1
1 3
2 3
3 2
2 1

5. Объяснение результатов работы программы.

1) Шаблонная функция center() возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, y – аналогично x. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно.

2) Функция print() печатает координаты всех точек данной фигуры или кортежа. Она определена для моих фигур и tuple. Во втором случае все дело вычисляется рекурсивно.

3) Функция square() вычисляет площадь данной фигуры или совокупности точек в кортеже по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.

6. Вывод.

Выполняя данную лабораторную я получил опыт работы с шаблонами в C++, с системой сборки Cmake, с системой контроля версий git. Узнал о применении шаблонов в метапрограммировании. Понял как трудно писать свои библиотеки, ведь чтобы сделать это нужно хорошо освоить тонкости шаблонов. Также я познакомился с полезными заголовочными файлами <tuple> и <type_traits>, освоил enable_if, decltype и базовые вещи для работы с tuple.
Данная лабораторная работа показала многогранность и мощь языка C++.

7.Литература.

1)лекции по ООП МАИ.
2) Г.Шилдт «C++».