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: | Черненко И.Д |
| Группа: | М80-206Б-18 |
| Преподаватель: | Журавлев А.А. |
| Вариант: | 24 |
| Оценка: | |
| Дата: | |

Москва
2019

## 1. Код программы на языке C++:

**templates.h:**

```
#ifndef TEMPLATES_H_
#define TEMPLATES_H_
#include <utility>
#include <type_traits>
#include <tuple>
#include <cmath>
#include <typeinfo>
#include "triangle.h"
#include "octagon.h"
#include "square.h"
#include "vertex.h"
template<class T>
struct is_vertex : std::false_type {};

template<class T>
struct is_vertex<std::pair<T,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_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 Recursiveprint(const T& t) {
   if constexpr (ID < std::tuple_size_v<T>){
      single_print<ID>(t);
      Recursiveprint<ID+1>(t);
      return ;
   }
   return;
}

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

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>, std::pair<double,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>
std::pair<double,double> single_center(const T& t) {
    std::pair<double,double> v;
    v.first = std::get<ID>(t).first;
    v.second = std::get<ID>(t).second;
    //v /= std::tuple_size_v<T>;
    return v;
}

template<size_t ID, class T>
std::pair<double,double> Recursivecenter(const T& t) {
    if constexpr (ID < std::tuple_size_v<T>){
        return  (single_center<ID>(t) + Recursivecenter<ID+1>(t));
    } else {
        std::pair<double,double> v;
        v.first = 0;
        v.second = 0;
        return v;
    }
}

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

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();
}
```

**square.h:**

```
#ifndef OOP_EXERCISE_04_SQUARE_H
#define OOP_EXERCISE_04_SQUARE_H
template<class T>
struct TSquare {
    using type = T;
    using vertex = std::pair<T,T>;
    vertex A, B, C, D;
    TSquare(T x1, T y1, T x2, T y2, T x3, T y3, T x4, T y4) :
            A(x1, y1), B(x2, y2), C(x3,y3), D(x4,y4)
    {
        if (!is_square())
            throw std::logic_error("not a square\n");
    }
    TSquare(std::istream& is) {
        std::cin >> A >> B >> C >> D;
        if (!is_square())
            throw std::logic_error("not a square\n");

    }
    std::pair<double,double> center() const;
    void print() const;
    double area() const;
    bool is_square();
};

template <class T>

double TSquare<T>::area() const{
    return ((C.first - A.first) * (C.first - A.first) + (C.second - A.second) * (C.second -
A.second)) * 0.5;
}

template <class T>
std::pair<double, double> TSquare<T>::center() const{
    return      std::make_pair(static_cast<double>(A.first      +      C.first)      /      2,
static_cast<double>(A.second + C.second) / 2);
}
```

**octagon.h:**

```
#ifndef OOP_EXERCISE_04_OCTAGON_H
```

```
#define OOP_EXERCISE_04_OCTAGON_H
template<class T>
struct TOctagon{
    using type = T;
    using vertex = std::pair<T,T>;
    vertex A, B, C, D, E, F, G, H;
    TOctagon(T x1, T y1, T x2, T y2, T x3, T y3, T x4, T y4, T x5, T y5, T x6, T y6, T
x7, T y7, T x8, T y8) :
            A(x1, y1), B(x2, y2), C(x3, y3), D(x4, y4), E(x5, y5), F(x6, y6), G(x7, y7),
H(x8, y8)
    {}
    TOctagon(std::istream& is) {
        std::cin >> A >> B >> C >> D >> E >> F >> G >> H;
    }
    std::pair<double,double> center() const;
    void print() const;
    double area() const;
};


template <class T>

double TOctagon<T>::area() const{
    return fabs(((A.first * B.second) + (B.first * C.second) + (C.first * D.second) +
(D.first * E.second) + (E.first * F.second) + (F.first * G.second) + (G.first *
H.second) + (H.first * A.second) - (B.first * A.second) - (C.first * B.second) - (D.first
* C.second) - (E.first * D.second) - (F.first * E.second) - (G.first * F.second) - (H.first
* G.second) - (A.first * H.second)) * 0.5);
}

template <class T>
void TOctagon<T>::print() const{
    std::cout << A << " " << B << " " << C << " " << D << " " << E << " " << F << " "
<< G << " " << H << "\n";
}

template <class T>

std::pair<double, double> TOctagon<T>::center() const{
    return std::make_pair(static_cast<double>(A.first + B.first + C.first + D.first +
E.first + F.first + G.first + H.first) / 8,static_cast<double>(A.second + B.second +
C.second + D.second + E.second + F.second + G.second + H.second) / 8);
}
#endif //OOP_EXERCISE_04_OCTAGON_H
```

**triangle.h:**

```
#ifndef OOP_EXERCISE_04_TRIANGLE_H
#define OOP_EXERCISE_04_TRIANGLE_H
template<class T>
struct TTriangle {
    using type = T;
    using vertex = std::pair<T,T>;
    vertex A, B, C;
    TTriangle(T x1, T y1, T x2, T y2, T x3, T y3) :
            A(x1, y1), B(x2, y2), C(x3, y3)
    {
        double l = (sqrt((A.first - B.first) * (A.first - B.first) + (A.second - B.second) *
(A.second - B.second)));
        double k = (sqrt((B.first - C.first) * (B.first - C.first) + (B.second - C.second) *
(B.second - C.second)));
        double p = (sqrt((C.first - B.first) * (C.first - B.first) + (C.second - B.second) *
(C.second - B.second)));
        if (l + k <= p || l + p <= k || p + k <= l) {
            throw std::logic_error("Triangle doesn't exist");
        }
    }
    TTriangle(std::istream& is) {
        is >> A >> B >> C;
        double l = (sqrt((A.first - B.first) * (A.first - B.first) + (A.second - B.second) *
(A.second - B.second)));
        double k = (sqrt((B.first - C.first) * (B.first - C.first) + (B.second - C.second) *
(B.second - C.second)));
        double p = (sqrt((C.first - B.first) * (C.first - B.first) + (C.second - B.second) *
(C.second - B.second)));
        if (l + k <= p || l + p <= k || p + k <= l) {
            throw std::logic_error("Triangle doesn't exist");
        }
    }

    std::pair<double,double> center() const;
    void print() const;
    double area() const;
};


template <class T>
double TTriangle<T>::area() const{
    return (fabs((A.first - C.first) * (B.second - C.second) - (B.first - C.first) *
(A.second - C.second)) * 0.5);
}
```

```
template <class T>
void TTriangle<T>::print() const{
    std::cout << A << " " << B << " " << C << "\n";
}

template <class T>
std::pair<double,double> TTriangle<T>::center() const {
    double x0 = (A.first + B.first + C.first) / 3;
    double y0 = (A.second + B.second + C.second) / 3;
    return std::make_pair(x0,y0);
}
#endif //OOP_EXERCISE_04_TRIANGLE_H
```

## 2. Ссылка на репозиторий на GitHub

https://github.com/IllCher/oop_exercise_04

## 3. Набор googletests.

```
#include <gtest/gtest.h>
#include "../src/templates.h"

TEST(TriangleValidation, TestingValidation) {
    ASSERT_NO_THROW(TTriangle(0, 1, 0, 0, 2, -1));
    ASSERT_NO_THROW(TTriangle(2.2, 3.2, 5.32, 6.7, 89.43, -143.3));
    ASSERT_ANY_THROW(TTriangle(0, 0, 0, 0, 1, 1));
    ASSERT_ANY_THROW(TTriangle(0.0, 0.0, 0.0, 0.0, 1.1, 1.1));
}

TEST(TemplateAreaTest, TestingAreaTriangle) {
    TTriangle<int> triangle1(0, 2, 2, -1, -3, -1);
    EXPECT_EQ(area(triangle1), 7.5);
    TTriangle<double> triangle2(0, 1, 2, -1, -2, -1);
    EXPECT_DOUBLE_EQ(area(triangle2), 4);
}

TEST(TemplateAreaTest, TestingAreaOctagon) {
    TOctagon<int> octagon1(2, 2, 2, 5, 4, 7, 7, 7, 9, 5, 9, 2, 7, -1, 4, -1);
    EXPECT_EQ(area(octagon1), 46);
    TOctagon<double> octagon2(2, 2, 2, 5, 4, 7, 7, 7, 9, 5, 9, 2, 7, -1, 4, -1);
    EXPECT_DOUBLE_EQ(area(octagon2), 46);
}

TEST(TemplateAreaTest, TestingAreaSquare) {
    TSquare<int> square1(2, 0, 0, -2, -2, 0, 0, 2);
    EXPECT_EQ(area(square1), 8);
```

```
    TSquare<double> square2(2, 0, 0, -2, -2, 0, 0, 2);
    EXPECT_DOUBLE_EQ(area(square2), 8);
}

TEST(TemplateCenterTest, TestingCenterTriangle) {
    TTriangle<int> triangle1(2, 2, 3, 0, -2, 1);
    std::pair pair1 = center(triangle1);
    EXPECT_EQ(pair1.first, 1);
    EXPECT_EQ(pair1.second, 1);
    TTriangle<double> triangle2(1, 1, 2, -1, -2, -1);
    std::pair pair2 = center(triangle2);
    EXPECT_DOUBLE_EQ(pair2.first, 0.33333333333333331);
    EXPECT_DOUBLE_EQ(pair2.second, -0.33333333333333331);
}

TEST(TemplateCenterTest, TestingCenterOctagon) {
    TOctagon<int> octagon1(0, 0,0, 2,2, 3,4, 3,6, 2,6, 0,4, -1,2, -1);
    std::pair pair1 = center(octagon1);
    EXPECT_EQ(pair1.first,3);
    EXPECT_EQ(pair1.second,1);
    TOctagon<double> octagon2(2, 2,
    2, 5,
    4, 7,
    7, 7,
    9, 5,
    9, 2,
    7, -1,
    4, -1);
    std::pair pair2 = center(octagon2);
    EXPECT_DOUBLE_EQ(pair2.first,5.5);
    EXPECT_DOUBLE_EQ(pair2.second,3.25);
}

TEST(TemplateCenterTest, TestingCenterSquare) {
    TSquare<int> square1(0, 0, 0, 2, 2, 2, 2, 0);
    std::pair pair1 = center(square1);
    EXPECT_EQ(pair1.first,1);
    EXPECT_EQ(pair1.second,1);
    TSquare<double> square2(0, 0, 0, 2, 2, 2, 2, 0);
    std::pair pair2 = center(square2);
    EXPECT_DOUBLE_EQ(pair2.first,1);
    EXPECT_DOUBLE_EQ(pair2.second,1);
}

TEST(TemplateAreaTestTuple, TestingAreaTuple) {
```

```
    using vertex1 = std::pair<int, int>;
    using vertex2 = std::pair<double, double>;
    using std::make_pair;
    using std::get;

    std::tuple<vertex1, vertex1, vertex1, vertex1> tSquareI1(make_pair(2, 0),
make_pair(-2, 0), make_pair(0, 2), make_pair(-2, 0));
    EXPECT_EQ(area(tSquareI1), 8);
    std::tuple<vertex1, vertex1, vertex1, vertex1> tSquareI2(make_pair(0, 0),
make_pair(0, 1), make_pair(1, 1), make_pair(1, 0));
    EXPECT_DOUBLE_EQ(area(tSquareI2), 1.0);
    std::tuple<vertex2, vertex2, vertex2, vertex2> tSquareD(make_pair(2.5, 1),
make_pair(-1.25, -0.25), make_pair(-2.5, 3.5), make_pair(1.25, 4.75));
    EXPECT_DOUBLE_EQ(area(tSquareD), 15.625);

    std::tuple<vertex1, vertex1, vertex1> tTriangleI1(make_pair(0, 1), make_pair(2, -
1) , make_pair(-2,-1));
    EXPECT_EQ(area(tTriangleI1), 4);
    std::tuple<vertex1, vertex1, vertex1> tTriangleI2(make_pair(0, 0), make_pair(0, 1),
make_pair(5,0));
    EXPECT_DOUBLE_EQ(area(tTriangleI2), 2.5);
    std::tuple<vertex2, vertex2, vertex2> tTriangleD(make_pair(2.5, 1), make_pair(-
2.5, 3.5), make_pair(3.7, 8.9));
    EXPECT_DOUBLE_EQ(area(tTriangleD), 21.25);

    std::tuple<vertex1, vertex1, vertex1, vertex1, vertex1, vertex1, vertex1, vertex1>
tOctagonI1(make_pair(2, 2), make_pair(2, 5) , make_pair(4,7), make_pair(7, 7),
make_pair(9, 5) , make_pair(9,2), make_pair(7, -1), make_pair(4, -1));
    EXPECT_EQ(area(tOctagonI1), 46);
    std::tuple<vertex1, vertex1, vertex1, vertex1, vertex1, vertex1, vertex1, vertex1>
tOctagonI2(make_pair(1, 2), make_pair(4, 1) , make_pair(4,-1), make_pair(1, -3),
make_pair(-1, -3) , make_pair(-3,-1), make_pair(-3, 1), make_pair(-1, 2));
    EXPECT_DOUBLE_EQ(area(tOctagonI2), 27.5);
    std::tuple<vertex2, vertex2, vertex2, vertex2, vertex2, vertex2, vertex2, vertex2>
tOctagonD(make_pair(1.5, 2.6), make_pair(4.9, 1.34) , make_pair(4.54,-1.34),
make_pair(1.4534, -3.213), make_pair(-1.065, -3.0932) , make_pair(-3.3434,-1.44),
make_pair(-3.0001, 1.001), make_pair(-1.12, 2.435));
    EXPECT_DOUBLE_EQ(area(tOctagonD), 36.580140829999998);
}
int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

## 5. Объяснение результатов работы программы.

1) При запуске программы вводится одна из 6 возможных команд в виде строки.
2) tri – пользователь вводит координаты треугольника, в стандартный поток вывода выводятся полученные координаты (типа int), координаты центра треугольника и его площадь.
3) trd – пользователь вводит координаты треугольника, в стандартный поток вывода выводятся полученные координаты (типа double), координаты центра треугольника и его площадь.
4) oi – пользователь вводит координаты восьмиугольника, в стандартный поток вывода выводятся полученные координаты (типа int), координаты центра восьмиугольника и его площадь.
5) od – пользователь вводит координаты восьмиугольника, в стандартный поток вывода выводятся полученные координаты (типа double), координаты центра восьмиугольника и его площадь.
6) si – пользователь вводит координаты диагоналей квадрата, в стандартный поток вывода выводятся полученные координаты (типа int), координаты центра квадрата и его площадь.
7) si – пользователь вводит координаты диагоналей квадрата, в стандартный поток вывода выводятся полученные координаты (типа double), координаты центра квадрата и его площадь.

## 6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с метапрограммированием в C++ и реализовал общие методы для различных классов фигур с различными типами значения, изучив и применив такой механизм языка, как шаблоны. Кроме того, ознакомился с системой тестирования googletests.