Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «ООП»

Тема:
Основы метапрограммирования.

Студент:	Вахрамян К.О.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	3
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

point.h

```
#ifndef VERTEX_H
#define VERTEX_H 1

#include <iostream>
#include <algorithm>
#include <cmath>
#include <cassert>

template<class T>
struct TPoint {
	TPoint() {}
	TPoint(T a, T b) : x(a), y(b){}
	T x;
	T y;


};

template<class T>
std::ostream& operator << (std::ostream& os, const TPoint<T>& p)
{
	os << p.x << " " << p.y << " ";
	return os;
}

template <class T>
std::istream& operator >> (std::istream& is, TPoint<T>& p)
{
	is >> p.x >> p.y;
	return is;
}

template <class T>
TPoint<T> operator /= ( TPoint<T>& p, int val)
{
	p.x = p.x / val;
	p.y = p.y / val;
	return p;
}

template <class T>
```

```
TPoint<T> operator + (const TPoint<T>& p1, const TPoint<T>& p2)
{
    TPoint<T> p;
    p.x = p1.x + p2.x;
    p.y = p1.y + p2.y;
    return p;
}

template <class T>
TPoint<T> operator - (const TPoint<T> p1, const TPoint<T> p2)
{
    TPoint<T> p;
    p.x = p1.x - p2.x;
    p.y = p1.y - p2.y;
    return p;
}

#endif
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H 1

#include "point.h"


template <class T>
struct TRectangle {
    TPoint<T> a, b, c, d;
    TRectangle(std::istream&);
    double Square() const;
    TPoint<T> Center() const;
    void Print() const;
};

template <class T>
TRectangle<T>::TRectangle(std::istream& is) {
    is >> a >> b >> c >> d;
    TPoint<T> ab, ad, cb, cd;
    ab.x = b.x - a.x;
    ab.y = b.y - a.y;
    ad.x = d.x - a.x;
    ad.y = d.y - a.y;
    cb.x = b.x - c.x;
    cb.y = b.y - c.y;
    cd.x = d.x - c.x;
    cd.y = d.y - c.y;

    assert(acos((ab.x * ad.x + ab.y * ad.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(ad.x * ad.x + ad.y * ad.y))) / M_PI == 0.5 && acos((cb.x * cd.x + cb.y * cd.y) / (sqrt(cb.x * cb.x + cb.y * cb.y) * sqrt(cd.x * cd.x + cd.y * cd.y))) / M_PI == 0.5);

}

template <class T>
double TRectangle<T>::Square() const {
    double ans = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
    return fabs(ans);
}


template <class T>
TPoint<T> TRectangle<T>::Center() const {
    TPoint<T> p;
    T x = (a.x + b.x + c.x + d.x) / 4;
    T y = (a.y + b.y + c.y + d.y) / 4;
    p.x = x;
    p.y = y;
    return p;
}

template <class T>
void TRectangle<T>::Print() const {
    std::cout << a << " " << b  << " " << c  << " " << d;
}

#endif
```

trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H 1


#include "point.h"

template <class T>
struct TTrapezoid {
    TPoint<T> a, b, c, d;
```

```
	TTrapezoid(std::istream&);
	double Square() const;
	TPoint<T> Center() const;
	void Print() const;
};

template <class T>
TTrapezoid<T>::TTrapezoid(std::istream& is) {
	is >> a >> b >> c >> d;
	TPoint<T> ab, ad, bc, dc;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	dc.x = c.x - d.x;
	dc.y = c.y - d.y;
	assert(acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x + dc.y * dc.y))) == 0 || acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) * sqrt(bc.x * bc.x + bc.y * bc.y))) == 0);

}

template <class T>
double TTrapezoid<T>::Square() const {
	TPoint<T> p = this->Center();
	T t1 = 0.5 * fabs((b.x - a.x) * (p.y - a.y) - (p.x - a.x) * (b.y - a.y));
	T t2 = 0.5 * fabs((c.x - b.x) * (p.y - b.y) - (p.x - b.x) * (c.y - b.y));
	T t3 = 0.5 * fabs((d.x - c.x) * (p.y - c.y) - (p.x - c.x) * (d.y - c.y));
	T t4 = 0.5 * fabs((a.x - d.x) * (p.y - d.y) - (p.x - d.x) * (a.y - d.y));
	return t1 + t2 + t3 + t4;
}

template <class T>
TPoint<T> TTrapezoid<T>::Center() const {
	TPoint<T> p;
	T x = (a.x + b.x + c.x + d.x) /4;
	T y = (a.y + b.y + c.y + d.y) /4;
	p.x = x;
	p.y = y;

	return p;
}
```

```
template <class T>
void TTrapezoid<T>::Print() const {
	std::cout << a << " " << b << " " << c << " " << d;
}




#endif
```

rhombus.h

```
#ifndef RHOMBUS_H
#define RHOMBUS_H 1
#include "point.h"


template <class T>
struct TRhombus {
	TPoint<T> a, b, c, d;
	TRhombus(std::istream&);
	double Square() const;
	TPoint<T> Center() const;
	void Print() const;
};

template <class T>
TRhombus<T>::TRhombus(std::istream& is) {
	is >> a >> b >> c >> d;
	TPoint<T> ab, bc, cd, da;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	cd.x = d.x - c.x;
	cd.y = d.y - c.y;
	da.x = a.x - d.x;
	da.y = a.y - d.y;
	assert(sqrt(ab.x * ab.x + ab.y * ab.y) ==  sqrt(bc.x * bc.x + bc.y * bc.y) && sqrt(bc.x * bc.x + bc.y * bc.y) == sqrt(cd.x * cd.x + cd.y * cd.y) && sqrt(cd.x * cd.x + cd.y * cd.y) == sqrt(da.x * da.x + da.y * da.y));

}
```

```
template <class T>
double TRhombus<T>::Square() const{
    double ans = 0.5 * sqrt(pow(a.x - c.x, 2) + pow(a.y - c.y, 2)) * sqrt(pow(b.x - d.x,
2) + pow(b.y - d.y, 2));
    return fabs(ans);
}

template <class T>
TPoint<T> TRhombus<T>::Center() const{
    TPoint<T> p;
    T x = (a.x + b.x + c.x + d.x) / 4;
    T y = (a.y + b.y + c.y + d.y) / 4;
    p.x = x;
    p.y = y;
    return p;
}

template <class T>
void TRhombus<T>::Print() const{
    std::cout << a << " " << b << " " << c << " " << d;
}

#endif
```

template.h

```
#ifndef TEMPLATES_H
#define TEMPLATES_H 1

#include <type_traits>
#include <tuple>
#include "point.h"
#include "rhombus.h"
#include "trapezoid.h"
#include "rectangle.h"



template <class T>
struct is_point : std::false_type {};

template <class T>
struct is_point<TPoint <T>> : std::true_type {};

template <class T>
```

```
struct is_figure_like_tuple : std::false_type {};

template <class Head, class ... Tail>
struct is_figure_like_tuple <std::tuple<Head, Tail ... >> :
      std::conjunction <is_point<Head>, std::is_same<Head, Tail> ...> {};

template <class Type, size_t size>
struct is_figure_like_tuple <std::array<Type, size>> : is_point<Type> {};

template <class T>
inline constexpr bool is_figure_like_tuple_v = is_figure_like_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 ;
    }
    return ;
}
```

```
template <class T>
std::enable_if_t <is_figure_like_tuple_v<T>, void>
      print(const T& fake) {
            return recursive_print<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>, TPoint<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>
TPoint<double> single_center(const T& t) {
    TPoint<double> p;
    p = std::get<ID>(t);
    p /= tuple_size_v<T>;
    return p;
}


template<size_t ID, class T>
TPoint<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{
        TPoint<double> p(0, 0);

        return p;
    }
}

template<class T>
std::enable_if_t<is_figure_like_tuple_v<T>, TPoint<double>>
center(const T& fake) {
    return recursive_center<0>(fake);
}


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_center_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_figure_like_tuple_v<T>, double>
Square(const T& fake) {
    return recursive_square<2>(fake);
}




#endif
```

main.cpp

```
#include "rectangle.h"
#include "point.h"
#include "trapezoid.h"
#include "rhombus.h"
#include "template.h"
#include <string>


int main()
{

    std::string cmd;
    std::cout << "enter:\nrectangle - to add rectangle and calculate square and center
of rectangle;\n"
               << "rhombus - to add rhombus and calculate square and center of
rhombus;\n"
               << "trapezoid - to add trapezoid and calculate square and center of
trapezoid;\n"
              << "help - to show this manual;\n"
              << "exit - to finish execution of program.\n";
    while (true) {
        std::cin >> cmd;
        if (cmd == "rectangle") {
            TRectangle<double> real_rectangle(std::cin);
            std::tuple<TPoint<double>, TPoint<double>, TPoint<double>,
TPoint<double>>
```

```
        tuple_rectangle{real_rectangle.a, real_rectangle.b, real_rectangle.c,
real_rectangle.d};
        std::cout << "\nreal_rectangle\n";
        std::cout << "coordinates: ";
        print(real_rectangle);
        std::cout << "\nsquare: " << Square(real_rectangle);
        std::cout << "\ncenter: " << center(real_rectangle);
        std::cout << "\ntuple_rectangle\n";
        std::cout << "coordinates: ";
        print(tuple_rectangle);
        std::cout << "\nsquare: " << Square(tuple_rectangle);
        std::cout << "\ncenter: " << center(tuple_rectangle) << "\n";


    } else if (cmd == "rhombus") {
        TRhombus<double> real_rhombus(std::cin);
        std::tuple<TPoint<double>, TPoint<double>, TPoint<double>,
TPoint<double>>
        tuple_rhombus{real_rhombus.a, real_rhombus.b, real_rhombus.c,
real_rhombus.d};
        std::cout << "\nreal_rhombus\n";
        std::cout << "coordinates: ";
        print(real_rhombus);
        std::cout << "\nsquare: " << Square(real_rhombus);
        std::cout << "\ncenter: " << center(real_rhombus);
        std::cout << "\ntuple_rhombus\n";
        std::cout << "coordinates: ";
        print(tuple_rhombus);
        std::cout << "\nsquare: " << Square(tuple_rhombus);
        std::cout << "\ncenter: " << center(tuple_rhombus) << "\n";



    } else if (cmd == "trapezoid") {
        TTrapezoid<double> real_trapezoid(std::cin);
        std::tuple<TPoint<double>, TPoint<double>, TPoint<double>,
TPoint<double>>
        tuple_trapezoid{real_trapezoid.a, real_trapezoid.b, real_trapezoid.c,
real_trapezoid.d};
        std::cout << "\nreal_trapezoid\n";
        std::cout << "coordinates: ";
        print(real_trapezoid);
        std::cout << "\nsquare: " << Square(real_trapezoid);
        std::cout << "\ncenter: " << center(real_trapezoid);
        std::cout << "\ntuple_trapezoid\n";
```

```
            std::cout << "coordinates: ";
            print(tuple_trapezoid);
            std::cout << "\nsquare: " << Square(tuple_trapezoid);
            std::cout << "\ncenter: " << center(tuple_trapezoid) << "\n";
        } else if (cmd == "help") {
            std::cout << "\nenter:\nrectangle - to add rectangle and calculate square and center of rectangle;\n"
              << "rhombus - to add rhombus and calculate square and center of rhombus;\n"
              << "trapezoid - to add trapezoid and calculate square and center of trapezoid;\n"
             << "help - to show this manual;\n"
             << "exit - to finish execution of program.\n";
            } else if (cmd == "exit"){
                break;
            } else {
                std::cout << "wrong comand, try again\n";
                continue;
            }


    }

}
```

2. Ссылка на репозиторий на GitHub

https://github.com/vebcreatex7/oop_exercise_04

3. Набор googletests.

```
courage@courage-X550LC:~/oop/oop_exercise_04/cmake-build-debug$ ./oop_exercise_04
enter:
rectangle - to add rectangle and calculate square and center of rectangle;
rhombus - to add rhombus and calculate square and center of rhombus;
trapezoid - to add trapezoid and calculate square and center of trapezoid;
help - to show this manual;
exit - to finish execution of program.
rectangle
0 0 0 3 4 3 4 0

real_rectangle
coordinates: 0 0  0 3  4 3  4 0
square: 12
center: 2 1.5
tuple_rectangle
```

```
coordinates: 0 0 0 3 4 3 4 0
square: 12
center: 2 1.5
rhombus
3 0 0 6 3 12 6 6

real_rhombus
coordinates: 3 0  0 6  3 12  6 6
square: 36
center: 3 6
tuple_rhombus
coordinates: 3 0 0 6 3 12 6 6
square: 36
center: 3 6
help

enter:
rectangle - to add rectangle and calculate square and center of rectangle;
rhombus - to add rhombus and calculate square and center of rhombus;
trapezoid - to add trapezoid and calculate square and center of trapezoid;
help - to show this manual;
exit - to finish execution of program.
trapezoid
0 0 2 3 5 3 6 0

real_trapezoid
coordinates: 0 0  2 3  5 3  6 0
square: 13.5
center: 3.25 1.5
tuple_trapezoid
coordinates: 0 0 2 3 5 3 6 0
square: 13.5
center: 3.25 1.5
exit
courage@courage-X550LC:~/oop/oop_exercise_04/cmake-build-debug$
```

5. Объяснение результатов работы программы.

При запуске программы появляется контекстное меню, которое предлагает варианты реализованных фигур. При выборе одной из трех фигур необходимо ввести ее координаты, если фигура оказывается неправильной, то происходит остановка программы при помощи assert() в конструкторе класса структуры. Если координаты введены верно, создается объект класса и высчитывается его площадь и центр, также создается объект tuple, представляющий из себя ту же фигуру, методы которой реализованы при помощи рекурсии шаблонов.

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.