

Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 4: «Основы метапрограммирования»

Группа:	М8О-208Б-18, №19
Студент:	Овечкин Виталий Андреевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	09.01.2020

Москва, 2019

1. Тема: Основы метапрограммирования

2. Цель работы: Изучение основ работы с шаблонами в C++

3. Задание (вариант № 19):

Разработать шаблоны классов согласно варианту задания.

Параметром шаблона должен являться скалярный тип данных, задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения.

Создать набор шаблонов, реализующий функции:

- Вычисление геометрического центра фигуры
- Вывод в стандартный поток `std::cout` координат вершин фигуры
- Вычисление площади фигуры

Параметром шаблона должен являться тип класса фигуры. Помимо самого класса фигуры, шаблонная функция должна уметь работать с `tuple`.

Фигуры (Вариант 3) :

Прямоугольник, трапеция, ромб.

4. Адрес репозитория на GitHub https://github.com/vitalouivi/oop_exercise_04

5. Код программы на C++

main.cpp

```
#include
<iostream>

#include <tuple>
#include "point.h"
#include "trapezoid.h"
#include "rectangle.h"
#include "rhombus.h"
#include "templates.h"
template<class T>
void running(std::istream& is, std::ostream& os) {
    if constexpr (is_figurelike_tuple<T>::value) {
        int ang;
        std::cout << "Input number of angles" << std::endl;
        std::cin >> ang;
        if (ang == 4) {
            point<double> A, B, C, D;
            is >> A >> B >> C >> D;
            auto object = std::make_tuple(A, B, C, D);
            print(os, object);
            os << area(object) << std::endl;
            os << center(object) << std::endl;
        } else if (ang == 3) {
            point<double> A, B, C;
            is >> A >> B >> C;
            auto object = std::make_tuple(A, B, C);
            print(os, object);
            os << area(object) << std::endl;
            os << center(object) << std::endl;
        }
    }
}
```

```

    }else {
        T object(is);
        print(os, object);
        os << '\n' << area(object) << std::endl;
        os << center(object) << std::endl;
    }
}

int main() {
    char obj_type;
    std::cout << "Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4
- tuple or 'q' to quit" << std::endl;
    while (std::cin >> obj_type){
        if(obj_type == '4') {
            running<std::tuple<point<double>>> (std::cin, std::cout);
        }else if(obj_type == '1'){
            running<trapezoid<double>>(std::cin, std::cout);
        }else if(obj_type == '2'){
            running<rhombus<double>>(std::cin, std::cout);
        }else if(obj_type == '3'){
            running<rectangle<double>>(std::cin, std::cout);
        }else if(obj_type == 'q'){
            return 0;
        }else{
            std::cout << "Wrong. Try 1 - trapezoid, 2 - rhombus, 3 -
rectangle, 4 - tuple or 'q' to quit" << std::endl;
        }
    }
}

```

point.h

```

#ifndef
POINT_H_

#define POINT_H_
#include <iostream>
template<class T>
struct point {
    T x;
    T y;
};
template<class T>
point<T> operator+(const point<T>& A, const point<T>& B) {
    point<T> res;
    res.x = A.x + B.x;
    res.y = A.y + B.y;
    return res;
}
template<class T>
point<T> operator/=(point<T>& A, const double B) {

```

```

        A.x /= B;
        A.y /= B;
        return A;
    }
    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;
    }
}
#endif

```

rhombus.h

```

#ifndef
RHOMBUS_H_

#define RHOMBUS_H_
#include <iostream>
#include <cmath>
#include "point.h"
template<class T>
struct rhombus {
    point<T> points[4];
    rhombus(std::istream& is);
    double area() const;
    point<T> center() const;
    void print(std::ostream& os) const;
};
template<class T>
rhombus<T>::rhombus(std::istream& is) {
    for(int i = 0; i < 4; ++i){
        is >> points[i];
    }
    double a, b, c, d;
    a = sqrt((points[1].x - points[0].x) * (points[1].x - points[0].x) +
(points[1].y - points[0].y) * (points[1].y - points[0].y));
    b = sqrt((points[2].x - points[1].x) * (points[2].x - points[1].x) +
(points[2].y - points[1].y) * (points[2].y - points[1].y));
    c = sqrt((points[2].x - points[3].x) * (points[2].x - points[3].x) +
(points[2].y - points[3].y) * (points[2].y - points[3].y));
    d = sqrt((points[3].x - points[0].x) * (points[3].x - points[0].x) +
(points[3].y - points[0].y) * (points[3].y - points[0].y));
    if(a != b || a != c || a != d)
        throw std::logic_error("It`s not a rhombus");
}

```

```

template<class T>
double rhombus<T>::area() const {
    const T d1 = sqrt((points[0].x - points[2].x) * (points[0].x - points[2].x) +
(points[0].y - points[2].y) * (points[0].y - points[2].y));
    const T d2 = sqrt((points[1].x - points[3].x) * (points[1].x - points[3].x) +
(points[1].y - points[3].y) * (points[1].y - points[3].y));
    return d1 * d2 / 2;
}
template<class T>
point<T> rhombus<T>::center() const {
    point<T> res;
    res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4;
    res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4;
    return res;
}
template<class T>
void rhombus<T>::print(std::ostream& os) const {
    for(int i = 0; i < 4; ++i){
        os << points[i];
        if(i + 1 != 4){
            os << ' ';
        }
    }
}
#endif

```

rectangle.h

```

#ifndef
RECTANGLE_H_

#define RECTANGLE_H_
#include <iostream>
#include "point.h"
#include <cmath>
template<class T>
struct rectangle {
    point<T> points[4];
    rectangle(std::istream& is);
    double area() const;
    point<T> center() const;
    void print(std::ostream& os) const;
};
template<class T>
rectangle<T>::rectangle(std::istream& is) {
    for(int i = 0; i < 4; ++i){
        is >> points[i];
    }
    double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
    a = sqrt((points[1].x - points[0].x) * (points[1].x - points[0].x) +

```

```

(points[1].y - points[0].y) * (points[1].y - points[0].y));
    b = sqrt((points[2].x - points[1].x) * (points[2].x - points[1].x) +
(points[2].y - points[1].y) * (points[2].y - points[1].y));
    c = sqrt((points[2].x - points[3].x) * (points[2].x - points[3].x) +
(points[2].y - points[3].y) * (points[2].y - points[3].y));
    d = sqrt((points[3].x - points[0].x) * (points[3].x - points[0].x) +
(points[3].y - points[0].y) * (points[3].y - points[0].y));
    d1 = sqrt((points[1].x - points[3].x) * (points[1].x - points[3].x) +
(points[1].y - points[3].y) * (points[1].y - points[3].y));
    d2 = sqrt((points[2].x - points[0].x) * (points[2].x - points[0].x) +
(points[2].y - points[0].y) * (points[2].y - points[0].y));
    ABC = (a * a + b * b - d2 * d2) / 2 * a * b;
    BCD = (b * b + c * c - d1 * d1) / 2 * b * c;
    CDA = (d * d + c * c - d2 * d2) / 2 * d * c;
    DAB = (a * a + d * d - d1 * d1) / 2 * a * d;
    if(ABC != BCD || ABC != CDA || ABC != DAB)
        throw std::logic_error("It`s not a rectangle");
}
template<class T>
double rectangle<T>::area() const {
    const T a = sqrt((points[1].x - points[0].x) * (points[1].x - points[0].x) +
(points[1].y - points[0].y) * (points[1].y - points[0].y));
    const T b = sqrt((points[2].x - points[1].x) * (points[2].x - points[1].x) +
(points[2].y - points[1].y) * (points[2].y - points[1].y));
    return a * b;
}
template<class T>
point<T> rectangle<T>::center() const {
    point<T> res;
    res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4;
    res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4;
    return res;
}
template<class T>
void rectangle<T>::print(std::ostream& os) const {
    for(int i = 0; i < 4; ++i){
        os << points[i];
        if(i + 1 != 4){
            os << ' ';
        }
    }
}
#endif

```

trapezoid.h

#ifndef

TRAPEZOID_H_

#define TRAPEZOID_H_

```

#include <iostream>
#include <cmath>
#include "point.h"
template<class T>
struct trapezoid {
    point<T> points[4];
    trapezoid(std::istream& is);
    double area() const;
    point<T> center() const;
    void print(std::ostream& os) const;
};

template<class T>
trapezoid<T>::trapezoid(std::istream& is) {
    for(int i = 0; i < 4; ++i){
        is >> points[i];
    }
    if((points[2].y - points[1].y) / (points[2].x - points[1].x) != (points[3].y
- points[0].y) / (points[3].x - points[0].x))
        throw std::logic_error("It`s not a trapezoid");
}

template<class T>
double trapezoid<T>::area() const {

    return 0.5 * std::abs( points[0].x * points[1].y + points[1].x * points[2].y
+ points[2].x * points[3].y + points[3].x * points[0].y - points[1].x *
points[0].y - points[2].x * points[1].y - points[3].x * points[2].y -
points[0].x * points[3].y);
}

template<class T>
point<T> trapezoid<T>::center() const {
    point<T> res;
    res.x = (points[0].x + points[1].x + points[2].x + points[3].x) / 4;
    res.y = (points[0].y + points[1].y + points[2].y + points[3].y) / 4;
    return res;
}

template<class T>
void trapezoid<T>::print(std::ostream& os) const {
    for(int i = 0; i < 4; ++i){
        os << points[i];
        if(i + 1 != 4){
            os << ' ';
        }
    }
}

#endif

```

templates.h

```
#ifndef
```

TEMPLATES_H_

```
#define TEMPLATES_H_
#include <tuple>
#include <type_traits>
#include "point.h"
template<class T>
struct is_point : std::false_type {};
template<class T>
struct is_point<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_point<Head>, std::is_same<Head, Tail>...> {};
template<class T>
inline constexpr bool is_figurelike_tuple_v = is_figurelike_tuple<T>::value;
template<class T, class = void>
struct has_method_area : std::false_type {};
template<class T>
struct has_method_area<T, std::void_t<decltype(std::declval<const
T&>().area())>> : std::true_type {};
template<class T>
inline constexpr bool has_method_area_v = has_method_area<T>::value;
template<class T>
std::enable_if_t<has_method_area_v<T>, double> area(const T& object) {
    return object.area();
}
template<class T, class = void>
struct has_method_center : std::false_type {};
template<class T>
struct has_method_center<T, std::void_t<decltype(std::declval<const
T&>().center())>> : std::true_type {};
template<class T>
inline constexpr bool has_method_center_v = has_method_center<T>::value;
template<class T>
std::enable_if_t<has_method_center_v<T>, point<double>> center(const T& object)
{
    return object.center();
}
template<class T, class = void>
struct has_method_print : std::false_type {};
template<class T>
struct has_method_print<T, std::void_t<decltype(std::declval<const
T&>().print(std::cout))>> : std::true_type {};
template<class T>
inline constexpr bool has_method_print_v = has_method_print<T>::value;
template<class T>
std::enable_if_t<has_method_print_v<T>, void> print(std::ostream& os, const T&
```



```

object) {
    object.print(os);
}
template<size_t Id, class T>
double compute_area(const T& tuple) {
    if constexpr (Id >= std::tuple_size_v<T>){
        return 0;
    }else{
        const auto x1 = std::get<Id - 0>(tuple).x - std::get<0>(tuple).x;
        const auto y1 = std::get<Id - 0>(tuple).y - std::get<0>(tuple).y;
        const auto x2 = std::get<Id - 1>(tuple).x - std::get<0>(tuple).x;
        const auto y2 = std::get<Id - 1>(tuple).y - std::get<0>(tuple).y;
        const double local_area = std::abs(x1 * y2 - y1 * x2) * 0.5;
        return local_area + compute_area<Id + 1>(tuple);
    }
}
template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, double>
area(const T& object) {
    if constexpr (std::tuple_size_v<T> < 3){
        throw std::logic_error("It`s not a figure");
    }else{
        return compute_area<2>(object);
    }
}
template<size_t Id, class T>
point<double> tuple_center(const T& object) {
    if constexpr (Id >= std::tuple_size<T>::value) {
        return point<double> {0, 0};
    } else {
        point<double> res = std::get<Id>(object);
        return res + tuple_center<Id+1>(object);
    }
}
template<class T>
point<double> compute_center(const T &tuple) {
    point<double> res{0, 0};
    res = tuple_center<0>(tuple);
    res /= std::tuple_size_v<T>;
    return res;
}
template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, point<double>>
center(const T& object) {
    if constexpr (std::tuple_size_v<T> < 3){
        throw std::logic_error("It`s not a figure");
    }else{
        return compute_center(object);
    }
}

```

```

    }
}
template<size_t Id, class T>
void step_print(const T& object, std::ostream& os) {
    if constexpr (Id >= std::tuple_size<T>::value) {
        std::cout << "\n";
    } else {
        os << std::get<Id>(object) << " ";
        step_print<Id + 1>(object, os);
    }
}
template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, void>
print(std::ostream& os, const T& object) {
    if constexpr (std::tuple_size_v<T> < 3){
        throw std::logic_error("It`s not a figure");
    }else{
        step_print<0>(object, os);
    }
}
#endif

```

CMakeLists.txt

```

cmake_minimum_required (VERSION
3.5)

```

```

project(lab4)
add_executable(oop_exercise_04
    main.cpp
)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -
Wall -Wextra -std=c++1z -g3")
set_target_properties(oop_exercise_04
    PROPERTIES CXX_STANDARD 14
    CXX_STANDARD_REQUIRED ON)

```

6. Ha6op testcases

test_01.txt

```

1
0 0 1 1 2 1 3 0
2
0 0 -1 2 0 4 1 2
3
0 0 0 1 4 1 4 0
q

```

test_02.txt

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

7. Результаты выполнения тестов

Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 - tuple or 'q' to quit

```
1
0 0 1 1 2 1 3 0
[0 0] [1 1] [2 1] [3 0]
2
[1.5 0.5]
2
0 0 -1 2 0 4 1 2
[0 0] [-1 2] [0 4] [1 2]
4
[0 2]
3
0 0 0 1 4 1 4 0
[0 0] [0 1] [4 1] [4 0]
4
[2 0.5]
q
```

C:\lab4\lab4\Debug\lab4.exe (процесс 17196) завершил работу с кодом 0.

Input figure type: 1 - trapezoid, 2 - rhombus, 3 - rectangle, 4 - tuple or 'q' to quit

```
4
Input number of angles
3
0 0 1 1 2 0
[0 0] [1 1] [2 0]
1
[1 0.333333]
4
Input number of angles
4
0 0 0 2 2 2 2 0
[0 0] [0 2] [2 2] [2 0]
4
[1 1]
q
```

C:\lab4\lab4\Debug\lab4.exe (процесс 13468) завершил работу с кодом 0.

8. Объяснение результатов работы программы - вывод

В файлах `rectangle.h`, `trapezoid.h` и `rhombus.h` описаны фигуры. В `templates.h` описаны шаблоны для работы с этими фигурами и `tuple`.

В ходе выполнения данной лабораторной работы были получены навыки работы с шаблонами, а также хэдером `<type_traits>`, создания шаблонных классов.