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:	Косогоров В.В.
Группа:	М8О-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	10
Оценка:	
Дата:	

Москва
2019

1. Код на С++:

vertex.h:

```
#ifndef D_VERTEX_H_
#define D_VERTEX_H_

#include <iostream>

template<class T>
struct vertex {
    T x, y;
};

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

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

template<class T>
vertex<T> operator+(vertex<T> lhs,vertex<T> rhs){
    vertex<T> res;
    res.x = lhs.x + rhs.x;
    res.y = lhs.y + rhs.y;
    return res;
}

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

#endif
```

templates.h:

```
#ifndef D_TEMPLATES_H_
#define D_TEMPLATES_H_ 1

#include <tuple>
#include <type_traits>

#include "square.h"
```

```
#include "rectangle.h"
#include "trapezoid.h"
#include "vertex.h"

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

template<class T>
struct is_vertex<vertex<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>, vertex<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>
vertex<double> single_center(const T& t) {
    vertex<double> v;
    v.x = std::get<ID>(t).x;
    v.y = std::get<ID>(t).y;
    v /= std::tuple_size_v<T>;
    return v;
}

template<size_t ID, class T>
vertex<double> RecursiveCenter(const T& t) {
    if constexpr (ID < std::tuple_size_v<T>){
        return  single_center<ID>(t) + RecursiveCenter<ID+1>(t);
    } else {
        vertex<double> v;
        v.x = 0;
        v.y = 0;
        return v;
    }
}

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, vertex<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();
}

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 RecursiveArea(const T& t) {
    if constexpr (ID < std::tuple_size_v<T>){
        return single_area<ID>(t) + RecursiveArea<ID + 1>(t);
    }
    return 0;
}

template<class T>
std::enable_if_t<is_figurelike_tuple_v<T>, double>
Area(const T& fake) {
    return RecursiveArea<2>(fake);
```

```
}

#endif // D_TEMPLATES_H_
```

trapezoid.h

```
#ifndef D_TRAPEZOID_H_
#define D_TRAPEZOID_H_

#include <iostream>
#include <assert.h>
#include "vertex.h"

template<class T>
struct Trapezoid {
    Trapezoid(std::istream &is);
    int IsCorrect() const;

    vertex<double> Center() const;
    void Print() const;
    double Area() const;

private:
    vertex<T> one,two,three,four;
};

template<class T>
Trapezoid<T>::Trapezoid(std::istream &is){
    is >> one >> two >> three >> four;
    assert(IsCorrect());
}

template<class T>
int Trapezoid<T>::IsCorrect() const {
    T vec1_x = four.x - one.x;
    T vec1_y = four.y - one.y;

    T vec2_x = three.x - two.x;
    T vec2_y = three.y - two.y;

    T vec3_x = two.x - one.x;
    T vec3_y = two.y - one.y;

    T vec4_x = three.x - four.x;
    T vec4_y = three.y - four.y;

    if ((vec1_x / vec2_x == vec1_y / vec2_y) || (vec3_x / vec4_x == vec3_y / vec4_y) ||
//отношение соответствующих координат
            (vec1_x == 0 && vec2_x == 0) || (vec1_y == 0 && vec2_y == 0) || (vec3_x == 0 &&
vec4_x == 0) || (vec3_y == 0 && vec4_y == 0)) {
        return 1;
    }
    return 0;
}

template<class T>
vertex<double> Trapezoid<T>::Center() const {
    vertex<double> center;

    center = one + two + three + four;
    center /= 4;

    return center;
}

template<class T>
void Trapezoid<T>::Print() const {
    std::cout << one << " " << two << " " << three << " " << four << '\n';
}

template<class T>
double Trapezoid<T>::Area() const {

    const T area1 = 0.5 * abs((three.x - two.x) * (four.y - two.y) - (four.x - two.x) * (three.y - 
two.y));
    const T area2 = 0.5 * abs((one.x - two.x) * (four.y - two.y) - (four.x - two.x) * (one.y - two.y));

    return area1 + area2;
}

#endif
```

square.h

```
#ifndef D_SQUARE_H_
#define D_SQUARE_H_

#include <iostream>
#include <assert.h>
#include <math.h>

#include "vertex.h"

template<class T>
struct Square {
    Square(std::istream &is);
    int IsCorrect() const;

    vertex<double> Center() const;
    void Print() const;
    double Area() const;

private:
    vertex<T> one,two,three,four;
```

```
};

template<class T>
Square<T>::Square(std::istream &is){
    is >> one >> two >> three >> four;
    assert(IsCorrect());
}

template<class T>
int Square<T>::IsCorrect() const {
    const T vec1_x = two.x - one.x;
    const T vec1_y = two.y - one.y;

    const T vec2_x = three.x - two.x;
    const T vec2_y = three.y - two.y;

    const T vec3_x = four.x - one.x;
    const T vec3_y = four.y - one.y;

    const T vec4_x = four.x - three.x;
    const T vec4_y = four.y - three.y;

    const T dotProduct1 = vec1_x * vec2_x + vec1_y * vec2_y;
    const T dotProduct2 = vec3_x * vec1_x + vec3_y * vec1_y;
    const T dotProduct3 = vec3_x * vec4_x + vec3_y * vec4_y;

    const T vec1_length = sqrt(vec1_x * vec1_x + vec1_y * vec1_y);
    const T vec2_length = sqrt(vec2_x * vec2_x + vec2_y * vec2_y);

    if (dotProduct1 == 0 && dotProduct2 == 0 && dotProduct3 == 0 && vec1_length ==
vec2_length) {
        return 1;
    }
    return 0;
}

template<class T>
vertex<double> Square<T>::Center() const {
    vertex<double> center;
    center.x = (one.x + three.x) / 2;
    center.y = (one.y + three.y) / 2;
    return center;
}

template<class T>
void Square<T>::Print() const {
    std::cout << one << " " << two << " " << three << " " << four << '\n';
}

template<class T>
double Square<T>::Area() const {
    const T vecX = two.x - one.x;
    const T vecY = two.y - one.y;

    return vecX * vecX + vecY * vecY;
}

#endif
```

rectangle.h

```
#ifndef D_RECTANGLE_H_
#define D_RECTANGLE_H_

#include <iostream>
#include <assert.h>
#include <math.h>

#include "vertex.h"

template<class T>
struct Rectangle {
    Rectangle(std::istream &is);
    int IsCorrect() const;

    vertex<double> Center() const;
    void Print() const;
    double Area() const;

private:
    vertex<T> one,two,three,four;
};

template<class T>
Rectangle<T>::Rectangle(std::istream &is){
    is >> one >> two >> three >> four;
    assert(IsCorrect());
}

template<class T>
int Rectangle<T>::IsCorrect() const {
    const T vec1_x = two.x - one.x;
    const T vec1_y = two.y - one.y;

    const T vec2_x = three.x - two.x;
    const T vec2_y = three.y - two.y;

    const T vec3_x = four.x - one.x;
    const T vec3_y = four.y - one.y;

    const T vec4_x = four.x - three.x;
    const T vec4_y = four.y - three.y;

    const T dotProduct1 = vec1_x * vec2_x + vec1_y * vec2_y;
    const T dotProduct2 = vec3_x * vec1_x + vec3_y * vec1_y;
    const T dotProduct3 = vec3_x * vec4_x + vec3_y * vec4_y;

    if (dotProduct1 == 0 && dotProduct2 == 0 && dotProduct3 == 0) {
        return 1;
    }
    return 0;
}

template<class T>
vertex<double> Rectangle<T>::Center() const {
    vertex<double> center;
    center.x = (one.x + three.x) / 2;
    center.y = (one.y + three.y) / 2;
    return center;
}

template<class T>
void Rectangle<T>::Print() const {
    std::cout << one << " " << two << " " << three << " " << four << '\n';
}

template<class T>
double Rectangle<T>::Area() const {
    const T xHeight = two.x - one.x;
    const T yHeight = two.y - one.y;

    const T xWidth = three.x - two.x;
    const T yWidth = three.y - two.y;

    return sqrt(xHeight * xHeight + yHeight * yHeight) * sqrt(xWidth * xWidth + yWidth * 
yWidth);
}

#endif
```

main.cpp

```
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"
#include "templates.h"

int main() {
    int input;

    while (true) {
        std::cout << "++++++++++++++++++++++++++++++++++++++" << std::endl;
        std::cout << "0: Exit" << std::endl;
        std::cout << "1: Fake figure demonstration" << std::endl;
        std::cout << "2: Array figure demonstration" << std::endl;
        std::cout << "3: Real figure demonstration" << std::endl;
```

```
std::cin >> input;

if (input == 0) {
    break;
}

if (input > 3) {
    std::cout << "Invalid input" << std::endl;
}

switch (input) {
    case 1: {
        std::cout << "Fake Square (float):" << std::endl;
        std::tuple<vertex<float>, vertex<float>, vertex<float>, vertex<float>>
                fakeSquare{{0, 0}, {0, 0.5}, {0.5, 0.5}, {0.5, 0}};
        std::cout << "Coordinates: ";
        Print(fakeSquare);
        std::cout << std::endl;
        std::cout << "Center: " << Center(fakeSquare) << std::endl;
        std::cout << "Area: " << Area(fakeSquare) << std::endl << std::endl;

        std::cout << "Fake Rectangle (int):" << std::endl;
        std::tuple<vertex<int>, vertex<int>, vertex<int>, vertex<int>>
                fakeRectangle{{0, 0}, {0, 2}, {10, 2}, {10, 0}};
        std::cout << "Coordinates: ";
        Print(fakeRectangle);
        std::cout << std::endl;
        std::cout << "Center: " << Center(fakeRectangle) << std::endl;
        std::cout << "Area: " << Area(fakeRectangle) << std::endl << std::endl;

        std::cout << "Fake Trapezoid (double):" << std::endl;
        std::tuple<vertex<double>, vertex<double>, vertex<double>, vertex<double>>
                fakeTrapezoid{{0, 0}, {0.5, 2}, {2, 2}, {15.5, 0}};
        std::cout << "Coordinates: ";
        Print(fakeTrapezoid);
        std::cout << std::endl;
        std::cout << "Center: " << Center(fakeTrapezoid) << std::endl;
        std::cout << "Area: " << Area(fakeTrapezoid) << std::endl << std::endl;
    break;
    }

    case 2: {
        std::cout << "Array Square (double):" << std::endl;
        std::array<vertex<double>, 4>
                array_Square{{{0, 0}, {0, 2}, {2, 2}, {2, 0}}};
        std::cout << "Coordinates: ";
        Print(array_Square);
        std::cout << std::endl;
        std::cout << "Center: " << Center(array_Square) << std::endl;
        std::cout << "Area: " << Area(array_Square) << std::endl << std::endl;
        std::cout << "Array Trapezoid (float):" << std::endl;
        std::array<vertex<float>, 4>
                array_Trapezoid{{{0, 0}, {1, 2}, {2, 2}, {3, 0}}};
        std::cout << "Coordinates: ";
        Print(array_Trapezoid);
        std::cout << std::endl;
        std::cout << "Center: " << Center(array_Trapezoid) << std::endl;
        std::cout << "Area: " << Area(array_Trapezoid) << std::endl << std::endl;
    break;
    }

    case 3: {
        int realID;

        std::cout << "Input real figure id:" << std::endl;
        std::cout << "1: Square" << std::endl;
        std::cout << "2: Rectangle" << std::endl;
        std::cout << "3: Trapezoid" << std::endl;

        std::cin >> realID;

        switch (realID) {
            case 1: {
                std::cout << "Input 4 coordinates in a sequence" << std::endl;
                Square<double> realSquare(std::cin);
                std::cout << "Coordinates: ";
                Print(realSquare);
                std::cout << std::endl;
                std::cout << "Center: " << Center(realSquare) << std::endl;
                std::cout << "Area: " << Area(realSquare) << std::endl << std::endl;
            break;
            }

            case 2: {
                std::cout << "Input 4 coordinates in a sequence" << std::endl;
                Rectangle<double> realRectangle(std::cin);
                std::cout << "Coordinates: ";
                Print(realRectangle);
                std::cout << std::endl;
                std::cout << "Center: " << Center(realRectangle) << std::endl;
                std::cout << "Area: " << Area(realRectangle) << std::endl << std::endl;
            break;
            }

            case 3: {
                std::cout << "Input 4 coordinates in a sequence" << std::endl;
                Trapezoid<double> realTrapezoid(std::cin);
                std::cout << "Coordinates: ";
                Print(realTrapezoid);
                std::cout << std::endl;
                std::cout << "Center: " << Center(realTrapezoid) << std::endl;
                std::cout << "Area: " << Area(realTrapezoid) << std::endl << std::endl;
```

```
                    break;
                    }
                }
            break;
            }
        }
    }
    return 0;
}
```

2. Ссылка на репозиторий в GitHub:

https://github.com/vladiq/oop_exercise_04

3. Набор testcases:

test_01.test:

```
1

2

3

1

0 0

1 0

1 1

0 1

3

2

0 0

10 0

10 20

0 20

3

3

0 0

1 2

2 2

3 0

0
```

test_02.test:

```
3
1
1234134 131
1312 321
321 2343
13 321
```

test_03.test:

```
3
1
0 100
100 100
100 0
0 0
3
2
0 0
0.4 0
0.4 1000
0 1000
3
3
0 0
50 50
150 50
100 0
0
```

4.Результаты выполнения программы:

test_01.result

```
++++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Fake Square (float):
Coordinates: (0, 0) (0, 0.5) (0.5, 0.5) (0.5, 0)
Center: (0.25, 0.25)
Area: 0.25

Fake Rectangle (int):
Coordinates: (0, 0) (0, 2) (10, 2) (10, 0)
Center: (5, 1)
Area: 20

Fake Trapezoid (double):
Coordinates: (0, 0) (0.5, 2) (2, 2) (15.5, 0)
Center: (4.5, 1)
Area: 17

++++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Array Square (double):
Coordinates: (0, 0) (0, 2) (2, 2) (2, 0)
Center: (1, 1)
Area: 4

Array Trapezoid (float):
Coordinates: (0, 0) (1, 2) (2, 2) (3, 0)
```

```
Center: (1.5, 1)
Area: 4

+++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Input real figure id:
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
Coordinates: (0, 0)  (1, 0)  (1, 1)  (0, 1)

Center: (0.5, 0.5)
Area: 1

+++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Input real figure id:
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
Coordinates: (0, 0)  (10, 0)  (10, 20)  (0, 20)

Center: (5, 10)
Area: 200

+++++++++++++++++++++++++++++++++++
0: Exit
```

```
1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Input real figure id:

1: Square

2: Rectangle

3: Trapezoid

Input 4 coordinates in a sequence

Coordinates: (0, 0)  (1, 2)  (2, 2)  (3, 0)


Center: (1.5, 1)

Area: 4


++++++++++++++++++++++++++++++++++++

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration
```

test_02.result

```
++++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Input real figure id:
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
oop_exercise_04: /home/vladislav/Рабочий стол/prog_3_sem/oop_labs/lab_04/square.h:26:
Square<T>::Square(std::istream&) [with T = double; std::istream = std::basic_istream<char>]: Assertion
`IsCorrect()' failed.
[1]    25916 abort (core dumped)  ./oop_exercise_04 < test_02.test > test_02.result
```

test_03.result

```
++++++++++++++++++++++++++++++++++++

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration

Input real figure id:
```

```
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
Coordinates: (0, 100)  (100, 100)  (100, 0)  (0, 0)

Center: (50, 50)
Area: 10000

++++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Input real figure id:
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
Coordinates: (0, 0)  (0.4, 0)  (0.4, 1000)  (0, 1000)

Center: (0.2, 500)
Area: 400

++++++++++++++++++++++++++++++++++++
0: Exit
1: Fake figure demonstration
2: Array figure demonstration
3: Real figure demonstration
Input real figure id:
1: Square
2: Rectangle
3: Trapezoid
Input 4 coordinates in a sequence
Coordinates: (0, 0)  (50, 50)  (150, 50)  (100, 0)
```

```
Center: (75, 25)

Area: 5000


++++++++++++++++++++++++++++++++++++

0: Exit

1: Fake figure demonstration

2: Array figure demonstration

3: Real figure demonstration
```

5. Объяснение результатов работы программы:

Пользователь выбирает демонстрацию работы программы с использованием массива точек, tuple или заданных классов. При нажатии 1 выводятся координаты вершин, геометрический центр и площадь трёх фигур: квадрата с вершинами типа float, прямоугольника с вершинами типа int и трапеции с вершинами типа double. При нажатии 2 выводятся координаты вершин, геометрический центр и площадь квадрата и трапеции, заданных в виде массива. При нажатии 3 пользователь сам выбирает одну из трёх фигур и вводит координаты их вершин. После этого выводятся координаты вершин и геометрического центра и площадь введённой фигуры.

6. Вывод:

В ходе выполнения данной работы я изучил основы метапрограммирования, применения шаблонов класса в рамках реализации классов фигур с вершинами с переменным типом данных, причём методы данных классов могут работать с tuple.