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:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Main.cpp

```
#include <iostream>
#include "pentagon.h"
#include "trapeze.h"
#include "templates.h"
#include "rhombus.h"
void help () {
   std::cout << "1 - pentagon\n"
             "2 - trapeze\n"
             "3 - rhombus\n"
             "4 - exit\n";
}

int main() {
   int choice;
   point<double> v1,v2,v3,v4,v5;
   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) {
         trapeze<double> t(std::cin);
         std::cout << "area: " << area(t) << '\n' << "center: " << center(t) << '\n';
         print(t, std::cout);
         std::cout << "Enter a trapezoidal tuple\n";
         std::cin >> v1 >> v2 >> v3 >> v4;
         std::tuple<point<double>, point<double>, point<double>, point<double>>
t1{v1, v2, v3, v4};
         std::cout << "area: " << area(t1) << '\n' << "center: " << center(t1) << '\n';
         print(t1, std::cout);
      } else if (choice == 3) {
         rhombus<double> r(std::cin);
         std::cout << "area: " << area(r) << '\n' << "center: " << center(r) << '\n';
         print(r, std::cout);
```

```
            std::cout << "Enter a rhombus tuple\n";
            std::cin >> v1 >> v2 >> v3 >> v4;
            std::tuple<point<double>,  point<double>,  point<double>,    point<double>>
r1{v1, v2, v3, v4};
            std::cout << "area: " << area(r1) << '\n' << "center: " << center(r1) << '\n';
            print(r1, std::cout);
        } else {
            std::cout << "The command is uncertain\n";
        }
        std::cin >> choice;
    }
    return 0;
}
```

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;
}
```

Rhombus.h

```
#include "point.h"
#pragma once
template<class T>
struct rhombus {
private:
    point<T> a1,a2,a3,a4;
public:
    point<T> center() const;
    void print(std::ostream& os) const;
    double area() const;
    rhombus(std::istream& is);
};
template<class T>
rhombus<T>::rhombus(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4;
    if ( VectProd(operator-(a1, a2), operator-(a4, a3)) == 0  &&
         VectProd(operator-(a1, a4), operator-(a2, a3)) == 0 &&
         ScalProd(operator-(a3, a1), operator-(a4, a2)) == 0 ) {
        std::cout << "Correct" << std::endl;
    }
    else {
        std::cout << "Wrong" << std::endl;
        throw 1;


    }
}
template<class T>
double rhombus<T>::area() const {
    return VectNorm(a3, a1) * VectNorm(a4, a2) / 2;
}

template<class T>
point<T> rhombus<T>::center() const {
    return point<T> {((a1.x + a2.x + a3.x + a4.x) / 4), ((a1.y + a2.y + a3.y + a4.y) /
4)};
}
```

```
template<class T>
void rhombus<T>::print(std::ostream &os) const {
    os << a1 << "\n"<< a2 << "\n" << a3 << "\n" << a4 << "\n";
}
```

Trapeze.h

```
#pragma once

#include "point.h"

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

template<class T>
point<T> trapeze<T>::center() const {
    T x,y;
    x = (a1.x + a2.x + a3.x + a4.x ) / 4;
    y = (a1.y + a2.y + a3.y + a4.y ) / 4;
    return {x,y};
}

template<class T>
trapeze<T>::trapeze(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4;
    if ( ( VectProd(operator-(a1,a2), operator-(a4,a3)) == 0) ||
         (VectProd(operator-(a2,a3), operator-(a1,a4)) == 0) ) {
        std::cout << "Correct" << std::endl;
    } else {
        std::cout << "Wrong" << std::endl;
        throw 1;
    }
}

template<class T>
void trapeze<T>::print(std::ostream& os) const {
```

```
    os << a1 << "\n"<< a2 << "\n" << a3 << "\n" << a4 << "\n";
}

template<class T>
double trapeze<T>::area() const {
    if ( (VectProd(operator-(a1,a2), operator-(a3,a4)) == 0) && (VectProd(operator-
(a2,a3), operator-(a1,a4)) == 0) ) {
        return fabs((VectProd(operator-(a1,a2), operator-(a1,a4)))) ;
    } else if (VectProd(operator-(a1,a2), operator-(a4,a3)) == 0) {
        return ((VectNorm(a1, a2) + VectNorm(a4, a3)) / 2) * sqrt(
            VectNorm(a4, a1) * VectNorm(a4, a1) - (
                pow((
                    (pow((VectNorm(a4, a3) - VectNorm(a1, a2)), 2) +
                     VectNorm(a4, a1) * VectNorm(a4, a1) - VectNorm(a2, a3) *
VectNorm(a2, a3)) /
                    (2 * (VectNorm(a4, a3) - VectNorm(a1, a2)))
                ), 2)
            )
        );

    } else if (VectProd(operator-(a2,a3), operator-(a1,a4)) == 0) {
        return ((VectNorm(a2, a3) + VectNorm(a1, a4)) / 2) * sqrt(
            VectNorm(a1, a2) * VectNorm(a1, a2) - (
                pow((
                    (pow((VectNorm(a1, a4) - VectNorm(a2, a3)), 2) +
                     VectNorm(a1, a2) * VectNorm(a1, a2) - VectNorm(a3, a4) *
VectNorm(a3, a4)) /
                    (2 * (VectNorm(a1, a4) - VectNorm(a2, a3)))
                ), 2)
            )
        );
    }
}
```

Pentagon.h

```
#include "point.h"
#pragma once
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>
pentagon<T>::pentagon(std::istream &is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}
template<class T>
point<T> pentagon<T>::center() const {
    return point<T> {((a1.x + a2.x + a3.x + a4.x + a5.x) / 5), ((a1.y + a2.y + a3.y +
a4.y + a5.y) / 5) } ;
}
template<class T>
double pentagon<T>::area() const {
    return TrAngle(a1,a2,a3) + TrAngle(a3,a4,a5) + TrAngle(a1,a3,a5);
}

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

Point.h

```
#pragma once

#include <iostream>
#include <algorithm>
#include <cmath>
template<class T>
struct point {
    T x;
    T y;
};
template<class T>
point<T> operator- (point<T> l, point<T> r) {
    return { r.x - l.x, r.y - l.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;
}

template<class T>
T VectNorm(point<T> l, point<T> r) {
    point<T> vect = operator-(l, r);
    double res = sqrt(vect.x * vect.x + vect.y * vect.y);
    return res;
}

template<class T>
T ScalProd(point<T> l, point<T> r) {
    return std::abs(l.x * r.x + l.y * r.y);
}
template<class T>
T TrAngle(point<T> a, point<T> b, point<T> c) {
    point<T>  v1, v2;
    v1 = operator-(a, b);
    v2 = operator-(a, c);
    return std::abs(v1.x * v2.y - v2.x * v1.y) / 2;
}

template<class T>
T VectProd(point<T> a, point<T> b) {
    return a.x * b.y - b.x * a.y;
}
```

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_04

3.Набор testcases.

```
3

0 0
-1 1
0 2
1 1

0 0
-1 1
```

0 2
1 1

2

0 0
0 1
1 1
1 0

0 0
0 1
1 1
1 0

5. Объяснение результатов работы программы.

1) При запуске программы вводится одна из 4 возможных команд в виде строки.
    1- Ввести координаты пятиугольника
    2- Ввести координаты трапеции
    3- Ввести координаты ромба.
    4- Завершить работу

После корректного их ввода предлагается ввести координаты какой-либо фигуры еще раз, для обработки координат при помощи std::tuple.

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с метапрограммированием в C++ и реализовал общие методы для различных классов фигур , изучив и применив такой механизм языка, как шаблоны.