Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 4
Вариант 27
Основы метапрограммирования

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

Main.cpp

```
// //  Created by kosha on 2/01/2020.
// //  Copyright © 2019 kosha. All rights reserved.
// //
#include <iostream>
#include <tuple>
#include "Pair.h"
#include "trapezoid.h"
#include "rectangle.h"
#include "rhombus.h"
#include "templates.h"

template<class T>
void init(std::istream& is, std::ostream& os) {
   if constexpr (is_figurelike_tuple<T>::value) {
      int arg;
      std::cin >> arg;
      std::cout << "Input coordinates: " << std::endl;
      if (arg == 4) {
         Pair<double> A, B, C, D;
         is >> A >> B >> C >> D;
         auto object = std::make_tuple(A, B, C, D);
         print(os, object);
         os << "Area: " << area(object) << std::endl;
         os << "Center: " << center(object) << std::endl;
      }else if(arg == 3){
         Pair<double> A, B, C;
         is >> A >> B >> C;
         auto object = std::make_tuple(A, B, C);
         print(os, object);
         os << "Area: " << area(object) << std::endl;
         os << "Center: " << center(object) << std::endl;
      }
   }else {
      T object(is);
      print(os, object);
      os << '\n' <<"Area: " << area(object) << std::endl;
      os << "Center: " << center(object) << std::endl;
   }
}
//4
//3
//0 0 1 1 2 0

int main() {
   char obj_type;
   std:: cout<< " Available input:\n1 - input Trapezoid\n2 - input Rhombus\n3 - input Rectangle\n4 - Tuple\n5 - Exit"
<<std::endl;
   while (std::cin >> obj_type){
      if(obj_type == '4') {
         std::cout << "Input number of vertices: " << std::endl;
```

```
            init<std::tuple<Pair<double>>> (std::cin, std::cout);
        }else if(obj_type == '1'){
            std::cout << "Input Trapezoid coordinates: " << std::endl;
            init<trapezoid<double>>(std::cin, std::cout);
        }else if(obj_type == '2'){
            std::cout << "Input Rhombus coordinates: " << std::endl;
            init<rhombus<double>>(std::cin, std::cout);
        }else if(obj_type == '3'){
            std::cout << "Input Rectangle coordinates: " << std::endl;
            init<rectangle<double>>(std::cin, std::cout);
        }else if(obj_type == '5'){
                    return 0;
        }else{
                    std::cout << "Try another key" << std::endl;
        }
    }
}
```

Pair.h

```
#ifndef Pair_H_
#define Pair_H_

#include <iostream>

template<class T>
struct Pair {
  T x;
  T y;
};

template<class T>
Pair<T> operator+(const Pair<T>& A, const Pair<T>& B) {
    Pair<T> res;
    res.x = A.x + B.x;
    res.y = A.y + B.y;
    return res;
}

template<class T>
Pair<T> operator/=(Pair<T>& A, const double B) { //деление на число
    A.x /= B;
    A.y /= B;
    return A;
}

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

template<class T>
std::ostream& operator<< (std::ostream& os, const Pair<T>& p) {
  os << '(' << p.x << ';' << p.y << ')' ;
  return os;
```

```
}

#endif
```

Rectangle.h

```
#ifndef TRIANGLE_H_
#define TRIANGLE_H_
#include <iostream>
#include "Pair.h"
#include <cmath>

template<class T>
struct rectangle {
  Pair<T> p[4];
  rectangle(std::istream& is);
  double area() const;
  Pair<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 >> p[i];
  }
  double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
  a = sqrt((p[1].x - p[0].x) * (p[1].x - p[0].x) + (p[1].y - p[0].y) * (p[1].y - p[0].y));
  b = sqrt((p[2].x - p[1].x) * (p[2].x - p[1].x) + (p[2].y - p[1].y) * (p[2].y - p[1].y));
  c = sqrt((p[2].x - p[3].x) * (p[2].x - p[3].x) + (p[2].y - p[3].y) * (p[2].y - p[3].y));
  d = sqrt((p[3].x - p[0].x) * (p[3].x - p[0].x) + (p[3].y - p[0].y) * (p[3].y - p[0].y));
  d1 = sqrt((p[1].x - p[3].x) * (p[1].x - p[3].x) + (p[1].y - p[3].y) * (p[1].y - p[3].y));
  d2 = sqrt((p[2].x - p[0].x) * (p[2].x - p[0].x) + (p[2].y - p[0].y) * (p[2].y - p[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("Wrong coordinates. It's not a rectangle.");
}

template<class T>
double rectangle<T>::area() const {
  const T a =  sqrt((p[1].x - p[0].x) * (p[1].x - p[0].x) + (p[1].y - p[0].y) * (p[1].y - p[0].y));
  const T b =  sqrt((p[2].x - p[1].x) * (p[2].x - p[1].x) + (p[2].y - p[1].y) * (p[2].y - p[1].y));
  return a * b;
}

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

template<class T>
```

```
void rectangle<T>::print(std::ostream& os) const {
  for(int i = 0; i < 4; ++i){
     os << p[i];
    if(i + 1 != 4){
     os << ' ';
    }
  }
}
#endif
```

Rhombus.h

```
#ifndef RHOMBUS_H_
#define RHOMBUS_H_
#include <iostream>
#include <cmath>
#include "Pair.h"


template<class T>
struct rhombus {
  Pair<T> p[4];
  rhombus(std::istream& is);
  double area() const;
  Pair<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 >> p[i];
  }
  double a, b, c, d;
  a = sqrt((p[1].x - p[0].x) * (p[1].x - p[0].x) + (p[1].y - p[0].y) * (p[1].y - p[0].y));
  b = sqrt((p[2].x - p[1].x) * (p[2].x - p[1].x) + (p[2].y - p[1].y) * (p[2].y - p[1].y));
  c = sqrt((p[2].x - p[3].x) * (p[2].x - p[3].x) + (p[2].y - p[3].y) * (p[2].y - p[3].y));
  d = sqrt((p[3].x - p[0].x) * (p[3].x - p[0].x) + (p[3].y - p[0].y) * (p[3].y - p[0].y));
  if(a != b || a != c || a != d)
     throw std::logic_error("Wrong coordinates. It's not a rhombus.");
}

template<class T>
double rhombus<T>::area() const {
  const T d1 =  sqrt((p[0].x - p[2].x) * (p[0].x - p[2].x) + (p[0].y - p[2].y) * (p[0].y - p[2].y));
  const T d2 =  sqrt((p[1].x - p[3].x) * (p[1].x - p[3].x) + (p[1].y - p[3].y) * (p[1].y - p[3].y));
  return d1 * d2 / 2;
}

template<class T>
Pair<T> rhombus<T>::center() const {
  Pair<T> res;
  res.x = (p[0].x + p[1].x + p[2].x + p[3].x) / 4;
  res.y = (p[0].y + p[1].y + p[2].y + p[3].y) / 4;
  return res;
```

```
}

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

#endif
```

Trapezoid.h

```
#ifndef trapezoid_H_
#define trapezoid_H_
#include <iostream>
#include <cmath>
#include "Pair.h"


template<class T>
struct trapezoid {
  Pair<T> p[4];
  trapezoid(std::istream& is);
  double area() const;
  Pair<T> center() const;
  void print(std::ostream& os) const;
};
//0 0 1 1 2 1 3 0
template<class T>
trapezoid<T>::trapezoid(std::istream& is) {
  for(int i = 0; i < 4; ++i){
    is >> p[i];
  }
  if((p[2].y - p[1].y) / (p[2].x - p[1].x) != (p[3].y - p[0].y) / (p[3].x - p[0].x))
      throw std::logic_error("Wrong coordinates. It's not a trapezoid");
}

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

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

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

}
```

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

#endif
```

Templates.h

```
#ifndef TEMPLATES_H_
#define TEMPLATES_H_

#include <tuple>
#include <type_traits>
#include "Pair.h"


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

template<class T>
struct is_Pair<Pair<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_Pair<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>, Pair<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>
Pair<double> tuple_center(const T& object) {
    if constexpr (Id >= std::tuple_size<T>::value) {
        return Pair<double> {0, 0};
    } else {
        Pair<double> res = std::get<Id>(object);
        return res + tuple_center<Id+1>(object);
    }
}

template<class T>
Pair<double> compute_center(const T &tuple) {
    Pair<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>, Pair<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
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/oop_exercise_04

3.Набор testcases

1. **Входные данные:**
2
0 2
2 4
4 2
2 0

2. Входные данные:
1
0 0
1 1
2 1
4 0

3. Входные данные:
3
1 2
3 4
4 3
2 1

4. Входные данные:
4
3
0 0 1 1 2 0

4. Результат выполнения тестов

1.
(0;2) (2;4) (4;2) (2;0)
Area: 8
Center: (2;2)

2.
(0;0) (1;1) (2;1) (4;0)
Area: 2.5
Center: (1.75;0.5)

3.
(1;2) (3;4) (4;3) (2;1)

Area: 4
Center: (2.5;2.5)

4.

(0;0) (1;1) (2;0)
Area: 1
Center: (1;0.333333)

5. Объяснение результатов программы

В файле main.cpp реализован интерфейс пользователя и печать фигур.
В файле Pair.h реализована перегрузка операторов для координат точек.
Файлы trapezoid.h, rhombus.h и rectangle.h – это файлы фигур. В них реализованы методы подсчета площади и центра каждой фигуры.
В templates.h описаны шаблоны для этих фигур, а так же tuple.

Вывод: Проделав данную работу я изучила шаблоны, получила навык в создании шаблонных классов.