Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 4

Тема: Основы метапрограммирования

Студент: Подоляка Елена

Группа: М8О-308Б-18

Преподаватель: Журавлев А.А.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных, задающий тип данных для оси координат. Классы должны иметь публичные поля. Создать набор шаблонов, реализующих :

Вычисление геометрического центра фигуры

Вывод в стандартный поток вывода координат вершины фигуры

Вычисление площади фигуры

Параметром шаблона должен являться тип класса фигуры. Помимо самого класса, шаблонная функция должна уметь работать с tuple. Создать программу, которая позволяет вводить фигуры из стандартного ввода и вызвать для них шаблонные функции.

Вариант задания 20:

Трапеция

Ромб

Пятиугольник

2. **Репозиторий github**

https://github.com/markisonka/oop_exercise_04

3. **Описание программы**

Реализованы шаблонные функции area, center, а так же операторы ввода и вывода из потоков. Для шаблонных параметров вызываются функции, позволяющие интерпретировать их либо как фигуру, либо как кортеж, состоящий из точек, инстанцированных от одного типа.

Далее, для параметров, являющихся фигурами просто вызываются соответствующие методы, а для котрежей применяется рекурсивное вычисление площади, центра или рекурсивный ввод и вывод элементов этого кортежа.

Реализована шаблонная функция process, отвечающая за ввод и вывод фигур, их центров и площадей.

4. **Набор testcases**

Тестовые файлы: **test_01.test**, **test_02.test**

test_01.test:

trapeze 0 0 1 3 3 2 4 0

rhombus 0 0 3 4 8 4 5 1

pentagon 0 0 3 4 8 4 5 0 2 -2

Проверка обработки фигур, не удовлетворяющих условиям, а так же проверка конструирования пятиугольника.

Результат работы программы

At least 2 sides of trapeze must be parallel

This is not rhombus, sides arent equal

Pentagon, p1: 0 0, p2: 3 4, p3: 8 4, p4: 5 0, p5: 2 -2

Area of figure: 25

Center of figure: 3.6 1.2

test_02.test:

rhombus 0 0 3 4 8 4 5 0

trapeze 0 0 2 2 4 2 6 0

pentagon 0 0 2 3 7 2 7 -4 2 -5

tuple 3 0 0 2 2 4 0

tuple 4 0 0 3 4 8 4 5 0

tuple 5 0 0 2 3 7 2 7 -4 3 -3

Проверка правильности работы с кортежами, а так же проверка результатов вычисления площадей и центров для данных фигур.

Результат работы программы

Rhombus, p1: 0 0, p2: 3 4, p3: 8 4, p4: 5 0

Area of figure: 20

Center of figure: 4 2

Trapeze p1:0 0, p2:2 2, p3:4 2, p4:6 0

Area of figure: 12

Center of figure: 3 1

Pentagon, p1: 0 0, p2: 2 3, p3: 7 2, p4: 7 -4, p5: 2 -5

Area of figure: 43

Center of figure: 3.6 -0.8

Enter size of tuple

Figurelike tuple of 3 elements: (0 0) (2 2) (4 0)

Area of figurelike tuple: 4

Center of figurelike tuple: 2 0.666667

Enter size of tuple

Figurelike tuple of 4 elements: (0 0) (3 4) (8 4) (5 0)

Area of figurelike tuple: 20

Center of figurelike tuple: 4 2

Enter size of tuple

Figurelike tuple of 5 elements: (0 0) (2 3) (7 2) (7 -4) (3 -3)

Area of figurelike tuple: 34

Center of figurelike tuple: 3.8 -0.4

5. Результаты выполнения тестов

Все тесты успешно пройдены, программа выдаёт верные результаты.

6. Листинг программы

main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <type_traits>
#include <exception>
#include <tuple>
#include "Point.h"
#include "Trapeze.h"
#include "templates.h"
#include "Rhombus.h"
#include "Pentagon.h"
int main() {
```

```
  std::string command;
  while (std::cin >> command) {
    if (command == "exit") {
      break;
    } else if (command == "pentagon") {
      try {
        Pentagon<double> fig;
        process(fig);
      } catch (std::exception& ex) {
        std::cout << ex.what() << "\n";
        continue;
      }
    } else if (command == "rhombus") {
      try {
        Rhombus<double> fig;
        process(fig);
      } catch (std::exception& ex) {
        std::cout << ex.what() << "\n";
        continue;
      }
    } else if (command == "trapeze") {
      try {
        Trapeze<double> fig;
        process(fig);
      } catch (std::exception& ex) {
        std::cout << ex.what() << "\n";
        continue;
      }
    } else if (command == "tuple") {
      int size;
      std::cout << "Enter size of tuple\n";
      std::cin >> size;
      if (size == 3) {
        std::tuple<Point<double>,Point<double>,Point<double>> tuple;
        process(tuple);
      } else if (size == 4) {
        std::tuple<Point<double>,Point<double>,Point<double>,Point<double>>
tuple;
        process(tuple);
      } else if (size == 5) {

std::tuple<Point<double>,Point<double>,Point<double>,Point<double>,Point<dou
ble>> tuple;
        process(tuple);
      } else if (size == 5) {

std::tuple<Point<double>,Point<double>,Point<double>,Point<double>,Point<dou
ble>, Point<double>> tuple;
        process(tuple);
```

```
        } else {
          std::cout << "Wrong size\n";
          continue;
        }
      } else {
        std::cout << "Wrong type\n";
        continue;
      }
    }
    return 0;
}
```

Trapeze.h

```
#pragma once
#include "Point.h"
#include <exception>
template <typename T>
class Trapeze {
public:
    Trapeze() = default;
    Trapeze(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4);
    Point<T> Center() const;
    double Area() const;
    void Print(std::ostream& os) const;
    void Scan(std::istream& is);
private:
    Point<T> p1_, p2_, p3_, p4_;
};
template <typename T>
Trapeze<T>::Trapeze(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4)
      :  p1_(p1), p2_(p2), p3_(p3), p4_(p4){
    Vector v1(p1_, p2_), v2(p3_, p4_);
    if (v1 = Vector(p1_, p2_), v2 = Vector(p3_, p4_), is_parallel(v1, v2)) {
      if (v1 * v2 < 0) {
        std::swap(p3_, p4_);
      }
    } else if (v1 = Vector(p1_, p3_), v2 = Vector(p2_, p4_), is_parallel(v1, v2)) {
      if (v1 * v2 < 0) {
        std::swap(p2_, p4_);
      }
      std::swap(p2_, p3_);
    } else if (v1 = Vector(p1_, p4_), v2 = Vector(p2_, p3_), is_parallel(v1, v2)) {
      if (v1 * v2 < 0) {
        std::swap(p2_, p3_);
      }
      std::swap(p2_, p4_);
      std::swap(p3_, p4_);
    } else {
      throw std::logic_error("At least 2 sides of trapeze must be parallel");
```

```
  }
}
template <typename T>
Point<T> Trapeze<T>::Center() const {
  return (p1_ + p2_ + p3_ + p4_) / 4;
}
template <typename T>
double Trapeze<T>::Area() const {
  double height = point_and_line_distance(p1_, p3_, p4_);
  return (Vector<T>(p1_, p2_).length() + Vector<T>(p3_, p4_).length()) * height / 2;
}
template <typename T>
void Trapeze<T>::Print(std::ostream& os) const {
  os << "Trapeze p1:" << p1_ << ", p2:" << p2_ << ", p3:" << p3_ << ", p4:" <<
p4_;
}
template <typename T>
void Trapeze<T>::Scan(std::istream &is) {
  Point<T> p1,p2,p3,p4;
  is >> p1 >> p2 >> p3 >> p4;
  *this = Trapeze(p1,p2,p3,p4);
}
```

Rhombus.h

```
#pragma once
#include "Point.h"
template <typename T>
class Rhombus {
public:
  Rhombus() = default;
  Rhombus(Point<T> p1_, Point<T> p2_, Point<T> p3_, Point<T> p4_);
  Point<T> Center() const;
  double Area() const;
  void Print(std::ostream& os) const;
  void Scan(std::istream& is);
private:
  Point<T> p1_, p2_, p3_, p4_;
};
template <typename T>
Rhombus<T>::Rhombus(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4)
    : p1_(p1), p2_(p2), p3_(p3), p4_(p4) {
  if (Vector<T>(p1_, p2_).length() == Vector<T>(p1_, p4_).length()
    && Vector<T>(p3_, p4_).length() == Vector<T>(p2_, p3_).length()
    && Vector<T>(p1_, p2_).length() == Vector<T>(p2_, p3_).length()) {
  } else if (Vector<T>(p1_, p4_).length() == Vector<T>(p1_, p3_).length()
        && Vector<T>(p2_, p3_).length() == Vector<T>(p2_, p4_).length()
        && Vector<T>(p1_, p4_).length() == Vector<T>(p2_, p4_).length()) {
    std::swap(p2_, p3_);
```

```
  } else if (Vector<T>(p1_, p3_).length() == Vector<T>(p1_, p2_).length()
        && Vector<T>(p2_, p4_).length() == Vector<T>(p3_, p4_).length()
        && Vector<T>(p1_, p2_).length() == Vector<T>(p2_, p4_).length()) {
    std::swap(p3_, p4_);
  } else {
    throw std::logic_error("This is not rhombus, sides arent equal");
  }
}
template <typename T>
double Rhombus<T>::Area() const {
  return Vector<T>(p1_, p3_).length() * Vector<T>(p2_, p4_).length() / 2;
}
template <typename T>
Point<T> Rhombus<T>::Center() const {
  return (p1_ + p3_) / 2;
}
template <typename T>
void Rhombus<T>::Print(std::ostream& os) const {
  os << "Rhombus, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ << ", p4:
" << p4_;
}
template <typename T>
void Rhombus<T>::Scan(std::istream &is) {
  Point<T> p1,p2,p3,p4;
  is >> p1 >> p2 >> p3 >> p4;
  *this = Rhombus(p1,p2,p3,p4);
}
```

Pentagon.h

```
#pragma once
#include "Point.h"
template <typename T>
class Pentagon {
public:
  Pentagon() = default;
  explicit Pentagon(const Point<T>& p1, const Point<T>& p2, const Point<T>& p3,
const Point<T>& p4, const Point<T>& p5);
  Point<T> Center() const;
  double Area() const;
  void Print(std::ostream& os) const;
  void Scan(std::istream& is);
private:
  Point<T> p1_, p2_, p3_, p4_, p5_;
};
template <typename T>
Pentagon<T>::Pentagon(const Point<T>& p1, const Point<T>& p2, const Point<T>&
p3, const Point<T>& p4, const Point<T>& p5)
    : p1_(p1), p2_(p2), p3_(p3), p4_(p4), p5_(p5) {}
```

```
template <typename T>
double Pentagon<T>::Area() const {
  return
      point_and_line_distance(p1_, p2_, p3_) * Vector<T>(p2_, p3_).length() / 2
      + point_and_line_distance(p1_, p3_, p4_) * Vector<T>(p3_, p4_).length() / 2
      + point_and_line_distance(p1_, p4_, p5_) * Vector<T>(p4_, p5_).length() / 2;
}
template <typename T>
Point<T> Pentagon<T>::Center() const {
  return (p1_ + p2_ + p3_ + p4_ + p5_) / 5;
}
template <typename T>
void Pentagon<T>::Print(std::ostream& os) const {
  os << "Pentagon, p1: " << p1_ << ", p2: " << p2_ << ", p3: " << p3_ << ", p4:
" << p4_ << ", p5: " << p5_;
}
template <typename T>
void Pentagon<T>::Scan(std::istream &is) {
  Point<T> p1, p2, p3, p4, p5;
  is >> p1 >> p2 >> p3 >> p4 >> p5;
  *this = Pentagon(p1,p2,p3,p4,p5);
}
```

Point.h

```
#pragma once
#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
template <typename T>
struct Point {
  T x = 0;
  T y = 0;
};
template <typename T>
class Vector {
public:
  explicit Vector(T a, T b);
  explicit Vector(Point<T> a, Point<T> b);
  bool operator == (Vector rhs);
  Vector operator - ();
  double length() const;
  T x;
  T y;
};
template <typename T>
Point<T> operator + (Point<T> lhs, Point<T> rhs) {
```

```
  return {lhs.x + rhs.x, lhs.y + rhs.y};
}
template <typename T>
Point<T> operator - (Point<T> lhs, Point<T> rhs) {
  return {lhs.x - rhs.x, lhs.y - rhs.y};
}
template <typename T>
Point<T> operator / (Point<T> lhs, double a) {
  return { lhs.x / a, lhs.y / a};
}
template <typename T>
Point<T> operator * (Point<T> lhs, double a) {
  return {lhs.x * a, lhs.y * a};
}
template <typename T>
bool operator < (Point<T> lhs, Point<T> rhs) {
  return (lhs.x * lhs.x + lhs.y * lhs.y) < (lhs.x * lhs.x + lhs.y * lhs.y);
}
template <typename T>
double operator * (Vector<T> lhs, Vector<T> rhs) {
  return lhs.x * rhs.x + lhs.y * rhs.y;
}
template <typename T>
bool is_parallel(const Vector<T>& lhs, const Vector<T>& rhs) {
  return (lhs.x * rhs.y - lhs.y * rhs.y) == 0;
}
template <typename T>
bool Vector<T>::operator == (Vector<T> rhs) {
  return
      std::abs(x - rhs.x) < std::numeric_limits<double>::epsilon() * 100
      && std::abs(y - rhs.y) < std::numeric_limits<double>::epsilon() * 100;
}
template <typename T>
double Vector<T>::length() const {
  return sqrt(x*x + y*y);
}
template <typename T>
Vector<T>::Vector(T a, T b)
    : x(a), y(b) {
}
template <typename T>
Vector<T>::Vector(Point<T> a, Point<T> b)
    : x(b.x - a.x), y(b.y - a.y){
}
template <typename T>
Vector<T> Vector<T>::operator - () {
  return Vector(-x, -y);
}
template <typename T>
```

```cpp
bool is_perpendecular(const Vector<T>& lhs, const Vector<T>& rhs) {
  return (lhs * rhs) == 0;
}
template <typename T>
double point_and_line_distance(Point<T> p1, Point<T> p2, Point<T> p3) {
  double A = p2.y - p3.y;
  double B = p3.x - p2.x;
  double C = p2.x*p3.y - p3.x*p2.y;
  return (std::abs(A*p1.x + B*p1.y + C) / std::sqrt(A*A + B*B));
}
template <typename T>
std::ostream& operator << (std::ostream& os, const Point<T>& p) {
  return os << p.x << " " << p.y;
}
template <typename T>
std::istream& operator >> (std::istream& is, Point<T>& p) {
  return is >> p.x >> p.y;
}
```

templates.h

```cpp
#pragma once
template <typename T>
struct is_point : std::false_type {};
template <typename T>
struct is_point<Point<T>> : std::true_type {};
template <typename T, typename = void>
struct is_figure : std::false_type {};
template <typename T>
struct is_figure<T, std::void_t<decltype(std::declval<const T&>().Area()),
            decltype(std::declval<const T&>().Center()),
            decltype(std::declval<const T&>().Print(std::cout)),
            decltype(std::declval<T&>().Scan(std::cin))>> : std::true_type {};


template <typename T> //площадь для всего, что имеет площадь
decltype(std::declval<const T&>().Area()) area(const T& figure) {
  return figure.Area();
}
template <typename T> //центр для всего, что имеет центр
decltype(std::declval<const T&>().Center()) center(const T& figure) {
  return figure.Center();
}
template <typename T, typename PrintReturnType = decltype(std::declval<const
T&>().Print(std::cout))> //вывод для классов с функцией Print
std::ostream& operator << (std::ostream& os, const T& figure) {
  figure.Print(os);
  return os;
}
```

```cpp
template <typename T, typename PrintReturnType =
decltype(std::declval<T&>().Scan(std::cin))> //вывод для классов с функцией Print
std::istream& operator >> (std::istream& is, T& figure) {
  figure.Scan(is);
  return is;
}
template<class T>
struct is_figurelike_tuple : std::false_type {};
template<class Head, class... Tail>
struct is_figurelike_tuple<std::tuple<Head, Tail...>> : //проверяет, является ли T
кортежом, из которых можно составить фигуру
    std::conjunction<is_point<Head>, std::is_same<Head, Tail>...> {};
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 dx1 = std::get<Id - 0>(tuple).x - std::get<0>(tuple).x;
    const auto dy1 = std::get<Id - 0>(tuple).y - std::get<0>(tuple).y;
    const auto dx2 = std::get<Id - 1>(tuple).x - std::get<0>(tuple).x;
    const auto dy2 = std::get<Id - 1>(tuple).y - std::get<0>(tuple).y;
    const double local_area = std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
    return local_area + compute_area<Id + 1>(tuple);
  }
}
template<typename T>
std::enable_if_t<is_figurelike_tuple<T>::value, double> area(const T& object) {
  if constexpr (std::tuple_size_v<T> < 3){
    return 0;
  }else{
    return compute_area<2>(object);
  }
}
template<size_t Id, typename T>
std::tuple_element_t<0,T> compute_center(const T& tuple) {
  if constexpr (Id == std::tuple_size_v<T> - 1){
    return std::get<Id>(tuple);
  } else {
    return std::get<Id>(tuple) + compute_center<Id + 1>(tuple);
  }
}
template<typename T>
std::enable_if_t<is_figurelike_tuple<T>::value, std::tuple_element_t<0,T>>
center(const T& object) {
  return compute_center<0>(object) / std::tuple_size_v<T>;
}
template<size_t Id, typename T>
void print_tuple(std::ostream& os, const T& tuple) {
  if constexpr (Id == std::tuple_size_v<T> - 1) {
```

```
    os << "(" << std::get<Id>(tuple) << ")";
  } else {
    os << "(" << std::get<Id>(tuple) << ") ";
    print_tuple<Id + 1>(os, tuple);
  }
}
template <typename T>
typename std::enable_if<is_figurelike_tuple<T>::value, std::ostream&>::type
operator << (std::ostream& os, const T& tuple) {
  os << "Figurelike tuple of " << std::tuple_size_v<T> << " elements: ";
  print_tuple<0>(os, tuple);
  return os;
}
template<size_t Id, typename T>
void scan_tuple(std::istream& is, T& tuple) {
  if constexpr (Id == std::tuple_size_v<T> - 1) {
    is >> std::get<Id>(tuple);
  } else {
    is >> std::get<Id>(tuple);
    scan_tuple<Id + 1>(is, tuple);
  }
}
template <typename T>
typename std::enable_if<is_figurelike_tuple<T>::value, std::istream&>::type
operator >> (std::istream& is, T& tuple) {
  scan_tuple<0>(is, tuple);
  return is;
}
template <typename T>
typename std::enable_if<is_figurelike_tuple<T>::value, void>::type process(T& tup)
{
  std::cin >> tup;
  std::cout << tup << "\n";
  std::cout << "Area of figurelike tuple: " << area(tup) << "\n";
  std::cout << "Center of figurelike tuple: " << center(tup) << "\n";
}
template <typename T>
typename std::enable_if<is_figure<T>::value, void>::type process(T& fig) {
  std::cin >> fig;
  std::cout << fig << "\n";
  std::cout << "Area of figure: " << area(fig) << "\n";
  std::cout << "Center of figure: " << center(fig) << "\n";
}
```

7. Вывод

В результате данной работы я улучшила свои навыки работы с шаблонами в C++, узнала много нового о type_traits, а так же о функциях

delcval и decltype. Более подробно познакомилась с идиомой SFINAE. Данная лабораторная работа показала мне многогранность и мощь языка C++.