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: Кудинов Сергей

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. Создаь программу, которая позволяет вводить фигуры из стандартного ввода и вызвать для них шаблонные функции.

Вариант задания 11:

- Трапеция
- Ромб
- Прямоугольник

2. **Репозиторий github**

https://github.com/StormStudioAndroid2/oop_exercise_04/

3. **Описание программы**

Реализованы шаблонные функции getSquare, getCenter, 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_03.test**

test_01.test:

1

5 0 10 0 7 5 10 5

Проверка обработки фигуры трапеция

Результат работы программы

Trapeze 5 0 p1 7 5 p2 10 0 p3 10 5 p4

Area of figure: 20

Center of figure: 8 2.5

test_02.test:

2

3 4 8 4 5 0 0 0

Проверка правильности работы с ромбом

Результат работы программы

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

Area of figure: 20

Center of figure: 4 2

test_03.test:

tuple

4

3 4 8 4 5 0 0 0

Проверка правильности работы с кортежем

Результат работы программы

Enter size of tuple

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

Area of figurelike tuple: 20

Center of figurelike tuple: 4 2

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 "Rhombus.h"
#include "Rectangle.h"
#include "templates.h"

int main() {
    std::string command;
    while (std::cin >> command) {
        if (command == "exit") {
            break;
        } else if (command == "3") {
            try {
                Rectangle<double> rectangle;
                process(rectangle);
            } catch (std::exception& ex) {
                std::cout << ex.what() << "\n";
                continue;
            }
        } else if (command == "2") {
            try {
                Rhombus<double> fig;
                process(fig);
            } catch (std::exception& ex) {
                std::cout << ex.what() << "\n";
                continue;
            }
        } else if (command == "1") {
            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 {
                std::cout << "Wrong size\n";
                continue;
            }
        } else {
            std::cout << "Wrong type\n";
            continue;
        }
    }
    return 0;
}
```

Trapeze.h

```
#pragma once

#include <vector>
template <typename T>

class Trapeze {
    public:
    Point<T> points[4];
    Trapeze<T>() = default;
    Trapeze(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4);
    double getSquare() const;
    Point<T> getCenter() const ;
    void scan(std::istream &is);
    void print(std::ostream& os) const;
};
template <typename T>
Trapeze<T>::Trapeze(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4) {
       if (IsParallel(p1,p2,p3,p4)
        && !IsParallel(p1,p3,p2,p4)) {
        std::swap(p2, p3);

    } else if (!IsParallel(p1,p2,p3,p4)
        && IsParallel(p1,p3,p2,p4)) {


    } else {
        throw std::logic_error("not Trapeze");
    }
    this->points[0] = p1;
    this->points[1] = p2;
    this->points[2] = p3;
    this->points[3] = p4;
    }


template <typename T>
double Trapeze<T>::getSquare() const {

             return   (length(this->points[0],this->points[2])+length(this-
>points[1],this->points[3]))*fabs((this->points[0].y-this-
```

```
>points[1].y))*(0.5);
}
template <typename T>
void Trapeze<T>::print(std::ostream& os) const {
    os << "Trapeze ";
    for (int i = 0; i < 4; ++i) {
        os << this->points[i] << " p" << i+1 <<" ";
    }
        os << std::endl;

}

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);
}

template <typename T>
Point<T> Trapeze<T>::getCenter() const {

    Point<T> p;
    p.x = 0;
    p.y = 0;
    for (size_t i = 0; i <4; ++i) {
        p = p+(points[i]/4);
    }
    return p;
}
```

Rhombus.h

```
#pragma once
#include "Point.h"

#include <vector>
template <typename T>
class Rhombus {
    public:
        Point<T> points[4];
        Rhombus() = default;
        Rhombus(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4);
        double getSquare() const;
        Point<T> getCenter() const;
        void print(std::ostream& os) const;
        void scan(std::istream &is);
};
template <typename T>
Rhombus<T>::Rhombus(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4) {
       if (length(p1, p2) == length(p1, p4)
        && length(p3, p4) == length(p2, p3)
        && length(p1, p2) == length(p2, p3)) {

    } else if (length(p1, p4) == length(p1, p3)
           && length(p2, p3) == length(p2, p4)
           && length(p1, p4) == length(p2, p4)) {
```

```
        std::swap(p2, p3);
    } else if (length(p1, p3) == length(p1, p2)
              && length(p2, p4) == length(p3, p4)
              && length(p1, p2)== length(p2, p4)) {
        std::swap(p3, p4);
    } else {
        throw std::logic_error("not rhombus");
    }
    this->points[0] = p1;
    this->points[1] = p2;
    this->points[2] = p3;
    this->points[3] = p4;
}
template <typename T>
double Rhombus<T>::getSquare() const {
      return           length(this->points[1],this->points[3])*length(this-
>points[0],this->points[2])*0.5;
    }


template <typename T>
Point<T> Rhombus<T>::getCenter() const {

    Point<T> p;
    p.x = 0;
    p.y = 0;
    for (int i = 0; i < 4; ++i) {
        p = p+(points[i]/4);
    }
    return p;
}



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);
}
template <typename T>
      void Rhombus<T>::print(std::ostream& os) const {
    os << "Rhombus: ";
    for (int i = 0; i < 4; ++i) {
        os << this->points[i] << "p" << i+1 <<" ";
    }
        os << std::endl;

    }
```

Rectangle.h

```
#pragma once

#include <vector>
template <typename T>

class Rectangle {
    public:
    Point<T> points[4];
```

```
    Rectangle<T>() = default;
    Rectangle(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4);
    double getSquare() const;
    Point<T> getCenter() const;
    void scan(std::istream &is);
    void print(std::ostream& os) const;
};


template <typename T>
Rectangle<T>::Rectangle(Point<T> p1, Point<T> p2, Point<T> p3, Point<T> p4)
{
       if (IsRectangle(p1,p2,p3,p4)) {

    } else if (IsRectangle(p2, p3, p1, p4)) {
        std::swap(p2, p1); std::swap(p3,p2);
    } else if (IsRectangle(p3, p1, p2, p4)) {
        std::swap(p3, p1); std::swap(p3,p2);
    } else {
        throw std::logic_error("not rectangle");
    }
    this->points[0] = p1;
    this->points[1] = p2;
    this->points[2] = p3;
    this->points[3] = p4;
    }

    template <typename T>
double Rectangle<T>::getSquare() const {

      return           length(this->points[0],this->points[1])*length(this-
>points[0],this->points[3]);
}
template <typename T>
void Rectangle<T>::print(std::ostream& os) const {
    os << "Rectangle p1: ";
    for (int i = 0; i < 4; ++i) {
        os << this->points[i] << "p" << i+1 <<" ";
    }
    os << std::endl;
}
template <typename T>
void Rectangle<T>::scan(std::istream &is) {
    Point<T> p1,p2,p3,p4;
    is >> p1 >> p2 >> p3 >> p4;
    *this = Rectangle(p1,p2,p3,p4);
}

template <typename T>
Point<T> Rectangle<T>::getCenter() const {

    Point<T> p;
    p.x = 0;
    p.y = 0;
    for (size_t i = 0; i < 4; ++i) {
        p = p+(points[i]/4);
    }
    return p;
}
```

Point.h

```
#pragma once

#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>

template <typename T>
struct Point {
    T x;
    T y;
};
template <typename T>
Point<T> operator+(Point<T> left, Point<T> right) {
     Point<T> p;
    p.x = left.x+right.x;
    p.y = left.y+right.y;
        return p;

}

template <typename T>
 Point<T> operator+( Point<T> left, int right) {
     Point<T> p;
    p.x = left.x+right;
    p.y = left.y+right;
        return p;

}

template <typename T>
 Point<T> operator-( Point<T> left,  Point<T> right) {
     Point<T> p;
    p.x = left.x-right.x;
    p.y = left.y-right.y;
        return p;

}

template <typename T>
 Point<T> operator-(Point<T> left, double right) {
     Point<T> p;
    p.x = left.x-right;
    p.y = left.y-right;
        return p;

}

template <typename T>
 Point<T> operator/( Point<T> left, double right) {
    Point<T> p;
    p.x = left.x/right;
    p.y = left.y/right;
        return p;

}
```

```
template <typename T>
 Point<T> operator*(Point<T> left, double right) {
     Point<T> p;
    p.x = left.x*right;
    p.y = left.y*right;
    return p;
}
template <typename T>
double length( Point<T> left,  Point<T> right) {
    return sqrt((left.x-right.x)*(left.x-right.x)+(left.y-right.y)*(left.y-
right.y));
}
template <typename T>
bool IsOrthogonal( Point<T> a,  Point<T> b,  Point<T> c)
{
    return (b.x - a.x) * (b.x - c.x) + (b.y - a.y) * (b.y - c.y) == 0;
}
template <typename T>

bool IsParallel( Point<T> a,  Point<T> b,  Point<T> c,  Point<T> d)
{
     Point<T> a1 = a-b;
     Point<T> a2 = c-d;
          return  ((a1.x*a2.x+a1.y*a2.y)/(length(a,b)*length(c,d))<=-1  ||
(a1.x*a2.x+a1.y*a2.y)/(length(a,b)*length(c,d))>=1);
}

template <typename T>
int IsRectangle( Point<T> a,  Point<T> b,  Point<T> c,  Point<T> d)
{
    return
        IsOrthogonal(a, b, c) &&
        IsOrthogonal(b, c, d) &&
        IsOrthogonal(c, d, a);
}

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

```
#pragma once

#include <utility>
#include "Point.h"

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&>().getSquare()),
                    decltype(std::declval<const T&>().getCenter()),
                    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&>().getSquare()) square(const T& figure) {
    return figure.getSquare();
}

template <typename T> //центр для всего, что имеет центр
decltype(std::declval<const T&>().getCenter()) center(const T& figure) {
    return figure.getCenter();
}

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;
}

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 computeSquare(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 localSquare = std::abs(dx1 * dy2 - dy1 * dx2) * 0.5;
        return localSquare + computeSquare<Id + 1>(tuple);
    }
}

template<typename T>
std::enable_if_t<is_figurelike_tuple<T>::value, double> square(const T&
object) {
    if constexpr (std::tuple_size_v<T> < 3){
        return 0;
    }else{
        return computeSquare<2>(object);
    }
}


template<size_t Id, typename T>
std::tuple_element_t<0,T> computeCenter(const T& tuple) {
    if constexpr (Id == std::tuple_size_v<T> - 1){
        return std::get<Id>(tuple);
    } else {
        return std::get<Id>(tuple) + computeCenter<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 computeCenter<0>(object) / std::tuple_size_v<T>;
}

template<size_t Id, typename T>
void printTuple(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) << ") ";
        printTuple<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: ";
    printTuple<0>(os, tuple);
    return os;
}
```

```
template<size_t Id, typename T>
void scanTuple(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);
        scanTuple<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) {
    scanTuple<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: " << square(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: " << square(fig) << "\n";
    std::cout << "Center of figure: " << center(fig) << "\n";

}
```

7. Вывод

В результате данной работы я улучшил свои навыки работы с шаблонами в C++, узнал много нового о type_traits, а так же о функциях delcval и decltype.