МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №3**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Попов Андрей Викторович, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:** Спроектировать и запрограммировать на языке C++ классы трёх фигур. Классы должны удовлетворять следующим правилам:

* + Должны быть названы как в вариантах задания и расположены в раздельных файлах;
  + Иметь общий родительский класс Figure;
  + Содержать конструктор, принимающий координаты вершин фигуры из стандартного потока std::cin, расположенных через пробел (например: 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0);
  + Содержать набор общих методов:
    - size\_t VertexesNumber() – метод, возвращающий количество вершин фигуры
    - double Area() – метод расчета площади фигуры

**Вариант №20:**

* + Фигура 1: Трапеция (Trapezoid)
  + Фигура 2: Ромб (Rhombus)
  + Фигура 3: Пятиугольник (Pentagon)

**Описание программы:**

Исходный код разделён на 10 файлов:

* point.h – описание класса точки
* point.cpp – реализация класса точки
* figure.h – описание класса фигуры
* trapezoid.h – описание класса трапеции (наследуется от фигуры)
* trapezoid.cpp – реализация класса трапеции
* rhombus.h – описание класса ромб (наследуется от фигуры)
* rhombus.cpp – реализация класса ромб
* pentagon.h – описание класса пятиугольник (наследуется от фигуры)
* pentagon.cpp – реализация класса пятиугольник
* main.cpp – основная программа

**Дневник отладки:**

Возникли проблемы при вычислении площади трапеции. Этот недочет удалось заметить при тестировании и, изменив формулу рассчета площади на более общую, исправить.

**Вывод:**  
В данной лабораторной работе я познакомился с принципами и концепциями объектно- ориентированного программирования: инкапсуляцией, наследованием и полиморфизмом. Научился проектировать классы и работать с ними, а также поработал с конструкторами, деструкторами и виртуальными функциями в С++.

**Исходный код:**

**figure.h**

**#ifndef FIGURE\_H**

**#define FIGURE\_H**

**#include "point.h"**

**using namespace std;**

**class Figure {**

**public:**

**virtual size\_t VertexesNumber() = 0;**

**virtual double Area() = 0;**

**virtual void Print(ostream &os) = 0;**

**~Figure() {};**

**};**

**#endif // FIGURE\_H**

**main.cpp**

**#include <iostream>**

**#include <vector>**

**#include "figure.h"**

**#include "trapezoid.h"**

**#include "rhombus.h"**

**#include "pentagon.h"**

**#define ll long long**

**using namespace std;**

**int main(void)**

**{**

**Point x1(3, 4);**

**Point x2(6, 2);**

**Point x3(3, 0);**

**Point x4(0, 2);**

**Point x5(0, 4);**

**Rhombus r(x1, x2, x3, x4);**

**Pentagon s(x1, x2, x3, x4, x5);**

**Trapezoid t(x1, x2, x3, x4);**

**vector<Figure\*> vec;**

**vec.push\_back(&r);**

**vec.push\_back(&t);**

**vec.push\_back(&s);**

**cout << r << s << t;**

**for (ll unsigned int i = 0; i < vec.size(); ++i) {**

**cout << "\nvec[" << i << "]:\n";**

**vec[i]->Print(cout);**

**cout << "Area = " << vec[i]->Area() << "\nVertexes number = ";**

**cout << vec[i]->VertexesNumber() << "\n";**

**}**

**}**

**point.h**

**#ifndef POINT\_H**

**#define POINT\_H**

**#include <iostream>**

**using namespace std;**

**class Point**

**{**

**public:**

**Point();**

**Point(istream &is);**

**Point(double x, double y);**

**double dist(Point &other);**

**double dist2(Point &other);**

**double getX();**

**double getY();**

**friend istream &operator>>(istream &is, Point &p);**

**friend ostream &operator<<(ostream &os, Point &p);**

**private:**

**double x;**

**double y;**

**};**

**#endif // POINT\_H**

**point.cpp**

**#include "point.h"**

**#include <cmath>**

**using namespace std;**

**Point::Point() : x(0.0), y(0.0) {}**

**Point::Point(double x, double y) : x(x), y(y) {}**

**Point::Point(istream &is)**

**{**

**is >> x >> y;**

**}**

**double Point::dist(Point &other)**

**{**

**double a = (other.x - x);**

**double b = (other.y - y);**

**return sqrt(a \* a + b \* b);**

**}**

**double Point::dist2(Point &other)**

**{**

**double a = (other.x - x);**

**double b = (other.y - y);**

**return a \* a + b \* b;**

**}**

**double Point::getX()**

**{**

**return x;**

**}**

**double Point::getY()**

**{**

**return y;**

**}**

**istream &operator>>(istream &is, Point &p)**

**{**

**is >> p.x >> p.y;**

**return is;**

**}**

**ostream &operator<<(ostream &os, Point &p)**

**{**

**os << "(" << p.x << ", " << p.y << ")";**

**return os;**

**}**

**trapezoid.h**

**#ifndef TRAPEZOID\_H**

**#define TRAPEZOID\_H**

**#include "figure.h"**

**using namespace std;**

**class Trapezoid : public Figure**

**{**

**public:**

**size\_t VertexesNumber();**

**double Area();**

**void Print(ostream &os);**

**Trapezoid();**

**Trapezoid(Point at, Point bt, Point ct, Point dt);**

**Trapezoid(istream &is);**

**friend istream &operator>>(istream &is, Trapezoid &figure);**

**friend ostream &operator<<(ostream &os, Trapezoid &figure);**

**private:**

**Point a;**

**Point b;**

**Point c;**

**Point d;**

**};**

**#endif // TRAPEZOID\_H**

**trapezoid.cpp**

**#include <cmath>**

**#include "trapezoid.h"**

**using namespace std;**

**Trapezoid::Trapezoid() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0) {}**

**Trapezoid::Trapezoid(Point at, Point bt, Point ct, Point dt) : a(at), b(bt), c(ct), d(dt) {}**

**Trapezoid::Trapezoid(istream &is)**

**{**

**is >> a >> b >> c >> d;**

**}**

**size\_t Trapezoid::VertexesNumber()**

**{**

**return 4;**

**}**

**/\*double Trapezoid::Area() // 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0**

**{**

**double AB = a.dist(b);**

**double BC = b.dist(c);**

**double CD = c.dist(d);**

**double DA = d.dist(a);**

**if (DA > BC) {**

**double t = DA;**

**DA = BC;**

**BC = t;**

**}**

**double p = (AB + BC + CD + DA) / 2.0;**

**double num = sqrt((p - BC) \* (p - DA) \* (p - DA - AB) \* (p - DA - CD));**

**double denum = abs(BC - DA);**

**return (num / denum);**

**}\*/**

**double Trapezoid::Area() // 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0**

**{**

**double la = a.dist(d);**

**double lb = b.dist(c);**

**double lc = c.dist(d);**

**double ld = a.dist(b);**

**if (la > lb) {**

**double t = la;**

**la = lb;**

**lb = t;**

**}**

**double numenator = (lb - la) \* (lb - la) + lc \* lc - ld \* ld;**

**double denumenator = 2 \* (lb - la);**

**if (denumenator == 0) {**

**return (la \* lc);**

**}**

**double h = sqrt(lc \* lc - ((numenator \* numenator) / (denumenator \* denumenator)));**

**return ((la + lb) / 2 \* h);**

**}**

**void Trapezoid::Print(std::ostream &os)**

**{**

**os << "Trapezoid: " << a << " " << b << " " << c << " " << d << "\n";**

**}**

**istream &operator>>(istream &is, Trapezoid &r)**

**{**

**is >> r.a >> r.b >> r.c >> r.d;**

**return is;**

**}**

**ostream &operator<<(ostream &os, Trapezoid &r)**

**{**

**os << "Trapezoid: " << r.a << " " << r.b << " " << r.c << " " << r.d << "\n";**

**return os;**

**}**

**rhombus.h**

**#ifndef RHOMBUS\_H**

**#define RHOMBUS\_H**

**#include "figure.h"**

**using namespace std;**

**class Rhombus : public Figure**

**{**

**public:**

**size\_t VertexesNumber() override;**

**double Area() override;**

**void Print(ostream &os) override;**

**Rhombus();**

**Rhombus(Point at, Point bt, Point ct, Point dt);**

**Rhombus(istream &is);**

**friend istream &operator>>(istream &is, Rhombus &figure);**

**friend ostream &operator<<(ostream &os, Rhombus &figure);**

**protected:**

**Point a;**

**Point b;**

**Point c;**

**Point d;**

**};**

**#endif // RHOMBUS\_H**

**rhombus.cpp**

**#include "figure.h"**

**#include "rhombus.h"**

**using namespace std;**

**Rhombus::Rhombus() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0) {}**

**Rhombus::Rhombus(Point at, Point bt, Point ct, Point dt) : a(at), b(bt), c(ct), d(dt) {}**

**Rhombus::Rhombus(istream &is)**

**{**

**is >> a >> b >> c >> d;**

**}**

**size\_t Rhombus::VertexesNumber()**

**{**

**return 4;**

**}**

**double Rhombus::Area()**

**{**

**return (a.dist(c) \* b.dist(d)) / 2;**

**}**

**void Rhombus::Print(std::ostream &os)**

**{**

**os << "Rhombus: " << a << " " << b << " " << c << " " << d << "\n";**

**}**

**istream &operator>>(istream &is, Rhombus &r)**

**{**

**is >> r.a >> r.b >> r.c >> r.d;**

**return is;**

**}**

**ostream &operator<<(ostream &os, Rhombus &r)**

**{**

**os << "Rhombus: " << r.a << " " << r.b << " " << r.c << " " << r.d << "\n";**

**return os;**

**}**

**trapezoid.h**

**#ifndef TRAPEZOID\_H**

**#define TRAPEZOID\_H**

**#include "figure.h"**

**using namespace std;**

**class Trapezoid : public Figure**

**{**

**public:**

**size\_t VertexesNumber();**

**double Area();**

**void Print(ostream &os);**

**Trapezoid();**

**Trapezoid(Point at, Point bt, Point ct, Point dt);**

**Trapezoid(istream &is);**

**friend istream &operator>>(istream &is, Trapezoid &figure);**

**friend ostream &operator<<(ostream &os, Trapezoid &figure);**

**private:**

**Point a;**

**Point b;**

**Point c;**

**Point d;**

**};**

**#endif // TRAPEZOID\_H**

**trapezoid.cpp**

**#include <cmath>**

**#include "trapezoid.h"**

**using namespace std;**

**Trapezoid::Trapezoid() : a(0.0, 0.0), b(0.0, 0.0), c(0.0, 0.0), d(0.0, 0.0) {}**

**Trapezoid::Trapezoid(Point at, Point bt, Point ct, Point dt) : a(at), b(bt), c(ct), d(dt) {}**

**Trapezoid::Trapezoid(istream &is)**

**{**

**is >> a >> b >> c >> d;**

**}**

**size\_t Trapezoid::VertexesNumber()**

**{**

**return 4;**

**}**

**/\*double Trapezoid::Area() // 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0**

**{**

**double AB = a.dist(b);**

**double BC = b.dist(c);**

**double CD = c.dist(d);**

**double DA = d.dist(a);**

**if (DA > BC) {**

**double t = DA;**

**DA = BC;**

**BC = t;**

**}**

**double p = (AB + BC + CD + DA) / 2.0;**

**double num = sqrt((p - BC) \* (p - DA) \* (p - DA - AB) \* (p - DA - CD));**

**double denum = abs(BC - DA);**

**return (num / denum);**

**}\*/**

**double Trapezoid::Area() // 0.0 0.0 1.0 0.0 1.0 1.0 0.0 1.0**

**{**

**double la = a.dist(d);**

**double lb = b.dist(c);**

**double lc = c.dist(d);**

**double ld = a.dist(b);**

**if (la > lb) {**

**double t = la;**

**la = lb;**

**lb = t;**

**}**

**double numenator = (lb - la) \* (lb - la) + lc \* lc - ld \* ld;**

**double denumenator = 2 \* (lb - la);**

**if (denumenator == 0) {**

**return (la \* lc);**

**}**

**double h = sqrt(lc \* lc - ((numenator \* numenator) / (denumenator \* denumenator)));**

**return ((la + lb) / 2 \* h);**

**}**

**void Trapezoid::Print(std::ostream &os)**

**{**

**os << "Trapezoid: " << a << " " << b << " " << c << " " << d << "\n";**

**}**

**istream &operator>>(istream &is, Trapezoid &r)**

**{**

**is >> r.a >> r.b >> r.c >> r.d;**

**return is;**

**}**

**ostream &operator<<(ostream &os, Trapezoid &r)**

**{**

**os << "Trapezoid: " << r.a << " " << r.b << " " << r.c << " " << r.d << "\n";**

**return os;**

**}**

**Пример работы:**

**Rhombus: (3, 4) (6, 2) (3, 0) (0, 2)**

**Pentagon: (3, 4) (6, 2) (3, 0) (0, 2) (0, 4)**

**Trapezoid: (3, 4) (6, 2) (3, 0) (0, 2)**

**vec[0]:**

**Rhombus: (3, 4) (6, 2) (3, 0) (0, 2)**

**Area = 12**

**Vertexes number = 4**

**vec[1]:**

**Trapezoid: (3, 4) (6, 2) (3, 0) (0, 2)**

**Area = 12**

**Vertexes number = 4**

**vec[2]:**

**Pentagon: (3, 4) (6, 2) (3, 0) (0, 2) (0, 4)**

**Area = 15**

**Vertexes number = 5**