МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №3**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студентка: *Ивченко Анна Владимировна*

Группа:  *М8О-208Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Москва, 2021

**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 28:**

* + Фигура 1: Trapezoid (трацеция)
  + Фигура 2: Rhombus (ромб)
  + Фигура 3: Pentagon (пятиугольник)

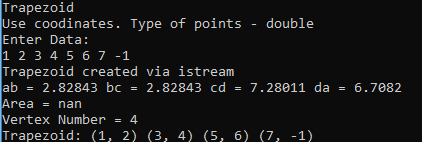
**Описание программы:**

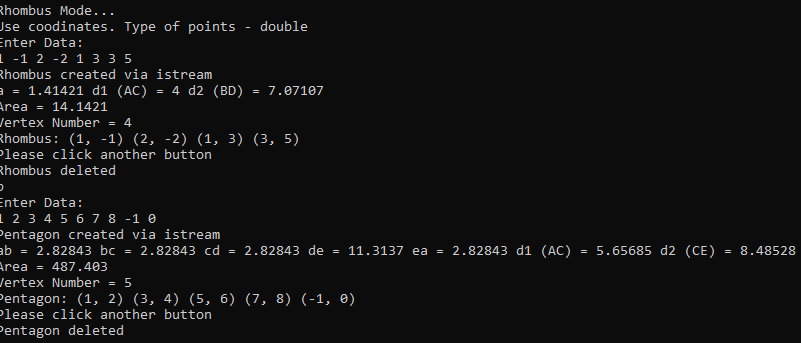
Программа состоит из 10 файлов:

* main.cpp ­­­– основная программа
* figure.h – описание класса фигуры
* point.cpp – описание класса точки
* point.h – реализация класса точки
* trapezoid.cpp – реализация класса трапеции
* trapezoid.h – описисание класса трапеции, наследуется от фигуры
* rhombus.h – описание класса ромба, наследуется от фигуры
* rhombus.cpp – реализация класса рома
* pentagon.cpp – реалицая класса пятиугольника
* pentagon.h – описание класса пятиугольника, наследуется от фигуры **Днвник отладки:**

В ходе выполнения лабораторной работы, программа не нужндалась в отладке, ошибки, возникающие при компиляции исправлялись уже после первой попытки и программа работала исправно

**Тестирование программы:**





**Вывод:**

В ходе проделанной работы, было ознакомление с основными принципами ООП: инкапсуляцией, наследованием и полиморфизмом. Мы создавали классы различных фигур и наследовали их от класса figure. Также была работа с конструторами и деструкторами, познакомились с перезагрузкоми операторов ввода и вывода. Знания, полученные в этой лабораторной работой еще не раз пригодятся для дальшейших лабораторных работ.

**Исходный код:**

**main.cpp**

#include "trapezoid.h"

#include "rhombus.h"

#include "pentagon.h"

int main()

{

std::cout << "press 1 for start: ";

int xui;

std::cin >> xui;

if (xui) {

char c;

std::cout << "Press '?':\n";

while ((c = getchar()) != EOF) {

if (c == '?') {

std::cout << "press t -- for trapezoid\n";

std::cout << "press r -- for Rhombus\n";

std::cout << "press p -- for Pentagon\n";

std::cout << "press e -- Exit\n";

}

else if (c == 't') {

std::cout << "Trapezoid \nUse coodinates. Type of points - double\n";

Trapezoid a(std::cin);

std::cout << "ab = " << a.len\_ab << " bc = " << a.len\_bc << " cd = " << a.len\_cd << " da = " << a.len\_da << std::endl;

std::cout << "Area = " << a.Area() << std::endl;

std::cout << "Vertex Number = " << a.VertexesNumber() << std::endl;

a.Print(std::cout);

std::cout << "Please click another button\n";

}

else if (c == 'r') {

std::cout << "Rhombus Mode...\nUse coodinates. Type of points - double\n";

Rhombus b(std::cin);

std::cout << "a = " << b.len\_a << " d1 (AC) = " << b.d\_ac << " d2 (BD) = " << b.d\_bd << std::endl;

std::cout << "Area = " << b.Area() << std::endl;

std::cout << "Vertex Number = " << b.VertexesNumber() << std::endl;

b.Print(std::cout);

std::cout << "Please click another button\n";

}

else if (c == 'p') {

Pentagon c(std::cin);

std::cout << "ab = " << c.len\_ab

<< " bc = " << c.len\_bc

<< " cd = " << c.len\_cd

<< " de = " << c.len\_de

<< " ea = " << c.len\_ea

<< " d1 (AC) = " << c.d\_ac

<< " d2 (CE) = " << c.d\_ce

<< std::endl;

std::cout << "Area = " << c.Area() << std::endl;

std::cout << "Vertex Number = " << c.VertexesNumber() << std::endl;

c.Print(std::cout);

std::cout << "Please click another button\n";

}

else if (c == 'e') {

std::cout << "End" << std::endl;

return 0;

}

else if (!(c == 't' || c == 'r' || c == 'p' || c == '?') && (c != ' ' && c != '\n' && c != '\t')) {

std::cout << "try again\n";

}

}

}

return 0;

}

**figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

virtual void Print(std::ostream& os) = 0;

virtual ~Figure() {};

};

**point.cpp**

#include "point.h

#include <iostream>

#include <cmath>

Point::Point() : x\_(0.0), y\_(0.0) {}

Point::Point(double x, double y) : x\_(x), y\_(y) {}

Point::Point(std::istream &is) {

is >> x\_ >> y\_;

}

double Point::dist(Point& other) {

double dx = (other.x\_ - x\_);

double dy = (other.y\_ - y\_);

return std::sqrt(dx\*dx + dy\*dy);

}

std::istream& operator>>(std::istream& is, Point& p) {

is >> p.x\_ >> p.y\_;

return is;

}

std::ostream& operator<<(std::ostream& os, Point& p) {

os << "(" << p.x\_ << ", " << p.y\_ << ")";

return os;

}

**point.h**

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

double dist(Point& other);

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

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

protected:

double x\_;

double y\_;

};

#endif // POINT\_H

**trapezoid.cpp**

#include "trapezoid.h"

#include <cmath>

static Point a\_o, b\_o, c\_o, d\_o;

Trapezoid::Trapezoid()

: len\_ab(0.0),

len\_bc(0.0),

len\_cd(0.0),

len\_da(0.0) {

std::cout << "Default Trapezoid created" << std::endl;

}

Trapezoid::Trapezoid(double ab, double bc, double cd, double da)

: len\_ab(ab),

len\_bc(bc),

len\_cd(cd),

len\_da(da) {

std::cout << "Trapezoid created" << std::endl;

}

Trapezoid::Trapezoid(std::istream &is) {

std::cout << "Enter Data:" << std::endl;

is >> a\_o >> b\_o >> c\_o >> d\_o;

len\_ab = a\_o.dist(b\_o);

len\_bc = b\_o.dist(c\_o);

len\_cd = c\_o.dist(d\_o);

len\_da = d\_o.dist(a\_o);

std::cout << "Trapezoid created via istream" << std::endl;

}

Trapezoid::Trapezoid(const Trapezoid& other)

: Trapezoid(other.len\_ab, other.len\_bc, other.len\_cd, other.len\_da) {

std::cout << "Made copy of Trapezoid" << std::endl;

}

size\_t Trapezoid::VertexesNumber() {

return 4;

}

double Trapezoid::Area() {

double p = (len\_ab + len\_bc + len\_cd + len\_da) / 2;

return (len\_bc + len\_da) \*

std::sqrt((p - len\_bc) \*

(p - len\_da) \*

(p - len\_da - len\_ab) \*

(p - len\_da - len\_cd)) /

std::abs(len\_bc - len\_da);

}

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

std::cout << "Trapezoid: ";

os << a\_o; std::cout << " ";

os << b\_o; std::cout << " ";

os << c\_o; std::cout << " ";

os << d\_o; std::cout << std::endl;

}

Trapezoid::~Trapezoid() {

std::cout << "Trapezoid deleted" << std::endl;

}

**trapezoid.h**

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include "figure.h"

#include <iostream>

#include "point.h"

class Trapezoid : public Figure {

public:

Trapezoid();

Trapezoid(double a, double b, double c, double d);

Trapezoid(std::istream &is);

Trapezoid(const Trapezoid& other);

virtual ~Trapezoid();

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

public:

double len\_ab, len\_bc, len\_cd, len\_da;

};

#endif // TRAPEZOID\_H

**rhombus.cpp**

#include "rhombus.h"

#include <cmath>

static Point a\_o, b\_o, c\_o, d\_o;

Rhombus::Rhombus()

: len\_a(0.0),

d\_ac(0.0),

d\_bd(0.0) {

std::cout << "Default Rhombus created" << std::endl;

}

Rhombus::Rhombus(double a\_, double d\_ac\_, double d\_bd\_)

: len\_a(a\_),

d\_ac(d\_ac\_),

d\_bd(d\_bd\_) {

std::cout << "Rhombus created" << std::endl;

}

Rhombus::Rhombus(std::istream &is) {

std::cout << "Enter Data:" << std::endl;

is >> a\_o >> b\_o >> c\_o >> d\_o;

len\_a = a\_o.dist(b\_o);

d\_ac = a\_o.dist(c\_o);

d\_bd = b\_o.dist(d\_o);

std::cout << "Rhombus created via istream" << std::endl;

}

Rhombus::Rhombus(const Rhombus& other)

: Rhombus(other.len\_a, other.d\_ac, other.d\_bd) {

std::cout << "Made copy of Rhombus" << std::endl;

}

size\_t Rhombus::VertexesNumber() {

return 4;

}

double Rhombus::Area() {

return (d\_bd \* d\_ac) / 2;

}

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

std::cout << "Rhombus: ";

os << a\_o; std::cout << " ";

os << b\_o; std::cout << " ";

os << c\_o; std::cout << " ";

os << d\_o; std::cout << std::endl;

}

Rhombus::~Rhombus() {

std::cout << "Rhombus deleted" << std::endl;

}

**rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include "figure.h"

#include <iostream>

#include "point.h"

class Rhombus : public Figure {

public:

Rhombus();

Rhombus(double a\_, double d\_ac\_, double d\_bd\_);

Rhombus(std::istream &is);

Rhombus(const Rhombus& other);

virtual ~Rhombus();

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

public:

double len\_a, d\_ac, d\_bd;

};

#endif // RHOMBUS\_H

**pentagon.cpp**

#include "pentagon.h"

#include <cmath>

static Point a\_o, b\_o, c\_o, d\_o, e\_o;

double triangle\_area(double a, double b, double c) {

double p = (a + b + c);

double s = std::sqrt(p \* (p - a) \* (p - b) \* (p - c));

return s;

}

Pentagon::Pentagon()

: len\_ab(0.0),

len\_bc(0.0),

len\_cd(0.0),

len\_de(0.0),

len\_ea(0.0),

d\_ac(0.0),

d\_ce(0.0) {

std::cout << "Default Pentagon created" << std::endl;

}

Pentagon::Pentagon(double ab, double bc, double cd, double de,

double ea, double d\_ac\_, double d\_ce\_)

: len\_ab(ab),

len\_bc(bc),

len\_cd(cd),

len\_de(de),

len\_ea(ea),

d\_ac(d\_ac\_),

d\_ce(d\_ce\_) {

std::cout << "Pentagon created" << std::endl;

}

Pentagon::Pentagon(std::istream &is) {

std::cout << "Enter Data:" << std::endl;

is >> a\_o >> b\_o >> c\_o >> d\_o >> e\_o;

len\_ab = a\_o.dist(b\_o);

len\_bc = b\_o.dist(c\_o);

len\_cd = c\_o.dist(d\_o);

len\_de = d\_o.dist(e\_o);

len\_ea = e\_o.dist(a\_o);

d\_ac = a\_o.dist(c\_o);

d\_ce = c\_o.dist(e\_o);

std::cout << "Pentagon created via istream" << std::endl;

}

Pentagon::Pentagon(const Pentagon& other)

: Pentagon(other.len\_ab, other.len\_bc, other.len\_cd,

other.len\_de, other.len\_ea, other.d\_ac, other.d\_ce) {

std::cout << "Made copy of Pentagon" << std::endl;

}

size\_t Pentagon::VertexesNumber() {

return 5;

}

double Pentagon::Area() {

return triangle\_area(len\_ab, len\_bc, d\_ac) +

triangle\_area(d\_ac, d\_ce, len\_ea) +

triangle\_area(len\_cd, len\_de, d\_ce);

}

void Pentagon::Print(std::ostream& os) {

std::cout << "Pentagon: ";

os << a\_o; std::cout << " ";

os << b\_o; std::cout << " ";

os << c\_o; std::cout << " ";

os << d\_o; std::cout << " ";

os << e\_o; std::cout << std::endl;

}

Pentagon::~Pentagon() {

std::cout << "Pentagon deleted" << std::endl;

}

**pentagon.h**

#ifndef PENTAGON\_H

#define PENTAGON\_H

#include "figure.h"

#include <iostream>

#include "point.h"

class Pentagon : public Figure {

public:

Pentagon();

Pentagon(double ab, double bc, double cd, double de,

double ea, double d\_ac\_, double d\_ce\_);

Pentagon(std::istream &is);

Pentagon(const Pentagon& other);

virtual ~Pentagon();

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

public:

double len\_ab, len\_bc, len\_cd, len\_de, len\_ea, d\_ac, d\_ce;

};

#endif // PENTAGON\_H