Лабораторная работа №3

**Хашимов Амир Азизович**

**М8О-207Б-20**

**Вариант - 27**

# Задание

Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:

* Должны быть названы также, как в вариантах задания и расположенны в раздельных файлах: отдельно заголовки (имя\_класса\_с\_маленькой\_буквы.h), отдельно описание методов (имя\_класса\_с\_маленькой\_буквы.cpp).
* Иметь общий родительский класс Figure;
* Содержать конструктор, принимающий координаты вершин фигуры из стандартного потока std::cin, расположенных через пробел.

Пример: "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;
  + void Print(std::ostream& os) - метод печати типа фигуры и ее координат вершин в поток вывода os в формате: "Rectangle: (0.0, 0.0) (1.0, 0.0) (1.0, 1.0) (0.0, 1.0)" с переводом строки в конце.

# Структура проекта и алгоритм программы:

* Структура Point (point.h) с double полями x и y, а также методами ToString() и DistanceTo(Point endPoint) [считает по Пифагору]
* Родительский класс Figure (figure.h + figure.cpp) с реализованными виртуальными публичными методами VertexesNumber() [возвращает apixes.size()], Print(std::ostream& os) [выводит с помощью Point.ToString()], чисто виртуальным методом Area() и protected полями std::string figureName и std::vector<Point>apixes
  + Класс-наследник Rectangle (rectangle.h и rectangle.cpp), переопределяющий метод Area() [возвращает произведение двух смежных сторон] (в конструкторе кроме ввода определяется figureName = “Rectangle”)
  + Класс-наследник Trapezoid (trapezoid.h и trapezoid.cpp), переопределяющий метод Area() [возвращает полупроизведение двух диагоналей и угла между ними] (в конструкторе кроме ввода определяется figureName = “Trapezoid”)
  + Класс-наследник Rhombus (rhombus.h и rhombus.cpp), переопределяющий метод Area() [возвращает полупроизведение двух диагоналей] (в конструкторе кроме ввода определяется figureName = “ Rhombus ”)

// point.h

#pragma once

#include <string>

struct Point {

double x, y;

Point(double x, double y)

{

this->x = x;

this->y = y;

}

std::string ToString()

{

return "(" + std::to\_string(x) + ", " + std::to\_string(y) + ")";

}

double DistanceTo(Point endPoint)

{

return sqrt((x - endPoint.x) \* (x - endPoint.x)

+ (y - endPoint.y) \* (y - endPoint.y));

}

};

// figure.cpp

#include "figure.h"

size\_t Figure::VertexesNumber()

{

return apixes.size();

}

void Figure::Print(std::ostream& os)

{

std::string output = figureName + ": ";

for (auto& apix : apixes)

{

output += apix.ToString();

}

os << output + "**\n**";

}

// figure.h

#pragma once

#include <ostream>

#include "point.h"

#include <vector>

class Figure

{

protected:

std::vector<Point> apixes;

std::string figureName;

public:

virtual size\_t VertexesNumber();

virtual double Area() = 0;

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

};

// rectangle.cpp

#include "rectangle.h"

#include "point.h"

#include <iostream>

double Rectangle::Area()

{

return apixes[0].DistanceTo(apixes[1])

\* apixes[1].DistanceTo(apixes[2]);

}

Rectangle::Rectangle(std::istream& inputStream)

{

figureName = "Rectangle";

for (int i = 0; i < 4; i++)

{

Point inputPoint(0, 0);

inputStream >> inputPoint.x >> inputPoint.y;

apixes.push\_back(inputPoint);

}

}

// rectangle.h

#pragma once

#include "figure.h"

class Rectangle :

public Figure

{

public:

double Area() override;

Rectangle(std::istream&);

};

// rhombus.cpp

#include "rhombus.h"

#include <iostream>

double Rhombus::Area()

{

return apixes[0].DistanceTo(apixes[2])

\* apixes[1].DistanceTo(apixes[3]) / 2;

}

Rhombus::Rhombus(std::istream& inputStream)

{

figureName = "Rhombus";

for (int i = 0; i < 4; i++)

{

Point inputPoint(0, 0);

inputStream >> inputPoint.x >> inputPoint.y;

apixes.push\_back(inputPoint);

}

}

// rhombus.h

#pragma once

#include "figure.h"

class Rhombus:

public Figure

{

public:

double Area() override;

Rhombus(std::istream&);

};

// trapezoid.cpp

#include "rhombus.h"

#include <iostream>

double Trapezoid::Area()

{

return apixes[0].DistanceTo(apixes[2])

\* apixes[1].DistanceTo(apixes[3]) / 2;

}

Trapezoid::Trapezoid (std::istream& inputStream)

{

figureName = "Trapezoid";

for (int i = 0; i < 4; i++)

{

Point inputPoint(0, 0);

inputStream >> inputPoint.x >> inputPoint.y;

apixes.push\_back(inputPoint);

}

}

// trapezoid.h

#pragma once

#include "figure.h"

class Trapezoid:

public Figure

{

public:

double Area() override;

Trapezoid (std::istream&);

};