МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

по курсу “Объектно-ориентированное программирование» 1 семестр,

2021/22 уч. год

Студент: *Колпакова Диана Саргаевна, группа М8О-208Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Задание

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

- Должны быть названы так же, как в вариантах задания, и расположены в раздельных файлах: отдельно заголовки (имя_класса_с_маленькой_буквы.h), отдельно описания методов (имя_класса_с_маленькой_буквы.cpp);
- Иметь общий родительский класс Figure;
- Содержать конструктор по умолчанию;
- 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)\n

Программа должна позволять:

- Вводить произвольные фигуры и добавлять их в общий контейнер. Разрешается использовать стандартные контейнеры std;
- Распечатывать содержимое контейнера.

Вариант 9:

Фигура №1	**Имя класса**	**Фигура №2**	**Имя класса**	**Фигура №3**	**Имя класса**
Треугольник	Triangle	Квадрат	Square	Прямоугольник	Rectangle

Описание программы

Исходный код лежит в 11 файлах:

1. main.cpp: часть программы, отвечающая за взаимодействие с пользователем через консоль. В ней происходит инициализация объектов и вызов функций работы с ними, заполнение стандартного контейнера вектор введенными объектами и печать его содержимого;
2. point.h: описание класса Point точек A(a1, a2);
3. point.cpp: реализация класса Point;
4. figure.h: описание абстрактного класса-родителя Figure;
5. figure.cpp: реализация класса Figure;
6. triangle.h: описание класса Triangle треугольников, заданных по трем точкам, наследника Figure;
7. triangle.cpp: реализация класса Triangle;
8. rectangle.h: описание класса Rectangle, наследника Figure (объект – прямоугольник, заданный по точкам);
9. rectangle.cpp: реализация класса Rectangle;
10. square.h: описание класса Square, наследника Figure (объект – квадрат, заданный по точкам);
11. square.cpp: реализация класса Square.

Также используется файл CMakeLists.txt с конфигурацией CMake для автоматизации сборки программы.

Дневник отладки

Проблем не возникло

Вывод

В данной лабораторной работе я продолжила знакомиться с основами ООП в языке C++. Реализовала абстрактный класс и виртуальную функцию. Использовала три основных принципа ООП – наследование, инкапсуляция и полиморфизм.

Базовым классом в наследовании стал класс Figure, от которого дочерние Triangle, Rectangle и Square унаследовали общие имена методов VertexesNumber,

Area, Print со своей реализацией для каждого класса, что возможно в C++ благодаря полиморфизму.

В данной работе я столкнулась с тем, как может быть реализовано объектно-ориентированное программирование в C++. Это было полезно для расширения кругозора.

Исходный код

main.cpp:

```
// OOP, Lab 1, variant 9, Diana Kolpakova
// Figure, Triangle, Square, Rectangle

#include <iostream>
#include <vector>

#include "figure.h"
#include "triangle.h"
#include "square.h"
#include "rectangle.h"

using namespace std;

int main()
{
    cout << "oop_exercise_1 (c) Diana Kolpakova" << endl;
    vector<Figure*> figures;

    cout << "Enter triangle points:";
    Triangle triangle1(cin);
    triangle1.Print(cout);
    cout << "Number of points: " << triangle1.VertexesNumber() << endl;
    cout << "Area:" << triangle1.Area() << endl;
    figures.push_back(&triangle1);

    cout << "Enter rectangle points:";
    Rectangle rectangle1(cin);
    rectangle1.Print(cout);
    cout << "Number of points: " << rectangle1.VertexesNumber() << endl;
    cout << "Area:" << rectangle1.Area() << endl;
    figures.push_back(&rectangle1);

    cout << "Enter square points:";
    Square square1(cin);
    square1.Print(cout);
    cout << "Number of points: " << square1.VertexesNumber() << endl;
    cout << "Area:" << square1.Area() << endl;
    figures.push_back(&square1);

    cout << "Enter rectangle points:";
    Rectangle rectangle2(cin);
```

```
    rectangle2.Print(cout);
    cout << "Number of points: " << rectangle2.VertexesNumber() << endl;
    cout << "Area:" << rectangle2.Area() << endl;
    figures.push_back(&rectangle2);

    cout << "Figures in container:" << endl;
    for (int i = 0; i < figures.size(); i++)
    {
        figures[i]->Print(cout);
    }
}
```

point.h:

```
#pragma once
#include <iostream>

using namespace std;

class Point
{
private:
   double x;
   double y;

public:
   Point();
   Point(double x, double y);

   static double Distance (const Point& point1, const Point& point2);

   friend istream& operator>>(istream& is, Point& point);
   friend ostream& operator<<(ostream& os, Point& point);
};
```

point.cpp:

```
#include <cmath>
#include "point.h"

using namespace std;

Point::Point()
{
   this->x = 0.0;
   this->y = 0.0;
}

Point::Point(double x, double y)
{
   this->x = x;
   this->y = y;
}

double Point::Distance(const Point& point1, const Point& point2)
{
   double dx = point1.x - point2.y;
```

```
    double dy = point1.y - point2.y;
    double distance = sqrt(dx * dx + dy * dy);
    return distance;
}

istream& operator>>(istream& is, Point& point)
{
    is >> point.x >> point.y;
    return is;
}

ostream& operator<<(ostream& os, Point& point)
{
    os << "(" << point.x << ", " << point.y << ")";
    return os;
}
```

figure.h:

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

class Figure
{
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print(ostream& os) = 0;
};
```

triangle.h:

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

class Triangle : public Figure
{
private:
    Point point1;
    Point point2;
    Point point3;

public:
    Triangle();
    Triangle(Point point1, Point point2, Point point3);
    Triangle(const Triangle& other);
    Triangle(istream& is);

    virtual size_t VertexesNumber() override;
    virtual double Area() override;
    virtual void Print(ostream& os) override;
};
```

triangle.cpp:

```cpp
#include "triangle.h"

using namespace std;

Triangle::Triangle()
{
    this->point1 = Point();
    this->point2 = Point();
    this->point3 = Point();
}

Triangle::Triangle(Point point1, Point point2, Point point3)
{
    this->point1 = point1;
    this->point2 = point2;
    this->point3 = point3;
}

Triangle::Triangle(const Triangle& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
    this->point3 = other.point3;
}

Triangle::Triangle(istream& is)
{
    is >> point1 >> point2 >> point3;
}

size_t Triangle::VertexesNumber()
{
    return 3;
}

double Triangle::Area()
{
    double length12 = Point::Distance(point1, point2);
    double length23 = Point::Distance(point2, point3);
    double length31 = Point::Distance(point3, point1);
    double semiPerimeter = (length12 + length23 + length31) / 2.0;
    return sqrt(semiPerimeter * (semiPerimeter - length12) * (semiPerimeter -
length23) * (semiPerimeter - length31));
}

void Triangle::Print(ostream& os)
{
    os << "Triangle: " << point1 << ", " << point2 << ", " << point3 << endl;
}
```

square.h:

```cpp
#pragma once
#include "figure.h"

class Square : public Figure
{
private:
```

```
    Point point1;
    Point point2;
    Point point3;
    Point point4;

public:
    Square();
    Square(Point point1, Point point2, Point point3, Point point4);
    Square(const Square& other);
    Square(istream& is);

    virtual size_t VertexesNumber() override;
    virtual double Area() override;
    virtual void Print(ostream& os) override;
};
```

square.cpp:

```
#include "square.h"

Square::Square()
{
    this->point1 = Point();
    this->point2 = Point();
    this->point3 = Point();
    this->point4 = Point();
}

Square::Square(Point point1, Point point2, Point point3, Point point4)
{
    this->point1 = point1;
    this->point2 = point2;
    this->point3 = point3;
    this->point4 = point4;
}

Square::Square(const Square& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
    this->point3 = other.point3;
    this->point4 = other.point4;
}

Square::Square(istream& is)
{
    is >> point1 >> point2 >> point3 >> point4;
}

size_t Square::VertexesNumber()
{
    return 4;
}

double Square::Area()
{
    double size = Point::Distance(point1, point2);
```

```
    return size * size;
}

void Square::Print(ostream& os)
{
    os << "Square: " << point1 << ", " << point2 << ", " << point3 << ", " <<
point4 << endl;
}
```

rectangle.h:

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

class Rectangle : public Figure
{
private:
   Point point1;
   Point point2;
   Point point3;
   Point point4;

public:
   Rectangle();
   Rectangle(Point point1, Point point2, Point point3, Point point4);
   Rectangle(const Rectangle& other);
   Rectangle(istream& is);

   virtual size_t VertexesNumber() override;
   virtual double Area() override;
   virtual void Print(ostream& os) override;
};
```

rectangle.cpp:

```
#include "rectangle.h"

Rectangle::Rectangle()
{
    this->point1 = Point();
    this->point2 = Point();
    this->point3 = Point();
    this->point4 = Point();
}

Rectangle::Rectangle(Point point1, Point point2, Point point3, Point point4)
{
    this->point1 = point1;
    this->point2 = point2;
    this->point3 = point3;
    this->point4 = point4;
}

Rectangle::Rectangle(const Rectangle& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
```

```
    this->point3 = other.point3;
    this->point4 = other.point4;
}

Rectangle::Rectangle(istream& is)
{
    is >> point1 >> point2 >> point3 >> point4;
}

size_t Rectangle::VertexesNumber()
{
    return 4;
}

double Rectangle::Area()
{
    double width = Point::Distance(point1, point2);
    double height = Point::Distance(point2, point3);
    return width * height;
}

void Rectangle::Print(ostream& os)
{
    os << "Rectangle: " << point1 << ", " << point2 << ", " << point3 << ", "
<< point4 << endl;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.21)
project(oop_exercise_1)

set(CMAKE_CXX_STANDARD 14)

include_directories(.)

add_executable(oop_exercise_1
        figure.cpp
        figure.h
        main.cpp
        point.cpp
        point.h
        rectangle.cpp
        rectangle.h
        square.cpp
        square.h
        triangle.cpp
        triangle.h)
```