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: Примаченко Александр Александрович, группа М80-208Б-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 №18:

- Фигура 1: Квадрат (Square)
- Фигура 2: Прямоугольник (Rectangle)
- Фигура 3: Трапеция (Trapezoid)

Описание программы:

Исходный код разделён на 10 файлов:

- `point.h` – описание класса точки
- `point.cpp` – реализация класса точки
- `figure.h` – описание класса фигуры
- `square.h` – описание класса квадрат
- `square.cpp` – реализация класса квадрат
- `rectangle.h` – описание класса прямоугольник
- `rectangle.cpp` – реализация класса прямоугольник
- `trapezoid.h` – описание класса трапеция
- `trapezoid.cpp` – реализация класса трапеция
- `main.cpp` – основная программа

Дневник отладки:

Программа в отладке не нуждалась, необходимый функционал был реализован довольно быстро и безошибочно.

Вывод:

При выполнении лабораторной работы я применил на практике основные столпы ООП, а именно инкапсуляцию - реализовывая классы и определял доступ к данным, наследование – описывая классы фигур на основе figure.h и полиморфизм – переопределяя методы классов фигур. Как результат работы была написана программа с применением основных столпов и использовании различных конструкторов, перегрузки методов и дружественных функций для работы с геометрическими фигурами, а именно определении площади, количество углов и вывод их в стандартный поток вывода, что дало мне более полное понимании принципов и преимуществ с использованием объектно-ориентированного подхода в программировании.

Исходный код:

Figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H

#include <cstddef>
#include <iostream>
#include <vector>
#include "point.h"

class Figure {
public:
   virtual void Print(std::ostream& os) = 0;

   virtual double Area() = 0;

   virtual size_t VertexesNumbers() = 0;

   virtual  ~Figure() = default;
};

#endif //FIGURE_H
```

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);

    friend class Square;
    friend class Rectangle;
    friend class Trapezoid;

private:
    double x_;
    double y_;
};

#endif //POINT_H
```

Point.cpp:

```
#include "point.h"

#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;
}
```

Main.cpp:

```
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"

using namespace std;

int main () {
    Square Sqr1;
    Point a(0, 0);
    Point b(0, 1);
    Point c(1, 1);
    Point d(1, 0);
    vector<Point> v{a, b, c, d};
    Square Sqr2(v);
    Square Sqr3(cin);
    Square Sqr4(Sqr3);
    Sqr1.Print(cout);
    cout << Sqr2.VertexesNumbers() << endl;
    cout << "Square area is " << Sqr2.Area() << endl;
    Figure *f = new Square(Sqr2);
    delete f;

    Rectangle Rec1;
    Rectangle Rec2(v);
    Rectangle Rec3(cin);
    Rectangle Rec4(Rec3);
    Rec1.Print(cout);
    cout << Rec1.VertexesNumbers() << endl;
    cout << "Rectangle area is " << Rec2.Area() << endl;
    f = new Rectangle(Rec2);
    delete f;

    Trapezoid Trp1;
    Point atr(0,0), btr(1, 1), ctr(2, 1), dtr(3, 0);
    vector<Point> tr{atr, btr, ctr, dtr};
    Trapezoid Trp2(tr);
    Trapezoid Trp3(cin);
    Trapezoid Trp4(Trp1);
    cout << "Trapezoid area is " << Trp2.Area() << endl;
    Trp3.Print(cout);
    cout << Trp4.VertexesNumbers() << endl;
    f = new Trapezoid(Trp4);
    delete f;
    return 0;
}
```

Rectangle.h:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"
```

```
using namespace std;

class Rectangle : public Figure {
public:
    Rectangle();

    Rectangle(vector<Point> v);

    Rectangle(istream &is);

    Rectangle(const Rectangle &other);

    void Print(ostream &os);

    size_t VertexesNumbers();

    double Area() override;

    virtual ~Rectangle();

private:
    Point a, b, c, d;
};


#endif
```

Rectangle.cpp:

```
#include "rectangle.h"

void Rectangle::Print(ostream &os) {
    cout << "Rectangle: " << a << ", " << b << ", " << c << ", " << d << endl;
}

double Rectangle::Area(){
    return (abs(a.x_ - b.x_) * abs(a.y_ - b.y_));
}

Rectangle::Rectangle() : a(), b() {
    cout << "Default rectangle is created" << endl;
}

Rectangle::Rectangle(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
    cout << "Rectangle with vertices " << a << ", " << b << ", " << c << ", " << d << " was created"
    << endl; // создание с помощью вектора координат вершин
}

Rectangle::Rectangle(istream &is) {
    cout << "Enter lower left coordinate" << endl;
    cin >> a;
    cout << "Enter upper left coordinate" << endl;
    cin >> b;
    cout << "Enter upper right coordinate" << endl;
```

```
    cin >> c;
    cout << "Enter lower right coordinate" << endl;
    cin >> d;
    cout << "Rectangle was created via stream" << endl;
}

Rectangle::Rectangle(const Rectangle &other) : a(other.a), b(other.b), c(other.c), d(other.d) {
    cout << "Made copy of rectangle" << endl;
}

Rectangle::~Rectangle() {
    cout << "Rectangle was deleted" << endl;
}

size_t Rectangle::VertexesNumbers() {
    return 4;
}
```

Square.h:

```
#ifndef SQUARE_H
#define SQUARE_H

#include "figure.h"

using namespace std;

class Square : public Figure {
public:
    Square();

    Square(vector<Point> v);

    Square(istream &is);

    Square(const Square &other);

    void Print(ostream &os);

    size_t VertexesNumbers();

    double Area() override;

    virtual ~Square();

private:
    Point a, b, c, d;
};


#endif //SQUARE_H
```

Square.cpp:

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

void Square::Print(ostream& os) {
   cout << "Trapezoid: " << a << ", " << b << ", " << c << ", " << d << endl;
}

double Square::Area() {
   double k = (a.y_ - d.y_) / (a.x_ - d.x_);
   double m = a.y_- k * a.x_;
   double h = abs(b.y_- k * b.x_ - m) / sqrt(1 + k * k);
   return 0.5 * (a.dist(d) + b.dist(c)) * h;
}

Square::Square() : a(), b(), c(), d() {
   cout << "Default square was created" << endl;
}

Square::Square(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
   cout << "Square with vertices " << a << ", " << b << ", " << c << ", " << d << " was created" << endl;
}

Square::Square(istream &is) {
   cout << "Enter lower left coordinate" << endl;
   cin >> a;
   cout << "Enter upper left coordinate" << endl;
   cin >> b;
   cout << "Enter upper right coordinate" << endl;
   cin >> c;
   cout << "Enter lower right coordinate" << endl;
   cin >> d;
   cout << "Square was created" << endl;
}

Square::Square(const Square &other) : a(other.a), b(other.b), c(other.c), d(other.d) {
   cout << "Made copy of square" << endl;
}

Square::~Square() {
   cout << "Square was deleted" << endl;
}

size_t Square::VertexesNumbers() {
   return 4;
}
```

Trapezoid.h:

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

#include "figure.h"

using namespace std;
```

```
class Trapezoid : public Figure {
public:
    Trapezoid();

    Trapezoid(vector<Point> v);

    Trapezoid(istream &is);

    Trapezoid(const Trapezoid &other);

    void Print(ostream &os);

    size_t VertexesNumbers();

    double Area() override;

    virtual ~Trapezoid();

private:
    Point a, b, c, d;
};

#endif //TRAPEZOID_H
```

Trapezoid.cpp:

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

void Trapezoid::Print(ostream& os) {
    cout << "Trapezoid: " << a << ", " << b << ", " << c << ", " << d << endl;
}

double Trapezoid::Area() {
    double k = (a.y_ - d.y_) / (a.x_ - d.x_);
    double m = a.y_- k * a.x_;
    double h = abs(b.y_- k * b.x_ - m) / sqrt(1 + k * k);
    return 0.5 * (a.dist(d) + b.dist(c)) * h;
}

Trapezoid::Trapezoid() : a(), b(), c(), d() {
    cout << "Default trapezoid was created" << endl;
}

Trapezoid::Trapezoid(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
    cout << "Trapezoid with vertices " << a << ", " << b << ", " << c << ", " << d << " was created" << endl;
}

Trapezoid::Trapezoid(istream &is) {
    cout << "Enter lower left coordinate" << endl;
    cin >> a;
    cout << "Enter upper left coordinate" << endl;
    cin >> b;
    cout << "Enter upper right coordinate" << endl;
    cin >> c;
    cout << "Enter lower right coordinate" << endl;
```

```
    cin >> d;
    cout << "Trapezoid was created" << endl;
}

Trapezoid::Trapezoid(const Trapezoid &other) : a(other.a), b(other.b), c(other.c), d(other.d) {
    cout << "Made copy of trapezoid" << endl;
}

Trapezoid::~Trapezoid() {
    cout << "Trapezoid was deleted" << endl;
}

size_t Trapezoid::VertexesNumbers() {
    return 4;
}
```

Результат работы:

```
C:\Users\SashaPaladin\CLionProjects\OOP\lab1\cmake-build-debug\lab1.exe
Default square was created
Square with vertices (0, 0), (0, 1), (1, 1), (1, 0) was created
Enter lower left coordinate

0 1
Enter upper left coordinate
1 1
Enter upper right coordinate
2 2
Enter lower right coordinate
2 1
Square was created
Made copy of square
Trapezoid: (0, 0), (0, 0), (0, 0), (0, 0)
4
Square area is 1
Made copy of square
Square was deleted
Default rectangle is created
Rectangle with vertices (0, 0), (0, 1), (1, 1), (1, 0) was created
Enter lower left coordinate
0 0
Enter upper left coordinate
0 1
Enter upper right coordinate
1 1
Enter lower right coordinate
1 2
Rectangle was created via stream
Made copy of rectangle
Rectangle: (0, 0), (0, 0), (0, 0), (0, 0)
4
Rectangle area is 0
Made copy of rectangle
Rectangle was deleted
Default trapezoid was created
Trapezoid with vertices (0, 0), (1, 1), (2, 1), (3, 0) was created
Enter lower left coordinate
0 0
Enter upper left coordinate
1 1
```

```
Enter upper right coordinate
2 2
Enter lower right coordinate
3 2
Trapezoid was created
Made copy of trapezoid
Trapezoid area is 2
Trapezoid: (0, 0), (1, 1), (2, 2), (3, 2)
4
Made copy of trapezoid
Trapezoid was deleted
Trapezoid was deleted
Trapezoid was deleted
Trapezoid was deleted
Trapezoid was deleted
Rectangle was deleted
Rectangle was deleted
Rectangle was deleted
Rectangle was deleted
Square was deleted
Square was deleted
Square was deleted
Square was deleted

Process finished with exit code 0
```