МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Макаров Глеб Александрович, группа М8О-207Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 25: Треугольник, Квадрат, Прямоугольник. Необходимо спроектировать и запрограммировать на языке C++ классы трех фигур, согласно варианту задания. Классы должны удовлетворять следующим правилам:

1. 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).
2. Иметь общий родительский класс Figure;
3. 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 1.00.0 1.0"
4. 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 вершинв 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.

Описание программы

Исходный код лежит в файлах:

1. main.cpp: основная программа
2. figure.h: описание абстрактного класса фигур
3. point.h: описание класса точки
4. pentagon.h: описание класса пятиугольника, наследующегося от figure
5. octagon.h: описание класса восьмиугольника, наследующегося от figure
6. hexagon.h: описание класса шестиугольника, наследующегося от rectangle
7. point.cpp: реализация класса точки
8. pentagon.cpp: реализация класса пятиугольника, наследующегося от figure
9. octagon.cpp: реализация класса восьмиугольника, наследующегося от figure
10. hexagon.cpp: реализация класса шестиугольника, наследующегося от rectangle

Дневник отладки
Ошибок не замечено

Недочёты

Недочетов не обнаружено

Выводы

В ходе лабораторной работы удалось поработать с парадигмой объектно-ориентированного программирования - наследование, полиморфизм, инкапсуляция.

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <cstring>

class Figure {
public:
   virtual void Print() = 0;
   virtual double Area() = 0;
   virtual size_t VertexesNumber() = 0;
   virtual ~Figure() {};
};


#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 get_x();
  double get_y();
  double dist(Point& other);

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

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::get_x() { return x_; }
double Point::get_y() { return 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 <iostream>
#include "pentagon.h"
#include "hexagon.h"
#include "octagon.h"
#include "point.h"

using namespace std;

int main()
{
    Point point1;
    std::cin >> point1;
    std::cout << point1;

    Pentagon a(std::cin);
    a.Print();
    std::cout << "square: " << a.Area() << std::endl;

    Octagon b(std::cin);
    b.Print();
    std::cout << "square: " << b.Area() << std::endl;

    Hexagon c(std::cin);
    c.Print();
    std::cout << "square: " << c.Area() << std::endl;

    return 0;
```

pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H

#include "figure.h"
#include "point.h"
#include <iostream>


class Pentagon : public Figure {
public:

    Pentagon(std::istream &is);
    virtual ~Pentagon();
    size_t VertexesNumber();
    void Print();
    double Area();

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

#endif
```

pentagon.cpp

```
#include "pentagon.h"

#include <cmath>

Pentagon::Pentagon(std::istream &is) {
  is >> a;
  is >> b;
  is >> c;
  is >> d;
  is >> e;
}


void Pentagon::Print() {
  std::cout << "Pentagon: ";
  std::cout << a << " ";
  std::cout << b << " ";
  std::cout << c << " ";
  std::cout << d << " ";
  std::cout << e << "\n" << std::endl;
}

double Pentagon::Area() {
    double p = fabs(a.get_x()*b.get_y()-b.get_x()*a.get_y()+b.get_x()*c.get_y()-
c.get_x()*b.get_x()+c.get_x()*d.get_y()-d.get_x()*c.get_y()+d.get_x()*e.get_y()-
e.get_x()*d.get_y()+e.get_x()*a.get_y()-a.get_x()*e.get_y())/2;
    return p;
}

size_t Pentagon::VertexesNumber() {
  return 5;
}

Pentagon::~Pentagon() {
    std::cout << "Pentagon deleted" << std::endl;
}
```

hexagon.h

```
#ifndef HEXAGON_H
#define HEXAGON_H

#include "figure.h"
#include "point.h"
#include <iostream>


class Hexagon : public Figure {
public:

    Hexagon(std::istream &is);


    virtual ~Hexagon();
    size_t VertexesNumber();
    void Print();
    double Area();

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

#endif
```

hexagon.cpp

```
#include "hexagon.h"

#include <cmath>

Hexagon::Hexagon(std::istream &is) {
  std::cout << "Enter data:" <<  std::endl;
  is >> a;
  is >> b;
  is >> c;
  is >> d;
  is >> e;
  is >> f;

}

void Hexagon::Print() {
    std::cout << "Hexagon: ";
    std::cout << a << ", ";
    std::cout << b << ", ";
    std::cout << c << ", ";
    std::cout << d << ", ";
    std::cout << e << ", ";
    std::cout << f << "\n" << std::endl;
}

double Hexagon::Area() {
    double p = fabs(a.get_x()*b.get_y()-b.get_x()*a.get_y()+b.get_x()*c.get_y()-
c.get_x()*b.get_x()+c.get_x()*d.get_y()-d.get_x()*c.get_y()+d.get_x()*e.get_y()-
e.get_x()*d.get_y()+e.get_x()*f.get_y()-f.get_x()*e.get_y()+f.get_x()*a.get_y()-
a.get_x()*f.get_y())/2;
    return p;
}

size_t Hexagon::VertexesNumber() {
  return 6;
}

Hexagon::~Hexagon() {
    std::cout << "Hexagon deleted" << std::endl;
}
```

octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include "figure.h"
#include "point.h"
#include <iostream>


class Octagon : public Figure {
public:

    Octagon(std::istream &is);


    virtual ~Octagon();
    size_t VertexesNumber();
    void Print();
    double Area();

private:
    Point a, b, c, d, e, f, g, h;
};

#endif
```

octagon.cpp

```
#include "octagon.h"

#include <cmath>


Octagon::Octagon(std::is
tream &is) {

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

  is >> a;

  is >> b;

  is >> c;

  is >> d;

  is >> e;

  is >> f;

  is >> g;

  is >> h;

}

void Octagon::Print() {

   std::cout << "Octagon:
";

   std::cout << a << ", ";

   std::cout << b << ", ";

   std::cout << c << ", ";

   std::cout << d << ", ";

   std::cout << e << ", ";

   std::cout << f << ", ";

   std::cout << g << ", ";

   std::cout << h << "\n"
<< std::endl;

}
```

```
double Octagon::Area() {
    double p =
fabs(a.get_x()*b.get_y()-
b.get_x()*a.get_y()+b.get
_x()*c.get_y()-
c.get_x()*b.get_x()+c.get
_x()*d.get_y()-
d.get_x()*c.get_y()+d.get
_x()*e.get_y()-
e.get_x()*d.get_y()+e.get
_x()*f.get_y()-
f.get_x()*e.get_y()+f.get_
x()*g.get_y()-
g.get_x()*f.get_y()+g.get_
x()*h.get_y()-
h.get_x()*g.get_y()+h.get
_x()*a.get_y()-
a.get_x()*h.get_y())/2;
    return p;
}


size_t
Octagon::VertexesNumb
er() {
  return 8;
}


Octagon::~Octagon() {
    std::cout << "Octagon
deleted" << std::endl;
}
```