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: *Варламова Анна Борисовна*

Группа: *М8О-207Б-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 №8:

- Фигура 1: 8-угольник (Octagon)
- Фигура 2: Треугольник (Triangle)
- Фигура 3: Квадрат (Square)

Описание программы:

Исходный код разделён на 10 файлов:

- `point.h` – описание класса точки
- `point.cpp` – реализация класса точки
- `figure.h` – описание класса фигуры
- `octagon.h` – описание класса 8-угольника (наследуется от фигуры)
- `octagon.cpp` – реализация класса 8-угольника
- `square.h` – описание класса квадрата (наследуется от фигуры)
- `square.cpp` – реализация класса квадрата
- `triangle.h` – описание класса треугольника (наследуется от фигуры)
- `triangle.cpp` – реализация класса треугольника
- `main.cpp` – основная программа

После запуска исполяемого файла можно будет ввести координаты точек для вершин и узнать из них площадь треугольника, квадрата, восьмиугольника.

Дневник отладки:

Сначала возникла проблема с коллизией имён: я перепутала названия точек в конструкторе, но исправила по примеру класса точки.

Тестирование:

```
Enter triangle (points):
-3 0 -1 4 3 0
Triangle: (-3, 0) (-1, 4) (3, 0)
The number of vertexes: 3
Area: 12

Enter square (points):
1 -10 1 -6 5 -6 5 -10
Square: (1, -10) (1, -6) (5, -10) (5, -6)
The number of vertexes: 4
Area: 16

Enter octagon (points):
-2 1 -2 3 1 6 4 5 5 2 4 -1 2 -2 -1 -1
Octagon: (-2, 1) (-2, 3) (1, 6) (4, 5) (5, 2) (4, -1) (2, -2) (-1, -1)
The number of vertexes: 8
Area: 40.5

Enter triangle (points):
-3.4 0 -1 4.23 3 0
Triangle: (-3.4, 0) (-1, 4.23) (3, 0)
The number of vertexes: 3
Area: 13.536

Enter square (points):
-2 -2 -2 -1 -1 -1 -1 -2
Square: (-2, -2) (-2, -1) (-1, -2) (-1, -1)
The number of vertexes: 4
Area: 1
```

```
Enter octagon (points):
1 1 2 2 4 3 5 3 6 3 7 2 8 1 5 0
Octagon: (1, 1) (2, 2) (4, 3) (5, 3) (6, 3) (7, 2) (8, 1) (5, 0)
The number of vertexes: 8
Area: 13
```

Вывод:

При выполнении работы я на практике познакомилась с базовыми принципами ООП: инкапсуляция, наследование, полиморфизм. Наследование: описание основных функций содержится в описании класса *Figure* и передаётся остальным классам; инкапсуляция: при работе с точкой не приходится обращаться к разным объектам – координатам точки, всё решается через методы класса *Point*. Полиморфизм: в классах *Octagon*, *Square*, *Triangle* есть методы подсчёта вершин, печати фигуры, расчёта площади, но все они выполняют разные функции в зависимости от класса. Кроме написания конструкторов, были написаны деструкторы и операторы копирования, а также перегрузка операторов ввода/вывода. Полученные навыки являются фундаментом для дальнейших лабораторных ООП.

Исходный код:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"

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

octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <iostream>

#include "figure.h"

class Octagon : public Figure {
public:
  Octagon();
  Octagon(Point t_1, Point t_2, Point t_3, Point t_4,
          Point t_5, Point t_6, Point t_7, Point t_8);
  Octagon(std::istream &is);
  Octagon(const Octagon& other);

  size_t VertexesNumber();
  double Area();
  void Print(std::ostream& os);

  virtual ~Octagon();

  friend std::istream& operator>>(std::istream& is, Octagon& o);
  friend std::ostream& operator<<(std::ostream& os, Octagon& o);

private:
  Point t1;
  Point t2;
  Point t3;
  Point t4;
  Point t5;
  Point t6;
  Point t7;
  Point t8;
};

#endif // OCTAGON_H
```

octagon.cpp

```
#include "octagon.h"

#include <iostream>
#include <cmath>

Octagon::Octagon()
  : t1(0.0, 0.0), t2(0.0, 0.0), t3(0.0, 0.0), t4(0.0, 0.0),
  t5(0.0, 0.0), t6(0.0, 0.0), t7(0.0, 0.0), t8(0.0, 0.0) {}

```

```
10. Octagon::Octagon(Point t_1, Point t_2, Point t_3, Point t_4,
11.                  Point t_5, Point t_6, Point t_7, Point t_8)
12.   : t1(t_1), t2(t_2), t3(t_3), t4(t_4),
13.   t5(t_5), t6(t_6), t7(t_7), t8(t_8) {}
14.
15. Octagon::Octagon(std::istream &is) {
16.   is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;
17. }
18.
19. Octagon::Octagon(const Octagon& other)
20.   : Octagon(other.t1, other.t2, other.t3, other.t4,
21.           other.t5, other.t6, other.t7, other.t8) {}
22.
23. std::istream& operator>>(std::istream& is, Octagon& o) {
24.   is >> o.t1 >> o.t2 >> o.t3 >> o.t4 >> o.t5 >> o.t6 >> o.t7 >> o.t8;
25.   return is;
26. }
27.
28. std::ostream& operator<<(std::ostream& os, Octagon& o) {
29.   os << "Octagon: " << o.t1 << " " << o.t2 << " " << o.t3 << " " << o.t4
30.           << " " << o.t5 << " " << o.t6 << " " << o.t7 << " " << o.t8;
31.   return os;
32. }
33.
34. size_t Octagon::VertexesNumber()
35. {
36.   return (size_t)8;
37. }
38.
39. double Heron(Point A, Point B, Point C) {
40.   double AB = A.dist(B);
41.   double BC = B.dist(C);
42.   double AC = A.dist(C);
43.   double p = (AB + BC + AC) / 2;
44.   return sqrt(p * (p - AB) * (p - BC) * (p - AC));
45. }
46.
47. double Octagon::Area() {
48.   double area1 = Heron(t1, t2, t3);
49.   double area2 = Heron(t1, t4, t3);
50.   double area3 = Heron(t1, t4, t5);
51.   double area4 = Heron(t1, t5, t6);
52.   double area5 = Heron(t1, t6, t7);
53.   double area6 = Heron(t1, t7, t8);
54.   return area1 + area2 + area3 + area4 + area5 + area6;
55. }
56.
57. void Octagon::Print(std::ostream& os)
58. {
59.   std::cout << "Octagon: " << t1 << " " << t2 << " " << t3 << " " << t4
```

```
60.             << " " << t5 << " " << t6 << " " << t7 << " " << t8 << "\n";
61. }
62.
63. Octagon::~Octagon() {}
```

square.h

```
1.  #ifndef SQUARE_H
2.  #define SQUARE_H
3.
4.  #include <iostream>
5.
6.  #include "figure.h"
7.
8.  class Square : public Figure {
9.  public:
10.   Square();
11.   Square(Point a_, Point b_, Point c_, Point d_);
12.   Square(std::istream &is);
13.   Square(const Square& other);
14.
15.   size_t VertexesNumber();
16.   double Area();
17.   void Print(std::ostream& os);
18.
19.   virtual ~Square();
20.
21.   friend std::istream& operator>>(std::istream& is, Square& s);
22.   friend std::ostream& operator<<(std::ostream& os, Square& s);
23.
24. private:
25.   Point A;
26.   Point B;
27.   Point C;
28.   Point D;
29. };
```

square.cpp

```
1.  #include "square.h"
2.
3.  #include <iostream>
4.  #include <cmath>
5.
6.  Square::Square()
7.    : A(0.0, 0.0), B(0.0, 0.0), C(0.0, 0.0), D(0.0, 0.0) {}
8.
9.  Square::Square(Point a_, Point b_, Point c_, Point d_)
10.   : A(a_), B(b_), C(c_), D(d_) {}
11.
```

```
12. Square::Square(std::istream &is) {
13.   is >> A >> B >> C >> D;
14. }
15.
16. std::istream& operator>>(std::istream& is, Square& s) {
17.   is >> s.A >> s.B >> s.C >> s.D;
18.   return is;
19. }
20.
21. std::ostream& operator<<(std::ostream& os, Square& s) {
22.   os << "Square: " << s.A << " " << s.B << " "  << s.D << " " << s.C;
23.   return os;
24. }
25.
26. Square::Square(const Square& other)
27.   : Square(other.A, other.B, other.C, other.D) {}
28.
29. size_t Square::VertexesNumber()
30. {
31.   return (size_t)4;
32. }
33.
34. double Square::Area() {
35.   double side = A.dist(B);
36.   return side * side;
37. }
38.
39. void Square::Print(std::ostream& os)
40. {
41.   std::cout << "Square: " << A << " " << B << " "  << D << " " << C << "\n";
42. }
43.
44. Square::~Square() {}
```

triangle.h

```
1.  #ifndef TRIANGLE_H
2.  #define TRIANGLE_H
3.
4.  #include <iostream>
5.
6.  #include "figure.h"
7.
8.  class Triangle : public Figure {
9.  public:
10.   Triangle();
11.   Triangle(Point a_, Point b_, Point c_);
12.   Triangle(std::istream &is);
13.   Triangle(const Triangle& other);
14.
```

```
15.   size_t VertexesNumber();
16.   double Area();
17.   void Print(std::ostream& os);
18.
19.   virtual ~Triangle();
20.
21.   friend std::istream& operator>>(std::istream& is, Triangle& t);
22.   friend std::ostream& operator<<(std::ostream& os, Triangle& t);
23.
24. private:
25.   Point A;
26.   Point B;
27.   Point C;
28. };
29.
30. #endif // TRIANGLE_H
```

triangle.cpp

```
1.   #include "triangle.h"
2.
3.   #include <iostream>
4.   #include <cmath>
5.
6.   Triangle::Triangle()
7.     : A(0.0, 0.0), B(0.0, 0.0), C(0.0, 0.0) {}
8.
9.   Triangle::Triangle(Point a_, Point b_, Point c_)
10.   : A(a_), B(b_), C(c_) {}
11.
12. Triangle::Triangle(std::istream &is) {
13.   is >> A >> B >> C;
14. }
15.
16. Triangle::Triangle(const Triangle& other)
17.   : Triangle(other.A, other.B, other.C) {}
18.
19. std::istream& operator>>(std::istream& is, Triangle& t) {
20.   is >> t.A >> t.B >> t.C;
21.   return is;
22. }
23.
24. std::ostream& operator<<(std::ostream& os, Triangle& t) {
25.   os << "Triangle: " << t.A << " " << t.B << " " << t.C;
26.   return os;
27. }
28.
29. size_t Triangle::VertexesNumber()
30. {
31.   return (size_t)3;
```

```
32. }
33.
34. double Triangle::Area() {
35.   double AB = A.dist(B);
36.   double BC = B.dist(C);
37.   double AC = A.dist(C);
38.   double p = (AB + BC + AC) / 2;
39.   return sqrt(p * (p - AB) * (p - BC) * (p - AC));
40. }
41.
42. void Triangle::Print(std::ostream& os)
43. {
44.   std::cout << "Triangle: " << A << " " << B << " " << C << "\n";
45. }
46.
47. Triangle::~Triangle() {}
```