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

ЛАБОРАТОРНАЯ РАБОТА №1

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

Студент: *Сафонникова Анна Романовна, группа М8О-207Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Условие

Задание: Вариант 22: 5-угольник, 6-угольник, 8-угольник. 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 1.0 0.0 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 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.

Описание программы

Исходный код лежит в 10 файлах:

1. src/main.cpp: основная программа, взаимодействие с пользователем посредством комманд из меню
2. include/figure.h: описание абстрактного класса фигур
3. include/point.h: описание класса точки
4. include/pentagon.h: описание класса 5-угольника, наследующегося от figures
5. include/hexagon.h: описание класса 6-угольника, наследующегося от figures
6. include/octagon.h: описание класса 8-угольника, наследующегося от figures
7. include/point.cpp: реализация класса точки
8. include/pentagon.cpp: реализация класса треугольника, наследующегося от figures
9. include/hexagon.cpp: реализация класса прямоугольника, наследующегося от figures
10. include/octagon.cpp: реализация класса квадрата, наследующегося от figures

Дневник отладки

Ошибка:
Невозможно было создать объект, подобный данному
Исправление:
Прописала конструктор копирования во всех 3х фигурах

Ошибка:
Ошибка пересечения namespace'ов, т е компилятор не понял, к какому конкретно классу описала функцию
Исправление:
Добавила перед одной из функций Octagon::

Недочёты

Недочётов не заметила.

Вывод

В данной лабораторной работе мы работали с таким понятием, как парадигма ООП (наследование, полиморфизм, инкапсуляция, абстракция). В начале было сложно, так как нужно сначала понять, зачем мы вообще делаем именно таким способом всё, перестроиться на другой язык, а потом поэтапно красиво написать. Эти навыки пригодятся мне при реализации каких-либо проектов, где намного проще будет применить ООП, нежели другой вид программирования.

Исходный код

figure.h

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

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

```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

```cpp
#include "pentagon.h"
#include "hexagon.h"
#include "octagon.h"

void print_menu() {
  std::cout << "Choose a geometric shape:" << std::endl;
  std::cout << "1 - pentagon" << std::endl;
  std::cout << "2 - hexagon" << std::endl;
  std::cout << "3 - octagon" << std::endl;
```

```
}

int main() {
  int c;
  print_menu();
  while (std::cin >> c) {
    switch (c) {
      case 1: {
        std::cout << "Enter pentagon points:";
        Pentagon pen(std::cin);
        pen.Print(std::cout);
        std::cout << "The number of vertexes: " << pen.VertexesNumber() << "\n";
        std::cout << "Area:" << pen.Area() << "\n";
        break;
      }
      case 2: {
        std::cout << "Enter hexagon points:";
        Hexagon hex(std::cin);
        hex.Print(std::cout);
        std::cout << "The number of vertexes: " << hex.VertexesNumber() << "\n";
        std::cout << "Area:" << hex.Area() << "\n";
        break;
      }
      case 3: {
        std::cout << "Enter octagon points:";
        Octagon oct(std::cin);
        oct.Print(std::cout);
        std::cout << "The number of vertexes: " << oct.VertexesNumber() << "\n";
        std::cout << "Area:" << oct.Area() << "\n";
        break;

      }

    }
  }

}
```

pentagon.h

```
#ifndef PENTAGON_H
#define PENTAGON_H
```

```
#include <iostream>

#include "figure.h"

class Pentagon : public Figure {

 private:
  Point t1;
  Point t2;
  Point t3;
  Point t4;
  Point t5;

 public:
  Pentagon();
  Pentagon(const Pentagon &pentagon);
  Pentagon(std::istream &is);
  size_t VertexesNumber();
  double Area();
  void Print(std::ostream &os);
  static double Heron(Point, Point, Point);
};

#endif // PENTAGON_H
```

pentagon.cpp

```
#include "pentagon.h"
#include "hexagon.h"

#include <iostream>
#include <cmath>

Pentagon::Pentagon()
    : t1(0, 0), t2(0, 0), t3(0, 0), t4(0, 0),
      t5(0, 0) {}

Pentagon::Pentagon(const Pentagon &pentagon) {
  this->t1 = pentagon.t1;
  this->t2 = pentagon.t2;
  this->t3 = pentagon.t3;
  this->t4 = pentagon.t4;
  this->t5 = pentagon.t5;
```

```
}

Pentagon::Pentagon(std::istream &is) {
  is >> t1 >> t2 >> t3 >> t4 >> t5;
}

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

double Pentagon::Heron(Point A, Point B, Point C) {
  double AB = A.dist(B);
  double BC = B.dist(C);
  double AC = A.dist(C);
  double p = (AB + BC + AC) / 2;
  return sqrt(p * (p - AB) * (p - BC) * (p - AC));
}

double Pentagon::Area() {
  double area1 = Heron(t1, t2, t3);
  double area2 = Heron(t1, t4, t3);
  double area3 = Heron(t1, t4, t5);
  return area1 + area2 + area3;
}

void Pentagon::Print(std::ostream &os) {
  std::cout << "Pentagon: " << t1 << " " << t2 << " " << t3 << " " << t4
            << " " << t5 << " " << "\n";
}
```

hexagon.h

```
#ifndef HEXAGON_H
#define HEXAGON_H

#include <iostream>

#include "figure.h"

class Hexagon : public Figure {

 private:
  Point t1;
```

```
  Point t2;
  Point t3;
  Point t4;
  Point t5;
  Point t6;

 public:
  Hexagon();
  Hexagon(const Hexagon &hexagon);
  Hexagon(std::istream &is);
  size_t VertexesNumber();
  double Area();
  void Print(std::ostream &os);
  static double Heron(Point, Point, Point);
};

#endif // HEXAGON_H
```

hexagon.cpp

```
#include "hexagon.h"

#include <iostream>
#include <cmath>

Hexagon::Hexagon()
    : t1(0, 0), t2(0, 0), t3(0, 0), t4(0, 0),
      t5(0, 0), t6(0, 0) {}

Hexagon::Hexagon(const Hexagon &hexagon) {
  this->t1 = hexagon.t1;
  this->t2 = hexagon.t2;
  this->t3 = hexagon.t3;
  this->t4 = hexagon.t4;
  this->t5 = hexagon.t5;
  this->t6 = hexagon.t6;
}
Hexagon::Hexagon(std::istream &is) {
  is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6;
}

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

```
}

double Hexagon::Heron(Point A, Point B, Point C) {
  double AB = A.dist(B);
  double BC = B.dist(C);
  double AC = A.dist(C);
  double p = (AB + BC + AC) / 2;
  return sqrt(p * (p - AB) * (p - BC) * (p - AC));
}

double Hexagon::Area() {
  double area1 = Heron(t1, t2, t3);
  double area2 = Heron(t1, t4, t3);
  double area3 = Heron(t1, t4, t5);
  double area4 = Heron(t1, t5, t6);
  return area1 + area2 + area3 + area4;
}

void Hexagon::Print(std::ostream &os) {
  std::cout << "Hexagon: " << t1 << " " << t2 << " " << t3 << " " << t4
            << " " << t5 << " " << t6 << "\n";
}

//Hexagon::~Hexagon() {}

#endif //MAI_OOP_TRIANGLE_H
```

octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <iostream>

#include "figure.h"

class Octagon : public Figure {

private:
    Point t1;
    Point t2;
    Point t3;
    Point t4;
```

```
    Point t5;
    Point t6;
    Point t7;
    Point t8;

public:
    Octagon();
    Octagon(const Octagon& octagon);
    Octagon(std::istream &is);
    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);

};

#endif // OCTAGON_H
```

octagon.cpp

```
#include "octagon.h"

#include <iostream>
#include <cmath>

Octagon::Octagon()
    : t1(0, 0), t2(0, 0), t3(0, 0), t4(0, 0),
      t5(0, 0), t6(0, 0), t7(0, 0), t8(0, 0) {}

Octagon::Octagon(const Octagon &octagon) {
  this->t1 = octagon.t1;
  this->t2 = octagon.t2;
  this->t3 = octagon.t3;
  this->t4 = octagon.t4;
  this->t5 = octagon.t5;
  this->t6 = octagon.t6;
  this->t7 = octagon.t7;
  this->t8 = octagon.t8;
}

Octagon::Octagon(std::istream &is) {
  is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;
}
```

```
size_t Octagon::VertexesNumber() {
  return 8;
}

double Heron(Point A, Point B, Point C) {
  double AB = A.dist(B);
  double BC = B.dist(C);
  double AC = A.dist(C);
  double p = (AB + BC + AC) / 2;
  return sqrt(p * (p - AB) * (p - BC) * (p - AC));
}

double Octagon::Area() {
  double area1 = Heron(t1, t2, t3);
  double area2 = Heron(t1, t4, t3);
  double area3 = Heron(t1, t4, t5);
  double area4 = Heron(t1, t5, t6);
  double area5 = Heron(t1, t6, t7);
  double area6 = Heron(t1, t7, t8);
  return area1 + area2 + area3 + area4 + area5 + area6;
}

void Octagon::Print(std::ostream &os) {
  std::cout << "Octagon: " << t1 << " " << t2 << " " << t3 << " " << t4
            << " " << t5 << " " << t6 << " " << t7 << " " << t8 << "\n";
}

//Octagon::~Octagon() {}
```