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 *Абаровский Олег Александрович, группа М8О-207Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 1: 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 1.0 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 11 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/triangle.h: описание класса треугольника, наследующегося от figures
5. include/rectangle.h: описание класса прямоугольника, наследующегося от figures
6. include/square.h: описание класса квадрата, наследующегося от rectangle
7. include/point.cpp: реализация класса точки
8. include/triangle.cpp: реализация класса треугольника, наследующегося от figures
9. include/rectangle.cpp: реализация класса прямоугольника, наследующегося от figures
10. include/square.cpp: реализация класса квадрата, наследующегося от rectangle

Дневник отладки

12.11 Ошибка: неправильно реализован метод печати типа фигуры и координат её вершин в поток вывода os.
Решение: исправить формат вывода координат фигуры.

Недочёты

Недочётов нет

Выводы

Работа была полезной, в ходе её выполнения я получил представление об основных принципах объектно-ориентированного программирования. В моём варианте можно не только наследовать классы каждой фигуры от класса figures, но и унаследовать класс одной фигуры от класса другой(квадрат наследуется из прямоугольника)! Это хорошо иллюстрирует принцип наследуемости. Таким образом, эта работа оказалась для меня отличным введением в объектно-ориентированное программирование.

Исходный код

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

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

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

};

#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);
  Point &operator=(const 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;
}

Point &Point::operator=(const Point &p) {
  this->x_ = p.x_;
  this->y_ = p.y_;
  return *this;
}
```

main.cpp

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

int main()
{
    std::cout << "Enter triangle:\n";
    Triangle trngl(std::cin);
    trngl.Print(std::cout);
    std::cout << "The number of vertexes: " << trngl.VertexesNumber() << "\n";
    std::cout << "Area: " << trngl.Area() << "\n\n";

    std::cout << "Enter square:\n";
    Square sqr(std::cin);
    sqr.Print(std::cout);
    std::cout << "The number of vertexes: " << sqr.VertexesNumber() << "\n";
    std::cout << "Area: " << sqr.Area() << "\n\n";

    std::cout << "Enter rectangle:\n";
    Rectangle rect(std::cin);
    rect.Print(std::cout);
    std::cout << "The number of vertexes: " << rect.VertexesNumber() << "\n";
    std::cout << "Area: " << rect.Area() << "\n";
    return 0;
}
```

triangle.h

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include "figure.h"

class Triangle : public Figure {
 private:
  Point a_, b_, c_;
 public:
  Triangle();
  Triangle(const Triangle &triangle);
  Triangle(std::istream &is);
  size_t VertexesNumber();
  double Area();
  void Print(std::ostream &os);

};

#endif //TRIANGLE_H
```

triangle.cpp

```cpp
#include "triangle.h"
#include <math.h>

Triangle::Triangle() : a_(0, 0), b_(0, 0), c_(0, 0) {}

Triangle::Triangle(const Triangle &triangle) {
  this->a_ = triangle.a_;
  this->b_ = triangle.b_;
  this->c_ = triangle.c_;
}

Triangle::Triangle(std::istream &is) {
  std::cin >> a_ >> b_ >> c_;
}

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

double Triangle::Area() {
  double a = a_.dist(b_);
  double b = b_.dist(c_);
  double c = c_.dist(a_);
  double p = (a + b + c) / 2;
  return sqrt(p * (p - a) * (p - b) * (p - c));
}

void Triangle::Print(std::ostream &os) {
  std::cout << "Triangle " << a_ << b_ << c_ << std::endl;
}
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"

class Rectangle : public Figure {
 private:
  Point a_, b_, c_, d_;

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

};

#endif //RECTANGLE_H
```

rectangle.cpp

```cpp
#include "rectangle.h"

Rectangle::Rectangle() : a_(0, 0), b_(0, 0), c_(0, 0), d_(0, 0) {}

Rectangle::Rectangle(const Rectangle &rectangle) {
  this->a_ = rectangle.a_;
  this->b_ = rectangle.b_;
  this->c_ = rectangle.c_;
  this->d_ = rectangle.d_;
}

Rectangle::Rectangle(std::istream &is) {
  std::cin >> a_ >> b_ >> c_ >> d_;
}

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

double Rectangle::Area() {
  double a = a_.dist(b_);
  double b = b_.dist(c_);
  return a * b;
}

void Rectangle::Print(std::ostream &os) {
  std::cout << "Rectangle " << a_ << b_ << c_ << d_ << std::endl;
}
```

square.h

```
#ifndef SQUARE_H
#define SQUARE_H

#include "figure.h"

class Square : public Figure {
 private:
  Point a_, b_, c_, d_;
 public:
  Square();
  Square(const Square &square);
  Square(std::istream &is);
  size_t VertexesNumber();
  double Area();
  void Print(std::ostream &os);
};

#endif //SQUARE_H
```

square.cpp

```cpp
#include "square.h"

Square::Square() : a_(0, 0), b_(0, 0), c_(0, 0), d_(0, 0) {}

Square::Square(const Square &square) {
  this->a_ = square.a_;
  this->b_ = square.b_;
  this->c_ = square.c_;
  this->d_ = square.d_;
}

Square::Square(std::istream &is) {
  std::cin >> a_ >> b_ >> c_ >> d_;
}

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

double Square::Area() {
  double a = a_.dist(b_);
  return a * a;
}

void Square::Print(std::ostream &os) {
  std::cout << "Square " << a_ << b_ << c_ << d_ << std::endl;
}
```