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 Абдуль-Хади Филипп, группа М8О-207Б-21
Преподаватель Дорохов Евгений Павлович

Условие

Задание: Вариант 1: Квадрат, Прямоугольник, Трапеция. 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 стандарт-ного 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 ввода координат вершин и получения информации о фигуре
2. include/figure.h: описание абстрактного класса фигур
3. include/point.h: описание класса точки
4. include/square.h: описание класса квадрата, наследующегося от figure
5. include/rectangle.h: описание класса прямоугольника, наследующегося от figure
6. include/trapezoid.h: описание класса трапеции, наследующегося от figure
7. include/figure.cpp: реализация класса figure
8. include/point.cpp: реализация класса точки
9. include/square.cpp: реализация класса квадрата, наследующегося от figure
10. include/rectangle.cpp: реализация класса прямоугольника, наследующегося от figure
11. include/trapezoid.cpp: реализация класса трапеции, наследующегося от figure

Дневник отладки

Проблема	**Исправление**
Неудобно собирать без make файла	Создал make файл

Недочёты

Отсутствуют

Выводы

Наследование и прочие конструкции языка C++ очень полезны при разработке приложений. Благодаря наследованию можно упростить написание кода, и не писать одни и те же поля по несколько раз в нескольких классах (вместо этого мы вынесем общие поля и функционал в отдельный класс).

Исходный код ниже:

main.cpp

```cpp
#include <iostream>
#include "point.h"
#include "figure.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"

using namespace std;

int main(int argc,char *argv[])
{
  cout<<"Variant 2:"<<endl;
  cout<<"Enter rectangle coords (for example: 0 0 5 0 5 5 0 5)"<<endl;
  Figure* fig = new Rectangle(cin);
  fig->Print(cout);
  cout<<"Vertices: "<<(fig->VertexesNumber())<<endl;
  cout<<"Area: "<<(fig->Area())<<endl;
  delete fig;
  cout<<"Enter square coords (for example: 0 0 5 0 5 5 0 5)"<<endl;
  fig = new Square(cin);
  fig->Print(cout);
  cout<<"Vertices: "<<(fig->VertexesNumber())<<endl;
  cout<<"Area: "<<(fig->Area())<<endl;
  delete fig;
  cout<<"Enter trapezoid coords (for example: 1 1 2 5 5 5 6 1)"<<endl;
  fig = new Trapezoid(cin);
  fig->Print(cout);
  cout<<"Vertices: "<<(fig->VertexesNumber())<<endl;
  cout<<"Area: "<<(fig->Area())<<endl;
  delete fig;
  return 0;
}
```

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;
                double calcTriangleArea(Point p1, Point p2, Point p3);
                virtual ~Figure() {}

                friend std::ostream& operator<<(std::ostream& os, Figure& p)
                {
                        p.Print(os);
                        return os;
                }
};

#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 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);
};

#endif // POINT_H
```

square.h

```
#ifndef SQUARE_H
#define SQUARE_H

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

class Square: public Figure
{
        public:
                Square(std::istream &is);
                Square(Point pnt1, Point pnt2, Point pnt3, Point pnt4);

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

        friend std::istream& operator>>(std::istream& is, Square& r);
        friend std::ostream& operator<<(std::ostream& os, Square& r);

        private:
                Point p1;
                Point p2;
                Point p3;
                Point p4;
};

#endif
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

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

class Rectangle: public Figure
{
        public:
                Rectangle(std::istream &is);
                Rectangle(Point pnt1, Point pnt2, Point pnt3, Point pnt4);

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

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

        private:
                Point p1;
                Point p2;
                Point p3;
                Point p4;
};

#endif
```

trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

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

class Trapezoid: public Figure
{
        public:
                Trapezoid(std::istream &is);
                Trapezoid(Point p1, Point p2, Point p3, Point p4);

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

        friend std::istream& operator>>(std::istream& is, Trapezoid& r);
        friend std::ostream& operator<<(std::ostream& os, Trapezoid& r);

        private:
                Point p1;
                Point p2;
                Point p3;
                Point p4;
};

#endif
```

figure.cpp

```cpp
#include "figure.h"
#include <cmath>

double Figure::calcTriangleArea(Point p1, Point p2, Point p3)
{
        return abs((p1.x_-p3.x_)*(p2.y_-p3.y_)-(p2.x_-p3.x_)*(p1.y_-
p3.y_))/2.0;
}
```

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;
}
```

square.cpp

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

Square::Square(std::istream &is)
{
        is >> *this;
}

Square::Square(Point pnt1, Point pnt2, Point pnt3, Point pnt4)
{
        p1 = pnt1;
        p2 = pnt2;
        p3 = pnt3;
        p4 = pnt4;
}

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

double Square::Area()
{
        return calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);
}

void Square::Print(std::ostream& os)
{
        os << *this;
}

std::istream& operator>>(std::istream& is, Square& r) {
  is >> r.p1 >> r.p2 >> r.p3 >> r.p4;
  return is;
}

std::ostream& operator<<(std::ostream& os, Square& r) {
  os << "Square: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4 << "\
n";
  return os;
}
```

rectangle.cpp

```cpp
#include "rectangle.h"
#include <cmath>

Rectangle::Rectangle(std::istream &is)
{
        is >> *this;
}

Rectangle::Rectangle(Point pnt1, Point pnt2, Point pnt3, Point pnt4)
{
        p1 = pnt1;
        p2 = pnt2;
        p3 = pnt3;
        p4 = pnt4;
}

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

double Rectangle::Area()
{
        return calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);
}

void Rectangle::Print(std::ostream& os)
{
        os << *this;
}

std::istream& operator>>(std::istream& is, Rectangle& r) {
  is >> r.p1 >> r.p2 >> r.p3 >> r.p4;
  return is;
}

std::ostream& operator<<(std::ostream& os, Rectangle& r) {
  os << "Rectangle: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4 <<
"\n";
  return os;
}
```

trapezoid.cpp

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

Trapezoid::Trapezoid(std::istream &is)
{
        is >> *this;
}

Trapezoid::Trapezoid(Point p1, Point p2, Point p3, Point p4)
{
        this->p1 = p1;
        this->p2 = p2;
        this->p3 = p3;
        this->p4 = p4;
}

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

double Trapezoid::Area()
{
        return calcTriangleArea(p1,p2,p3)+calcTriangleArea(p3,p4,p1);
}

void Trapezoid::Print(std::ostream& os)
{
        os << *this;
}

std::istream& operator>>(std::istream& is, Trapezoid& r) {
  is >> r.p1 >> r.p2 >> r.p3 >> r.p4;
  return is;
}

std::ostream& operator<<(std::ostream& os, Trapezoid& r) {
  os << "Trapezoid: " << r.p1 << " " << r.p2 << " " << r.p3 << " " << r.p4 <<
"\n";
  return os;
}
```