**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 *Зинин Владислав Владимирович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Условие**

Задание: Вариант 5: Rhombus, Hexagon, Pentagon. 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 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню

2. include/figure.h: описание абстрактного класса фигур

3. include/point.h: описание класса точки

4. include/pentagon.h: описание класса пятиугольника, наследующегося от figures 5. include/hexagon.h: описание класса шестиугольника, наследующегося от figures 6. include/rhombus.h: описание класса ромба, наследующегося от figures 7. include/point.cpp: реализация класса точки

8. include/pentagon.cpp: реализация класса пятиугольника, наследующегося от

figures

9. include/hexagon.cpp: реализация класса шестиугольника, наследующегося от figures

10. include/rhombus.cpp: реализация класса ромба, наследующегося от figure

**Дневник отладки**

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки. После их исправления программа работала так, как было задумано изначально.

**Недочеты**

Во время выполнения лабораторной работы недочетов в программе обнаружено не было.

**Выводы:**

Основная цель лабораторной работы №3 - знакомство с парадигмой объектно-ориентированного программирования на языке С++. Могу сказать, что справился с этой целью весьма успешно: усвоил “3 китов ООП”: полиморфизм, наследование, инкапсуляция, освоил базовые понятия ООП, такие как классы, методы, конструкторы, деструкторы… Ознакомился с ключевыми словами virtual, friend, private, public... Повторил тему “директивы условной компиляции”, “перегрузка функций/операторов”, работа со стандартными потоками ввода-вывода. **Лабораторная работа №3 прошла для меня успешно.**

**Исходный код**

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include "point.h"

class Figure

{

public:

    virtual ~Figure(){};

    virtual double Area() = 0;

    virtual void Print(std::*ostream* &*os*) = 0;

    virtual size\_t VertexesNumber() = 0;

};

#endif //FIGURE\_H

point.h

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point

{

public:

    Point();

    Point(double *x*, double *y*);

    Point(std::*istream* &*is*);

    double dist(*Point* &*other*);

    friend double get\_x(*Point* &*other*);

    friend double get\_y(*Point* &*other*);

    friend std::*istream*& operator>>(std::*istream*& *is*, *Point*& *p*);

    friend std::*ostream*& operator<<(std::*ostream*& *os*, *Point*& *p*);

private:

    double x\_, y\_;

};

#endif //POINT\_H

point.cpp

#include <iostream>

#include "point.h"

*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\_;

}

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*;

}

double get\_x(*Point* &*other*)

{

    return *other*.x\_;

}

double get\_y(*Point* &*other*)

{

    return *other*.y\_;

}

pentagon.h

#ifndef PENTAGON\_H

#define PENTAGON\_H

#include <iostream>

#include "figure.h"

class Pentagon : public *Figure*

{

public:

    Pentagon(std::*istream* &*is*);

    double Area();

    void Print(std::*ostream* &*os*);

    size\_t VertexesNumber();

    virtual ~Pentagon();

private:

    Point a, b, c, d, e;

};

#endif // PENTAGON\_H

pentagon.cpp

#include "pentagon.h"

#include <cmath>

#include <iostream>

Pentagon::Pentagon(std::istream &is)

{

    is >> a;

    is >> b;

    is >> c;

    is >> d;

    is >> e;

}

void Pentagon::Print(std::*ostream* &*os*)

{

    os << "Pentagon" << std::endl;

    os << a << "," << b << "," << c << "," << d << "," << e << std::endl;

}

double Pentagon::Area()

{

    return 0.5 \* fabs(get\_x(a)\*get\_y(b) + get\_x(b)\*get\_y(c) + get\_x(c)\*get\_y(d) + get\_x(d)\*get\_y(e) + get\_x(e)\*get\_y(a) - get\_x(b)\*get\_y(a) - get\_x(c)\*get\_y(b) - get\_x(d)\*get\_y(c) - get\_x(e)\*get\_y(d) - get\_x(a)\*get\_y(e));

}

size\_t Pentagon::VertexesNumber()

{

    return 5;

}

Pentagon::~Pentagon()

{

    std::cout << "Pentagon deleted" << std::endl;

}

rhombus.h

#ifndef RHOMBUX\_H

#define RHOMBUX\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Rhombus : public *Figure*

{

public:

    Rhombus(std::*istream* &*is*);

    double Area();

    void Print(std::*ostream* &*os*);

    size\_t VertexesNumber();

    virtual ~Rhombus();

private:

    Point a, b, c, d;

};

#endif //RHOMBUX\_H

rhombus.cpp

#include <iostream>

#include "rhombus.h"

#include <math.h>

Rhombus::Rhombus(std::istream &is)

{

    is >> a;

    is >> b;

    is >> c;

    is >> d;

}

void Rhombus::Print(std::*ostream* &*os*)

{

    os << "Rhombus" << std::endl;

    os << a << ',' << b << ',' << c << ',' << d << std::endl;

}

double Rhombus::Area()

{

    return 0.5 \* fabs(get\_x(a)\*get\_y(b) + get\_x(b)\*get\_y(c) + get\_x(c)\*get\_y(d) + get\_x(d)\*get\_y(a) - get\_x(b)\*get\_y(a) - get\_x(c)\*get\_y(b) - get\_x(d)\*get\_y(c) - get\_x(a)\*get\_y(d));

}

Rhombus::~Rhombus()

{

    std::cout << "Rhombus deleted" << std::endl;

}

size\_t Rhombus::VertexesNumber()

{

    return 4;

}

hexagon.h

#ifndef HEXAGON\_H

#define HEXAGON\_H

#include <iostream>

#include "figure.h"

class Hexagon : public *Figure*

{

public:

    Hexagon(std::*istream* &*is*);

    double Area();

    void Print(std::*ostream* &*os*);

    size\_t VertexesNumber();

    virtual ~Hexagon();

private:

    Point a, b, c, d, e, f;

};

#endif // HEXAGON\_H

hexagon.cpp

#include "hexagon.h"

#include <cmath>

#include <iostream>

Hexagon::Hexagon(std::istream &is)

{

    is >> a;

    is >> b;

    is >> c;

    is >> d;

    is >> e;

    is >> f;

}

void Hexagon::Print(std::*ostream* &*os*)

{

    os << "Hexagon" << std::endl;

    os << a << "," << b << "," << c << "," << d << "," << e << "," << f << std::endl;

}

size\_t Hexagon::VertexesNumber()

{

    return 6;

}

double Hexagon::Area()

{

    return 0.5 \* fabs(get\_x(a)\*get\_y(b) + get\_x(b)\*get\_y(c) + get\_x(c)\*get\_y(d) + get\_x(d)\*get\_y(e) + get\_x(e)\*get\_y(f) + get\_x(f)\*get\_y(a) - get\_x(b)\*get\_y(a) - get\_x(c)\*get\_y(b) - get\_x(d)\*get\_y(c) - get\_x(e)\*get\_y(d) - get\_x(f)\*get\_y(e) - get\_x(a)\*get\_y(f));

}

Hexagon::~Hexagon()

{

    std::cout << "Hexagon deleted" << std::endl;

}

main.cpp

#include <iostream>

#include "rhombus.h"

#include "pentagon.h"

#include "hexagon.h"

int main()

{

    Rhombus a(std::cin);

    std::cout << "Square = " << a.Area() << std::endl;

    a.Print(std::cout);

    Pentagon b(std::cin);

    std::cout << "Square = " << b.Area() << std::endl;

    b.Print(std::cout);

    Hexagon c(std::cin);

    std::cout << "Square = " << c.Area() << std::endl;

    c.Print(std::cout);

    return 0;

}