Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа
по курсу «ООП»

Тема:
Наследование, полиморфизм.

Студент:	Черненко И.Д
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	24
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

point.hpp:

```
#ifndef OOP_EXERCISE_03_POINT_H
#define OOP_EXERCISE_03_POINT_H
#include <iostream>
struct point {
   double x,y;
};
point operator+ (point lhs, point rhs);
point operator- (point lhs, point rhs);
point operator* (point p, double t);
point operator/ (point p, double t);
std::istream &operator>> (std::istream &is, point &p );
std::ostream &operator<< (std::ostream &os, const point &p );
#endif //OOP_EXERCISE_03_POINT_H
```

point.cpp:

```
#include "point.h"
point operator+ (point lhs, point rhs) {
   return {lhs.x + rhs.x, lhs.y + rhs.y};
}
point operator- (point lhs, point rhs) {
   return {lhs.x - rhs.x, lhs.y - rhs.y};
}
point operator/ (point p, double t) {
   return {p.x / t, p.y / t};
}
point operator* (point p, double t) {
   return {p.x * t, p.y * t};
}
std::istream &operator>>(std::istream &is, point &p) {
   is >> p.x >> p.y;
   return is;
}
std::ostream &operator<< (std::ostream &os, const point &p ) {
   os << p.x << " " << p.y << std::endl;
   return os;
}
```

figure.h:

```
#include <iostream>
#include "point.h"
#include <cmath>
#ifndef OOP_EXERCISE_03_FIGURE_H
#define OOP_EXERCISE_03_FIGURE_H
struct figure {
   virtual point center() const = 0;
```

```
    virtual std::ostream& print(std::ostream& os) const = 0;
    virtual double square() const = 0;
    virtual ~figure() {}
};
#endif //OOP_EXERCISE_03_FIGURE_H
```
triangle.h:
```
#ifndef OOP_EXERCISE_03_TRIANGLE_H
#define OOP_EXERCISE_03_TRIANGLE_H
#include "figure.h"
class triangle : public figure {
public:
    triangle();
    triangle(const point& a, const point& b, const point& c);
    triangle(std::istream& is) ;
    double square() const override;
    point center() const override;
    std::ostream& print(std::ostream& os) const override;
private:
    point _a;
    point _b;
    point _c;
};
#endif //OOP_EXERCISE_03_TRIANGLE_H
```
triangle.cpp:
```
#include "triangle.h"
triangle::triangle(): _a{0, 0}, _b{0, 0}{}
triangle::triangle(const point& a, const point& b, const point& c) : _a{a}, _b{b}, _c{c} {
    double l, k, p;
    l = sqrt((_a.x - _b.x) * (_a.x - _b.x) + (_a.y - _b.y) * (_a.y - _b.y));
    k = sqrt((_b.x - _c.x) * (_b.x - _c.x) + (_b.y - _c.y) * (_b.y - _c.y));
    p = sqrt((_c.x - _b.x) * (_c.x - _b.x) + (_c.y - _b.y) * (_c.y - _b.y));
    if (l + k <= p || l + p <= k || p + k <= l) {
        throw std::logic_error("Triangle doesn't exist");
    }
}
triangle::triangle(std::istream& is) {
    is >> _a >> _b >> _c;
    double l, k, p;
    l = sqrt((_a.x - _b.x) * (_a.x - _b.x) + (_a.y - _b.y) * (_a.y - _b.y));
    k = sqrt((_b.x - _c.x) * (_b.x - _c.x) + (_b.y - _c.y) * (_b.y - _c.y));
    p = sqrt((_c.x - _b.x) * (_c.x - _b.x) + (_c.y - _b.y) * (_c.y - _b.y));
    if (l + k <= p || l + p <= k || p + k <= l) {
        throw std::logic_error("Triangle doesn't exist");
    }
```

```
}
double triangle::square() const {
    return fabs(((_a.x - _c.x) * (_b.y - _c.y) - (_b.x - _c.x) * (_a.y - _c.y)) / 2);
}
point triangle::center() const {
    return point{(_a.x + _b.x + _c.x) / 3, (_a.y + _b.y + _c.y) / 3};
}
std::ostream& triangle::print(std::ostream& os) const {
    os << _a  <<  _b  << _c << std::endl;
    return os;
}
```

quadrate.h:

```
#ifndef OOP_EXERCISE_03_QUADRATE_H
#define OOP_EXERCISE_03_QUADRATE_H
#include "figure.h"
class quadrate : public figure {
public:
    quadrate();
    quadrate(const point& a, const point& c);
    quadrate(std::istream& is);
    double square() const override;
    point center() const override;
    std::ostream& print(std::ostream& os) const override;
private:
    point _a;
    point _c;
};
#endif //OOP_EXERCISE_03_QUADRATE_H
```

quadrate.cpp:

```
#include "quadrate.h"
quadrate::quadrate(): _a{0, 0}, _c{0, 0}{}
quadrate::quadrate(const point& a, const point& c) : _a{a}, _c{c}{}
quadrate::quadrate(std::istream &is) {
    is >> _a >> _c;
};
double quadrate::square() const {
    return (_c.x - _a.x) * (_c.x - _a.x);
}
point quadrate::center() const {
    return point{(_a.x + _c.x) / 2, (_a.y + _c.y) / 2};
}
std::ostream& quadrate::print(std::ostream& os) const {
    point m = center();
    point b, d;
    b.x = m.x - _c.y + m.y;
```

```
    b.y = m.y + _c.x - m.x;
    d.x = m.x - _a.y + m.y;
    d.y = m.y + _a.x - m.x;
    os << _a  << b << _c << d << std::endl;
    return os;
}
```

octagon.h:

```
#ifndef OOP_EXERCISE_03_OCTAGON_H
#define OOP_EXERCISE_03_OCTAGON_H
#include "figure.h"
class octagon : public figure {
public:
    octagon();
    octagon(const point& a, const point& b, const point& c, const point& d, const point& e, const point& f, const point& g, const point& h);
    octagon(std::istream& is);
    double square() const override;
    point center() const override;
    std::ostream& print(std::ostream&) const override;
private:
    point _a;
    point _b;
    point _c;
    point _d;
    point _e;
    point _f;
    point _g;
    point _h;
};
#endif //OOP_EXERCISE_03_OCTAGON_H
```

octagon.cpp:

```
#include "octagon.h"
octagon::octagon(): _a{0, 0}, _b{0, 0}{}
octagon::octagon(std::istream& is) {
    is >> _a >> _b >> _c >> _d >> _e >> _f >> _g >> _h;
}
octagon::octagon(const point& a, const point& b, const point& c, const point& d, const point& e, const point& f, const point& g, const point& h) : _a{a}, _b{b}, _c{c}, _d{d}, _e{e}, _f{f}, _g{g}, _h{h}{}
double octagon::square() const {
    return fabs(((_a.x * _b.y) + (_b.x * _c.y) + (_c.x * _d.y) + (_d.x * _e.y) + (_e.x * _f.y) + (_f.x * _g.y) + (_g.x * _h.y) + (_h.x * _a.y) - (_b.x * _a.y) - (_c.x * _b.y) - (_d.x * _c.y) - (_e.x * _d.y) - (_f.x * _e.y) - (_g.x * _f.y) - (_h.x * _g.y) - (_a.x * _h.y)) / 2);
```

```
}
point octagon::center() const {
    return point{(_a.x + _b.x + _c.x + _d.x + _e.x + _f.x + _g.x + _h.x) / 8, (_a.y + _b.y + _c.y + _d.y + _e.y + _f.y + _g.y + _h.y) / 8};
}
std::ostream& octagon::print(std::ostream& os) const {
    os << _a <<  _b << _c <<  _d << _e <<  _f << _g <<  _h << std::endl;
    return os;
}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.14)
project(oop_exercise_03)
set(CMAKE_CXX_STANDARD 17)
add_executable(oop_exercise_03 main.cpp figure.h point.h point.cpp figure.cpp triangle.cpp triangle.h quadrate.cpp quadrate.h octagon.cpp octagon.h)
```

2. Ссылка на репозиторий на GitHub

https://github.com/IllCher/oop_exercise_03

3. Набор testcases.

test_00:

```
1
0
2 2
2 5
4 7
7 7
9 5
9 2
7 -1
4 -1
1
1
1 -3
0 2
-3 -2
1
2
1 -3
0 2
-3 -2
2
0
2
1
2
```

2
3
1
2
0
0
test_01:
1
1
0 0
0 0
1 1
test_02:
1
3
test_03:
3
5
test_04:
1
0
0 0
0 2
2 3
4 3
6 2
6 0
4 -1
2 -1
1
1
0 0
0 2
2 0
1
2
0 0
2 2
2
0
2
1
2
2
0

4. Результаты выполнения тестов.

test_00:
2 2
2 5
4 7
7 7
9 5
9 2
7 -1
4 -1

1 -3
0 2
-3 -2

2 2
2.5 4.5
5 4
4.5 1.5

46

9.5

9

5.5 3.25

-0.666667 -1

3.5 3

2 2
2 5
4 7
7 7
9 5
9 2
7 -1
4 -1

2 2
2.5 4.5

5 4
4.5 1.5
test_01:
terminate called after throwing an instance of 'std::logic_error'
  what(): Triangle doesn't exist
test_02:
no such a figure

test_03:
no such an element

Process finished with exit code -1
test_04:
0 0
0 2
2 3
4 3
6 2
6 0
4 -1
2 -1

0 0
0 2
2 0

0 0
0 2
2 2
2 0

20

2

4

3 1

0.666667 0.666667

1 1

5. Объяснение результатов работы программы.

1) При запуске программы с аргументом test_??.txt программа получает на вход последовательность команд и их аргументов, содержащихся в файлах test_??.txt.
2) Далее в программе создается вектор из указателей на объекты типа figure.
3) В дальнейшем вводятся команды:
   0 – выход из цикла и прекращение ввода команд.
   1 – создание фигуры и ввод её координат по figure_id (при вводе 0 – создается восьмиугольник, 1 – создается треугольник, 2 – создается квадрат).
   2 – выполнение над всеми фигурами в векторе определенных действий и вывод результатов, где набор действий определяется переменной function_id, значение которой вводится пользователем (0 – вывод координат фигур, 1 – вывод площадей фигур, 2 – вывод координат центров фигур).
   3 – удаление фигуры из вектора по её индексу: id, вводимому пользователем.
4) Далее продолжается ввод таких команд в бесконечном цикле.
5) После введения команды для завершения работы программы, из созданного вектора удаляются указатели на использованные фигуры.

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с механизмами наследования классов в C++ и полиморфизмом (общие методы для различных фигур: center(), square(), print(), по-разному определенные в самих классах фигур, что позволяет работать с ними в едином интерфейсе), кроме того было изучено такое понятие, как полностью виртуальная функция (метод), т.е. метод, который в классе родителе не определен, но определяется в классах наследниках.